TYPICAL QUESTIONS & ANSWERS

PART - I

OBJECTIVE TYPE QUESTIONS

Each Question carries 2 marks.

Choose correct or the best alternative in the following:

Q.1 What is the output of the following program?

```
main ( )
{     int x = 2, y = 5;
       if (x < y) return (x = x+y); else printf ("z1");
       printf("z2");
}
```

(A) z2 **(B)** z1z2
(C) Compilation error **(D)** None of these

Ans: D
There is no compilation error but there will no output because function is returning a value and if statement is true in this case.

Q.2 Choose the correct one

(A) Address operator can not be applied to register variables
(B) Address operator can be applied to register variables
(C) Use of register declaration will increase the execution time
(D) None of the above

Ans: D
A register access is much faster than a memory access, keeping the frequently accessed variables in the register will lead to faster execution of programs.

Q.3 What is the following program doing?

```
main ()
{ int d = 1;
do
            printf("%d\n", d++);
while (d < = 9);}
```

(A) Adding 9 integers **(B)** Adding integers from 1 to 9
(C) Displaying integers from 1 to 9 **(D)** None of these

Ans: C
d starting from 1 is incrementing one by one till d=9 so the printf statement is printing numbers from 1 to 9.

Q.4 What is the output of the following program?

```
main ( )
{    extern int x;
      x = 20;
       printf("\n%d", x);
}
```

(A) 0 **(B)** 20
(C) error **(D)** garbage value

Ans: C
Output of the given program will be "Linker error-undefined symbol x". External variables are declared outside a function.

Q.5 If x is one dimensional array, then pick up the correct answer
(A) *(x + i) is same as &x[i] **(B)** *&x[i] is same as x + i
(C) *(x + i) is same as x[i] +1 **(D)** *(x + i) is same as *x[i]

Ans: A
num[i] is same as *(num+i)

Q.6 Consider the following declaration

```
int a, *b = &a, **c = &b;
```

The following program fragment

```
a = 4;
**c = 5;
```

(A) does not change the value of a **(B)** assigns address of c to a
(C) assigns the value of b to a **(D)** assigns 5 to a

Ans: D
The given statements assigns 5 to a

Q.7 Choose the correct answer
(A) enum variable can not be assigned new values
(B) enum variable can be compared
(C) enumeration feature increase the power of C
(D) None of the above

Ans: C
The enumerated data types give an opportunity to invent our own data typeand define what value the variable of this data type can take.

Q.8 The content of file will be lost if it is opened in
(A) w mode **(B)** w+ mode
(C) a mode **(D)** a+ mode

Ans: A
When the mode is writing, the contents are deleted and the file is opened as a new file.

Q.9 Consider the following code segment:

```
int a[10], *p1, *p2;
p1 = &a[4];
p2 = &a[6];
```

Which of the following statements is incorrect w.r.t. pointers?
(A) p1 + 2 **(B)** p2 – 2
(C) p2 + p1 **(D)** p2 – p1

Ans: C

Addition of two pointers is not allowed.

Q.10 The second expression (j – k) in the following expression will be evaluated

```
(i + 5) && (j - k)
```

(A) if expression (i + 5) is true.
(B) if expression (i + 5) is false.
(C) irrespective of whether (i + 5) is true or false.
(D) will not be evaluated in any case.

Ans: A
In a compound logical expression combined with &&, the second expression is evaluated only if first is evaluated in true.

Q.11 In the **for** statement: `for (exp1; exp2; exp3) { … }`
where exp1, exp2 and exp3 are expressions. What is optional?
(A) None of the expressions is optional.
(B) Only exp1 is optional.
(C) Only exp1 and exp3 are optional.
(D) All the expressions are optional.

Ans: D
All the expressions are optional. For (;;) is a valid statement in C.

Q.12 The output of the following code segment will be

```
char x = 'B';
switch (x) {
case 'A': printf("a");
case 'B': printf("b");
case 'C': printf("c");
}
```

(A) B **(B)** b
(C) BC **(D)** bc

Ans: D
Since there is no break statement, all the statement after case'B' are executed.

Q.13 What will be the output of the following code segment?

```
main( ) {
char  s[10];
strcpy(s, "abc");
printf("%d %d", strlen(s), sizeof(s));
}
```

(A) 3 10 **(B)** 3 3
(C) 10 3 **(D)** 10 10

Ans: A
strlen(s) give the length of the string, that is 3 and sizeof(s) give the size of array s that is 10.

Q.14 Which of the following is the odd one out?
(A) j = j + 1; **(B)** j =+ 1;
(C) j++; **(D)** j += 1;

Ans: B
j=+1 is odd one out as rest all means incrementing the value of variable by 1.

Q.15 Which of the following is true for the statement:

```
NurseryLand.Nursery.Students = 10;
```

(A) The structure Students is nested within the structure Nursery.
(B) The structure NurseryLand is nested within the structure Nursery.
(C) The structure Nursery is nested within the structure NurseryLand.
(D) The structure Nursery is nested within the structure Students.

Ans: C
The structure Nursery is nested within the structure NurseryLand.

Q.16 What will be the output of the following code segment, if any?

```
    myfunc ( struct test t) {
    strcpy(t.s, "world");
    }
 main( ) {
struct test { char s[10]; } t;
    strcpy(t.s, "Hello");
    printf("%s", t.s);
    myfunc(t);
    printf("%s", t.s);
}
```

(A) Hello Hello
(B) world world
(C) Hello world
(D) the program will not compile

Ans: D
The program will not compile because undefined symbol s for myfunc( ) function. Structure should be defined before the main and the function where it is called.

Q.17 If a function is declared as **void fn(int *p)**, then which of the following statements is valid to call function **fn**?

(A) **fn(x)** where x is defined as **int x;**
(B) **fn(x)** where x is defined as **int *x;**
(C) **fn(&x)** where x is defined as **int *x;**
(D) **fn(*x)** where x is defined as **int *x;**

Ans: B
Function **void fn(int *p)** needs pointer to int as argument. When x is defined as int *x, then x is pointer to integer and not *x.

Q.18 What is the following function computing? Assume a and b are positive integers.

```
 int fn( int a, int b) {
   if (b == 0)
         return b;
   else
         return (a * fn(a, b - 1));
}
```

(A) Output will be 0 always
(B) Output will always be b
(C) Computing a^b
(D) Computing a + b

Ans: A
The output is always be 0 because b is decremented in recursive function fn each time by 1 till the terminating condition b==0 where it will return 0.

Q.19 What is the output of the following C program?

```
# include <stdio.h>
main ( )
{
 int a, b=0;
 static int c [10]={1,2,3,4,5,6,7,8,9,0};
 for (a=0; a<10;+ + a)
 if ((c[a]%2)= = 0) b+ = c [a];
 printf ("%d", b);
  }
```

(A) 20 **(B)** 25
(C) 45 **(D)** 90

Ans: A
printf statement will print b which is sum of the those values from array c which get divided by 2, that is 2+4+6+8=20.

Q.20 If a, b and c are integer variables with the values a=8, b=3 and c=-5. Then what is the value of the arithmetic expression:

```
2 *  b + 3 * (a-c)
```

(A) 45 **(B)** 6
(C) -16 **(D)** -1

Ans: A
the value of the arithmetic expression is 45 as 2*3+3*(8—5)=6+3*13=6+39=45

Q.21 A global variable is a variable
(A) declared in the main ( ) function.
(B) declared in any function other than the main ( ) function.
(C) declared outside the body of every function.
(D) declared any where in the C program.

Ans: C
A global variable is declared outside the body of every function.

Q.22 main ( ) is an example of
(A) library function **(B)** user defined function
(C) header **(D)** statement

Ans: A
main() is a special function used by C system to tell the computer where the program starts.

Q.23 While incrementing a pointer, its value gets increased by the length of the data type to which it points. This length is called
(A) scale factor **(B)** length factor
(C) pointer factor **(D)** increment factor
Ans: D

While incrementing a pointer, its value gets increased by the length of the data type to which it points.

Q.24 The first digit of a decimal constant must be

(A) a zero **(B)** a non zero number
(C) a negative number **(D)** an integer

Ans: D
Decimal constants consist of a set of digit, 0 to 9, preceded by an optional – or + sign.

Q.25 What is the output of the following statement:

```
printf ("%-3d",12345);
```

(A) 1 2 3 **(B)** -1 2 3
(C) 1 2 3 4 5 **(D)** 12

Ans: C
printf statement would print 12345.

Q.26 A single character input from the keyboard can be obtained by using the function.

(A) printf ( ) **(B)** scanf ( )
(C) putchar ( ) **(D)** getchar ( )

Ans: D
Reading a single character can be done by using the function getchar( ).

Q.27 The function ftell ( )

(A) reads a character from a file
(B) reads an integer from a file
(C) gives the current position in the file
(D) sets the position to the beginning of the file.

Ans: C
ftell( ) takes a file pointer and returns a number of type long, that corresponds to the current position.

Q.28 If the variables i, j and k are assigned the values 5,3 and 2 respectively, then the expression `i = j + ( k + + = 6 ) + 7`

(A) gives an error message **(B)** assigns a value 16 to i
(C) assigns a value 18 to i **(D)** assigns a value 19 to i

Ans: A
It gives an error message-Lvalue required.

Q.29 If an integer needs two bytes of storage, then the maximum value of a signed integer is

(A) 2^{16}-1 **(B)** 2^{15}-1
(C) 2^{16} **(D)** 2^{15}

Ans: B
If we use a 16 bit word length, the size of the integer value is limited to the range -2^{15} to 2^{15}-1

Q.30 Literal means

(A) a string **(B)** a string constant
(C) a character **(D)** an alphabet

Ans: B
Literal means a string constant.

Q.31 If 'y' is of integer type then the expressions
$3*(y-8)/9$ and $(y-8)/9*3$
(A) must yield the same value.
(B) must yield different values.
(C) may or may not yield the same value.
(D) none of the above.

Ans: C
The expression may or may not yield the same value.

Q.32 In the following code fragment

```
int x, y = 2, z, a;
x=(y*=2) + (z=a=y);
printf ('%d',x);
```

(A) prints 8
(B) prints 6
(C) prints 6 or 8 depending on the compiler
(D) is syntactically wrong

Ans: A
It will print 8 because x=(y*=2)+(z=a=y)=4+4=8.

Q.33 A possible output of the following program fragment is

```
for (i=getchar();; i=get.char())
          if (i=='x') break;
          else putchar(i);
```

(A) mi **(B)** mix
(C) mixx **(D)** none of the above

Ans: D
None of the above as it is wrong syntax.

Q.34 In a for loop, if the condition is missing, then,
(A) It is assumed to be present and taken to be false.
(B) It is assumed to be present and taken to be true.
(C) It results in a syntax error.
(D) Execution will be terminated abruptly.

Ans: B

Q.35 If storage class is missing in the array definition, by default it will be taken to be
(A) automatic
(B) external
(C) static

(D) either automatic or external depending on the place of occurrence.

Ans: A
A variable declared inside inside a function without storage class specification is, by default, an automatic variable.

Q.36 The maximum number of dimensions an array can have in C is
(A) 3 **(B)** 4
(C) 5 **(D)** compiler dependent

Ans: D
C allows arrays of three or more dimensions. The exact limit is determined by the compiler.

Q.37 puts(argv[0]);
(A) prints the name of the source code file.
(B) prints argv.
(C) prints the number of command line arguments.
(D) prints the name of the executable code file.

Ans: D
argv[0] represent the filename where the executable code of the program is stored.

Q.38 `printf("%-10s", "ABDUL");` displays
(A) ABDUL␣␣␣␣␣ **(B)** ␣␣␣␣␣ABDUL
(C) ABDUL␣␣␣␣␣␣␣␣␣␣ **(D)** ␣␣␣␣␣␣␣␣␣␣ABDUL

Ans: A
-10s will print ABDUL in 10 space ABDUL followed by 5 blank space.

Q.39 Which amongst the following expression uses bitwise operator?
(A) a++ **(B)** !a>5
(C) a|b **(D)** a!=b

Ans: C
| is bitwise OR.

Q.40 The output of the following program is

```
main( )
{    float y;
     y=198.7361;
     printf("%7.2f", y);
}
```

(A)

1	9	8	.	7	3	6

(B)

1	9	8	.	7	3	

(C)

	1	9	8	.	7	4

(D)

1	9	8	.	7	4	

Ans: C
The printf statement is giving formatted output till two places of decimal.

Q.41 Which is not dynamic memory allocation function?

(A) malloc **(B)** free
(C) alloc **(D)** calloc

Ans: C
Three dynamic memory allocation functions are: malloc, calloc and free

Q.42 Which header file is used for screen handling function:-

(A) IO.H **(B)** STDLIB.H
(C) CONIO.H **(D)** STDIO.H

Ans: D
The header file stdio.h contains definitions of constants,macros and types, along with function declarations for standard I/O functions.

Q.43 Choose the directive that is used to remove previously defined definition of the macro name that follows it -

(A) # remdef **(B)** # pragma
(C) # undef **(D)** # define

Ans: C
The preprocessor directive #undef OKAY would cause the definition of OKAY to be removed from the system.

Q.44 The output of the following is

```
x = 'a';
printf("%d", x);
```

(A) 'a' **(B)** a
(C) 97 **(D)** None of the above

Ans: C
The printf statement is printing ascii value of a, that is 97.

Q.45 Consider the following statement

```
int j, k, p;
float q, r, a;
a = j/k;
p=q/r;
If q=7.2, r=20, j=3, k=2
```

The value of a and p is

(A) a=1.5, p=3.6 **(B)** a=2, p=3
(C) a=1.5, p=4 **(D)** a=1, p=3

Ans: C
a=3/2=1.5 and p=q/r=7.2/2=3.6 is rounded off to 4.

Q.46 Choose the function that returns remainder of x/y -

(A) remainder( ) **(B)** mod( )
(C) modulus( ) **(D)** rem( )

Ans: C

modulus( ) function produces the reminder of an integer division.

Q.47 What is the output of following program:-

```
int q, *p, n;
q = 176;                If the address of q is 2801
p = &q;                 and p is 2600
n = *p;
printf("%d", n);
```

(A) 2801 **(B)** 176
(C) 2600 **(D)** None of the above

Ans: B
n is assigned a the value which is present at the address of q and that value is 176.

Q.48 Consider the following statements-

```
x = 5;
y = x >3 ? 10 : 20;
```

The value of y is
(A) 10 **(B)** 20
(C) 5 **(D)** 3

Ans: A
Since x=5 is greater than 3 so y is assigned value 10. It is equivalent to if-else statements.

Q.49 Determine which of the following is an invalid character constant.
(A) '\a' **(B)** 'T'
(C) '\0' **(D)** '/n'

Ans: D
newline character constant is "\n" not "/n".

Q.50 What is the name of built-in function for finding square roots?
(A) square(x) **(B)** sqr(x)
(C) sqrt(x) **(D)** No built-in function

Ans: C
sqrt(x) is a built-in function for finding square roots.

Q.51 What is the output of following statement?

```
for(i=1; i<4; i++)
printf("%d",(i%2) ? i : 2*i);
```

(A) 1 4 3 **(B)** 1 2 3
(C) 2 4 6 **(D)** 2 2 6

Ans: A
for i=1, (i%2) is true so the statement will print 1; for for i=2, (i%2) is false so the statement will print 2*i=2*2=4; for for i=3, (i%2) is again true so the statement will print 3; for i=4, the statement is out from the loop.

Q.52 Which of the following statement is true about a function?
(A) An invoking function must pass arguments to the invoked function.
(B) Every function returns a value to the invoker.

(C) A function may contain more than one return statement.
(D) Every function must be defined in its own separate file.

Ans: A
An invoking function must pass arguments to the invoked function.

Q.53 What is the output of the following program?

```
main( )
{
    int i=4, z=12;
    if(i=5 || z>50)
       printf("hello");
    else
       printf("hye");
}
```

(A) hello **(B)** hye
(C) syntax error **(D)** hellohye

Ans: A
i=5 will assign value 5 to i so the if statement (i=5||z>50) is true so "printf" statement will print hello.

Q.54 For implementing recursive function the data structure used is:
(A) Queue **(B)** Stack
(C) Linked List **(D)** Tree

Ans: B
For implementing recursive function, stack is used as a data structure.

Q.55 The size of array int a[5]={1,2} is
(A) 4 **(B)** 12
(C) 10 **(D)** 6

Ans: C
The size of int array is 2*5=10 bytes as int takes 2 bytes of storage.

Q.56 Which of the following is not an escape sequence?
(A) \n **(B)** \r
(C) \' **(D)** \p

Ans: D
\p is not an escape sequence.

Q.57 The output of the following statements is

```
char ch[6]={'e', 'n', 'd', '\0', 'p'};
printf("%s", ch);
```

(A) endp **(B)** end0p
(C) end **(D)** error

Ans: C
printf statement will print end because string is terminated at "\0" and in array after d, we have null character.

Q.58 How many times the following code prints the string "hello".

```
for(i=1; i<=1000; i++);
printf("hello");
```

(A) 1 **(B)** 1000
(C) Zero **(D)** Syntax error

Ans: A
The "for" loop is terminated by a semicolon so the next statement is execute that is printing hello.

Q.59 Find the invalid identifiers from the following:-
(i) nA (ii) 2nd (iii) ROLL NO (iv) case
(A) (i), (ii) and (iv) **(B)** (i) and (iii)
(C) (ii), (iii) and (iv) **(D)** (ii), (i) and (iii)

Ans: C
Identifier cannot start with a digit; it cannot have a space and case is a keyword.

Q.60 The void type is used for

(A) Returning the value **(B)** creating generic pointers
(C) Creating functions **(D)** Avoid error

Ans: B
The void type is used to create generic pointers.

Q.61 The valid octal constants from the following
(i) 0245 (ii) 0387 (iii) 04.32 (iv) –0467
(A) (i) and (ii) **(B)** (iii) and (iv)
(C) (ii) and (iii) **(D)** (i) and (iv)

Ans: D
(i) and (iv) are valid octal constants.

Q.62 The variable that are declared outside all the functions are called _____.
(A) Local variable **(B)** Global variable
(C) Auto variable **(D)** None of the above
Ans: B
The variables that are declared outside all functions are called global variable.

Q.63 Consider the following statements:-

```
int x = 6, y=8, z, w;
y = x++;
z = ++x;
```

The value of x,y,z by calculating the above expressions are:-
(A) y=8, z=8, x=6 **(B)** y=6, x=8, z=8
(C) y=9, z=7, x=8 **(D)** y=7, x=8, z=7

Ans: B
y is assigned value of x that is 6, then x in incremented that is value of x=7, z is assigned value of x after incrementing that is z =8 so value of x =8.

Q.64 To declare an array S that holds a 5-character string, you would write

(A) char S[5] **(B)** String S[5]
(C) char S[6] **(D)** String S[6]

Ans: A
A string is nothing but a char array.

Q.65 The function used to read a character from a file that has been opened in read mode is

(A) putc **(B)** getc
(C) getchar **(D)** putchar

Ans: B
getc is used to read a character from a file that has been opened in read mode.

Q.66 The function that allocates requested size of bytes and returns a pointer to the first byte of the allocated space is -

(A) realloc **(B)** malloc
(C) calloc **(D)** none of the above

Ans: B
malloc allocates requested size of bytes and returns a pointer to the first byte of the allocated space.

Q.67 The constructed datatype of C is known as

(A) Pointers **(B)** String
(C) Structure **(D)** Array

Ans: C
Structure is a constructed datatype of C

Q.68 The postfix form of the following infix notation is : $(A+B)*(C*D-E)*F$

(A) $AB+CD*E-*F*$ **(B)** $AB+CDE+-*F*$
(C) $AB+CD-EF+-**$ **(D)** $ABCDEF*-+*+$

Ans: (A)

Q.69 The number of nodes in a complete binary tree of depth d (with root at depth 0) is

(A) $2^{d-1}+1$ **(B)** $2^{d+1}-1$
(C) $2^{d-1}-1$ **(D)** $2^{d+1}+1$

Ans: (B)

Q.70 The average case of quick sort has order

(A) $O(n^2)$ **(B)** $O(n)$
(C) $O(n \log n)$ **(D)** $O(\log n)$

Ans: (C)

Q.71 Inorder to get the information stored in a BST in the descending order, one should traverse it in which of the following order?

(A) left, root, right **(B)** root, left, right
(C) right, root, left **(D)** right, left, root

Ans: (C)

Q.72 Every internal node in a B-tree of minimum degree 2 can have

(A) 2, 3 or 4 children **(B)** 1, 2 or 3 children
(C) 2, 4 or 6 children **(D)** 0, 2 or 4 children

Ans: (B)

Q.73 Which sorting algorithm is the best if the list is already in order?

(A) Quick sort **(B)** Merge sort
(C) Insertion sort **(D)** Heap sort

Ans: (C)

Q.74 In _____ the difference between the height of the left sub tree and height of right sub tree, for each node, is not more than one

(A) BST **(B)** Complete Binary Tree
(C) AVL-tree **(D)** B-tree

Ans: (C)

Q.75 The number of comparisons required to sort 5 numbers in ascending order using bubble sort is

(A) 7 **(B)** 6
(C) 10 **(D)** 5

Ans: (C)

Q.76 The complexity of adding two matrices of order m*n is

(A) m + n **(B)** mn
(C) max(m, n) **(D)** min(m, n)

Ans: (B)

Q.77 The second largest number from a set of n distinct numbers can be found in

(A) O(n) **(B)** O(2n)
(C) $O(n^2)$ **(D)** O(log n)

Ans: (A)

Q.78 If the inorder and preorder traversal of a binary tree are D,B,F,E,G,H,A,C and A,B,D,E,F,G,H,C respectively then the postorder traversal of that tree is

(A) D,F,G,A,B,C,H,E **(B)** F,H,D,G,E,B,C,A
(C) C,G,H ,F,E,D,B,A **(D)** D,F,H,G,E,B,C,A

Ans: (D)

Q.79 In a binary tree, the number of terminal or leaf nodes is 10. The number of nodes with two children is

(A) 9 **(B)** 11
(C) 15 **(D)** 20

Ans: (A)

Q.80 Which amongst the following cannot be a balance factor of any node of an AVL tree?

(A) 1 **(B)** 0
(C) 2 **(D)** –1

Ans: (C)

Q.81 How many distinct binary search trees can be formed which contains the integers 1, 2, 3?

(A) 6 **(B)** 5
(C) 4 **(D)** 3

Ans: (B)

Q.82 The sort which inserts each elements A(K) into proper position in the previously sorted sub array A(1), ..., A(K–1)

(A) Insertion sort **(B)** Radix sort
(C) Merge sort **(D)** Bubble sort

Ans: (A)

Q.83 Direct or random access of elements is not possible in

(A) Linked list **(B)** Array
(C) String **(D)** None of these

Ans: (A)

Q.84 Level of any node of a tree is

(A) Height of its left subtree minus height of its right subtree
(B) Height of its right subtree minus height of its left subtree
(C) Its distance from the root
(D) None of these

Ans: (C)

Q.85 A desirable choice for the partitioning element in quick sort is

(A) First element of the list
(B) Last element of the list
(C) Randomly chosen element of the list
(D) Median of the list

Ans: (A)

Q.86 lg (n!) =_____

(A) O (n) **(B)** O (lg n)
(C) O (n^2) **(D)** O (n lg n)

Ans: (D)

$n! = n(n-1)(n-2)\text{-----}3X2X1$
$\geq (n/2)^{n/2}$
$\log n! \geq n/2 \log n/2$
$\geq n/2(\log n - \log 2)$
$\geq n/2 (\log n - 1)$
$\leq n \log n$
$= O(n \log n)$

Q.87 The result of evaluating the following postfix expression is
5, 7, 9, *, +, 4, 9, 3, /, +, -

(A) 50 **(B)** 65
(C) 61 **(D)** 69

Ans: (C)

Q.88 A graph with **n** vertices will definitely have a parallel edge or self loop if the total number of edges are

(A) more than **n** **(B)** more than **n+1**
(C) more than **(n+1)/2** **(D)** more than **n(n-1)/2**

Ans: (D)

Q.89 Out of the following, the slowest sorting procedure is

(A) Quick Sort **(B)** Heap Sort
(C) Shell Sort **(D)** Bubble Sort

Ans: (D)

Q.90 In _____, it is possible to traverse a tree without using stacks either implicitly or explicitly.

(A) Threaded binary trees. **(B)** AVL Tree
(C) B^+ tree **(D)** Heap

Ans: (C)

Q.91 The order of a B-Tree with 2, 3, 4 or 5 children in every internal node is

(A) 2 **(B)** 3
(C) 4 **(D)** 5

Ans: (C)

Q.92 The number of nodes that have no successors in a complete binary tree of depth 4 is

(A) 0 **(B)** 8
(C) 16 **(D)** 4

Ans: (B)

Q.93 One can make an exact replica of a Binary Search Tree by traversing it in
(A) Inorder **(B)** Preorder
(C) Postorder **(D)** Any order

Ans: (B)

Q.94 A complete Binary Tree with 15 nodes contains_____edges
(A) 15 **(B)** 30
(C) 14 **(D)** 16

Ans: (C)

Q.95 The minimum number of comparisons required to find the largest number from 4 different numbers are
(A) 4 **(B)** 3
(C) 5 **(D)** 6

Ans: (B)

Q.96 An infix expression can be converted to a postfix expression using a
(A) Stack **(B)** Queue
(C) Dequeue **(D)** None of these

Ans: (A)

Q.97 A data structure in which an element is added and removed only from one end, is known as
(A) Queue **(B)** Stack
(C) In-built structure **(D)** None of the above

Ans: (B)

Q.98 A complete binary tree with the property that the value of each node is at least as large as the values of its children is known as
(A) Binary Search Tree. **(B)** AVL Tree.
(C) Heap. **(D)** Threaded Binary Tree.

Ans: (C)

Q.99 A sorting algorithm is stable if
(A) its time complexity is constant irrespective of the nature of input.
(B) preserves the original order of records with equal keys.
(C) its space complexity is constant irrespective of the nature of input.
(D) it sorts any volume of data in a constant time.

Ans: (B)

Q.100 A tree in which, for every node, the difference between the height of its left subtree and right subtree is not more than one is

(A) AVL Tree. **(B)** Complete Binary Tree.

(C) B – Tree. **(D)** B^+ Tree.

Ans: (A)

Q.101 The data structure needed to convert a recursion to an iterative procedure is

(A) Queue. **(B)** Graph.

(C) Stack. **(D)** Tree.

Ans: (C)

Q.102 A binary tree stored using linked representation can be converted to its mirror image by traversing it in

(A) Inorder. **(B)** Preorder.

(C) Postorder. **(D)** Any order.

Ans: (B)

Q.103 The prefix form of an infix expression A+B-C*D is

(A) +AB-*CD. **(B)** -+A B C * D.

(C) -+A B * C D. **(D)** - + *ABCD.

Ans: (C)

Q.104 The number of edges in a simple, n-vertex, complete graph is

(A) n*(n-2). **(B)** n*(n-1).

(C) n*(n-1)/2. **(D)** n*(n-1)*(n-2)

Ans: (C)

Q.105 The largest and the second largest number from a set of n distinct numbers can be found in

(A) O (n). **(B)** O (2n).

(C) O $\left(n^2\right)$. **(D)** O (log n).

Ans: (A)

Q.106 To implement Sparse matrix dynamically, the following data structure is used

(A) Trees **(B)** Graphs

(C) Priority Queues **(D)** Linked List

Ans: (D)

Q.107 The depth $d_{n,}$ of complete binary tree of n nodes, where nodes are labeled from 1 to n with root as node 1 and last leaf node as node n is

(A) $\lfloor \log_2 n - 1 \rfloor$ **(B)** $\lfloor \log_2 n + 1 \rfloor$

(C) $\lceil \log_2 n + 1 \rceil$ **(D)** $\lceil \log_2 n - 1 \rceil$

Ans: (C)

Q.108 The balance factor for an AVL tree is either
(A) 0,1 or –1 **(B)** –2,–1 or 0
(C) 0,1 or 2 **(D)** All the above

Ans: (A)

Q.109 Applications of Linked List are
(A) Simulation , event driven systems
(B) Postfix and prefix manipulations
(C) Dictionary systems, polynomial manipulations
(D) Fixed block storage allocation, garbage collection

Ans: (D)

Q.110 AVL trees have LL, LR, RR, RL rotations to balance the tree to maintain the balance factor (LR : Insert node in Right sub tree of Left sub tree of node A, etc). Among rotations the following are single and double rotations
(A) LL, RL and LR, RR **(B)** LL, RR and LR, RL
(C) LR, RR and LL, RL **(D)** LR, RL and LR, RL

Ans: (B)

Q.111 Hashing collision resolution techniques are
(A) Huffman coding, linear hashing **(B)** Bucket addressing, Huffman coding
(C) Chaining, Huffman coding **(D)** Chaining, Bucket addressing

Ans: (D)

Q.112 The running time of the following sorting algorithm depends on whether the partitioning is balanced or unbalanced
(A) Insertion sort **(B)** Selection sort
(C) Quick sort **(D)** Merge sort

Ans: (C)

Q.113 Graphs are represented using
(A) Adjacency tree **(B)** Adjacency linked list
(C) Adjacency graph **(D)** Adjacency queue

Ans: (B)

Q.114 The average case complexity of Insertion Sort is
(A) $O(2^n)$ **(B)** $O(n^3)$
(C) $O(n^2)$ **(D)** $O(2n)$

Ans: (C)

Q.115 Infinite recursion leads to
(A) Overflow of run-time stack **(B)** Underflow of registers usage
(C) Overflow of I/O cycles **(D)** Underflow of run-time stack

Ans: (A)

Q.116 The number of unused pointers in a complete binary tree of depth 5 is
(A) 4 **(B)** 8
(C) 16 **(D)** 32

Ans: (C)

Q.117 The running time for creating a heap of size n is
(A) O (n) **(B)** O (log n)
(C) O (n log n) **(D)** O (n^2)

Ans: (C)

Q.118 What would be returned by the following recursive function after we call test (0, 3)

```
int test (int a, int b)
{
     if (a==b) return (1);
               else if (a>b) return(0);
               else return (a+test(a+1, b));
 }
```

(A) 1 **(B)** 2
(C) 3 **(D)** 4

Ans: (D)

Q.119 The extra key inserted at the end of the array is called a
(A) End Key **(B)** Stop Key
(C) Sentinel **(D)** Transposition

Ans: (C)

Q.120 Which of the following operations is performed more efficiently by doubly linked list than by singly linked list
(A) Deleting a node whose location is given.
(B) Searching of an unsorted list for a given item.
(C) Inserting a new node after node whose location is given.
(D) Traversing the list to process each node.

Ans: (A)

Q.121 One can determine whether a Binary tree is a Binary Search Tree by traversing it in
(A) Preorder **(B)** Inorder
(C) Postorder **(D)** Any of the three orders

Ans: (B)

Q.122 The spanning tree of connected graph with 10 vertices contains
(A) 9 edges **(B)** 11 edges
(C) 10 edges **(D)** 9 vertices

Ans: (A)

Q.123 A sorted file contains 16 items. Using binary search, the maximum number of comparisons to search for an item in this file is

(A) 15 **(B)** 8
(C) 1 **(D)** 4

Ans: (D)

Q.124 One can determine whether an infix expression has balanced parenthesis or not by using

(A) Array **(B)** Queue
(C) Stack **(D)** Tree

Ans: (C)

Q.125 The average number of key comparisons done in successful sequential search in a list of length n is

(A) log n **(B)** (n-1)/2
(C) n/2 **(D)** (n+1)/2

Ans: (D)

Q.126 The maximum number of nodes in a binary tree of depth 5 is

(A) 31 **(B)** 16
(C) 32 **(D)** 15

Ans: (A)

Q.127 n elements of a Queue are to be reversed using another queue. The number of "ADD" and "REMOVE" operations required to do so is

(A) 2*n **(B)** 4*n
(C) n **(D)** The task cannot be accomplished

Ans: (D)

Q.128 A complete binary tree with n leaf nodes has

(A) n+1 nodes **(B)** 2n-1 nodes
(C) 2n+1 nodes **(D)** n(n-1)/2 nodes

Ans: (B)

Q.129 A binary tree can be converted in to its mirror image by traversing it in

(A) Inorder **(B)** Preorder
(C) Postorder **(D)** Anyorder

Ans: (B)

Q.130 One can convert an infix expression to a postfix expression using a

(A) stack **(B)** Queue
(C) Deque **(D)** none of these

Ans: (A)

Q.131 Which of the following types of expressions do not require precedence rules for evaluation?

(A) fully parenthesised infix expression
(B) postfix expression
(C) partially parenthesised infix expression
(D) more than one of the above

Ans: (A)

Q.132 Overflow condition in linked list may occur when attempting to_____

(A) Create a node when free space pool is empty.
(B) Traverse the nodes when free space pool is empty.
(C) Create a node when linked list is empty.
(D) None of these.

Ans: (A)

Q.133 Linked lists are not suitable data structures for which one of the following problems

(A) insertion sort **(B)** binary search
(C) radix sort **(D)** polynomial manipulation

Ans: (B)

Q.134 The sorting technique where array to be sorted is partitioned again and again in such a way that all elements less than or equal to partitioning element appear before it and those which are greater appear after it, is called

(A) merge sort **(B)** quick sort
(C) selection sort **(D)** none of these

Ans: (B)

Q.135 The search technique for searching a sorted file that requires increased amount of space is

(A) indexed sequential search (B) interpolation search
(C) sequential search (D) tree search

Ans: (A)

The search technique for searching a sorted file that requires increased amount of space is indexed sequential search. Because in this search technique we need to maintain a separate index file which requires additional storage space.

Q.136 The postfix form of A ^ B * C - D + E/ F/ (G + H),

(A) AB^C*D-EF/GH+/+ (B) AB^CD-EP/GH+/+*

(C) ABCDEFGH+//+-*^ (D) AB^D +EFGH +//*+

Ans: (A)

Q.137 The prefix of (A+B)*(C-D)/E*F

(A) /+-AB*CD. (B) /*+-ABCD*EF.

(C) */*+AB-CDEF. (D) **AB+CD/EF.

Ans: (C)
Prefix of (A+B) * (C - D) / E*F
(+AB) * (-CD) / E*F
*+AB-CD E*F
/+AB-CDEF

Q.138 A sorting technique which uses the binary tree concept such that label of any node is larger than all the labels in the subtrees, is called
(A) selection sort. (B) insertion sort.

(C) Heap sort. (D) quick sort.

Ans: (C)
A Sorting technique which uses the binary tree concept such that label of any node is larger than all the, labels in the sub trees, is called Heap sort because heap sort works on a complete binary tree with the property that the value at any node 'N' of the tree should be greater than or equal to the value at all its children nodes.

Q.139 A full binary tree with 'n' non-leaf nodes contains

(A) $\log_2$ n nodes . (B) n+1 nodes.

(C) 2n nodes. (D) 2n+l nodes.

Ans: (D)

Q.140 A graph 'G' with 'n' nodes is bipartite if it contains
(A) n edges. (B) a cycle of odd length.
(C) no cycle of odd length. (D) n^2 edges.

Ans: (C)

Q.141 Recursive procedures are implemented by using ____ data tructure.
(A) queues. (B) stacks.

(C) linked lists. (D) strings.

Ans: (B)
Recursive procedures are implemented by using stacks because stacks are LIFO data structure and we need this feature to store return addresses of various recursive calls in recursive procedures.

Q.142 In _____, the difference between the height of the left sub tree and height of the right tree, for each node, is almost one.

(A) Binary search tree (B) AVL - tree

(C) Complete tree (D) Threaded binary tree

Ans: (B)

PART – II

DESCRIPTIVES

Q.1 Write a C program to compute the sum of first n terms (n ≥ 1) of the following series using 'for' loop.

1 – 3 + 5 – 7 + 9 - …… **(6)**

Ans:

A C program to compute the sum of first n terms of the series 1-3+5-7+9-… is listed below:

```
main()
{
 int start=1,i=1,no=0,sum=0;
 clrscr();
 printf ("\nEnter number of terms to be added:-> ");
 scanf("%d",&no);
 for (i=1;i<=no;i++)
 {
  if (i%2!=0)
  {
         sum+=start;
         if (i==1)
                printf ("%d",start);
         else
                printf ("+%d",start);
  }
  else
  {
         sum-=start;
         printf ("-%d",start);
  }
  start+=2;
 }
 printf ("=%d",sum);
 getch();
}
```

Q.2 Write a C program to convert a binary number to its corresponding octal number. **(8)**

Ans:

A C program to convert a binary number to its binary octal number is as follows:

```
main()
{
 long int bin_no,no;
 int oct_no,oct_p,i,rem,pow2,pow8,inter_oct;
 clrscr();
 printf ("\nEnter the Binary number: -> ");
 scanf("%ld",&bin_no);
 no=bin_no;
 pow8=1;
 oct_no=0;
 while (no>0)
 {
        i=0;
        inter_oct=0;
        pow2=1;
        while (i<=2)
        {
               if (no==0)break;
```

```
                rem=no%10;
                inter_oct+=rem*pow2;
                i++;
                pow2*=2;
                no/=10;
            }
            oct_no+=inter_oct*pow8;
            pow8*=10;
        }
        printf ("\nOctal Equivalent for %ld = %d",bin_no,oct_no);
        getch();
    }
```

Q.3 Write a C program to print out n values of the following sequence.
1 –1 1 –1 1 … **(6)**

Ans:
A C program to print n values of following sequence 1 –1 1 –1 1 … is listed below:

```
#include<stdio.h>
#include<conio.h>
main()
{
int i,j,n,k;
clrscr();
printf("\n enter number \n");
scanf("%d",&n);
k=1;
for(i=1;i<=n;i++)
{
for(j=1;j<=i;j++)
printf("%d",k);
printf("\t");
}
getch();
}
```

Q.4 Write a C program to test whether a given pair of numbers are amicable numbers. (Amicable number are pairs of numbers each of whose divisors add to the other number) **(8)**

Ans:
A C program to test whether a given pair of numbers is amicable numbers is as follows:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
int i,j,n,sum_i=0,sum_j=0;
clrscr();
printf("enter any two numbers  >");
scanf("%d%d",&i,&j);
 for(n=1;n<i;n++)
 {
  if (i%n==0)
  sum_i+=n;
 }
 for (n=1;n<j;n++)
```

```
 {
  if (j%n==0)
  sum_j+=n;
 }
if ((sum_j==i) && (sum_i==j))
 printf ("\nAmicable");
else
 printf ("\nNot Amicable");
getch();
}
```

Q.5 Write a C program to rearrange the elements of an array so that those originally stored at odd suffixes are placed before those at even suffixes. **(6)**

Ans:

A C program to rearrange the elements of an array so that those originally stored at odd suffixes are placed before those at even suffixs:

```
main()
{
 char a[25],ch,temp;
 int i;
 clrscr();
 printf ("\nEnter the string:-> ");
 gets(a);
 for (i=0;a[i]!='\0';i++)
 {
      if (a[i+1]=='\0')break;
      temp=a[i];
      a[i]=a[i+1];
      a[i+1]=temp;
      i++;
 }
 puts(a);
 getch();
}
```

Q.6 Admission to a college in science branch is given if the following conditions are satisfied

(i) Maths marks >= 80
(ii) Physics marks >= 75
(iii) Chemistry marks >= 70
(iv) Total percentage in all three subjects >= 80

Given the marks in three subjects, write a program to process the applications to list the eligible candidates. **(8)**

Ans:

A C program to process the applications to list the eligible candidates is listed below:

```
 #include<stdio.h>
 #include<conio.h>
void main()
{
 float math,phy,che;
 float per;
 char ch;
 clrscr();
 printf("\nDo u want to enter any record:-> ");
 ch=getch();
```

```
while(ch=='y' || ch=='Y')
{
 clrscr();
 printf("\nEnter Marks in Math:-> ");
 scanf("%f",&math);
 printf("\nEnter Marks in Physics:-> ");
 scanf("%f",&phy);
 printf("\nEnter Marks in Chemistry:-> ");
 scanf("%f",&che);
 per=(math+phy+che)/3.00;
 if(math>=80.00 && phy>=75.00 && che>=70.00 && per>=80.00)
       printf("\nYou are eligible for selection.");
 else
       printf("\nSorry you are not eligible!!");
 printf("\nDo u want to enter new record:-> ");
 ch=getch();
}
printf ("\nThanks for using it!");
getch();
}
```

Q.7 Write a C program using while loop to reverse the digits of a given number. (for example, If number is=12345 then output number is= 54321) **(5)**

Ans:

A C program to reverse the digits of a given number:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int n,r;
clrscr();
printf("enter an integer");
scanf("%d",&n);
printf("\nreverse of %d : ",n);
while(n>0)
{
r=n%10;
printf("%d",r);
n=n/10;
}
getch();
}
```

Q.8 Write C program to produce 10 rows of the following form of Floyd's triangle **(5)**

```
1
0   1
1   0   1
0   1   0   1
```

Ans:

A C program to produce 10 rows of Floyd's triangle is:

```
#include<conio.h>
void main()
{
  int i,j;
clrscr();
for(i=1;i<=10;i++)
{
```

```
   for(j=1;j<=i;j++)
   {    printf(" ");
 if(i%2==0)
     if(j%2==0)
        printf("1");
     else
        printf("0");
      else
        if(j%2==0)
          printf("0");
        else
          printf("1");
        }
 printf("\n\n");
 }
   getch();
}
```

Q.9 Consider the following macro definition

```
#define root (a, b) sqrt((a) * (a) + (b) * (b))
```

What will be the result of the following macro call statement

```
root(a++, b++) if a = 3 and b = 4
```

(3)

Ans:

Result of this macro is:5

sqrt(3*3+4*4)

sqrt(9+16)=sqrt(25)=5

Q.10 Write a nested macro that gives the minimum of three values. **(3)**

Ans:

A nested macro that gives the minimum of three variables is listed below:

```
#include<stdio.h>
#include<conio.h>
#define min(a,b) ((a>b)?b:a)
#define minthree(a,b,c) (min(min(a,b),c))
void main()
{
int x,y,z,w;
clrscr();
printf("enter three numbers :\n");
scanf("%d%d%d",&x,&y,&w);
z=minthree(x,y,w);
printf("Minimum of three value is %d",z);
getch();
}
```

Q.11 Given are two one dimensional arrays A and B which are stored in ascending order. Write a program to merge them into a single sorted array C that contains every element of A and B in ascending order.

(8)

Ans:

A program to merge two arrays into single sorted array that contains every element of arrays into a ascending order:

```
#include<stdio.h>
#include<conio.h>
void sort(int*,int);
void merge(int*,int*,int,int);
void main()
```

```
{
 int a[10],b[10];
 int i,j,m,n;
 clrscr();
 printf("how many numbers u want to enter in 1st array : ");
   scanf("%d",&n);
 printf("enter numbers in ascending order :\n");
 for(i=0;i<n;i++)
   scanf("%d",&a[i]);
 printf("how many numbers u want to enter in 2nd array : ");
   scanf("%d",&m);
 printf("enter numbers in ascending order :\n");
 for(i=0;i<m;i++)
   scanf("%d",&b[i]);
 merge(a,b,n,m);
  getch();
}
void merge(int *a,int *b,int n,int m)
{
 int i=0,c[20],j=0,k=0,count=0;
 while(i<=n&&j<=m)
 {
     if(a[i]<b[j])
    {
       c[k]=a[i];
       i++;
       k++;
    }
    if(a[i]>b[j])
   {
      c[k]=b[j];
      j++;
      k++;
    }
    if(a[i]==b[j])
   {
      c[k]=a[i];
      k++;
      i++;
      j++;
      count++;
    }
 }
 if(i<=n&&j==m)
 {
    while(i<=n)
    {
        c[k]=a[i];
        i++;
        k++;
     }
  }
 if(i==n&&j<=m)
 {
    while(j<=m)
   {
       c[k]=b[j];
       i++;
       j++;
    }
  }
    for(i=0;i<m+n-count;i++)
       printf("%d\t",c[i]);
```

```
}
```

Q.12 Write a C program that reads the text and counts all occurrences of a particular word. **(6)**

Ans:

A C program that reads the text and count all occurrences of a particular word:

```
#include <string.h>
       void main()
       {
char a[100],b[20],c[20];
int i,count=0,k=0,len;
clrscr();
printf ("\nEnter a string:-> ");
gets(a);
printf("\nEnter the word to be searched:-> ");
gets(b);
len=strlen(a);
for(i=0;i<=len;i++)
{
 //Assuming Space , Tab and NULL as word separator
 if(a[i]==32 || a[i]=='\t' || a[i]=='\0')
 {
       c[k]='\0';
       if(strcmp(b,c)==0)count++;
       k=0;
 }
 else
 {
       c[k]=a[i];
       k++;
 }
}
printf("Occurance of %s is = %d",b,count);
getch();
         }
```

Q.13 Write a C program that reads a string from keyboard and determines whether the string is palindrome or not. (A string is palindrome if it is read from left or right gives you the same string) **(8)**

Ans:

A C program to check if input string is palindrome or not:

```
#include<stdio.h>
      #include<conio.h>
      void main()
      {
 int i,j,k,flag=1;
 char a[20];
 clrscr();
 printf("enter any word:\n");
 scanf("%s",a);
 i=0;
 while(a[i]!='\0')
 i++;
 k=i;
 for(j=0;j<k/2;)
 {
       if(a[j]!=a[i-1])
       {
           flag=0;
```

```
            break;
          }
          if(a[j]==a[i-1])
         {
              j++;
              i--;
          }
}
if(flag==0)
printf("it is a not palindrome");
else
printf("it is a palindrome");
getch();
         }
```

Q.14 Write a C function strend(s, t), which returns 1 if the string t occurs at the end of the string s, and zero otherwise. **(7)**

Ans:

C function strend(s, t), which returns 1 if the string t occurs at the end of the string s, and zero otherwise.

```
#include<conio.h>
#include<string.h>
strend ()
{
 char s[100],t[20],c[20];
 int i,j,k=0,len1,len2;
 clrscr();
 printf("\n Enter the string : ");
 gets(s);
 printf("\n Enter the string to be searched for : ");
 gets(t);
 len1=strlen(s);
 len2=strlen(t);
 for(i=len1-(len2);i<=len1;i++)
 {
        c[k++]=s[i];
 }
 c[k]='\0';
 if(strcmp(t,c)==0)
        return 1;
 else
        return 0;
 getch();
}
```

Q.15 Write a function rightrot(x, n) that returns the value of the integer x rotated to the right by n positions. **(7)**

Ans:

A function rightrot(x, n) that returns the value of the integer x rotated to the right by n positions.

```
#include<conio.h>
rightrot(x,n)
{
 int x,n,k=0,num,rem,i,j,dig;
 int arr1[10],arr2[10];
 clrscr();
 printf("\n Enter the integer to be rotated : ");
 scanf("%d",&x);
```

```
 printf("\n Enter the positions by which %d is to be rotated
: ",x);
 scanf("%d",&n);
 for(num=x;num!=0;num/=10)
 {
        rem=num%10;
        arr1[k++]=rem;
 }
 i=k-1;
 k=0;
 for(;i>=0;i--)
        arr2[k++]=arr1[i];
 k-=1;
 for(i=1;i<=n;i++)
 {
        dig=arr2[k];
        for(j=k;j>0;j--)
               arr2[j]=arr2[j-1];
        arr2[0]=dig;
 }
 printf("\n\n The original number is : “);
  return x;
 printf("\n The rotated number is : ");
 for(i=0;i<=k;i++)
        printf("%d",arr2[i]);
 getch();
}
```

Q.16 Write recursive function in C to obtain n^{th} term of the Fibonacci series. **(7)**

Ans:

recursive function in C to obtain n^{th} term of the Fibonacci series

```
    #include<stdio.h>
    #include<conio.h>
     void main()
     {
int n;
void fibonac(int );
clrscr();
printf("enter the length for the series");
scanf("%d",&n);
printf("Fibonacci sequence upto %d terms is:\n\n",n);
fibonac(n);
getch();
      }
void fibonac(int n)
{ static int f1=0,f2=1;
   int temp;
   if(n<2)
   { f1=0;
     f2=1;
   }
  else
  {
     fibonac(n-1);
     temp=f2;
     f2=f1+f2;
     f1=temp;
  }
  printf("%5d",f1);
}
```

Q.17 Write C function named 'fiddle' that takes two arguments, x and y and changes both values. x is an int while y is a pointer to int **(7)**

Ans:

C function named 'fiddle' that takes two arguments, x and y and changes both values. x is an int while y is a pointer to int

```
#include<conio.h>
void main()
{
  int x,z,*y,temp;
  clrscr();
  printf("\n Enter two numbers : \n");
  scanf("%d %d",&x,&z);
  y=&z;
  printf("\n\n x = %d and y = %d",x,*y);
  fiddle(&x,y);
  printf("\n After changing their values ");
  printf("\n x = %d and y = %d",x,*y);
  getch();
}
fiddle(int *a,int *b)
{
  int temp;
  temp=*a;
  *a=*b;
  *b=temp;
  getch();
}
```

Q.18 What is an unsigned integer constant? What is the significance of declaring a constant as unsigned? **(4)**

Ans:

An integer constant is any number in the range -32768 to +32767, because an integer constant always occupies two bytes in memory and in two bytes we cannot store a number bigger than +32767 or smaller than -32768. Out of the two bytes to store an integer, the highest bit is used to store the sign of the integer. This bit is 1 if the number is negative and 0 if number is positive. Unsigned integer constant is an integer constant which has the permissible range from 0 to 65536. Thus significance of declaring a constant as unsigned almost doubles the size of the largest possible value. This happens because on declaring a constant as unsigned, the sixteenth bit is free and is not used to store the sign of the constant.

Q.19 Explain pointers and structures by giving an example of pointer to structure variable? **(5)**

Ans:

We can have a pointer pointing to a structure just the same way a pointer pointing to an int, such pointers are known as structure pointers. For example consider the following example:

```
#include<stdio.h>
#include<conio.h>
struct student
{
     char name[20];
     int roll_no;
};
```

```
void main()
{
     struct student stu[3],*ptr;
     clrscr();
     printf("\n Enter data\n");
     for(ptr=stu;ptr<stu+3;ptr++)
    {  printf("Name");
        scanf("%s",ptr->name);
        printf("roll_no");
        scanf("%d",&ptr->roll_no);
    }
    printf("\nStudent Data\n\n");
    ptr=stu;
    while(ptr<stu+3)
   {
        printf("%s   %5d\n",ptr->name,ptr->roll_no); ptr++;
    }
getch();
}
```

Here ptr is a structure pointer not a structure variable and dot operator requires a structure variable on its left. C provides arrow operator "->" to refer to structure elements. "ptr=stu" would assign the address of the zeroth element of stu to ptr. Its members can be accessed by statement like "ptr->name". When the pointer ptr is incremented by one, it is made to point to the next record, that is stu[1] and so on.

Q.20 Compare the following pairs of statements with respect to their syntax and function:

a. break and continue.
b. goto and break. **(4)**

Ans:

a. break and continue

Two keywords that are very important to looping are break and continue. The **break** command will exit the most immediately surrounding loop regardless of what the conditions of the loop are. Break is useful if we want to exit a loop under special circumstances.

```
#include <stdio.h>
void  main() {
 int a;
  printf("Pick a number from 1 to 4:\n");
 scanf("%d", &a);
  switch (a) {
   case 1:
     printf("You chose number 1\n");
     break;
   case 2:
     printf("You chose number 2\n");
     break;
   case 3:
     printf("You chose number 3\n");
     break;
   case 4:
     printf("You chose number 4\n");
     break;
   default:
     printf("That's not 1,2,3 or 4!\n");
 }
 getch();
```

```
}
```

Continue is another keyword that controls the flow of loops. If we are executing a loop and hit a continue statement, the loop will stop its current iteration, update itself (in the case of for loops) and begin to execute again from the top. Essentially, the continue statement is saying "this iteration of the loop is done; let's continue with the loop without executing whatever code comes after me."
The syntax of continue statement is simple continue;

b. goto and break

C support **goto statement** to branch unconditionally from one point to another in the program.The goto keyword is followed by a label, which is basically some identifier placed elsewhere in the program where the control is to be transferred.
During running of a program, the statement like "goto label1;" cause the flow of control to the statement immediately following the label "label1". We can have a forward jump or a backward jump.

```
#include <stdio.h>
        void main()
         {
 int attempt, number = 46;
 looping: /* a label */
  printf("Guess a number from 0-100\n");
  scanf("%d", &attempt);
  if(number==attempt) {
   printf("You guessed correctly!\n\n");
  }
  else {
   printf("Let me ask again...\n\n");
   goto looping; /* Jump to the label*/
  }
getch();
    }
```

Q.21 Explain the difference between the following:
(i) Program testing and debugging.
(ii) Top down and bottom up approaches.
(iii) Interpreted and compiled languages. **(6)**

Ans:

(i) Program testing and debugging:

Program testing is the process of checking program, to verify that it satisfies its requirements and to detect errors. These errors can be of any type-Syntax errors, Run-time errors, Logical errors and Latent errors. Testing include necessary steps to detect all possible errors in the program. This can be done either at a module level known as unit testing or at program level known as integration testing.
Debugging is a methodical process of finding and reducing the number of bugs in a computer program making it behave as expected. One simple way to find the location of the error is to use print statement to display the values of the variables. Once the location of the error is found, the error is corrected and debugging statement may be removed.

(ii) Top Down and Bottom up approaches

A **top-down** approach is essentially breaking down a system to gain insight into its compositional sub-systems. In a top-down approach an overview of the system is first formulated, specifying but not detailing any first-level subsystems. Each subsystem is then refined in greater detail, sometimes in many additional

subsystem levels, until the entire specification is reduced to base elements. In short the top down approach means decomposing of the solution procedure into subtasks. This approach produces a readable and modular code that can be easily understood and maintained.

A **bottom-up** approach is essentially piecing together systems to give rise to grander systems, thus making the original systems sub-systems of the emergent system. In a bottom-up approach the individual base elements of the system are first specified in great detail. These elements are then linked together to form larger subsystems, which then in turn are linked, sometimes in many levels, until a complete top-level system is formed.

(iii)Interpreted and Compiled Languages

In **interpreted languages**, the instructions are executed immediately after parsing. Both tasks are performed by the interpreter. The code is saved in the same format that we entered. Interpreted languages include the MS-DOS Batch language (the OS itself is the interpreter), shell scripts in Unix/Linux systems, Java, Perl etc. Advantages of interpreted languages include relative ease of programming and no linker requirement. Disadvantages include poor speed performance and that we do not generate an executable (and therefore distributable) program. The interpreter must be present on a system to run the program.

In **compiled languages** the instructions into machine code and store them in a separate file for later execution. Many modern compilers can compile (parse) and execute in memory, giving the 'appearance' of an interpreted language. However, the key difference is that parsing and execution occurs in two distinct steps. Examples of compiled languages include Visual Basic, C/C++, Delphi and many others. In a compiled environment, we have several separate files like: source code (the text instructions), object code (the parsed source code) and the executable (the linked object code). There is definitely an increase in complexity in using compilers, but the key advantages are speed performance and that one can distribute stand-alone executables.

Q.22 Write a complete C program for reading an employees file containing {emp_number, name, salary, address}. Create an output file containing the names of those employees along with their salary and address whose salary is > 20,000. **(8)**

Ans:

```
#include<stdio.h>
struct employee
{
 int code;
 char name[30];
 char address[50];
 char phone[10];
 float sal;
}emp,tmp;
FILE *fp1,*fp2;
void main()
{
 char ch='y';
 int i=1;
 //Assuming that the file already exists
 fp1=fopen("employee.txt","r");
 if (fp1==NULL)
 {
fp1=fopen("employee.txt","w");
if (fp1==NULL)
```

```
        {
              printf ("\nUnable to create Employee.txt file");
              getch();
              exit();
        }
        printf ("\nNo Data found in Employee.txt. Add new
      records");
        while (1)
        {
              printf ("\nEnter Employee Code:-> ");
              flushall();
              scanf ("%d",&emp.code);
              printf ("\nEnter Employee Name:-> ");
              flushall();
              gets(emp.name);
              printf ("\nEnter Employee Address:-> ");
              flushall();
              gets(emp.address);
              printf ("\nEnter Employee Phone Number:-> ");
              flushall();
              gets(emp.phone);
              printf ("\nEnter Employee Salary:-> ");
              flushall();
              scanf("%f",&emp.sal);
              //Writing records to Employee.txt file
              fwrite(&emp, sizeof(emp),1,fp1);
              printf ("\nDo u want to add more records (y/n):->
             ");
              flushall();
              scanf("%c",&ch);
              if (ch=='n' || ch=='N')break;
        }
        fclose(fp1);
        fp1=fopen("employee.txt","r");
}
printf ("\nList of Employees having Salary greater than
 20000\n");
printf ("\nCode\tName\t\tAddress\tPhone No.\tSalary\n");
i=1;
fp2=fopen("Output.txt","w");
getch();
if (fp2==NULL)
{
       printf ("\nUnable to create Output file\n");
       getch();
       exit();
}
while (1)
{
       i=fread(&emp,sizeof(emp),1,fp1);
       if(i==0)break;
       if (emp.sal>20000)
              fwrite (&emp,sizeof(emp),1,fp2);
}
fclose (fp2);
fp2=fopen("output.txt","r");
i=1;
while (1)
{
       i=fread(&emp,sizeof(emp),1,fp2);
       if(i==0)break;
```

```
 printf
("\n%d\t%s\t\t%s\t%s\t%f",emp.code,emp.name,emp.address,emp.phone,e
mp.sal);
 }
 getch();
 }
```

Q.23 Write a function to display the binary number corresponding to the integer passed to it as an argument. **(6)**

Ans:

```
#include<stdio.h>
#include<conio.h>
void dis_bin(int n)
{
  num=n;
  for(i=0;num!=0;i++)
  {
        q=num/2;
        r[i]=num%2;
        num=q;
  }
  clrscr();
  printf("\n\n The integer the number is : %d\n",n);
  printf("\n The binary conversion is : ");
  for(i-=1;i>=0;i--)
        printf("%d",r[i]);
  getch();
}
```

Q.24 Write a function, my_atoi, similar to the library function **atoi** that returns the numeric value corresponding to the string passed to it as an argument. **(6)**

Ans:

```
#include <stdio.h>
main()
{
    long int n=0;
    char str[25];
    clrscr();
    printf ("\nEnter the string:-> ");
    gets(str);
    my_atoi(str,&n);
    printf ("\nInteger Equvalent of string %s=%ld",str,n);
    getch();
}
my_atoi(char str[25],long int *num)
{
   int i=1;
   long int s=1,l=0,j;
   float po10=1;
   //Trimming the string for integers
   for (i=0;str[i]!='\0';i++)
   {
         if (str[i]==' ')
         {
               for (j=i;str[j]!='\0';j++)
                     str[j]=str[j+1];
                     i--;
         }
```

```
            else if (str[i]>=48 && str[j]<=57)
                  continue;
            else
            {
                  str[i]='\0';
                  break;
            }
      }
      l=strlen(str);
      for (i=l-1;i>=0;i--)
      {
       switch (str[i])
       {
       case 48: s*=0;break;
       case 49: s*=1;break;
       case 50: s*=2;break;
       case 51: s*=3;break;
       case 52: s*=4;break;
       case 53: s*=5;break;
       case 54: s*=6;break;
       case 55: s*=7;break;
       case 56: s*=8;break;
       case 57: s*=9;break;
       }
       *num+=s;
       po10*=10;
       s=po10;
      }
}
```

Q.25 Briefly explain what do you understand by stepwise refinement of the program? **(4)**

Ans:

Stepwise refinement of the Program:

The concept of "Stepwise refinement" means to take an object and move it from a general perspective to a precise level of detail and this cannot be done in one jump but in steps. The number of steps needed to decompose an object into sufficient detail is ultimately based on the inherent nature of the object. "Stepwise refinement" of program represents a "divide and conquer" approach to design. In simple words, break a complex program into smaller, more manageable modules that can be reviewed and inspected before moving to the next level of detail. Step-wise refinement is a powerful paradigm for developing a complex program from a simple program by adding features incrementally.

Some of the steps taken for refinement of a program are:

1. Analysis of the algorithm used to develop the program, and analysis of various parts of the program and then making appropriate changes to improve the efficiency of the program.
2. Keep the program simple to improve memory efficiency.
3. Evaluation and incorporation of memory compression features.
4. Debugging and testing performed first to module level and then to interface level.
5. Take care of correct working and clarity while making it faster.

Q.26 Write a function to remove duplicates from an ordered array. For example, if input is: a,a,c,d,q,q,r,s,u,w,w,w,w; then the output should be a,c,d,q,r,s,u,w. **(8)**

Ans:

A C program to remove duplicates from an odered array:

```
#include<stdio.h>
#include<conio.h>
#include<string.h>
void main()
{
   int i,j,k,l,flag=0;
   char a[50];
   clrscr();
   printf("enter the characters in the array");
   gets(a);
   l=strlen(a);
   for(i=0;i<l-1;i++)
    for(j=i+1;j<l;j++)
   {
      if(a[i]==a[j])
      { l=l-1;
        for(k=j;k<l;k++)
        a[k]=a[k+1];
        flag=1;
        j=j-1;
      }
   }
   if(flag==0)
      printf("No duplicates found");
   else
      for(i=0;i<l;i++)
   printf("%c",a[i]);
   getch();
}
```

Q.27 Write a function to sort the characters of the string passed to it as argument. **(8)**

Ans:

A C function to sort the characters of the string:

```
main()
{
 char a[100],ch;
 int i,j;
 printf ("\nEnter the string:-> ");
 gets(a);
 for (i=0;a[i]!='\0';i++)
 {
  for (j=i+1;a[j]!='\0';j++)
  {
         if (a[i]>a[j])
         {
                ch=a[i];
                a[i]=a[j];
                a[j]=ch;
         }
  }
 }
 printf ("\nString after sorting\n");
 puts(a);
 getch();
}
```

Q.28 Give the outputs of the following code segments, if any and justify your answers.

```
(i)   #define CUBE(x)                (x * x * x)
main( )
{
    printf("%d", CUBE(4+5));
}
(ii)  int j = 5;
        printf("%d", j = j == 6);
        printf("%d", j = ++j == 6);
(iii)for (j = 0; j = 3; j++)
        printf("%d", j);
(iv)  main( )
    {
      static char a[ ] = "Test String";
      static char *b = "Test String";
      printf("%d %d", sizeof(a), sizeof(b));
    }
(v) main( ) {
    enum test {RED, BLUE, GREEN};
    enum test t = BLUE;
    printf("%d", t);
    }
(vi)  main( ) {
      union U { int j; char c; float f; } u;
      u.j = 10; u.c = 'A'; u.f = 99.99;
      printf("u.j = %d u.c = %c u.f = %f", u.j, u.c, u.f);
       }
```

(2 x 6)

Ans:

(i) Output is: 49
macro cube(x *x*x) is expanded into (4+5*4+5*4+5) which is evaluated as 4+20+20+5=49 due to priority of operators. So output is 49.

(ii) Output is: 0 0
Both the expressions are evaluated as 0 so the output is 0.

(iii) Output is: infinite loop
There is an incorrect assignment, j=3 in test expression.

(iv) Output is:12 2,
The printf is printing the size of char array a and char b, null char is included.

(v) Output is:1,
The compiler automatically assigns integer digits beginning with 0 to all the enumeration constants so BLUE has value 1.

(vi) Output is: u.j=1311,u.c='β',u.f=99.989998
The first two outputs are erroneous output which is machine dependent. During accessing a union member, we should make sure that we are accessing the member whose value is currently stored otherwise we will get errors.

Q.29 What are preprocessor directives? List three types of them. What is the difference between the following directives: #include <filename> and #include "filename"?

(4)

Ans:

Preprocessor directives: These are the commands given to a program known as preprocessor that processes the source code before it passes through the compiler. Each of these preprocessor directives begins with a # symbol. These can be placed anywhere in the program but usually placed before main ( ) or at the beginning of the program. Before the source code passes through the compiler, it is examined by the preprocessor for any directives. If there are any, appropriate action is taken and the source program is handed over to compiler. These directives can be divided into following three categories:

1. Macro substitution directives
2. File inclusion directives.
3. Compiler control directives.

#include "filename": The search for the file is made first in the current directory and then in the standard directories as mentioned in the include search path.

#include <filename>: This command would look for the file in the standard list of directories.

Both of these directives cause the entire contents of filename to be inserted into the source code at that point in the program.

Q.30 Write a function to compute the frequency of 'the' in a file. **(8)**

Ans:

A C function to compute the frequency of 'the':

```
void main()
{
  int i=0,j=0,k,n=0,m=0,l;
  char a[100],b[10],c[10]="the";
  clrscr();
  printf("enter the sentence");
  gets(a);
  while(a[i]!='\0')
  {
       if(a[i]==' '|| a[i]=='.')
       m++;
       i++;
  }
  i=0;
  l=0;
  while(l!=m+1)
  {
     while(a[i]!=' ' && a[i]!='.' && a[i]!='\0')
     {
         b[j]=a[i];
         j++;
         i++;
     }
     b[j]='\0';
     k=strcmp(b,c);
     if(k==0)
     {
         n++;
         j=0;
         if(a[i]!='\0')
          i++;
     }
     else
```

```
        {
            j=0;
            i++;
        }
        l++;
    }
    if(n!=0)
    printf("it is present %d times",n);
    else
    printf("it is not present");
    getch();
}
```

Q.31 Write a recursive function to print the reverse of a string passed to it as an argument. **(8)**

Ans:

A recursive function to print the reverse of a string is given below:

```
void main()
{
   char str[100];
   clrscr();
   printf("enter a string\n");
   gets(str);
   printf("reverse of the entered string is\n");
   rev(str);
   getch();
 }
    rev (char *string)
    {
    if (*string)
    {
       rev(string+1);
     putchar(*string);
    }
}
```

Q.32 Write a program which accepts two file names from the command line and prints whether the contents of the two files are same or not. If not, then print the first line number and then the contents of the lines from which they differ. Assume that the lines in both the files have at most 80 characters. **(12)**

Ans:

A program that accepts two file names from command prompt and check if two files are same or not:

```
#include<stdio.h>
main(int argc, char *argv[])
{
 FILE *fp1, *fp2;
 char ch1[80],ch2[80];
 int i=1,j=1,l1=0,l2=0;
 if (argc!=3)
 {
        printf ("\nWrong number of arguments.");
        getch();
        exit();
 }
 fp1=fopen(argv[1],"r");
 if (fp1==NULL)
```

```
{
        printf ("\nunable to open the file %s",argv[1]);
        getch();
        exit();
}
fp2=fopen(argv[2],"r");
if (fp2==NULL)
{
        printf ("\nunable to open the file %s",argv[2]);
        getch();
        exit();
}
l1=0;
l2=0;
while (i!=0 && j!=0)
{
        i=fgets(ch1,80,fp1);l1++;
        j=fgets(ch2,80,fp2);l2++;
        if (strcmp(ch1,ch2)!=0)
        {
                printf ("\nContents of both Files are not
Equal");
                printf     ("\nLine     Number\tContents\tFile
name\n");
                printf ("%d\t\t%s\t\tFrom %s",l1,ch1,argv[1]);
                printf ("%d\t\t%s\t\tFrom %s",l2,ch2,argv[2]);
                exit();
        }
}
if (i==0 && j==0)
        printf ("\nBoth Files are equal");
else
        printf("\nBoth files are not equal");
getch();
}
```

Q.33 Differentiate between the following:

(i) call by value and call by reference

(ii) do..while and while loops **(4)**

Ans:

(i) **Call by value and Call by reference**

Call by value means sending the values of the arguments- The value of each of the actual arguments in the calling function is copied into corresponding formal arguments of the called function. The changes made to the formal arguments have no effect on the values of actual arguments in the calling function. This technique of passing arguments is called call by value illustrated by swapv(int x, int y) function in the following example.

Call by reference means sending the addresses of the arguments- the addresses of actual arguments in the calling function are copied into formal arguments of the called function. Using these addresses we are actually working on actual argument so changes will be reflected in the calling function. This technique of passing arguments is called call by reference, illustrated by swapr(int *x,int *y) in following example.

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main(){
```

```
int i=10,j=20;
clrscr();
printf("The values before swap is i: %d, j:%d\n",i,j);
swapv(i,j);
printf("The values after swap is i: %d, j:%d\n",i,j);
printf("\n");
swapr(&i,&j);
printf("The values after swap is i: %d, j:%d\n",i,j);
printf("\n");
getch();
}
swapv(int x,int y)
{ int temp;
   temp=x;
   x=y;
   y=temp;
  }
swapr(int *x,int *y)
{
  int temp;
   temp=*x;
   *x=*y;
   *y=temp;
  }
```

The value of i and j is 10 and 20 only after calling function swapv, that is call by value. However the result of calling swapr(), call by reference is i=20 and j=10

(ii) do-while and while loops

while statement: The basic format of while statement is

```
while (conditional expression)
{
        ...block of statements to execute...
}
```

The while loop continues to loop until the conditional expression becomes false. Once this expression become false, the control is transferred out of the loop. On exit, the program continues with the statement immediately after the body of the loop. For example:

```
i=0;
        while(i<10)
       {
              printf("Hello world\n");
       i++;
}
```

This statement will print "Hello world" 10 times in a new line and come out of the loop when 'i' become 10.

Do statement: This loop construct is of the form:

```
      do
        {    ...block of statements to execute...
      }while(conditional expression);
```

While construct checks the conditional expression before the loop is executed. Sometimes it is necessary to execute the body of the loop before the conditional expression is evaluated. Such situations are handled by do-while loop construct. On reaching the do statement, the body of the loop is evaluated and at the end of the loop, the conditional expression is checked for true or false. If true,it continues to evaluate the body again and when condition become false, the control is transferred to the statement immediately after the while statement.

For example:

```
do
```

```
    {
        printf( "Input a character\n");
ch=getch( );
  }while(ch='n');
```

This segment of program reads a character from the keyboard until 'n' is keyed in.

Q.34 Write a program to generate a triangle as shown below for n = 4. The program should take the input from the user.

```
          A
        A B A
      A B C B A
    A B C D C B A
```

(8)

Ans:

A C program to generate a triangle for n=4 is listed below:

```
#include<conio.h>
void main()
{
  int i,j,k,ch;
  clrscr();
  for(i=3;i>=0;i--)
  {
        ch=65;
        printf("\n\n");

        for(j=i;j>0;j--)
              printf("  ");
        for(k=i;k<4;k++)
              printf("%c ",ch++);
        ch-=2;
        for(j=i;j<3;j++)
              printf("%c ",ch--);
        for(k=i;k>0;k--)
              printf("  ");
  }
  getch();
}
```

Q.35 Write a function that accepts two strings str1 and str2 as arguments and finds which of the two is alphabetically greater (without using the library functions). The function should return 1 if str1 is greater than str2, 0 if str1 is equal to str2, and -1 is str1 is smaller than str2. **(8)**

Ans:

A function that accept two strings to check which one is alphabetically greater:

```
void main()
{
   char a[10],b[10];
   int k;
   clrscr();
   printf("enter 1st string");
   gets(a);
   printf("enter 2nd string");
   gets(b);
   k=comp(a,b);
   if(k==1)
     printf("%s is greater than %s",a,b);
   if(k==-1)
```

```
            printf("%s is less than %s",a,b);
         if(k==0)
            printf("%s is equal to %s",a,b);
         getch();
      }
      int comp(char *a,char *b)
      {
         int i=0,j=0,k;
         while(a[i]!='\0'&& b[j]!='\0)
         {  if (a[i]<b[j])
              return (-1);
            else if(a[i]>b[j])
               return (1);
            else
               i++;
               j++;
         }
         if(i==j)
          return (0);
         if (i<j)
           return (-1);
          if(i>j)
            return (1);
       }
```

Q.36 Consider a linked list to store a polynomial, that is, every node of the linked list has coefficient, exponent and pointer to the next node in the list.

(i) Define a structure for node of such a list.

(ii)Write a function to subtract two such polynomials. The function should accept pointers to the two polynomials as arguments and return the pointer to the resultant polynomial. Assume that the polynomials passed to the function are in decreasing order on the exponents. **(2 + 8)**

Ans:

(i) Structure for polynomial node

```
struct polynode
{
  float coeff ;
  int exp ;
  struct polynode *link ;
};
```

(ii) A function that subtract two polynomials:

```
struct polynode * poly_sub ( struct polynode *x, struct polynode
*y )
{
struct polynode *z,*temp,*s=NULL;
/* if both linked lists are empty */
if ( x == NULL && y == NULL )
       return ;
/* traverse till one of the list ends */
while ( x != NULL || y != NULL )
{
       /* create a new node if the list is empty */
       if ( s == NULL )
       {
              s = malloc ( sizeof ( struct polynode ) ) ;
              z = s ;
       }
       /* create new nodes at intermediate stages */
```

```
        else
        {
        z -> link = malloc ( sizeof ( struct polynode ) )  ;
               z = z -> link ;
        }
        if (y==NULL && x!=NULL)
        {
               z -> coeff = x -> coeff;
               z -> exp = x -> exp;
               z -> link = NULL;
               x=x -> link;
               continue;
        }
        if (x==NULL && y!=NULL)
        {
               z -> coeff = y -> coeff;
               z -> exp = y -> exp;
               z -> link = NULL;
               y = y -> link;
               continue;
        }
        /* store a term of the larger degree polynomial */
        if ( x -> exp < y -> exp )
        {
               z -> coeff = y -> coeff ;
               z -> exp = y -> exp ;
               z -> link = NULL;
               y = y -> link ;  /* go to the next node */
        }
        else
        {
               if ( x -> exp > y -> exp )
               {
                      z -> coeff = x -> coeff ;
                      z -> exp = x -> exp ;
                      z -> link = NULL;
                      x = x -> link ;  /* go to the next node */
               }
               else
               {
        /* add the coefficients, when exponents are equal */
                      if ( x -> exp == y -> exp )
                      {
                             /* assigning the added coefficient */
                      z -> coeff = x -> coeff - y -> coeff ;
                             z -> exp = x -> exp ;
                             z -> link = NULL;
                             /* go to the next node */
                             x = x -> link ;
                             y = y -> link ;
                      }
               }
        }
}
return (s);
}
```

Q.37 Differentiate between the following:

(i) compiler and interpreter
(ii) unit testing and integration testing
(iii) syntax errors and logical errors **(6)**

Ans:

(i) Compiler and Interpreter: These are two types of language translators.

A **compiler** converts the source program (user-written program) into an object code (machine language by checking the entire program before execution. If the program is error free, object program is created and loaded into memory for execution. A compiler produces an error list of the program in one go and all have to be taken care even before the execution of first statement begin. It takes less time for execution.

An **interpreter** is also a language translator that translates and executes statements in the program one by one. It work on one statement at a time and if error free, executes the instruction before going to second instruction. Debugging is simpler in interpreter as it is done in stages. An interpreter takes more time for execution of a program as compared to a compiler.

(ii) Unit testing and integration testing:

Unit testing focuses on the smallest element of software design viz. the module. Unit test is conducted on each of the modules to uncover errors within the boundary of the module. It becomes simple when a module is designed to perform only one function. Unit testing makes heavy use of white-box testing.

Integration testing is a systematic approach for constructing program structure while conducting tests to uncover errors associated with interfacing. There are likely to be interfacing problems, such as data mismatch between the modules. In *Incremental* integration the program is constructed and tested in small segments. We have two types of integration testing:

-Top-Down Integration testing

-Bottom-Up Integration testing

(iii) Syntax errors and logical errors:

Syntax errors also known as compilation errors are caused by violation of the grammar rules of the language. The compiler detects, isolate these errors and give terminate the source program after listing the errors.

Logical errors: These are the errors related with the logic of the program execution. These errors are not detected by the compiler and are primarily due to a poor understanding of the problem or a lack of clarity of hierarchy of operators. Such errors cause incorrect result.

Q.38 Write a function to reverse the links in a linked list such that the last node becomes the first and the first becomes the last by traversing the linked list only once. **(8)**

Ans:

A C function to reverse the linked list:

```
struct node
{
  int data ;
  struct node *link ;
} ;
void reverse ( struct node **x )
{
  struct node *q, *r, *s ;
  q = *x ;
  r = NULL ;
  /* traverse the entire linked list */
  while ( q != NULL )
  {
        s = r ;
        r = q ;
```

```
            q = q -> link ;
            r -> link = s ;
        }
        *x = r ;
    }
```

Q.39 Define a structure for an employee of an organization having employee code, name, address, phone number and number of dependents. Assume that "allEmployees" is an array of employees in ascending order on the employee code. Write a function to display the details of an employee given its employee code. **(8)**

Ans:

A structure for an employee and a function to display the details of an employee:

```
    struct employee
    {
      int emp_code;
      char emp_name[30];
      char emp_address[50];
      char emp_ph_num[10];
      int no_of_dep;
    }allEmployees[100],b;
void display()
{
int ctr=0;
fp=fopen("employee.c","r");
rewind(fp);
while(fread(&allEmployees,sizeof(allEmployees[i]),1,fp)==1)
{
       ctr++;
       clrscr();
       heading();
       printf("\n\n\n\tFollowing are the details :-");
       printf("\n\n\tRecord #%d",ctr);
       printf("\n\n\t\tCode : %d",allEmployees[i].emp_code);
       printf("\n\n\t\tName : %s",allEmployees[i].emp_name);
printf("\n\n\t\tAddress : %s",allEmployees[i].emp_address);
printf("\n\n\t\tPhoneNumber:%s",allEmployees[i].emp_ph_num);
 printf("\n\n\t\tNumber of Dependents
                    :%s",allEmployees[i].no_of_dep);
       printf("\n\n\n\n\t\tPlease Press Enter...");
       getch();
}
}
```

Q.40 Explain the concept of top-down design for a program. **(5)**

Ans:

Top down Design:

A **top-down** approach is essentially breaking down a system to gain insight into its compositional sub-systems. In a top-down approach an overview of the system is first formulated, specifying but not detailing any first-level subsystems. Each subsystem is then refined in greater detail, sometimes in many additional subsystem levels, until the entire specification is reduced to base elements. In short the top down approach means decomposing of the solution procedure into subtasks. This approach produces a readable and modular code that can be easily understood and maintained. Top-down programming is a programming style in which design begins by specifying complex pieces and then dividing them into successively

smaller pieces. Eventually, the components are specific enough to be coded and the program is written. The technique for writing a program using top-down methods is to write a main procedure that names all the major functions it will need. Later, the programming team looks at the requirements of each of those functions and the process is repeated. These compartmentalized sub-routines eventually will perform actions so simple they can be easily and concisely coded. When all the various sub-routines have been coded the program is done.

Q.41 What are compilers and interpreters? List the advantages of an interpreter over a compiler. Under which situations you will prefer to use interpreter over compiler. **(8)**

Ans:

Compiler and Interpreter: These are two types of language translators.

A compiler converts the source program (user-written program) into an object code (machine language by checking the entire program before execution. If the program is error free, object program is created and loaded into memory for execution. A compiler produces an error list of the program in one go and all have to be taken care even before the execution of first statement begin. It takes less time for execution. An interpreter is also a language translator that translates and executes statements in the program one by one. It work on one statement at a time and if error free, executes the instruction before going to second instruction. Debugging is simpler in interpreter as it is done in stages. An interpreter takes more time for execution of a program as compared to a compiler.

Q.42 Give any two characteristics of a good programming language. **(3)**

Ans:

Two characteristics of a good programming language are:

1. It should support a large number of data types, built-in functions and powerful operators. In simple terms, a programming language should be robust, only then it can efficient and fast.
2. It should be portable means that programs written for one computer can be run on another with little or no modification.

Q.43 Write a program in C to demonstrate the use of scope resolution operator. **(8)**

Ans:

The scope resolutions operator is used to find the value of a variable out of the scope of the variable.

Example:

```
int i=10;
 main(){
 int i =5;
 cout<<::i;
cout<<i;
}
```

::i refers to the value just before the scope (i.e.10).

Q.44 Why '&' operator is not used with array names in a scanf statement? **(2)**

Ans:

In scanf() statement, the "address of" operator (&) either on the element of the array (e.g marks[i]) or on the variables (e.g &rate) are used. In doing so, the address of this particular array element is passed to the scanf() function, rather then its value; which is what scanf() requires. BUT many times the address of zeroth element (also called as base address) can be passed by just passing the name of the array.

Q.45 Write a C program to read a set of numbers from the keyboard and to sort to given array of elements in ascending order using a function. **(7)**

Ans:

A C program to sort given array of elements in ascending order:

```
#include<stdio.h>
#include<conio.h>
void sort(int*,int);
void main()
{
    int a[10];
   int i,n;
   clrscr();
   printf("how many numbers u want to enter in 1st array :
");
   scanf("%d",&n);
   printf("enter numbers :\n");
   for(i=0;i<n;i++)
   scanf("%d",&a[i]);
   sort(a,n);
   for(i=0;i<n;i++)
   printf("%d",a[i]);
   getch();
}
void sort(int *a,int m)
{
    int i,j,temp;
    for(i=0;i<m-1;i++)
   {
       for(j=i+1;j<m;j++)
      {
           if(a[i]>a[j])
          {
              temp=a[j];
              a[j]=a[i];
              a[i]=temp;
           }
       }
    }
}
```

Q.46 Write a recursive function to calculate the factorial of a positive integer. **(6)**

Ans:

A recursive function to calculate the factorial of a given positive number:

```
void main()
{
  int n,c;
```

```
clrscr();
printf("enter the number   :");
scanf("%d",&n);
c=fact(n);
printf("factorial of %d = %d",n,c);
getch();
    }
    fact(int n)
   {
int factorial;
   if(n==1||n==0)
       return(1);
   else
       factorial=n*fact(n-1);
 return (factorial);
}
```

Q.47 How does an enumstatement differ from a typedef statement? **(3)**

Ans:

Typedef statement allows user to define an identifier that would represent an exiting data type. The user-defined data type identifier can be used further to declare variables. It has the following syntax

```
typedef   datatype identifier;
```

where datatype refers to exiting data type and identifier is the new name given to this datatype. For example

```
typedef int nos;
```

nos here symbolizes int type and now it can be used later to declare variables like

```
nos num1,num2,num3;
```

enum statement is used to declare variables that can have one of the values enclosed within braces known as enumeration constant. After declaring this, we can declare variables to be of this 'new' type. It has the following syntax

```
enum identifier {value1, value2, value3……,valuen);
```

where value1, value2 etc are values of the identifier. For example

```
enum day { Monday, Tuesday, Wednesday, Thursday, Friday,
Saturday}
```

We can define later

```
enum day working_day;
working_day=Monday;
```

The compiler automatically assigns integer digits beginning with 0 to all enumeration constants.

Q.48 Define a structure. How it is different from union? Write a C program to illustrate a structure. **(8)**

Ans:

Structures and union

A structure contains an ordered group of data objects. Unlike the elements of an array, the data objects within a structure can have varied data types. Each data object in a structure is a member or field. A union is an object similar to a structure except that

1. In union, one block is used by all the member of the union but in case of structure, each member has their own memory space. All of union members start at the same location in memory. A union variable can represent the value of only one of its members at a time.

2. The size of the union is equal to the size of the largest member of the union where as size of the structure is the sum of the size of all members of the structure.

```
For example
        struct book
       {
         char    name;
          int  pages;
          float price;
        };
```

Now if we define struct book book1, then it will assign 1+2+4=7 bytes of memory for book1.

If we define it as union like

```
union book
      {
         char name;
          int  pages;
          float price;
        };
```

The compiler allocates a piece of storage that is large enough to store the largest variable types in union. All three variables will share the same address and 4 bytes of memory is allocated to it. This program of structure will read name, roll no and marks in 6 subject of 3 students & then display name and roll of student who scored more than 70% marks in total.

```
#include<stdio.h>
#include<conio.h>
struct student
{
  char name[10];
  int roll_no;
  int marks[6];
  int total;
  int per;
};
void main()
{
   struct student stu[3];
   int i,j,req;
   clrscr();
   for(i=0;i<3;i++)
   {
     stu[i].total=0;
     printf("enter data for %d students :",i+1);
     printf("\nenter name");
     scanf("%s",stu[i].name);
     printf("\nenter roll no   ");
     scanf("%d",&stu[i].roll_no);
     printf("\nenter marks in subjects\t");
     for(j=0;j<6;j++)
     {
         printf("\nenter marks in %d subject\t",j+1);
         scanf("%d",&stu[i].marks[j]);
         stu[i].total=stu[i].total+stu[i].marks[j];
     }
     stu[i].per=stu[i].total/6;
     printf("\n");
   }
   for(i=0;i<3;i++)
   {
      if(stu[i].per>70)
      {
          printf("\nSTUDENT    %d",i+1);
```

```
                printf("\nname  :");
                printf("%s",stu[i].name);
                printf("\nroll no");
                printf("%d",stu[i].roll_no);
                for(j=0;j<6;j++)
               {
                   printf("\nmarks in %d subject\t",j+1);
                   printf("%d",stu[i].marks[j]);
                }
           printf("\nTOTAL    :%d",stu[i].total);
       }
       }
       getch();
   }
```

Q.49 Write a C program to concatenate two strings. **(8)**

Ans:

A program to concatenate two strings is given below:

```
#include<stdio.h>
#include<conio.h>
void main()
{
char a[100],b[100];
int i=0,j=0;
clrscr();
printf("enter the set of lines");
gets(a);
while(a[i]!='\0')
{
while(a[i]!=' '&&a[i]!='\t'&&a[i]!='\0')
{
b[j]=a[i];
j++;
i++;
}
while(a[i]==' '||a[i]=='\t')
i++;
}
b[j]='\0';
printf("%s",b);
getch();
}
```

Q.50 What is a pointer? How it is declared? Write a C program to reverse a string using pointers. **(2+3+4)**

Ans:

A pointer is a variable which contains the address in memory of another variable. We can have a pointer to any variable type. The unary or monadic operator **&** gives the "address of a variable". The indirection or dereference operator ***** gives the "contents of an object pointed to by a pointer".

A pointer is declared as follows:

```
int *ptr;
```

where *ptr is a pointer of int type.

A program to reverse a string using pointer is listed below:

```
#include<stdio.h>
#include<conio.h>
void main()
{
```

```
 int i,j;
 char *a;
 clrscr();
 printf("enter a string");
 scanf("%s",a);
 i=0;
 while(*(a+i)!='\0')
 i++;
 for(j=i-1;j>=0;j--)
 printf("%c",*(a+j));
 getch();
}
```

Q.51 Differentiate between pointers and arrays? Write a C program to display the contents of an array using a pointer arithmetic. **(2+5)**

Ans:

Pointer and arrays:

Pointers and arrays are very closely linked in C. Consider the following statements:

```
int a[10], x;
int *ptr; /* ptr is a pointer variable*/
ptr = &a[0];  /* ptr points to address of a[0] */
x = *ptr;
/* x = contents of ptr (a[0] in this case) */
```

A pointer is a variable so we can do

`ptr = a` and `ptr++;`

while an array is not a variable so statements

`a = pa` and `a++` are illegal.

A C program to display the contents of an array using a pointer arithmetic is listed below:

```
//display the contents of an array using pointer
#include<conio.h>
void main()
{
 int *p,sum,i;
 static int x[5] = {5,9,6,3,7};
 i=0;
 p=x;
 sum=0;
 clrscr();
 printf("\nElement   Value   Address\n\n");
 while(i<5)
 {
       printf("  x[%d]     %d       %u\n",i,*p,p);
       sum+=*p;
       i++;
       *p++;
 }
 printf("\n Sum   = %d\n",sum);
 printf("\n &x[0] = %u\n",&x[0]);
 printf("\n p     = %u\n",p);
 getch();
}
```

Q.52 What is a linked list? List different types of linked list. Write a C program to demonstrate a simple linear linked list. **(2+2+6)**

Ans:

Linked list: A linked list is a self referential structure which contain a member field that point to the same structure type. In simple term, a linked list is collections of nodes that consists of two fields, one containing the information about that node, item and second contain the address of next node. Such a structure is represented as follows:

```
struct node
{       int item;
         struct node *next;
};
```

Info part can be any of the data type; address part should always be the structure type.

Linked lists are of following types:

1. **Linear Singly linked list**: A list type in which each node points to the next node and the last node points to NULL.
2. **Circular linked list**: Lists which have no beginning and no end. The last node points back to the first item.
3. **Doubly linked list or Two-way linked list**: These lists contain double set of pointers, one pointing to the next item and other pointing to the preceding item. So one can traverse the list in either direction.
4. **Circularly doubly linked list**: It employs both the forward and backward pointer in circular form.

A program to demonstrate simple linear linked list is given below:

```
#include<stdio.h>
#include<stdlib.h>
#define NULL 0
struct linked_list
{
  int number;
  struct linked_list *next;
};
typedef struct linked_list node;
void main()
{
  node *head;
  void create(node *p);
  int count(node *p);
  void print(node *p);
  clrscr();
  head=(node *)malloc(sizeof(node));
  create(head);
  printf("\n");
  print(head);
  printf("\n");
  printf("\n Number of items  = %d \n",count(head));
  getch();
}
void create(node *list)
{
  printf("Input a number\n");
  printf("(type -999 to end) : ");
  scanf("%d",&list->number);
  if(list->number == -999)
         list->next=NULL;
  else
  {
         list->next=(node *)malloc(sizeof(node));
         create(list->next);
```

```
  }
  return;
}
void print(node *list)
{
  if(list->next!=NULL)
  {
        printf("%d - ->",list->number);
  if(list->next->next == NULL)
        printf("%d",list->next->number);
  print(list->next);
  }
  return;
}
int count(node *list)
{
  if(list->next == NULL)
        return(0);
  else
        return(1+count(list->next));
}
```

Q.53 Explain the different types of memory allocations in C. **(6)**

Ans:

Different types of memory allocation function are:

(i) malloc( ): It is a memory allocation function that allocates requested size of bytes and returns a pointer to the first byte of the allocated space. The malloc function returns a pointer of type void so we can assign it to any type of pointer. It takes the the following form:

```
ptr= (cast type *) malloc(byte-size);
```

where ptr is a pointer of type cast-type. For example, the statement

`x=(int *) malloc(10 *sizeof(int))` means that a memory space equivalent to 10 times the size of an int byte is reserved and the address of the first byte of memory allocated is assigned to the pointer x of int type.

The malloc function can also allocate space for complex data types such as structures. For example:

`ptr= (struct student*) malloc(sizeof (struct student));`

where ptr is a pointer of type struct student.

(ii) calloc( ): It is another memory allocation function that allocates space for an array of elements, initializes them to zero and then returns a pointer to the memory. This function is normally used for requesting memory space at run time. It takes the following form:

```
ptr= (cast type *) calloc(n,element-size);
```

This statement allocates contiguous space for n blocks, each of size element-size bytes.

(iii) realloc( ): realloc is a memory allocation function that modifies the size of previously allocated space. Sometime it may happen that the allocated memory space is larger than what is required or it is less than what is required. In both cases, we can change the memory size already allocated with the help of the realloc function known as reallocation of memory. For example, if the original allocation is done by statement

```
ptr= malloc(size);
```

then reallocation is done by the statement

`ptr=realloc(ptr,newsize);` which will allocate a new memory space of size newsize to the pointer variable ptr and returns a pointer to the first byte of the new memory block.

Q.54 Explain the following file functions.

(i) fgetc( ) (ii) ftell
(iii) fgets( ) (iv) rewind( )
(v) fseek **(5)**

Ans:

(i)getc( ): reads a character from a file. This function is used to read a character from a file that has been opened in the read mode. For example, the statement c=getc(fp2); would read a character from the file whose file pointer is fp2.

(ii)ftell( ): gives the current position in the file from the start. ftell takes a file pointer and returns a number of type long, that corresponds to the current position. For example:n=ftell(p); would give the relative offset n in bytes of the current position. This means that n bytes have already been read (or written).

(iii)fgets( ): To read a line of characters from a file,we use the fgets() library function. The prototype is `char *fgets(char *str, int n, FILE *fp);` The argument str is a pointer to a buffer in which the input is to be stored, n is the maximum number of characters to be input, and fp is the pointer to type FILE that was returned by fopen() when the file was opened.

(iv) rewind( ):sets the position to the beginning of the file. It also takes a file pointer and reset the position to the start of the file. For example:

```
rewind(fp);
```

`n=ftell(fp);` would assign 0 to n because file pointer has been set to the start of the file by rewind.

(v)fseek( ):sets the position to a desired point in the file. fseek function is used to move the file position to a desired location within the file. For example: `fseek(fp,m,0);` would move the file pointer to (m+1)th byte in the file.

Q.55 Write a program that reads the following information from the keyboard – student_id, student name and total marks and writer to a file. After taking the information it closes the file and displays the information about the student whose student_id has been entered by the user. **(9)**

Ans:

```
//display records on the basis of student id entered.
#include<stdio.h>
struct student
{
  int stu_id;
  char stu_name[30];
  int marks;
}s[100],b;
void menu();
void create();
void display();
void search();
void heading();
void disp();
int i=0;
FILE *fp,*ft;
void main()
{
```

```
 int first_ch;
 clrscr();
 heading();
 printf("\n\n\n\t\t\t\t    MENU");
 printf("\n\n\n\t\t1.Create a new database.");
 printf("\n\n\t\t2.Continue with the existing database.");
 printf("\n\n\t\t3. Exit.");
 printf("\n\n\n\t\tEnter your choice( 1 - 3): ");
 scanf("%d",&first_ch);
 switch(first_ch)
 {
        case 1:
               create();
               break;
        case 2:
               menu();
                        break;
                      case 3:
              exit();
              break;
        default:
printf("\n\n\n\t\tYou  have  entered  wrong  choice...!!!..The
                     program will exit now.");
 }
 getch();
}
void create()
{
 int ctr=0;
 char ch = 'y';
 fp=fopen("student.c","w");
 while(ch == 'y')
 {
        ctr++;
        clrscr();
        heading();
 printf("\n\n\n\n\tEnter the Employee Records  :\n ");
        printf("\n\n\t\t  Student id : ");
        fflush(stdin);
        scanf("%d",&s[i].stu_id);
        printf("\n\n\t\t        Name : ");
        fflush(stdin);
        scanf("%s",&s[i].stu_name);
        printf("\n\n\t\t Total Marks : ");
        fflush(stdin);
        scanf("%d",&s[i].marks);
        fwrite(&s,sizeof(s[i]),1,fp);
        printf("\n\n\n\t\tDo  you  wish  to  enter  more  records
(y/n) ?");
        ch=getch();
 }
 printf("\n\n\n\t%d   Records   have   been   written   in   the
file.",ctr);
 fclose(fp);
 menu();
}
void menu()
{
        char ch;
        clrscr();
        heading();
        printf("\n\n\n\t\t\t    MENU :-");
        printf("\n\n\t\t1.To DISPLAY all the records");
```

```
        printf("\n\n\t\t2.To SEARCH a record");
        printf("\n\n\t\t3.EXIT");
        printf("\n\n\n\t\tEnter your choice (1 - 3) : ");
        scanf("%d",&ch);
        switch(ch)
        {
               case 1:
                      display();
                      break;
               case 2:
                      search();
                      break;
               case 3:
                      exit();
                      break;
               default:
                       printf("\n\n\tYou have entered a WRONG
                          CHOICE..!!..Please   Re-enter   your
                          choice");
                      menu();
        }
}
void display()
{
 int ctr=0;
 fp=fopen("student.c","r");
 rewind(fp);
 while(fread(&s,sizeof(s[i]),1,fp)==1)
 {
        ctr++;
        clrscr();
        heading();
        printf("\n\n\n\tFollowing are the details :-");
        printf("\n\n\tRecord #%d",ctr);
        printf("\n\n\t\tStudent id : %d",s[i].stu_id);
        printf("\n\n\t\t      Name : %s",s[i].stu_name);
        printf("\n\n\t\tTotal Marks: %d",s[i].marks);
        printf("\n\n\n\n\t\tPlease Press Enter...");
        getch();
 }
 menu();
}
void search()
{
 int f,flag=0;
 char ch='y';
 while(ch=='y')
 {
        clrscr();
        flag=0;
        heading();
 printf("\n\n\n\tEnter the Student id to be searched : ");
        fflush(stdin);
        scanf("%d",&b.stu_id);
        fp=fopen("student.c","r");
        rewind(fp);
        while(fread(&s,sizeof(s[i]),1,fp)==1)
        {
        if(s[i].stu_id == b.stu_id)
               {
                      clrscr();
                      flag=1;
                      heading();
```

```
                                printf("\n\n\n\tThe details of the record
                                       having  Student  id  %d  are  :-
                                       ",b.stu_id);
                                disp();
                              }
                        }
                        fcloseall();
                        if(flag==0)
                        printf("\n\n\n\t\tNo Match found !!");
                printf("\n\n\n\tDo u wish to search for more records  (y/n) ?
                ");
                        ch=getch();
                  }
                menu();
                }
                void heading()
                {
                  printf("\n\n\t\t\tSTUDENT DATABASE MANAGEMENT");
                  printf("\n\n\t\t\t----------------------------");
        }
```

Q.56 Define macros. **(2)**

Ans:

A **macro** is a pre-processor directive which is a program that processes the source code before it passes through the compiler. These are placed in the source program before the main. To define a macro, # define statement is used. This statement, also known as macro definition takes the following general form:

#define identifier string

The pre-processor replaces every occurrence of the identifier in the source code by the string. The preprocessor directive definition is not terminated by a semicolon. For example

#define COUNT 100 will replace all occurrences of COUNT with 100 in the whole program before compilation.

Q.57 Write a C program that reads two strings str1 and str2 and finds the no of occurrence of smaller strings in large string. **(8)**

Ans:

```
#include<conio.h>
#include<string.h>
void main()
{
  char str1[100],str2[100],a[100],b[100],c[100];
  int len1,len2,l1,l2,k=0,i,j,count=0,n;
  clrscr();
  printf("\n Enter the first string : ");
  gets(str1);
  printf("\n Enter the second string : ");
  gets(str2);
  len1=strlen(str1);
  len2=strlen(str2);
  if(len1>len2)
  {
        strcpy(a,str1);
        strcpy(b,str2);
        l1=len1;
        l2=len2;
```

```
  }
  else if(len2>len1)
  {
         strcpy(a,str2);
         strcpy(b,str1);
         l1=len2;
         l2=len1;
  }
  else
         printf("\n\n Please enter one string smaller than the
other!!!");

  for(i=0;i<l1;i++)
  {
         n=i;
         for(j=1;j<=l2;j++)
                c[k++]=a[n++];
         c[k]='\0';
         if(strcmp(b,c)==0)
                ++count;
         k=0;
  }
  printf("\n The number of occurences is : %d",count);
  getch();
}
```

Q.58 Explain path testing. **(6)**

Ans:

Path Testing: Testing in which all paths in the program source code are tested at least once. Path testing has been one of the first test methods, and even though it is a typical white box test, it is nowadays also used in black box tests. First, a certain path through the program is chosen. Possible inputs and the correct result are written down. Then the program is executed by hand, and its result is compared to the predefined. Possible faults have to be written down at once.

Q.59 Distinguish between malloc( ) and calloc( ). **(2)**

Ans:

While malloc allocates a single block of storage space, calloc allocates multiple block of storage, each of the same size, and then sets all bytes to zero. (Explained in Q53)

Q.60 What is a compiler? What type of errors can be detected by it? How does it differ from an interpreter? **(2+3+3)**

Ans:

A **Compiler** is a program that accepts a program written in a high level language and produces an object program. The types of errors detected by a compiler are:

- **Syntax errors** -- The compiler cannot understand a program because it does not follow the **syntax** that is the rules and grammar of a particular language. Common syntax errors include
    - missing or misplaced ; or },
    - missing return type for a procedure,
    - Missing or duplicate variable declaration.
    - Type errors that include

- type mismatch on assignment,
- type mismatch between actual and formal parameters.

An **interpreter** is a program that appears to execute a source program as if it were machine language.

Q.61 What is a subroutine? How do subroutines help in program writing? **(2+2)**

Ans:
A subroutine is a named, independent section of C code that performs a specific task and optionally returns a value to the calling program. Some of the important characteristics of subroutine that help in program writing are:

- A subroutine is named, each have a unique name. By using that name in another part of the program, one can execute the statements contained in the subroutine.
- A subroutine is independent that can perform its task without interference from or interfering with other parts of the program.
- A subroutine performs a specific task. A task is a discrete job that a program must perform as part of its overall operation, such as sending a line of text to a printer, sorting an array into numerical order, or calculating a cube root.
- A subroutine can return a value to the calling program. When a program calls a subroutine, then statements it contains are executed. These statements can pass information back to the calling program.

Q.62 Define the term 'complexity of an algorithm; and explain worst-case and average case analysis of an algorithm. **(2+4)**

Ans:
Complexity of an algorithm is the measure of analysis of algorithm. Analyzing an algorithm means predicting the resources that the algorithm requires such as memory, communication bandwidth, logic gates and time. Most often it is computational time that is measured for finding a more suitable algorithm. This is known as time complexity of the algorithm. The running time of a program is described as a function of the size of its input. On a particular input, it is traditionally measured as the number of primitive operations or steps executed. Worst case analysis of an algorithm is an upper bound on the running time for any input. Knowing that in advance gives us a guarantee that the algorithm will never take any longer. Average case analysis of an algorithm means expected running time of an algorithm on average input data set. The average case is often as bad as the worst case. One problem with performing an average case analysis is that it is difficult to decide what constitutes an "average" input for a particular problem. Often we assume that all inputs of a given size are equally likely.

Q.63 Write a C program to generate a series of Armstrong numbers. A number is an Armstrong number if the sum of the cube of its digit is equal to the number. **(6)**

Ans:
A C program to generate a series of Armstrong numbers is listed below:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
   int n,r,k,sum=0,up;
```

```
    clrscr();
    printf("enter the limit of series ");
    scanf("%d",&n);
    up=n;
    while (up>1)
    {
         k=n;
         sum=0;
         while(n>0)
         {
             r=n%10;
             sum=sum+pow(r,3);
             n=n/10;
          }
         if(sum==k)
         printf("%d\t",k);
         up--;
         n=up;
    }
     getch();
}
```

Q.64 Write a C program to generate the following series:

$$\text{sum} = x + \frac{x^2}{2!} + \frac{x^4}{4!} - - - \frac{x^n}{x!}$$ **(8)**

Ans:

A C program to generate an exponential series is given below:

```
main()
{
  int n,i=0,j=1;
  long int fact=1;
  clrscr();
  printf ("\nEnter the limit for the series:-> ");
  scanf("%d",&n);
  if (n%2!=0)
          n-=1;
    printf("X +");
  for (i=1;i<=n;)
  {
         fact=1;j=1;
         while (j<=i)
         {
                fact*=j;
                j++;
         }
         printf("\tX^%d/%ld\t+",i,fact);
         i*=2;
  }
  getch();
}
```

Q.65 What is bubble sort? Write a program to sort a list of n elements of an array using bubble sort. **(8)**

Ans:

Bubble Sort:

The basic idea in bubble sort is to scan the array to be sorted sequentially several times. Each pass puts the largest element in its correct position by comparing each element in the array with its successor and interchanging the two elements if they are not in proper

order. The first pass put the largest element at (n-1)th position in an array of n elements, next pass put second largest element at (n-2)th position and so on. This way each element slowly bubbles up to its proper position that is why it is called bubble sort. A program sorting array element using bubble sort is given below:

```
#include<stdio.h>
       #include<conio.h>
       void main()
       {
 int a[20],i,j,n,c,flag;
 clrscr();
 printf("how many numbers u want to enter :\n");
 scanf("%d",&n);
 printf("\nenter the numbers :\n");
 for(i=0;i<n;i++)
 scanf("%d",&a[i]);
 for(i=0;i<n-1;i++)
 {
  for(j=0;j<n-1-i;j++)
  {
      if(a[j]>a[j+1])
     {
         c=a[j];
         a[j]=a[j+1];
         a[j+1]=c;
         flag=0;
      }
  }
 if(flag)
  break;
 else
  flag=1;
 }
 printf("sorted elements :\n");
 for(i=0;i<n;i++)
 printf("%d\t",a[i]);
 printf("\n");
 getch();
}
```

Q.66 Explain the difference between a function declaration and function definition. **(4)**

Ans:

Function declaration and Function definition:

A function declaration contains the name of the function, a list of variables that must be passed to it, and the type of variable it returns, if any. For example in the following program in line 2, the function cube is declared. The variables to be passed to the function are called arguments, and they are enclosed in parentheses following the function's name. In this example, the function's argument is long x. The keyword before the name of the function indicates the type of variable the function returns. In this case, a type long variable is returned. The function itself is called the function definition. In the following example, a function cube is defined from line 13 to 18.A function definition has following several parts:

- **Function header**: The function starts out with a function header on line 13. The function header is at the start of a function, and it gives the function's name, the function's return type and describes its arguments. The function header is identical to the function declaration minus the semicolon.

- **Function Body**: The body of the function, lines 14 through 18, is enclosed in braces. The body contains statements that are executed whenever the function is called. Local variables are declared within a function body. Finally, the function concludes with a return statement on line 17, which signals the end of the function and it passes a value back to the calling program.

The following program uses a function to calculate the cube of a number.

```
1: #include <stdio.h>
2: long cube(long x);
3: long a, result;
4: main()
5: {
6: printf("Enter an integer value: ");
7: scanf("%d", &a);
8: result= cube(a);
9: printf("\nThe cube of %ld is %ld.\n", a, result);
10: }
11:
12: /* Function: cube() - Calculates the cube value of a
variable */
13: long cube(long x)
14: {
15: long y;
16: y = x * x * x;
17: return y;
18: }
```

Q.67 What is a pre-processor? Which pre-processor is used to define a macro? **(4)**

Ans:

A pre-processor is a program that processes the source code before it passes through the compiler. It operates under the control of preprocessor directive. These are placed in the source program before the main.

To define a macro, # define statement is used. This statement, also known as macro definition takes the following general form:

```
#define identifier string
```

The pre-processor replaces every occurrence of the identifier in the source code by the string. The preprocessor directive definition is not terminated by a semicolon. For example

`#define COUNT 100` will replace all occurrences of COUNT with 100 in the whole program before compilation.

Q.68 What are the different storage classes in C? **(4)**

Ans:

Storage classes in C

There are four storage classes in C:

a. **Automatic storage class**: The features of variables are as follows

-Storage: Memory
-Default initial value: Garbage value
-Scope: Local to the block in which defined
-Life: till the control remains within the block in which defined.

b. **Register storage class**: The features of variables are as follows

-Storage: CPU registers

-Default initial value: Garbage value
-Scope: Local to the block in which defined
-Life: till the control remains within the block in which defined

c. **Static storage class:** The features of variables are as follows
-Storage: Memory
-Default initial value: Zero
-Scope: Local to the block in which defined
-Life: value of variable persists between different function calls.

d. **External storage class**: The features of variables here are as follows
-Storage: Memory
-Default initial value: Zero
-Scope: global
-Life: As long as program execution does not come to an end.

Q.69 Differentiate between recursion and iteration. **(4)**

Ans:

Recursion and iteration:

The factorial of a number, written as n!, can be calculated as follows:
n! = n * (n-1) * (n-2) * (n-3) * ... * (2) * 1
This approach is called iteration, that is step by step calculation.
However, we can also calculate n! like:
n! = n * (n-1)!
Going one step further, we can calculate (n-1)! using the same procedure:
(n-1)! = (n-1) * (n-2)!
We can continue calculating recursively until the value of n is 1. This approach is called recursion. It refers to a situation in which a function calls itself either directly or indirectly. Indirect recursion occurs when one function calls another function that then calls the first function.

Q.70 Differentiate between pointer (*) and address (&) operator using examples. **(4)**

Ans:

The indirection operator (*) gets the value stored in the memory location whose address is stored in a pointer variable. The address of (&) operator returns the address of the memory location in which the variable is stored. The output of the following example shows the difference between * and &.

```
//difference between * and &.
#include<conio.h>
void main()
{
 int k;
 int *ptr;
 clrscr();
 k=10;
 ptr=&k;
 printf("\n Value of k is %d\n\n",k);
 printf("%d is stored at addr %u\n",k,&k);
 printf("%d is stored at addr %u\n",*ptr,ptr);
 *ptr=25;
 printf("\n Now k = %d\n",k);
 getch();
}
Output: Value of k is 10
```

```
10 is stored at addr 65524
10 is stored at addr 65524
      Now k=25
```

Q.71 Write a program to find the highest of three numbers using pointer to function. **6)**

Ans:

A C program to find the highest of three numbers using pointer to function is listed below:

```
#include<conio.h>
void main()
{
 int x,y,z;
 clrscr();
 printf("\n Enter three numbers : ");
 scanf("%d %d %d",&x,&y,&z);
 printf("\n\n The highest of the three numbers is : ");
 highest(&x,&y,&z);
 getch();
}
highest(a,b,c)
int *a,*b,*c;
{
 if(*a > *b && *a > *c)
        printf("%d",*a);
 else if(*b > *a && *b > *c)
        printf("%d",*b);
 else
        printf("%d",*c);
}
```

Q.72 Differentiate between

i. Static variables and auto variables.
ii. Execution error and compilation error. **(6)**

Ans:

(i) Static variables and auto variables:

Static variables: The features are as follows

Declaration place:-may be declared internally or externally.

Declaration syntax:-we use the keyword static to declare a static variable.

```
Static int age;
```

Default initial value:- Zero

Scope:-in case of internal static variable, the scope is local to the function in which defined while scope of external static variable is to all the functions defined in the program.

Life:- value of variable persists between different function calls.

Automatic variables: The features are as follows

Declaration place:-declared inside a function in which they are to be utilized, that's why referred as local or internal variables.

Declaration syntax:- A variable declared inside a function without storage class specification by default is an automatic variable. However, we may use the keyword auto to declare it explicitly.

```
main()
{
 auto int age;
}
```

Default initial value:-Garbage value

Scope:-created when the function is called and destroyed on exit from the function.

Life:- till the control remains within the block in which defined.

(ii) Execution error and compilation error:

Errors such as mismatch of data types or array out of bound error are known as execution errors or runtime errors. These errors are generally go undetected by the compiler so programs with run-time error will run but produce erroneous results.

Compilation error also known as syntax errors are caused by violation of the grammar rules of the language. The compiler detects, isolate these errors and terminate the source program after listing the errors.

Q.73 Explain the use of structures with pointers. **(5)**

Ans:

C allows a pointer to a structure variable. In fact a pointer to a structure is similar to a pointer to any other variable. The pointer points to the first element of the structure. A structure can have members which points to a structure variable of the same type; such structures are called self-referential structures and widely used in dynamic data structures like trees, linked list etc. A program showing the usage of pointer with structure variable is listed below:

```
#include<stdio.h>
#include<conio.h>
struct student
{
    char name[20];
    int roll_no;
};
void main()
{
   struct student stu[3],*ptr;
   clrscr();
   printf("\n Enter data\n");
   for(ptr=stu;ptr<stu+3;ptr++)
   {  printf ("Name");
       scanf("%s",ptr->name);
       printf("roll_no");
       scanf("%d",&ptr->roll_no);
   }
   printf("\nStudent Data\n\n");
   ptr=stu;
   while(ptr<stu+3)
   {
     printf("%s   %5d\n",ptr->name,ptr->roll_no);
     ptr++;
    }
   getch();
}
```

Q.74 Can a structure be used within a structure? Give appropriate examples to support your answer. **(6)**

Ans:

Yes, a structure can be used within a structure called nesting of structures. For example, inside a structure rectangle, a point (x,y) needs to be represented. So we can first declare a structure point as:

```
struct point
{ int x;
   int y;
};
```

We can now use this definition in defining points in any of geometrical figure. For example

```
struct rectangle
{   struct point pt;
     int  diagonal;
}
```

Q.75 What are the different modes in which a file can be opened? **(5)**

Ans:

Different modes for opening a file are tabulated below:

Modes	Operations
"r"	Open text file for reading
"w"	Open text file for writing, previous content, if any, discarded
"a"	Open or create file for writing at the end of the file
"r+"	Open text file for update
"w+"	Create or open text file for update, previous content lost, if any
"a+"	Open or create text file for update, writing at the end.

Q.76 Write a C program to insert a new node at a specified position in a link list.**(8)**

Ans:

A C program to insert a new node at a specified position in a linked list is listed below:

```
#include<stdio.h>
#include<stdlib.h>
#define NULL 0
struct linked_list
{
  int number;
  struct linked_list *next;
};
typedef struct linked_list node;
void main()
{
  node *head;
  node *n;
  void create(node *p);
  void print(node *p);
  node *insert(node *p);
  clrscr();
  head=(node *)malloc(sizeof(node));
  create(head);
  printf("\n");
  print(head);
  printf("\n");
  n=insert(head);
  printf("\n");
```

```
 print(n);
 getch();
}
void create(node *list)
{
 printf("Input a number\n");
 printf("(type -999 to end) : ");
 scanf("%d",&list->number);
 if(list->number == -999)
        list->next=NULL;
 else
 {
        list->next=(node *)malloc(sizeof(node));
        create(list->next);
 }
 return;
}
void print(node *list)
{
 if(list->next!=NULL)
 {
        printf("%d - ->",list->number);
 if(list->next->next == NULL)
        printf("%d",list->next->number);
 print(list->next);
 }
 return;
}
node *insert(node *head)
{
 node *find(node *p,int a);
 node *new,*tmp;
 node *n1;
 int key;
 int x,i=0;
 printf("\nValue of new item ? ");
 scanf("%d",&x);
 printf("\nValue of key index ? (type -999 if last): ");
 scanf("%d",&key);
 tmp=head;
 while (1)
 {
        if (key==0)
        {
               new=(node *)malloc(sizeof(node));
               new->number=x;
               new->next=head;
               head=new;
        }
        else if(i==key-1)
        {
               new=(node *)malloc(sizeof(node));
               new->number=x;
               new->next=tmp->next;
               tmp->next=new;
        }
        i++;
        tmp=tmp->next;
        if (tmp==NULL)
        {
               printf("\nKey Index out of Reach");
               break;
        }
```

```
  }
  return(head);
}
node *find(node *list,int key)
{
  if(list->next->number==key)
        return(list);
  else
        if(list->next->next==NULL)
              return(NULL);
        else
              find(list->next,key);
}
```

Q.77 Distinguish between the functions islower() and tolower(). **(2)**

Ans:

islower( ) and tolower( ):

islower(c) is a character testing function defined in ctype.h header file. This function determines if the passed argument, in this case c, is lowercase. It returns a nonzero value if true otherwise 0. tolower (c) is a conversion function defined in ctype.h header file that convert argument c to lowercase.

Q.78 Write a C program that reads two strings and copies the smaller string into the bigger string. **(6)**

Ans:

A program to copy smaller string to a bigger string:

```
#include<stdio.h>
#include<conio.h>
void main()
{
   char str1[20],str2[20];
   int m,i,flag=0,j;
   clrscr();
   printf("enter the 1st string");
   gets(str1);
   printf("enter the 2nd string");
   gets(str2);
   if(strlen(str1)<strlen(str2))
   {
       char *str;
        str=str2;
        str2=str1;
        str1=str;
   }
   printf("enter the index after which u want to insert 2nd
   string in 1st : ");
   scanf("%d",&m);
   i=0;
   while(i<=m)
   {
       i++;
   }
   j=0;
   while(str2[j]!='\0')
   {
        str1[i]=str2[j];
        i++;
```

```
            j++;
            if(str1[i]=='\0')
            flag=1;
        }
        if(flag==1)
            str1[i]='\0';
        printf("%s",str1);
        getch();
    }
```

Q.79 Are the following statements valid? Justify your answer

(i) k = (char*)& m

(ii) m= (float*)& p **(4)**

Ans:

Type of m is not known. If it is of type other than char and k is a pointer to char type then statement is valid otherwise it is invalid. Similarly, second statement is valid if m is a pointer to float and p is of type other than float and pointing to more thant four consecutive bytes.

Q.80 Why is C standard library needed? **(4)**

Ans:

C Standard Library:

C would have been a tough programming language in absence of the standard library. The standard library in C includes header files and standard functions in these files. Header files are collection of macros, types, functions and constants. Any program that needs those functions has to include these header files. For example, the standard library functions, included in "stdlib.h" header file, are used for number conversions, storage allocation and other functions on the same line. These functions are called utility functions. C does not have any built-in input/output statements as part of its syntax. All I/O operations are carried out through function calls such as printf and scanf. There are many functions that have become standard for I/O operations in C, collectively known as standard I/O library "stdio.h". Because of this standard input-output header file, standard I/O functions can be used on many machines without any change.

Q.81 What are the functions of the following header files:

(i) ctype.h

(ii) string.h **(4)**

Ans:

(i) ctype.h: It is a header file that contains character testing and conversion functions. For example isalpha (c) returns an int type data and determine if the argument c is alphabetic. It returns nonzero value if true; 0 otherwise.

toascii(c): is a conversion function defined in ctype.h that converts value of argument to ascii.

(ii) string.h: It is also a standard header file that contains string manipulation funcions. For example strcmp(c1,c2) function compares two strings c1 and c2 lexicographically. It returns a negative value if c1<c2; 0 if c1 and c2 are identical; and a positive value if c1>c2.

Q.82 Explain pointers and structures by giving an example of pointer to structure variable? **(5)**

Ans:

We can have a pointer pointing to a structure just the same way a pointer pointing to an int, such pointers are known as structure pointers. For example consider the following example:

```
#include<stdio.h>
#include<conio.h>
struct student
{
     char name[20];
     int roll_no;
};
void main()
{
     struct student stu[3],*ptr;
     clrscr();
     printf("\n Enter data\n");
     for(ptr=stu;ptr<stu+3;ptr++)
    {  printf("Name");
        scanf("%s",ptr->name);
        printf("roll_no");
        scanf("%d",&ptr->roll_no);
    }
    printf("\nStudent Data\n\n");
    ptr=stu;
    while(ptr<stu+3)
   {
        printf("%s   %5d\n",ptr->name,ptr->roll_no);
        ptr++;
    }
getch();
}
```

Here ptr is a structure pointer not a structure variable and dot operator requires a structure variable on its left. C provides arrow operator "->" to refer to structure elements. "ptr=stu" would assign the address of the zeroth element of stu to ptr. Its members can be accessed by statement like "ptr->name". When the pointer ptr is incremented by one, it is made to point to the next record, that is stu[1] and so on.

Q.83 Explain various steps for analysing an algorithm. **(6)**

Ans:

The various steps involved in analysis of an algorithm are:

1. For any algorithm, the first step should be to prove that it always returns the desired output for all legal instances of the problem.
2. Second step to analyze an algorithm is to determine the amount of resources such as time and storage necessary to execute it. Usually the efficiency or complexity of an algorithm is stated as a function relating the input length to the number of steps (time complexity) or storage locations (space complexity). In theoretical analysis of algorithms it is common to estimate their complexity in asymptotic sense, i.e., to estimate the complexity function for reasonably large length of input. Big O notation, omega notation and theta notation are used for this purpose. There are many techniques for solving a particular problem. We must analyze these algorithms and select the one which is simple to follow, takes less execution time and produces required results.

Q.84 What are Translators? Explain its various types. **(3)**

Ans:
A program that translates between high-level languages is usually called a language translator, source to source translator, or language converter.
The various types of translator are:
A compiler converts the source program (user-written program) into an object code (machine language by checking the entire program before execution. If the program is error free, object program is created and loaded into memory for execution. A compiler produces an error list of the program in one go and all have to be taken care even before the execution of first statement begin. It takes less time for execution.
An interpreter is also a language translator that translates and executes statements in the program one by one. It work on one statement at a time and if error free, executes the instruction before going to second instruction. Debugging is simpler in interpreter as it is done in stages. An interpreter takes more time for execution of a program as compared to a compiler.

Q.85 Design an algorithm to generate all the primes in the first *n* positive integers. **(7)**

Ans:
This algorithm will generate all prime numbers less than n.

```
void prime()
{
      int i,n,j;
      printf("enter any number");
      scanf("%d",&n);/*scan the number */
          i=2;
      while(i<n)
        { j=2;
           while(j<=i-1)
      {
        if(i%j==0)
          break;
        else
            j++;
      }
      if(j==i)
       printf("%d\n",i);
      i++;
         }
        }
```

Q.86 Explain various classes of datatypes of C **(4)**

Ans:
Following are the major classes of data types in C language:
1. **Primary data types**: Also known as fundamental/basic data types. All C compliers support four basic data types namely: integer(int), character(char), floating(float), and double precision floating point (double). Then there are extended data types such as long int, double int.
2**. Derived data types**: also known as secondary or user-defined data types. These data types, derived from basic data types, include arrays, pointer, structure, union, enum etc.

Q.87 What are escape sequences characters? List any six of them. **(4)**

Ans:

The characters which when used with output functions like printf( ),putc(),put() etc. helps in formatting the output are known as Escape sequence character. The following is a list of six escape sequences.

\n	Newline
\t	Horizontal Tab
\v	Vertical Tab
\b	Backspace
\r	Carriage Return
\\	Backslash

Q.88 Write a C program to calculate the average of a set of *N* numbers. **(8)**

Ans:

A C program to calculate the average of a set of n numbers:

```
#include<stdio.h>
#include<conio.h>
void main()
{
   int i,a[1000],n,sum;
   float average;
   clrscr();
   printf("how many elements you want to enter");
   scanf("%d",&n);
   printf("enter  values: \n");
   for(i=0;i<n;i++)
   scanf("%d",&a[i]);
   sum=0;
   for(i=0;i<n;i++)
   {
       sum=sum+a[i];
   }
   printf("\n%d",sum);
   average=(float)sum/n;
   printf("\n%f",average);
   getch();
}
```

Q.89 Compare the use of switch statement with the use of nested if-else statement. **(6)**

Ans:

If-else statement: When there are multiple conditional statements that may all evaluate to true, but we want only one if statement's body to execute. We can use an "else if" statement following an if statement and its body; that way, if the first statement is true, the "else if" will be ignored, but if the if statement is false, it will then check the condition for the else if statement. If the if statement was true the else statement will not be checked. It is possible to use numerous else if statements to ensure that only one block is executed.

```
#include <stdio.h>
      void main()
           {
          int age;
          printf( "Please enter your age" );
          scanf( "%d", &age );
          if ( age < 100 ) {
           printf ("You are pretty young!\n" );  }
          else if ( age == 100 ) {
```

```
            printf( "You are old\n" );
              }
          else {
            printf( "You are really old\n" );
          }
        }
```

Switch case statements are a substitute for long if statements that compare a variable to several "integral" values ("integral" values are simply values that can be expressed as an integer, such as the value of a char).The value of the variable given into switch is compared to the value following each of the cases, and when one value matches the value of the variable, the computer continues executing the program from that point. The condition of a switch statement is a value. The case says that if it has the value of whatever is after that case then do whatever follows the colon. The break is used to break out of the case statements. Break is a keyword that breaks out of the code block, usually surrounded by braces, which it is in. In this case, break prevents the program from falling through and executing the code in all the other case statements.

```
#include<stdio.h>
#include<conio.h>
void main()
{
    int flag;
    printf( "Enter any value\n" );
    scanf( "%d", &flag );
    switch ( flag ) {
        case 1:
            printf( "It is hot weather!\n" );
            break;
        case 2:
            printf( "It is a stormy weather!\n" );
            break;
        case 3:
            printf( "It is a sticky weather!\n" );
            break;
        default:
            printf( "It is a pleasant weather!\n" );
            break;
    }
    getch();
}
```

Q.90 What do you mean by underflow and overflow of data. **(2)**

Ans:

Underflow and overflow of data:

When the value of the variable is either too long or too small for the data type to hold, the problem of data overflow or underflow occurs. The largest value that a variable can hold depends on the machine. Since floating point values are rounded off to the number of significant digits allowed, an overflow results in the largest possible real value whereas an underflow results in zero. C does not provide any warning or indication of integer overflow, it simply give erroneous result.

Q.91 Write a C program to multiply two matrices (maximum size of the two matrices is 20 x 20 each). **(8)**

Ans:

A C program to multiply two matrices:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[20][20],b[20][20],c[20][20],i,j,k,m,n,p,q;
clrscr();
printf("enter no. of rows and column for 1st matrix\n");
printf("no.of rows\n");
scanf("%d",&m);
printf("no. of columns");
scanf("%d",&n);
printf("\nenter no. of rows and columns for 2nd matrix");
printf("\nno. of rows");
scanf("%d",&p);
printf("\nno. of columns");
scanf("%d",&q);
if(n==p)
{
   printf("enter elements for matrix A");
   for(i=0;i<m;i++)
  {
     for(j=0;j<n;j++)
       scanf("%d",&a[i][j]);
   }
   printf("enter elements for matrix B");
   for(i=0;i<p;i++)
  {
     for(j=0;j<q;j++)
        scanf("%d",&b[i][j]);
   }
    printf("\nMATRIX A:\n");
                for(i=0;i<m;i++)
  {
     for(j=0;j<n;j++)
     {
         printf("%d",a[i][j]);
         printf("\t");
      }
      printf("\n");
    }
    printf("\nMATRIX B:\n");
    for(i=0;i<p;i++)
    {
    for(j=0;j<q;j++)
   {
      printf("%d",b[i][j]);
      printf("\t");
    }
    printf("\n");
}
printf("\n");
printf("MULTIPLICATION :\n");
for(i=0;i<m;i++)
{
    for(j=0;j<q;j++)
   {
       for(k=0;k<p;k++)
      {
           c[i][j]=c[i][j]+(a[i][k]*b[k][j]);
       }
```

```
            printf("%d",c[i][j]);
            c[i][j]=0;
            printf("\t");
          }
          printf("\n");
        }
      }
      else
          printf("multiplication is not possible");

      getch();
      }
```

Q.92 Explain, in brief the purpose of the following string handling functions:
(i) strcat (ii) strcmp (iii) strcpy
Use suitable examples **(6)**

Ans:

(i) strcat() Function concatenates two strings together and has the following form:

```
strcat(string1,string2);
```

When this function is executed, string2 is appended to string1 by removing the null character at the end of string1.
C permits nesting of strcat functions as strcat(strcat(string1,string2),string3);
(ii) strcmp( ) is the string comparison function defined in string.h header file. It has the following form:.

```
int strcmp ( const char *s1, const char *s2 );
```

strcmp will accept two strings. It will return an integer. This integer will either be:
Negative if s1 is less than s2.
Zero if s1 and s2 are equal.
Positive if s1 is greater than s2.
Strcmp performs a case sensitive comparison; if the strings are the same except for a difference in case, then they're countered as being different. Strcmp also passes the address of the character array to the function to allow it to be accessed.
(iii) strcpy ( ) function is just like a string-assignment operator which take the following form:

```
char *strcpy ( char *dest, const char *src );
```

strcpy is short for string copy, which means it copies the entire contents of src into dest. The contents of dest after strcpy will be exactly the same as src such that strcmp (dest, src) will return 0.src may be a character array variable or a string constant.

Q.93 Write a C program to read a line of text containing a series of words from the terminal. **(7)**

Ans:

```
#include<stdio.h>
#include<conio.h>
void main()
{
   char text[100],ch;
   int i=0;
   clrscr();
   printf("Enter text. Press <return> at the end\n");
   do
   { ch=getchar();
     text[i]=ch;
     i++;
   }while(ch!='\n');
```

```
        i=i-1;
        text[i]='\0';
                    printf("The entered text is");
        printf("\n%s\n",text);
        getch();
    }
```

Q.94 Explain the need for user-defined functions. **(3)**

Ans:

The need for user-defined function:

1. A programmer may have a block of code that he has repeated forty times throughout the program. A function to execute that code would save a great deal of space, and it would also make the program more readable.
2. It is easy to locate and isolate a faulty function. Having only one copy of the code makes it easier to make changes.
3. Another reason for functions is to break down a complex program into logical parts. For example, take a menu program that runs complex code when a menu choice is selected. The program would probably best be served by making functions for each of the actual menu choices, and then breaking down the complex tasks into smaller, more manageable tasks, which could be in their own functions. In this way, a program can be designed that makes sense when read. And has a structure that is easier to understand quickly. The worst
programs usually only have the required function, main, and fill it with pages of jumbled code.
4. A function may be used by many other programs. A programmer can use already compiled function instead of starting over from scratch.

Q.95 Write a program in C to find the sum of the first 100 natural numbers
Sum=1+2+3+…….100 **(8)**

Ans:

A program to find the sum of first 100 natural numbers:

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
void main()
{
    int i,sum;
    clrscr();
    sum=0;
    for(i=1;i<=100;i++)
    {
        sum=sum+i;
     }
    printf("%d\t",sum);
    getch();
              }
```

Q.96 List any six commonly found programming errors in a C program. **(6)**

Ans:

Six commonly found errors in a C program are:

1. Missing or misplaced ; or }, missing return type for a procedure, missing or duplicate variable declaration.

2. Type mismatch between actual and formal parameters, type mismatch on assignment.
3. Forgetting the precedence of operators, declaration of function parameters.
4. Output errors means the program runs but produces an incorrect result. This indicates an error in the meaning of the program.
5. Exceptions that include division by zero, null pointer and out of memory.
6. Non-termination means the program does not terminate as expected, but continues running "forever."

Q.97 Define a structure in C, which stores subject-wise marks of a student. Using a student array, write a C program to calculate the total marks in each subject for all the students. **(8)**

Ans:

```
#include<stdio.h>
#include<conio.h>
struct marks
{
int sub1,sub2,sub3;
int total;
};
void main()
{
int i,n;
struct marks student[20];
struct marks total;
total.sub1=0;
total.sub2=0;
total.sub3=0;
clrscr();
printf("enter no. of students");
scanf("%d",&n);
for(i=0;i<n;i++)
{
 printf("\nenter marks :");
 scanf("%d%d%d",&student[i].sub1,
&student[i].sub2,&student[i].sub3);
 total.sub1=total.sub1+student[i].sub1;
 total.sub2=total.sub2+student[i].sub2;
 total.sub3=total.sub1+student[i].sub3;
 printf("\n");
}
printf("SUBJECT       TOTAL\n");
printf("Sub1          %d",total.sub1);
printf("\n");
printf("Sub2          %d",total.sub2);
printf("\n");
printf("Sub3          %d",total.sub3);
printf("\n");
getch();
    }
```

Q.98 (**a**) What is dynamic memory allocation? Explain the various memory allocation function with its task. **(3)**

(b) Why a linked list is called a dynamic data structure? What are the advantages of using linked list over arrays? **(6)**

(c) What is a macro? How is it different from a C variable name? What are the advantages of using macro definitions in a program. **(6)**

(d) What are the different modes in which a file can be opened. **(4)**

Ans:

(a) The mechanism of allocating required amount of memory at run time is called dynamic allocation of memory. Sometimes it is required to allocate memory at runtime. When we declare array in any program, the compiler allocates memory to hold the array. Now suppose the numbers of items are larger than the defined size then it is not possible to hold all the elements and if we define an array large enough and data to be stored is less, in that case the allocated memory is wasted leading to a need for dynamic memory allocation. In this mechanism we use a pointer variable to which memory is allocated at run time.

The various memory allocation functions are described below:

(i) malloc( ): It is a memory allocation function that allocates requested size of bytes and returns a pointer to the first byte of the allocated space. The malloc function returns a pointer of type void so we can assign it to any type of pointer. It takes the the following form:

```
ptr= (cast type *) malloc(byte-size);
```

where ptr is a pointer of type cast-type. For example, the statement

`x=(int *) malloc(10 *sizeof(int))` means that a memory space equivalent to 10 times the size of an int byte is reserved and the address of the first byte of memory allocated is assigned to the pointer x of int type.

The malloc function can also allocate space for complex data types such as structures. For example:

`ptr= (struct student*) malloc(sizeof (struct student));` where ptr is a pointer of type struct student.

(ii) calloc( ): It is another memory allocation function that allocates space for an array of elements, initializes them to zero and then returns a pointer to the memory. This function is normally used for requesting memory space at run time. While malloc allocates a single block of storage space, calloc allocates multiple block of storage, each of the same size, and then sets all bytes to zero. It takes the following form:

```
ptr= (cast type *) calloc(n,element-size);
```

This statement allocates contiguous space for n blocks, each of size element-size bytes.

(iii) realloc( ): realloc is a memory allocation function that modifies the size of previously allocated space. Sometime it may happen that the allocated memory space is larger than what is required or it is less than what is required. In both cases, we can change the memory size already allocated with the help of the realloc function known as reallocation of memory. For example, if the original allocation is done by statement

```
ptr= malloc(size);
```

then reallocation is done by the statement

`ptr=realloc(ptr,newsize);` which will allocate a new memory space of size newsize to the pointer variable ptr and returns a pointer to the first byte of the new memory block

(b) A linked list is called a dynamic data structure because it can be used with a data collection that grows and shrinks during program execution. The major advantage of using linked list over arrays is in implementing any data structure like stack or queue. The arrays are fixed in size and so a lot of memory gets wasted if we declare in prior the estimated size and the used space is less. Also it is not time efficient to perform deletion, insertion and updating of information in an array implementation because of

its fixed length memory storage. A linked list allocation storage result in efficient use of memory and computer time. Linked lists are useful over arrays because:
The exact amount of memory space required by a program depends on the data being processed and so this requirement cannot be found in advance. Linked list uses run-time allocation of memory resulting in no wastage of memory space.
Some programs require extensive use of data manipulation like insertion of new data, deletion and modification of old data. Linked lists are self referential structures so provide an efficient time complexity for these operations.
(c) A macro is a pre-processor directive which is a program that processes the source code before it passes through the compiler. These are placed in the source program before the main. To define a macro, # define statement is used. This statement, also known as macro definition takes the following general form:
#define identifier string
The pre-processor replaces every occurrence of the identifier in the source code by the string. The preprocessor directive definition is not terminated by a semicolon. For example
#define COUNT 100 will replace all occurrences of COUNT with 100 in the whole program before compilation.
(d) Different modes for opening a file are tabulated below:

Modes	Operations
"r"	Open text file for reading
"w"	Open text file for writing, previous content, if any, discarded
"a"	Open or create file for writing at the end of the file
"r+"	Open text file for update
"w+"	Create or open text file for update, previous content lost, if any
"a+"	Open or create text file for update, writing at the end.

Q.99 Explain the following directives:
#elif #pragma #error **(6)**

Ans:

(i) '#elif' is a preprocessor directive that provides alternate test facility. `#elif' stands for "else if". Like `#else', it goes in the middle of a conditional group and subdivides it; it does not require a matching `#endif' of its own. Like `#if', the `#elif' directive includes an expression to be tested. The text following the `#elif' is processed only if the original `#if'-condition failed and the `#elif' condition succeeds.

For example, suppose we have following set of statements:

```
#if X == 1
...
#else /* X != 1 */
#if X == 2
...
#else /* X != 2 */
...
#endif /* X != 2 */
#endif /* X != 1 */
```

Using *`#elif'*, we can abbreviate this as follows:

```
#if X == 1
...
#elif X == 2
```

```
...
#else /* X != 2 and X != 1*/
...
#endif /* X != 2 and X != 1*/
```

(ii) #pragma is an implementation oriented directive that specifies various instructions to be given to the compiler.

```
#pragma  name
```

causes compiler to perform "name"

(iii)#error is a preprocessor directive used to produce diagnostic messages during debugging.

```
#error message
```

causes the compiler to display the error message and terminate processing on encountering this directive.

Q.100 Using recursion, write a C program to reverse a given number. **(6)**

Ans:

```
#include<stdio.h>
#include<conio.h>
void main()
{
      int n,r;
      clrscr();
      printf("enter an integer");
      scanf("%d",&n);
      rev(n);
      getch();
}
rev (int n)
{
       if (n>0)
       {
         printf ("%d",n%10);
                rev(n/10);
       }
}
```

Q.101 Write a C function to delete a given item from a single linked list. Check for duplicate elements. **(8)**

Ans:

```
struct node
{
  int data ;
  struct node * link ;
} ;
void delete ( struct node **q, int num )
{
  struct node *old, *temp ;
  temp = *q ;

  while ( temp != NULL )
  {
        if ( temp -> data == num )
        {
/* if node to be deleted is the first node in the linked list */
              if ( temp == *q )
                    *q = temp -> link ;
```

```
            /* deletes the intermediate nodes in the linked list */
                  else
                        old -> link = temp -> link ;
                  /* free the memory occupied by the node */
                  free ( temp ) ;
                  return ;
            }
      /* traverse the linked list till the last node is reached */
            else
            {
                  old = temp ;  /* old points to the previous node */
                  temp = temp -> link ;  /* go to the next node */
             }
      }
      printf ( "\nElement %d not found", num ) ;
   }
```

Q.102 Consider the following:
P_1 is an integer pointer
P_2 is a long integer pointer
P_3 is a character type pointer
The initial value of P_1 is 2800, P_2 is 1411 and P_3 is 1201.
What is the new value of P_1 after $P_1=P_1+1$, P_2 after $P_2=P_2+1$ and P_3 after $P_3=P_3+1$; **(4)**

Ans:
The initial value of P_1 is 2800 which is an integer pointer so new value of P_1 is 2802 after $P_1= P_1+1$
The initial value of P_2 is 1411 which is a long integer pointer so new value of P_1 is 1415 after $P_2= P_2+1$ because long takes 4 bytes of memory.
The initial value of P_3 is 1201 which is a char type pointer so new value of P_3 is 1202 after $P_3= P_3+1$

Q.103 Differentiate between White Box and Black Box Testing. **(4)**

Ans:
White box testing strategy deals with the internal logic and structure of the code. White box testing also known as glass, structural, open box or clear box testing, tests code written, branches, paths, statements and internal logic of the code etc.
In order to implement white box testing, the tester has to deal with the code and hence is needed to possess knowledge of coding and logic i.e. internal working of the code. White box test also needs the tester to look into the code and find out which unit/statement/chunk of the code is malfunctioning. White box testing is applicable at unit, integration and system level however it is mainly done at unit level.
Black box testing Black-box test design treats the system as a "black-box", so it doesn't explicitly use knowledge of the internal structure. It takes an external perspective of the test object to derive test cases. These tests can be functional or non-functional, though usually functional. The test designer selects valid and invalid input and determines the correct output. There is no knowledge of the test object's internal structure.
This method of test design is applicable to all levels of software testing: unit, integration, functional testing, system and acceptance.

Q.104 Write a C program using pointers to compute the sum of all elements stored in an array. **(8)**

Ans:

```
void main()
{
       int i,*p,s;
       static int a[10]={10,20,30,40,50,60,70,80,90,100};
       clrscr();
       i=0;
      p=a;
       s=0;
      while(i<10)
      {
         s=s+ *p;
         i++;
  p++;
}
 printf("sum=%d",s);
 getch();
   }
```

Q.105 Write a C program to create a file contains a series of integer numbers and then reads all numbers of this file and write all odd numbers to other file called odd and write all even numbers to a file called even. **(8)**

Ans:

```
#include<stdio.h>
#include<conio.h>
void main()
   {
  FILE *f1,*f2,*f3;
  int i,j;
  printf("Enter the data\n");
  f1=fopen("file1","w");
  for(i=0;i<=10;i++)
  { scanf("%d",&j);
      if(j== -1) break;
     putw(j,f1);
   }
   fclose(f1);
   f1=fopen("file1","r");
   f2=fopen("od","w");
   f3=fopen("ev","w");

   while((j=getw(f1))!=EOF)
   { if(j%2==0)
      putw(j,f3);
      else
      putw(j,f2);
   }
   fclose(f1);
   fclose(f2);
   fclose(f3);
   f2=fopen("od","r");
   f3=fopen("ev","r");
   printf("\nContents of odd file\n");
   while((j=getw(f2))!=EOF)
      printf("%4d",j);
   printf("\nContents of even file\n");
```

```
    while((j=getw(f3))!=EOF)
       printf("%4d",j);
      fclose(f2);
      fclose(f3);
      getch();
}
```

Q.106 What is structured programming? Explain its advantages. **(8)**

Ans:

Structured Programming: means the collection of principles and practices that are directed toward developing correct programs which are easy to maintain and understand. A structured program is characterized by clarity and simplicity in its logical flow.

The Three important constructs upon which structured programming is built are:

- Sequence-Steps are performed one after the other.
- Selection-Logic is used to determine the execution of a sequence of steps. For example,if-then-else construct.
- Iteration-A set of actions are repeated until a certain condition is met. For example while construct.

The various advantages of structured programming are:

1. Complexity of program reduces.
2. Easy maintenance.
3. Simplified understanding.
4. Reusability of code increases.
5. Testing and debugging become easier.

Q.107 What are the essential features of an algorithm? Write an algorithm to obtain H.C.F. of two given positive integers. **(8)**

Ans:

Essential features of an algorithm:

1.Input: The algorithm should take zero or more input.

2. Output: The algorithm should produce one or more outputs.

3. Definiteness: Each and every step of algorithm should be defined unambiguously.

4. Effectiveness: A human should be able to calculate the values involved in the procedure of the algorithm using paper and pencil.

5. Termination: An algorithm must terminate after a finite number of steps.

Writing algorithms is an art so the language used to write algorithms should be simple and precise. Each and every step of the algorithm should be clear and unambiguous and should not convey more than one meaning to the programmer.

A C language algorithm to obtain H.C.F. of two given positive integers is listed below:

```
void hcf()
{
 int a,b,r,h,k,c;
 clrscr();
 printf("enter two numbers");
 scanf("%d%d",&a,&b);
 h=a;
 k=b;
 while(r!=0)
```

```
  {
     r=a%b;
     a=b;
     b=r;
  }
  printf("H.C.F. of %d and %d = %d",h,k,a);
}
```

Q.108 How do you calculate the complexity of sorting algorithms? Find the complexity of Insertion sort and Bubble Sort. **(8)**

Ans:

The complexity of sorting algorithms depends on the input: sorting a thousand numbers takes longer than sorting three numbers. The best notion of input size depends on the problem being studied. For sorting algorithms, the most natural measure is the number of items in the input that is array size n. We start by associating time "cost" of each statement and the number of times each statement is executed assuming that comments are not executable statements, and so do not take any time.

Complexity of Bubble sort: $O(n^2)$

Complexity of Insertion sort: $O(n^2)$

Q.109 (**a**) Write a C program to compare and copy the structure variables with example. **(8)**

(b) Write a C program using pointers to determine the length of a character string. **(8)**

(**c**) Compare the 2 constructs Arrays of structures and Arrays within structure. Explain with the help of examples. **(8)**

Ans:

(**a**) A C program to compare and copy the structure variables is listed below:

```
#include<stdio.h>
 #include<conio.h>
 struct student
 {
    char name[10];
    int marks;
 };
 void main()
 {
    int i,j;
    static struct student stu1={"Simmi",78};
    static struct student stu2={"Ruci",90};
    static struct student stu3={"Seema",95};
    clrscr();
    stu1=stu3;
    if((stu1.marks==stu3.marks))
    {
        printf(" Student 1 is same as student 3\n");
        printf("%s",stu1.name);
        printf("\t");
        printf("%d",stu3.marks);
        printf("\n");
    }
    else
    printf("\nStudents are different\n");
    getch();
```

```
}
```

(b) A C program to determine the length of a character string:

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
void main()
{  char *a,*ptr;
   int i;
    clrscr();
 printf("enter the string");
 gets(a);
 ptr=a;
 while(*ptr!='\0')
{   ptr++;
  }
  i=ptr-a;
printf("length=%d",i);
            getch();
             }
```

(c) Array of structures and arrays within structures:

Structures variables are just like ordinary variables so we can create an array of structures. However arithmetic operations cannot be performed on structure variables. For example, we can create an array of structure student as shown below:

```
struct student
      {
            char name[30];
            int marks;
      };
```

`struct student list[4]={ { "aaa",90},{"bbb",80},{"ccc",70}};` declare and initialize an array of structure student.

A Structure is a heterogeneous collection of variables grouped under a single logical entity. An array is also a use-defined data type and hence can be used inside a structure just like an ordinary variable. As in the above example, name is a char array within the structure student.

Q.110 Describe the output that will be generated by the following 'C' program. **(4)**

```
#include<stdio.h>
main( )
{
    int i=0, x=0;
    while(i<20)
    { if(i%5 = = 0)
        {x + = i;
          printf("%d", x);
         }
         ++i;
     }
printf("\nx = %d", x);
}
```

Ans:

The output generated by given C program is:

051530

x=30

initially i=0<20, (i%5==0) is true so x=x+i=0+0=**0**, i is incremented to 2, again the next while loop, however condition (i%5==0) is false so i incremented and so on till i=5.

At i=5, (i%5==0) is true so x=x+i=0+5=**5**, i is incremented to 6, again the next while loop, however condition (i%5==0) is false so i incremented and so on till i=10.
At i=10, (i%5==0) is true so x=x+i=5+10=**15**, i is incremented to 11, again the next while loop, however condition (i%5==0) is false so i incremented and so on till i=15.
At i=15, (i%5==0) is true so x=x+i=15+15=**30**, i is incremented to 16, again the next while loop, however condition (i%5==0) is false so i incremented and out of while loop at i=20.
Outer block, it is printing the value of x which is **30**.

Q.111 Write a complete program to create a singly linked list. Write functions to do the following operations

(i) Count the number of nodes
(ii) Add a new node at the end
(iii) Reverse the list. **(10)**

Ans:

A complete program to create a linked list, counting number of nodes, adding new node at the end and reversing the list is as follows:

```
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
/* structure containing a data part and link part */
struct node
{
 int data ;
 struct node *link ;
} ;
void append ( struct node **, int ) ;
void reverse ( struct node ** ) ;
void display ( struct node * ) ;
int count ( struct node * ) ;
void main( )
{
 struct node *p ;
 int n;
 p = NULL ;  /* empty linked list */
 append ( &p, 14 ) ;
 append ( &p, 30 ) ;
 append ( &p, 25 ) ;
 append ( &p, 42 ) ;
 append ( &p, 17 ) ;
 clrscr( ) ;
 display ( p ) ;
 printf ( "\nNo. of elements in the linked list = %d", count (
p ) ) ;
   printf("\n Enter the element you want to insert");
 scanf("%d",&n);
 append (&p,n);
 display ( p ) ;
 printf ( "\nNo. of elements in the linked list = %d", count (
p ) ) ;
 reverse ( &p ) ;
 printf("\n Elements after reversing the linked list\n");
 display (p);
}
/* adds a node at the end of a linked list */
void append ( struct node **q, int num )
{
 struct node *temp, *r ;
```

```
  if ( *q == NULL ) /* if the list is empty, create first node
*/
  {
        temp = malloc ( sizeof ( struct node ) ) ;
        temp -> data = num ;
        temp -> link = NULL ;
        *q = temp ;
  }
  else
  {
        temp = *q ;
        /* go to last node */
        while ( temp -> link != NULL )
              temp = temp -> link ;
        /* add node at the end */
        r = malloc ( sizeof ( struct node ) ) ;
        r -> data = num ;
        r -> link = NULL ;
        temp -> link = r ;
  }
}
void reverse ( struct node **x )
{
  struct node *q, *r, *s ;
  q = *x ;
  r = NULL ;
  /* traverse the entire linked list */
  while ( q != NULL )
  {
        s = r ;
        r = q ;
        q = q -> link ;
        r -> link = s ;
  }
  *x = r ;
}
int count(struct node *list)
{
  if(list->next == NULL)
        return(0);
  else
        return(1+count(list->next));
}
/* displays the contents of the linked list */
void display ( struct node *q )
{
  printf ( "\n" ) ;
  /* traverse the entire linked list */
  while ( q != NULL )
  {
        printf ( "%d ", q -> data ) ;
        q = q -> link ;
  }
}
```

Q.112 What is recursion? Write a recursive program to reverse a string. **(8)**

Ans:

Recursion is a special case of function call where a function calls itself. These are very useful in the situations where solution can be expressed in terms of

successively applying same operation to the subsets of the problem. For example, a recursive function to calculate factorial of a number n is given below:

```
fact(int n)
{
int factorial;
if(n==1||n==0)
      return(1);
else
factorial=n*fact(n-1);
return (factorial);
}
```

Assume n=4, we call fact(4)

Since n≠1or 0, factorial=n*fact(n-1)

Factorial=4*fact(3)	(again call fact function with n=3)
=4*3*fact(2)	(again call fact function with n=2)
=4*3*2*fact(1)	(again call fact function with n=1)
=4*3*2*1	(terminating condition)
=24	

We should always have a terminating condition with a recursive function call otherwise function will never return.

A recursive program to reverse a string is:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    char str[100];
    clrscr();
    printf("enter a string");
    gets(str);
    printf("reverse of string is:\n");
    rev(str);
    getch();
}
rev (char *string)
{
    if (*string)
   {
       rev(string+1);
       putchar(*string);
    }
}
```

Q.113 Distinguish between the following:

(i) Automatic and static variables

(ii) Global and local variables. **(8)**

Ans:

(i) **Automatic and Static variables:**

Automatic variables: The features are as follows

Declaration place:-declared inside a function in which they are to be utilized, that's why referred as local or internal variables.

Declaration syntax:- A variable declared inside a function without storage class specification by default is an automatic variable. However, we may use the keyword auto to declare it explicitly.

```
main()
{
```

```
auto int age;
   }
```

Default initial value:- Garbage value
Scope:-created when the function is called and destroyed on exit from the function.
Life:- till the control remains within the block in which defined.
Static variables: The features are as follows
Declaration place:-may be declared internally or externally.
Declaration syntax:-we use the keyword static to declare a static variable.
Static int age;
Default initial value:- Zero
Scope:-in case of internal static variable, the scope is local to the function in which defined while scope of external static variable is to all the functions defined in the program.
Life:- value of variable persists between different function calls.
(ii) **Global and Local Variables**
Global variables: The features are as follows
Declared outside of all functions or before main.
These can be used in all the functions in the program.
It need not be declared in other functions.
A global variable is also known as an external variable.
Local variables: The features are as follows
Declared inside a function where it is to be used.
These are not known to other function in the program.
These variables are visible and meaningful inside the functions in which they are declared.
For example:

```
#include<stdio.h>
int m;
void main( )
{ int i;
  …..
…..
 Fun1();
}
Fun1( )
{ int j;
  ….
…..
}
```

Here m is a global variable , i is local variable local to main( ) and j is local variable local to Fun1( ).

Q.114 What would be the output of the program given below? **(6)**

```
typedef struct soldier{
char *name;
char *rank;
int serial_number;} SOLDIER;
SOLDIER soldier1, soldier2, soldier3, *ptr;
ptr = &soldier3;
soldier3.name = "Anand Mohanti";
printf("\n%s", (*ptr).name);
printf("\n%c", *ptr->name);
printf("\n%c", *soldier3.name);
```

```
printf("\n%c", *(ptr-> name + 4));
```

Ans:

Output of the program would be:

Anand Mohanti
A
A
d

ptr is pointing to address of soldier3 so (*ptr).name would print the soldier3.name that is Anand Mohanti

*ptr->name would print first character of soldier3 name so printed 'A'.

Similarly 3rd line.

ptr->name means first char of name so ptr->name+4 points to 4th location from beginning so this statement will print d.

Q.115 Define a structure to store the following information about an employee Name, Sex(male, female), Marital_status(single, married, divorced or widowed), age.(using bit fields) **(4)**

Ans:

Definition of a structure to store information of an employee:

```
struct employee
{
              char name[20];
     unsigned sex:  1;
     unsigned martial_status:     1;
     unsigned age:  7;
}emp;
```

Q.116 How is multidimensional arrays defined in terms of an array of pointer? What does each pointer represent? **(6)**

Ans:.

An element in a multidimensional array like two-dimensional array can be represented by pointer expression as follows:

```
*(*(a+i)+j)
```

It represent the element a[i][j]. The base address of the array a is &a[0][0] and starting at this address, the compiler allocates contiguous space for all the element row-wise. The first element of second row is immediately after last element of first row and so on.

Q.117 Write a program to implement the stack using linked list. **(8)**

Ans:

A C program to implement the stack using linked list is given below:

```
/*  Program to implement stack as linked list*/
#include <stdio.h>
#include <conio.h>
#include <alloc.h>
/* structure containing data part and linkpart */
struct node
{
  int data ;
  struct node *link ;
```

```
} ;
void push ( struct node **, int ) ;
int pop ( struct node ** ) ;
void delstack ( struct node ** ) ;
void display ( struct node * ) ;
void main( )
{
 struct node *s = NULL ;
 int i ;
 clrscr( ) ;
 push ( &s, 1 ) ;
 push ( &s, 2 ) ;
 push ( &s, 3 ) ;
 push ( &s, 4 ) ;
 push ( &s, 5 ) ;
 push ( &s, 6 ) ;
 printf("The stack currently contains\n");
       display ( s ) ;
 i = pop ( &s ) ;
 printf ( "\nItem popped: %d", i ) ;
 i = pop ( &s ) ;
  printf ( "\nItem popped: %d", i ) ;
 i = pop ( &s ) ;
 printf ( "\nItem popped: %d", i ) ;
 printf("Stack after deleting three items\n");
       display (s);
 delstack ( &s ) ;
 getch( ) ;
}
/* adds a new node to the stack as linked list */
void push ( struct node **top, int item )
{
 struct node *temp ;
temp = ( struct node * ) malloc ( sizeof ( struct node ) ) ;
 if ( temp == NULL )
       printf ( "\nStack is full." ) ;
 temp -> data = item ;
 temp -> link = *top ;
 *top = temp ;
}
/* pops an element from the stack */
int pop ( struct node **top )
{
 struct node *temp ;
 int item ;
 if ( *top == NULL )
 {
       printf ( "\nStack is empty." ) ;
       return NULL ;
 }
 temp = *top ;
 item = temp -> data ;
 *top = ( *top ) -> link ;
 free ( temp ) ;
 return item ;
}
/* displays the contents of the stack */
void display ( struct node *q )
{
 printf ( "\n" ) ;
 /* traverse the entire stack */
 while ( q != NULL )
 {
```

```
        printf ( "%d ", q -> data ) ;
        q = q -> link ;
  }
}
/* deallocates memory */
void delstack ( struct node **top )
{
  struct node *temp ;
  if ( *top == NULL )
        return ;
  while ( *top != NULL )
  {
        temp = *top ;
        *top = ( *top ) -> link ;
        free ( temp ) ;
  }
}
```

Q.118 Write a C program to calculate the standard deviation of an array of values. The array elements are read from the terminal. Use functions to calculate standard deviations and mean Standard Deviation **(10)**

Ans:

A C program to calculate mean and standard deviation of an array of value:

```
#include<math.h>
#include<conio.h>
#define MAXSIZE 100
void main()
{
  int i,n;
  float
value[MAXSIZE],deviation,sum,sumsqr,mean,variance,stddeviation
;
  sum=sumsqr=n=0;
  clrscr();
  printf("\n Input values : input -1 to end \n");
  for(i=1;i<MAXSIZE;i++)
  {
        scanf("%f",&value[i]);
        if(value[i]==-1)
              break;
        sum+=value[i];
        n+=1;
  }
  mean=sum/(float)n;
  for(i=1;i<=n;i++)
  {
        deviation=value[i]-mean;
        sumsqr+=deviation*deviation;
  }
  variance=sumsqr/(float)n;
  stddeviation=sqrt(variance);
  printf("\n Number of items : %d\n",n);
  printf("              Mean : %f\n",mean);
  printf("Standard Deviation : %f\n",stddeviation);
  getch();
    }
```

Q.119 A file of employees contains data (eno, name and salary) for each employee of a company. Write a program to do the following:

(i) create the file

(ii) insertion in a file
(iii) deletion from a file
(iv) modification in a file **(12)**

Ans:

A program to perform creation, insertion, deletion and modification in a file:

```
#include<stdio.h>
struct rec
{
    int empno;
    char name[25],add[50];
    float salary;
}e;
FILE *fp1,*fp2;
main()
{
   int ch;
    while (1)
    {
clrscr();printf    ("\n1.Insert    a    new    Record\n2.Display
All\n3.Modify One\n4.Delete One\n5.Exit\nEnter your choice:-> ");
    scanf ("%d",&ch);
    switch (ch)
    {
           case 1:      insert();break;
           case 2:      display();break;
           case 3:      modify();break;
           case 4:      del();break;
           case 5:      exit();
           default: printf ("\nOut of Range");
    }
}
getch();
}
insert ()
{
if (fp1==NULL)
       fp1=fopen("Employee.txt","w");
else
       fp1=fopen("Employee.txt","a");
       fseek(fp1,0L,SEEK_END);
       printf ("\nEnter the Employee No.:-> ");
       flushall();
       scanf ("%d",&e.empno);
       printf("\nEnter name of Employee:-> ");
       flushall();
       gets(e.name);
       printf ("\nEnter Address of Employee:-> ");
       flushall();
       gets(e.add);
       printf ("\nEnter Salary of Employee:-> ");
       flushall();
       scanf("%f",&e.salary);
       fwrite (&e,sizeof(e),1,fp1);
       fclose(fp1);
       printf ("\nRecord Added.");
       getch();
}
display()
{
 int i=1;
```

```
 fp1=fopen("Employee.txt","r");
 if (fp1==NULL)
 {
        printf("\nUnable to open file");
        getch();
        return ;
 }
 printf("\nEmp. No\tName\tAddress\tSalary\n");
 while(1)
 {
        i=fread(&e,sizeof(e),1,fp1);
        if (i==0)break;
        printf("\n%d\t%s\t%s\t%f",e.empno,e.name,e.add,e.salary);
 }
 fclose(fp1);
 printf("\nThanks");
 getch();
}
del()
{
 char name[25];
 int flag=0,i=0,sz=0,count=0;
 fp1=fopen("Employee.txt","r");
 fp2=fopen("Temp.Tmp","w");
 if (fp2==NULL)
 {
        printf ("\nMemory Allocation not possible");
        getch();
        return ;
 }
 if (fp1==NULL)
 {
        printf("\nUnable to open file");
        getch();
        return ;
 }
 printf ("\nEnter Name of Employee:-> ");
 flushall();
 gets(name);
 i=0;
 while(1)
 {
        flag=0;
        i=fread(&e,sizeof(e),1,fp1);
        if (i==0)
               break;
        if (strcmp(e.name,name)==0)
        {
               flag=1;
               printf("\nThis is the data u want to modify\n");
               printf
("%d\t%s\t%s\t%f\n",e.empno,e.name,e.add,e.salary);
        }
        else
        {
               fwrite (&e,sizeof(e),1,fp2);
        }
 }
 fclose(fp1);
 fclose(fp2);
 rename(fp2,fp1);
 getch();
}
```

```
modify()
{
 char name[25];
 int flag=0,i=0,sz=0,count=0;
 fp1=fopen("Employee.txt","r");
 fp2=fopen("Temp.Tmp","w");
 if (fp2==NULL)
 {
        printf ("\nMemory Allocation not possible");
        getch();
        return ;
 }
 if (fp1==NULL)
 {
        printf("\nUnable to open file");
        getch();
        return ;
 }
 printf ("\nEnter Name of Employee:-> ");
 flushall();
 gets(name);
 i=0;
 while(1)
 {
        flag=0;
        i=fread(&e,sizeof(e),1,fp1);
        if (i==0)
               break;

        if (strcmp(e.name,name)==0)
        {
               flag=1;
        printf("\nThis is the data u selected for change\n");
               printf
("%d\t%s\t%s\t%f\n",e.empno,e.name,e.add,e.salary);
               printf ("\nEnter new data\n");
               printf ("\nEnter the Employee No.:-> ");
               flushall();
               scanf ("%d",&e.empno);
               printf("\nEnter name of Employee:-> ");
               flushall();
               gets(e.name);
               printf ("\nEnter Address of Employee:-> ");
               flushall();
               gets(e.add);
               printf ("\nEnter Salary of Employee:-> ");
               flushall();
               scanf("%f",&e.salary);
               fwrite(&e,sizeof(e),1,fp2);
               printf("\nRecord Modified");
        }
        else
        {
               fwrite (&e,sizeof(e),1,fp2);
        }
 }
 fclose(fp1);
 fclose(fp2);
 rename(fp2,fp1);
 getch();
}
```

Q.120 What is insertion sort? Write a program to sort the given list of integers using insertion sort. **(4)**

Ans:

Insertion Sort: One of the simplest sorting algorithms is the insertion sort. Insertion sort consists of n - 1 passes. For pass p = 2 through n, insertion sort ensures that the elements in positions 1 through p are in sorted order. Insertion sort makes use of the fact those elements in positions 1 through p - 1 are already known to be in sorted order. To insert a record, we must find the proper place where insertion is to be made.

A C program to sort integers using insertion sort is given below:

```
#include<stdio.h>
#include<conio.h>
void main()
{
int a[20],i,j,n,val;
clrscr();
printf("how many numbers u want to enter :\n");
scanf("%d",&n);
printf("\nenter the numbers :\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
for(i=1;i<n;i++)
     {
      int val=a[i];
      for(j=i-1;j>=0&&val<a[j];j--)
        a[j+1]=a[j];
      a[j+1]=val;
printf("sorted elements :\n");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
printf("\n");
getch();
}
```

Q.121 Define a macro. Write a nested macro that gives the minimum of three values. **(5)**

Ans:

Macro is a preprocessor directive, also known as macro definition takes the following general form:

```
#define identifier string
```

The pre-processor directive replaces every occurrence of the identifier in the source code by the string. The preprocessor directive definition is not terminated by a semicolon. For example

`#define COUNT 100` will replace all occurrences of COUNT with 100 in the whole program before compilation.

A nested macro that gives minimum of three values is listed below:

```
#include<stdio.h>
#include<conio.h>
#define min(a,b) ((a>b)?b:a)
#define minthree(a,b,c) (min(min(a,b),c))
void main()
{
int x,y,z,w;
clrscr();
printf("enter three numbers :\n");
scanf("%d%d%d",&x,&y,&w);
```

```
z=minthree(x,y,w);
printf("Minimum of three value is %d",z);
getch();
}
```

Q.122 Write a program to check whether a box is cube, rectangle or semi-rectangle. Prepare a test procedure to test this program. **(5)**

Ans:

A C program to check if a box is cube, rectangle or semi-rectangle is:

```
#include<stdio.h>
#include<conio.h>
void main()
{
    float length, width, height;
    clrscr();
    printf("enter the dimension of box\n");
    printf("Length=  ");
    scanf("%f",&length);
    printf("width=  ");
    scanf("%f",&width);
    printf("height=  ");
    scanf("%f",&height);
if(length==width)
{  if(length==height)
     printf("\nThe box is a cube\n");
   else
     printf("\nThe box is a rectangle\n");
}
if(width==(0.5*length))
    printf("\nThe box is a semirectangle\n");
getch();
}
```

Q.123 A program has been compiled and linked successfully. When you run this program you face one or more of the following situations

(i) Program executed, but no output
(ii) It produces incorrect answers
(iii) It does not stop running.

What are the possible causes in each case and what steps would you take to correct them? **(6)**

Ans:

A program has been compiled and linked successfully.

(i) Program executed, but no output: this usually happens due to run time errors in the program like referencing an out-of-range array element or mismatch of data types.

(ii) It produces incorrect answers: Then there may be logical errors in the program like failure to consider a particular condition or incorrect translation of the algorithm into the program or incorrect order of evaluation of statements etc.

(iii) It does not stop running: This happens when we make use of correct syntax statement but incorrect logic like `if(code=1) count++;`

Instead of using comparison operator we are using assignment which is syntactically correct so count++ is always executed resulting in infinite loop. Similar mistakes may occur in other control statements, such as for and while loop that causes infinite loops and does not stop running.

Q.124 Design an algorithm to generate n^{th} member of the Fibonacci sequence. **(8)**

Ans:
An algorithm to generate nth member of Fibonacci sequence is:

1. start
2. Scan the number 'n' upto which the series to be generated.
3. Initialize Sum=0, x=0, y=1 and i=3.
4. Print x and y as the part of series.
5. Repeat a-e until i<=n
   a. Calculate Sum=x+y
   b. Print Sum as part of series.
   c. Assign y to x
   d. Assign sum to y
   e. i=i+1
6. Stop.

Q.125 Write an algorithm to sort an arrays of integers using bubble sort technique. **(8)**

Ans:
A C algorithm to sort an array using bubble sort technique:

```
void main()
{
int a[20],i,j,n,c,flag;
printf("how many numbers u want to enter :\n");
scanf("%d",&n);
printf("\nenter the numbers :\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
for(i=0;i<n-1;i++)
{
for(j=0;j<n-1-i;j++)
{
if(a[j]>a[j+1])
{
c=a[j];
a[j]=a[j+1];
a[j+1]=c;
flag=0;
}
}
if(flag)
 break;
else
 flag=1;
}
printf("sorted elements :\n");
for(i=0;i<n;i++)
printf("%d\t",a[i]);
printf("\n");
}
```

Q.126 In a company an employee is paid as under:
(i) HRA=10% of Basic salary and DA=55% of Basic salary.

(ii) If his Basic salary is greater than 1500 then HRA=500/- and DA=60% of Basic Salary.
Write a C program to find Gross salary of the employee. **(8)**

Ans:
A C program to find gross salary of an employee is listed below:

```
void main()
 {
 float basic,hra,da,gross;
 clrscr();
 printf("\n Enter the basic salary of the employee : Rs.");
 scanf("%f",&basic);
 if(basic>1500)
 {
        hra=500;
        da=(60.00/100.00)*basic;
 }
 else
 {
        hra=(10.00/100.00)*basic;
        da=(55.00/100.00)*basic;
 }
 gross=basic+hra+da;
 printf("\n The Gross Salary is : Rs. %f",gross);
 getch();
}
```

Q.127 What is looping? **(5)**

Ans:
Loop is a control structure used to perform repetitive operation. Some programs involve repeating a set of instruction either a specified number of times or until a particular condition is met. This is done using a loop control structure. A program loop consists of two parts: Body of the loop and control statement. The control statement tests certain conditions and then decides repeated execution or termination of statements. Most real programs contain some construct that loops within the program, performing repetitive actions on a stream of data or a region of memory.

Q.128 What are Multidimensional Arrays? Using multidimensional array, write a program in C to sort a list of names in alphabetical order. **(11)**

Ans:
Multidimensional array: Multidimensional arrays can be described as "arrays of arrays". For example, a bidimensional array can be imagined as a bidimensional table made of elements, all of them of a same uniform data type.

`int arr[3][5];` represents a bidimensional array of 3 per 5 elements of type `int`.

Similarly a three dimensional array like `int arr[3][4][2];` represent an outer array of three elements , each of which is a two dimensional array of four rows, each of which is a one dimensional array of five elements.
Multidimensional arrays are not limited to two or three indices (i.e., two dimensional or three dimensional). They can contain as many indices as needed. However the amount of memory needed for an array rapidly increases with each dimension. For example:

`char arr [100][365][24][60][60];` declaration would consume more than 3 gigabytes of memory.

Memory does not contain rows and columns, so whether it is a one dimensional array or two dimensional arrays, the array elements are stored linearly in one continuous chain. For example, the multidimensional array is stored in memory just like an one-dimensional array shown below:

```
int arr[3][4][2]= {
                    { {1,2},{3,4},{5,6},{7,8} },
                  { {9,1},{1,2},{3,7},{4,7} },
                   { {6,1},{18,19},{20,21},{22,23} },
                   };
```

1	2	3	4	5	6	7	8	9	1	1	2	3	7	4	7	6	1	18	19	20	21	22	23

101 **117** **133**

Multidimensional arrays are just an abstraction for programmers, since we can obtain the same results with a simple array just by putting a factor between its indices.

A C program to sort a list of names in alphabetical order:

```
#include<stdio.h>
#include<conio.h>
void main()
  {
int i,j,n;
char a[10][20],b[20];
clrscr();
printf("how namy names u want to enter");
scanf("%d",&n);
printf("enter names :\n");
for(i=0;i<n;i++)
scanf("%s",a[i]);
for(i=0;i<=n-2;i++)
    {
     for(j=0;j<n-i-1;j++)
     {
          if(cmp(a[j],a[j+1])>0)
          {
           cpy(b,a[j]);
           cpy(a[j],a[j+1]);
           cpy(a[j+1],b);
            }
     }
    }
    for(i=0;i<n;i++)
    printf("\n%s\n",a[i]);
    getch();
}
void cpy(char *b,char *a)
{
     int i=0;
     while(a[i]!='\0')
     {
      b[i]=a[i];
      i++;
     }
     b[i]='\0';
}
int cmp(char *a,char *b)
{
```

```
        int i=0;
        while(a[i]!='\0'||b[i]!='\0')
        {
          if(a[i]>b[i])
              break;
          if(b[i]>a[i])
              break;
          else
          {
               i++;
          }
        }
        if(a[i]>b[i])
         return(1);
        if(b[i]>a[i])
          return(-1);
}
```

Q.129 What do you understand by scope, lifetime and visibility of the variables? **(6)**

Ans:

Scope, Visibility and Lifetime of variables:

The scope of a variable determines the region of the program in which it is known. An identifier's "visibility" determines the portions of the program in which it can be referenced—its "scope." An identifier is visible only in portions of a program encompassed by its "scope," which may be limited to the file, function or block in which it appears.

File scope: The variables and functions with file scope appear outside any block or list of parameters and is accessible from any place in the translation unit after its declaration. Identifier names with file scope are often called "global" or "external." The scope of a global identifier begins at the point of its definition or declaration and terminates at the end of the translation unit. A function has file scope.

Function scope: A label is the only kind of identifier that has function scope. A label is declared implicitly by its use in a statement. Label names must be unique within a function however a label having the same name in two different functions is allowed.

Block scope: The variables with block scope appear inside a block or within the list of formal parameter declarations in a function definition. It is visible only from the point of its declaration or definition to the end of the block containing its declaration or definition.

Q.130 What is the smallest value of n such that an algorithm whose running time is $50n^3$ runs faster than an algorithm whose running time is 3^n on the same machine? Justify your answer. **(6)**

Ans:

The running time of an algorithm depends upon various characteristics and slight variation in the characteristics varies the running time. The algorithm efficiency and performance in comparison to alternate algorithm is best described by the order of growth of the running time of an algorithm. Suppose one algorithm for a problem has time complexity as c_3n^2 and another algorithm has $c_1n^3 + c_2n^2$ then it can be easily observed that the algorithm with complexity c_3n^2 will be faster than the one with complexity $c_1n^3 + c_2n^2$ for sufficiently larger values of n. Whatever be the value of c_1,

$c_{2 \text{ and }} c_3$ there will be an 'n' beyond which the algorithm with complexity c_3n^2 is faster than algorithm with complexity $c_1n^3 + c_2n^2$, we refer this n as breakeven point.
Here running time of algorithms are $50*n^3$ and 3^n, if we compare both as shown in the following table, we find that 10 is the smallest value of n (9.8) for which $50*n^3$ will run faster than 3^n.

n	**$50*n^3$**	**3^n**
2	400	9
4	3200	81
5	6250	243
9	36450	19683
10	50000	59049

Q.131 Define a sparse matrix. Explain an efficient way of storing sparse matrix in the memory of a computer. Write an algorithm to find the transpose of sparse matrix using this representation. **(10)**

Ans:
Sparse Matrix Definition: - A matrix in which number of zero entries are much higher than the number of non zero entries is called sparse matrix.
An efficient way of storing sparse matrix
The natural method of representing matrices in memory as two-dimensional arrays may not be suitable for sparse matrices. One may save space by storing only nonzero entries. For example matrix A (4*4 matrix) represented below

0	0	0	15
0	0	0	0
0	9	0	0
0	0	4	0

Here the memory required is 16 elements x 2 bytes = 32 bytes
The above matrix can be written in sparse matrix form as:

4	4	3
0	3	15
2	1	9
3	2	4

Here the memory required is 12 elements x 2 bytes = 24 bytes
where first row represent the dimension of matrix and last column tells the number of non zero values; second row onwards it is giving the position and value of non zero number.
An algorithm to find transpose of a sparse matrix is as below:

```
void transpose(x,r,y)
int x[3][3],y[3][3],r;
{
     int i,j,k,m,n,t,p,q,col;
     m=x[0][0];
      n=x[0][1];
      t=x[0][2];
   y[0][0]=n;
   y[0][1]=m;
   y[0][2]=t;
  if(t>0)
{
```

```
     q=1;
      for (col=0;col<=n;col++)
         for(p=1;p<=t;p++)
           if(x[p][1]==col)
           {
                y[q][0]=x[p][1];
     y[q][1]=x[p][0];
     y[q][2]=x[p][2];
        q++;
 }
}
return;
}
```

Q.132 What do you understand by row-major order and column-major order of arrays? Derive their formulae for calculating the address of an array element in terms of the row-number, column-number and base address of array. **(8)**

Ans:

A two dimensional array is declared similar to the way we declare a one-dimensional array except that we specify the number of elements in both dimensions. For example,

```
int grades[3][4];
```

The first bracket ([3]) tells the compiler that we are declaring 3 pointers, each pointing to an array. We are not talking about a pointer variable or pointer array. Instead, we are saying that each element of the first dimension of a two dimensional array reference a corresponding second dimension. In this example, all the arrays pointed to by the first index are of the same size. The second index can be of variable size. For example, the previous statement declares a two-dimensional array where there are 3 elements in the first dimension and 4 elements in the second dimension.

Two-dimensional array is represented in memory in following two ways:

1. **Row major representation**: To achieve this linear representation, the first row of the array is stored in the first memory locations reserved for the array, then the second row and so on.
2. **Column major representation**: Here all the elements of the column are stored next to one another.

In row major representation, the address is calculated in a two dimensional array as per the following formula

```
The address of a[i] [j] =base (a) + (i*m+ j) * size
```

where *base*(a) is the address of a[0][0], *m* is second dimension of array a and *size* represent size of the data type.

Similarly in a column major representation, the address of two dimensional array is calculated as per the following formula

```
The address of a[i] [j] = base (a) + (j*n +i) * size
```

Where *base (a)* is the address of a [0] [0], *n* is the first dimension of array a and *size* represents the size of the data type.

Q.133 Using stacks, write an algorithm to determine whether an infix expression has balanced parenthesis or not. **(8)**

Ans:

The algorithm to determine whether an infix expression has a balanced parenthesis or not.

```
Matching = TRUE
```

```
Clear the stack;
Read a symbol from input string
While not end of input string and matching do
                    {
if (symbol= '(' or symbol = '{' or symbol = '[' )
                  push (symbol, stack)
else (if symbol = ')' or symbol = '}' or symbol = ']'
)
                  if stack is empty
                        matching = FALSE
write ("more right parenthesis than left parentheses")
                  else
                  c=pop (stack)
                  match c and the input symbol
                  if not matched
                  {
                        matching = FALSE
            write ("error, mismatched parentheses")
                  }
            read the next input symbol
      }
      if stack is empty then
      write ( " parentheses are balanced properly")
      else
write ("more left parentheses then right parentheses")
```

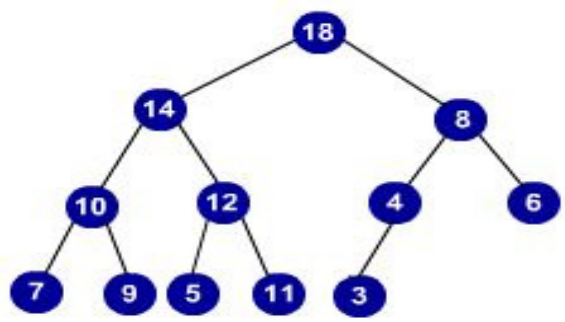
Q.134 How will you represent a max-heap sequentially? Explain with an example. Write an algorithm to insert an element to a max-heap that is represented sequentially. **(8)**

Ans:

Heap:

A binary heap(Minheap) is a complete binary tree in which each node other than root is smaller than its parent. A Maxheap is a complete binary tree in which each node other than root is bigger then its parent.

Heap example: (Maxheap)



Heap Representation:

- A Heap can be efficiently represented as an array
- The root is stored at the first place, i.e. a[1].
- The children of the node *i* are located at 2**i* and 2**i* +1.
- In other words, the parent of a node stored in *i* th location is at [i/2] floor.

- The array representation of a heap is given in the figure below.

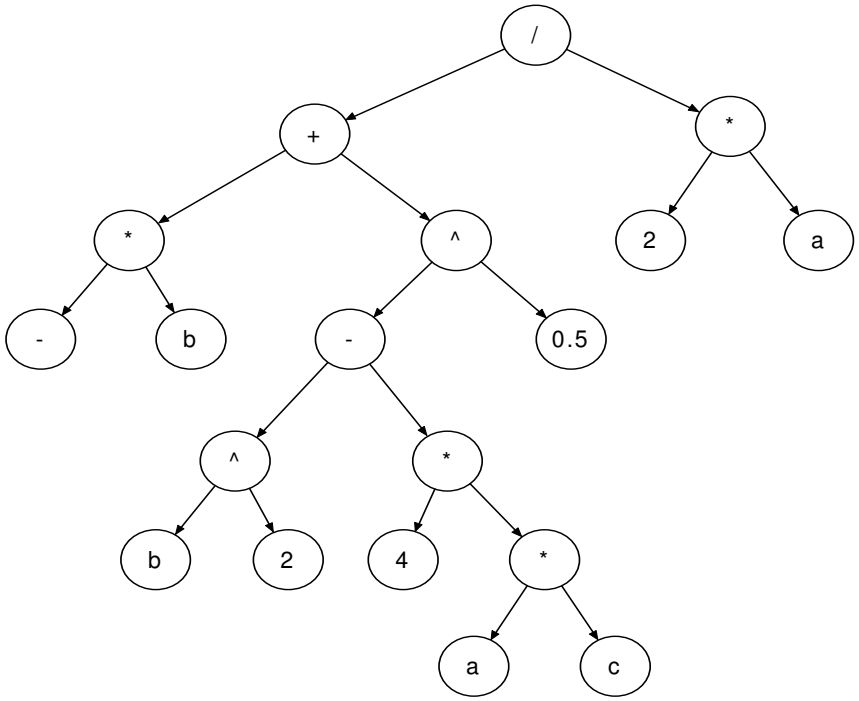
1	2	3	4	5	6	7	8	9	10	11	12
18	14	8	10	12	4	6	7	9	5	11	3

This is how max-heap is represented sequentially. An algorithm to create a heap of size i by inserting a key to a heap of size i-1 where i>=1 is as follows:
s=i;

```
/*find the   parent node of i in the array */
parent = s div 2;
key[s] = newkey;
while (s <> 0 && key [parent] <= key [s])
{
 /* interchange parent and child */
 temp = key [parent];
 key [parent] = key [s];
 key [s] = temp;
 /* advance  one level up in the tree */
 s = parent;
 parent = s div 2;
}
```

Q.135 Construct an expression tree for the expression $\left(-b+\sqrt{b^2-4ac}\right)\Big/2a$. Show all the steps and give pre-order, in-order and post-order traversals of the expression tree so formed. **(8)**

Ans :
The expression tree for the given formula is as follows



The Pre-order traversal of the tree is
/ + * - b ^ - ^ b 2 * 4 * a c 0.5 * 2 a
The Inorder traversal of the tree is
- * b + b ^ 2 - 4 * a * c ^ 0.5 / 2 * a
The Post-order traversal of the tree is
- b * b 2 * 4 a c * * / 0.5 ^ + 2 a * /

Q.136 Write an algorithm to find solution to the Towers of Hanoi problem. Explain the working of your algorithm (with 3 disks) with diagrams. **(10)**

Ans:

The aim of the tower of Hanoi problem is to move the initial n different sized disks from needle A to needle C using a temporary needle B. The rule is that no larger disk is to be placed above the smaller disk in any of the needle while moving or at any time, and only the top of the disk is to be moved at a time from any needle to any needle.

We could formulate a recursive algorithm to solve the problem of Hanoi to move n disks from A to C using B as auxiliary.

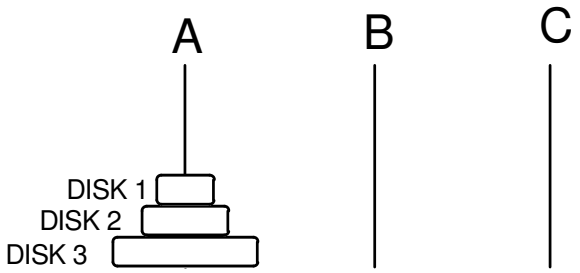
Step1: If n=1, move the single disk from A to C and return,
Step2: If n>1, move the top n-1 disks from A to B using C as temporary.
Step3: Move the remaining disk from A to C.
Step4: Move the n-1 disk disks from B to C, using A as temporary.
If we inplement this algorithm in a C language program, it would be like--

```
void hanoi ( int n, char initial, char final, char temp)
      {
            if ( n==1 )
            {
            printf (" move disk 1 from needle %c to %c\n",
intial,final);
                  return;
            }
            hanoi( n-1, initial ,temp , final);
            printf (“” nove diak %d from %c to %c “, n,
initial,final);
            hanoi(n-1, temp, final, initial);
      }
      void main()
      {
            int n;
            printf (“enter no of disks = “);
            scanf (“%d”,&n);
            hanoi ( n, ‘A’, ‘B’, ‘C’);
}
```

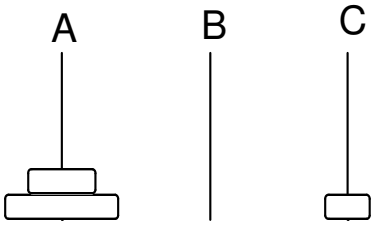
Now let us workout the output when n=3

The output will be as follows:-
Move disk 1 from needle A to needle C
Move disk 2 from needle A to needle B
Move disk 1 from needle C to needle B
Move disk 3 from needle A to needle C
Move disk 1 from needle B to needle A
Move disk 2 from needle B to needle C
Move disk 1 from needle A to needle C

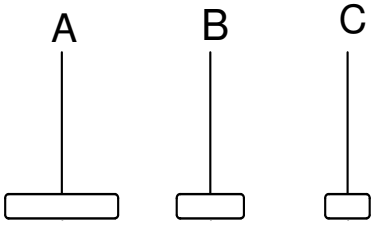
Initially the disk is at needle A



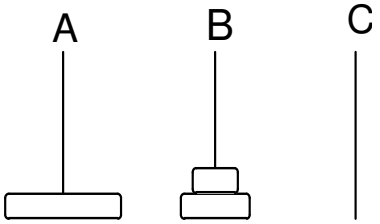
Move disk 1 from needle A to needle C



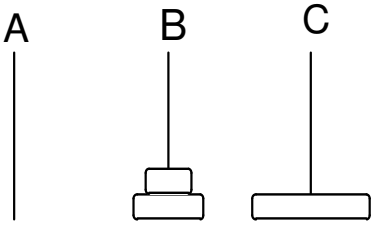
Move disk 2 from needle A to needle B



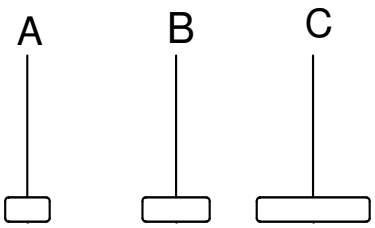
Move disk 1 from needle C to needle B



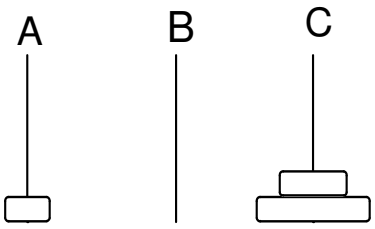
Move disk 3 from needle A to needle C



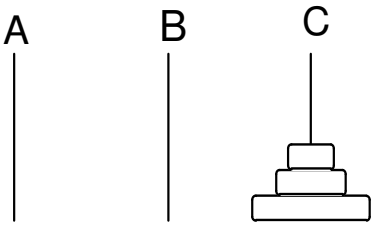
Move disk 1 from needle B to needle A



Move disk 2 from needle B to needle C



Move disk 1 from needle A to needle C



Thus finally all the three disks is in the required position after the completion of the algorithm for n=3 as shown above.

Q.137 What is recursion? A recursive procedure should have two properties. What are they? What type of recursive procedure can be converted in to iterative procedure without using stack? Explain with example. **(6)**

Ans:

Recursion: - Recursion is defined as a technique of defining a set or a process in terms of itself.

There are two important conditions that must be satisfied by any recursive procedure. They are: -

1. A smallest, **base case** that is processed without recursion and acts as decision criterion for stopping the process of computation and
2. A general method that makes a particular case to reach nearer in some sense to the base case.

For eg the problem of finding a factorial of a given number by recursion can be written as

```
Factorial (int n)
     {
int x;
if n = 0
       return (1);
x = n – 1;
return ( n* factorial (x) );
       }
```

in this case the contents of stack for n and x as [n, x] when first pop operation is carried out is as follows

[4, 3] [3, 2] [2, 1] [1, 0] [0, -] →top of stack

Here the relation is simple enough to be represented in an iterative loop, and moreover it can be done without taking the help of stack. So such kind of recursive procedures can always be converted into an iterative procedure.

The iterative solution for the above problem would be

```
Factorial (int n)
{
       int x, fact = 1;
       for ( x = 1 ;  x <= n  ;  x++ )
              fact = fact * x;
       return fact;
}
```

If there are no statements after the recursion call, that recursion can be converted to iterative programme very easily.for example:

```
   void fff (parameters)
            {
             if (condition)
            {
              step1
              step2
              step3
               .
               .
             fff (parameter)
              }
             }
```

whereas, if there are statements after a recursive call, a user stack will be required to convert them into iterative programme. for example:

```
        void fff (parameters)
              {
               if (condition)
              {
                step1
                step2
                fff (parameters)
                step3
                step4
                fff (parameter)
               }
             }
```

Q.138 Give an O(n) time non-recursive procedure that reverses a singly linked list of n nodes. The procedure should not use more than constant storage beyond that needed for the list itself. **(8)**

Ans:

The O (n) time non-recursive procedure that reverses a singly linked list of n nodes is as follows.

```
reverse (struct node **st)
{
struct node *p, *q, *r;
p = *st;
q = NULL;
while (p != NULL)
{
```

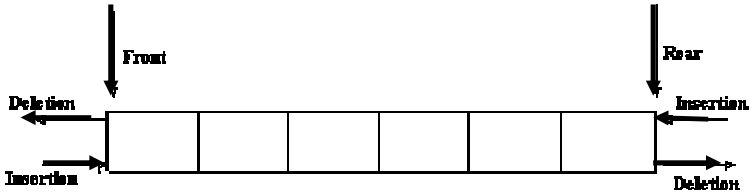
```
            r =q;
            q = p;
            p = p → link;
            q → link = r;
        }
        *st = q;
      }
```

Q.139 Device a representation for a list where insertions and deletions can be made at either end. Such a structure is called a Deque (Double Ended Queue). Write functions for inserting and deleting at either end. **(8)**

Ans :

Dequeue

A Dequeue can be represented as follows



Let the total size of elements be n in this Dequeue represented as an array
int array[n];

```
Then
front = index of the first element
Rear = index of the last element
If front = rear then we suppose that the Dequeue is empty
```

The Dequeue grows from front side till the index of the front becomes 0 and from rear side till the index of the rear equals the array size (n-1).

Now the functions for inserting at the front end of the Dequeue is:-

```
Insert_front ( int data )
{
        if ( front == 0 )
        printf ("Dequeue is full from the front end");
        else
        {
                front--;
                array[front] = data;
        }
   }
 Inserting from the rear end is:-
insert_rear ( int data )
{
        if (rear == n-1)
                printf("Dequeue is full from rear end");
        else
        {
                rear++;
                array [ rear ] = data;
        }
      }
Deleting from the front end: -
 int delete_front ()
 {
       if (front == rear)
              printf ( "the Dequeue is empty");
```

```
                else
                return ( array [ front++ ] );
 }
Deleting from the rear end:-
 int delete_rear ()
 {
        if ( front == rear )
                printf(" the Dequeue is empty ");
        else
                return ( array [ rear-- ] );
}
```

Q.140 The height of a tree is the length of the longest path in the tree from root to any leaf. Write an algorithmic function, which takes the value of a pointer to the root of the tree, then computes and prints the height of the tree and a path of that length. **(8)**

Ans

```
height(Left,Right,Root,height)
1. If Root=Null then height=0;
       return;
2. height(Left,Right,Left(Root),heightL)
3. height(Left,Right,Right(Root),heightR)
4. If heightL≥heightR then
            height=heightL+1;
      Else
            height=heightR+1;
5.  Return
```

Q.141 Two Binary trees are similar if they are both empty or if they are both non-empty and left and right subtrees are similar. Write an algorithm to determine if two binary trees are similar. **(8)**

Ans :

We assume two trees as tree1 and tree2. The algorithm two determine if two Binary trees are similar or not is as follows:

```
similar(*tree1,*tree2)
{
        while ((tree1!=null) && (tree2!=null))
        {
                if ((tree1->root) == (tree2->root))
                        similar (tree1->left,tree2->left);
                        similar (tree1->right,tree2->right);
        }
}
```

Q.142 Write an algorithm for Binary search. What are the conditions under which sequential search of a list is preferred over binary search? **(7)**

Ans :

Algorithm for Binary Search:-

Assuming that a [] is the array of items to be searched, n is the number of items in the array a, and *target* is the value of the item that is to be searched.

```
int binsearch( int target, int a[], int n)
    {
    int low=0;
    int high=1;
    int mid;
    while (low<=high)
    {
```

```
                mid=(low+high)/2;
                if (target==a[mid])
                        return (mid);
                if (target<a[mid])
                        high=mid-1;
                else
                        low=mid+1;
        }
        return (-1);
}
```

Sequential search is preferred over binary search in the following conditions:-
i). If the list is short sequential search is easy to write and efficient than binary search.
ii) If the list is unsorted then binary search cannot be used, in that case we have to use sequential search.
iii) If the list is unordered and haphazardly constructed, the linear search may be the only way to find anything in it.

Q.143 Work through Binary Search algorithm on an ordered file with the following keys {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15}. Determine the number of key comparisons made while searching for keys 2, 10 and 15. **(9)**

Ans:
The given list is 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15.
where n=14
initially low=0, high=14, mid=(low+high)/2=7

Searching for 2

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(low) (mid) (high)

Comparison 1: 2< a[mid]
so high=mid-1=6 and mid=(0+6)/2=3

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(low) (mid) (high)

Comparison 2: 2< a[mid]
so high=mid-1=2 and mid=(0+2)/2=1

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(low)(mid)(high)

comparison 3: 2=a[mid]
So by the **third** comparison, we find the element in the list.

Searching for 10

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(low) (mid) (high)

Comparison 1: 10 > a[mid]
so low=mid+1=8 and mid=(8+14)/2=11

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(low) (mid) (high)

Comparison 2: 10< a[mid]
so high=mid-1=10 and mid=(8+10)/2=9

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(low)(mid)(high)

comparison 3: 10=a[mid]
So by the **third** comparison, we find the element in the list.

Searching for 15

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(low) (mid) (high)

Comparison 1: 15 > a[mid]
so low=mid+1=8 and mid=(8+14)/2=11

1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(low) (mid) (high)

Comparison 2: 15> a[mid]
so low=mid+1=12 and mid=(12+14)/2=13

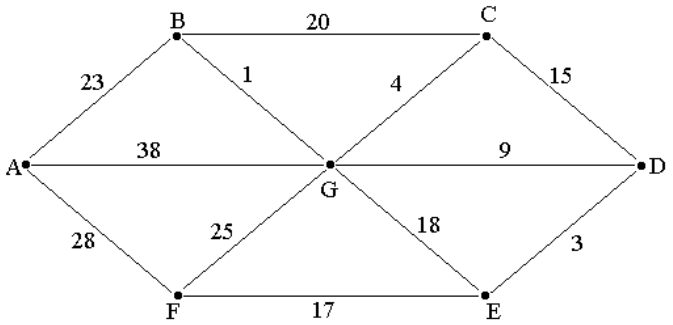
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
(low)(mid)(high)

comparison 3: 15>a[mid]
so low=mid+1=14 and mid=(14+14)/2=14

comparison 4: 15=a[mid]

So by the **fourth** comparison, we find the element in the list.

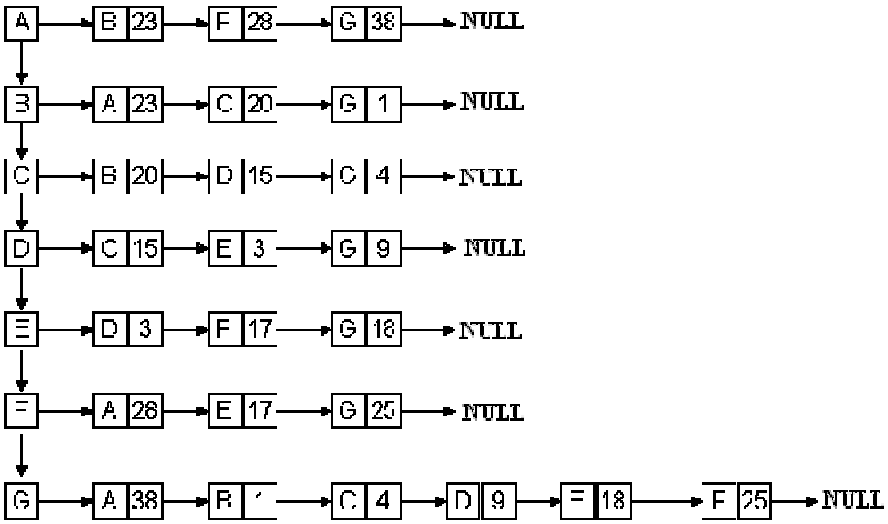
Q.144 Consider the following undirected graph:



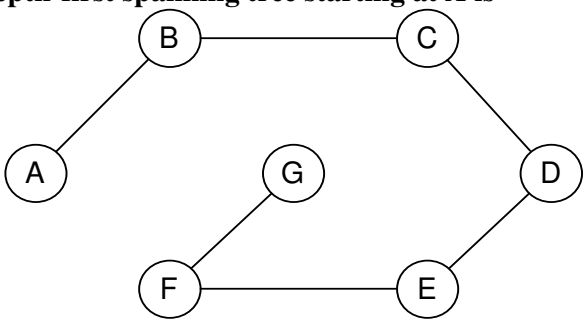
(i) Find the adjacency list representation of the graph.
(ii) Find a depth-first spanning tree starting at A.
(iii) Find a breadth-first spanning tree starting at A.
(iv) Find a minimum cost spanning tree by Kruskal's algorithm.**(4 x 3 = 12)**

Ans :

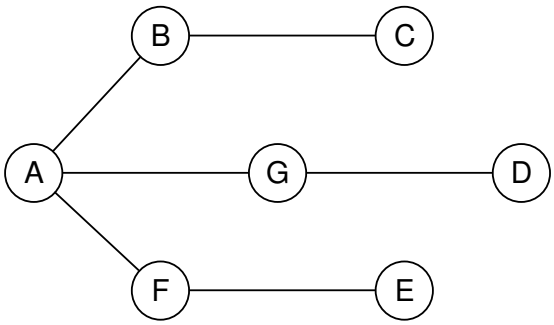
(i) The adjacency list representation of the above graph is



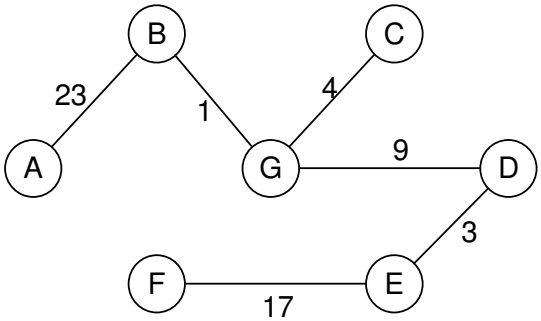
(ii) The depth-first spanning tree starting at A is



(iii) The breadth-first spanning tree starting at A is



(iv) The minimum cost spanning tree using Kruskal's Algo is:-



Q.145 Suppose that a graph has a minimum spanning tree already computed. How quickly can the minimum spanning tree be updated if a new vertex and incident edges are added to G? **(4)**

Ans : If the new vertex and the new edges are not forming a cycle on the MST, pick up the smallest edge from the set of new edges. If the new vertex and the corresponding edges are forming cycle on the MST break the cycle by removing the edge with largest weight. This will update the minimum spanning tree.

Q.146 Write code to implement a queue using arrays in which insertions and deletions can be made from either end of the structure (such a queue is called deque). Your code should be able to perform the following operations

(i) Insert element in a deque
(ii) Remove element from a deque
(iii) Check if deque is empty
(iv) Check if it is full

(v) Initialize it **(12)**

Ans : Here there are 4 operations

1. q insert front
2. q insert rear
3. q delete front
4. q delete rear

when front = rear = -1, queue is empty when (rear+1) mod size of queue = front queue is full. Here the queue is to be used circularly.

(1) q insert front

```
if (front==rear==-1)
then front=rear=0
q (front)=item
else if(rear+1)mod sizeof q=front
q is full, insertion not possible
else front = (front + sizeof q-1) mod sizeof q
```

(2) q insert rear

```
if (front==rear==-1)
front=rear=0
q(rear)=item
else if (rear+1) mod sizeof queue + front
queue is full, insertion not possible
else rear = (rear+1) mode sizeof queue
q(rear) = item
```

(3) q delete front

```
if (front==rear==-1)
queue is empty
else if (front==rear)
k = q(front), front = rear= -1
return(k)
else k=q(front)
front = (front+1) mod sizeof queue
return (k)
```

(4) q delete rear

```
if (front==rear==-1)
queue is empty
else if (front==rear)
k = q(rear), front=rear=-1
return(k)
else
k=q(rear)
rear = (rear +size of queue -1) mod sizeof q
return (k)
```

Q.147 What are stacks? List 6 different applications of stacks in a computer system. **(4)**

Ans :

A **stack** is an abstract data type in which items are added to and removed only from one end called TOP. For example, consider the pile of papers on your desk. Suppose you add papers only to the top of the pile or remove them only from the top of the pile. At any point in time, the only paper that is visible is the one on the top. This structure is called *stack*. The six various application of **stack in computer application are:**

1. Conversion of infix to postfix notation and vice versa.
2. Evaluation of arithmetic expression.

3. Problems involving recursion are solved efficiently by making use of stack. For example-Tower of Hanoi problem
4. Stack is used to maintain the return address whenever control passes from one point to another in a program.
5. To check if an expression has balanced number of parenthesis or not.
6. To reverse string.

Q.148 Write a recursive code to compute the sum of squares as shown in the series $m^2 + (m+1)^2 + \ldots + n^2$ for m, n integers $1 \le m \le n$ **(8)**

Ans :

A recursive code to compute sum of squares from m to n.

Make int n a global variable so that it is not passed in every recursive cell.

```
int  n;
void main()
{
    int m,sum;
    clrscr();
    printf (" Value of m : ");
    scanf ("%d",&m);
    printf (" Value of n : ");
    scanf ("%d",&n);
    sum=sum_recursive(m);
    printf ("\n\n\n\n%d",sum);
}
int sum_recursive(int m)
{
    int sum;
    if (m == n)
          return (n*n);
    else
          sum = (m*m) + sum_recursive(m+1);
    printf ("\t%d",sum);
    return sum;
}
```

Q.149 Give mathematical recursive definition of an AVL tree. **(4)**

Ans :

AVL tree definition:

1. An empty binary tree a is an AVL tree
2. If 'B_t' is the non-empty binary tree with 'B_{tL}' and 'B_{tR}' as its left and right-subtrees, then 'B_t' is an AVL tree if only if:

- 'B_{tL}' and 'B_{tR}' are AVL trees and
- $|h_L - h_R| \le 1$ where h_L and h_R are the heights of 'B_{tL}' and 'B_{tR}' respectively. $|h_L - h_R|$ is called balance factor of a node, its value can be {-1, 0, 1}

Q.150 Given the following recursive code. Point out the possible errors in the code.

```
//num is a positive integer
int fac(int num)
{
if (num≤1)
      return 1;
else {
      return num*fac(num+1);
        }
```

```
}
```
(4)

Ans : There is no misplaced else.Two round brackets are missing for the return statements.it should have been

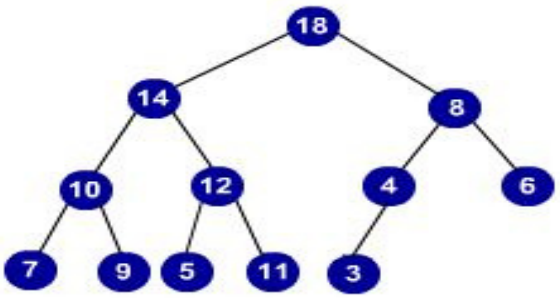
```
return (1)
return (num* fac (num-1))
```

Note :- it is not (num+1) but (num-1)

Q.151 What are the properties of a *heap*? Distinguish between a *max heap* and a *min heap*.)

Ans :

A complete binary tree, each of whose elements contains a value that is greater than or equal to the value of each of its children is called a Heap

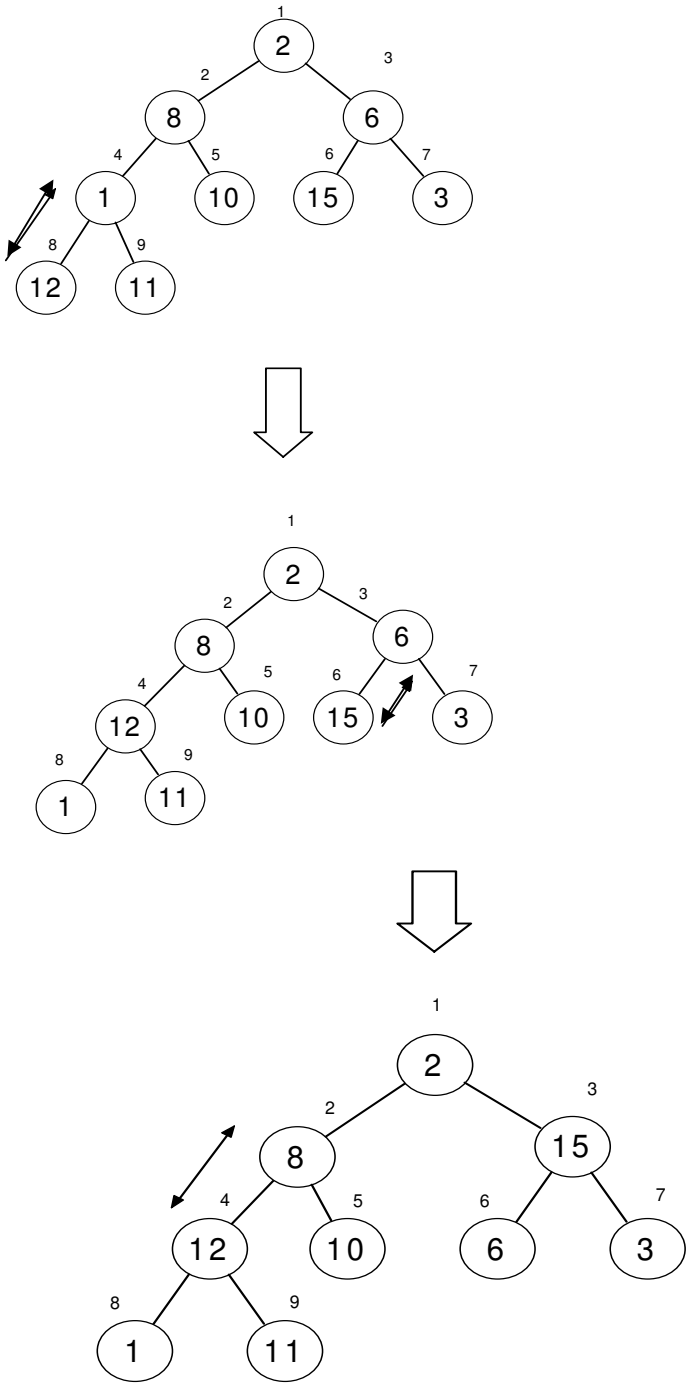


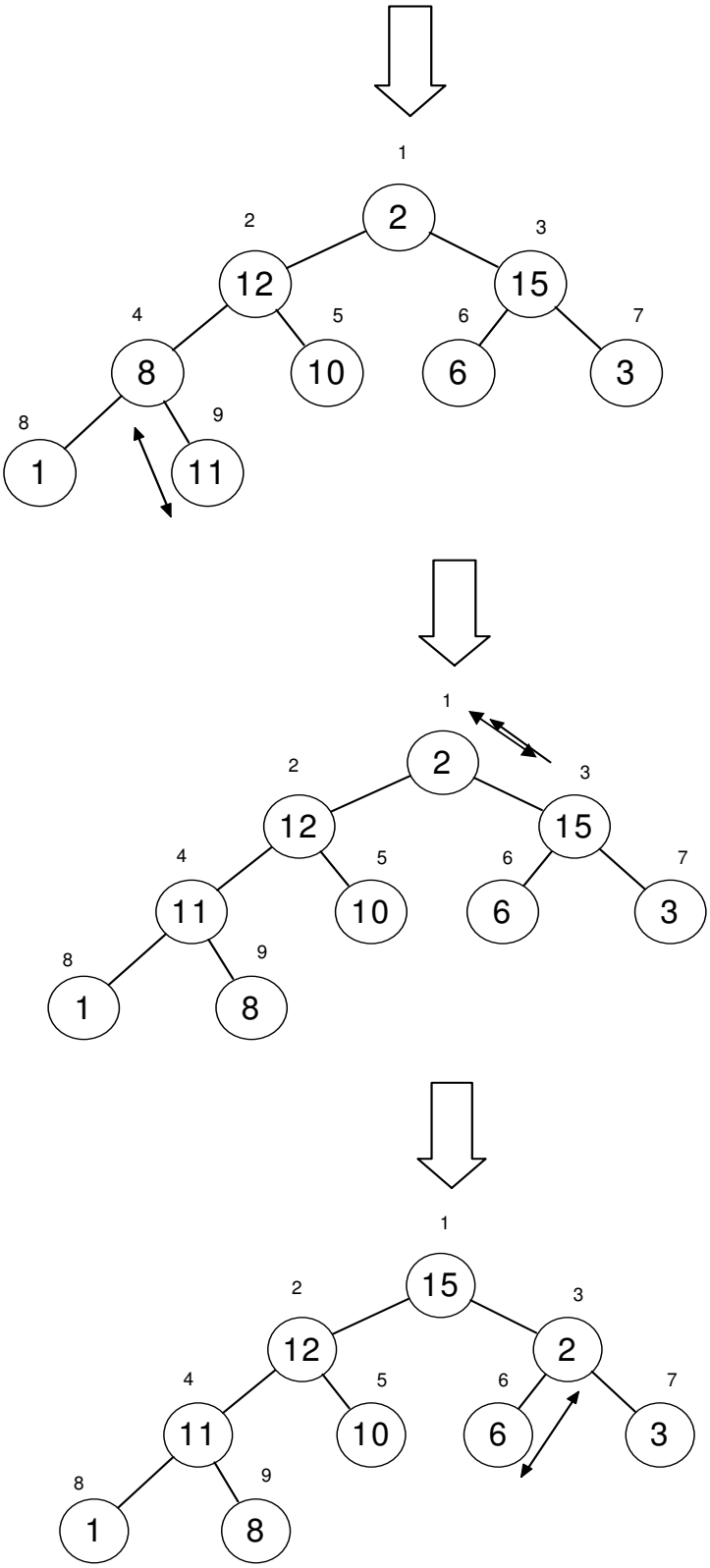
For example, above is a heap of size 12. Because of the order property of heap, the root node always contains the largest value in the heap. When we refer to such a heap, it is called a "maximum heap (*max heap*)," because the root node contains the maximum value in the structure. Similarly we can also create a "minimum heap," each of whose elements contains a value that is *less* than or equal to the value of each of its children

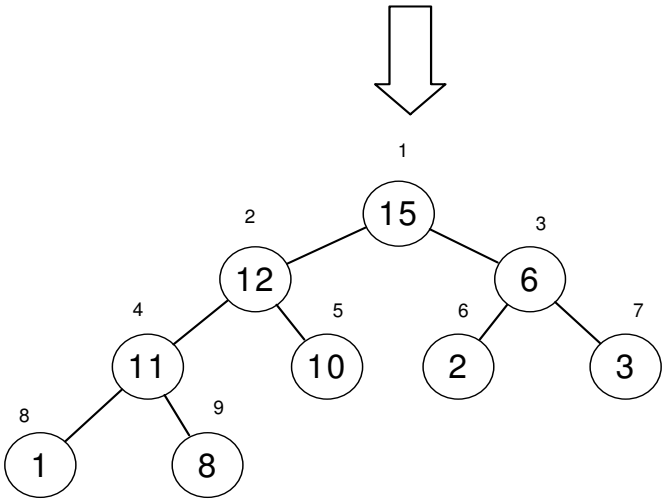
Q.152 Transform the array 2 8 6 1 10 15 3 12 11 into a heap with a bottom up method **(8)**

Ans :

Given array is 2, 8, 6, 1, 10, 15, 3, 12, 11


1
2
2
8
3
6
4
1
5
10
6
15
7
3
8
12
9
11
1
2
2
8
3
6
4
12
5
10
6
15
7
3
8
1
9
11
1
2
2
8
3
15
4
12
5
10
6
6
7
3
8
1
9
11


1
2
2
12
3
15
4
8
5
10
6
6
7
3
8
1
9
11
1
2
2
12
3
15
4
11
5
10
6
6
7
3
8
1
9
8
1
15
2
12
3
2
4
11
5
10
6
6
7
3
8
1
9
8



Final heap

Q.153 What are threaded binary trees? Explain inorder threading using an example. **(4)**

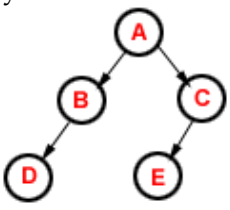
Ans :

Threaded Binary Tree: If a node in a binary tree is not having left or right child or it is a leaf node then that absence of child node is represented by the null pointers. The space occupied by these null entries can be utilized to store some kind of valuable information. One possible way to utilize this space is to have special pointer that point to nodes higher in the tree that is ancestors. These special pointers are called threads and the binary tree having such pointers is called threaded binary tree. There are many ways to thread a binary tree each of these ways either correspond either in-order or pre-order traversal of T.A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its INORDER successor. By doing this threading we avoid the recursive method of traversing a Tree, which makes use of stacks and consumes a lot of memory and time.

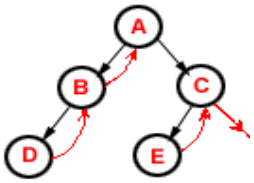
The node structure for a threaded binary tree varies a bit and its like this

```
struct NODE
{
 struct NODE *leftchild;
 int node_value;
 struct NODE *rightchild;
 struct NODE *thread;
}
```

Let's make the Threaded Binary tree out of a normal binary tree...



The INORDER traversal for the above tree is -- D B A E C. So, the respective **Threaded Binary tree will be --**



B has no right child and its inorder successor is A and so a thread has been made in between them. Similarly, for D and E. C has no right child but it has no inorder successor even, so it has a hanging thread.

Q.154 Write an algorithm to implement Depth-first search? How is Depth-first search different from Breadth-first search? **(10)**

Ans :

Depth First Search Algorithm

An algorithm that does depth first search is given below:

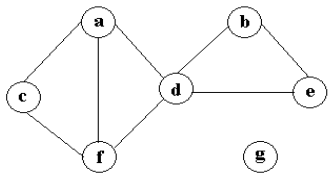
```
struct node
{
  int data ;
  struct node *next ;
} ;
int visited[MAX] ;
void dfs ( int v, struct node **p )
{
  struct node *q ;
  visited[v - 1] = TRUE ;
  printf ( "%d\t", v ) ;
  q = * ( p + v - 1 ) ;
  while ( q != NULL )
  {
        if ( visited[q -> data - 1] == FALSE )
              dfs ( q -> data, p ) ;
        else
              q = q -> next ;
  }
}
```

Depth-first search is different from Breadth-first search in the following ways:

A depth search traversal technique goes to the deepest level of the tree first and then works up while a breadth-first search looks at all possible paths at the same depth before it goes to a deeper level. When we come to a dead end in a depth-first search, we back up as *little* as possible. We try another route from a recent vertex-the route on top of our stack. In a breadth-first search, we want to back up as *far* as possible to find a route originating from the earliest vertices. So the stack is not an appropriate structure for finding an early route because it keeps track of things in the order opposite of their occurrence-the latest route is on top. To keep track of things in the order in which they happened, we use a FIFO queue. The route at the front of the queue is a route from an earlier vertex; the route at the back of the queue is from a later vertex.

Q.155 Represent the following graph as

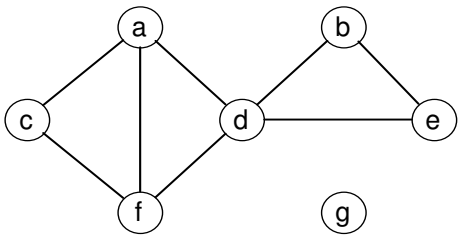
(i) Adjacency list
(ii) Adjacency matrix
(iii) Incidence matrix



(6)

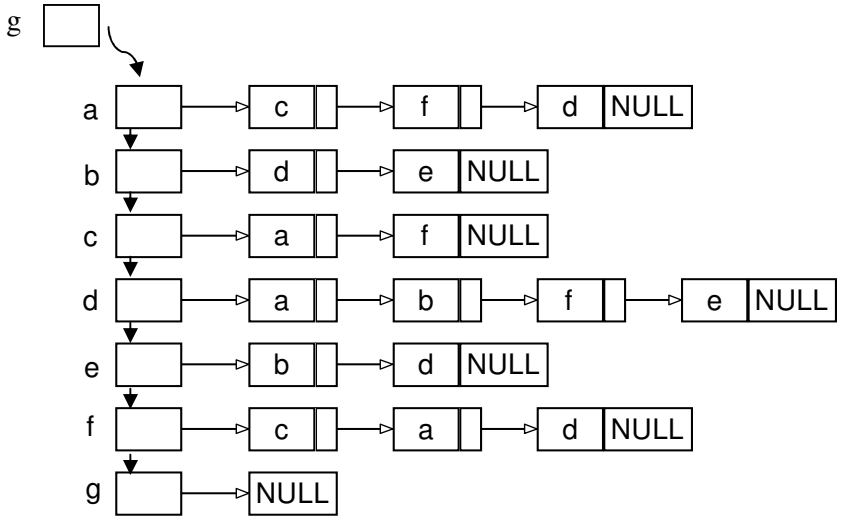
Ans :

The given graph is –



The representation of the above graph is as follows: -

(i) Adjacency List : -



(ii) Adjacency Matrix : -

	a	b	c	d	e	f	g
a			1	1		1	
b				1	1		
c	1					1	
d	1	1			1	1	
e		1		1			
f	1		1	1			
g							

(iii) **Incidence Matrix** : - This is the incidence matrix for an undirected group. For directed graphs, the vertex from where an edge is originating will have +1 and the vertex where the edge is reaching will have a value -1.

	C1	C2	C3	C4	C5	C6	C7	C8
a	1	1	1					
b				1	1			
c	1					1		
d		1		1			1	1
e					1		1	
f			1			1		1
g								

Rest of the entries are zero.

Q.156 Define B-tree of order m? When is it preferred to use B-trees? **(4)**

Ans :

A B-Tree of order M is either the empty tree or it is an M-way search tree T with the following properties:

(i) The root of T has at least two subtrees and at most M subtrees.

(ii) All internal nodes of T (other than its root) have between [M / 2] and M subtrees.

(iii)All external nodes of T are at the same level.

Just as AVL trees are balanced binary search trees, B-trees are balanced M-way search trees. By imposing a balance condition, the shape of an AVL tree is constrained in a way which guarantees that the search, insertion, and withdrawal operations are all O(log n), where n is the number of items in the tree. The shapes of B-Trees are constrained for the same reasons and with the same effect.

Q.157 Write an algorithm to search a key in a B-tree? What is the worst case of searching in a B-tree? List the possible situations that can occur while inserting a key in a B-tree? **(8)**

Ans :

B-tree search makes an n-way choice. By performing the linear search in a node, correct child Ci of that node is chosen. Once the value is greater than or equal to the desired value is obtained, the child pointer to the immediate left of the value is followed. Otherwise rightmost child pointer is followed. If desired value is obtained, the search is terminated.

B-tree search (x, k) /*This function searches the key from the given B-tree*/

The Disk_Read operation indicates that all references to a given node be preceded by a read operation.

1. Set i ← 1
2. While ($i < n[x]$ and $k > key_i [x]$)
   Set i = i + 1
3. If ($i <= n[x]$ and $k = key_i [x]$) then
   { return (x, i) }
4. If (leaf [x]) then
   return NIL
   else Disk_read ($c_i [x]$)
   return B-tree search (ci [x], k)

The worst case of searching is when a B-tree has the smallest allowable number of pointers per non-root made, q=[m/2], the search has to reach a leaf (either for a successful or an unsuccessful search)

There are three common similarities that are encountered when inserting a key in a B-tree.

1) A key is placed in a leaf that still has some room.
2) The leaf in which the key should be placed is full. In this case, the leaf is split, creating a new leaf and half of the keys are moved from the full leaf to the new leaf. But the new leaf has to be incorporated in to the B-tree. The last key of the old leaf is moved to the parent and a pointer to the new leaf is placed in the parent as well. The same procedure can be repeated for each internal node of the B-tree. Such a split ensures that each leaf never has less than [m/2]-1 keys.
3) A special case arises if the root of the B-tree is full. In this case, a new root and a new sibling of the existing root has to be created. This split results in two new nodes in the B-tree.

Q.158 What is the complexity of the following code?

```
int counter = 0;
for (i=0; i<n; i++)
        for (j=0; j<n*n; j++)
                counter++;
```

(4)

Ans :

The complexity of given code is $O(n^2)$.

Q.159 Show under what order of input, the insertion sort will have worst-case and best-case situations for sorting the set {142, 543, 123, 65, 453, 879, 572, 434}. **(8)**

Ans :

The given list is

142 543 123 65 453 879 572

As we know that insertion sort is based on the idea of inserting records into an existing sorted file. To insert a record, we must find the proper place for the record to be inserted and this requires searching the list.

An insertion sort will have the worst-case when the list is reversely sorted i.e. in a decreasing order. So for the above given list the insertion sort exhibits its **worse case** when the list is in following order

879 572 543 453 434 142 123 65

Again since it is the property of the insertion sort that it searches for the correct position in the list for an element, so this technique exhibits its **best case** computing when the list is already sorted in ascending order as given below.

65 123 142 434 453 543 572 879

Q.160 Explain how Merge Sort sorts the following sequence of numbers using a diagram {142, 543, 123, 65, 453, 879, 572, 434}. **(8)**

Ans :

Sorting using Merge Sort

The given sequence of numbers is:-

Original Sequence [434]	[142]	[543]	[123]	[65]	[453]	[879]	[572]
Pass one [572	[142	543]	[65	123]	[453	879]	[434
Pass two 879]	[65	123	142	543]	[434	453	572
Pass three [879	[65	123	142	434	453	543	572

This number sequence is sorted in three passes using merge sort.

Q.161 Given a stack *s* and a queue *q*, show the contents of each after the indicated operations. The starting contents of *s* and *q* are shown. If an operation would result in an error, write "error" and assume the contents of *s* and *q* do not change. If there is no change otherwise, leave the cell blank or use ditto marks ("). **(8)**

Operation	**Contents of stacks Top bottom**	**Contents of queue q front rear**
Start	*empty*	2, 4
s.pop()		
s.push(3)		
q.add(5)		
s.push(q.peek())		
q.add(s.peek())		
q.add(s.pop())		
s.push(s.pop())		
q.add(q.remove())		

Ans :

pop(): Removing an element from top of the satck
push(element): inserting element on the top of stack
add(element): inserting element from rear of the queue
remove(): removing element from front of the queue
peek():reading from the front of the queue.

We have left the cells blank where there is no change from the previous state.

Operation	Contents of Stack Top bottom	Content of queue q Front rear
start	empty	2,4
s.pop()	error	-do-
s.push(3)	3	-do-
q.add(5)	-do-	2,4,5
s.push(q.peek())	2,3	-do-
q.add(s.peek())	-do-	2,4,5,2
q.add(s.pop())	3	2,4,5,2,2
s.push(s.pop())	-do-	-do-
q.add(q.remove())	-do-	4,5,2,2,2

Q.162 If x is a pointer to a node in a doubly linked list, then x-> prev is the pointer to the node before it and x->next is the pointer to the node after it.
What does this pair of statements, executed in order, do?

```
(x->next)->prev = x->prev;
(x->prev)->next = x->next;
```

If we then execute this pair of statements (after executing the first pair of statements), what happens?

```
(x->next)->prev = x;
(x->prev)->next = x;
```

(4)

Ans :

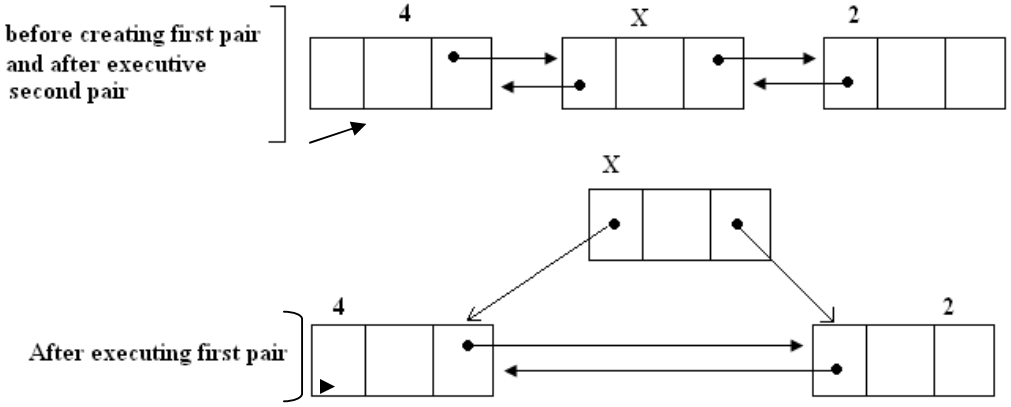
In the above given condition, if the following

```
(x->next)->prev=x->prev;
(x->prev)-next=x->next;
```

statements are executed then the resulting linked list will no longer have x as one of its elements i.e. the element x gets deleted from the doubly linked list.
Now, after executing the above pair of statements; if we execute the following pair i.e.-

```
(x->next)->prev=x;
(x->prev)->next=x;
```

then this causes the element x to become a part of the linked list again(i.e. it gets inserted again) and at the same position as before.



Q.163 Consider an insertion of the key = 222 into the hash table shown in the figure. Calculate and indicate the sequence of table entries that would be probed and the final location of the insertion for the key for linear probing and quadratic probing method. The hash function is h(y) = y mod 10, so for the key = 222 , h(x) = 2 mod 10. In both cases start with the same initial table. Example calculation given at index 2 **(8)**

0		
1		
2	152	h(222) = 2 mod 10
3	53	
4		
5	75	
6	136	
7	27	
8		
9	999	

Ans :

Linear Probing:

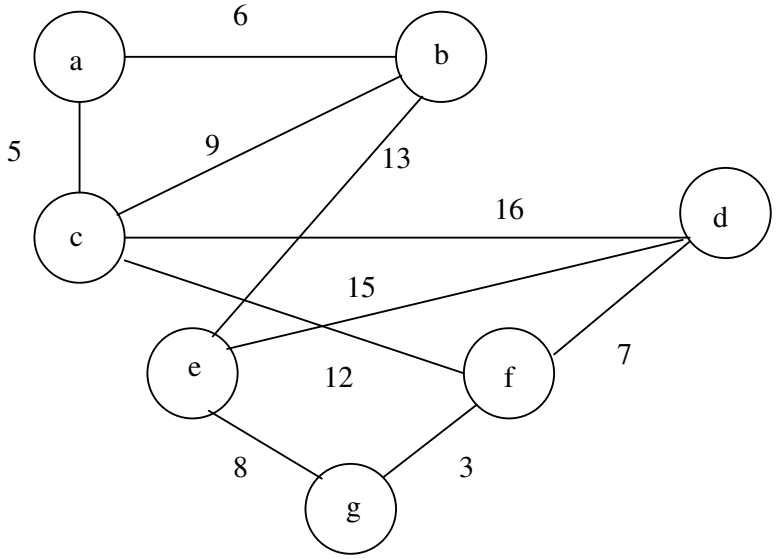
i. Initially all locations marked 'empty'
ii. When new items are stored, filled locations marked 'full'
iii. In case of collisions, newcomers are stored at the next available location, found via probing by incrementing the pointer (mod n) until either an empty location is found or starting point is reached.
So in the given case we have 222 mod 10=2, index 2 is probed. Since it is already filled so newcomer will go to the next available location that is index 4.

Rehashing:

i. The collision key is move to the considerable distance from the initial collision when the hashing function compute the value then the quadratic probing work as follows
(Hash value+1^2) mod table size
ii. If there is a collision, take the second hash function as
(Hash value+2^2) mod table size and so on.
iii. The probability that two key values will map to the same address with two different hash functions is very low.

So in the given case, we see $222+1^2$ mod 10=3 there is collision; Use next hash function.
$222+2^2$ mod 10=222+4=226 mod 10=6 , there is collision
$222+3^2$ mod 10=222+9=231 mod 10=1 so newcomer can be inserted at index 1.

Q.164 Using Dijkstra's method find a spanning tree of the following graph. **(8)**



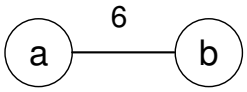
Ans :

Disjkstra algorithm Method is never used for finding Minimum-spanning tree
Minimum-spanning tree

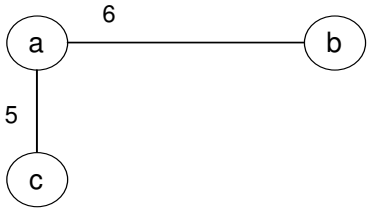
tree=null;
edges=an unsorted sequence of all edges of graph;
for j=1 to $|E|$
add e_j to tree;
If there is cycle in tree

Remove an edge with maximum weight from this only cycle.
The spanning tree for given graph is given below:

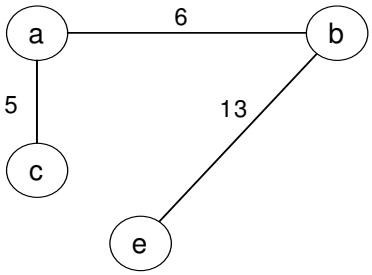
Step 1



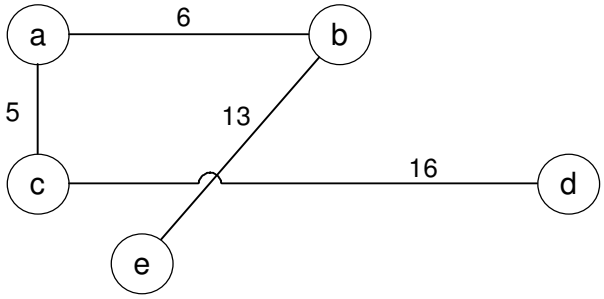
Step 2



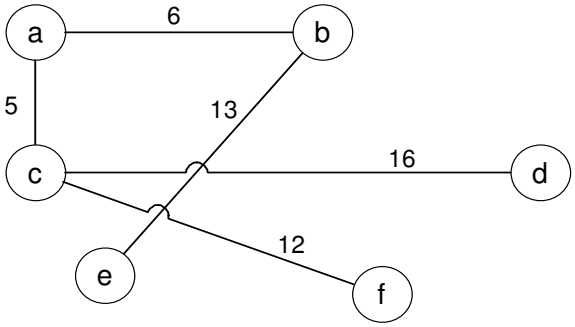
Step 3



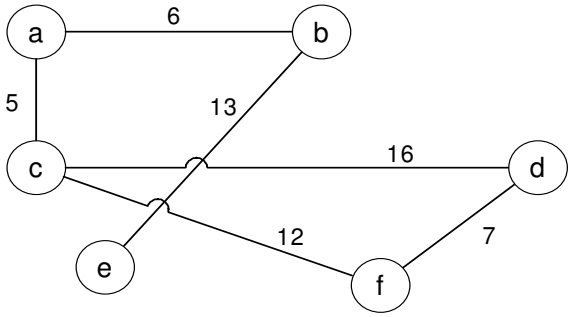
Step 4



Step 5

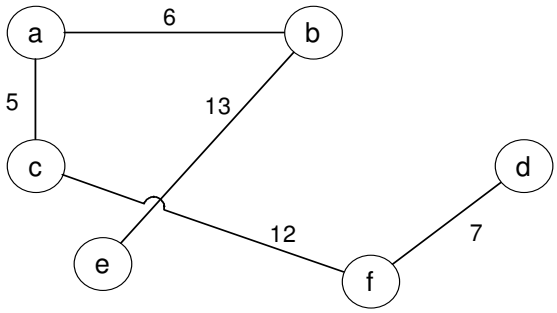


Step 6

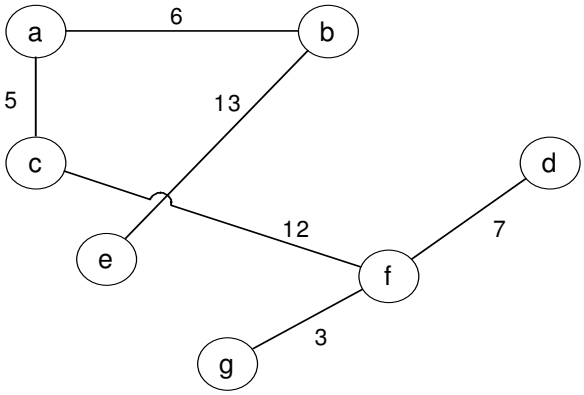


Since the above representation creates a cycle so we delete the edge having the highest weight in the cycle and the new representation would be as follows: -

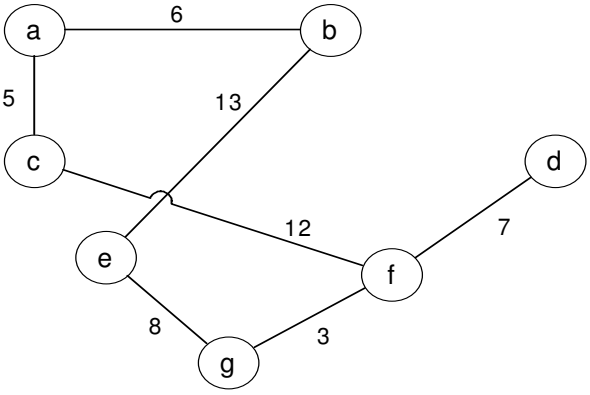
Step 7



Step 8

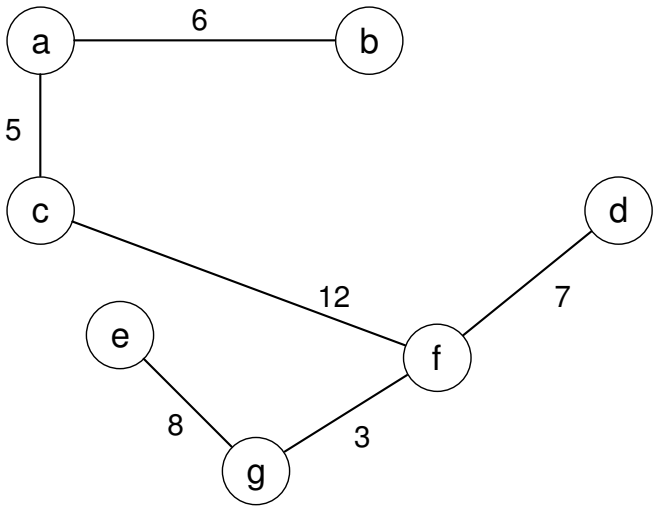


Step 9



The above representation creates cycle so we'll delete the edge having the highest weight as shown below.

Step 10



Q.165 What are the three different ways of representing a polynomial using arrays? Represent the following polynomials using any three different methods and compare their advantages and disadvantages.

(i) $7x^5 - 8x^4 + 5x^3 + x^2 + 2x + 15$

(ii) $3x^{100} - 5x^{55} - 10$ **(8)**

Ans :

A general polynomial A(x) can be written as $a_nx^n+a_{n-1}x^{n-1}+\ldots+a_1x+a_0$ where $a_n \neq 0$ then we can represent A(x)

1. As an ordered list of coefficients using a one dimensional array of length n+2, A=(n, a_n, a_{n-1},…,a_1, a_0) , the first element is the degree of A followed by n+1 coefficients in order of decreasing exponent. So the given polynomial can be represented as

i. (5,7, -8,5,1,2,15)

ii. (100,3,0,0,0,0…(45 times), -5, 0,0,0…(55 times),-10)

Advantage of this method is simple algorithms for addition and multiplication as we have avoided the need to explicitly store the exponent of each term. However there is a major disadvantage of this method and that is wastage of memory for certain polynomial like example (ii) because we are storing more zero values than non-zero values.

2. As an ordered list where we keep only non-zero coefficient along with their exponent like $(m,e_{m-1},b_{m-1},e_{m-2},b_{m-2},\ldots,e_0,b_0)$ where first entry is the number of nonzero terms, then for each term there are two entries representing an exponent-coefficient pair. So the given polynomial can be represented as

i. (6,5,7,4,-8,3,5,2,1,1,2,0,15)

ii. (3,100,3,55,-5,0,-10)

This method is solves our problem of wasted memory like what happened in example 2.

Q.166 Write an algorithm to evaluate the following polynomial. The number of additions and multiplications taken should be 5.

$9x^5 - 10x^4 + 6x^3 + 5x^2 - 10x + 15$ **(8)**

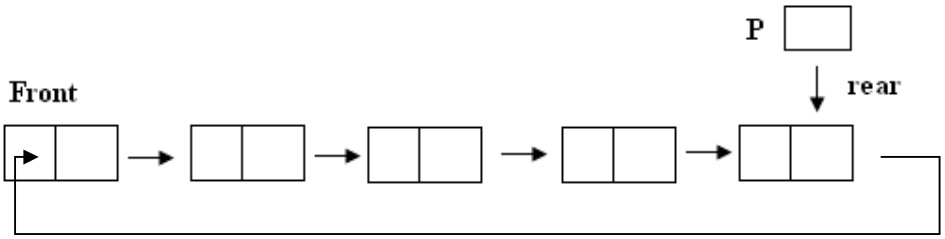
Ans :

The given polynomial can be evaluated using Hormer's rule as

$9x^5-10x^4+6x^3+5x^2-10x+15$
$\equiv x^4[9x-10]+6x^3+5x^2-10x+15$
$\equiv x^3[x[9x-10]+6x]+5x^2-10x+15$
$\equiv x^2[x[x[9x-10]+6]+5]-10x+15$
$\equiv x[x[x[9x-10]+6]+5]-10x+15$
$\equiv x[x[x[x][9x-10]+6]+5]-10]+15$
$\equiv 0[5]$

Q.167 Let P be a pointer to a circularly linked list. Show how this list may be used as a queue. That is, write algorithms to add and delete elements. Specify the value for P when the queue is empty. **(10)**

Ans : External pointer should always point to the last node of the circularly linked list. Then both the insertion and deletion from the queue can be done in O(1) time.



To add a new node x (insert a new element in the queue)

```
x →     next = P  →   next
P →     next = x
P=x
```

To delete a node (Deleting front element from queue)

```
x = P  →  next
P →    next = x →    next
return (x)
```

Q.168 Give an algorithm for a singly linked list, which reverses the direction of the links. **(6)**

Ans :

The algorithm to reverse the direction of a singly link list is as follows:

```
reverse (struct node **st)
{
      struct node *p, *q, *r;
      p = *st;
      q = NULL;
      while (p != NULL)
      {
        r =q;
        q = p;
        p = p → link;
        q → link = r;
      }
      *st = q;
}
```

Q.169 Suppose that we have numbers between 1 and 1000 in a Binary Search Tree and want to search for the number 363. Which of the following sequences could not be the sequence of nodes examined? Explain your answer.

(i) 2, 252, 401, 398, 330, 344, 397, 363
(ii) 924, 220, 911, 244, 898, 258, 362, 363
(iii) 925, 202, 911, 240, 912, 245, 363
(iv) 2, 399, 387, 219, 266, 382, 381, 278, 363
(v) 935, 278, 347, 621, 299, 392, 358, 363 **(10)**

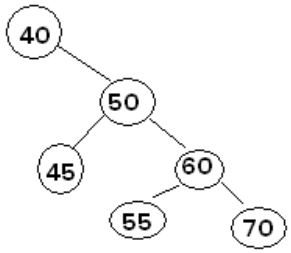
Ans :

(i) This is a valid representation of sequence.
(ii) This is also a valid representation of sequence.
(iii)This could not be the sequence since 240 is encountered after 911 so it must be its left child, and any other node henceforth must be less than 911(parent). But the node 912 breaks this sequence, which is greater than 911 and lies on the left subtree of 911, which violates the basic rule of a Binary Search Tree.
(iv)This is also a valid representation of sequence.
(v)This could not be a valid sequence since 621 is encountered after 347 so 621 must be its right child, and any other node henceforth must be greater than 347(parent). But this sequence is broken by the node 299 < 347 and lies on the right subtree of 347 which violates the basic rule of a Binary Search Tree.

Q.170 Professor Banyan thinks he has discovered a remarkable property of binary search trees. Suppose that the search for key k in a BST ends up in leaf. Consider three sets (1) set A, the keys to the left of the search path, (2) set B the keys on the search path and (3) set C, the keys on the right of the search path. Professor Banyan claims that any three keys $a \in A$, $b \in B$ and $c \in C$ must satisy $a \le b \le c$. Is his claim true? Otherwise give a counter example to the professor's claim. **(3)**

Ans :

Professor Banyan claim is completely wrong.
The claim of professor Banyan is wrong. Counter example is given below.

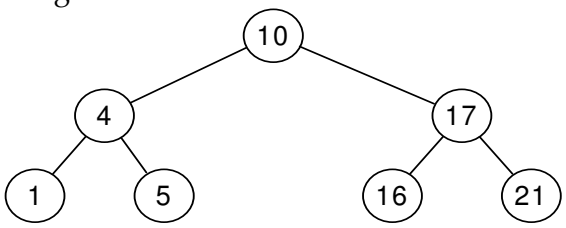


Let the key to be searched is 70 then:
Set A = {45, 55}, B = {40, 50, 60, 70} and C = { }
Here is very clear that $45 \nleq 40$ OR $55 \nleq 50$

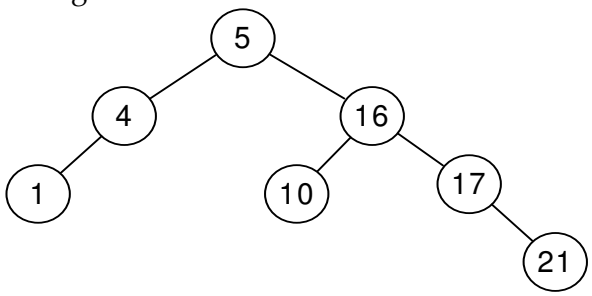
Q.171 Draw BSTs of height 2 and 3 on the set of keys {1, 4, 5, 10, 16, 17, 21}. **(3)**

Ans :

The BST of height 2 is:



The BST of height 3 is:



Q.172 Why don't we allow a minimum degree of t=1 for a B-tree? **(2)**

Ans :

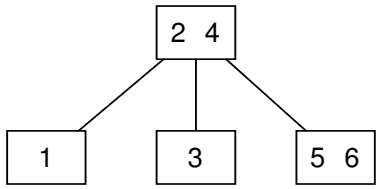
According to the definition of B-Tree, a B-Tree of order n means that each node in the tree has a maximum of n-1 keys and n subtrees. Thus a B-Tree of order 1 will have maximum 0 keys, i.e. no node can have any keys so such a tree is not possible and hence we can't allow a minimum degree of t=1 for a B-Tree.

Q.173 Show all legal B-Trees of minimum degree 3 that represent {1, 2, 3, 4, 5, 6}. **(4)**

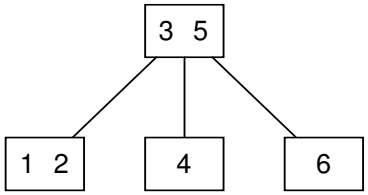
Ans :

B-Trees of minimum degree 3 are as follows:

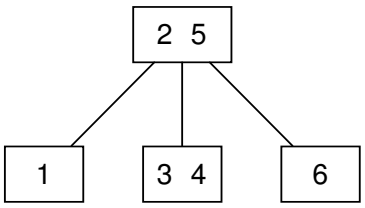
[1]



[2]



[3]

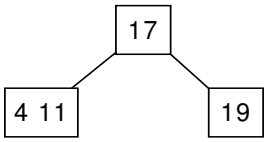


Q.174 Show the result of inserting the keys 4,19, 17, 11, 3, 12, 8, 20, 22, 23, 13, 18, 14, 16, 1, 2, 24, 25, 26, 5 in order in to an empty B-Tree of degree 3. Only draw the configurations of the tree just before some node must split, and also draw the final configuration. **(10)**

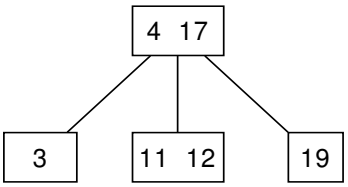
Ans :

4, 19

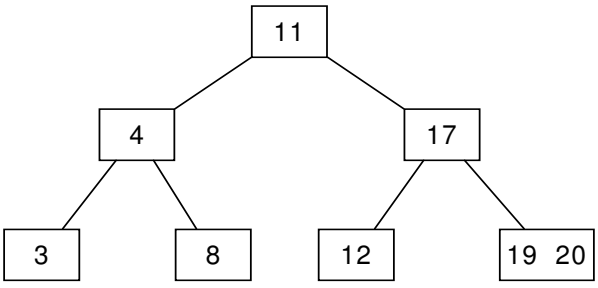
17, 11



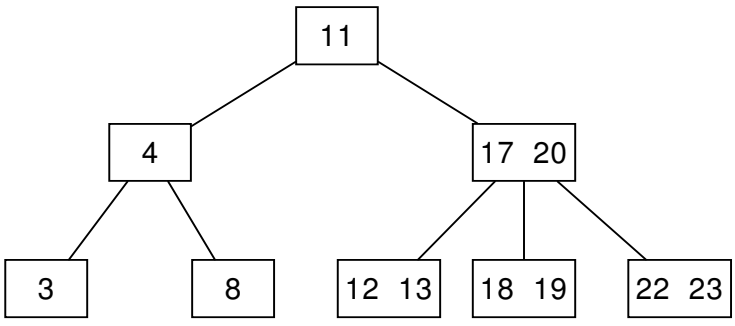
3, 12



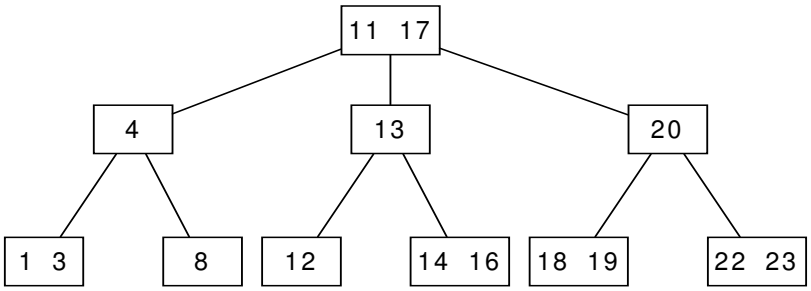
8, 20



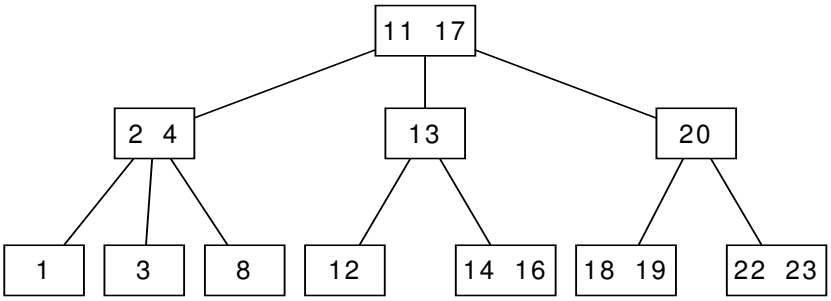
22, 23, 13, 18



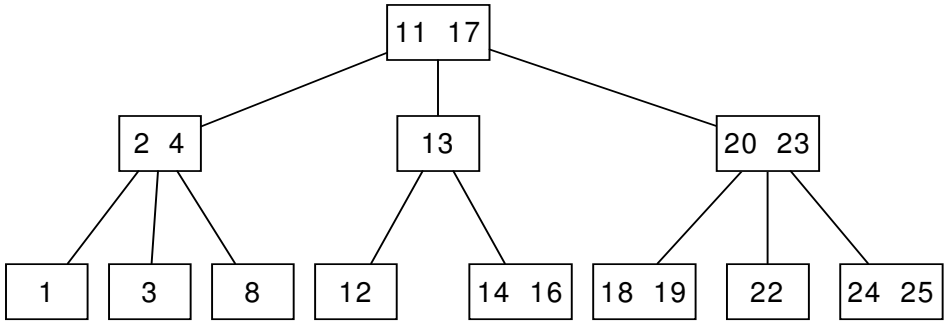
14, 16, 1



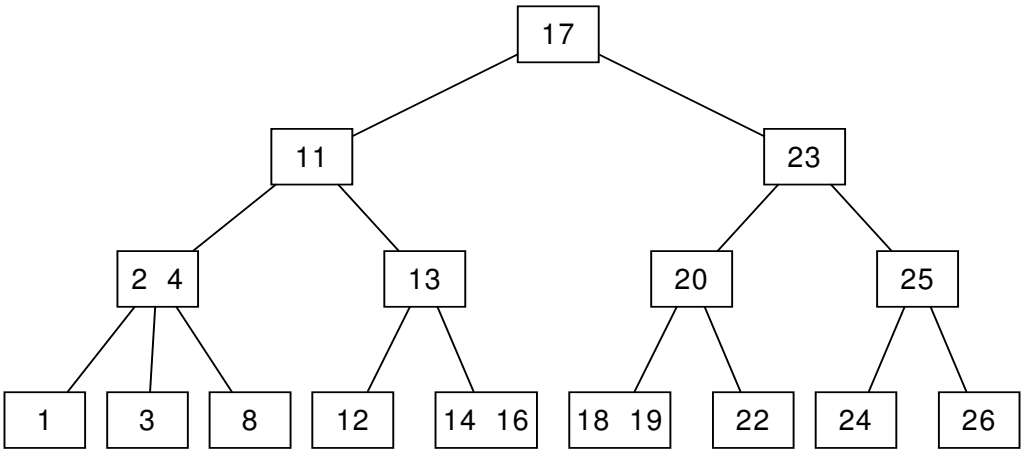
2



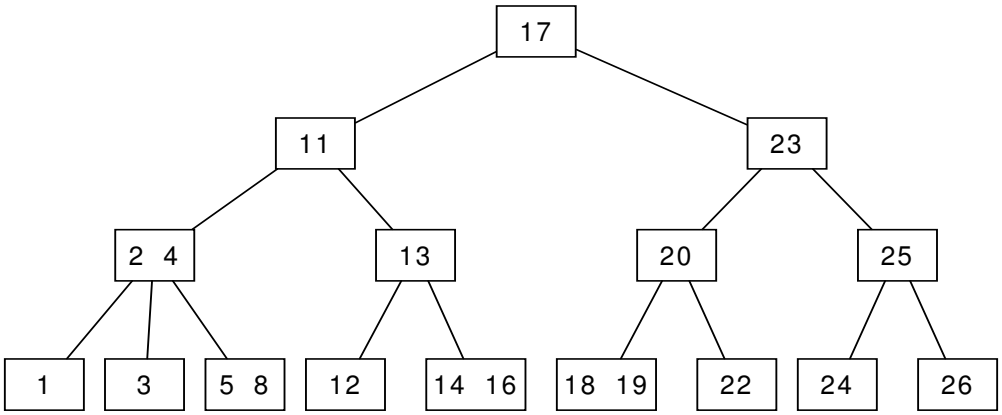
24, 25



26



Finally 5 is inserted



The above is the final tree formed after inserting all the elements given in the question.

Q.175 Write down the algorithm for quick sort.**(8)**

Ans :

The algorithm for quick sort is as follows:

```
void quicksort ( int a[ ], int lower, int upper )
{
  int i ;
  if ( upper > lower )
  {
        i = split ( a, lower, upper ) ;
        quicksort ( a, lower, i - 1 ) ;
        quicksort ( a, i + 1, upper ) ;
  }
}
int split ( int a[ ], int lower, int upper )
{
  int i, p, q, t ;
  p = lower + 1 ;
  q = upper ;
  i = a[lower] ;
  while ( q >= p )
  {
        while ( a[p] < i )
              p++ ;
        while ( a[q] > i )
              q-- ;
        if ( q > p )
        {
              t = a[p] ;
              a[p] = a[q] ;
              a[q] = t ;
        }
  }
  t = a[lower] ;
  a[lower] = a[q] ;
  a[q] = t ;
  return q ;
}
```

Q.176 Show how quick sort sorts the following sequences of keys:
5, 5, 8, 3, 4, 3, 2 **(7)**

Ans :

The given data element to be sorted using quick sort is

5 5 8 3 4 3 2

Choosing pivot 5 (Pass1)

(2) 5 8 3 4 3 (5)

2 5 (5) 3 4 3 (8)

2 5 (3) 3 4 (5) 8

choosing 2 as pivot (pass2)

(2) 5 3 3 4 (5 8)

choosing 5 as pivot (pass 3)

(2) (3 3 4 5) (5 8)

Pass (4), (5) and (6) will not change the sub list

Q.177 On which input data does the algorithm quick sort exhibit its worst-case behaviour? **(1)**

Ans :

The Quick Sort technique exhibits its worst-case behavior when the input data is "Already Completely Sorted".

Q.178 What do you mean by hashing? Explain any five popular hash functions. **(5)**

Ans :

Hashing

Hashing provides the direct access of record from the file no matter where the record is in the file. This is possible with the help of a hashing function H which map the key with the corresponding key address or location. It provides the key-to-address transformation.

Five popular hashing functions are as follows:-

Division Method: An integer key x is divided by the table size m and the remainder is taken as the hash value. It can be defined as

H(x)=x%m+1

For example, x=42 and m=13, H(42)=45%13+1=3+1=4

Midsquare Method: A key is multiplied by itself and the hash value is obtained by selecting an appropriate number of digits from the middle of the square. The same positions in the square must be used for all keys. For example if the key is 12345, square of this key is value 152399025. If 2 digit addresses is required then position 4^{th} and 5^{th} can be chosen, giving address 39.

Folding Method: A key is broken into several parts. Each part has the same length as that of the required address except the last part. The parts are added together, ignoring the last carry, we obtain the hash address for key K.

Multiplicative method: In this method a real number c such that 0<c<1 is selected. For a nonnegative integral key x, the hash function is defined as

H(x)=[m(cx%1)]+1

Here,cx%1 is the fractional part of cx and [] denotes the greatest integer less than or equal to its contents.

Digit Analysis: This method forms addresses by selecting and shifting digits of the original key. For a given key set, the same positions in the key and same rearrangement pattern must be used. For example, a key 7654321 is transformed to

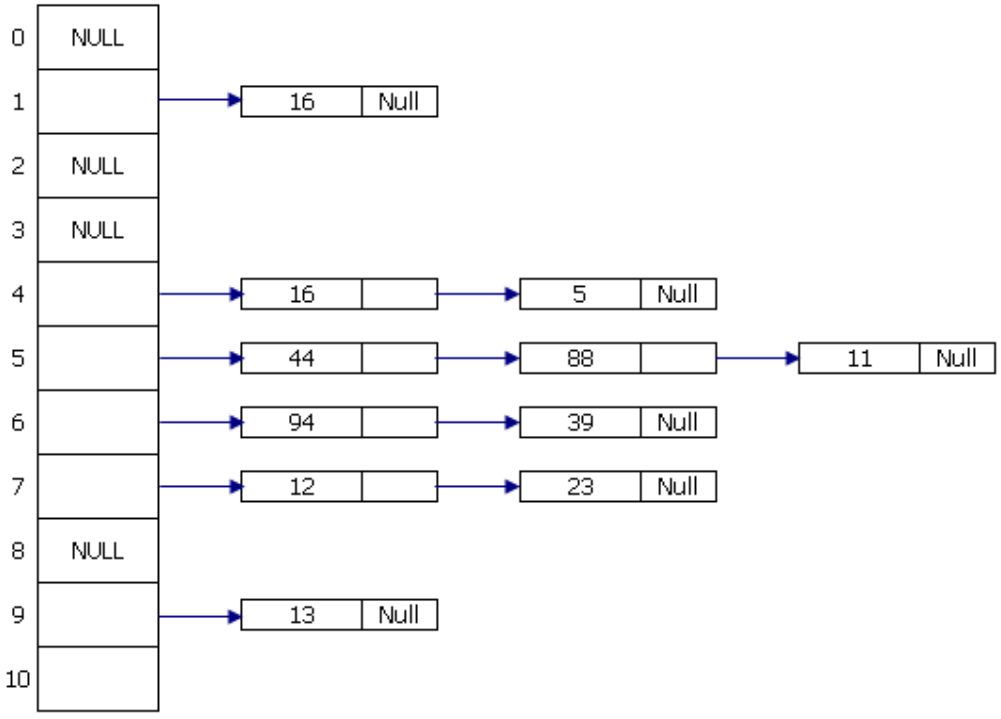
the address 1247 by selecting digits in position 1,2,4 and 7 then by reversing their order.

Q.179 Draw the 11-item hash table resulting from hashing the keys 12, 44, 13, 88, 23, 94, 11, 39, 20, 16 and 5 using the hash function h (i) = (2i+5) mod 11 **(7)**

Ans :

Table size is 11

h(12) = (24+5) mod 11 = 29 mod 11 = 7
h(44) = (88+5) mod 11 = 93 mod 11 = 5
h(13) = (26+5) mod 11 = 31 mod 11 = 9
h(88) = (176+5)mod 11= 181mod 11 = 5
h(23) = (46+5) mod 11 = 51 mod 11 = 7
h(94) = (188+5)mod 11= 193mod 11 = 6
h(11) = (22+5) mod 11 = 27 mod 11 = 5
h(39) = (78+5) mod 11 = 83 mod 11 = 6
h(20) = (40+5) mod 11 = 45 mod 11 = 1
h(16) = (24+5) mod 11 = 29 mod 11 = 4
h(5) = (10+5) mod 11 = 15 mod 11 = 4



Q.180 Explain any two methods to resolve collision during hashing. **(4)**

Ans :

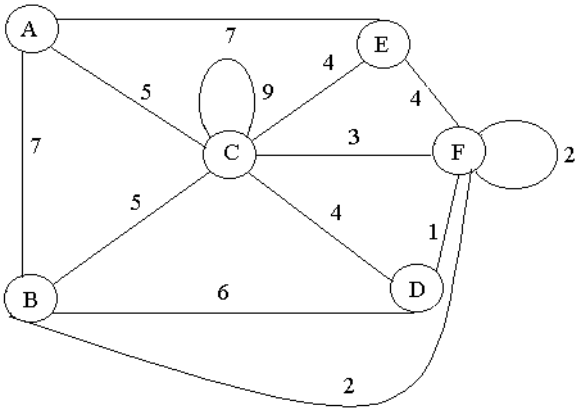
The two methods to resolve collision during hashing are:

Open addressing and Chaining.

Open addressing: The simplest way to resolve a collision is to start with the hash address and do a sequential search through the table for an empty location. The idea is to place the record in the next available position in the array. This method is called linear probing. An empty record is indicated by a special value called null. The major drawback of the linear probe method is clustering.

Chaining: In this technique, instead of hashing function value as location we use it as an index into an array of pointers. Each pointer access a chain that holds the element having same location.

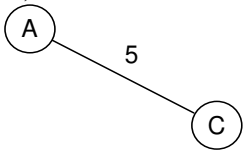
Q.181 Find out the minimum spanning tree of the following graph. **(8)**



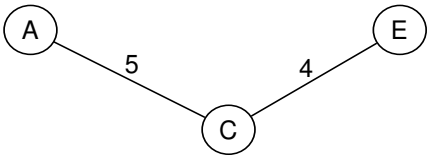
Ans :

The minimum spanning tree of given graph: -

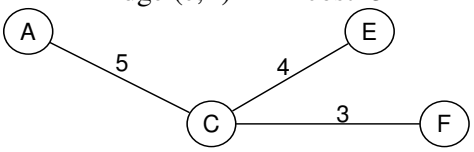
Step 1: Edge (a, c) cost=5 add to tree T



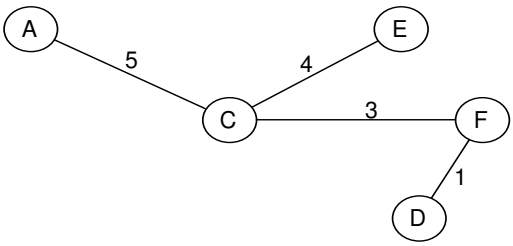
Step 2: Edge (c, e) cost=4 add to T



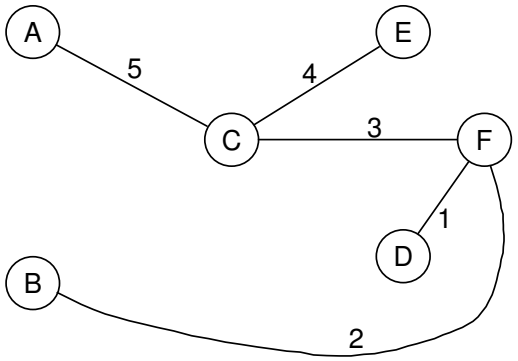
Step 3: Edge (c, f) cost=3 add to T



Step 4: Edge (f, d) cost=1 add to T



Step 5: Edge (f, b) cost=2 add to T



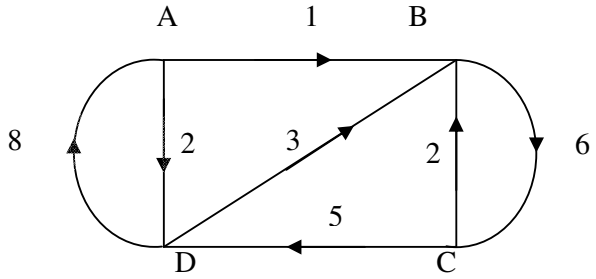
Thus the above tree is the minimum spanning tree of the graph.

Q.182 Give an algorithm to compute the second minimum spanning tree of a graph. **(8)**

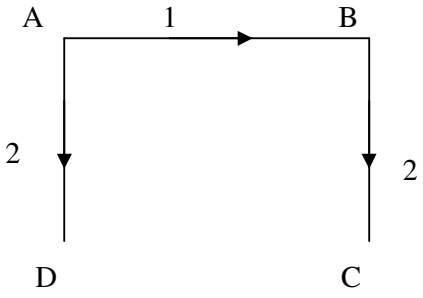
Ans : Find out the minimum spanning tree using Prim`s or Kruskal`s Algorithm.
Remove one edge from MST. One vertex will become isolated. Add an edge, which is not in the MST, such that the vertex gets connected. Find out the increase in the total weight of the spanning tree.
Repeat this process with every edge of the MST.
Remove that edge, whose removal and addition of some other edge, will increase the total weight of MST by minimum. This will give the second minimum spanning tree.

For example:-

Let G be as follows:-



Now MST:-



Remove AB from MST
Now to get a spanning tree, either DB is to be added or CD is to be added. Addition of DB will increase the weight by 2 units (i.e. 3 – 1) and addition of CD will increase the weight by 4 units (5-1). Therefore removal of AB will increase the weight of the next spanning tree by 2.

Now remove edge CB then CD or BC is to be added, edge CVD will increase the weight of the tree by 3 units (5-2)
Similarly if AD is to be removed, it will be replaced by CD which will increase the weight of the tree by 3. Thus the second minimum spanning tree is obtained by remaining AB and adding DB.

Q.183 Define a threaded binary tree. Write an algorithm for inorder traversal of a threaded binary tree. **(6)**

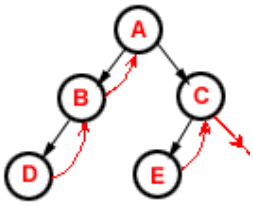
Ans :

Threaded Binary Tree:-

If a node in a binary tree is not having left or right child or it is a leaf node then that absence of child node is represented by the null pointers. The space occupied by these null entries can be utilized to store some kind of valuable information. One possible way to utilize this space is to have special pointer that point to nodes higher in the tree that is ancestors. These special pointers are called threads and the binary tree having such pointers is called threaded binary tree.
There are many ways to thread a binary tree each of these ways either correspond either in-order or pre-order traversal of T.A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its INORDER successor. By doing this threading we avoid the recursive method of traversing a Tree, which makes use of stacks and consumes a lot of memory and time.
The node structure for a threaded binary tree varies a bit and its like this

```
struct NODE
{
 struct NODE *leftchild;
 int node_value;
 struct NODE *rightchild;
 struct NODE *thread;
}
```

The INORDER traversal for the above tree is -- D B A E C. So, the respective Threaded Binary tree will be --



B has no right child and its inorder successor is A and so a thread has been made in between them. Similarly, for D and E. C has no right child but it has no inorder successor even, so it has a hanging thread.
The algorithm for doing the above task is as follows;

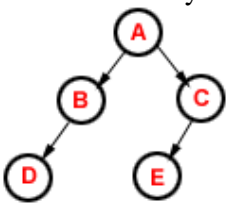
```
Void inorder (NODEPTR root)
{
 NODEPTR p, trail;
 p = root;
 do
       {
        trail = null;
        while ( p!= null)
        {
               trail = p;
               p = p -> left;
```

```
}
if (trail != null)
{
        printf (" %d\n", trail->info);
        p = trail -> right;
        while ( trail -> rt && p != NULL)
        {
                printf ("%d/x", p -> info);
                trail =p;
                p = p -> right;
        }
}
} while (trail != NULL);
  }
```

Q.184 Give an algorithm to sort data in a doubly linked list using insertion sort. **(10)**

Ans :

Simple ***Insertion sort*** is easily adaptable to doubly linked list. In this Let's make the Threaded Binary tree out of a normal binary tree...



method there is an array *link* of pointers, one for each of the original array elements. Initially link[i] = i + 1 for 0 < = I < n-1 and link [n-1] = -1. Thus the array can be thought of as a doubly link list pointed to by an external pointer *first* initialized to 0. To insert the kth element the linked list is traversed until the proper position for x[k] is found, or until the end of the list is reached. At that point x[k] can be inserted into the list by merely adjusting the list pointers without shifting any elements in the array. This reduces the time required for insertion but not the time required for searching for the proper position. The number of replacements in the link array is O (n).

```
void insertion_sort( input type a[ ], unsigned int n )
{
 unsigned int j, p;
 input type tmp;
        a[0] = MIN_DATA;          /* sentinel */
 for ( p=2; p <= n; p++ )
 {
              tmp = a[p];
              for ( j = p; tmp < a[j-1]; j-- )
                   a[j] = a [j-1];
              a[j] = tmp;
 }
}
```

Q.185 What is an Abstract Data Type (ADT)? Explain with an example. **(4)**

Ans :

Abstract data types or **ADTs** are a mathematical specification of a set of data and the set of operations that can be performed on the data. They are abstract in the sense that the focus is on the definitions of the constructor that returns an abstract handle

that represents the data, and the various operations with their arguments. The actual implementation is not defined, and does not affect the use of the ADT.
For **example**, rational numbers (numbers that can be written in the form a/b where a and b are integers) cannot be represented natively in a computer. A Rational ADT could be defined as shown below.
Construction: Create an instance of a rational number ADT using two integers, a and b, where a represents the numerator and b represents the denominator.
Operations: addition, subtraction, multiplication, division, exponentiation, comparison, simplify, conversion to a real (floating point) number.
To be a complete specification, each operation should be defined in terms of the data. For example, when multiplying two rational numbers a/b and c/d, the result is defined as ac/bd. Typically, inputs, outputs, preconditions, postconditions, and assumptions to the ADT are specified as well.
When realized in a computer program, the ADT is represented by an interface, which shields a corresponding implementation. Users of an ADT are concerned with the interface, but not the implementation, as the implementation can change in the future.
ADTs typically seen in textbooks and implemented in programming languages (or their libraries) include:

- String ADT
- List ADT
- Stack (last-in, first-out) ADT
- Queue (first-in, first-out) ADT
- Binary Search Tree ADT
- Priority Queue ADT
- Complex Number ADT (imaginary numbers)

There is a distinction, although sometimes subtle, between the abstract data type and the data structure used in its implementation. For example, a List ADT can be represented using an array-based implementation or a linked-list implementation. A List is an abstract data type with well-defined operations (add element, remove element, etc.) while a linked-list is a pointer-based data structure that can be used to create a representation of a List. The linked-list implementation is so commonly used to represent a List ADT that the terms are interchanged and understood in common use.
Similarly, a Binary Search Tree ADT can be represented in several ways: binary tree, AVL tree, red-black tree, array, etc. Regardless of the implementation, the Binary Search Tree always has the same operations (insert, remove, find, etc.)

Q.186 Suppose we wish to multiply four matrices of real numbers M1*M2*M3*M4 were M1 is 10*20, M2 is 20*50, M3 is 50*1 and M4 is 1*100. Assume that the multiplication of a p*q matrix by q*r matrix requires pqr scalar operations, find the optimal order in which to multiply the matrices so as to minimize the total number of scalar operations. **(5)**

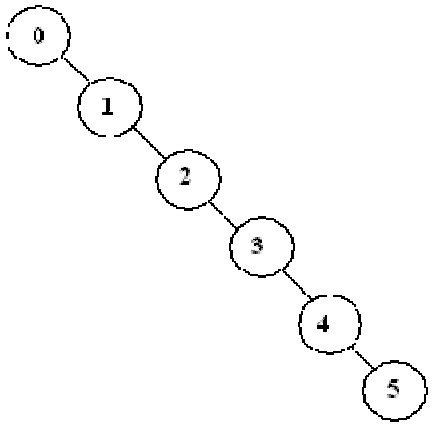
Ans :
Here M1 = 10x20
M2 = 20x50
M3 = 50x1 and
M4=1x100
Now optimal order is
1. M2*M3 = [20x50]*[50x1] = [20x1]
And no of scalar operations are 20*50*1 = 1000

2. M1*M2M3 = [10x20]*[20x1] = [10x1]
   And no of scalar operations are 10*20*1 = 200
3. M1M2M3*M4 = [10x1]*[10x100] = [10x100]
   And no of scalar operations are 10*1*100=1000
   So total no of scalar operations require are 1000+200+1000=2200 .

Q.187 Show a tree (of more than one node) for which the preorder and inorder traversals generate the same sequence. Is this possible for preorder and post order traversals? If it is, show an example. **(5)**

Ans :

A binary tree with 5 nodes 0 , 1, 2, 3 and 4 whose inorder and post order sequences are the same .



This is not possible for preorder and post order traversals.

Q.188 Write modules to do the following operations on a Binary Tree.

(i) Count the number of leaf nodes.
(ii) Count the number of nodes with two children.

(9)

Ans :

```
(i ) Leafcount (T)
        {
         static int n=0;
         if (T!= NULL)
        {leaf count (T→ left);
        if (T→left == NULL && T→right = NULL)
        n++
        leafcount (T→right)
        }
        return (n);}
  (ii) Leafcount (T)
        {
        static int n=0;
         if (T! = NULL)
        {leaf count (T→ left);
        if (T→left!= NULL && T→right! = NULL)
        n++
        leafcount (T→right)
        }
        return (n);}
```

Q.189 If n >=1, prove that for any n-key B-tree T of height h and minimum degree t >= 2, h <= $\log_t (n+1)/2$. **(7)**

Ans :

In the case of a B-tree with a minimum no of keys , there is one key in the root , 2 nodes at depth 1 , $2t^i-1$ nodes at depth i .

Let h be the height of the tree and n be the no of nodes . Then

$$n >= 1+(t-1) \sum_{i=1}^{h} 2t^i-1$$

which works out to $t^h <= (n+1) / 2$

So we take the $\log_t$ of both sides

$h <= \log_t(n+1)/2$

Q.190 A cocktail shaker sort designed by Donald Kunth is a modification of bubble sort in which the direction of bubbling changes in each iteration: in one iteration, the smallest element is bubbled up; in the next, the largest is bubbled down; in the next, the second smallest is bubbled up; and so forth. Write an algorithm to implement this and explore its complexity. **(14)**

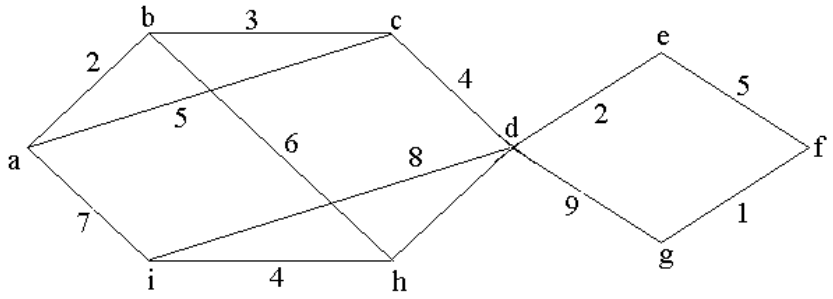
Ans :

Cocktail sort, also known as **bidirectional bubble sort**, **cocktail shaker sort**, **shaker sort**, **ripple sort**, or **shuttle sort**, is a stable sorting algorithm that varies from **bubble sort** in that instead of repeatedly passing through the list from top to bottom, it passes alternately from top to bottom and then from bottom to top. It can achieve slightly better performance than a standard bubble sort.

Complexity in Big O notation is $O(n^2)$ for a worst case, but becomes closer to O(n) if the list is mostly ordered at the beginning.

```
void cocktail_sort (int A[], int n)
 {
   int left = 0, right = n;
   bool finished;
   do
   {
    finished = true;
    --right;
    for (int i = left; i < right; i++)
     if (A[i] > A[i+1])
        {
      std::swap(A[i], A[i+1]);
      finished = false;
     }
   if (finished) return; finished = true;
  for (int i = right; i > left; i--)
   if (A[i] < A[i-1])
    {
     std::swap(A[i], A[i-1]);
     finished = false;
    }
  ++left;
 } while (!finished);
}
```

Q.191 Consider the following undirected graph:



(i) Find a minimum cost spanning tree by Kruskal's algorithm.
(ii) Find a depth-first spanning tree starting at ***a*** and at ***d***.
(iii) Find a breadth-first spanning tree starting at ***a*** and at ***d***.
(iv) Find the adjacency list representation of the graph. **(3.5 x 4 = 14)**

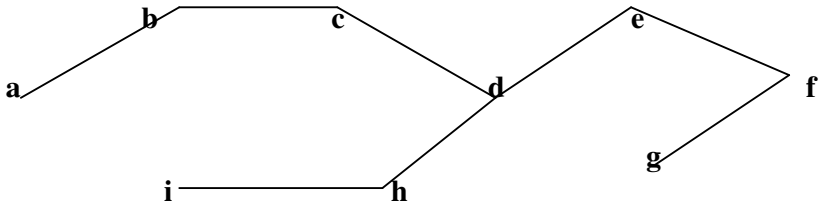
Ans :

(i) Find a minimum cost spanning tree by kruskal's algorithm .

Assume dh = 3

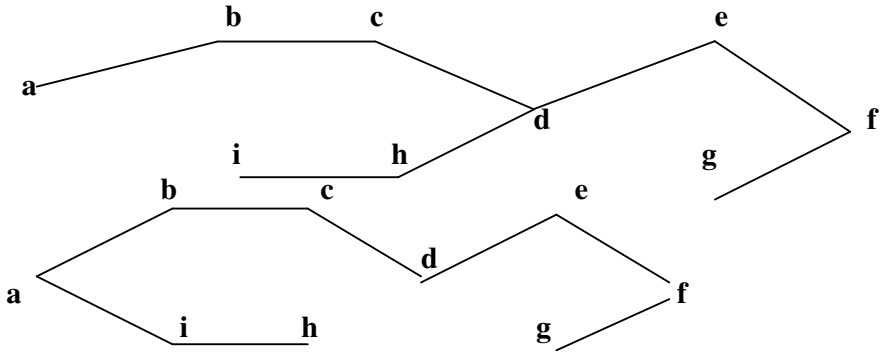
Edges	(g f)	=	1	Accept
	(a b)	=	2	Accept
	(d e)	=	2	Accept
	(b c)	=	3	Accept
	(d h)	=	3	Accept
	(c d)	=	4	Accept
	(i h)	=	4	Accept
	(e f)	=	5	Accept
	(a c)	=	5	Reject
	(b h)	=	6	Reject
	(a i)	=	7	Reject
	(i d)	=	8	Reject
	(d g)	=	9	Reject

So minimum cost spanning tree is :

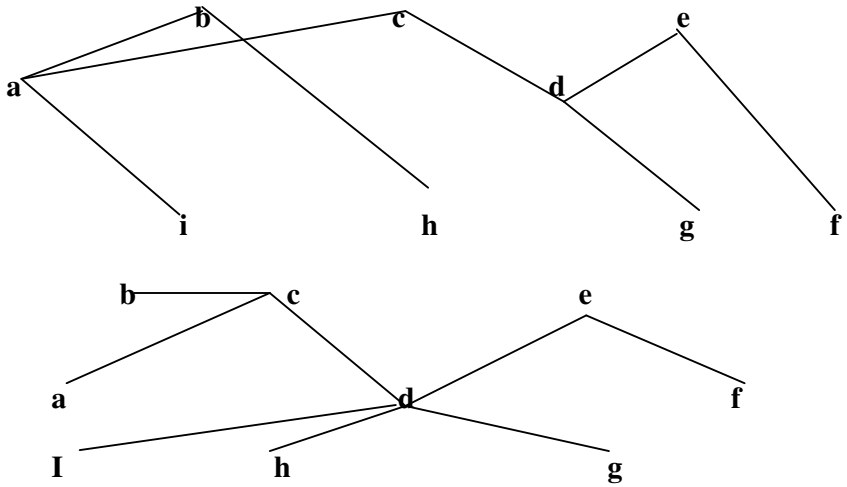


And Cost is 2+3+4+2+5+1+3+4 = 24

(ii) Find a Depth first spanning tree starting at a and d .



(iii) Find a breadth first spanning tree starting at a and d .



(iv) Find the adjacent list representation of the graph .

a	→	b,c,i
↓	→	
b	→	a,c,h
↓	→	
c	→	a,b,d
↓	→	
d	→	c,e,g,h,i
↓	→	
e	→	d,f
↓	→	
f	→	e,g
↓	→	
g	→	d,f
↓	→	
h	→	b,d,i
↓	→	
i	→	a,d,h

Q.192 Work through Binary Search algorithm on an ordered file with the following keys: {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}. Determine the number of key comparisons made while searching for keys 2, 10 and 15. **(6)**

Ans :

Here List={1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16}

Binary Search for key 2

(1) Here bottom =1 Top =16 and middle =(1+16)/2 = 8
Since 2 < list(8)

(2) bottom = 1 Top = middle-1=7 and middle=(1+7)/2 =4
2 < list(4)

(3) bottom = 1 Top = middle-1=3 and middle=(1+3)/2 = 2
2 = List(2)

So total number of comparisons require = 3

Binary Search for key = 10

(1) Here bottom=1 Top=16 and middle = 8
10 > List(8)

(2) bottom = middle+1=9 Top=16 middle=(9+16)/2=12
10 < List(12)

(3) bottom =9 Top=middle-1=11 middle=(9+11)/2=10
10 = List(10)

So total no of comparisons = 3

Binary Search for key = 15

(1) Here bottom=1 Top=16 and middle = 8
15 > List(8)

(2) bottom = middle+1=9 Top=16 middle=(9+16)/2=12
15 > List(12)

(3) bottom =middle+1=13 Top=16 middle=(13+16)/2=14
15 > List(14)

(4) bottom = middle+1 =15 Top=16 middle=(15+16)/2=15
15=List(15)

So total no of comparisons = 4

Q.193 What are threaded binary trees? What are the advantages and disadvantages of threaded binary trees over binary search trees? **(4)**

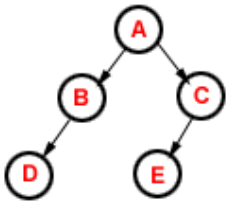
Ans :

A Threaded Binary Tree is a binary tree in which every node that does not have a right child has a THREAD (in actual sense, a link) to its INORDER successor. By doing this threading we avoid the recursive method of traversing a Tree, which makes use of stacks and consumes a lot of memory and time.

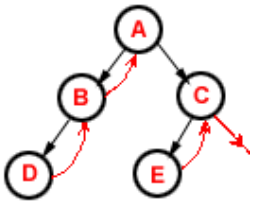
The node structure for a threaded binary tree varies a bit and its like this --

```
struct NODE
{
 struct NODE *leftchild;
 int node_value;
 struct NODE *rightchild;
 struct NODE *thread;
}
```

Let's make the Threaded Binary tree out of a normal binary tree...



The INORDER traversal for the above tree is -- D B A E C. So, the respective Threaded Binary tree will be --



B has no right child and its inorder successor is A and so a thread has been made in between them. Similarly, for D and E. C has no right child but it has no inorder successor even, so it has a hanging thread.

Advantage

1. By doing threading we avoid the recursive method of traversing a Tree , which makes use of stack and consumes a lot of memory and time .
2 . The node can keep record of its root .

Disadvantage

1. This makes the Tree more complex .
2. More prone to errors when both the child are not present & both values of nodes pointer to their ansentors .

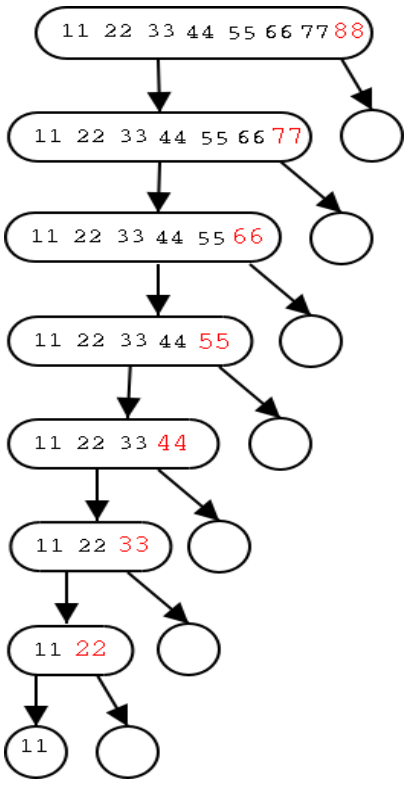
Q.194 Show that quick algorithm takes O $\left(n^2\right)$ time in the worst case. **(4)**

Ans :

The Worst Case for Quick-SortThe tree illustrates the worst case of quick-sort, which occurs when the input is already sorted!

The height of the tree is N-1 not O(log(n)). This is because the pivot is in this case the largest element and hence does not come close to dividing the input into two pieces each about half the input size. It is easy to see that we have the worst case. Since the pivot does not appear in the children, at least one element from level i does not appear in level i+1 so at level N-1 you can have at most 1 element left. So we have the highest tree possible. Note also that level i has at least i pivots missing so cna have at most N-i elements in all the nodes. Our tree achieves this maximum. So the time needed is proportional to the total number of numbers written in the diagram which is N + N-1 + N-2 + ... + 1, which is again the one summation we know N(N+1)/2 or $\Theta(N^2$.

Hence the ***worst case*** complexity of quick-sort is quadratic .



Q.195 Write an algorithm to convert an infix expression to a postfix expression. Execute your algorithm with the following infix expression as your input. $(m+n)*(k+p)/(g/h)\uparrow(a\uparrow b/c)$ **(8)**

Ans :

Algorithm: Polish (Q,P)

Suppose Q is an arithmatic expression written in fix notation. This algorithm finds the equivalent postfix expression P.

1. Push " (" onto STACK, and add ")" to the end of Q
2. Scan Q from left to right and repeat Steps 3 to 6 for each element of Q until the STACK is empty:
3. If an operand is encountered, add it to P
4. If a left parenthesis is encountered, push it onto STACK.
5. If an operator [x] is encountered, then:

(a) repeatedly pop from STACK and add to p each operator (on the top of stack) has the same precedence as or higher precedence than [x]

(b) add [x] to STACK.

[End of If structure]

6. If a right parenthesis is encountered, then:

(a) Repeatedly pop from STACK and add to P each operator (on the top of STACK) until a left parenthesis is encountered.

(b) Remove the left parenthesis. [Do not add the left parenthesis to P]

[End of If structure.]

[End of Step 2 loop.]

7. Exit

Example :
Input (m+n)*(k+p)/(g/h)^(a^b/c)

Step 1 : (| (| Output : nil

Step 2 : m | (| Output : m

Step 3 : + | + (| Output : m

Step 4 : n | + (| Output : mn

Step 5 :) | | Output : mn+

Step 6 : * | * | Output : mn+

Step 7 : (| (* | Output : mn+

Step 8 : k | (* | Output : mn+k

Step 9 : + | + (* | Output : mn+k

Step 10 : p | + (* | Output : mn+kp

Step 11 :) | * | Output : mn+kp+

Step 12 : / | / | Output : mn+kp+*

Step 13 :(| (/ | Output : mn+kp+*

Step 14 :g | (/ | Output : mn+kp+*g

Step 15 :/ | / (/ | Output : mn+kp+*g

Step 16 : h | / (/ | Output : mn+kp+*gh

Step 17 :^ | ^ / | Output : mn+kp+*gh/

Step 18 :(| (^ / | Output : mn+kp+*gh/

Step 19 : a | (^ / | Output : mn+kp+*gh/a

Step 20 : ^ | ^ (^ / | Output : mn+kp+*gh/a

Step 21 : b | ^ (^ / | Output : mn+kp+*gh/ab

Step 22 : / | / (^ / | Output : mn+kp+*gh/ab^

Step 23 : c | / (^ / | Output : mn+kp+*gh/ab^c

Step 24 :) | ^ / | Output : mn+kp+*gh/ab^c/

Step 25 : End | | Output : **mn+kp+*gh/ab^c/^/**

Q.196 What is recursion? A recursive procedure should have two properties. What are they? **(4)**

Ans :

Recursion means function call itself repeately .

Properties :

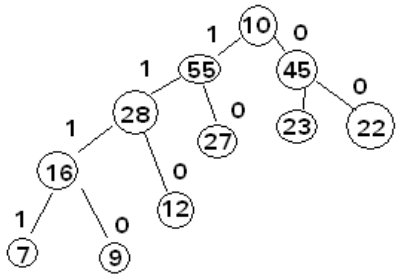
(1) There must be some **base case** where the condition end .

(2) Each recursive call to the procedure involve a smaller case of the problem .

Q.197 Suppose characters a,b,c,d,e,f have probabilities 0.07, 0.09, 0.12, 0.22, 0.23, 0.27 respectively. Find an optimal Huffman code and draw the Huffman tree. What is the average code length? **(10)**

Ans :

Huffman Tree :



Huffman Code are a=1111
b =1110
c=110
d=00
e=01
f=10

Average code length = P(a).4+P(b).4+P(c).3+P(d).2+P(e).2+P(f).2
= 0.07*4+0.09*4+0.12*3+0.22*2+0.23*2+0.27*2
= .28+.36+.36+.44+.46+.54 =2.44

Q.198 Prove that the number of nodes with degree 2 in any Binary tree is 1 less than the number of leaves. **(4)**

Ans :

The proof is by induction on the size n of T .

Let L(T) = No of leaves & $D_2(T)$= No of nodes of T of degree 2

To Prove D2(T)=L(T)-1

Basic Case : n=1 , then T consists of a single node which is a leaf .

L(T)=1 and D2(T)=0

So $D_2(T)$=L(T)-1

Induction Step : Let n > 1 and assume for all non empty trees T1 of size k<n that $D_2(T`)$=L(T`)-1

Since n >1 , at least one of x=left(T) or y=right(T) is non empty .

Assume x is non empty ; the other case is symetric

By the induction hypothesis ,

$D_2(x)$ = L(y)-1

IF y is empty , then again by the induction hypothesis ,

$D_2(y)$ =L(y)-1 and

$D_2(T)$ =$D_2(x)$+$D_2(y)$+1

=L(x)-1+L(y)-1+1

= L(x)+L(y)-1

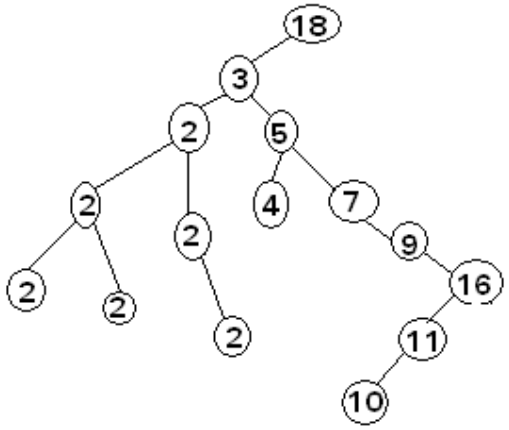
$D_2(T)$ = L(T)-1 Hence Proved

Q.199 Equal keys pose problem for the implementation of Binary Search Trees. What is the asymptotic performance the usual algorithm to insert n items with identical keys into an initially empty Binary Search Tree? Propose some technique to improve the performance of the algorithm. **(7)**

Ans :

A sympotic performance will be linear in O(n). That is the tree created will be skew symmetric. Identical keys should be attached either as left or right child of the parent, wherever positions are available. For eg. 18,3,5,2,4,7,2,9,2,16,2,11,2,10,2

The creation of the tree should be as follows:



i.e. similar keys should not skew to the BST. Then performance will become inferior.

Q.200 Consider a list of numbers 9, 20, 6, 10, 14, 8, 60, 11 given. Sort them using Quick Sort. Give step wise calculation. **(8)**

Ans:

9,20,6,10,14,8,60,11

choosing 9 as Pivot(Pass—1)

6,8,9,10,14,20,60,11

The sublist on the left side of 9 is sorted therefore, no more sorting on left sublist

Now choosing 10 as Pivot (pass – 2)

(10 is already in the position)

6,8,9,10,14,20,60,11

Choosing 14 as Pivot (pass –3)

6,8,9,10,11,14,60,20

Choosing 60 as Pivot [pass – 4)

6,8,9,10,11,14,20,60

Q.201 What is a sparse matrix? Explain an efficient way of storing a sparse matrix in memory. **(4)**

Ans:

Sparse Matrix

A matrix in which number of zero entries are much higher than the number of non zero entries is called sparse matrix. The natural method of representing matrices in memory as two-dimensional arrays may not be suitable foe sparse matrices. One may save space by storing for only non zero entries. For example matrix A (4*4 matrix) represented below

where first row represent the dimension of matrix and last column tells the number of non zero values; second row onwards it is giving the position and value of non zero number.

0	0	0	15
0	0	0	0
0	9	0	0
0	0	4	0

Here the memory required is 16 elements X 2 bytes = 32 bytes

The above matrix can be written in sparse matrix form as follows:

4	4	3
0	3	15
2	1	9
3	2	4

Here the memory required is 12elements X 2 bytes = 24 bytes

Q.202 Evaluate the following prefix expressions **(9)**

(i) + * 2 + / 14 2 5 1

(ii) – * 6 3 – 4 1

(iii) + + 2 6 + – 13 2 4

Ans:

Evaluating a prefix expression.

(i)	+	*	2	+	/	14	2	5	1
	=	+	*	2	+	7	5	1	
	=	+	*	2	12	1			
	=	+	24	1					
	=	25							
(ii)	-	*	6	3	-	4	1		
	=	-	18	-	4	1			
	=	-	18	3					
	=	15							
(iii)	+	+	2	6	+	-	13	2	4
	=	+	8	+	-	13	2	4	
	=	+	8	+	11	4			
	=	+	8	15					
	=	23							

Q.203 (i) Give four properties of Big – O Notations. **(4)**

(ii) Give the graphical notations of Big-O estimations for the following functions:

$\log_2 n$, n , $n \log_2 n$, n^2, n^3, 2^n **(3)**

Ans:

(i) Four properties of Big-O notations are:

1. Reflexivity
2. Symmetry
3. Transpose Symmetry
4. Transitivity

Reflexivity:

$f(n)=O(f(n))$

Symmetry and Transpose Symmetry

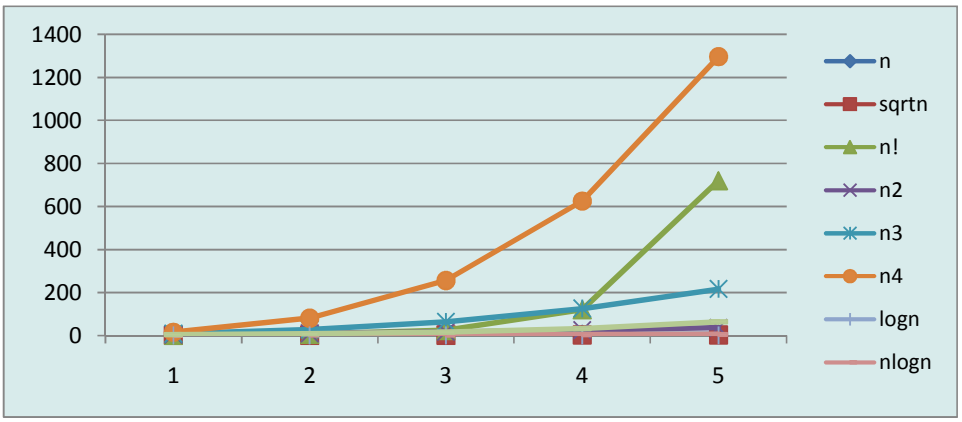
$f(n)=O(g(n))$ iff $g(n)=\Omega(f(n))$

Transitivity

$f(n)=O(g(n))$ and $g(n)=O(h(n))$ implies $f(n)= O(h(n))$

(ii)

n	Sqrt n	n!	n2	n3	n4	logn	nlogn	2n
2	1.41	2	4	8	16	0.30103	0.60206	4
3	1.73	6	9	27	81	0.477121	1.431364	8
4	2.00	24	16	64	256	0.60206	2.40824	16
5	2.24	120	25	125	625	0.69897	3.49485	32
6	2.45	720	36	216	1296	0.778151	4.668908	64
7	2.65	5040	49	343	2401	0.845098	5.915686	128
8	2.83	40320	64	512	4096	0.90309	7.22472	256
9	3.00	362880	81	729	6561	0.954243	8.588183	512
10	3.16	3628800	100	1000	10000	1	10	1024



Q.204 Write an algorithm to insert an element *k* in double linked list at **(8)**

(i) Start of linked list
(ii) After a given position *P* of list
(iii) End of linked list.

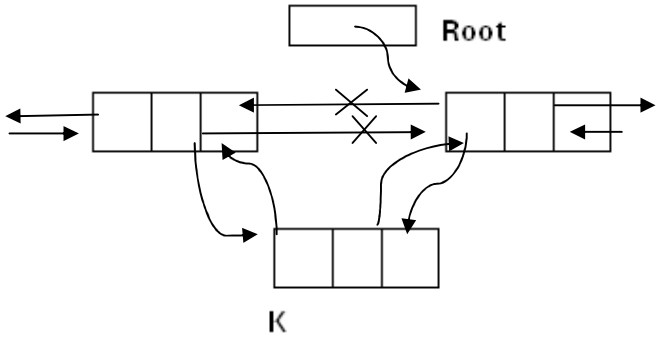
Ans:

Algorithm to insert an element k in double linked list at a position which is:-

(i) Start of a linked list

while inserting at the start of the linked list we have to change the root node and so we need to return the new root to the calling program.

```
nodeptr insertatstart ( nodeptr root, nodeptr k)
{
k->next = root;
k->back = root->back;
root->back->next = k;
root->back=k;
return k;
}
```

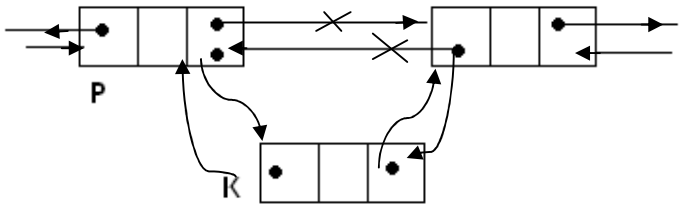


(ii) After a given position p of list

Here we assume that p points to that node in the doubly linked list after which insertion is required.

```
insertafter ( nodeptr p , nodeptr k )
{
        if (p == null)
        {
                printf ("void insertion /n");
                return;
        }
        k->back = p;
```

```
            k->next = p->next;
            p->next->back = k;
            p->next = k;
            return;
      }
```



(iii) End of linked list

Here k is the element to be inserted and root is the pointer to the first node of the doubly linked list.

```
      insertatend ( nodeptr root, nodeptr k )
      {
            nodeptr trav;
            while (trav->next != root)
                  trav = trav->next;
            k->next=root;
            root->back=k;
          trav->next = k;
            k->back=trav;
            return;
      }
```

Q.205 Write an algorithm to add two polynomials using linked list. **(8)**

Ans:

Algorithm to add two polynomials using linked list is as follows:-

```
Let p and q be the two polynomials represented by the linked list.
1. while p and q are not null, repeat step 2.
2. If powers of the two terms ate equal
 then if the terms do not cancel
 then insert the sum of the terms into the sum Polynomial
              Advance p
              Advance q
 Else if the power of the first polynomial> power of second
 Then insert the term from first polynomial into sum polynomial
              Advance p
 Else insert the term from second polynomial into sum polynomial
              Advance q
3. copy the remaining terms from the non empty polynomial into the
sum polynomial.
```

The third step of the algorithm is to be processed till the end of the polynomials has not been reached.

Q.206 Write modules to perform the following operation on Binary tree **(8)**

(i) count number of leaf nodes

(ii) find height of tree

Ans:

(i) count number of leaf nodes

```
int leaf count (node  *t)
{
static int count = 0;
      if (t! = NULL)
{
     leaf count (t→ left)
    if (t→left == NULL && t→right == NULL)
       count ++;
      leaf count (t→right);
}
       return (count);
}
```

(ii) Module to find height of tree

```
height(Left,Right,Root,height)
  1.  If Root=Null then height=0;
      return;
  2.  height(Left,Right,Left(Root),heightL)
  3.  height(Left,Right,Right(Root),heightR)
  4.  If heightL≥heightR then
            height=heightL+1;
       Else
            height=heightR+1;
  5.  Return;
```

Q.207 Create B-Tree of order 5 from the following list of data items: **(8)**
20, 30, 35, 85, 10, 55, 60, 25

Ans:

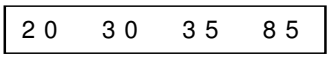
For the given list of elements as

20 30 35 85 10 55 60 25

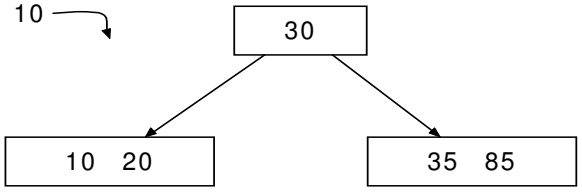
The B-Tree of order 5 will be created as follows

Step 1:

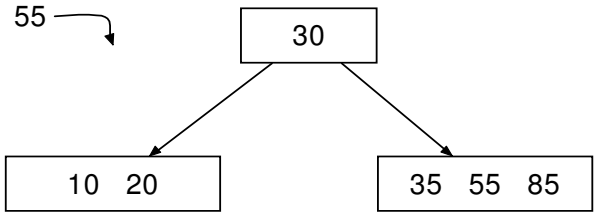
Insert 20, 30, 35 and 85



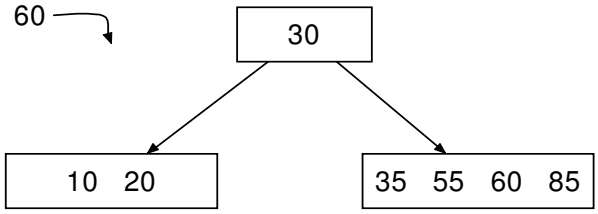
Step 2:



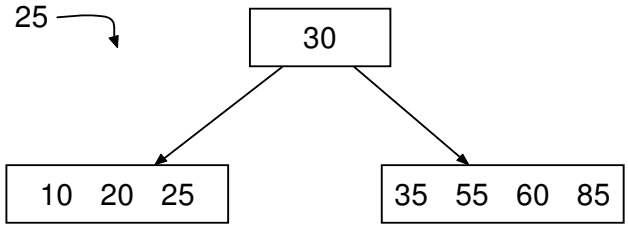
Step 3:



Step 4:



Step 5:



This is the final B-tree created after inserting all the given elements.

Q.208 Write an algorithm to insert an element *k* into binary search tree. Give the analysis and example. **(8)**

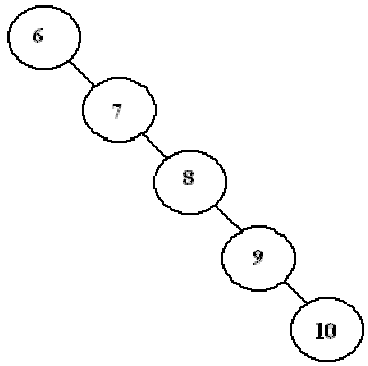
Ans:

The algorithm to insert an element k into binary search tree is as follows:-

```
/* Get a new node and make it a leaf*/
getnode (k)
left(k) = null
right(k) = null
info (k) = x
/* Initialize the traversal pointers */
p = root
trail = null
/* search for the insertion place */
while p <> null do
begin
trail = p
if info (p) > x
       then p = left (p)
else
       p = right (p)
end
/* To adjust the pointers */
If trail = null
Then root = k /* attach it as a root in the empty tree */
      else
             if info (trail) > x
             then
                    left (trail) = k
             else
                    right (trail) = k
```

Analysis : - We notice that the shape of binary tree is determined by the order of insertion. If the values are sorted in ascending or descending order, the resulting tree will have maximum depth equal to number of input elements. The shape of the tree is important from the point of view of search efficiency. The depth of the tree determines the maximum number of comparisons. Therefore we can maximize search efficiency by minimizing the height of the tree.

eg. For the values as 6, 7, 8, 9, 10
the binary tree formed would be as follows



Q.209 Fill in the following table, showing the *number of comparisons* necessary to either find a value in the array DATA or determine that the value is not in the array. **(8)**

DATA

26	42	96	101	102	162	197	201	243
[1]	[2]	[3]	[4]	[5]	[6]	[7]	[8]	[9]

Number of Comparisons

Value	**Sequential Ordered Search**	**Binary Search**
26		
2		
103		
244		

Ans:

Value	Sequential Ordered Search	Binary Search
26	1	4
2	1	4
103	6	3
244	9	4

Q.210 Explain the functionality of linear and quadratic probing with respect to hashing technique. **(8)**

Ans:

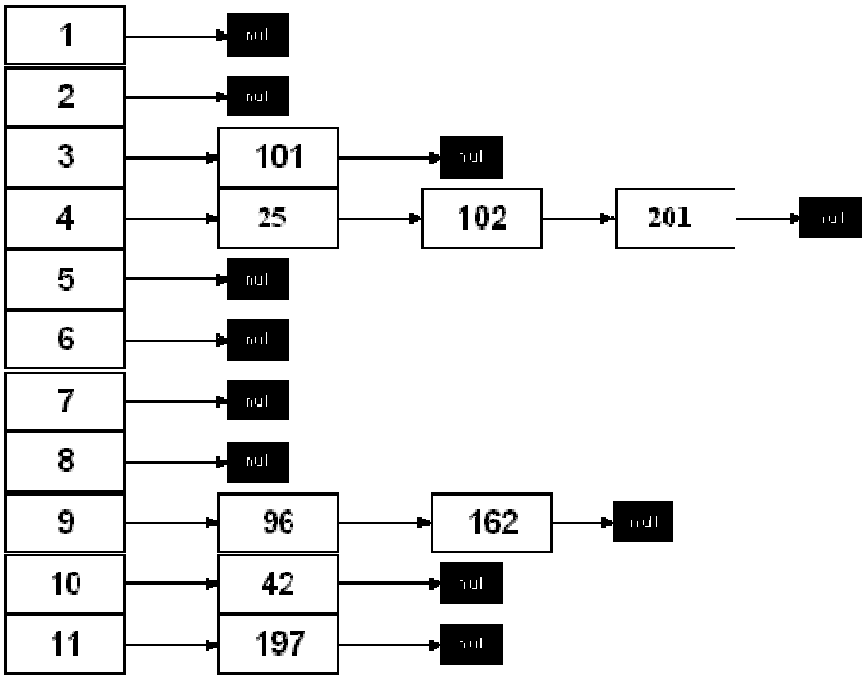
Linear Probing: The simplest way to resolve a collision is to start with the hash address and do a sequential search through the table for an empty location. The idea is to place the record in the next available position in the array. This method is called linear probing. An empty record is indicated by a special value called null. The array should be considered circular, so that when the last location is reached the search proceeds to the first record of the array. An unoccupied record location is always found using this method if atleast one is available; otherwise, the search halts unsuccessfully after scanning all locations. The major drawback of the linear probe method is that of clustering.

When the table is initially empty, it is equally likely that a new record will be inserted in any of the empty position but when the list becomes half full, records start to appear in long strings of adjacent positions with gaps between the strings. Therefore the search to find an empty position becomes longer.

Quadratic probing: In the above case when the first insertion is made the probability of new element being inserted in a particular position is 1/n where n is the size of the array. At the time of second insertion the probability becomes 2/n and so on for the k^{th} insertion the probability is k/n, which is k times as compared to any other remaining unoccupied position. Thus to overcome the phenomenon of long sequence of occupied positions to become even longer we use *quadratic rehash,* in this method if there is a collision at hash address h, this method probes the array at locations h+1, h+4, h+9, ... That is (h(key) $+i^2$ % hash size) for i=1,2,... gives the i^{th} hash of h.

Q.211 The following values are to be stored in a hash table
25, 42, 96, 101, 102, 162, 197, 201
Use division method of hashing with a table size of 11. Use sequential method of resolving collision. Give the contents of array. **(8)**

Ans:
Table size is given as 11
H (25) = (25) mod 11 + 1 = 3 + 1 = 4
H (42) = (42) mod 11 + 1 = 9 + 1 = 10
H (96) = (96) mod 11 + 1 = 8 + 1 = 9
H (101) = (101) mod 11 + 1 = 2 + 1 = 3
H (102) = (102) mod 11 + 1 = 3 + 1 = 4
H (162) = (162) mod 11 + 1 = 8 + 1 = 9
H (197) = (197) mod 11 + 1 = 10 + 1 = 11
H (201) = (201) mod 11 + 1 = 3 + 1 = 4



Q.212 Write Kruskal's algorithm and give the analysis. Compare it with Dijkstra's method. **(8)**

Ans:

Kruskal's Algorithm for minimum cost spanning tree

```
1. Make the tree T empty.
2. Repeat steps 3, 4 and 5 as long as T contains less
than n-1 edges and E not empty; otherwise proceed to step 6.
3. Choose an edge (v,w) from E of lowest cost.
4. Delete (v,w) from E.
5. If (v,w) does not create a cycle in T then add(v,w) to T
else dicard(v,w).
6. If T contains fewer than n-1 edges then print ('no spanning
tree').
```

The complexity of this algorithm is determined by the complexity of sorting method applied; for an efficient sorting it is $O(|E|\log|E|)$.It also depends on the complexity of the methods used for cycle detection which is of $O(|V|)$ as only $|V|$-1 edges are added to tree which gives a total of O(E logV) time. So complexity of Kruskal's algorithm is is $O(|V+E|\log|V|)$. As V is asymptotically no larger than E, the running time can also be expressed as O(E log V).

Dijkstra has not developed any algorithm for finding minimum spanning tree. It is Prim's algorithm. Dijkstra's algorithm to find the shortest path can be used to find MST also, which is not very popular.

Q.213 Write algorithm for Breadth First Search (BFS) and give the complexity. **(8)**

Ans:

This traversal algorithm uses a queue to store the nodes of each level of the graph as and when they are visited. These nodes are then taken one by one and their adjacent nodes are visited and so on until all nodes have been visited. The algorithm terminates when the queue becomes empty.

Algorithm for Breadth First Traversal is as follows:

```
clearq (q)
visited (v) = TRUE
while not empty (q) do
       for all vertices w adjacent to v do
       if not visited then
       {
              insert (w , q)
              visited (w) = TRUE
       }
delete (v, q);
```

Here each node of the graph is entered in the queue only once. Thus the while loop is executed n times, where n is the order of the graph. If the graph is represented by adjacency list, then only those nodes that are adjacent to the node at the front of the queue are checked therefore, the for loop is executed a total of E times, where E is the number of edges in the graph. Therefore, breadth first algorithm is O (N*E) for linked expression. If the graph is represented by an adjacency matrix, the for loop is executed once for each other node in the graph because the entire row of the adjacency matrix must be checked. Therefore, breadth first algorithm is $O(N^2)$ for adjacency matrix representation.

Q.214 A binary tree T has 10 nodes. The inorder and preorder traversals of T yield the following sequence of nodes:

Inorder : D B H E A I F J C G
Preorder : A B D E H C F I J G

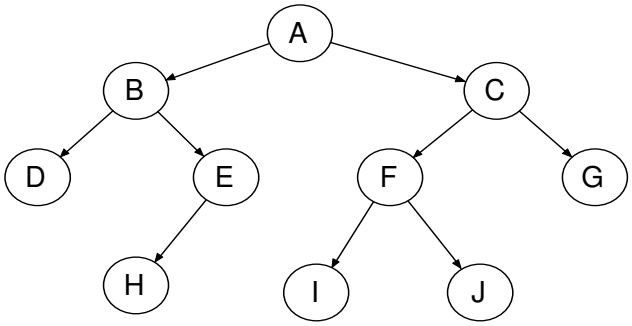
Draw the tree T. **(8)**

Ans:

Given of a tree, the

INORDER: D B H E A I F J C G
PREORDER: A B D E H C F I J G

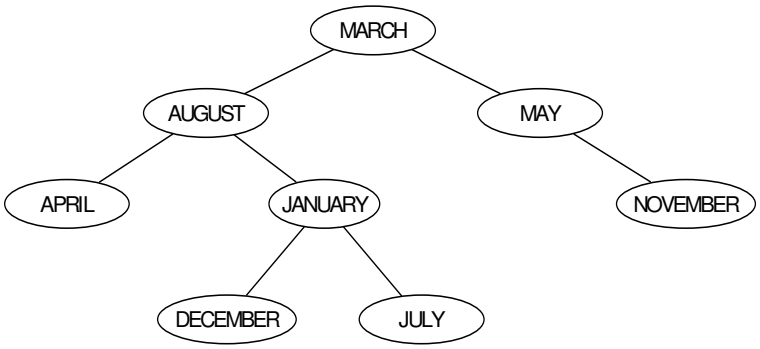
Then the tree of whose traversal is given is as follows



Q.215 Construct AVL tree from the following set of elements
{ March, May, November, August, April, January, December, July } **(8)**

Ans:

The AVL tree from the given set of elements is as follows:
{March, May, November, August, April, January, December, July}



Q.216 Evaluate the following postfix-expression using stack. Show the content of the stack in each step.

6 2 3 + - 3 8 2 / + * 2 $ 3 + **(6)**

Ans:

The content of stack in each iteration will be as follows:

Symbol	Opnd1	Opnd2	Value	Stack
6				6
2				6,2
3				6,2,3
+	2	3	5	6,5
-	6	5	1	1
3	6	5	1	1,3
8	6	5	1	1,3,8
2	6	5	1	1,3,8,2
/	8	2	4	1,3,4
+	3	4	7	1,7
*	1	7	7	7
2	1	7	7	7,2
$	7	2	49	49
3	7	2	49	49,3
+	49	3	52	52

The value returned is 52.

Q.217 Write an algorithm that will split a circularly linked list into two circularly linked lists. **(7)**

Ans:

A function that split a circular linked list into two linked list is as follows:

```
nodeptr splitlist(nodeptr p)
{
nodeptr p1, p2, p3;
p1 = p2 = p3 = p;
if (not p) return NULL;
do {
p3 = p2;
p2 = p2->next; /* advance 1 */
p1 = p1->next;
if (p1) p1 = p1->next; /* advance 2 */
} while (p1);
/* now form new list after p2 */
p3->next = NULL; /* terminate 1st half */
return p2;
} /* splitlist */
```

Q.218 Let A[n] be an array of n numbers. Design algorithms to perform the following operations:

Add (i, y): Add the value y to the i^{th} number in the array

Partial-sum(i) : returns the sum of the first i numbers in the array **(4)**

Ans:

Operation 1

```
add (i ,y)
{
        A[i] = A[i] + y;
 }
```

Operation 2

```
partialsum (i, y)
{
        int sum=0;
        for ( ;  i > 0 ; i--)
                sum = sum + A[i];
```

```
        return sum;
    }
```

Q.219 Calculate the efficiency of Quick sort. **(4)**

Ans:

Efficiency of Quick sort

Assume that the file size n is a power of 2

say $n=2^m$

$m=\log_2 n$ (taking log on both sides)..... (A)

Assume also that the proper position for the pivot always turns out to be exact middle of the sub array. In that case, there will be approx n comparisons (actually n-1) on the first pass, after which the file is split into two sub files each of size n/2. Each of them will have n/2 comparisons and file is again split into 4 files each of size n/4 and so on halving it m times.

Proceeding in this way:-

	No. of comparisons
1. file of size n	1*n=n
2. file of size n/2	2*n/2=n
3. file of size n/4	4*n/4=n
...	
...	
n file of size 1	n*1=n

Total no. of comparisons for entire sort

= n+n+n+.....+n (m times)

= nm

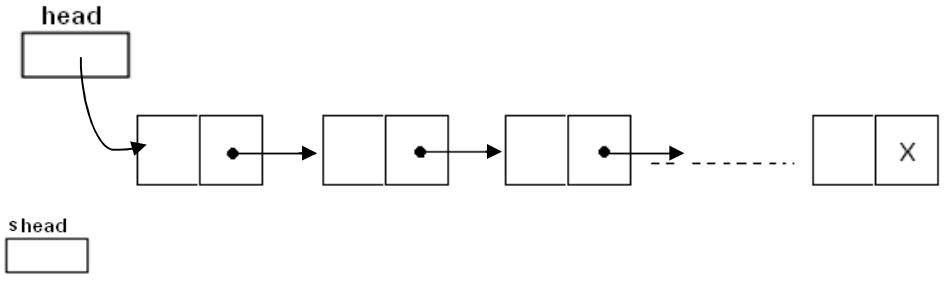
$=n \log_2 n$ (from A)

Therefore the efficiency of Quick sort is **O ($n \log_2 n$).**

Q.220 Formulate an algorithm to perform insertion sort on a linked list. **(8)**

Ans:

Simple ***Insertion sort*** is easily adaptable to singly linked list. In this method there is an array *link* of pointers, one for each of the original array elements. Initially link[i] = i + 1 for 0 < = I < n-1 and link [n-1] = -1. Thus the array can be thought of as a linear link list pointed to by an external pointer *first* initialized to 0. To insert the kth element the linked list is traversed until the proper position for x[k] is found, or until the end of the list is reached. At that point x[k] can be inserted into the list by merely adjusting the list pointers without shifting any elements in the array. This reduces the time required for insertion but not the time required for searching for the proper position. The number of replacements in the link array is O (n).



Each node from the unsorted linked list is to be detached one by one from the front and attached to the new list pointed by s head. While attaching, the value got stored in each node is considered so that it is inserted at the proper position.

```
S head = head
head=head    →next
S head→ next = NULL
While (head!=NULL)
{
P=head
head= head →next
q=S head
while (q→info<P→info & q!NULL)
{
r=q
q=q→next
}
P→next=q
r→next=P
}
```

Q.221 The following values are to be stored in a Hash-Table:
25, 42, 96, 101, 102, 162, 197, 201
Use the Division method of Hashing with a table size of 11. Use the sequential method for resolving collision. **(6)**

Ans:

The given values are as follows: -
25 42 96 101 102 162 197 201
Table size is = 11
Division method of Hashing: -
H (k) = {Hash address range from 0 to m-1. Key (mod) table-size.}
H (25)= 3
H (42) = 9
H (96) = 8
H (101) = 2
H (102) = 3
H (162) = 8
H (197) =10
H (201) = 3
The Hash table is as follows
Hash [0] = [197]
Hash [1] = [NULL]
Hash [2] = [101]
Hash [3] = [25] ← Collision Occurred
Hash [4] = [102] ← Inserted in next available slot
Hash [5] = [201]
Hash [6] = [NULL]
Hash [7] = [NULL]
Hash [8] = [96]
Hash [9] = [42]
Hash [10] = [162]

0	197	
1		
2	101	
3	25	---102(collision)
4	102	---201
5	201	
6		
7		
8	96	
9	42	
10	162	

Q.222 Rewrite your solution to part **(b)** using rehashing as the method of collision resolution. Use [(key+3) mod table-size] as rehash function. **(5)**

Ans: Rehash function [(key+3) mod table size]

H(25) = 28 mod 11
H(42) = 45 mod 11
H(96) = 99 mod 11
H(101) = 104mod 11
H(102) = 105mod 11
H(162) = 165 mod 11
H(197) = 200mod 11
H(201) = 204mod 11
The newly created hash function is as follows:-
Hash[0] = [96]
Hash[1] = [42]
Hash[2] = [162]
Hash[3] = [197]
Hash[4] = [NULL]
Hash[5] = [101]
Hash[6] = [25]
Hash[7] = [102]
Hash[8] = [201]
Hash[9] = [NULL]
Hash[10] = [NULL]

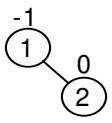
Q.223 How many AVL trees of 7 nodes with keys 1, 2, 3, 4, 5, 6, 7 are there? Explain your answer. **(6)**

Ans:

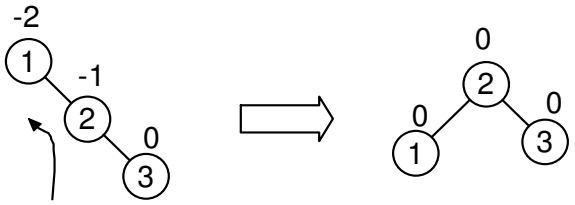
Step 1

No rebalancing required

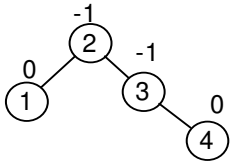
Step 2



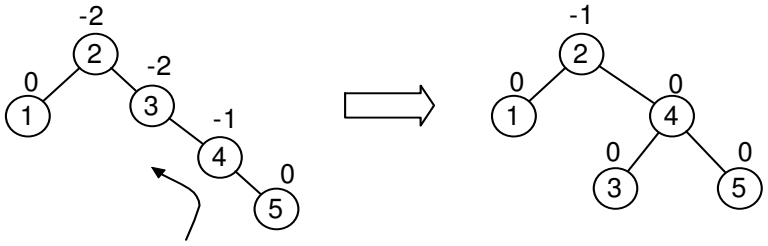
Step 3



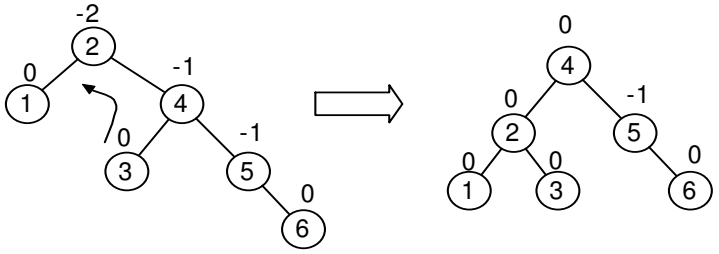
Step 4



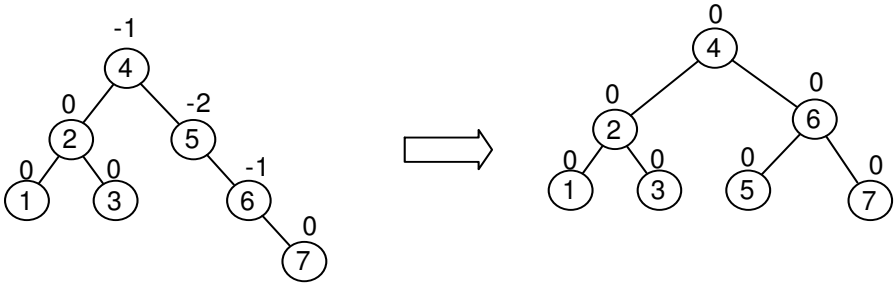
Step 5



Step 6



Step 7



Q.224 Describe the Dijkstra's algorithm for finding a shortest path in a given graph. **(8)**

Ans:

Let G = (v,e) be a simple graph. Let a & z be any two vertices of the graph. Suppose L(x) denotes the label of the vertex z which represents the length of the

shortest path from the vertex a to the vertex z. W_{ij} denotes the weight of the edge $e_{ig} = (V_i, V_j)$

1. let P = Q where p is the set of those vertices which have permanent labels &
T = {all vertices of the graph G}
Set L (a) = 0, L (x) = ∞for all x €T & x≠a

2. Select the vertex v in T which has the smallest label. This label is called the permanent label of v.
Also set P = P U {v} and T = T-{v}
if v=z then L (z) is the length of the shortest path from the vertex a to z and stop.

3. If v ≠z then revise the labels of vertices of T i.e. the vertices which do not have permanent labels. The new label of a vertex x in T is given by
L (x) = min{old L(x),L(v)+w(y,x)}
where w (v,x) is the weight of the edge joining the vertex v & x
If there is no direct edge joining v & x then take w(v,x) = ∞

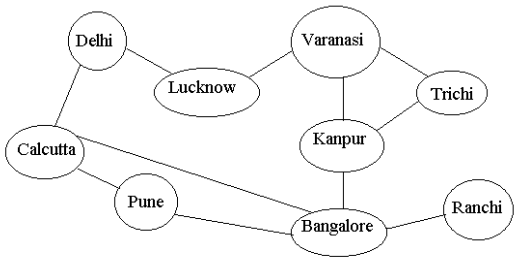
4. Repeat steps 2 and 3 until z gets the permanent label.

Q.225 Explain the difference between depth first and breadth first traversing techniques of a graph. **(4)**

Ans:

Depth-first search is different from Breadth-first search in the following ways:
A depth search traversal technique goes to the deepest level of the tree first and then works up while a breadth-first search looks at all possible paths at the same depth before it goes to a deeper level. When we come to a dead end in a depth-first search, we back up as *little* as possible. We try another route from a recent vertex-the route on top of our stack. In a breadth-first search, we want to back up as *far* as possible to find a route originating from the earliest vertices. So the stack is not an appropriate structure for finding an early route because it keeps track of things in the order opposite of their occurrence-the latest route is on top. To keep track of things in the order in which they happened, we use a FIFO queue. The route at the front of the queue is a route from an earlier vertex; the route at the back of the queue is from a later vertex.

Q.226 Consider the following undirected graph and answer the following questions.



Assume that the edges are ordered alphabetically (i.e. when facing with alternatives, choose the edges in alphabetical order)

(i) List the nodes (cities) of the graph by depth first search starting from **Varanasi.**

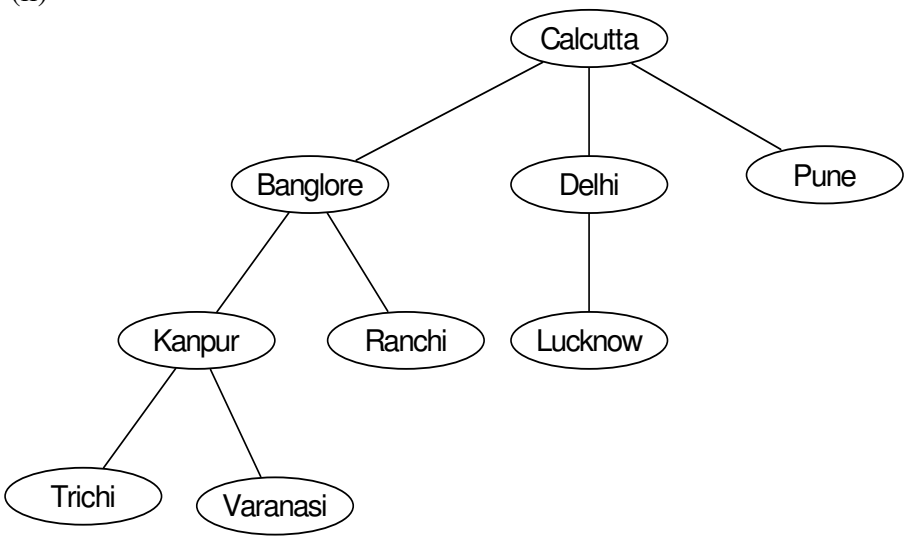
(ii) List the nodes (cities) of the graph by breadth first search starting from **Calcutta**. **(8)**

Ans:

(i) The list of nodes from the graph by performing Depth-First-Search starting from Varanasi is as follows

Varanasi → Kanpur → Bangalore → Calcutta → Delhi → Pune → Ranchi → Lucknow→Trichi

(ii)



The list of nodes from the graph by performing Breadth-First-Search starting from Calcutta is as follows

Calcutta → Bangalore → Delhi → Pune → Kanpur → Ranchi → Lucknow →Trichi →Varanasi

Q.227 Consider the following function

F(int n, int m)

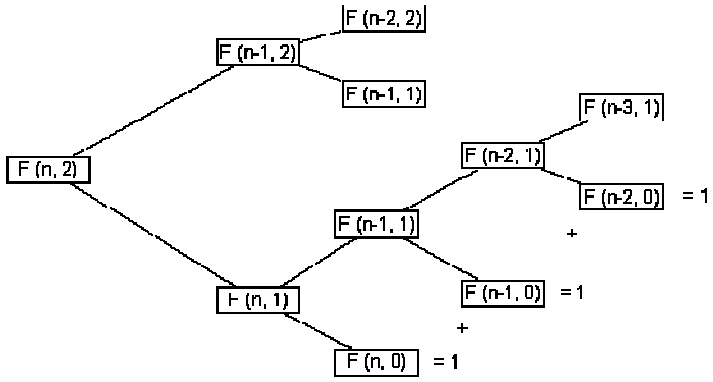
{if n < = 0 OR m<= 0 then return 1 else return (F(n-1, m) + F(n, m-1));}

Now answer the following questions assuming that n and m are positive integers.

(i) What is the value of F(n, 2), F(5, m), F(3, 2), F(n, m)?

(ii) How many recursive calls are made to the function F, including the original call when evaluating F(n, m)? **(4)**

Ans:



The recursion continues till every function call gets its absolute value. Order is

$$\frac{n\,(n-1)}{2} = O(n)$$

(i) The value of
   a. F (n,2) = O(n)
   b. F (5, m) = O (m)
   c. F (3, 2) = O (1).
   d. F (n, m) = O (n). O(m) = O(n.m).

(ii) While evaluating F(n, m) , since it is of order O (n, m) so the number of recursive calls required is of the order n x m.

Q.228 Write a recursive function that has one parameter which is an integer value called x. The function prints x asterisks followed by x exclamation points. Do not use any loops. Do not use any variables other than x. **(4)**

Ans:

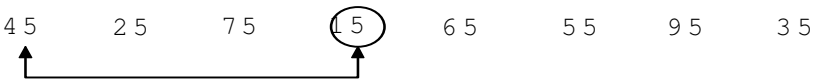
The recursive function printast () for the required task is as follows:-

```
void printast (int a)
{
 if (a > 0)
 {
        printf ("*");
        a--;
        printast (a);
 }
 if (a!=0)
        printf ("!");
}
```

Q.229 Sort the following list using selection Sort. Show each pass of the sort.
45, 25, 75, 15, 65, 55, 95, 35 **(7)**

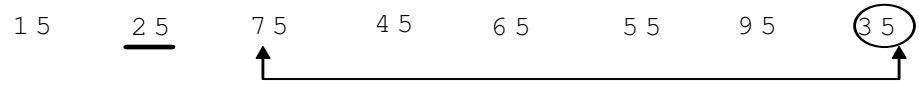
Ans:

The original array is 45, 25, 75, 15, 65, 55, 95, 35



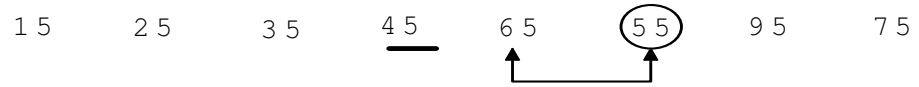
After Pass 1

After Pass 2



After Pass 3

After Pass 4



After Pass 5

After Pass 6

After Pass 7

The array is sorted after 7 passes of selection sort.

Q.230 What is Hashing? Can a perfect Hash function be made? Justify your answer. Explain in brief the various methods used to resolve collision. **(9)**

Ans:

Hashing : Hashing provides the direct access of record from the file no matter where the record is in the file. This is possible with the help of a hashing

function H which map the key with the corresponding key address or location. It provides the key-to-address transformation.

No a perfect hash function cannot be made because hashing is not perfect.

Occasionally, a collision occurs when two different keys hash into the same hash value and are assigned to the same array element. Programmers have come up with various techniques for dealing with this conflict.Two broad classes of collision resolution techniques are: open addressing and chaining.

Open addressing: The simplest way to resolve a collision is to start with the hash address and do a sequential search through the table for an empty location. The idea is to place the record in the next available position in the array. This method is called linear probing. An empty record is indicated by a special value called null. The major drawback of the linear probe method is clustering.

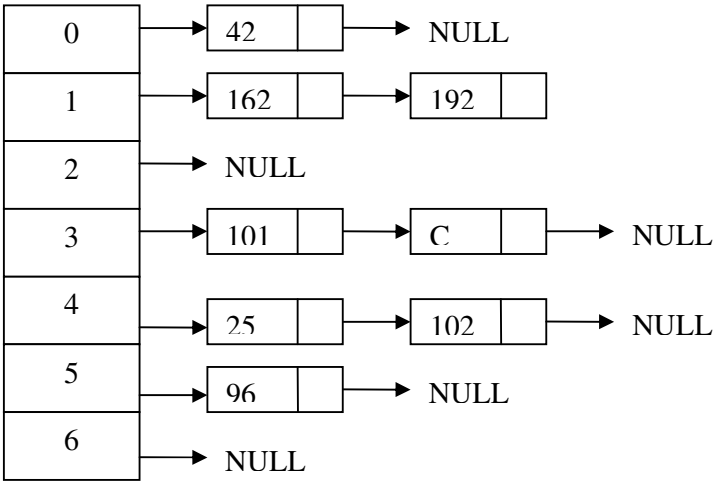
Chaining: In this technique, instead of hashing function value as location we use it as an index into an array of pointers. Each pointer access a chain that holds the element having same location.

Because two entries cannot be assigned the same array element, the programmer creates a linked list. The first user-defined structure is assigned to the pointer in the array element. The second isn't assigned to any array element and is instead linked to the first user-defined structure, thereby forming a linked list.

For example: in a table size of 7

42, 56 both are mapped to index 0 as 42%7=0 and 56%7=0.

25, 42, 96, 101, 102, 162, 197 can be mapped as follows:



Q.231 Define a B tree of order m.
Write algorithms to
(i) Search for a key in B-tree.
(ii) Insert a key in a B-tree.
(iii) Delete a key from a B-tree. **(10)**

Ans:

B Tree of order m

A balanced multiway search tree of order m in which each nonroot node contains at least m/2 keys is called a B-Tree of order m. where order means maximum number of sub-trees.

A B-Tree of order m is either the empty tree or it is an m-way search tree T with the following properties:

(i) The root of T has at least two subtrees and at most m subtrees.
(ii) All internal nodes of T (other than its root) have between [m / 2] and m subtrees.
(iii) All external nodes of T are at the same level.

Algorithm to search for a key in B-tree is as follows:

B-tree search makes an n-way choice. By performing the linear search in a node, correct child Ci of that node is chosen. Once the value is greater than or equal to the desired value is obtained, the child pointer to the immediate left of the value is followed. Otherwise rightmost child pointer is followed. If desired value is obtained, the search is terminated.

B-tree search (x, k) *This function searches the key from the given B-tree*

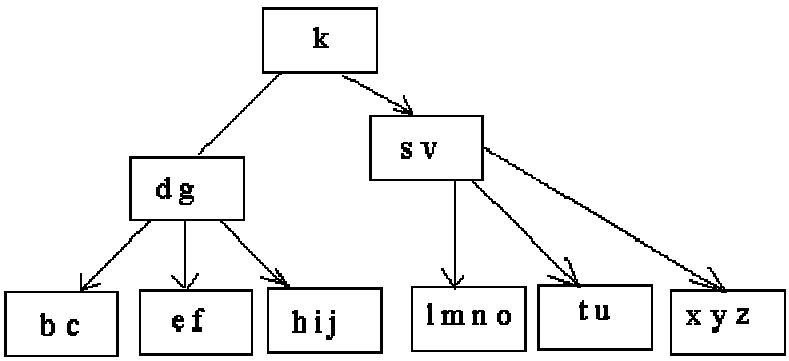
The Disk_Read operation indicates that all references to a given node be preceded by a read operation.

1. i ← 1
2. While ($i \leq n[x]$ and $k > key_i [x]$)
   Set i ← i + 1
3. If ($i <= n[x]$ and $k = key_i [x]$) then
   { return (x, i) }
4. If (leaf [x]) then
   return NIL
   else Disk_read ($c_i [x]$)
   return B-tree search (ci [x], k)

Algorithm to insert a key in B-tree is as follows:

1. First search is done for the place where the new record must be put. As the keys are inserted, they are sorted into the proper order.
2. If the node can accommodate the new record, insert the new record at the appropriate pointer so that number of pointers remains one more than the number of records.
3. If the node overflows because there is an upper bound on the size of a node, splitting is required. The node is split into three parts; the middle record is passed upward and inserted into parent, leaving two children behind. If n is odd (n-1 keys in full node and the new target key), median key int(n/2)+1 is placed in parent node, the lower n/2 keys are put in the left leaf and the higher n/2 keys are put in the right leaf. If n is even, we may have left biased or right biased means one key may be more in left child or right child respectively.
4. Splitting may propagate up the tree because the parent, into which splitted record is added, may overflow then it may be split. If the root is required to be split, a new record is created with just two children.

Q.232 Given the following tree

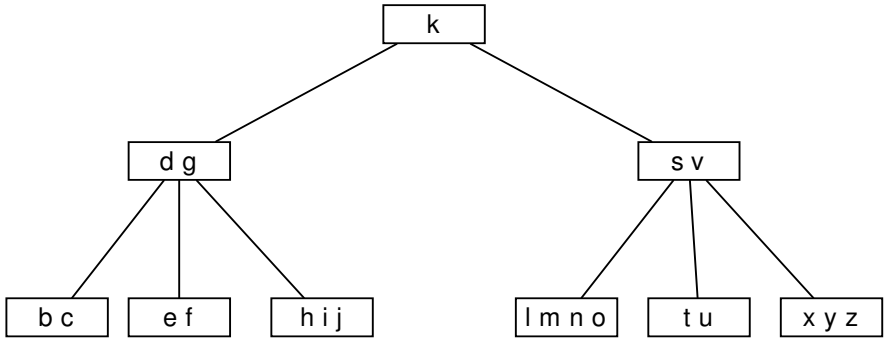


Show how the tree will change when the following steps are executed one after another

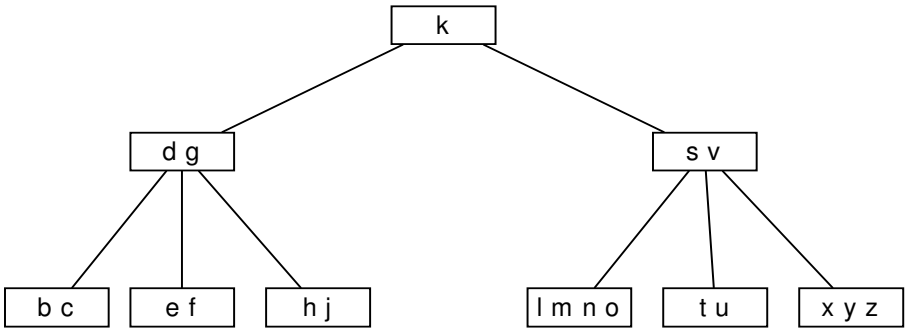
(i) Key 'i' is deleted
(ii) Key 'v' is deleted
(iii) Key 't' is deleted **(6)**

Ans:

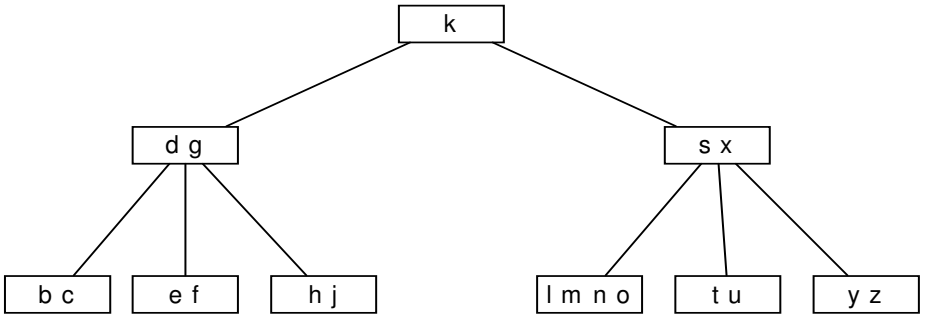
The given tree is:-



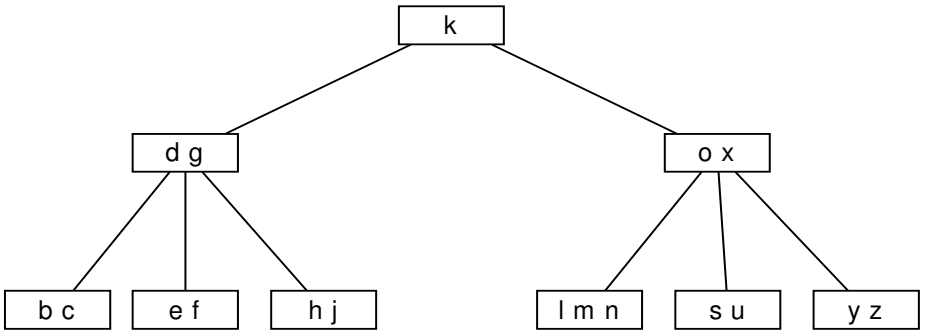
(i) **Key 'i' is deleted**



(ii) **Key 'v' is deleted**



(iii) **Key 't' is deleted**



Q.233 Write an algorithm to reverse a singly linked list. **(4)**

Ans:

The algorithm to reverse a singly link list is as follows:-

```
reverse (struct node **st)
{
      struct node *p, *q, *r;
      p = *st;
      q = NULL;
      while (p != NULL)
      {
        r =q;
        q = p;
        p = p → link;
        q → link = r;
      }
      *st = q;
}
```

Q.234 Consider a 2D array as char A[5] [6] stored in contiguous memory locations starting from location 1000" in column major order. What would the address of a[i] [d]? **(2)**

Ans:

Considering the given char array A[5][6] stored in contiguous memory in column major order; the address of a[i] [d] would be calculated as
base (a) + (d*n1 +i) * size ; where base(a) =1000 (given).
Since it is a char array so size=1 and n1 is the total no of rows
here n1=6 (0,1,2,3,4,5,) Substituting values we get the address of

A[i] [d] = 1000 + d*6 + i

Q.235 What are priority Queues? How can priority queues be implemented? Explain in brief. **(4)**

Ans:

Priority Queues:-

There are some queues in which we can insert items or delete items from any position based on some property. Now, those queues based on the property of priority of the task to be processed are referred to as **priority queues**.

To implement a priority queue we first need to identify different levels of priority and categorize them as the least priority through the most priority level. After having identified the different levels of priority required, we then need to classify each incoming item (job) as to what priority level should be assigned to it. While creating the queues, we need to create n queues if there are n priority levels defined initially. Let suppose there are three priority levels identified as P1, P2, and P3. In this case we would create three queues as Q1, Q2 and Q3 each representing priority levels P1, P2, and P3 respectively. Now every job would have a priority level assigned to them. Jobs with priority level as P1 will be inserted in Q1. Jobs with priority level P2 are inserted in Q2 and so on.

Each queue will follow FIFO behavior strictly. Jobs are always removed from the front end of any queue. Elements in the queue Q2 are removed only when Q1 is

empty and the elements of the queue Q3 are only removed when Q2 is empty and so on.

Q.236 Consider a linked list with n integers. Each node of the list is numbered from '1' to 'n'. Write an algorithm to split this list into 4 lists so that
first list contains nodes numbered 1,5, 9, 13- - -
second list contains nodes numbered 2, 6, 10, 14- - -
third list contains nodes numbered 3, 7, 11, 15- - -
and fourth list contains nodes numbered 4,8, 12, 16- - - **(8)**

Ans:

Algorithm for the above said task is as follows: -

Let us denote the pointer to an item in the original linked list as P
Then the data and consequent pointers will be denoted as
P->data and P->next respectively.

We assume that the functions list1.add (x), list2.add (x), list3.add (x) and list4.add (x) is defined to insert the "item x" into the linked list list1, list2, list3 and list4 respectively.

```
start
set i = 1
while ( P->next != NULL )
 Do
 If (i % 4 = 1)
 List1.add (P->data)
 Else if (i % 4 = 2)
        List2.add (P->data)
        Else if (i % 4 = 3)
               List3.add (P->data)
        Else
               List4.add (P->data)
 P = P->next //increment P to point to its next location
 i++ // increment i to count the next node in sequence.
 Done
 End
```

Q.237 Write a recursive algorithm to find Greatest Common Divisor of two integers with the help of Euclid's algorithm, as per the following formula

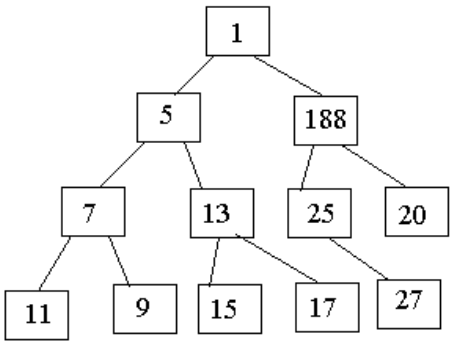
$$GCD(n,m) = \begin{cases} GCD(m,n) \text{ if } n < m \\ m \text{ if } n \geq m \text{ and } n \bmod m = 0 \\ GCD(m, n \bmod m) \text{ otherwise} \end{cases} \quad \textbf{(4)}$$

Ans:

Recursive algorithm to find the GCD of two numbers using Euclid's algorithm is as follows;

```
int gcd (n, m)
{
 if (n < m)
        return (gcd (m, n));
 else if ( n >= m  &&  n % m == 0)
        return (m);
 else
        return (gcd (m, n % m));
}
```

Q.238 Traverse the following binary tree in inorder, preorder and postorder. **(3)**



Ans:

The binary tree traversal is as follows:-

Preorder
1,5,7,11,9,13,15,17,188,25,27,20
Inorder
11,7,9,5,15,13,17,1,25,27,188,20
Postorder
11,9,7,15,17,13,5,27,25,20,188,1

Q.239 Define the following:

(i) Tree (recursive defination)
(ii) Level of a node.
(iii) Height of a tree.
(iv) Complete Binary tree.
(v) Internal Path length. **(2+1+1+2+1)**

Ans:

(i) **Tree (recursive definition)**

A tree is a finite set of one or more nodes such that.
(1) There is a specially designated node called the root.
(2) The remaining nodes are partitioned into n>=0 disjoint sets T1, T2 ... Tn where each of these sets is a tree. T1, T2 ... Tn are called the subtree of the root.

(ii) **Level of a node**

The root is at level 0(zero) and the level of any node is 1 more than the level of its parent.

(iii) **Height of a tree**

The length of the longest path from root to any node is known as the height of the tree.

(iv) **Complete Binary tree**

A complete binary tree can be defined as a binary tree whose non leaf nodes have nonempty left and right sub tree and all leaves are at the same level.

(v) **Internal Path length**

It is defined as the number of node traversed while moving through one particular node to any other node in the tree.

Q.240 What is an AVL tree? Explain how a node can be inserted into an AVL tree. **(6)**

Ans:

AVL Tree

An AVL tree is a binary tree in which the difference in heights between the left and the right subtree is not more than one for every node.

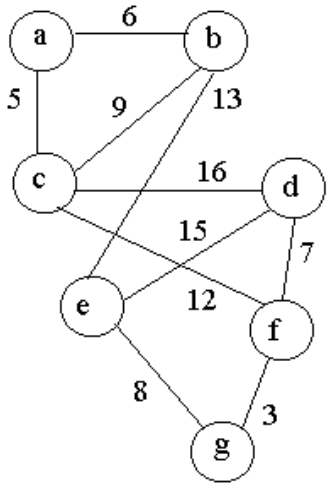
We can insert a node into an AVL tree by using the insertion algorithm for binary search trees in which we compare the key of the new node with that in the root and then insert the new node into the left subtree or right subtree depending on whether it is less than or greater than that in the root. Insertion may or may not result in change in the height of the tree. While inserting we must take care that the balance factor of any node not changes to values other than 0, 1 and -1. A tree becomes unbalanced if and only if

(i) The newly inserted node is a left descendant of a node that previously had a balance of 1.

(ii) The newly inserted node is a right descendent of a node that previously had a balance of -1.

If the balance factor of any node changes during insertion than one of the subtree is rotated one or more times to ensure that the balance factor is restored to correct values.

Q.241 Describe Kruskal's algorithm to find minimum spanning trees.



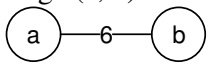
Apply Kruskal's algorithm for the above graph. **(10)**

Ans:

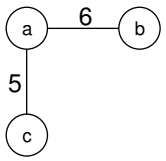
Applying Kruskal's Algorithm starting from the node a

Step 1: Start from node a, Add to tree T

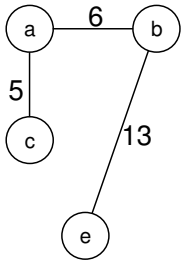
Step 2: Edge (a, b) cost=6 add to T



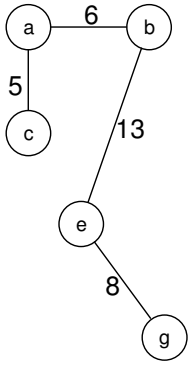
Step 3: Edge (a, c) cost=5 add to T



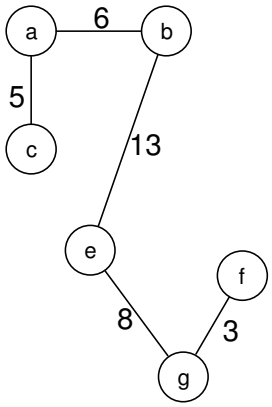
Step 4: Edge (b,e) cost=13 add to T



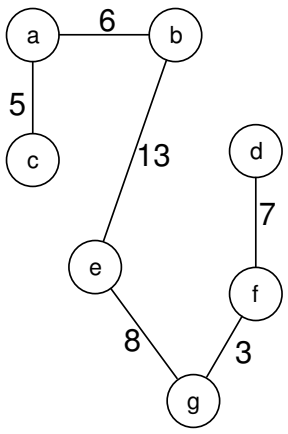
Step 5: Edge (e, g) cost=8 add to T



Step 6: Edge (g, f) cost=3 add to T



Step 7: Edge (f, d) cost=7 add to T



This is the minimum spanning tree of the given graph.

Q.242 Differentiate between
(i) Top-down and Bottom-up programming?
(ii) Structured and Modular programming. **(6)**

Ans:

(i) Top-down and Bottom up programming.

Top down method is also known as step-wise refinement. Here the problem is first divided into main tasks. Each of these is divided into subtasks and so on. These subtasks should more precisely describe how the final goal is to be reached. Whereas the Bottom-up programming is just the opposite of the top-down approach. In this technique, all small related tasks are grouped into a major task and the main task becomes the main program.

(ii) Structured and Modular programming.

Structured programming means the collection of principles and practices that are directed toward developing correct programs which are easy to maintain and easy to understand. It should read like a sequence from beginning to end instead of branching from later statements to earlier ones and back again. Whereas in modular programming the modularity of a system can be represented by a hierarchical structure which has a single main module at level 1 and gives a brief description of the system. At level 2 the main module refers to a number of sub program modules the give a more detailed description of the system and so on. An important aspect in this hierarchical structuring process is the desire to understand a module at a certain level independently of all other modules at the same level.

Q.243 How is recursion handled internally. Explain with the help of a suitable example. **(4)**

Ans:

Internally, each recursive call to a function needs to store the intermediate values of the parameters and local variables in a run time stack. The general algorithm for any recursive procedure contains the following steps:

1. Save the parameters, local variables and return address.

2. If the base criterion has been reached, then perform the final computation and go to step 3, otherwise perform the partial computation and go to step 1 with reduced parameter values (initiate a recursive call).

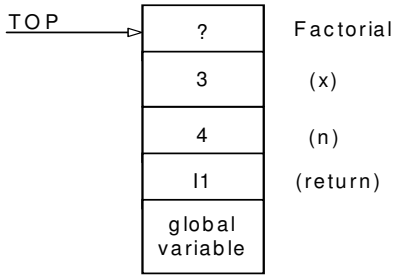
3. Restore the most recently saved parameters, local variables and return address. Go to this return address.

All parameters is accessed by their positions relative to the top of the stack. Whenever a subroutine is started the variables are pushed onto the stack and whenever it completes execution, the variables are popped off the stack. For ex. consider the factorial function.

```
Factorial (int n)
                {
  int x;
  if n = 0
         return (1);
  x = n - 1;
  return (n * factorial (x));
}
```

let the initial call is Y = factorial (4)

At first time, 4 locations are put in the run time stack



And on each subsequent calls the intermediate values of all these variables are pushed till the condition reaches where n=0. and then the contents are popped one by one and is returned to the place of the precious call. This continues until the control is back to the first call.

Q.244 Write an algorithm to add two polynomials using linked list. **(12)**

Ans:

Algorithm to add two polynomials using linked list is as follows:-

Let p and q be the two polynomials represented by linked lists

```
1. while p and q are not null, repeat step 2.
2. If powers of the two terms ate equal
       then if the terms do not cancel
       then insert the sum of the terms into the sum Polynomial
                    Advance p
                    Advance q
   Else if the power of the first polynomial> power of second
   Then insert the term from first polynomial into sum polynomial
                    Advance p
   Else insert the term from second polynomial into sum polynomial
                    Advance q
3. copy the remaining terms from the non empty polynomial into the
   sum polynomial.
```

The third step of the algorithm is to be processed till the end of the polynomials has not been reached.

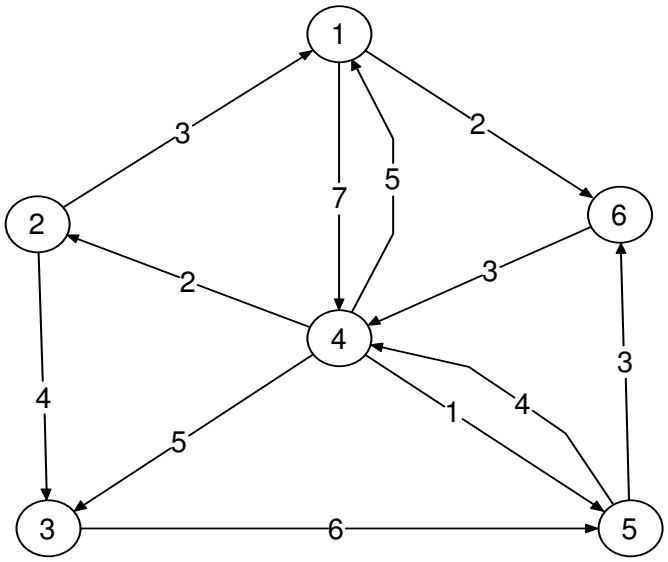
Q.245 Execute Dijkstra's algorithm on the following graph with vertices numbered 1 to 6. The graph is given in the form of an adjacency list.

i.	: (6, 2), (4, 7)
ii.	: (1,3), (3,4)
iii.	: (5,6)
iv.	: (2,2) (3,5), (1,5), (5,1)
v.	: (4,4), (6,3)
vi.	: (4,3)

Here (3,4) in vertex 2's adjacency list means that vertex 2 has a directed edge to vertex 3 with a weight of 4. Assuming the source vertex to be 2, find the shortest distance and the path to all the remaining vertices. **(16)**

Ans:

The source vertex is 2 (given).



Step 1

P=Q T = {1, 2, 3, 4, 5, 6}
L(2) = 0Starting Vertex
L(x) = ∞ for all x Є T, x≠2

Step 2

V=2 (vertex 2 has smallest label)
P = {2} T = {1, 3, 4, 5, 6}
Calculate new labels for all vertices of T.
L (1) = min (∞, 0+3) = 3
L (3) = min (∞, 0+4) = 4
L (4) = min (∞, 0+∞) = ∞
L (5) = ∞
L (6) = ∞

Step 3

V=1 (Permanent label L (1) = 3)
P = {2, 1} T = {3, 4, 5, 6}
Calculate new labels for all vertices of T.

L (3) = min (4, 3+∞) = 4
L (4) = min (∞, 3+7) = 10
L (5) = min (∞, 3+∞) =∞
L (6) = min (∞. 3+2) = 5

Step 4

V=3 (Permanent label L (3) = 4)
P = {2, 1, 3} T = {4, 5, 6}
Calculate new labels for all vertices of T.
L (4) = min (10, 4+∞) = 10
L (5) = min (∞, 4+6) = 10
L (6) = min (5. 4+∞) = 5

Step 5

V=6 (Permanent label L (6) = 5)
P = {2, 1, 3, 6} T = {4, 5}
Calculate new labels for all vertices of T.
L (4) = min (10, 5+3) = 8
L (5) = min (10, 5+∞) = 10

Step 6

V=4 (Permanent label L (4) = 8)
P = {2, 1, 3, 6, 4} T = {5}
Calculate new labels for all vertices of T.
L (5) = min (10, 8+1) = 9

The vertex 5 got its permanent label and all the vertices got their respective permanent labels as shown in the table below.

Vertex (v)	Permanent Label L (v)
1	3
3	4
4	8
5	9
6	5

Now, assuming the source vertex to be 2, the shortest distance and the path to all remaining vertices are as follows: -

Vertex 1

Shortest distance = 3
Path is 2→1

Vertex 3

Shortest distance = 4
Path is 2 → 3

Vertex 4

Shortest distance = 8
Path is 2→1 → 6→4

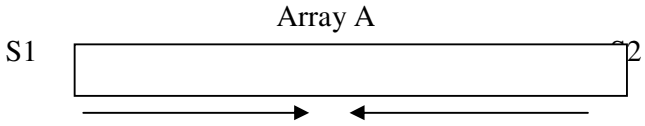
Vertex 5

Shortest distance = 9
Path is 2→1→ 6→4→5

Vertex 6

Shortest distance = 5
Path is 2→1→ 6

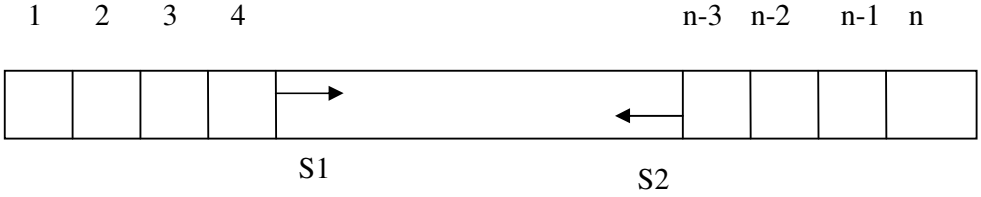
Q.246 Two stacks are implemented using an array. What should be the initial values of top for the two stacks? Arrows show the direction of growth of the stack.



How can we implement m stacks in a contiguous a memory. Explain how the stacks will grow and indicate the boundary condition. Write push and pop programs for such a scenario. **(10)**

Ans:

Two stacks s1 and s2 can be implemented in one array A [1, 2... N] as shown in the following figure



We define A[1] as the bottom of stack S1 and let S1 "grow" to the right and we define A[n] as the bottom of the stack S2 and S2 "grow" to the left. In this case, overflow will occur only S1 and S2 together have more than n elements. This technique will usually decrease the number of times overflow occurs. There will separate push1, push2, pop1 and pop2 functions to be defined separately for two stacks S1 and S2. These will be defined as follows:-

```
initial volume of top for
stack S1 is top1=0 and
for stack S2 is top 2=n+1
S1. Push 1(x)
{
if (top 1+1==top2)
print if ("Stack S1 is overflowing")
Else
{
top 1++
A[top1]= x
}
}
S1.POP()
{
if (top1==0)
print if ("stack S1 is empty")
return (-1)
}
return (A[top--])
}
S2.push2(x)
}
if (top1+1==top2)
print if ("S2 is over flowing")
else
{
top2–
A [top2]=x
```

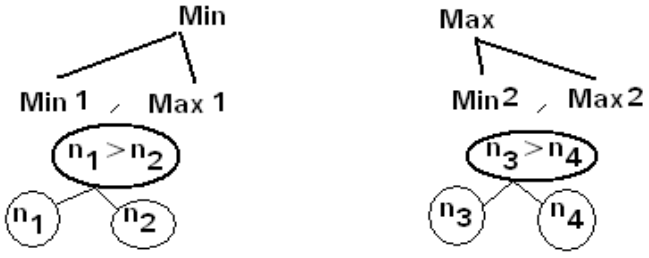
```
}
}
S2.POP2()
{
if (top2==n+1)
}
print if ("stack is empty")
return(-1)
}
else
return( A[top2++])
}
```

Q.247 How can we modify almost any algorithm to have a good best case running time? **(4)**

Ans: Guess an answer
Verify that it is correct, in that case stop.
Otherwise run the original algorithm.
OR
Check whether the input constitutes an input at the very beginning
Otherwise run the original algorithm.

Q.248 Sketch an algorithm to find the minimum and the maximum elements from a sequence of n elements. Your algorithm should not take more than (3n/2)-2 number of comparisons where n is a power of 2. **(6)**

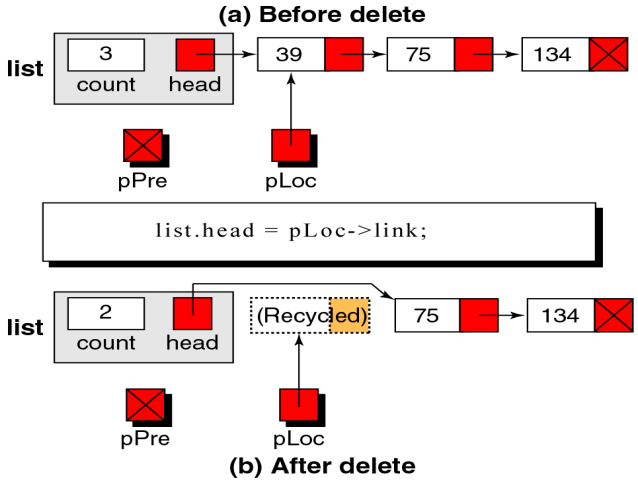
Ans:
Rather than processing each element of the input by comparing it against the correct minimum and maximum, at the cost of 2 comparisons per element, we process elements in pairs. We compare pairs of elements from the input first with each other, and then we compare the smaller to the current minimum and the larger to the current maximum, at the cost of 3 comparisons for every 3 elements.
If n is off, we set both the minimum and maximum to the value of first element and the process the rest of the elements in pairs.
If n is even, we perform 1 comparison on the first 2 elements to determine the initial values of minimum and maximum, and the process the rest of the elements in pairs.So if n is odd, we perform 3[n/2] comparisons.
If n is even, we perform one initial comparison followed by 3(n-2)/2 comparisons for a total of 3n/2-2. Thus in either, number of comparisons are almost 3[n/2]



Q.249 Write an O(1) algorithm to delete a node p in a singly linked list. Can we use this algorithm to delete every node? Justify. **(7)**

Ans:

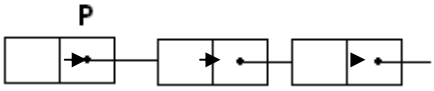
One deletion operation consists in deleting a node at the beginning of the list and returning the value stored in it. This operation is implemented by a member function deleteFromHead() given below. In this operation, the information from the first node is temporarily stored in local variable and then head is reset so that what was the second node becomes the first node. In this way, the former first node can be deleted in constant O(1) time.



Deleting the first node, from a linked list pointed by 'head' can be done is O(i) time

head = head →next

Deleting a note that follows a node with address P can also be done in O(i) time



P →next = (P →next) → next

But a node with address P cannot be deleted in O(i) time. It takes linear time.

```
//Delete the first node
deleteFromHead(struct node *q,int num)
{
        struct node *temp;
        temp=*q;
        *q=temp→link;
        free(temp);
}
```

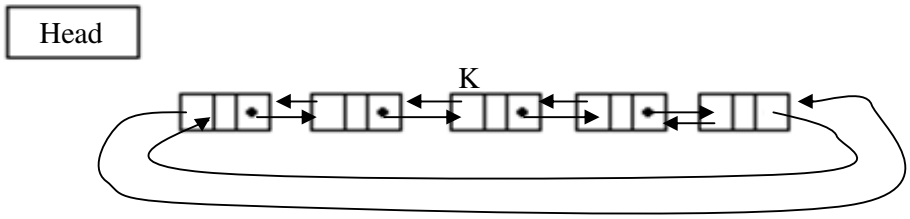
We can use this algorithm to delete every node by calling it recursively. The best case is when the head node is to be deleted, which takes O(1) time to accomplish. The worst case is when the last node needs to be deleted because in that case it will take O(n) performance.

Q.250 Suggest an efficient sorting algorithm to sort an array of n large records, while minimizing number of exchanges. **(2)**

Ans:

Selection sort

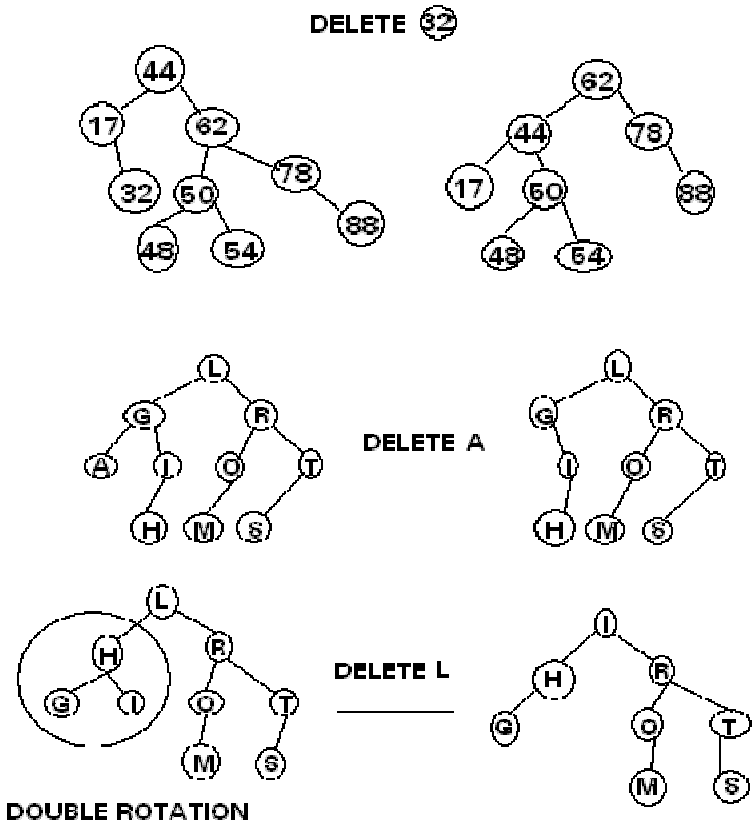
In all cases: $n^2/2$ comparisons and n exchanges execution time is not sensitive to original ordering of Input data.



Suppose K is the node after which the list is to be split. Then
head2 = k →front
k→front = head
head → back = k
P = head2
while (P→front:=head)
P=P→ front
P→front=head2
head2→back=P

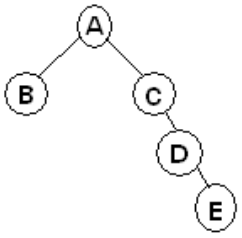
Q.251 Give an AVL tree for which the deletion of a node requires two double rotations. Draw the tree and explain why two rotations are needed? **(10)**

Ans:



Q.252 A funny tree is a binary tree such that, for each of its nodes x, the number of nodes in each sub tree of x is at most 2/3 the number of nodes in the tree rooted at x. Draw the tallest funny tree of 5 nodes. **(7)**

Ans:



Q.253 Explain the term step-wise refinement. (3)

Ans:

Step **Wise Refinement**

Refinement is a process of elaboration. Here one begins with a statement of function that is defined at a high-level abstraction. That is the statement describes the program/function conceptually but provides no information about internal working. Refinement causes the programmer to elaborate on the original statement providing more and more detail. In a stepwise refinement, at each step of refinement one or several instructions of the given program are decomposed into more detailed instructions. The successive refinement terminates when all the instructions are expressed in terms of atomic expressions of the language.

Q.254 What is the difference between top-down and bottom-up, programming? (4)

Ans: Top-down and Bottom-up Approaches

In Top-down programming approach, we start by identifying the major modules of the program, decomposing them into their lower modules and iterating until the desired level of detail is achieved. This is also termed as Step Wise Refinement; starting from an abstract design, in each step the design is refined to a more concrete level until we reach a level where no more refinement is needed and the design can be implemented directly, whereas in Bottom-up programming approach, we identify modules that are required by programs and decide how to combine these modules to provide larger ones; to combine those to provide even larger ones, and so on, till we arrive at one big module which is the whole of the desired program.

In Bottom-up approach, we need to use a lot of intuition to decide exactly what functionality a module should provide. This is not the case in Top-down approach.

Q.255 What do you mean by complexity of an algorithm? Derive the asymptotic time complexity of a non recursive, binary search algorithm. (7)

Ans:

The term complexity is used to describe the performance of an algorithm. Typically performance is measured in terms of time or space. Space complexity of an algorithm is the amount of memory is needed to run the algorithm. Time complexity of an algorithm is the amount of computer time it needs to run the algorithm. Most

common notation for describing time complexity 0(f(n)), where n is the input size and f is a function of n. An algorithm ~ 0(f(n)) means there exists N such that for all n > N the run time of the algorithm is less than c.f(n), where c is a constant.

Binary search can be applied on a sorted array to search for a particular item. Given array positions A[l], the search is conducted in the following way.

```
Binsearch (x, l,n)
{
L=l;
U = n;
While (L <= U)
{ mid = (L + U) / 2;
If (A[mid] = x)
Return.True;
        Else
If (A[mid ] > x)
L = mid + 1;
    Else
U = mid - 1;
  }
return False;
  }
```

Thus at each state the array is halved into two equal parts. In the worst case the algorithm terminates when the portion of the array considered is of length 1, Thus if n = 2 , in the worst case the while loop will have to repeat k times. Where k = log n. Thus asymptotic complexity is O(log n).

Q.256 Assume the declaration of multidimensional arrays A and B to be,
A (-2:2, 2:22) and B (1:8, -5:5, -10:5)
(i) Find the length of each dimension and the number of elements in A and B. **(3)**
(ii) Consider the element B[3,3,3] in B. Find the effective indices E_1, E_2, E_3 and the address of the element, assuming Base (B) = 400 and there are W = 4 words per memory location. **(4)**

Ans:
(i) A (-2:2,2:22), B (1:8, -5:5, -10:5) length of 1st dimension in A= 2-(-2)+1
= 2+2+1=5
length of 2nd dimension **in A** =22-2+1=21
LI = length of 1st dimension in B=8-l+l=8
L2 = length of 2nd dimension in B= 5-(-5)+1=11 L3= length of 3rd dimension in B=5-(-10)+l=16 No. of elements in *A*= 5*21=105
No. of elements in B=8*11*16=1408

(ii) B[3,3,3], Base(B)=400, W=4
El= 3-1=2
E2=3-(-5)=8
E3=3-(-10)=13
Address of B[3,3,3]=400+4(493)
=2372

E1L2=2*11=22
E1L2+E2=22+8=30 (E1L2+E2)L3=30*16=480
(E1L2+E2)L3+E3=480+13= 493

Q.257. (i) what is meant by the terms 'row-major order' and 'column-major order'? (2)
(ii) Can the size of an array be declared at runtime? (2)
(iii) The array DATA [10, 15] is stored in memory in 'row - major order'. If base address is 200 and element size is 1. Calculate the address of element DATA [7, 12]. (3)

Ans: (i) Storing the array column by column is known as column-major order and storing the array row by row is known as row-major-order. The particular order used depends upon the programming language, not the user. For example consider array A (3,4) and how this array will be stored in memory in both the cases is shown below

	(1,1)			(1,1)	
	(2,1)	Column1		(1,2)	Row1
	(3,1)			(1,3)	
	(1,2)			(1,4)	
	(2,2)	Column2		(2,1)	Row2
	(3,2)			(2,2)	
	(1,3)			(2,3)	
	(2,3)	Column3		(2,4)	
	(3,3)			(3,1)	Row3
	(1,4)			(3,2)	
	(2,4)	Column4		(3,3)	
	(3,4)			(3,4)	

(ii) No, the size of an array can't be declared at run time, we always need to mention the dimensions of an array at the time of writing program i.e before run time
(iii) Base address=200
Element Size=l
Address of element DATA [7,12]
= 200+[(7-l)*15+(12-l)]*l
= 200+[6*15+11]
= 200+[90+11]=301
Address of DATA [7,12]=301

Q.258 Write an algorithm to insert a node after a given node in a linear linked list. **(7)**

Ans:
Suppose we are given a value of LOC where LOC is the location of the node 'A' in the linked list.
The following algorithm inserts an 'ITEM' into LIST (given Linked list) so that 'TTEM' follows **node** 'A'.

```
1. If AVAIL = NULL , Then write overflow and exit
2. set NEW = AVAIL and AVAIL=:link[AVAIL] [remove first node from
AVAIL LIST]
3. set INFO[NEW]=ITEM [copies new data into new node]
4. if LOC=NULL then [insert as first node]
     set LINK[NEW]= START and START = NEW

     else [insert after node with

     location LOC]

     set LINK[NEW]=LINK[LOC] and LINK[LOC]=NEW

     [end of if structure]
5.    exit
```

Q.259 Show how the following polynomial can be represented using a linked list. **(4)**

$$7x^2y^2-4x^2y+5xy^2-2$$

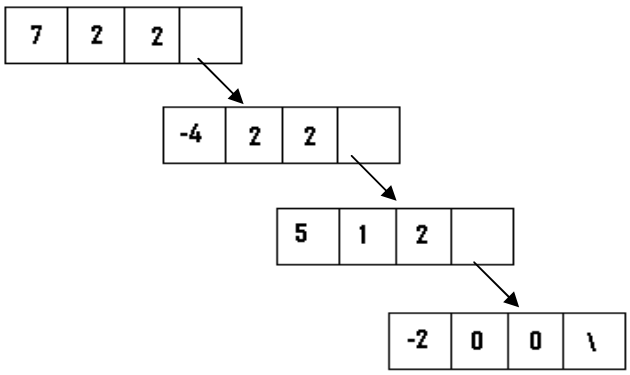
Ans:

Representation of Polynomial using Linked List

Each node will have four parts(as shown below)

Coeff	Power of'x'	Power of 'y'	Next address

The polynomial is



Q.260. What is the advantage of doubly linked list over singly linked list? **(3)**

Ans:

Advantages of the doubly linked list over singly linked list

1 A doubly linked list can be traversed in two directions; in the usual forward direction from the beginning of the list to the end, or in the backward direction from the end of the list to the beginning of the list.

2 Given the location of a node 'N' in the list, one can have immediate access to both the next node and the preceding node in the list.

3 Given a pointer to a particular node 'N', in a doubly linked list, we can delete the Node 'N' without traversing any part of the list. Similarly, insertion can also be made before or after 'N' without traversing the list.

Q.261 What is a binary tree? Write an algorithm for the preorder traversal of a binary tree using stacks. , (7)

Ans:

A binary tree 'T' is defined as

A finite set of elements, called nodes, such that: either (i) or (ii) happens:

(i) T is empty (called the 'null tree' or 'empty tree')

(ii) T contains a distinguished node R, called the 'root' of T and the remaining nodes of 'T' form an ordered pair of the disjoint binary tree Tl and T2.

If 'T' contains a root 'R' then the two trees Tl and T2 are called, the 'left' and 'right' sub trees of R**,** respectively.

Algorithm for the pre order traversal of a binary tree using a stack

A binary tree T is in memory, an array 'STACK' is used to temporarily hold the addresses of the nodes. "**PROCESS**" is the operation that will be applied to each node of the tree.

```
(1) Set TOP=1, STACK [1] =
 NULL and PTR= ROOT
 [initially push null onto
 Stack and initialize PTR]
(2) repeat steps (3) to (5) while PTR != NULL
(3) apply PROCESS to INFO[PTR]
(4)  [right child ?]
if RIGHT[PTR] != NULL then[push on STACK]
Set TOP=TOP+1 band STACK[TOP]=RIGHT[PTR]
[End of IF Structure]
(5)  [Left Child ?]
if LEFT[PTR] != NULL then:
set PTR= LEFT[PTR]
else [POP from STACK]
set PTR=STACK[TOP] and TOP=TOP+1
[End of IF structure]
(6)  Exit
```

Q.262. A binary tree T has 9 nodes. The in order and preorder traversals of T yield the following sequences of nodes:

In Order: EACKFHDBG

Pre order:FAEKCDHGB

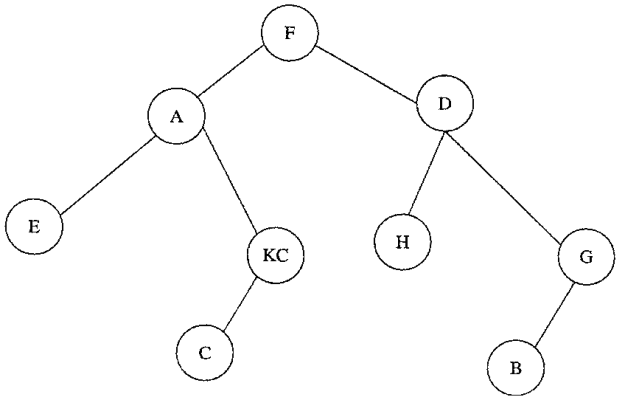
Draw the tree T. Also give the yield of the post order traversal. (7)

Ans:

Inorder :EACKFHDBG

Preocder : E A E K C D H G B

Tree 'T' is



Postorder Traversal **-.ECKAHBGDF**

Q.263 Write short notes on the following:

(i) Threaded binary tree.
(ii) Buddy systems.
(iii) B - trees.
(iv) Minimum spanning tree. (14)

Ans:

(i) **Threaded binary tree** : Consider the linked representation of a binary tree 'T. Approximately half of the entries is the pointer fields 'LEFT' and 'RIGHT' will contain null elements. This space may be more efficiently used by replacing the null entries by some other type of information. Specially we will replace certain null entries by special pointers which points to nodes in the tree. These special pointers 4 are called 'threads' and binary trees with such pointers are called 'threaded trees'.

There are two major types of threading: - one way threading and two way threading.

In the one way threading of T, a thread will appear in the right field of a node and will point to the next node in the inorder traversal of T and in the two way threading of T, a thread will also appear in tl left field of a node and will point to the preceding node in the inorder traversal of T. There is another kind of one way threading which corresponds to the preorder traversal of T.

(ii) **Buddy systems**: A method of handling the storage management problem is kept separate free lists for blocks of different sizes. Each list contains free blocks of only one specific size. For example, memory contains 1024 words then it might be divided into fifteen blocks, one block of 256 words, t blocks of 128 words, four blocks of 64 words, and 8 blocks of 32 words. Whenever sport is requested the smallest block is whose size is greater than or equal to the size needed to reserve for example a block of 97 words is field by a block of size of 128 but in this method there are several limitation, first, space is wasted due to internal fragmentation, second, a request of block size 300 cannot be filled. The largest size is maintained is 256, the source of this problem is that free space are never combined

A variation to this scheme is buddy system and is quite useful. In Buddy systems several free list constituets of various sized blocks are maintained. Adjacent free blocks are smaller size may be removed from list combined into free blocks of larger size, and placed on the larger size free list. These larger blocks can be used intact to satisfy a request for a large amount of memory or they can be split once into their smallest constituents to satisfy several smaller request.

(iii) **B-tree :** A B-tree is a balanced m-way tree. A node of the tree may contain many records or key and pointers to children. It is also known as the balanced sort tree. It finds its use in external sorting. It is not a binary tree.

B-tree of order m has following properties:

(1) each node has a maximum of m children and minimum of m/2 children or any number from 2 to maximum.

(2) The no. of keys in a node is one less than its no of children. The arrangement

$P_O\ K_I\ P_I$ .... $K_n\ P_n$

Δ Δ Δ

$T_o\ T_I\ T_n$ each T_I is a m-way tree

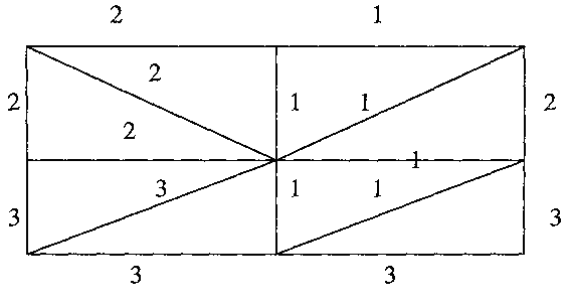
(3) The keys 'in a node Ki — K_n are arranged in sorted order $K_1 < K_2 — < K_n$. All the keys present in the subtree pointed to by a pointer P_i $K_i.P_i.K_{i+1}$ are greater than

(4) When a new key is to be inserted into a full node, the key is split into two nodes and the key with the median value is inserted in the parent node. In case the parent node is a root, a new root is created.

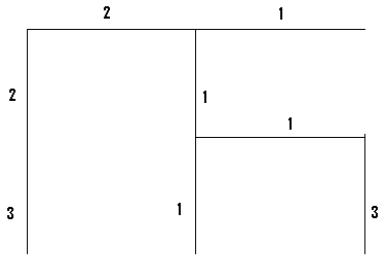
(5) All the leaves are on the same level i.e. there is no empty subtree above the level of the leaves. All the normal nodes of B-tree (Except 'root' and terminal nodes) have between m/2 and m children.

(iv) **Minimum Spanning Tree :** Given a weighted graph G, it is often desired to create a spanning tree T for G, such that the sum of weights of the edges in T is the least. Such a tree is called a minimum spanning tree and represents the cheapest way of connecting all the nodes in G . There are no. of techniques to create a minimum spanning tree for a weighted graph. For ex. Kruskal's algo. Prim's algorithm.

Example :-The given connected weighted graph G is



One of the minimum spanning trees of the graph G is:-



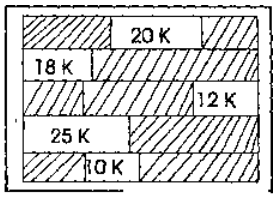
Minimum cost of the spanning tree = 14

Q.264. What is memory allocation? Explain the first fit and best fit algorithms for storage allocation with suitable examples. (7)

Ans:

Memory Allocation : It is the technique of allocating memory storage to program in order that the program can be run.

First fit algorithm: The question asked for examples not algorithms the first fit algorithm may be explained.



Suppose the memory allocation is as follows: suppose ▨ indicates allocated block & ▭ denotes a free block. The size of the free blocks are indicated. Suppose the current requirement is 8K. Then the first fit algorithm will allocate it from the block of 20K.

The advantage ; it is part disadvantage : fragmentation.

```
P=freeblock
aloc= null
 q=null;
while (p!=null && size (p) < n )
{
Q=p;
 p=next(P)
}/* end while */
if (p!=null){/* there is block large enough*/
s=size(p);
alloc=p+s-n; /* alloc contain the address of the designed
block*/
if(s==n)
/* remove the block from the free list*/
if (q==null)
freeblock=next(p);
else
 next(q)=next(p);
else
/* adjust the size of tile remaining free block*/
size(p)=s-n;
}/* end if*/
```

The Best fit algorithm : The best fit method obtains the smallest free block whose size is greater than or equal to n. An algorithm to obtain such a block by traversing the entire free list follows. We assume that the memsize is the total no. of words in memory.

The Best fit algorithm allocates the memory from the block that fits the requirement best i.e. the block that has size greater than the requirement, but no other block of lesser size can meet the requirement. Thus for the above problem the allocation will be made from block of size 10 K. Advantage : fragmentation is reduced

Disadvantage : more time.

```
p=freeblock;  /*p is used to traverse the free list */
 q==null;    /* q is one block behind p*/
 r=null;    /* r points to the desired block*/
rq=null;   /* rq is one block behind r */
 rsize=memsize+l; /* rsize is the size of the block at r */
alloc = null; /* alloc will point to block selected */
while(!=null)
{
 if(size(p)>= n && size(p)<rsize)
```

```
{
/* we have found a free block closer in size */
r=p;
rq=q;
rsize=size(p);
}
/* END IF */
/* continue traversing the free list */
q=P;
p=next (p);
}
/* end while */
if{r!=null)
{
/* there is block of sufficient size */
 alloc = r + r size - n ;
if (r size = =n){
/* remove the block from the free
list */ if(rq= = null )
freeblock = next (r );
else
next(rq)= next( r );
else size(r) = rsize- n ;
} /* end if */
```

Q.265. Consider the following graph

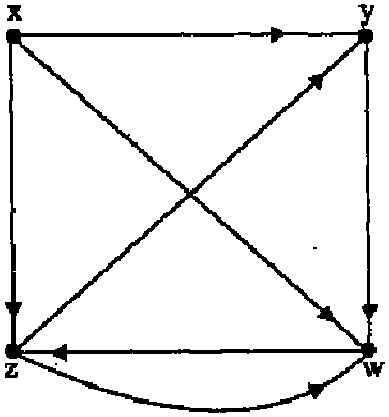


Fig 2

Let the nodes be stored in memory in an array G as :G; X, Y, Z, W

(i) Find the adjacency matrix A of the graph G.
(ii) Find the path matrix P of G using powers of the adjacency matrix -A.
(iii) Is G strongly connected? (3+3+1)

Ans:

Adjacency Matrix 'A' =

(i) =

	X	Y	Z	W
X	0	1	1	1
Y	0	0	0	1
Z	0	1	0	1
W	0	0	1	0

(ii) =

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$A^2 = \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$A^2 = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix}$$

$$A^3 = A^2.A = \begin{bmatrix} 0 & 1 & 1 & 2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$= \begin{bmatrix} 0 & 1 & 2 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix}$$

$$A^4 = A^3.A = \begin{bmatrix} 0 & 1 & 2 & 2 \\ 0 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 \end{bmatrix} \begin{bmatrix} 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$A^4 = \begin{bmatrix} 0 & 2 & 2 & 3 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 2 \\ 0 & 1 & 1 & 1 \end{bmatrix}$$

$A+A^2+A^3+A^4 =$ $\begin{bmatrix}0&1&1&1\\0&0&0&1\\0&1&0&1\\0&0&1&0\end{bmatrix}$ + $\begin{bmatrix}0&1&1&2\\0&0&1&0\\0&0&1&1\\0&1&0&1\end{bmatrix}$

+ $\begin{bmatrix}0&1&2&2\\0&1&0&1\\0&1&1&1\\0&0&1&1\end{bmatrix}$ + $\begin{bmatrix}0&2&2&3\\0&0&1&1\\0&1&1&2\\0&1&1&1\end{bmatrix}$

$= \begin{bmatrix}0&5&6&8\\0&1&2&3\\0&3&3&5\\0&2&3&3\end{bmatrix}$

Path matrix 'P' = $\begin{bmatrix}0&1&1&1\\0&1&1&1\\0&1&1&1\\0&1&1&1\end{bmatrix}$

(iii) No, G is not strongly connected.

Q.266. What is a hash function? Describe any three hash functions. (7)

Ans:

Hash function : This is the function from the set 'K' of keys into the set 'L' of memory addresses.

H : K→ L

These are used to determine the address of a record with the use of some key. Such a function 'H' may not yield distinct values: it is possible that two diff keys Kl and K2 will yield the same hash address;
This situation is called collision and some method must be used to resolve it.

Examples

1. Division method: Choose a number 'm' larger than the number 'n' of keys in K: (The number m is usually chosen to be a prime number). The hash function 'H' is defined by
H(K)= k (mod m)
Where K(mod m) denotes the remainder when 'K' is divided by m.

2. Midsquare **method:** The key 'K' is squared, then the hash function 'H' is defined by
H (K) =1 Where 1 is obtained by deleting digits from both ends of K^2.

3. Folding Method: The key **'K'** is partitioned into a no. of parts $k_1, k_2, \ldots k_T$, Where each part except , possibly the last, has the same number of digits as the required address. Then the parts are added together, ignoring the last carry, that is H (k) = $k_1, k_2, \ldots k_T$,. Where the leading digits carries, if any, are ignored.

Q.267 Write an algorithm for bubble sort. What is the asymptotic time complexity of bubble sort in the worst case. (7)

Ans:

Algorithm for bubble sort:

Let 'A' be a linear array of elements and temp be the variable for interchanging the position of the elements.

```
1.   Input 'n' elements of an array 'a'.
2.   initialize i=0
3.   repeat trough step 6 while (i<n)
4.   set j=0
5.   repeat through step 6 while (j<n-i-l)
6.   if (a[j]>a|j+l])
       (i) temp=a[j]
       (ii) a[j]=a[j+l]
       (iii) a[j+l]=temp
7.   display the sorted
elements of array 'a'
8.   Exit.
```

In this sorting technique, each element is compared with its adjacent element. If the first element is larger than the second one then the position of the elements are interchanged, otherwise it is not changed. Then next element are compared with its adjacent element and the same process is repeated for all the elements in the array. During the first pass the same process is repeated leaving the last element. During the second pass the second last element occupies the second last position. The same process is repeated until no more elements are left for the comparison. Finally the array is sorted. One may use a Hag to check whether there is any interchange in a particular pass. If there is no interchange in a pass, the array has already got sorted, and the algorithm can terminate.

Time complexity in worst case: in worst case, the array will be in reverse sorted order and the no. of comparisons are calculated as given below

$F(n) = l+2+3+4+$——— $+ (n-l)$

$= n (n-l)/2 = O (n^2)$

Q.268 Apply Kruskal's algorithm to find a minimum spanning tree of the graph in the following figure-Display execution of each step of the algorithm. (8)

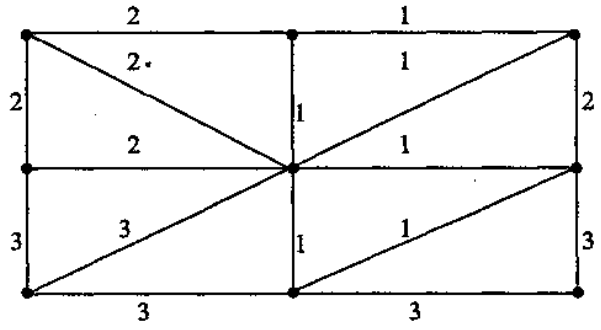
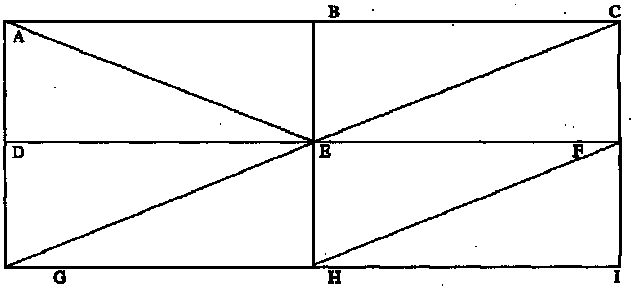


Fig 3

Describe the insertion sort algorithm. Compute its asymptotic time complexity for worst case and average case. (6)

Ans:

The given graph is name the vertices as shown



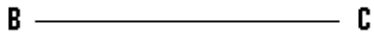
To find minimum spanning tree of the given graph :-

Edges :	AB	BC	AD	BE	CF	AE	CE	DE	EF
Weight :	2	1	2	1	2	2	1	2	1
Edges :	DG	EH	FI	GH	HI	GE	HE		
Weight :	3	1	3	3	3	3	1		

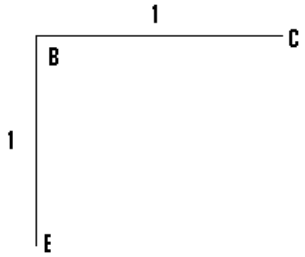
Edges in increasing order of weights

Edges:	BC	BE	CE	EF	EH	HF	AB	AD	CF	AE	DE	DG	FI	GH	HI	GE
Weight:	1	1	1	1	1	1	2	2	2	2	2	3	3	3	3	3
Add:	yes	yes	no	yes	yes	no	yes	yes	no	no	no	yes	yes	no	no	no

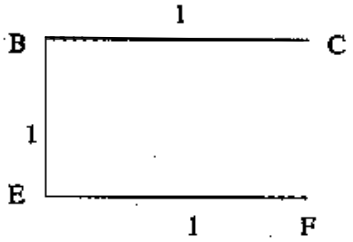
Step 1 Add 'BC'



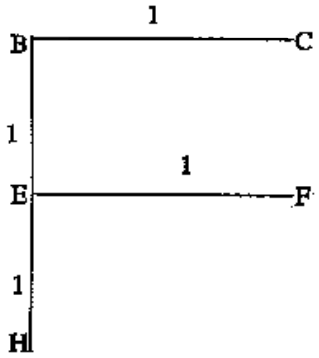
Step 2 Add 'BE'



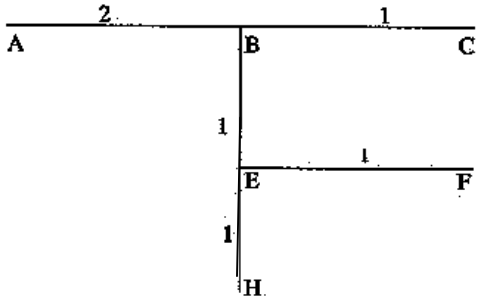
Step 3 Add 'EF'



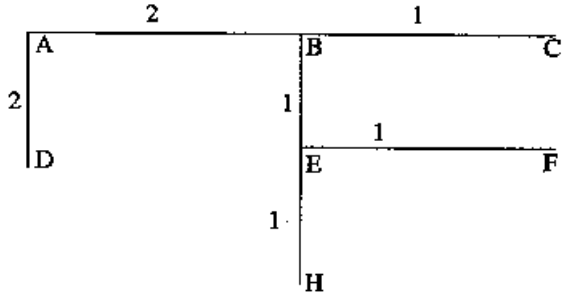
Step 4 Add 'EH'



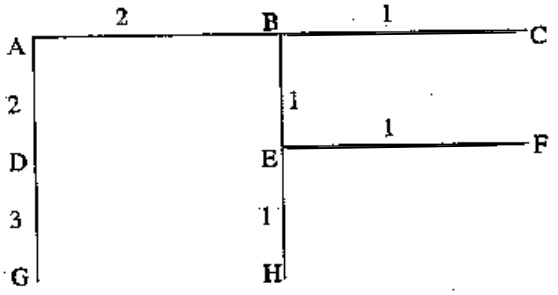
Step 5 Add 'AB'



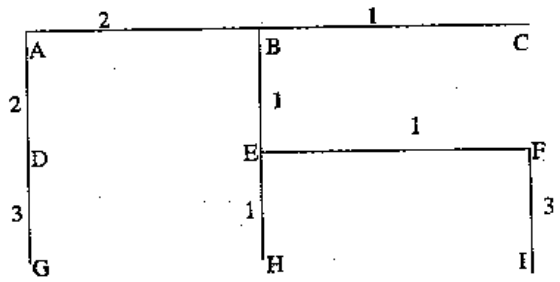
Step 6 Add 'AD'



STEP 7 Add **'DG'**



STEP 8 Add 'FI'



Cost of the spanning Tree= 1+2+2+1+3+1+3+1=14 This is the required spanning tree of the given graph.

Q.269 Define a heap. Describe the algorithm to insert an element into the heap. (2+5)

Ans:

Insertion Sort Algorithm:

```
       Consider an array 'A' with 'N'
elements
1.   set A[0]=-infinity [initialize sentinel element]
2.   Repeat steps 3 to 5 for K=2,3..,.......N
3.  set TEMP=A[K] and PTR=K-1 4   Repeat while TEMP <A[PTR]
(a) set A[PTR+1]=A[PTR] [moves element forward]
(b) set PTR=PTR-1 [End of loop]

5.   set A[PTR+1]=TEMP [inserts elements in proper place]
[End of step2 loop]

6.   Return
```

Complexity of insertion sort

Worst case: The worst case occurs when the -array 'A' is in reverse order and the inner loop must use the max number K-l of comparisons. Hence

$F(n)=l+2+.....+(n-l)=n(n-l)/2=0(n^2)$

Average case: In Average case there will be approximately (K-l)/2 comparisons in the inner loop. Accordingly, for the average case,

$F(n)=l/2+2/2+..... ...+(n-l)/2=n(n-l)/4=0(n^2)$

Q.270 Construct the heap showing the insertion of each of the following elements in separate figures. 44, 30, 50, 22, 60; 50 (7)

Ans:

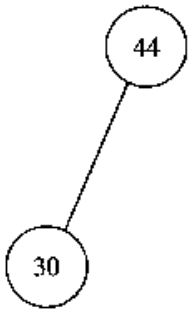
Heap : Suppose 'H' is a complete binary tree with 'n' elements. Then 'H' is called a Heap, as a 'maxheap', if each node 'N' of 'H' has the following property:
The value at 'N' is greater than or equal to the value at each of the children of N. Accordingly, the value at N is greater than or equal to the value at any of the descerdants of N.
Algorithm to insert an element in to the heap: Consider a heap 'H' with 'N' elements is stored in the array TREE and an ITEM of information is given. This procedure inserts 'ITEM' as a new element of 'H'.'PTR' gives the location of ITEM as it rises in the tree, and 'PAR' denotes the location of the parent of 'ITEM'.

```
1.   [ Add new node to 'H' and initialize PTR ] set N=N+1 and
PTR=N
2.   [ Find location
to insert ITEM ]
Repeat steps 3 to 6
while PTR>1
3.   set PAR= [PTR/2] (floor operation)
[Location of parent node] If ITEM<= TREE
[PAR] then step 4 else go to step 5.
4.   set TREE
[PTR]=ITEM and
return End of If
structure
5.  set TREE [PTR]= TREE [PAR]        [moves node down]
6   set PTR
=PAR//» updates
PTR
      //» End of
step2 loop
7.   //» Comments Assign
ITEM as the root of H] set
TREE [1] = ITEM
8.   Return
```

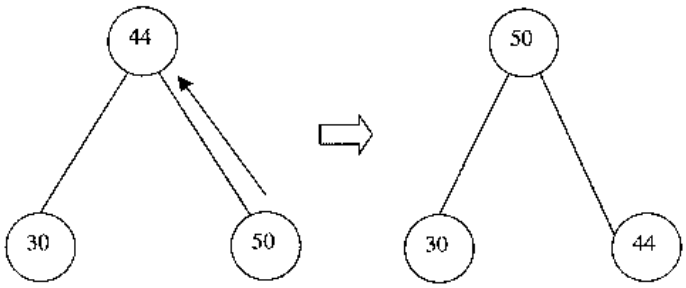
10. (b) Construction of heap :

Step 1 inserting 44

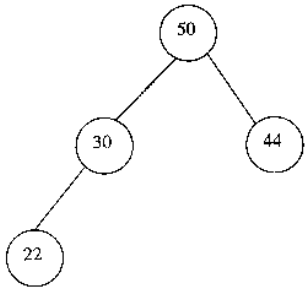
Step 2 Inserting 30



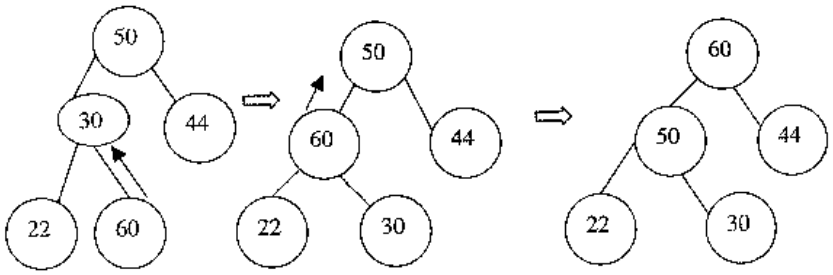
Step 3 Inserting 50



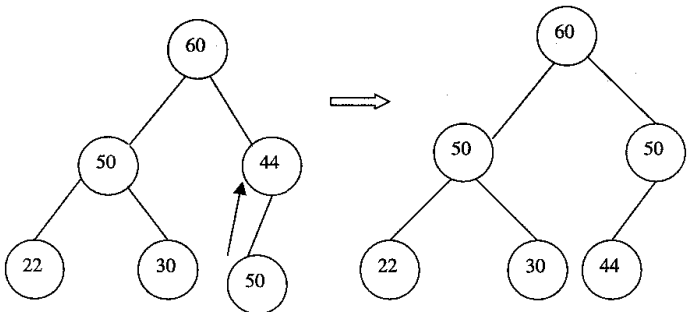
Step 4 inserting 22



Step 5 inserting 60



step 6 inserting 50



This is the required heap.

Q.271 Give the equivalent postfix expression for the following expressions:

(i) (A-B)/ ((D+E)*F) (ii) ((A+B)/D)↑((E-F)*G) (3+4)

Ans:

Convert the following infix expressions into postfire using a stack. Show at each step the stack contents and the output string.

(i) ((A-B)/((B+E)*F)) Add "(" to the starting and")"at the end of expr.

	Symbols scanned	Stack	Expression P
1	(	(	
2	(	((	
3	A	((	A
4	-	((-	A
5	B	((-	AB
6	)	(	AB-
7	/	(/	AB-
8	(	(/(	AB-
9	(	(/((	AB-
10	D	(/((	AB-D
11	+	(/((+	AB-D
12	E	(/((+	AB—DE
13	)	(/(	AB-DE+
14	*	(/(*	AB-DE+
15	F	(/(*	AB-DE+F
16	)	(/	AB-DE+F*
17	)		AB-DE+F*/

The postfix expression is P=AB-DE+F*/

(ii) The given expression is(((A+B)/D) ^ ((E-F)*G)) Add "(' to the start and ") at the end of the expression.

	Symbols scanned	Stack	Expression P
1	(	(	
2	(	((	
3	(	(((	
4	A	(((	A
5	+	(((+	A
6	B	(((+	AB
7	)	((	AB+
8	/	((/	AB+
9	D	((/	AB+D
10	)	(	AB+D/
11	↑	(↑	AB+D/
12	(	(↑(	AB+D/
13	(	(↑ ((	AB+D/
14	E	(↑((	AB+D/E
15	-	(↑ ((-	AB+D/E
16	F	(↑ ((-	AB+D/EF
17	)	(↑(	AB+D/EF-
18	*	(↑ (*	AB+D/EF-
19	G	(↑(*	AB+D/EF-G
20	)	(↑	AB+D/EF-G*
21	)		AB+D/EF-G*
The post fix expression 'P'= AB+D/EF-G*↑			

Q.272 Suppose a queue is maintained by a circular array QUEUE with N = 12 memory cells. Find the number of elements in QUEUE if

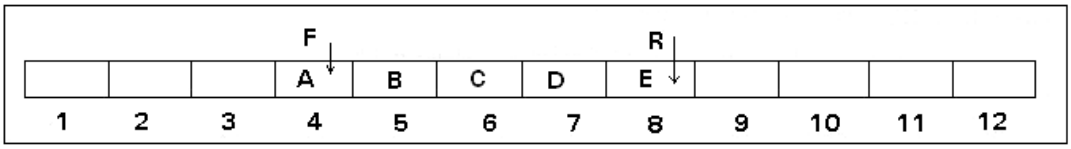
(i) Front = 4, Rear =8.
(ii) Front = 10, Rear = 3.
(iii) Front = 5, Rear = 6 and then two elements are deleted. (2+2+3)

Ans:

N=12

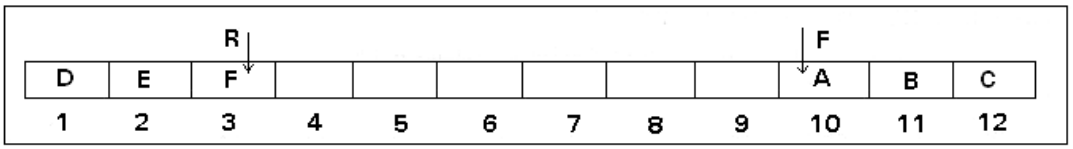
(i) Front=4, Rear=8

No of elements = 8-4 + 1 = 5

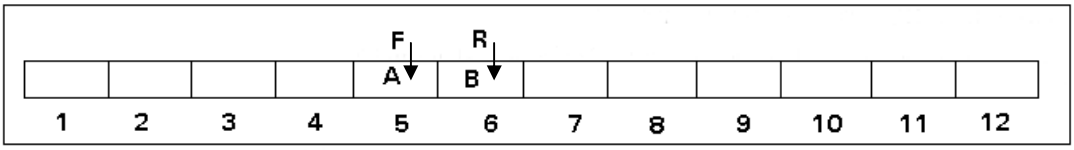


(ii)Front = 10, Rear = 3

No of elements = 6



(iii) Front = 5, Rear = 6



After deletion of two elements in the queue = 0

Q.273 Suppose we wish to partition the square roots of the integers from 1 to 100 in to two piles of fifty numbers each, such that the sum of the numbers in the first pile is as close as possible to the sum of the numbers in the second pile. If you could use minimum computer time to answer this question, what computations would you perform on the computer in that time? **(5)**

Ans :

According to question, sum of square roots in two piles should be as close as possible. For that we can add square roots of odd numbers. In one pile and square roots of even numbers in another pile. Since for natural numbers also if we want to divide into two piles according to nearest equal sum requirement above solution will work i.e.1+3+……+99, 2+4+ ……..+100.Square root is a strictly increasing function for positive numbers. So result holds for square root also.

Computations

1. Check whether no. is odd or even dividing by two, module is zero or not.
2. Add the variable sum computation time for n nos. It will require θ(n) time.

Q.274 Farely fractions of level one is defined as sequence (0/1, 1/1). This sequence is extended in level two to form a sequence (0/1, ½, 1/1), sequence (0/1, 1/3, ½, 2/3, 1/1) at level three, sequence (0/1, ¼, 1/3, ½, 2/3, ¾, 1/1) at level four, so that at each level n, a new fraction (a+b) / (c+d) is inserted between two neighbour fractions a/c and b/d only if c+d <= n. Devise a procedure, which for a number n entered by the user creates – by constantly extending it – a linked list of fractions at

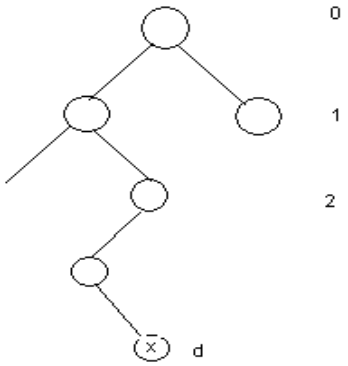
level n. **(8)**

Ans :

```
List{int info;
     int info;
     list*next;};
Farely fractions(n){
listI0,I1,x;
info1(I1)=00;
info2(I1)=1;
next(I1)=I2;
info1(I2)=1;
info(I2)=1;
next(I2)=NULL;
for(i=2;i<=n;i++){
   x=head[L];
   while(x!=nil&&info1(x)+info1(next)(xL)<=i){
      listI;
info(Ij)=info2(xo)+info2(next,x0));
info(Ij)=info1(x)+info1(next);
list insert(L,x);
x=next(x)   }}}
```

Q.275 Suppose that a Binary Search Tree is constructed by repeatedly inserting distinct values in to the tree. Argue that the number of nodes examined in searching for a value in the tree is one plus the number of nodes examined when the value was first inserted in to the tree. **(7)**

Ans :
Let us consider an element x to insert in a binary search tree. So for inserting the



element x first at level 0,then level 1 and suppose upto level (d-1). While examine at (d-1) level, x might have less or more in comaprison to element at (d-1). Then we insert x either left or right. In both cases no of examined code will be d. Now suppose we want to search for x, this time again traverses the same path as we traverse. While inserting the element, we stop at (d-1) the level but for searching we examine node at dth level also i.e. the node containing x. Thus number of node examine while inserting are d while incase of searching it is d+1 i.e. one more than while inserting, hence the result.