Biyani's Think Tank

Concept based notes

Data Structure

(MCA)

Nitasha Jain
Assistant Professor
Department of IT
Biyani Girls College, Jaipur

Published by :
Think Tanks
Biyani Group of Colleges

Concept & Copyright :
©Biyani Shikshan Samiti
Sector-3, Vidhyadhar Nagar,
Jaipur-302 023 (Rajasthan)
Ph : 0141-2338371, 2338591-95 • Fax : 0141-2338007
E-mail : acad@biyanicolleges.org
Website :www.gurukpo.com; www.biyanicolleges.org

Edition : 2012

Leaser Type Setted by :
Biyani College Printing Department

Preface

I am glad to present this book, especially designed to serve the needs of the students. The book has been written keeping in mind the general weakness in understanding the fundamental concepts of the topics. The book is self-explanatory and adopts the "Teach Yourself" style. It is based on question-answer pattern. The language of book is quite easy and understandable based on scientific approach.

Any further improvement in the contents of the book by making corrections, omission and inclusion is keen to be achieved based on suggestions from the readers for which the author shall be obliged.

I acknowledge special thanks to Mr. Rajeev Biyani, *Chairman* & Dr. Sanjay Biyani, *Director* (*Acad.*) Biyani Group of Colleges, who are the backbones and main concept provider and also have been constant source of motivation throughout this endeavour. They played an active role in coordinating the various stages of this endeavour and spearheaded the publishing work.

I look forward to receiving valuable suggestions from professors of various educational institutions, other faculty members and students for improvement of the quality of the book. The reader may feel free to send in their comments and suggestions to the under mentioned address.

Author

Syllabus

DATA STRUCTURE: Basic data structures such as arrays, stack and queues and their applications, linked and sequential representation. Linked list, representation of linked list, multi linked structures. Trees: definitions and basic concepts, linked tree representation, representations in contiguous storage, binary trees, binary tree traversal, searching insertion and deletion in binary trees, heap tree and heap sort algorithm, AVL trees.

Graphs and their application, sequential and linked representation of graph – adjacency matrix, operations on graph, traversing a graph, Dijkstra's algorithm for shortest distance, DFS and BFS, Hashing. Searching and sorting, use of various data structures for searching and sorting, Linear and Binary search, Insertion sort, Selection sort, Merge sort, Radix sort, Bubble sort, Quick sort, Heap Sort.

Reference Books

1.Data Structures in C/C++, Tanenbaum, PHI
2.Data Structures in C/C++, Horowitz, Sawhney.

Introduction

Q1. What is Data Structure?

Ans: A data structure is an arrangement of data in a computer's memory or even disk storage. It has a different way of storing and organizing data in a computer so that it can used efficiently. Different kind of Data structure is used in different application. For example, B-tree is widely used in implementation of databases.

Q2. Name different type of data structures?

Ans: There are basically two types of data structures
1 Linear data Structure
2 Non Linear data Structure

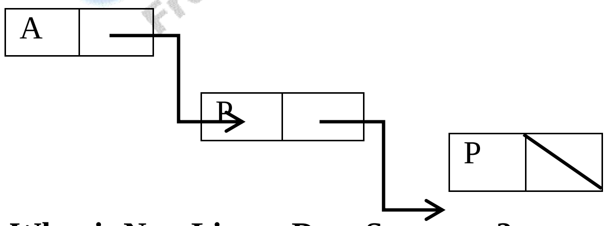
Q.3 What is Linear data Structure?

Ans: Linear data structure organize there elements in a linear fashion, where data elements are attached one after another. They are organized in a way similar to how the computer's memory is organized.
There are 2 ways of representing linear structure in memory:-

1) To have the linear relationship between the elements represented by means of sequential memory locations. For Example :- ARRAY

A	**P**	**P**	**L**	**E**
101	**102**	**103**	**104**	**105**

2) To have the linear relationship between the elements represented by means of pointers and links. For Example:- LINKED-LIST



Q.4 What is Non-Linear Data Structure?

Ans:- Non Linear Data Structures are constructed by attaching a data element to several other data elements in such a way that it reflects a specific relationship among them
For Example :- N-dimensional Array, Trees And Graphs

Arrays

Q.1 What is an Array?

Ans An Array can be defined as a data structure having fixed size sequenced collection of elements of same data type and which can be referenced through same variable name. An Array is a name given to a group of similar quantities.

Declaring an array

Type variable_name [length of array];

For example:-

double height[10];
float width[20];
int min[9];
int myArray[5] = {1, 2, 3, 4, 5}; //declare and initialize the array in one statement

In C language array starts at position 0. C language treats array name as the pointer to the first element and an iten in the array can be accessed through its index.

Q.2 How to read and write data in an array?

Ans

```
            #include<stdio.h>
            void main ()
            {
                   int array[10], n;

    // Reading an array
   printf(“Enter size of an array”);
    scanf ("%d", &n);
    printf(“Enter array elements”);
    for (i=0;i<n;i++)
  scanf ("%d", &array[i]);

 // writing an array
  printf(“array elements are”);
  for (i=0;i<n;i++)
  printf ("%d", array[i]);
}
```

Q.3 How to insert data in an array at a particular position?

Ans To insert a value into an array, you need to shift every value after the location you want to insert. the logic is very simple: start from the end and copy the previous item to the current element until you hit the location you want to insert.

Q.4 Write code to insert data in an array at a particular position

Ans

```
#include<stdio.h>
Void main ()
{
       int array [10], n elem, pos;

   // Reading an array
    printf (“Enter size of an array”);
    scanf ("%d", &n);
   printf (“Enter array elements”);
   for (i=0;i<n;i++)
  scanf ("%d", &array[i]);

      //Inserting an element at specified position
      printf (“Enter  element to be inserted”);
      scanf ("%d", &elem);
      printf(“Enter position where to insert the element”);
      scanf ("%d", &pos);
      pos--;
      for (i<n;i>=pos;i--)
            a[i+1]=a[i];
     a[pos]=elem;
      n++;

       // writing an array
              printf (“array elements are”);
      for (i=0;i<n;i++)
              printf ("%d", array[i]);
}
```

Q.5 How to delete an element from specified position from an array?

Ans : To delete a value from an array, you need to shift every value after the location you want to delete. The logic is very simple: start from the position you want to delete and copy the previous item to the current element until end of the array.

Q .6 Write code to delete data from an array from a particular position?

Ans :

```
#include<stdio.h>
void main ()
{
     int array[10], n elem ,pos;

     // Reading an array
     printf("Enter size of an array");
     scanf ("%d", &n);
     printf("Enter array elements");
     for (i=0;i<n;i++)
          scanf ("%d", &array[i]);

    //deleting  an element at specified position
   printf("Enter position from where to delete the element");
     scanf ("%d", &pos);
    pos--;
    for (i=pos ; i<n ;i++)
        a[i]= a[i+1];
        n--;

     // writing an array
          printf("array elements are");
     for (i=0;i<n;i++)
          printf ("%d", array[i]);
}
```

Q.7 How to merge two sorted arrays?

Ans : Assume, that both arrays are sorted in ascending order and we want resulting array to maintain the same order. Algorithm to merge two arrays A[0..m-1] and B[0..n-1] into an array C[0..m+n-1] is as following:

1. Introduce read-indices **i**, **j** to traverse arrays A and B, accordingly. Introduce write-index **k** to store position of the first free cell in the resulting array. By default **i** = **j** = **k** = 0.
2. At each step: if both indices are in range (**i** < m and **j** < n), choose minimum of (A[**i**], B[**j**]) and write it to C[**k**]. Otherwise go to step 4.
3. Increase **k** and index of the array, algorithm located minimal value at, by one. Repeat step 2.
4. Copy the rest values from the array, which index is still in range, to the resulting array.

Q.8 Write code to merge two sorted arrays ?

Ans

```
/* Program to merge two sorted arrays */
#include<stdio.h>
int merge(int[],int[],int[],int,int);
```

```
            void main()
            {
                    int a[20],b[20],c[40],n,m,i,p;
                   printf("\nEnter the no.of element present in the first array: ");
                   scanf("%d",&n);
                   printf("\nEnter the first array…….\n");
                   for(i=0;i<n;i++)
                       scanf("%d",&a[i]);
                   printf("\nEnter the no. of element present in the second array:
            ");
                   scanf("%d",&m);
                   printf("\nEnter the second array…….\n");
                   for(i=0;i<m;i++)
                      scanf("%d",&b[i]);
                   p=merge(a,b,c,n,m);
                   printf("\nThe merged array is………\n");
                   for(i=0;i<p;i++)
                     printf("%d  ",c[i]);
                     printf("\n\n");
            }
    int merge(int a[],int b[],int c[],int n,int m)
    {
           int i=0,j=0,k=0;
           while(i<n&&j<m)
           {
                if(a[i]<b[j])
                {
                    c[k]=a[i];
                     i++;
            k++;
      }
     else if(a[i]>b[j])
     {
            c[k]=b[j];
            j++;
            k++;
      }
      else           // to avoid duplication
      {
          c[k]=a[i];
          i++;
          j++;
                   k++;
                       }
```

```
              }
            for(;i<n;i++,k++)
               c[k]=a[i];
             for(;j<m;j++,k++)
              c[k]=b[j];
             return k;
}
```

Q.9 What is two-dimensional array?

Ans Two-dimensional arrays, the most common multidimensional arrays, are used to store information that we normally represent in table form. Two-dimensional arrays, like one-dimensional arrays, are homogeneous. This means that all of the data in a two-dimensional array is of the same type. The two-dimensional array may also be visualized as a one-dimensional array of arrays.

Examples of applications involving two-dimensional arrays include:

- a seating plan for a room (organized by rows and columns),
- a monthly budget (organized by category and month), and
- a grade book where rows might correspond to individual students and columns to student scores.

Declaration of Two-Dimensional Arrays

```
  int a[3][2];
```

We can declare and initialize an array A as follows:

```
//declaration
  int A[3][4] = {{8, 2, 6, 5},    //row 0
        {6, 3, 1 ,0},     //row 1
        {8, 7, 9, 6}};    //row 2
```

Q.10 How to read and write data in 2-D array?

Ans

```
#include <stdio.h>

void main()
{
         int a[3][2];
         int i,j;
          for(i = 0;i<3;i++){
          for(j=0;j<2 ;j++) {
        a[i][j]=2;
      }
```

```
        }

        for(i = 0;i<3;i++){
           for(j=0;j<2;j++) {
             printf("value in array %d\n",a[i][j]);
           }
        }

        for(i = 0;i<3;i++){
           for(j=0;j<2;j++){
           printf("value in array %d and address is %8u\n", a[i][j],&a[i][j]);
           }
        }
     }
```

Q .11 Write an algorithm to multiply two 2-Dimensional array?

Ans: **Multiplication of two matrixes:**

Rule: Multiplication of two matrixes is only possible if first matrix has size m X **n** and other matrix has size **n** x r. Where m, n and r are any positive integer.

Multiplication of two matrixes is defined as

$$[AB]_{i,j} = \sum_{s=1}^{n} A_{i,s} B_{s,j}$$

Where $1 \le i \le m$ and $1 \le j \le n$

For example:

Suppose two matrixes A and B of size of 2 x 2 and 2 x 3 respectively:

$$A = \begin{pmatrix} 1 & 2 \\ 3 & 4 \end{pmatrix} \quad B = \begin{pmatrix} 5 & 6 & 7 \\ 8 & 9 & 10 \end{pmatrix}$$

Multiplication of two matrixes:

$$A * B = \begin{pmatrix} 1*5 + 2*8 & 1*6 + 2*9 & 1*7 + 2*10 \\ 3*5 + 4*8 & 3*6 + 4*9 & 3*7 + 4*10 \end{pmatrix}$$

$$A * B = \begin{pmatrix} 21 & 24 & 27 \\ 47 & 54 & 61 \end{pmatrix}$$

Here **A** is a two – dimensional array with **M** rows and **N** columns and **B** is a two – dimensional array with **X** rows and **Y** columns. This algorithm multiplies these two arrays.

```
1. If (M ≠ Y) or (N ≠ X) Then
2. Print: Multiplication is not possible.
3. Else
4. Repeat For I = 1 to N
5. Repeat For J = 1 to X
6. Set C[I][J] = 0
7. Repeat For K = 1 to Y
8. Set C[I][J] = C[I][J] + A[I][K] * B[K][J]
[End of Step 7 For Loop]
[End of Step 5 For Loop]
[End of Step 4 For Loop]
[End of If]
```

9. Exit

Explanation: First we check whether the rows of A are equal to columns of B or the columns of A are equal to rows of B. If they are not equal, then multiplication is not possible. But, if they are equal, the first for loop iterates to total number of columns of A i.e. N and the second for loop iterates to the total number of rows of B i.e. X.

In step 6, all the elements of C are set to zero. Then the third for loop iterates to total

number of columns of B i.e. Y.

In step 8, the element A[I][K] is multiplied with B[K][J] and added to C[I][J] and the result is assigned to C[I][J] by the statement:

C[I][J] = C[I][J] + A[I][K] * B[K][J]

```
#include<stdio.h>
void main()
{
 int a[5][5],b[5][5],c[5][5],i,j,k,sum=0,m,n,o,p;
 printf("\nEnter the row and column of first  matrix");
 scanf("%d %d",&m,&n);
 printf("\nEnter the row and column of second matrix");
 scanf("%d %d",&o,&p);
 if(n!=o){
    printf("Matrix mutiplication is not possible");
    printf("\nColumn of first matrix must be same as row of second matrix");
 }
 else{
    printf("\nEnter the First matrix->");
    for(i=0;i<m;i++)
    for(j=0;j<n;j++)
        scanf("%d",&a[i][j]);
    printf("\nEnter the Second matrix->");
    for(i=0;i<o;i++)
    for(j=0;j<p;j++)
        scanf("%d",&b[i][j]);
    printf("\nThe First matrix is\n");
    for(i=0;i<m;i++){
    printf("\n");
    for(j=0;j<n;j++){
        printf("%d\t",a[i][j]);
    }
    }
    printf("\nThe Second matrix is\n");
    for(i=0;i<o;i++){
    printf("\n");
    for(j=0;j<p;j++){
        printf("%d\t",b[i][j]);
    }
    }
    for(i=0;i<m;i++)
    for(j=0;j<p;j++)
        c[i][j]=0;
    for(i=0;i<m;i++){ //row of first matrix
    for(j=0;j<p;j++){  //column of second matrix
        sum=0;
        for(k=0;k<n;k++)
           sum=sum+a[i][k]*b[k][j];
        c[i][j]=sum;
    }
    }
 }
 printf("\nThe multiplication of two matrix is\n");
 for(i=0;i<m;i++){
    printf("\n");
    for(j=0;j<p;j++){
        printf("%d\t",c[i][j]);
    }
 }
         getch();
}
```

Q.12 Wap to Sum of diagonal elements of a matrix in c?

Ans:

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

           int a[10][10],i,j,sum=0,m,n;

           printf("\nEnter the row and column of matrix: ");
```

```
 scanf("%d %d",&m,&n);

 printf("\nEnter the elements of matrix: ");
 for(i=0;i<m;i++)
     for(j=0;j<n;j++)
      scanf("%d",&a[i][j]);
printf("\nThe matrix is\n");

for(i=0;i<m;i++){
   printf("\n");
for(j=0;j<m;j++){
 printf("%d\t",a[i][j]);
  }
}
for(i=0;i<m;i++){
    for(j=0;j<n;j++){
         if(i==j)
             sum=sum+a[i][j];
    }
}
printf("\n\nSum of the diagonal elements of a matrix is: %d",sum);

getch();
}
```

output:

```
Enter the row and column of matrix: 3 3
Enter the elements of matrix: 2
3
5
6
7
9
2
6
7
The matrix is
2       3       5
6       7       9
```

2 6 7

Sum of the diagonal elements of a matrix is: 16

<u>Explanation</u>

$$A = \begin{pmatrix} \mathbf{A}_{11} & A_{12} & A_{13} \\ A_{21} & \mathbf{A}_{22} & A_{23} \\ A_{31} & A_{32} & \mathbf{A}_{33} \end{pmatrix}$$

Diagonal elements have been shown in the bold letter.

We can observer the properties any element A_{ij} will diagonal element if and only if i = j

Searching

Q .1 How to do Linear Search in an Array?

Ans : Linear search or Sequential is the simplest search algorithm for finding a particular value in an array, that consists of checking every one of its elements, one at a time and in sequence, until the desired one is found.

Q .2 Write code to do Linear Search in an Array?

Ans :

```
/* Program for linear search an element */

#include <stdio.h>
#include <conio.h>

void main()
{
        int array[10];
        int i, N, keynum, found=0;
       clrscr();

      printf("Enter the value of N\n");
      scanf("%d",&N);

     printf("Enter the elements one by one\n");
     for(i=0; i<N ; i++)
         scanf("%d",&array[i]);

     printf("Input array is\n");
     for(i=0; i<N ; i++)
        printf("%d\n",array[i]);

    printf("Enter the element to be searched\n");
   scanf("%d", &keynum);

 /* Linear search begins */
i=0;
While(i < n) && (found==0)
 {
     if( keynum == array[i] )
           found = 1;
     i++;
   }
   if ( found == 1)
        printf("SUCCESSFUL SEARCH\n");
   else
        printf("Search is FAILED\n");

} /* End of main */
```

Q.3 How to do Binary Search in an Array?

Ans : A Binary Search or Half-interval search algorithm finds the position of a specified value (the input "key") within a sorted array. In each step, the algorithm compares the input key value with the key value of the middle element of the array. If the keys match, then a matching element has been found so its index, or position, is returned. Otherwise, if the sought key is less than the middle element's key, then the algorithm repeats its action on the sub-array to the left of the middle element or, if the input key is greater, on the sub-array to the right. If the remaining array to be searched is reduced to zero, then the key cannot be found in the array and a special "Not found" indication is returned.

Q.4 Write 'C' code to do Binary Search in an Array?

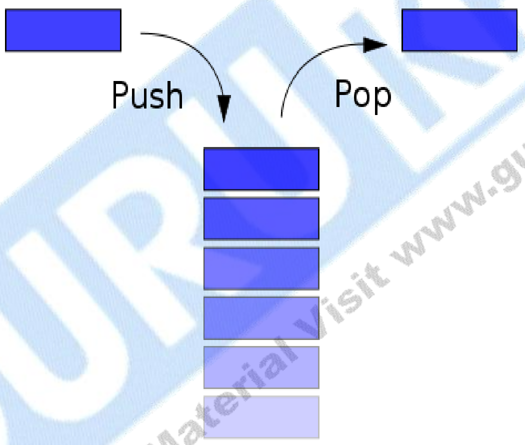
Ans :

```
#include <stdio.h>
#include <conio.h>

void main()
{
        int array[10];
        int i, n, elem, found=0;
       clrscr();

      printf("Enter the value of N\n");
      scanf("%d",&n);

     printf("Enter the elements one by one\n");
     for(i=0; i<n ; i++)
          scanf("%d",&array[i]);

     printf("Input array is\n");
     for(i=0; i<n; i++)
        printf("%d\n",array[i]);

    printf("Enter the element to be searched\n");
   scanf("%d", &elem);
   /* Binary search begins */
 l=0;
             u=n-1;
             While((l<u) && (found==0))
             {
                          mid=(l+u)/2;
                          if(a[mid]==elem)
                          found=1;
             else
                          if(a[mid]>elem)
                                       u=mid-1;
                          else
                                       l=mid+1;
             }
             mid=mid+1;
 if ( found == 1)
                          printf("SUCCESSFUL SEARCH\n");
 else
                          printf("Search is FAILED\n");

} /* End of main */
```

Stack

Q.1 What is Stack?

Ans A **stack** is a last in, first out (LIFO) abstract data type and linear data structure. A stack can have any abstract data type as an element, but is characterized by two fundamental operations, called *push* and *pop*. The push operation adds a new item to the top of the stack, or initializes the stack if it is empty. If the stack is full and does not contain enough space to accept the given item, the stack is then considered to be in an overflow state. The pop operation removes an item from the top of the stack. A pop either reveals previously concealed items, or results in an empty stack, but if the stack is empty then it goes into underflow state (It means no items are present in stack to be removed).



Q.2 Write a Program for Stack implementation through Array?

Ans

```
#include <stdio.h>
#include<ctype.h>
# define MAXSIZE 200

int stack[MAXSIZE];
int top;      //index pointing to the top of stack
void main()
{
   void push(int);
   int pop();
   int will=1,i,num;
   while(will ==1)
   {
                printf("MAIN MENU:
                1.Add element to stack
                2.Delete element from the stack
   ");
   Scanf ("%d", &will);

   switch(will)
   {
                 case 1: printf("Enter the data... ");
                             scanf("%d", &num);
                              push(num);
                             break;
                  case 2: num=pop();
                             printf("Value returned from pop function is  %d ",num);
                             break;
                default:  printf ("Invalid Choice . ");
   }
   printf("Do you want to do more operations on Stack ( 1 for yes, any other
key to exit) ");
   scanf("%d" , &will);
        } //end of  outer while
}              //end of main
void push(int elem)
{
   if(isfull())
                {
                             printf("STACK FULL");
                             return;
                }
   top++;
   stack[top]=elem;
   }
}

int pop()
{
   int a;
   if(isEmpty())
   {
                printf("STACK EMPTY");
                return 0;
   }
   a=stack[top];
```

```
    top--;
    }
    return(a);
  }

  int isFull()
  {
    if(top> MAXSIZE)
            return 0;
    else
            return 1;
  }
  int isEmpty()
  {
    if(top<=0)
            return 0;
    else
            return 1;
  }
```

Q.3 What is the main difference between ARRAY and STACK?

Ans Stack follows LIFO. Thus the item that is first entered would be the last removed. In array the items can be entered or removed in any order. Basically each member access is done using index and no strict order is to be followed here to remove a particular element .Array may be multi-dimensional or one dimensional but stack should be one-dimensional. Size of array is fixed, while stack can be grow or shrink. We can say stack is dynamic data structure.

Q .4 What are various Applications of Stack?

Ans: Some of the applications of stack are :

(i) Evaluating arithmetic expression
(ii) Balancing the symbols
(iii) Towers of Hanoi
(iv) Function Calls.
(v) 8 Queen Problem.

Q .5 How to Evaluating Arithmetic Expression?

Ans: There are 3 different ways of representing the algebraic expression. They are

* INFIX NOTATION
* POSTFIX NOTATION
* PREFIX NOTATION

INFIX NOTATION

In Infix notation, the arithmetic operator appears between the two operands to which it is being applied. For example: - A / B + C

POSTFIX NOTATION

The arithmetic operator appears directly after the two operands to which it applies. also called reverse polish notation. ((A/B) + C)For example: - AB / C +

PREFIX NOTATION

The arithmetic operator is placed before the two operands to which it applies. Also called as polish notation. ((A/B) + C)For example: - +/ABC
INFIX PREFIX (or) POLISH POSTFIX (or) REVERSEPOLISH
1. (A + B) / (C - D) /+AB - CD AB + CD - /
2. A + B*(C - D) +A*B - CD ABCD - * +
3. X * A / B - D - / * XABD X A*B/D-
4. X + Y * (A - B) / +X/*Y - AB - CD XYAB - *CD - / +(C - D)
5. A * B/C + D + / * ABCD AB * C / D +

To evaluate an arithmetic expressions, first convert the given infix expression to postfix expression and then evaluate the postfix expression using stack.

<u>Infix to Postfix Conversion</u>

Read the infix expression one character at a time until it encounters the delimiter. "#"
Step 1 : If the character is an operand, place it on to the output.
Step 2: If the character is an operator, push it onto the stack. If the stack operator has a higher or equal priority than input operator then pop that operator from the stack and place it onto the output.
Step 3: If the character is a left parenthesis, push it onto the stack.
Step 4: If the character is a right parenthesis, pop all the operators from the stack till item counters left parenthesis, discard both the parenthesis in the output.

<u>Evaluating Postfix Expression</u>

Read the postfix expression one character at a time until it encounters the delimiter `#'.
Step 1: - If the character is an operand, push its associated value onto the stack.
Step 2: - If the character is an operator, POP two values from the stack, apply the operator to them and push the result onto the stack.

<u>Infix to Postfix Example</u>

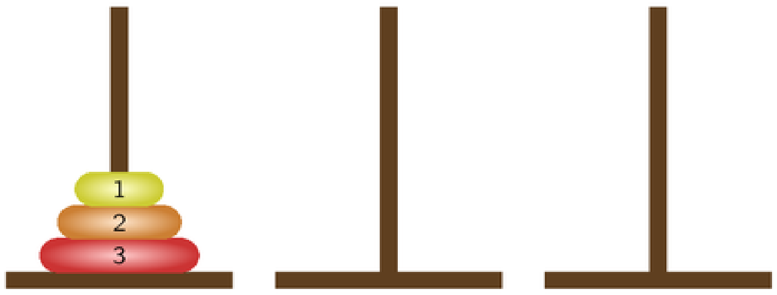
A + B * C - D / E

Infix	Stack(bot->top)	Postfix
a) A + B * C - D / E		

b)	+ B * C - D / E		A
c)	B * C - D / E	+	A
d)	* C - D / E	+	A B
e)	C - D / E	+ *	A B
f)	- D / E	+ *	A B C
g)	D / E	+ -	A B C *
h)	/ E	+ -	A B C * D
i)	E	+ - /	A B C * D
j)		+ - /	A B C * D E
k)			A B C * D E / - +

Q.6 Explain Tower Of Hanoi Problem ?

Ans: The Tower of Hanoi puzzle was invented by the French mathematician Edouard Lucas in 1883. We are given a tower of Three disks, initially stacked in increasing size on one of three pegs. The objective is to transfer the entire tower to one of the other pegs (the rightmost), moving only one disk at a time and never a larger one onto a smaller.



Solution for N=3

1. Move from Src to Dst
2. Move from Src to Mid
3. Move from Dst to Mid
4. Move from Src to Dst
5. Move from Mid to Src
6. Move from Mid to Dst
7. Move from Src to Dst

Recursive Algorithm for Tower of Hanoi

```
TowerOfHanoi(N, Src, Mid, Dst)
 if N is 0
     exit
 else
```

```
TowerOfHanoi(N - 1, Src, Dst, Mid)
Move from Src to Dst
TowerOfHanoi(N - 1, Mid, Src, Dst)
```

Program for explain tower of hanoi problem

```
#include<conio.h>
#include<stdio.h>

void towers(int,char,char,char);
void main()
{
 int n;
 clrscr();
 printf("Enter the no.of elements:");
 scanf("%d",&n);
 towers(n,'a','c','b');
 getch();
}
void towers(int n,char frompeg,char topeg,char auxpeg)
{
if(n==1)
{
printf("\n%s%c\t%s%c","movedisk1 frompeg",frompeg,"topeg",topeg);
return;
}
towers(n-1,frompeg,auxpeg,topeg);
printf("\n%s%d\t%s%c\t%s%c","movedisk",n,"frompeg",frompeg,"top
eg",topeg);
towers(n-1,auxpeg,topeg,frompeg);
}
```

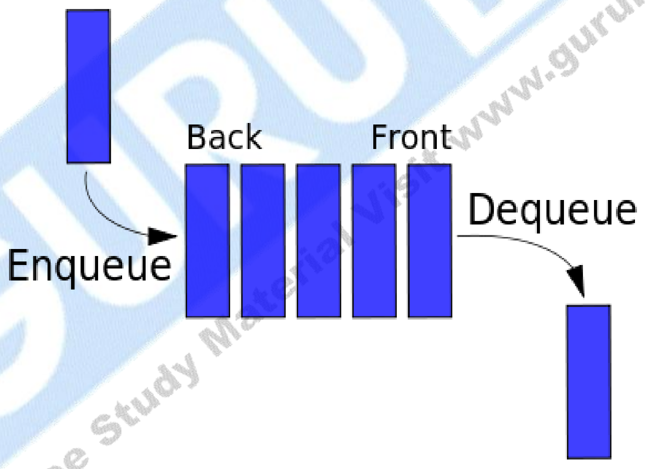
Queue

Q.1 What is Queue?

Ans : A **Queue** is a **first in, first out** (FIFO) abstract data type and linear data structure. A Queue can have any abstract data type as an element, but is characterized by two fundamental operations, called Enqueue and Dequeue. Queue has two pointers **Front** and **Rear.**

The **Enqueue** operation adds a new item to the front of the Queue, or initializes the queue if it is empty. If the Queue is full and does not contain enough space to accept the given item, the Queue is then considered to be in an overflow state.

The **Dequeue** operation removes an item from the front of the queue. A dequeue either reveals previously concealed items, or results in an empty queue, but if the queue is empty then it goes into underflow state (It means no items are present in queue to be removed).



While dealing with the circular queue we will use the following assumptions :

1. Front will always be pointing to the first element of the queue.
2. Each time a new element is inserted into the queue, the value of rear is incremented by one i.e., Rear = (Rear + 1);
3. Each time when an existing element is deleted from the queue, the value of rear is incremented by one i.e., Front = (Front + 1);
4. Front is the initial or first location of the queue where Rear indicates the last entry location in the queue.

Q.2 Write a code in 'C' for Linear Queue through Array?

Ans

```
#include <stdio.h>
#include<ctype.h>
# define MAXSIZE 200

int queue[MAXSIZE];
int front=-1, rear=-1;//index pointing to the top of stack
void main()
{
  void enqueue(int);
  int dequeue();
  int will=1,i,num,ch;
  while(will ==1)
  {
            printf("MAIN MENU:
            1. Enqueue
            2.Dequeue
            3. Check Queue Full
            4. Check Queue Empty ");

            Scanf ("%d", &ch);

            switch(ch)
            {
             case 1: printf("Enter the data to add... ");
                     scanf("%d", &num);
                     enqueue(num);
                     break;
              case 2: num=dequeue();
                     if(num==-1)
                              printf("Queue is empty");\
                     else
                              printf("Value returned from pop function is  %d
",num);
                     break;
              case 3 : isFull();
                     break;
              case 4 : isEmpty();
                     break;
            default:  printf ("Invalid Choice . ");
  }
  printf("Do you want to do more operations on Stack ( 1 for yes, any other
key to exit) ");
  scanf("%d" , &will);
```

```
        } //end of  outer while
}            //end of main
void enqueue (int elem)
{
  if(isfull())
            {
                      printf("QUEUE FULL");
                      return;
            }
  else{
            if(front==-1)
                      front=rear=0;
            else
                      rear++;
            queue[rear]=elem;
  }
}
int dequeue()
{
  int elem;
  if(isEmpty())
  {
            return -1;
  }
  else
  {
            elem=queue[front];
            if (front==rear)
                      front=rear=-1;
            else
                      front++;
  }
  return(elem);
}
int isFull()
{
  if(rear==MAXSIZE-1)
            return 0;
  else
            return 1;
}
int isEmpty()
{
  if((front==-1) && (front==rear))
```

```
                return 0;
    else
                return 1;
   }
```

Q.3 What is a Circular Queue?

Ans. A **circular queue** is one in which the insertion of new element is done at the very first location of the queue if the last location of the queue is full. Suppose if we have a Queue of n elements then after adding the element at the last index i.e. (n-1)th , as queue is starting with 0 index, the next element will be inserted at the very first location of the queue which was not possible in the simple linear queue. That's why linear queue leads to wastage of the memory, and this flaw of linear queue is overcome by circular queue. So, in all we can say that the circular queue is a queue in which first element come right after the last element, that means a circular queue has a starting point but no end point.

Q.4 Write a code in 'C' for Circular Queue through Array?

Ans

```
#include <stdio.h>
#include<ctype.h>
# define MAXSIZE 200

int queue[MAXSIZE];
int front=-1, rear=-1;//index pointing to the top of stack
void main()
{
  void enqueue(int);
  int dequeue();
  int will=1,i,num,ch;
  while(will ==1)
  {
            printf("MAIN MENU:
            1. Enqueue
            2.Dequeue
            3. Check Queue Full
            4. Check Queue Empty ");

            Scanf ("%d", &ch);

            switch(ch)
            {
              case 1: printf("Enter the data to add... ");
                      scanf("%d", &num);
```

```
                    enqueue(num);
                    break;
            case 2: num=dequeue();
                    if(num= = -1)
                              printf("Queue is empty");\
                    else
                              printf("Value returned from pop function is  %d
",num);
                    break;
            case 3 : isFull();
                    break;
            case 4 : isEmpty();
                    break;
           default:  printf ("Invalid Choice . ");
  }
  printf("Do you want to do more operations on Stack ( 1 for yes, any other
key to exit) ");
  scanf("%d" , &will);
      } //end of  outer while
}          //end of main

void enqueue (int elem)
{
  if(isfull())
           {
                    printf("QUEUE FULL");
                    return;
           }
  else{
           if(rear==-1)
                    front=rear=0;
           else
                    rear=(rear+1)% MAXSIZE;
           queue[rear]=elem;
  }
}
int dequeue()
{
  int elem;
  if(isEmpty())
  {
           return -1;
  }
  else
```

```
    {
            elem=queue[front];
            if (front==rear)
                    front=rear=-1;
            else
                    front= (front+1) % MAXSIZE ;
    }
    return(elem);
  }
  int isFull()
  {
    If(((front==0) && (rear==MAXSIZE-1)) || (front=rear+1))
            return 0;
    else
            return 1;
  }
  int isEmpty()
  {
    if((front==-1) && (front==rear))
            return 0;
    else
            return 1;
  }
```

Q.5 What is Priority Queue ?

Ans: A Priority Queue is a collection of elements such that each element has been assigned a priority and such that the order in which each elements are deleted and processed comes from the following rules :

- ➢ An element of higher priority is processes before any element of lower priority.
- ➢ Two elements with the same priority are processed according to the order in which they were added to the queue

A priority queue must at least support the following operations:

- **insert_with_priority** : add an element to the queue with an associated priority
- **pull_highest_priority_element:** remove the element from the queue that has the *highest priority*, and return it.

Algorithm to add an item in priority queue

This algorithm adds an item with priority number M to a priority Queue maintained by a 2-dimensional array Queue.

1. Insert Item as the rear element in row M of Queue.
2. Exit.

Algorithm to delete an item from a priority queue

This algorithm deleted and processes the first element in a priority queue maintained by a 2-dimensional array Queue

```
1. [Find the first nonempty queue]
   Find the smallest K such that FRONT[K]!=NULL.
2. Delete and process the front element in row K of Queue.
3. Exit.
```

Sorting

Q.1 What is selection sort?

Ans: This is the simplest sorting method. In order to sort data in ascending order, the 1st element is compared to all other elements and if found greater than the compared element, then they are interchanged. So after the first iteration the smallest element is placed at the 1st position. The same procedure is repeated for the 2nd element and so on for all the elements.

Example: Selection Sort

[0]	[1]	[2]	[3]	[4]	
5	1	3	7	2	find min
1	5	3	7	2	swap to index 0
1	5	3	7	2	find min
1	2	3	7	5	swap to index 1
1	2	3	7	5	find min
1	2	3	7	5	swap to index 2
1	2	3	7	5	find min
1	2	3	5	7	swap to index 3

Q.2 Write a 'C' code for selection sort?

Ans:

```
#include<stdio.h>
main ()
{
  int array[100], size, i, j, min, swap;

  printf("Enter number of elements\n");
  scanf("%d", &size);

  printf("Enter %d integers\n", size);
  for ( i = 0 ; i < size ; i++ )
                    scanf("%d", &array[i]);

  for ( i = 0 ; i < ( size - 1 ) ; i++ )
  {
                    min = i;
                    for ( j = i + 1 ; j < size ; j++ )
                    {
                             if ( array[min] > array[j] )
                             min = j;
```

```
                        }
                        if ( min != i )
                        {
                                swap = array[i];
                                array[i] = array[min];
                                array[min] = swap;
                        }
        }
        printf("Sorted list in ascending order:\n");
        for ( i = 0 ; i < size ; i++ )
                        printf("%d\n", array[i]);

        return 0;
}
```

Q.3 What is bubble sort?

Ans : **Bubble sort**, often incorrectly referred to as **sinking sort**, is a simple sorting algorithm that works by repeatedly stepping through the list to be sorted, comparing each pair of adjacent items and swapping them if they are in the wrong order. The pass through the list is repeated until no swaps are needed, which indicates that the list is sorted. The algorithm gets its name from the way smaller elements "bubble" to the top of the list. Because it only uses comparisons to operate on elements, it is a **comparison sort.**

For Example

Let us take the array of numbers "5 1 4 2 8", and sort the array from lowest number to greatest number using bubble sort algorithm. In each step, elements written in **bold** are being compared. Three passes will be required.

First Pass:

(**5 1** 4 2 8) ⟶(**1 5** 4 2 8), Here, algorithm compares the first two elements, and swaps them.
(1 **5 4** 2 8) ⟶(1 **4 5** 2 8), Swap since 5 > 4
(1 4 **5 2** 8) ⟶(1 4 **2 5** 8), Swap since 5 > 2
(1 4 2 **5 8**) ⟶(1 4 2 **5 8**), Now, since these elements are already in order (8 > 5), algorithm does not swap them.

Second Pass:

(**1 4** 2 5 8) ⟶(**1 4** 2 5 8)
(1 **4 2** 5 8) ⟶(1 **2 4** 5 8), Swap since 4 > 2
(1 2 **4 5** 8) ⟶(1 2 **4 5** 8)
(1 2 4 **5 8**) ⟶(1 2 4 **5 8**)

Now, the array is already sorted, but our algorithm does not know if it is completed. The algorithm needs one **whole** pass without **any** swap to know it is sorted.

Third Pass:

(**1 2** 4 5 8) $\longrightarrow$ (**1 2** 4 5 8)
(1 **2 4** 5 8) $\longrightarrow$ (1 **2 4** 5 8)
(1 2 **4 5** 8) $\longrightarrow$ (1 2 **4 5** 8)
(1 2 4 **5 8**) $\longrightarrow$ (1 2 4 **5 8**)

Q .5 Write a 'C' code for bubble sort ?

Ans :

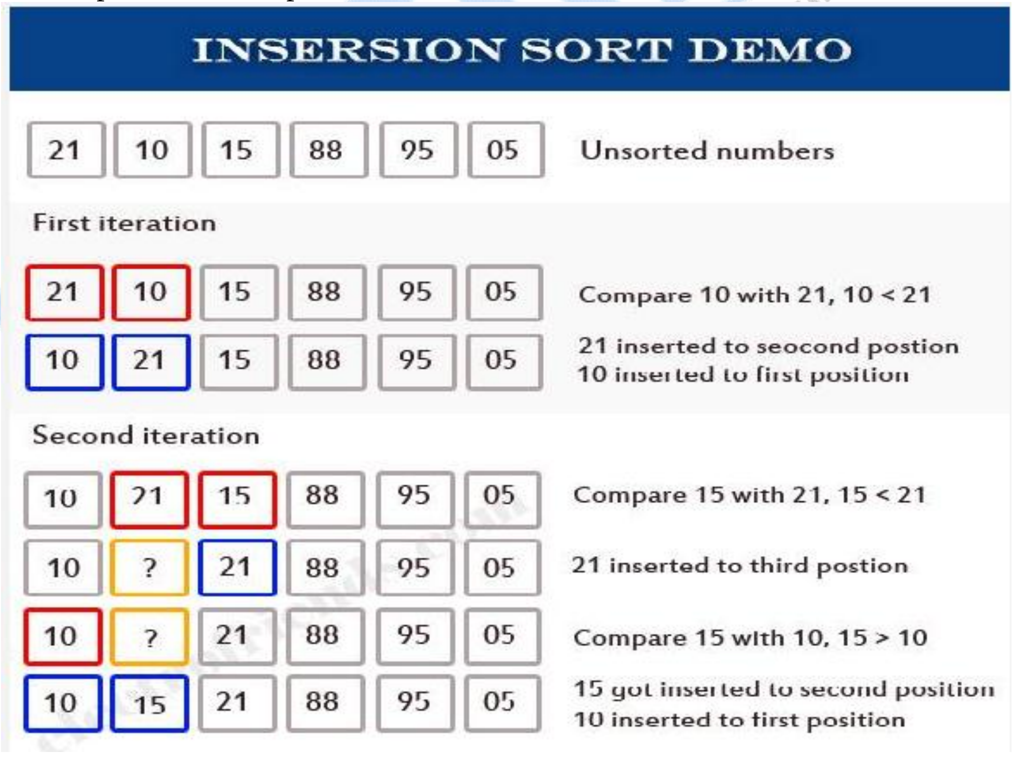
```
// Program of sorting using bubble sort method
#include <stdio.h>
#define MAX 20

main()
{
        int arr[MAX], i, j, k, temp, size, xchanges;
        printf("Enter the number of elements : ");
                scanf("%d",&n);
        printf("Enter array element");
for (i = 0; i < n; i++)
                scanf("%d",&arr[i]);

        printf("Unsorted list is :\n");
        for (i = 0; i < n; i++)
                printf("%d ", arr[i]);
        printf("\n");

/* Bubble sort*/
        for (i = 1; i < n ; i++)
        {
                for (j = 0; j <n-1-i; j++)
                {
                        if (arr[j] > arr[j+1])
                        {
                                temp = arr[j];
                                arr[j] = arr[j+1];
                                arr[j+1] = temp;
                        }/*End of if*/
                }/*End of inner for loop*/

        }/*End of outer for loop*/
```

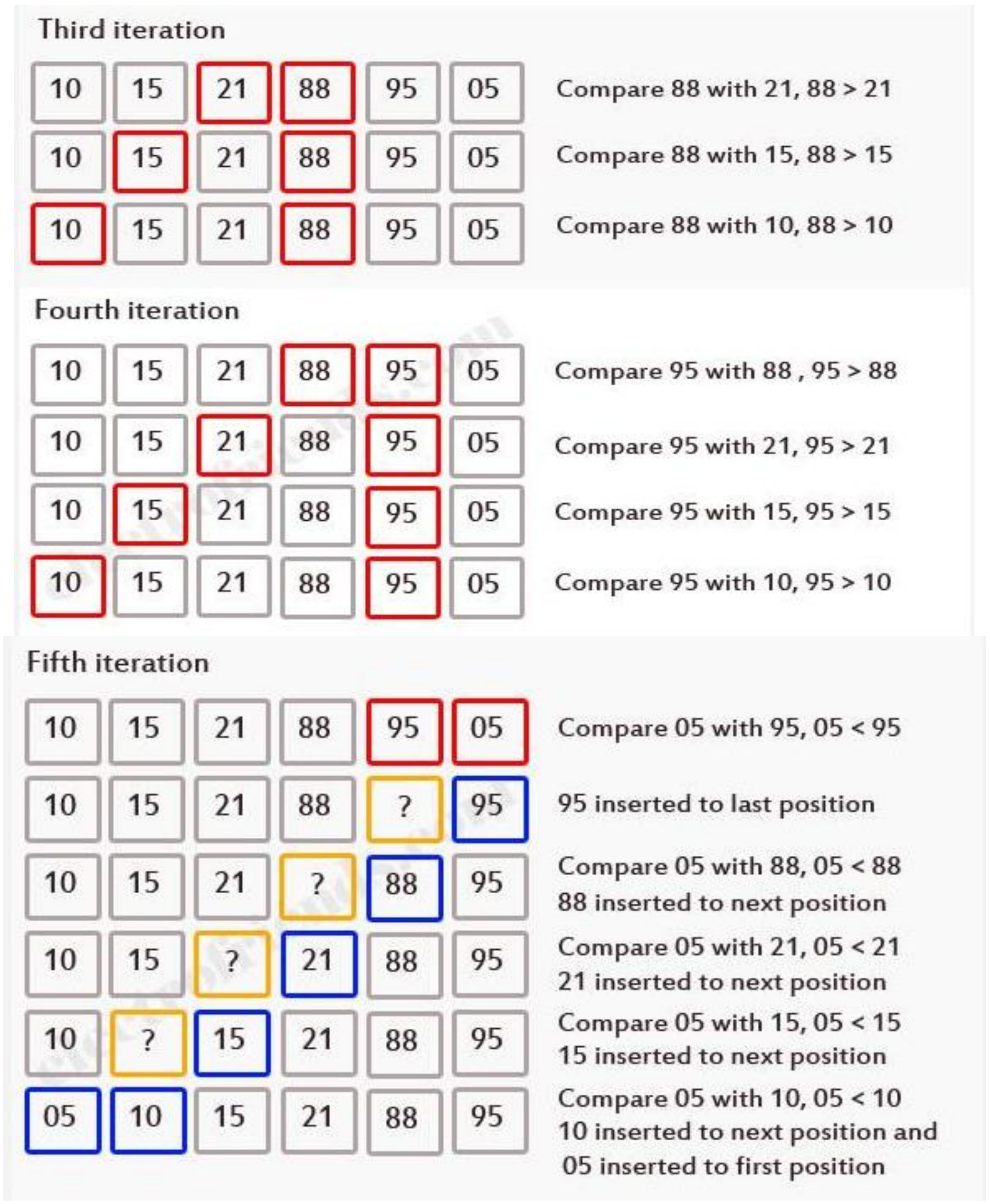
```
    printf("Sorted list is :\n");
    for (i = 0; i < n; i++)
            printf("%d ", arr[i]);
    printf("\n");
}/*End of main()*/
```

Q.6 What is Insertion sort?

Ans: Insertion sort is an elementary sorting algorithm. Here, sorting takes place by inserting a particular element at the appropriate position, that's why the name- insertion sorting.

- ➢ In the First iteration, second element A[1] is compared with the first element A[0].
- ➢ In the second iteration third element is compared with first and second element.
- ➢ In general, in every iteration an element is compared with all the elements before it.
- ➢ While comparing if it is found that the element can be inserted at a suitable position, then space is created for it by shifting the other elements one position up and inserts the desired element at the suitable position. This procedure is repeated for all the elements in the list.


INSERSION SORT DEMO
21 10 15 88 95 05
Unsorted numbers
First iteration
21 10 15 88 95 05
Compare 10 with 21, 10 < 21
10 21 15 88 95 05
21 inserted to seocond postion
10 inserted to first position
Second iteration
10 21 15 88 95 05
Compare 15 with 21, 15 < 21
10 ? 21 88 95 05
21 inserted to third postion
10 ? 21 88 95 05
Compare 15 with 10, 15 > 10
10 15 21 88 95 05
15 got inserted to second position
10 inserted to first position


Third iteration
10 15 21 88 95 05
Compare 88 with 21, 88 > 21
10 15 21 88 95 05
Compare 88 with 15, 88 > 15
10 15 21 88 95 05
Compare 88 with 10, 88 > 10
Fourth iteration
10 15 21 88 95 05
Compare 95 with 88 , 95 > 88
10 15 21 88 95 05
Compare 95 with 21, 95 > 21
10 15 21 88 95 05
Compare 95 with 15, 95 > 15
10 15 21 88 95 05
Compare 95 with 10, 95 > 10
Fifth iteration
10 15 21 88 95 05
Compare 05 with 95, 05 < 95
10 15 21 88 ? 95
95 inserted to last position
10 15 21 ? 88 95
Compare 05 with 88, 05 < 88
88 inserted to next position
10 15 ? 21 88 95
Compare 05 with 21, 05 < 21
21 inserted to next position
10 ? 15 21 88 95
Compare 05 with 15, 05 < 15
15 inserted to next position
05 10 15 21 88 95
Compare 05 with 10, 05 < 10
10 inserted to next position and
05 inserted to first position

Q.7 Write a 'C' code for insertion sort ?

Ans :

```
#include<stdio.h>
int main()
{
  int i,j,size,temp,a[20];
  clrscr();
  printf("\nEnter size of the array: ");
  scanf("%d",&size);
```

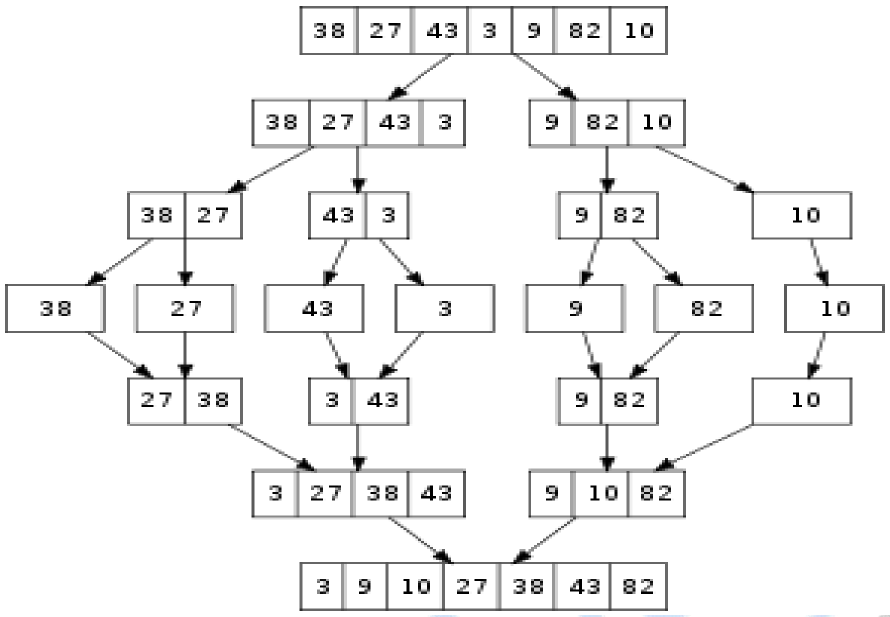
```
   printf("\nEnter elements in to the array:");
   for(i=0;i<size;i++)
      scanf("%d",&a[i]);
   for(i=1;i<size;i++)
   {
          temp=a[i];
          j=i-1;
          while((j>=0)&&(temp<a[j]))
          {
                   a[j+1]=a[j];
                  j=j-1;
          }
          a[j+1]=temp;
   }
   printf("\nAfter sorting the elements are: ");
   for(i=0;i<size;i++)
      printf(" %d",a[i]);
   return 0;
  }
```

Q.8 What is Merge sort?

Ans : Conceptually, merge sort works as follows:

1. Divide the unsorted list into two sublists of about half the size
2. Divide each of the two sublists recursively until we have list sizes of length 1, in which case the list itself is returned
3. Merge the two sorted sublists back into one sorted list.

For Example



Q.9 Write a 'C' code for Merge sort ?

Ans :

```
#include<stdio.h>
int main()
{
        int i,j,size,,a[50];
        clrscr();
        printf("\nEnter size of the array: ");
        scanf("%d",&size);
        printf("\nEnter elements in to the array:");
        for(i=0;i<size;i++)
                scanf("%d",&a[i]);
        mergesort(a,0,size-1)
        printf("\nAfter sorting the elements are: ");
        for(i=0;i<size;i++)
                 printf(" %d",a[i]);

}
void mergesort(int a[], int low, int high)
{
int mid;
if(low<high)
{
       mid=(low+high)/2;
       mergesort(a,low,mid);
       mergesort(a,mid+1,high);
```

```
          merge(a,low,high,mid);
}
}
merge(int a[], int low, int high, int mid)
{
 int i, j, k, c[50];
i=low;
j=mid+1;
k=low;
while((i<=mid)&&(j<=high))
{
        if(a[i]<a[j])
       {
                 c[k]=a[i];
                  k++;
                  i++;
       }
      else
      {
              c[k]=a[j];
               k++;
               j++;
      }
}
while(i<=mid)
{
           c[k]=a[i];
            k++;
            i++;
}
while(j<=high)
{
         c[k]=a[j];
          k++;
          j++;
}
for(i=low;i<k;i++)
{
         a[i]=c[i];
}
}
```

Q.10 What is Merge sort?

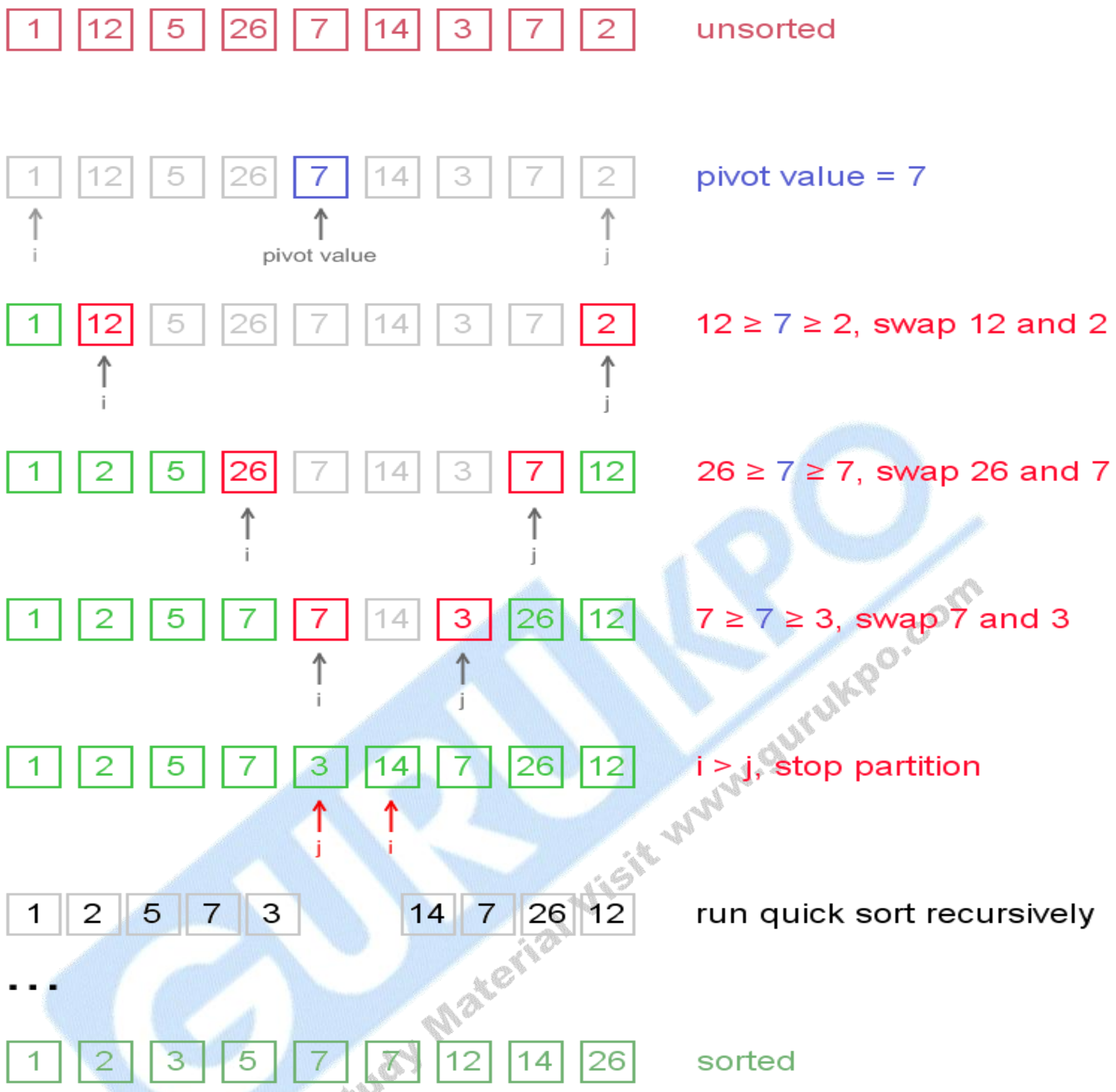
Ans : The divide-and-conquer strategy is used in quicksort. Below the recursion step is described:

1. **Choose a pivot value.** We take the value of the middle element as pivot value, but it can be any value, which is in range of sorted values, even if it doesn't present in the array.
2. **Partition.** Rearrange elements in such a way, that all elements which are lesser than the pivot go to the left part of the array and all elements greater than the pivot, go to the right part of the array. Values equal to the pivot can stay in any part of the array. Notice, that array may be divided in non-equal parts.
3. **Sort both parts.** Apply quicksort algorithm recursively to the left and the right parts.

Partition algorithm in detail

There are two indices **i** and **j** and at the very beginning of the partition algorithm **i** points to the first element in the array and **j** points to the last one. Then algorithm moves **i** forward, until an element with value greater or equal to the pivot is found. Index **j** is moved backward, until an element with value lesser or equal to the pivot is found. If **i ≤ j** then they are swapped and i steps to the next position (**i + 1**), j steps to the previous one **(j - 1)**. Algorithm stops, when **i** becomes greater than **j**.

After partition, all values before **i-th** element are less or equal than the pivot and all values after **j-th** element are greater or equal to the pivot.E*xample.* Sort {1, 12, 5, 26, 7, 14, 3, 7, 2} using quicksort.



Q.11 Write a 'C' code for Quick Sort ?

Ans :

```
#include<stdio.h>
void print(int a[])
{
        int i;
       for( i=0;i<=8;i++)
            printf("%d ",a[i]);
}
int quickSort(int a[], int low, int high)
{
```

```
          int  i = low ;
          int  j = high ;
          int temp=0;
          int pivot = a[(left+right)/2];

do
          {
                    while (a[i] < pivot)
                              i++;
                    while (a[j] > pivot)
                              j--;
                    if (i <= j)
                    {
                              temp = a[i];
                              a[i] = a[j];
                              a[j] = temp;
                              i++;
                              j--;
                    }
          }while (i <= j);

          if (low < j)
                    quickSort (a, low, j);
          if (i < high)
                    quickSort (a, i , high);

}
void main()
{
          int array[]={12,99,4,99,12,12,13,10,13};
          printf("Before sort:\n\n");
          print(array);
          quickSort(array,0,8);
          printf("\n\nAfter sort:\n\n");
          print(array);
          printf("");
}
```

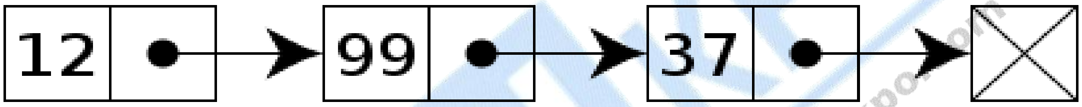
Linked List

Q.1 What is linked list?

Ans : A linked list, or one way list, is a linear collection of data elements, called nodes, where the linear order is given by means of pointers i.e., each node is divided into two parts :

1. Information of the element, and
2. The link field or next pointer field containing address of the next node in the list.

For Example :

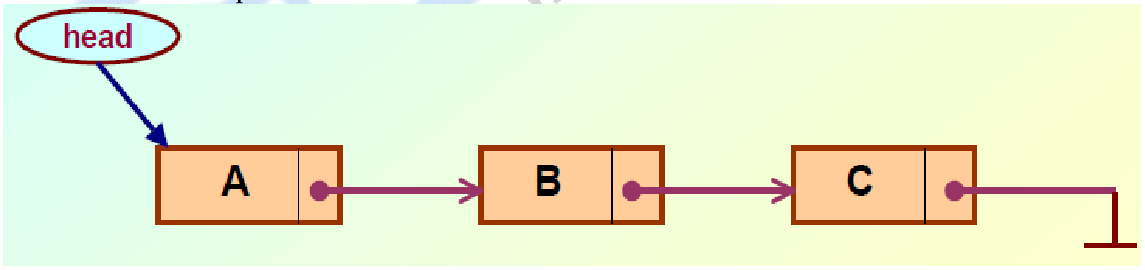


Q.2 How many types of Linked List are there?

Ans: Depending on the way in which the links are used to maintain adjacency, several different types of linked lists are possible.

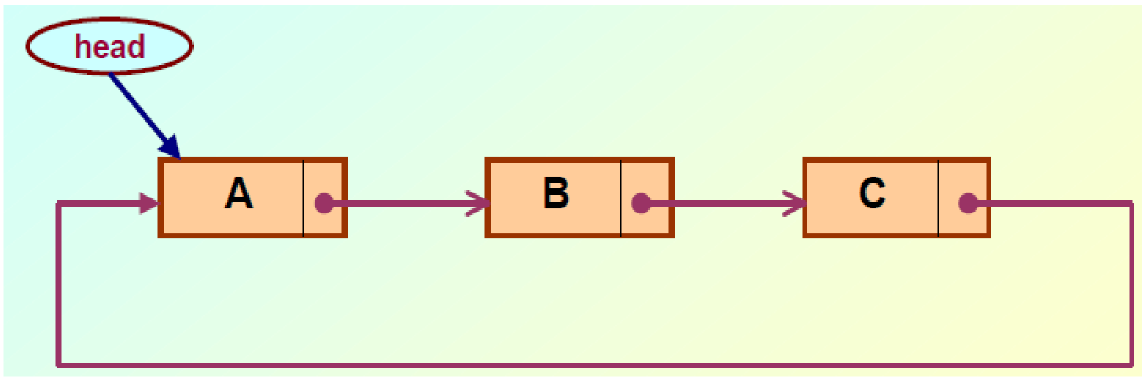
Linear singly-linked list (or simply linear list):

- Successive elements are connected by pointers
- Last element points to NULL.



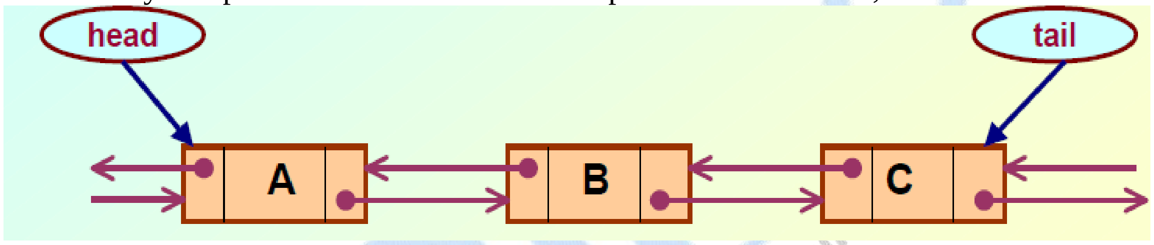
Circular linked list:

- The pointer from the last element in the list points back to the first element.
- Last node does not point to NULL i.e., the circular list is a list which does not have an end.



Doubly linked list

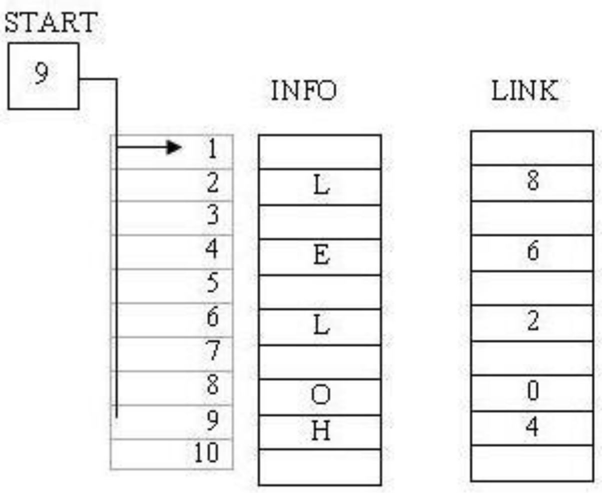
- Pointers exist between adjacent nodes in both directions.
- The list can be traversed either forward or backward.
- Usually two pointers are maintained to keep track of the list i.e., head and tail.



Q.3 How to represent Linked list?

Ans: Let LIST is linear linked list. It needs two linear arrays for memory representation. Let these linear arrays are INFO and LINK. INFO[K] contains the information part and LINK[K] contains the next pointer field of node K. A variable START is used to store the location of the beginning of the LIST and NULL is used as next pointer sentinel which indicates the end of LIST. It is shown below:

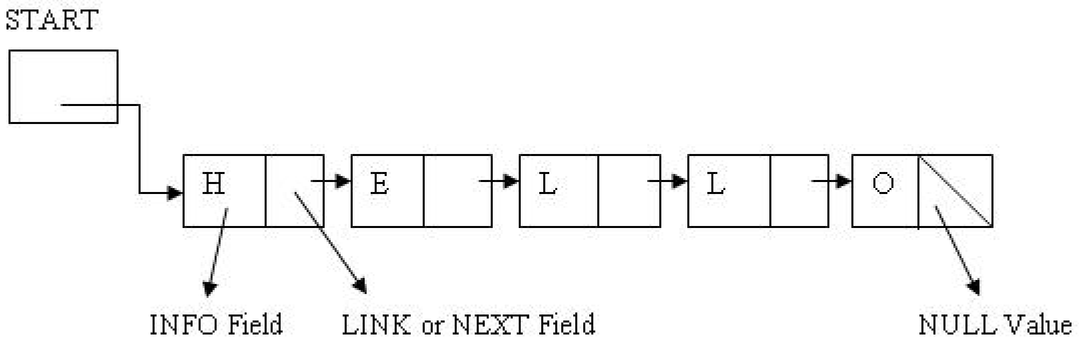
Array Representation



Here

START = 9	=>	INFO[9] = H is the first character.
LINK[9] = 4	=>	INFO[4] = E is the second character.
LINK[4] = 6	=>	INFO[6] = L is the third character.
LINK[6] = 2	=>	INFO[2] = L is the fourth character.
LINK[2] = 8	=>	INFO[8] = O is the fifth character.
LINK[8] = 0	=>	The NULL value, so the LIST ends here.

Linked Representation



Q.4 Write a 'C' code to create and traverse a linear linked list?

Ans: To create a linked list at first we need to design the structure of node i.e. what kind of data it is having. After creating node we will make a start node.

```
#include<conio.h>
#include<stdio.h>

void creation(int);
void display();
typedef struct linkedlist
```

```
{
  int num;
  struct linkedlist *next;
}node;

Node *head=NULL,*p=NULL;
void main()
{
  int n1,ch;
  clrscr();
  printf("enter the no of nodes to be entered : ");
  scanf("%d",&n1);
  do
  {
     clrscr();
     printf("\n1.creation\n2.insertion\n3.deletion\n4.display\n5.exit\n");
     printf("\nenter ur choice : ");
     scanf("%d",&ch);
     switch(ch)
     {
            case 1:
                        creation(n1);
                        break;
case 2:
                        display();
                        getch();
                        break;
case 3:
                        exit(0);
     }
  }while(ch!=3);
  getch();
}

/*  CREATION   */
void creation(int n1)
{
             int i, n ;
             head=((struct list *) malloc(sizeof(struct list)));
  p=head;
             for(i=0;i<n1;i++)
             {
                         printf("enter the %d node : ",i+1);
                        scanf("%d",&n);
                        p->num = n;
                        p->next = ((struct list *)malloc(sizeof(struct list)));
                        p=p->next;
            }
            p->next=0;
}
/*  DISPLAY   */
void display()
{
            p=head;
             printf("\nthe nodes entered are : ");
             while(p->next>0)
            {
                        printf ("%d\t", p->num);
                        p=p->next;
            }
}
```

Q.5 How to insert a node at first position in linear linked list?

Ans: In dynamic addition of NODE, we assume that the list is already there, means of non-zero length and there is a NODE pointer "target" that points to a node after which the operation(addnode) has to be done.

Algorithm:-

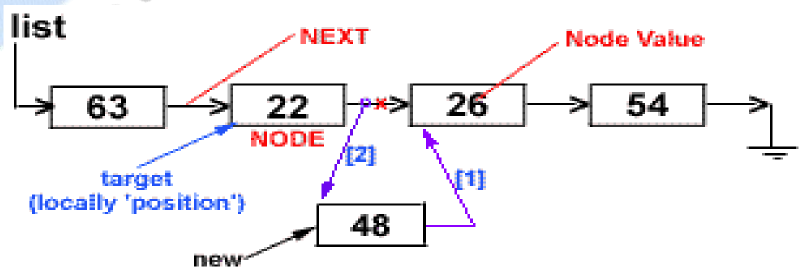
Step-1: Get the value for your NEW node to be added to the list and its Target position

Step-2: Create a NEW, empty node by calling malloc (). If malloc () returns no error then go to step-3 or else say "Memory shortage"

Step-3: Put the value inside the NEW node's node value field

Step-4: Add this NEW node at the desired position (pointed by the "target") in the LIST

Step-5: Go to step-1 till you have more values to be added to the LIST



Q.6 Write a 'C' code to insert a node in first position in a linear linked list?

Ans:

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
```

```
#include<math.h>

typedef struct linklist
{
   int data;
   struct linklist *next;
}node;

void main()
{
         node* create(node*);
         void display(node*);
         node* insert_beg(node*,int);
         node* delete_first(node*);
         node *head=NULL;
         int x;
         clrscr();

         head=create(head);
         display(head);
         printf("\nenter element to insert");
         scanf("%d",&x);
         head=insert_beg(head,x);
         printf("\ncreated link list is :");
         display(head);
         getch();
}

node* create(node *head)
{
         node *insert(node*,int);
         int x;
         printf("\nenter element to be inserted");
         scanf("%d",&x);
         while(x!=99)
         {
                  head=insert(head,x);
                  printf("\nenter element");
                  scanf("%d",&x);
         }
         return(head);
}
node* insert(node *head,int x)
{
```

```
            node* getnode(int);
            node *p,*q;
            p=getnode(x);
            q=head;
            if(head==NULL)
                        head=p;
            else
            {
                        while(q->next!=NULL)
                                    q=q->next;
                        q->next=p;
            }
            return(head);
}
node* insert_beg(node *head,int x)
{
            node* getnode(int x);
            node *p;
            p=getnode(x);
            if(head==NULL)
                        head=p;
            else
            {
                        p->next=head;
                        head=p;
            }
            return(head);
}
node* getnode(int x)
{
            node *p;
            p=(node*)malloc(sizeof(node));
            p->data=x;
            p->next=NULL;
            return(p);
}
void display(node *head)
{
                  node *p;
                        printf("\nvalues of the entered list is:");
                         p=head;
                        while(p!=NULL)
                        {
                                    printf("%d",p->data);
                        p=p->next;
                }

        node *p;
        p=head;
        while(p!=NULL)
        {

                p=p->next;
        }
        }
```

Q.7 How to delete a node from a first position in linear linked list?

Ans: Algorithm:-

Step-1: Take the value in the 'node value' field of the TARGET node in any intermediate variable

Step-2: Make the previous node of TARGET to point to where TARGET is pointing

Step-3: Free the TARGET

Step-4: Return the value in that intermediate variable

```
#include<stdio.h>
#include<stdlib.h>
#include<conio.h>
#include<math.h>

typedef struct linklist
{
   int data;
   struct linklist *next;
}node;

void main()
{
        node* create(node*);
        void display(node*);
        node* insert_beg(node*,int);
        node* delete_first(node*);
        node *head=NULL;
        int x;
        clrscr();

        head=create(head);
        display(head);
```

```
		head=delete_first(head);
		display(head);
		getch();
	}

	node* create(node *head)
	{
		node *insert(node*,int);
		int x;
		printf("\nenter element to be inserted");
		scanf("%d",&x);
		while(x!=99)
		{
			head=insert(head,x);
			printf("\nenter element");
			scanf("%d",&x);
		}
		return(head);
	}
	node* insert(node *head,int x)
	{
		node* getnode(int);
		node *p,*q;
		p=getnode(x);
		q=head;
		if(head==NULL)
			head=p;
		else
		{
			while(q->next!=NULL)
				q=q->next;
			q->next=p;
		}
		return(head);
	}
	node* delete_first(node *head)
	{
		node *p;
		p=head;
		if(head!=NULL)
			head=head->next;
		free(p);
		return(head);
	}
```

```
node* getnode(int x)
{
        node *p;
        p=(node*)malloc(sizeof(node));
        p->data=x;
        p->next=NULL;
        return(p);
}
void display(node *head)
{
        node *p;
        printf("\nvalues of the entered list is:");
         p=head;
        while(p!=NULL)
        {
                printf("%d",p->data);
                p=p->next;
        }
}
```

Q.8 Write an algorithm to search an element in a linear linked list?

Ans: Here **START** is a pointer variable which contains the address of first node. **ITEM** is the value to be searched.

```
1. Set PTR = START, LOC = 1 [Initialize PTR and LOC]
2. Repeat While (PTR != NULL)
3. If (ITEM == PTR->INFO) Then [Check if ITEM matches with INFO field]
4. Print: ITEM is present at location LOC
5. Return
6. Else
7. PTR = PTR->LINK [Move PTR to next node]
8. LOC = LOC + 1 [Increment LOC]
9. [End of If]
10. [End of While Loop]
11. Print: ITEM is not present in the list
12. Exit
```

Q.9 How to implement circular Linked List ?

Ans: In **Circular Linked List,** the list element are arranged in the same way as in the singly linked list having only one difference that the last element of the circular linked list always point to the first node of the list i.e. it means that last node does not point to NULL. In other words, we can say that the circular list is a list which does not have an end.

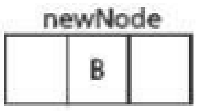
Algorithm to insert in circular Linked List

Q .10 What is Doubly LinkedList?

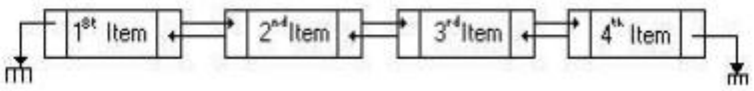
Ans: In doubly Linked list

A double linked list is defined as a collection of nodes in which each nodes has three parts:

- Data : It contains the data value for the node,
- Leftlink : leftlink contains the address of node before it
- Rightlink : rightlink contains the address of node after it.



This is how a doubly linked list looks like:



Note that

- Each node point to previous node and next node.
- Since there is no node before first node, so first node's llink contains null.
- Similarly, for the last node. There is no node after last node, hence last node's rlink contains null.

Here is the data type of node for Double Linked List

```
typedef stuct node
        {
        int data;
        node *llink;
        node *rlink;
        }
```

Q.11 How to insert an element in doubly linked list?

Ans: To insert an element in the list, first task is to get a free node, assign the element to be inserted to the info field of the node, and then new node is placed at the appropriate position by adjusting the appropriate pointer. The insertion in the list can take place at the following positions :

- ➢ At the beginning of the list
- ➢ At the end of the list
- ➢ After a given element
- ➢ Before a given element

Insertion at the beginning in doubly Linkedlist

To insert an element at the beginning of the list,first set the ’previous’ pointer field to the new node as NULL. Then we test whether the linked list is initially empty.If yes, then the element is inserted as the first and the only element of the list by performing the following steps :

- Assign NULL value to the next pointer field of the new node.
- Assign address of the new node to ‘head’ and ‘tail’ pointer vairiables

However ,if the list is initially not empty, then the element is inserted as the first element of the list by performing the following steps :

- Assign value of the head variable (the address of the first element of the existing list) to the next pointer field of the new node.
- Assigning address of the new node to the previous pointer field of the node currently pointed by ‘head’ variable i.e., first element of the existing list
- Finally, assign the address of the new node to head variable

Algorithm to insert at beginning

```
Step 1 : Begin
Step 2 : create an empty node ‘ptr’
Step 3 :set ptr -> info=item
Step 4: set ptr ->next=NULL
Step 4: if(head id NULL)  //list initially empty
Step 4.1:set ptr ->next=NULL
Step 4.2:set head=tail=ptr
Step 5: else
Step 5.1: set ptr ->next=head
Step 5.2: set head ->prev=ptr
Step 5.3: set head=ptr
Step 5.4:EndIf
Step 6    End
```

Insertion at the end in doubly Linkedlist

To insert an element at the end of the list, first set the ’next’ pointer field to the new node as NULL. Then we test whether the linked list is initially empty. If yes, then the element is inserted as the first and the only element of the list by performing the following steps :

- Assign NULL value to the previous pointer field of the new node.
- Assign address of the new node to ‘head’ and ‘tail’ pointer vairiables

However ,if the list is initially not empty, then the element is inserted as the first element of the list by performing the following steps :

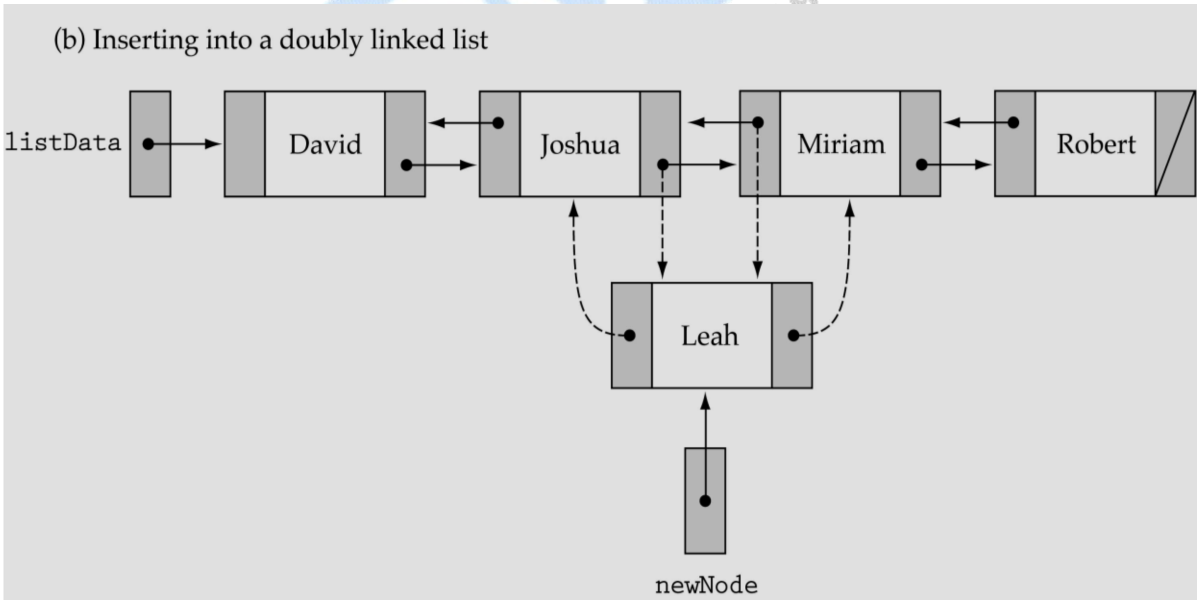
- Assign value of the tail variable (the address of the last element of the existing list) to the previous pointer field of the new node.

- Assigning address of the new node to the next pointer field of the node currently pointed by 'tail' variable i.e., last element of the existing list
- Finally, assign the address of the new node to 'tail' variable

Algorithm to insert at end in doubly linked list

```
Step 1 : Begin
Step 2 : create an empty node 'ptr'
Step 3 :set ptr -> info=item
Step 4: set ptr ->next=NULL
Step 4: if(head id NULL)  //list initially empty
Step 4.1:set ptr ->prev=NULL
Step 4.2:set head=tail=ptr
Step 5: else
Step 5.1: set ptr ->prev=tail
Step 5.2: set tail ->next=ptr
Step 5.3: set tail=ptr
Step 5.4:        EndIf
Step 6    End
```

<u>Insertion element after a given element in doubly Linkedlist</u>



Algorithm to Insert element after a given element in doubly Linkedlist

```
Step1:          Begin
Step2:          create two empty nodes 'ptr' and 'loc'
Step3:          set ptr=head
Step4:          loc=search(ptr,after)
```

```
Step5:        if(loc is NULL)     //the element after not found
Step5.1:      Exit
Step5.2       EndIf
Step6:        set ptr->info=item
Step7:        if(loc->next is NULL)
Step7.1:      set ptr->next=NULL
Step7.2:      set loc->next=ptr
Step7.3:      set ptr->prev=tail
Step7.4:      set tail=ptr
Step8:        Else
Step8.1:      set ptr->prev=loc
Step8.2       set ptr->next=loc->next
Step8.3       set(loc->next)->prev=ptr
Step8.4:      set loc->next=ptr
Step8.5:      EndIf
Step9:        End
```

<u>Insertion element before a given element in doubly Linkedlist</u>

Algorithm to Insert element after a given element in doubly Linkedlist

```
Step1: Begin
Step2:        create two empty nodes 'ptr' and 'loc'
Step3:        set ptr=head
Step4:        loc=search(ptr,after)
Step5:        if(loc is NULL)     //the element after not found
Step5.1:          Exit
Step5.2       EndIf
Step6:        set ptr->info=item
Step7:        if(loc->next is NULL)
Step7.1:          set ptr->next=NULL
Step7.2:          set loc->prev=ptr
Step7.3:          set ptr->next=head
Step7.4:          set head=ptr
Step8:        Else
Step8.1:          set ptr->prev=loc->prev
Step8.2           set ptr->next=loc
Step8.3           set(loc->prev)->next=ptr
Step8.4:          set loc->prev=ptr
Step8.5:          EndIf
Step9:  End
```

Q.12 How to delete an element from a doubly linked list?

Ans: To delete an element from the list, first the pointers are set properly and then the memory occupied by the node to be deleted is deallocated (free).
Deletion in the list can take place at the following positions.

- At the beginning of the list
- At the end of the list
- After a given element
- Before a given element

Deleting from the Beginning of the List

An element from the beginning of the list can be deleted by performing the following steps:

- Assign the value of head (address of the first element of the list) to a temporary variable (say temp)
- There are two further cases:
  1. If there is only one element in the existing list, both head and tail are set to NULL.
  2. If there is more than one element in the list then
     - Assign NULL to the prev pointer field of the second node.
     - Assign the address of the second node to head.
- Deallocate the memory occupied by the node pointed to by temp.

Algorithm to Delete an element from the Beginning of the List

```
Step 1:  Begin
Step 2:     If(head is NULL) then    //No item to delete-list empty
Step 2.1:     Exit
Step 2.2:   EndIf
Step 3:     create an empty node 'ptr'
Step 4:     set ptr=head
Step 5:     If('head' is eual to 'tail')         //one element only
Step 5.1:     set head=tail=NULL
Step 6:     Else
Step 6.1:     set (ptr->next)->prev=NULL
Step 6.2:     set head=ptr-next
Step 6.3:   EndIf
Step 7:  End
```

Deleting from the End of the List

An element from the end of the list can be deleted by performing the following steps:

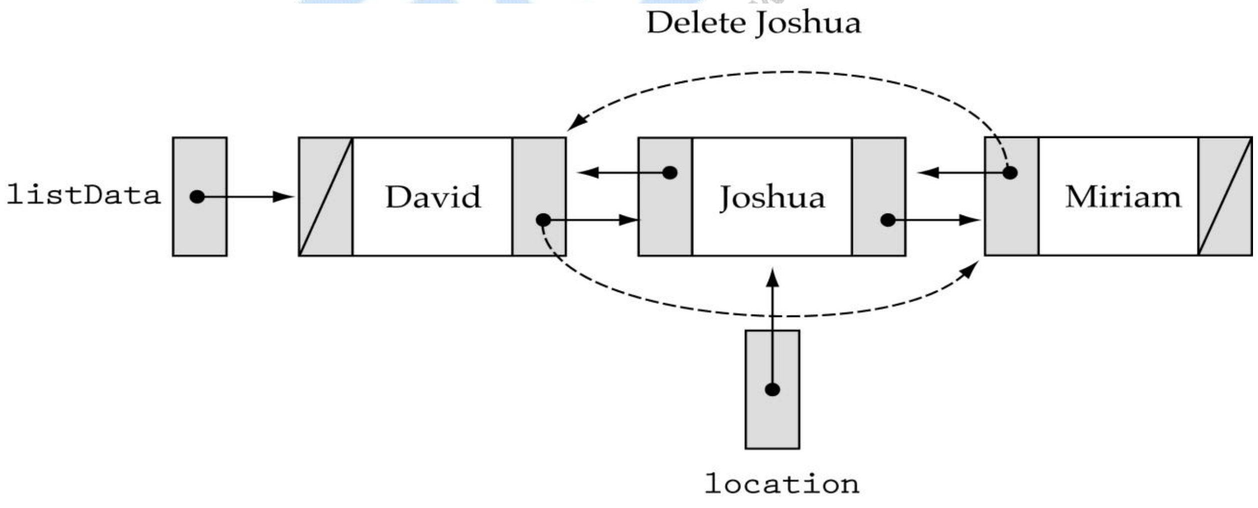
- Assign the value of tail (address of the last element of the list) to a temporary variable (say temp)
- Further there are two cases:
  1. If there is only one element in the existing list, set both head and tail to NULL.

2. If there is more than one element in the list then
    - Assign NULL to the next pointer field of the second last node.
    - Assign the address of the second last node to tail.
- Deallocate the memory occupied by the node pointed to by temp.

Algorithm to Delete an element from the end of the List

```
Step 1:  Begin
Step 2:     If(head is NULL) then    //No item to delete-list empty
Step 2.1:      Exit
Step 2.2:   EndIf
Step 3:     create an empty node 'ptr'
Step 4:     set ptr=head
Step 5:     If('head' is eual to 'tail')          //one element only
Step 5.1:      set head=tail=NULL
Step 6:     Else
Step 6.1:      set (ptr->prev)->next=NULL
Step 6.2:      set tail=ptr-prev
Step 6.3:   EndIf
Step 7:  End
```

Deleting after a Given Element



Algorithm to delete an element after a given element from list

```
Step 1: Begin
Step2:      create two empty nodes 'ptr' and 'loc'
Step3:      set ptr=head
Step4:      set loc=search(ptr,after)
Step4.1:   if(loc is NULL or loc->next is NULL)then
Step4.2       Exit
```

```
Step5.1:   Else If((loc->next)->next is NULL) then
Step5.2      set ptr=loc->next
Step5.3:     set loc->next=NULL
Step5.4:     set tail=loc
Step5.5:     freenode(ptr)
Step6:       Else
Step6.1      set ptr=loc->next
Step6.2:     set loc->next=ptr->next
Step6.3:     set (ptr->next)->prev=loc
Step6.4:     freenode(ptr)
Step 6.5:  EndIf
Step7:       End
```

Deleting before a Given Element

Algorithm to delete an element before a given element from list

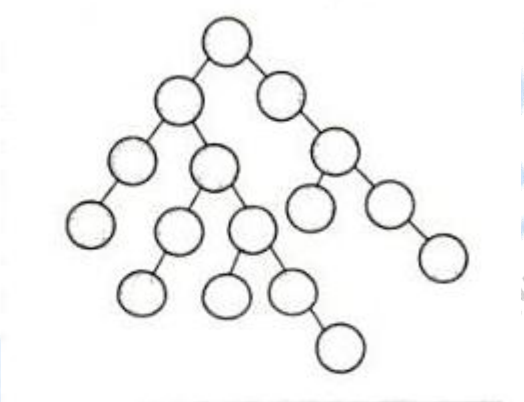
```
Step 1: Begin
Step2:       create two empty nodes 'ptr' and 'loc'
Step3:       set ptr=head
Step4:       set loc=search(ptr,before)
Step4.1:   if(loc is NULL or loc->prev is NULL)then
Step4.2      Exit
Step5.1:   Else If((loc->prev)->prev is NULL) then
Step5.2      set ptr=loc->prev
Step5.3:     set loc->prev=NULL
Step5.4:     set head=loc
Step5.5:     freenode(ptr)
Step6:       Else
Step6.1      set ptr=loc->prev
Step6.2:     set loc->prev=ptr->prev
Step6.3:     set (ptr->prev)->next=loc
Step6.4:     freenode(ptr)
Step 6.5:  EndIf
Step7:       End
```

Trees

Q.1 What is Tree Data Structure?

Ans:

- A tree is called non-linear data structure because the elements of this data structure are arranged in a non-linear fashion i.e., they use two dimensional representation.
- The tree data structure is most suitable where the hierarchy relationship among data are to be maintained.
- Advantage of this data structure is that insertion, deletion, searching etc., are more efficient than linear data structures



Tree—a non-linear (two-dimensional) representation of data

Definition

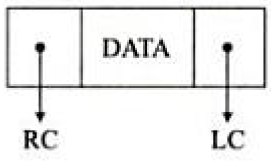
A tree is a collection of elements called nodes, one of which is designated as root along with a relation that places a hierarchical structure on the nodes.

Q.2 Explain various terminologies used in tree?

Ans:

Node

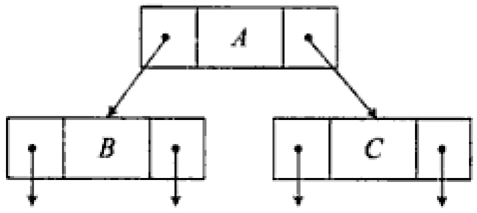
- This is the main component of any tree structure
- The concept of node in the tree is same as used in linked list
- Each individual element of a tree can be called a node.
- The node of the tree stores the actual data along with the links to other nodes**.**



Structure of a node in a tree

Parent

- The parent of a node is the immediate predecessor of that node.
- **A** is the parent nodes **B** and **C**



Child

- The immediate successors of a node are called child nodes.
- In fig ., **B** and **C** are the child nodes of **A.**
- A child which is placed at the left side is called the left child
- A child which is placed at the right side is called the right child

Link (Branch or Edge)

- A link is a pointer to a node in the tree
- A node can have more than one link
- In Fig., the node **A** has two links

Root

- A root is a specially designated node in a tree
- It is a node which has no parent.
- In fig., the node A is the root of the tree
- There can be only one root node in a tree

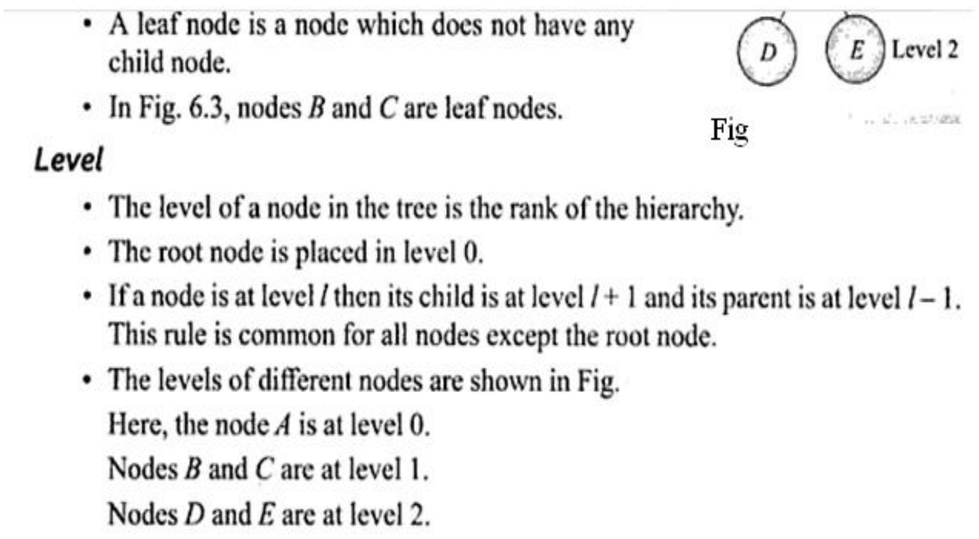
Leaf Node(or External node)

- A leaf node is a node which does not have any child node.
- In Fig., nodes B and C are leaf nodes

Level

- The level of a node in the tree is the rank of the hierarechy.
- The root node is placed in level 0.

- ➢ If a node is at level L the its child is at level L+1 and its parent is at level L-1



- A leaf node is a node which does not have any child node.
- In Fig. 6.3, nodes *B* and *C* are leaf nodes.

Fig

Level

- The level of a node in the tree is the rank of the hierarchy.
- The root node is placed in level 0.
- If a node is at level *l* then its child is at level *l* + 1 and its parent is at level *l* – 1. This rule is common for all nodes except the root node.
- The levels of different nodes are shown in Fig.
  Here, the node *A* is at level 0.
  Nodes *B* and *C* are at level 1.
  Nodes *D* and *E* are at level 2.

Q.3 Explain binary search tree with a suitable example ?

Ans: A tree is a finite set of one or more nodes such that:-There is a specially designated node called the root. The remaining nodes are partitioned into n>=0 disjoint sets T1, ..., Tn, where each of these sets is a tree.We call T1, ..., Tn the subtrees of the root.

<u>Binary Trees</u>

1. A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoint binary trees called *the left subtree* and *the right subtree*.
2. Any tree can be transformed into binary tree.by left child-right sibling representation
3 The left subtree and the right subtree are distinguished.

Additional Question
Structures Aptitude

Q.1 What is data structure?

Ans: A data structure is a way of organizing data that considers not only the items stored, but also their relationship to each other. Advance knowledge about the relationship between data items allows designing of efficient algorithms for the manipulation of data.

Q.2 List out the areas in which data structures are applied extensively?

Ans

- Compiler Design, Operating System, Database Management System,
- Statistical analysis package, Numerical Analysis,
- Graphics, Artificial Intelligence,Simulation

Q.3 What are the major data structures used in the following areas : RDBMS, Network data model & Hierarchical data model.

Ans

- RDBMS – Array (i.e. Array of structures)
- Network data model – Graph
- Hierarchical data model – Trees

Q.4 If you are using C language to implement the heterogeneous linked list, what pointer type will you use?

Ans The heterogeneous linked list contains different data types in its nodes and we need a link, pointer to connect them. It is not possible to use ordinary pointers for this. So we go for void pointer. Void pointer is capable of storing pointer to any type as it is a generic pointer type.

Q.5 Minimum number of queues needed to implement the priority queue?

Ans Two. One queue is used for actual storing of data and another for storing priorities.

Q.6 What is the data structures used to perform recursion?

Ans Stack. Because of its LIFO (Last In First Out) property it remembers its 'caller' so knows whom to return when the function has to return. Recursion makes use of system stack for storing the return addresses of the function calls. *Every recursive function has its equivalent iterative (non-recursive) function.* Even when such equivalent iterative procedures are written, explicit stack is to be used.

Q.7 ***What are the notations used in Evaluation of Arithmetic Expressions using prefix and postfix forms?***

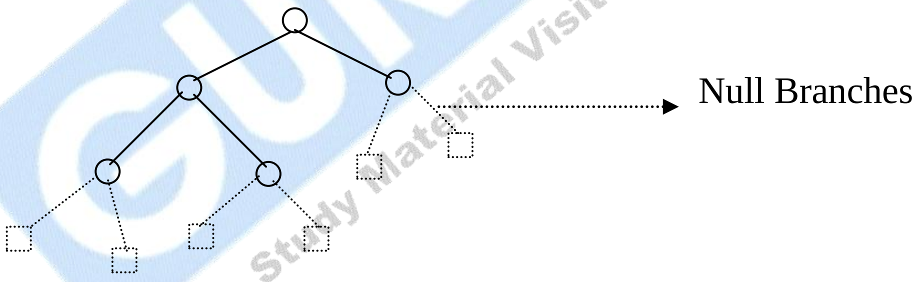
Ans Polish and Reverse Polish notations.

1. ***Convert the expression ((A + B) * C – (D – E) ^ (F + G)) to equivalent Prefix and Postfix notations.***
   Prefix Notation: ^ - * +ABC - DE + FG
   Postfix Notation: AB + C * DE - - FG + ^

2. ***Sorting is not possible by using which of the following methods?***
   (a) Insertion (b) Selection (c) Exchange (d) Deletion
   (d) Deletion.
   Using insertion we can perform insertion sort, using selection we can perform selection sort, using exchange we can perform the bubble sort (and other similar sorting methods). But no sorting method can be done just using deletion.

3. ***A binary tree with 20 nodes has null branches?***

   21
   Let us take a tree with 5 nodes (n=5)



It will have only 6 (ie,5+1) null branches. In general,
A binary tree with ***n*** *nodes has exactly* ***n+1*** *null nodes.*

Q.8 ***What are the methods available in storing sequential files ?***

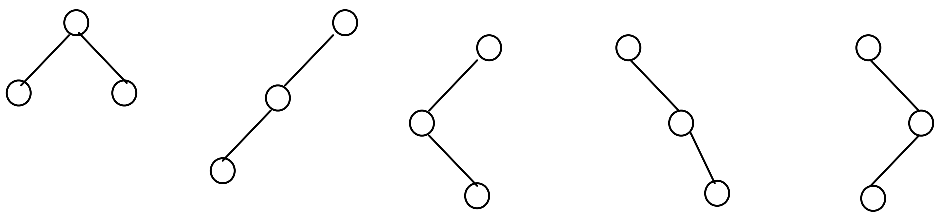
Ans

- Straight merging, Natural merging,
- Polyphase sort, Distribution of Initial runs.

Q.9 ***How many different trees are possible with 10 nodes ?***

Ans 1014

For example, consider a tree with 3 nodes(n=3), it will have the maximum combination of 5 different (ie, $2^3 - 3 = 5$) trees.



In general:
*If there are **n** nodes, there exist **2^n-n** different trees.*

Q.10 List out few of the Application of tree data-structure?
Ans

- The manipulation of Arithmetic expression,
- Symbol Table construction,
- Syntax analysis.

Q.11 List out few of the applications that make use of Multilinked Structures?
Ans

- Sparse matrix, Index generation.

Q.12 In tree construction which is the suitable efficient data structure?
Ans *(a) Array (b) Linked list (c) Stack (d) Queue (e) none*

(b) Linked list

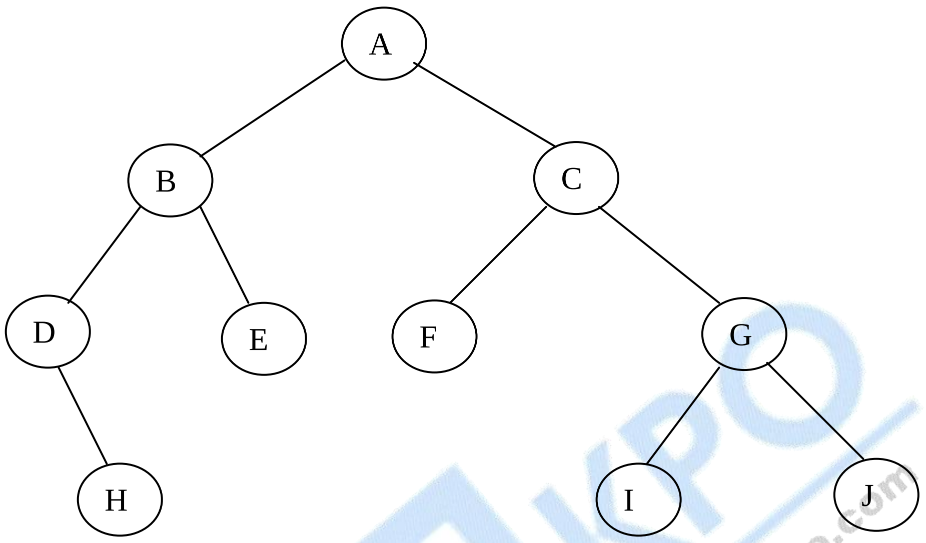
Q.13 What is the type of the algorithm used in solving the 8 Queens problem?
Ans Backtracking

Q.14 In an AVL tree, at what condition the balancing is to be done?
Ans If the 'pivotal value' (or the 'Height factor') is greater than 1 or less than –1.

Q.15 What is the bucket size, when the overlapping and collision occur at same time?
Ans One. If there is only one entry possible in the bucket, when the collision occurs, there is no way to accommodate the colliding value. This results in the overlapping of values.

Q.16 Traverse the given tree using Inorder, Preorder and Postorder traversals.
Ans

Given tree:



- Inorder : D H B E A F C I G J
- Preorder: A B D H E C F G I J
- Postorder: H D E B F I J G C A

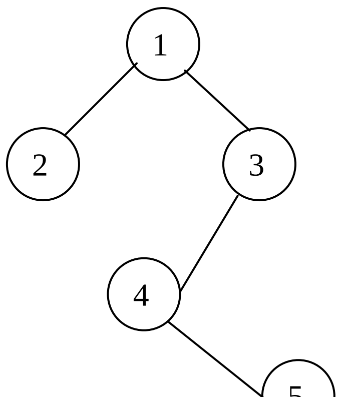
Q.17 There are 8, 15, 13, 14 nodes were there in 4 different trees. Which of them could have formed a full binary tree?

Ans 15.
In general:
There are $\mathbf{2^n}$**-1** *nodes in a full binary tree.*
By the method of elimination:

Full binary trees contain *odd* number of nodes. So there cannot be full binary trees with 8 or 14 nodes, so rejected. With 13 nodes you can form a *complete* binary tree but not a full binary tree. So the correct answer is 15.

Note:
Full and Complete binary trees are different. *All full binary trees are complete binary trees but not vice versa.*

Q.18 In the given binary tree, using array you can store the node 4 at which location?
Ans



At location 6

1	2	3	-	-	4	-	-	5
Root	LC1	RC1	LC2	RC2	LC3	RC3	LC4	RC4

where LCn means Left Child of node n and RCn means Right Child of node n

Q.19 Sort the given values using Quick Sort?

Ans

65 70 75 80 85 60 55 50 45

Sorting takes place from the pivot value, which is the first value of the given elements, this is marked bold. The values at the left pointer and right pointer are indicated using ^{L} and ^{R} respectively.

65 70^L 75 80 85 60 55 50 45^R

Since pivot is not yet changed the same process is continued after interchanging the values at ^{L} and ^{R} positions

65 45 75^L 80 85 60 55 50^R 70

65 45 50 80^L 85 60 55^R 75 70

65 45 50 55 85^L 60^R 80 75 70

65 45 50 55 60^R 85^L 80 75 70

When the L and R pointers cross each other the pivot value is interchanged with the value at right pointer. If the pivot is changed it means that the pivot has occupied its original position in the sorted order (shown in bold italics) and hence two different arrays are formed, one from start of the original array to the pivot position-1 and the other from pivot position+1 to end.

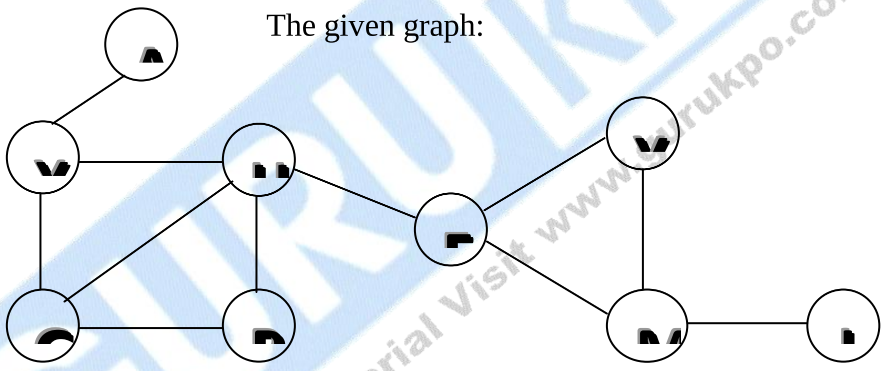
60 ^{L}	45	50	55 ^{R}	***65***	**85** ^{L}	80	75	70 ^{R}
55 ^{L}	45	50 ^{R}	***60***	***65***	**70** ^{R}	80 ^{L}	75	***85***
50 ^{L}	45 ^{R}	***55***	***60***	***65***	***70***	80 ^{L}	75 ^{R}	***85***

In the next pass we get the sorted form of the array.

45	***50***	***55***	***60***	***65***	***70***	***75***	***80***	***85***

Q.20 For the given graph, draw the DFS and BFS?
Ans

The given graph:



- *BFS:* A X G H P E M Y J
- *DFS:* A X H P E Y M J G

Q.20 Classify the Hashing Functions based on the various methods by which the key value is found.
Ans
- Direct method,
- Subtraction method,
- Modulo-Division method,
- Digit-Extraction method,
- Mid-Square method,
- Folding method,
- Pseudo-random method.

Q.21 What are the types of Collision Resolution Techniques and the methods used in each of the type?

Ans

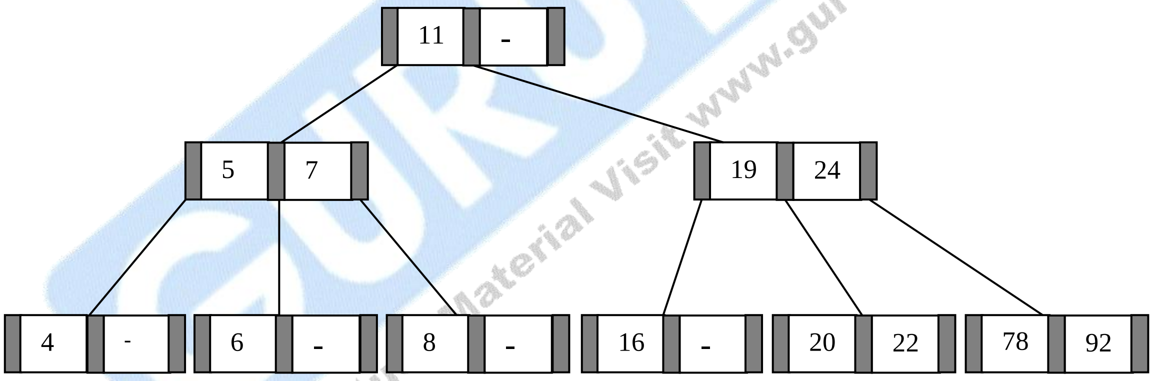
- Open addressing (closed hashing),
  The methods used include:
  Overflow block,
- Closed addressing (open hashing)
  The methods used include:
  Linked list,
  Binary tree…

Q.22 In RDBMS, what is the efficient data structure used in the internal storage representation?

Ans B+ tree. Because in B+ tree, all the data is stored only in leaf nodes, that makes searching easier. This corresponds to the records that shall be stored in leaf nodes.

Q.23 Draw the B-tree of order 3 created by inserting the following data arriving in sequence – 92 24 6 7 11 8 22 4 5 16 19 20 78

Ans



Q.24 Of the following tree structure, which is, efficient considering space and time complexities?

(a) Incomplete Binary Tree

(b) Complete Binary Tree

(c) Full Binary Tree

Ans (b) Complete Binary Tree.

By the method of elimination:

Full binary tree loses its nature when operations of insertions and deletions are done. For incomplete binary trees, extra storage is required and overhead of NULL node checking takes place. So complete binary tree is the better one since the property of complete binary tree is maintained even after operations like additions and deletions are done on it.

Q.25 ***What is a spanning Tree?***

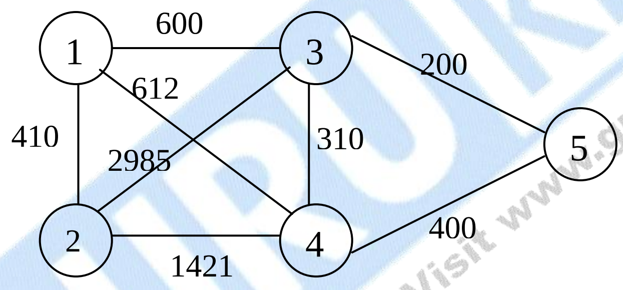
Ans A spanning tree is a tree associated with a network. All the nodes of the graph appear on the tree once. A minimum spanning tree is a spanning tree organized so that the total edge weight between nodes is minimized.

Q.26 ***Does the minimum spanning tree of a graph give the shortest distance between any 2 specified nodes?***

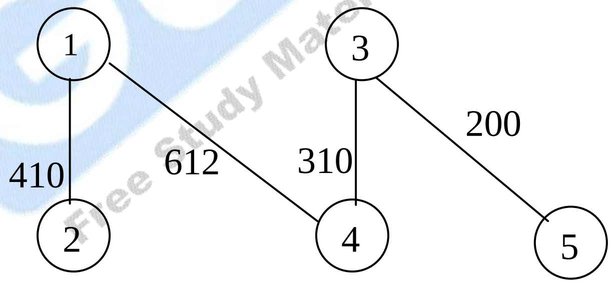
Ans No.Minimal spanning tree assures that the total weight of the tree is kept at its minimum. But it *doesn't* mean that the distance between any two nodes involved in the minimum-spanning tree is minimum.

Q.27 ***Convert the given graph with weighted edges to minimal spanning tree.***

Ans



the equivalent minimal spanning tree is:


1
3
2
4
5
410
612
310
200