UNIT-V
TREES AND BINARY TREES

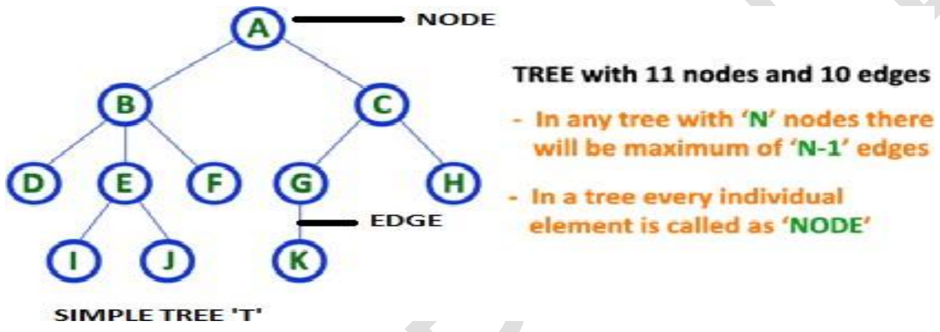
TREES

INTRODUCTION

In linear data structure data is organized in sequential order and in non-linear data structure data is organized in random order. A tree is a very popular non-linear data structure used in a wide range of applications. Tree is a non-linear data structure which organizes data in hierarchical structure and this is a recursive definition.

DEFINITION OF TREE:

Tree is collection of nodes (or) vertices and their edges (or) links. In tree data structure, every individual element is called as **Node**. Node in a tree data structure stores the actual data of that particular element and link to next element in hierarchical structure.

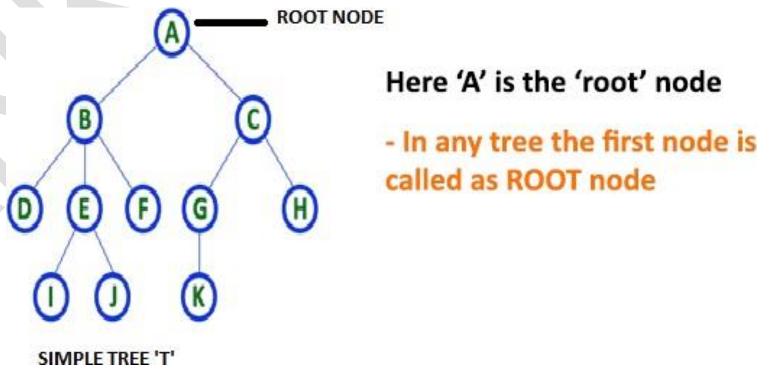

NODE
EDGE
SIMPLE TREE 'T'
TREE with 11 nodes and 10 edges
- In any tree with 'N' nodes there will be maximum of 'N-1' edges
- In a tree every individual element is called as 'NODE'

Note: 1. In a **Tree**, if we have **N** number of nodes then we can have a maximum of **N-1** number of links or edges.

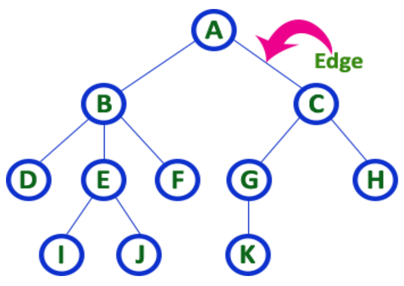
2. **Tree** has no cycles.

TREE TERMINOLOGIES:

1. Root Node: In a **Tree** data structure, the first node is called as **Root Node**. Every tree must have a root node. We can say that the root node is the origin of the tree data structure. In any tree, there must be only one root node. We never have multiple root nodes in a tree.

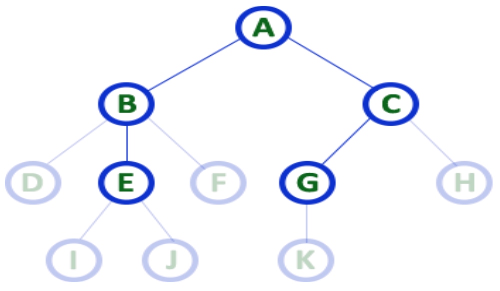

ROOT NODE
SIMPLE TREE 'T'
Here 'A' is the 'root' node
- In any tree the first node is called as ROOT node

2. Edge: In a **Tree**, the connecting link between any two nodes is called as **EDGE**. In a tree with '**N**' number of nodes there will be a maximum of '**N-1**' number of edges.



- In any tree, 'Edge' is a connecting link between two nodes.

3. Parent Node: In a **Tree**, the node which is a predecessor of any node is called as **PARENT NODE**. In simple words, the node which has a branch from it to any other node is called a parent node. Parent node can also be defined as "**The node which has child / children**".

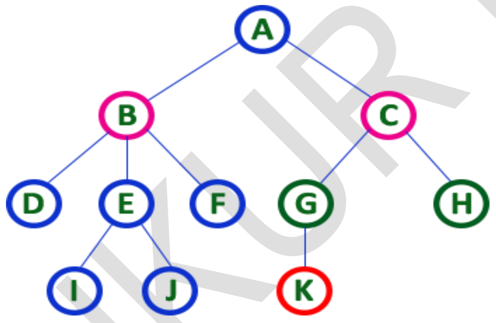


Here A, B, C, E & G are Parent nodes

- In any tree the node which has child / children is called 'Parent'
- A node which is predecessor of any other node is called 'Parent'

Here, A is parent of B&C. B is the parent of D,E&F and so on...

4. Child Node: In a **Tree** data structure, the node which is descendant of any node is called as **CHILD Node**. In simple words, the node which has a link from its parent node is called as child node. In a tree, any parent node can have any number of child nodes. In a tree, all the nodes except root are child nodes.



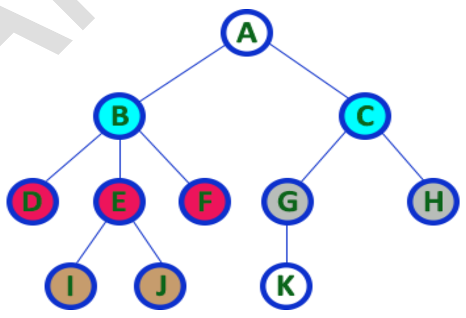
Here B & C are Children of A

Here G & H are Children of C

Here K is Child of G

- descendant of any node is called as CHILD Node

5. Siblings: In a **Tree** data structure, nodes which belong to same Parent are called as **SIBLINGS**. In simple words, the nodes with the same parent are called Sibling nodes.



Here B & C are Siblings

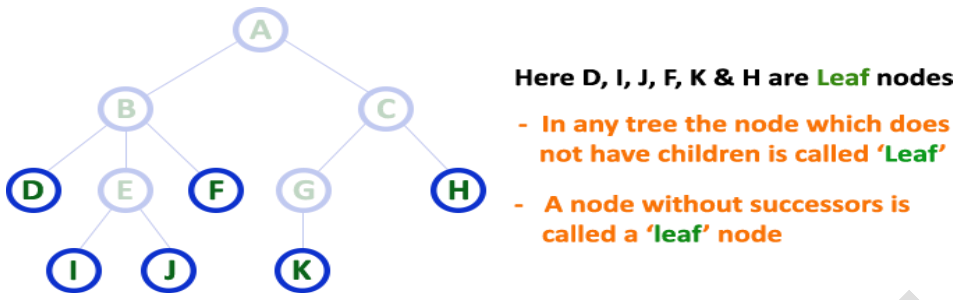
Here D E & F are Siblings

Here G & H are Siblings

Here I & J are Siblings

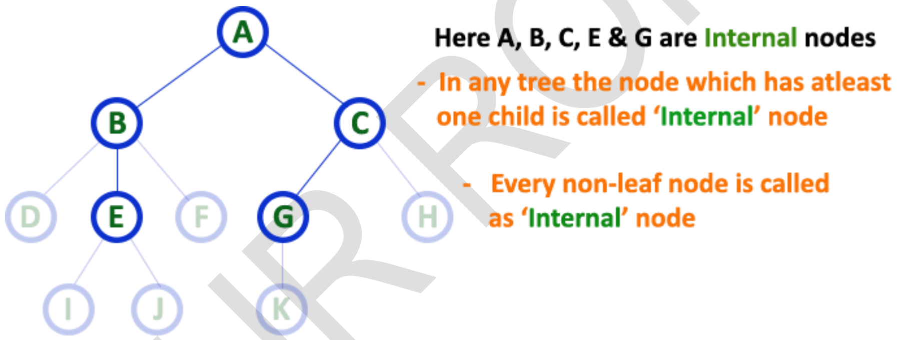
- In any tree the nodes which has same Parent are called 'Siblings'
- The children of a Parent are called 'Siblings'

6. Leaf Node: In a **Tree** data structure, the node which does not have a child is called as LEAF Node. In simple words, a leaf is a node with no child. In a tree data structure, the leaf nodes are also called as External Nodes. External node is also a node with no child. In a tree, leaf node is also called as 'Terminal' node.

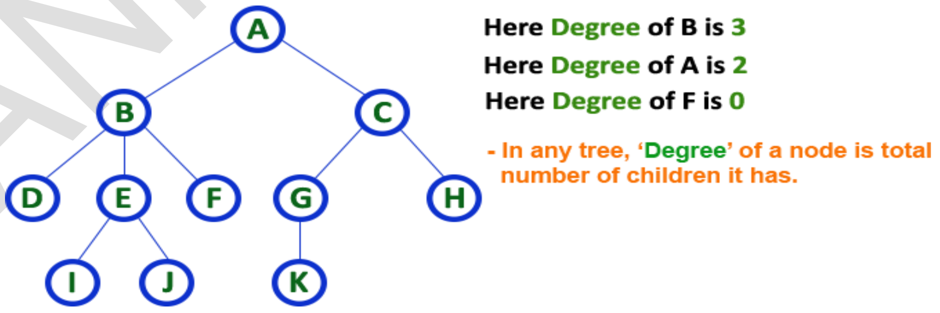

A
B
C
D
E
F
G
H
I
J
K
Here D, I, J, F, K & H are Leaf nodes
- In any tree the node which does not have children is called 'Leaf'
- A node without successors is called a 'leaf' node

7. Internal Nodes: In a **Tree** data structure, the node which has atleast one child is called as INTERNAL Node. In simple words, an internal node is a node with atleast one child.

In a **Tree** data structure, nodes other than leaf nodes are called as Internal Nodes. The root node is also said to be Internal Node if the tree has more than one node. Internal nodes are also called as 'Non-Terminal' nodes.

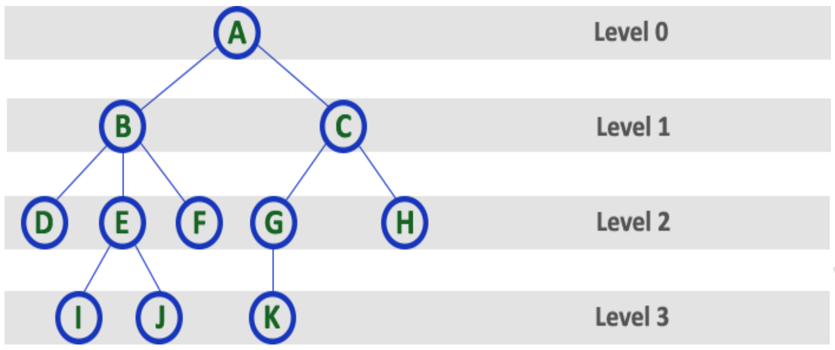

A
B
C
D
E
F
G
H
I
J
K
Here A, B, C, E & G are Internal nodes
- In any tree the node which has atleast one child is called 'Internal' node
- Every non-leaf node is called as 'Internal' node

8. Degree: In a **Tree** data structure, the total number of children of a node is called as **DEGREE** of that Node. In simple words, the Degree of a node is total number of children it has. The highest degree of a node among all the nodes in a tree is called as '**Degree of Tree**'

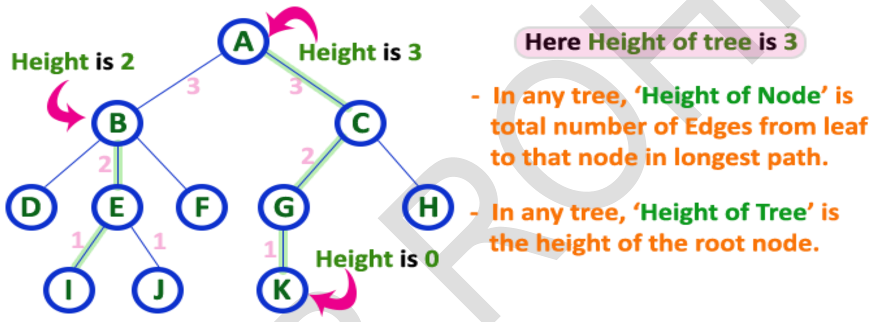


Degree of Tree is: 3

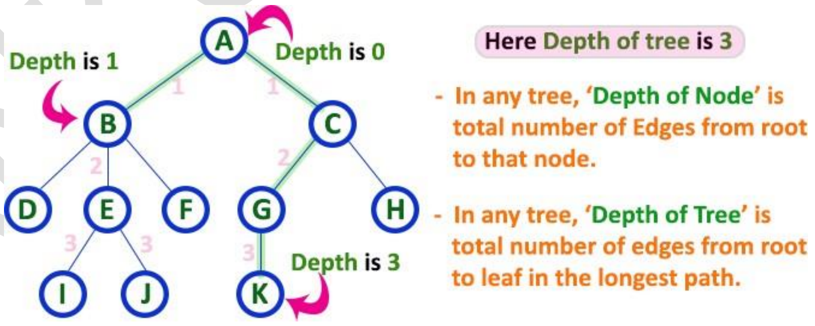
9. Level: In a **Tree** data structure, the root node is said to be at Level 0 and the children of root node are at Level 1 and the children of the nodes which are at Level 1 will be at Level 2 and so on... In simple words, in a tree each step from top to bottom is called as a Level and the Level count starts with '0' and incremented by one at each level (Step).



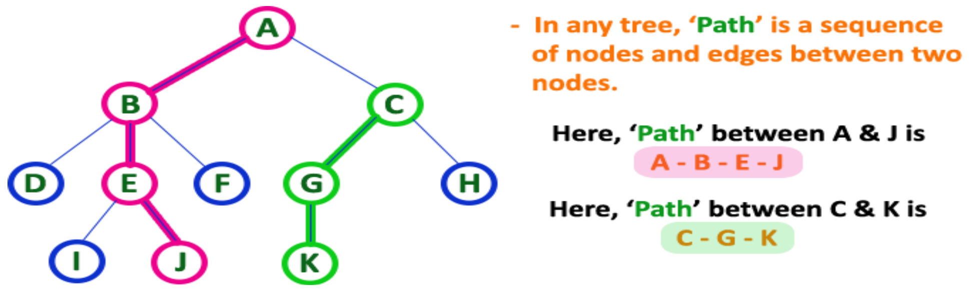
10. Height: In a **Tree** data structure, the total number of edges from leaf node to a particular node in the longest path is called as HEIGHT of that Node. In a tree, height of the root node is said to be height of the tree. In a tree, height of all leaf nodes is '0'.


Height is 2
Height is 3
Height is 0
Here Height of tree is 3
- In any tree, 'Height of Node' is total number of Edges from leaf to that node in longest path.
- In any tree, 'Height of Tree' is the height of the root node.

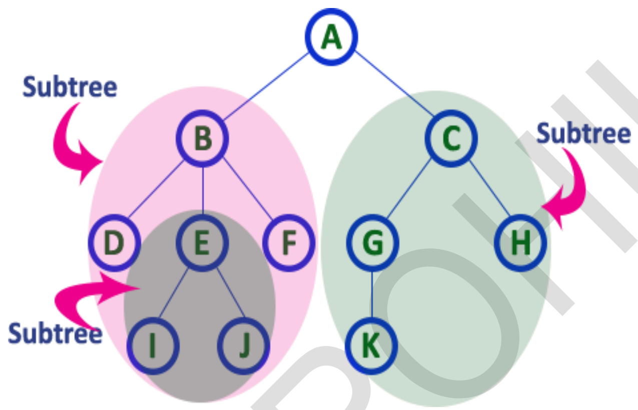
11. Depth: In a **Tree** data structure, the total number of egdes from root node to a particular node is called as **DEPTH** of that Node. In a tree, the total number of edges from root node to a leaf node in the longest path is said to be **Depth of the tree**. In simple words, the highest depth of any leaf node in a tree is said to be depth of that tree. In a tree, **depth of the root node is '0'.**


Depth is 1
Depth is 0
Depth is 3
Here Depth of tree is 3
- In any tree, 'Depth of Node' is total number of Edges from root to that node.
- In any tree, 'Depth of Tree' is total number of edges from root to leaf in the longest path.

12. Path: In a **Tree** data structure, the sequence of Nodes and Edges from one node to another node is called as **PATH** between that two Nodes. **Length of a Path** is total number of nodes in that path. In below example **the path A - B - E - J has length 4**.


- In any tree, 'Path' is a sequence of nodes and edges between two nodes.
Here, 'Path' between A & J is
A - B - E - J
Here, 'Path' between C & K is
C - G - K

13. Sub Tree: In a **Tree** data structure, each child from a node forms a subtree recursively. Every child node will form a subtree on its parent node.

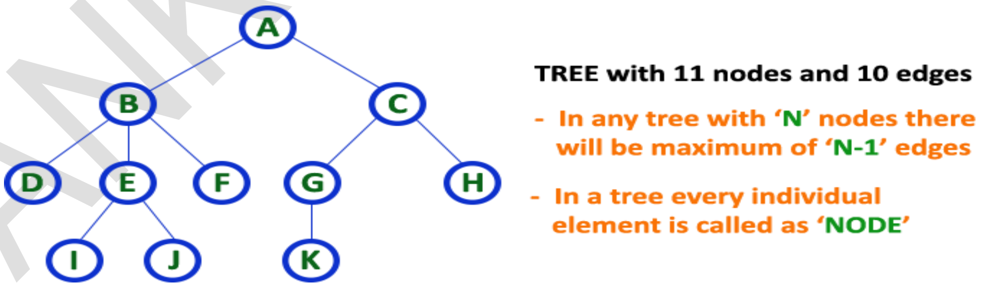


TREE REPRESENTATIONS:

A tree data structure can be represented in two methods. Those methods are as follows...

1. **List Representation**
2. **Left Child - Right Sibling Representation**

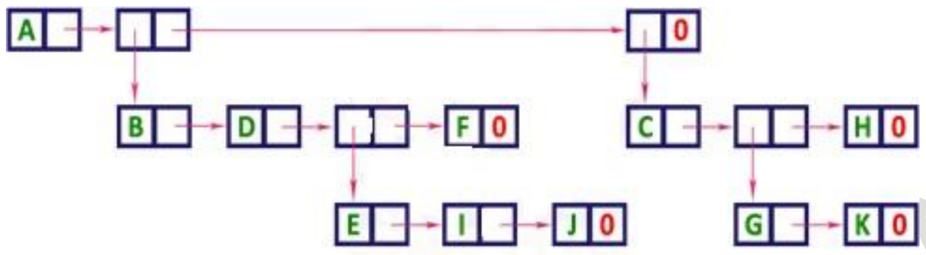
Consider the following tree...


TREE with 11 nodes and 10 edges
- In any tree with 'N' nodes there will be maximum of 'N-1' edges
- In a tree every individual element is called as 'NODE'

1. List Representation

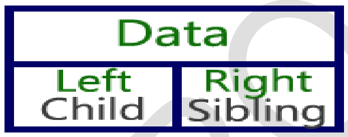
In this representation, we use two types of nodes one for representing the node with data called 'data node' and another for representing only references called 'reference node'. We start with a 'data node' from the root node in the tree. Then it is linked to an internal node through a 'reference node' which is further linked to any other node directly. This process repeats for all the nodes in the tree.

The above example tree can be represented using List representation as follows...



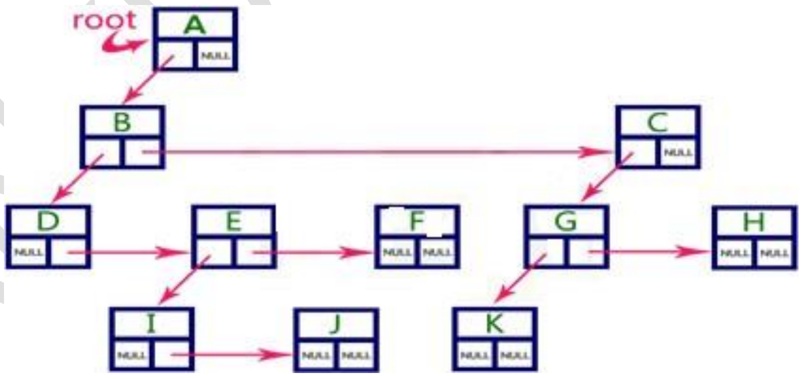
2. Left Child - Right Sibling Representation

In this representation, we use a list with one type of node which consists of three fields namely Data field, Left child reference field and Right sibling reference field. Data field stores the actual value of a node, left reference field stores the address of the left child and right reference field stores the address of the right sibling node. Graphical representation of that node is as follows...



In this representation, every node's data field stores the actual value of that node. If that node has left a child, then left reference field stores the address of that left child node otherwise stores NULL. If that node has the right sibling, then right reference field stores the address of right sibling node otherwise stores NULL.

The above example tree can be represented using Left Child - Right Sibling representation as follows...



BINARY TREE:

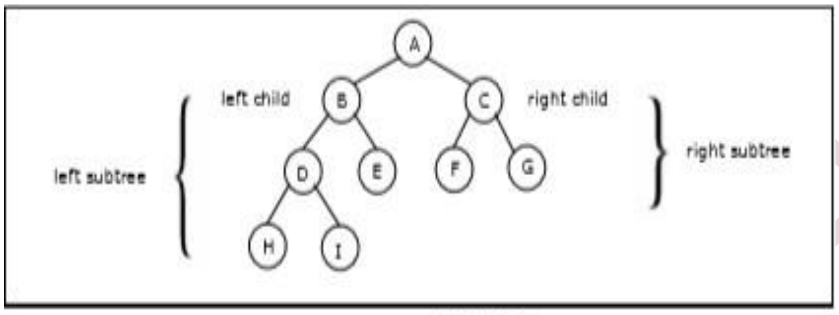
In a normal tree, every node can have any number of children. A binary tree is a special type of tree data structure in which every node can have a **maximum of 2 children**. One is known as a left child and the other is known as right child.

A tree in which every node can have a maximum of two children is called Binary Tree.

In a binary tree, every node can have either 0 children or 1 child or 2 children but not more than 2 children.

In general, tree nodes can have any number of children. In a binary tree, each node can have at most two children. A binary tree is either empty or consists of a node called the root together with two binary trees called the left subtree and the right subtree. A tree with no nodes is called as a null tree

Example:

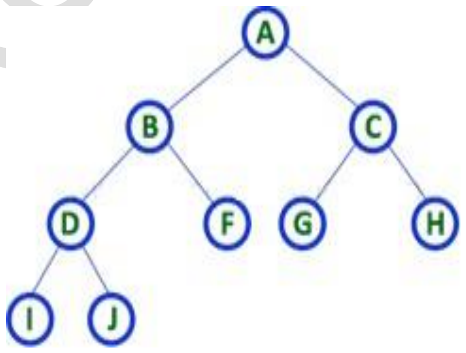


. Binary Tree

TYPES OF BINARY TREE:
1. Strictly Binary Tree:
In a binary tree, every node can have a maximum of two children. But in strictly binary tree, every node should have exactly two children or none. That means every internal node must have exactly two children. A strictly Binary Tree can be defined as follows...

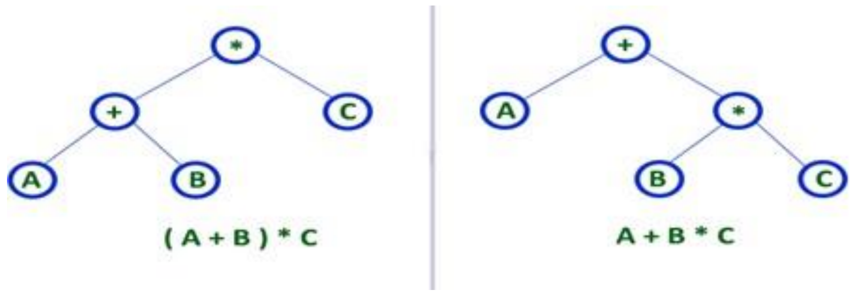
A binary tree in which every node has either two or zero number of children is called Strictly Binary Tree

Strictly binary tree is also called as **Full Binary Tree** or **Proper Binary Tree** or **2-Tree.**



Strictly binary tree data structure is used to represent mathematical expressions.

Example



2. Complete Binary Tree:

In a binary tree, every node can have a maximum of two children. But in strictly binary tree, every node should have exactly two children or none and in complete binary tree all the nodes must have exactly two children and at every level of complete binary tree there must be 2^{level} number of nodes. For example at level 2 there must be 2^2 = 4 nodes and at level 3 there must be 2^3 = 8 nodes.

A binary tree in which every internal node has exactly two children and all leaf nodes are at same level is called Complete Binary Tree.

Complete binary tree is also called as **Perfect Binary Tree.**

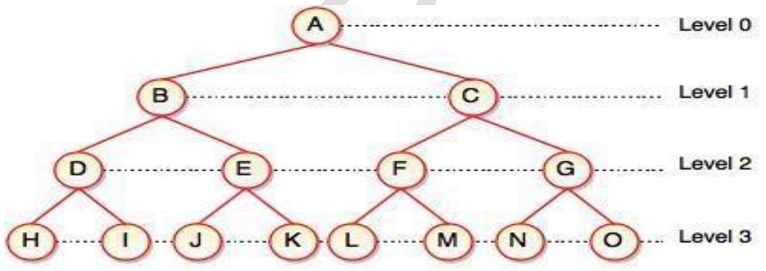
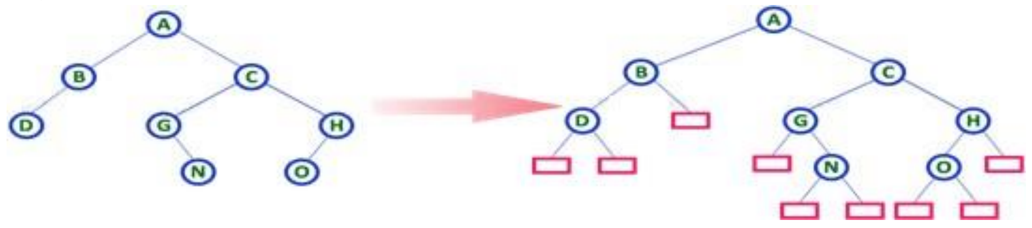


Fig. Complete Binary Tree

3. Extended Binary Tree:

A binary tree can be converted into Full Binary tree by adding dummy nodes to existing nodes wherever required.

The full binary tree obtained by adding dummy nodes to a binary tree is called as Extended Binary Tree.



In above figure, a normal binary tree is converted into full binary tree by adding dummy nodes.

4. Skewed Binary Tree:

If a tree which is dominated by left child node or right child node, is said to be a **Skewed Binary Tree**.

In a **skewed binary tree**, all nodes except one have only one child node. The remaining node has no child.

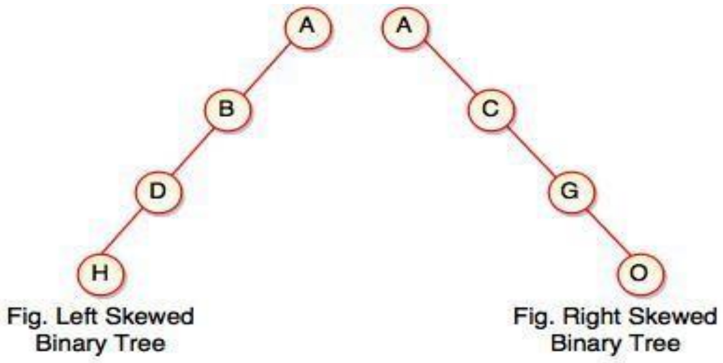


Fig. Left Skewed Binary Tree

Fig. Right Skewed Binary Tree

In a left skewed tree, most of the nodes have the left child without corresponding right child.

In a right skewed tree, most of the nodes have the right child without corresponding left child.

Properties of binary trees:

Some of the important properties of a binary tree are as follows:

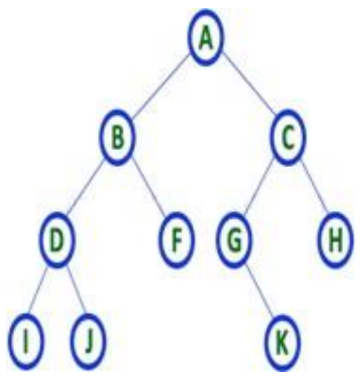
1. If h = height of a binary tree, then
   a. Maximum number of leaves = 2h
   b. Maximum number of nodes = 2h + 1 - 1
2. If a binary tree contains m nodes at level l, it contains at most 2m nodes at level l + 1.
3. Since a binary tree can contain at most one node at level 0 (the root), it can contain at most 2l node at level l.
4. The total number of edges in a full binary tree with n node is n –

BINARY TREE REPRESENTATIONS:

A binary tree data structure is represented using two methods. Those methods are as follows...

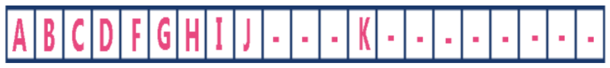
1. **Array Representation**
2. **Linked List Representation**

Consider the following binary tree...



1. Array Representation of Binary Tree

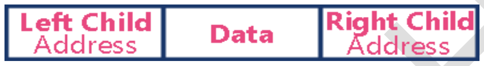
In array representation of a binary tree, we use one-dimensional array (1-D Array) to represent a binary tree.
Consider the above example of a binary tree and it is represented as follows...



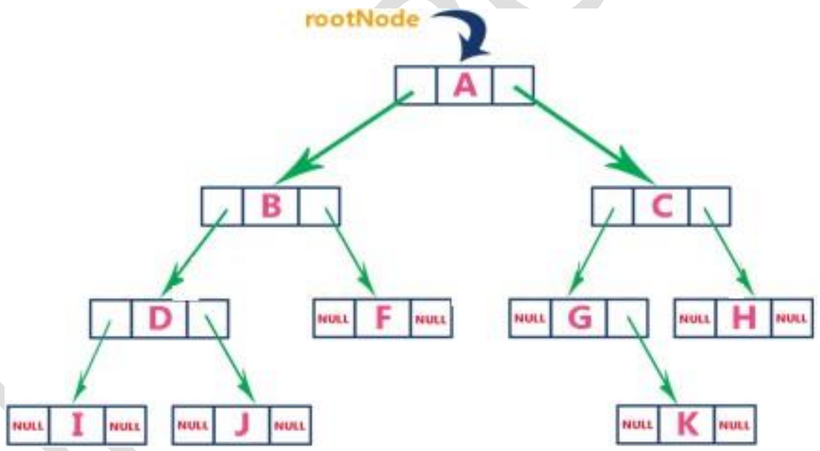
To represent a binary tree of depth **'n'** using array representation, we need one dimensional array with a maximum size of $\mathbf{2^{n+1}}$.

2. Linked List Representation of Binary Tree

We use a double linked list to represent a binary tree. In a double linked list, every node consists of three fields. First field for storing left child address, second for storing actual data and third for the right child address.
In this linked list representation, a node has the following structure...



The above example of the binary tree represented using Linked list representation is shown as follows...



BINARY TREE TRAVERSALS:

Unlike linear data structures (Array, Linked List, Queues, Stacks, etc) which have only one logical way to traverse them, binary trees can be traversed in different ways. Following are the generally used ways for traversing binary trees.
When we wanted to display a binary tree, we need to follow some order in which all the nodes of that binary tree must be displayed. In any binary tree, displaying order of nodes depends on the traversal method.
Displaying (or) visiting order of nodes in a binary tree is called as Binary Tree Traversal.
There are three types of binary tree traversals.

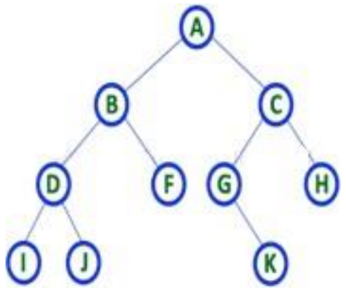
1. **In - Order Traversal**
2. **Pre - Order Traversal**
3. **Post - Order Traversal**

1. In - Order Traversal (left Child - root - right Child):

In In-Order traversal, the root node is visited between the left child and right child. In this traversal, the left child node is visited first, then the root node is visited and later we go for visiting the right child node. This in-order traversal is applicable for every root node of all sub trees in the tree. This is performed recursively for all nodes in the tree.

Algorithm:

Step-1: Visit the left subtree, using inorder.
Step-2: Visit the root.
Step-3: Visit the right subtree, using inorder.



In the above example of a binary tree, first we try to visit left child of root node 'A', but A's left child 'B' is a root node for left subtree. so we try to visit its (B's) left child 'D' and again D is a root for subtree with nodes D, I and J. So we try to visit its left child 'I' and it is the leftmost child. So first we visit **'I'** then go for its root node **'D'** and later we visit D's right child **'J'**. With this we have completed the left part of node B. Then visit **'B'** and next B's right child **'F'** is visited. With this we have completed left part of node A. Then visit root node **'A'**. With this we have completed left and root parts of node A. Then we go for the right part of the node A. In right of A again there is a subtree with root C. So go for left child of C and again it is a subtree with root G. But G does not have left part so we visit **'G'** and then visit G's right child K. With this we have completed the left part of node C. Then visit root node **'C'** and next visit C's right child **'H'** which is the rightmost child in the tree. So we stop the process.

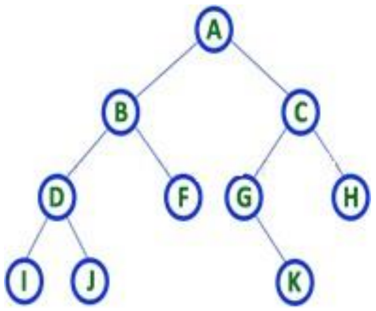
That means here we have visited in the order of **I - D - J - B - F - A - G - K - C - H** using In-Order Traversal.

2. Pre - Order Traversal (root - leftChild - rightChild):

In Pre-Order traversal, the root node is visited before the left child and right child nodes. In this traversal, the root node is visited first, then its left child and later its right child. This pre-order traversal is applicable for every root node of all subtrees in the tree. Preorder search is also called backtracking.

Algorithm:

Step-1: Visit the root.
Step-2: Visit the left subtree, using preorder.
Step-3: Visit the right subtree, using preorder.



In the above example of binary tree, first we visit root node **'A'** then visit its left child **'B'** which is a root for D and F. So we visit B's left child **'D'** and again D is a root for I and J. So we visit D's left child **'I'** which is the leftmost child. So next we go for visiting D's right child **'J'**. With this we have completed root, left and right parts of node D and root, left parts of node B. Next visit B's right child **'F'**. With this we have completed root and left parts of node A. So we go for A's right child **'C'** which is a root node for G and H. After visiting C, we go for its left child **'G'** which is a root for node K. So next we visit left of G, but it does not have left child so we go for G's right child **'K'**. With this, we have completed node C's root and left parts. Next visit C's right child **'H'** which is the rightmost child in the tree. So we stop the process.

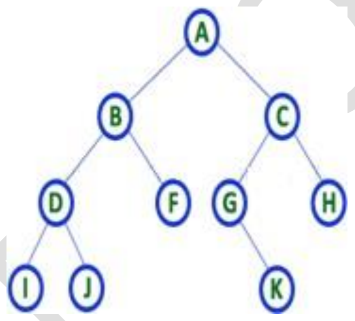
That means here we have visited in the order of **A-B-D-I-J-F-C-G-K-H** using Pre-Order Traversal.

3. Post - Order Traversal (leftChild - rightChild - root):

In Post-Order traversal, the root node is visited after left child and right child. In this traversal, left child node is visited first, then its right child and then its root node. This is recursively performed until the right most nodes are visited.

Algorithm:

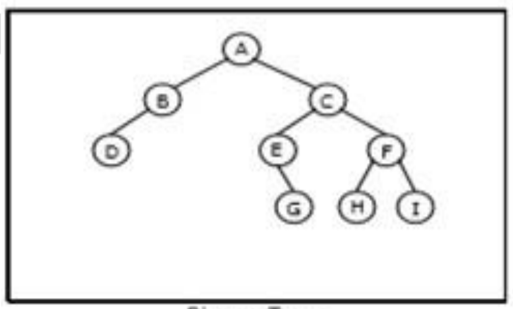
Step-1: Visit the left subtree, using postorder.
Step-2: Visit the right subtree, using postorder
Step-3: Visit the root.



Here we have visited in the order of **I - J - D - F - B - K - G - H - C - A** using Post-Order Traversal.

Example 1:

Traverse the following binary tree in pre, post, inorder and level order.



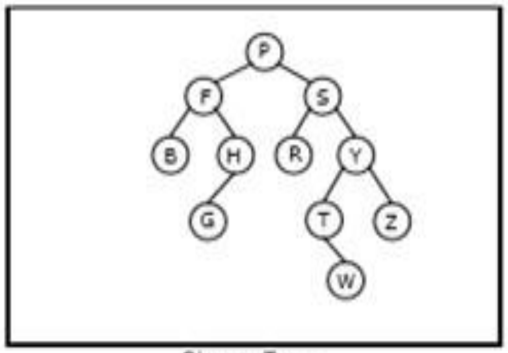
Binary Tree

- Preorder traversal yields:
  A, B, D, C, E, G, F, H, I
- Postorder traversal yields:
  D, B, G, E, H, I, F, C, A
- Inorder traversal yields:
  D, B, A, E, G, C, H, F, I

Pre, Post, Inorder and level order Traversing

Example 2:

Traverse the following binary tree in pre, post, inorder and level order.



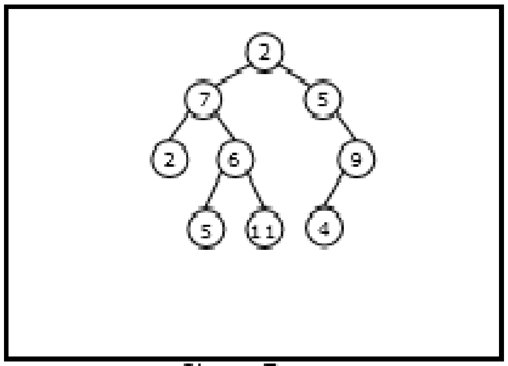
Binary Tree

- Preorder traversal yields:
  P, F, B, H, G, S, R, Y, T, W, Z
- Postorder traversal yields:
  B, G, H, F, R, W, T, Z, Y, S, P
- Inorder traversal yields:
  B, F, G, H, P, R, S, T, W, Y, Z

Pre, Post, Inorder and level order Traversing

Example 3:

Traverse the following binary tree in pre, post, inorder and level order.



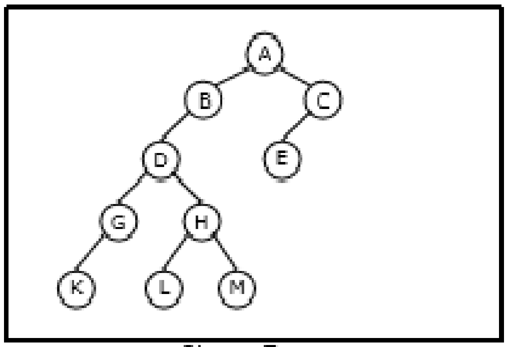
Binary Tree

- Preorder traversal yields:
  2, 7, 2, 6, 5, 11, 5, 9, 4
- Postorder travarsal yields:
  2, 5, 11, 6, 7, 4, 9, 5, 2
- Inorder travarsal yields:
  2, 7, 5, 6, 11, 2, 5, 4, 9

Pre, Post, Inorder and level order Traversing

Example 4:

Traverse the following binary tree in pre, post, inorder and level order.



Binary Tree

- Preorder traversal yields:
  A, B, D, G, K, H, L, M, C, E
- Postorder travarsal yields:
  K, G, L, M, H, D, B, E, C, A
- Inorder travarsal yields:
  K, G, D, L, H, M, B, A, E, C

Pre, Post, Inorder and level order Traversing

PROGRAMS ON DATA STRUCTURES

1. **Write a C program to implement stack using arrays.**
2. **Write a C program to implement queue using arrays.**
3. **Write a C program implement the following Stack applications**
   a) **infix into postfix**
   b) **Evaluation of the postfix expression**
4. **Write a C program to implement the following types of queues**
   a) **Priority queue**
   b) **Circular queue**
5. **Write a C program to implement the Singly Linked List**
6. **Write a C program to implement the doubly Linked List**
7. **Write a C program to implement the following search algorithms.**

   i) Linear search   ii) Binary search   iii) Fibonacci search
8. **Write a C program to implement the sorting algorithms.**
9. **Write a C program to implement binary tree using arrays and to perform binary traversals.**

   i) Inorder   ii) preorder   iii) post order
10. **Write a C program to balance a given tree.**

1: STACK USING ARRAYS

```
#include<stdio.h>
#include<conio.h>
#include<process.h>
int ch,max,item,top=-1,s[20]; void menu(void);
void push(int); int pop(void); void display(void); void main()
{
clrscr();
printf("ENTER STACK SIZE:"); scanf("%d",&max);
menu();
getch();
}
void menu()
{
printf("1.PUSH\n2.POP\n3.EXIT\n"); printf("ENTER YOUR CHOICE:");
fflush(stdin);
scanf("%d",&ch);
switch(ch)
{
case 1:printf("ENTER THE ELEMENT\n"); scanf("%d",&item);
push(item);
menu();
break;
case 2:item=pop(); menu();
break;
case 3:exit(0);
}
}
void push(int item)
{
if(top==max-1)
printf("STACK IS OVER FLOW\n"); else
{
top++;
s[top]=item;
}
display();
}
int pop()
{
if(top==-1)
{
printf("STACK IS UNDER FLOW\n"); return 0;
}
else
{
item=s[top]; top--;
}
display(); return item;
}
void display()
{
int i;
printf(" top -->");
for(i=top;i>=0;i--)
printf("%d\n\t",s[i]);
}
```

OUTPUT:

```
   Enter stack size: 3
1. Push
2. Pop
3. Exit
   Enter your choice:1
   Enter the element: 3
   Top: 3
1. Push
2. Pop
3. Exit
   Enter your choice:1
   Enter the element: 5
   Top:    5
           3
1. Push
2. Pop
3. Exit
   Enter your choice:1
   Enter the element: 9
   Top:  9    5     3
1. Push
2. Pop
3. Exit
   Enter your choice: 1
   Enter the element: 15
   Stack is overflow
   Top:    9 5 3
1. Push
2. Pop
3. Exit
   Enter your choice: 3
   Popped element is: 9
   Top:     5    3
1. Push
2. Pop
3. Exit
   Enter your choice: 2
   Popped element is: 5
   Top:    3
1. Push
```

```
2.      Pop
3.      Exit
Enter your choice: 2 Stack is underflow
```
1. Push
2. Pop
3. Exit

```
Enter your choice: 3
```

2. QUEUE USING ARRAYS

```
 #include<stdio.h>
#include<conio.h>
#include<stdlib.h> void insertion(void); void deletion(void); void display(void);
int q[10],n,i,f,r;
int f=0,r=0; void main()
{
int op;
clrscr();
printf("ENTER THE SIZE OF QUEUE:"); scanf("%d",&n);
while(1)
{
printf("\n1.INSERTION\n2.DELETION\n3.DISPLAY\n4.EXIT\n");
printf("ENTER YOUR OPTION:");
scanf("%d",&op);
switch(op)
{
case 1:insertion(); break;
case 2:deletion(); break;
case 3:display(); break; default:exit(0);
}       }       }
void insertion()
{
if(r>=n)
printf("QUEUE IS OVER FLOW"); else
{
r=r+1;
printf("\nENTER AN ELEMENT TO INSERT:"); scanf("%d",&q[r]);
if(f==0)
f=1;
}       }
void deletion()
{
if(f==0)
printf("THE QUEUE IS EMPTY"); else
{
printf("THE DELETING ELEMENT IS:%5d",q[f]); f=f+1;
if(f>r)
f=0,r=0;
}       }
```

```
void display()
{
if(f==0)
printf("QUEUE IS EMPTY"); else
for(i=f;i<=r;i++)
printf("%5d",q[i]);
}
```

OUTPUT:

```
Enter the size of queue: 2
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your option: 1
Enter an element to insert: q [1]:34
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your option: 3 34
1. Insertion
2. Deletion
3. Display
4. Exit
Enter your option: 4
```

3: STACK APPLICATIONS

a) INFIX INTO POSTFIX

b) EVALUATION OF THE POSTFIX EXPRESSION

Program:(a)

```
#include<stdio.h>
#include<conio.h>
#define MAX 50
char stack[MAX];
int top=-1
void push(char); char pop();
int priority(char); void main()
{
char a[MAX],ch; int i;
clrscr();
printf("Enter an infix expression:\t"); gets(a);
printf("\the postfix expression for the given expression is:\t"); for(i=0;a[i]!='\0';i++)
{
ch=a[i];
if((ch>='a') && (ch<='z')) printf("%c",ch);
else if(ch=='(') push(ch); else if(ch==')')
{
```

```
while((ch=pop())!='(')
printf("%c",ch);
}
else
{
while(priority(stack[top])>priority(ch))
printf("%c",pop());
push(ch);
}
}
while(top>-1) printf("%c",pop());
printf("\n");
getch();
}
void push(char ch)
{
if(top==MAX-1)
{
printf("STACK OVERFLOW"); return;
}
else
{
top++;
stack[top]=ch;  }          }
char pop()
{
int x; if(top==-1)
{
printf("STACK EMPTY");
}
else
{ x=stack[top]; top--; }
return x;
}
int priority(char ch)
{
switch(ch)
{
case '^': return 4; case '*':
case '/': return 3; case '+':
case '-': return 2; default : return 0;
} }
```

OUTPUT:

```
Enter an infix expression:
((a + b ((b ^ c – d))) * (e – (a / c)))
The postfix expression for the given expression is:
a b b c ^ d - + e a c / - *
```

Program:(b)

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<ctype.h>
void push(char);
char pop(void);
char ex[50],s[50],op1,op2; int i,top=-1;
void main()
{
clrscr();
printf("Enter the expression:"); gets(ex); for(i=0;ex[i]!='\0';i++)
{
if(isdigit(ex[i])) push(ex[i]-48); else
{
op2=pop();
op1=pop();
switch(ex[i])
{
  case '+':push(op1+op2);
             break;
 case '-':push(op1-op2);
              break;
 case '*':push(op1*op2);
            break;
   case '/':push(op1/op2);
                       break;
case '%':push(op1%op2);
            break;
case '^':push(pow(op1,op2));
            break;
}           }           }
printf("result is :%d",s[top]); getch();
}
void push(char a)
{    s[++top]=a; }
char pop()
{ return(s[top--]);
}
```

OUTPUT:

```
Enter the expression: 384 * 2 / +83----
Result is: 14
```

4: TYPES OF QUEUES

a).Priority queue
b).Circular queue

Program: (a)

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
typedef struct node
{
 int priority;
 int info;
 struct node *link;
}n;
n *getnode()
{
return ( (n *)malloc(sizeof(n)));
}
n*front=NULL,*temp=NULL,*ptr=NULL,*q=NULL;
void insertion();
void deletion();
void display();
void main()
{
  int ch;
  clrscr();
printf("\tMenu\n1.Insertion\n2.Deletion\n3.Display\n4.exit");
  while(1)
  {
    printf("Enter your choice");
    scanf("%d",&ch);
    switch(ch)
     {
      case 1:insertion();
                break;
      case 2:deletion();
                break;
      case 3:display();
                break;
      case 4:exit();
      default : printf("\nInvalid choice ");
                         break;
     }       }        }
void insertion()
{ int item,item_prty;
   temp=getnode();
   printf("Enter item to insert ");
   scanf("%d",&item);
   printf("Enter item prority    ");
   scanf("%d",&item_prty);
   temp->priority=item_prty;
   temp->info=item;
   if(front==NULL||item_prty>front->priority)
   {
    temp->link=front;
    front=temp;
   }
   else
   {
   q=front;
   while (q->link!=NULL &&q->link-> priority >=item_prty)
     q=q->link;
   temp->link=q->link;
   q->link=temp;
   }
}
void deletion()
{ if(front==NULL)
   printf("Queue is underflow");
   else
   {
    temp=front;
    printf("Deleted item is %d\n",
       temp->info);
    front=front->link;
    free(temp);
   }
}
void display()
{
 ptr=front;
   if(front==NULL)
   printf("Queue is underflow");
   else
   {
    printf("Queue is :\n");
    printf("priority item :\n");
    while(ptr!=NULL)
    {
     printf("%5d %5d\n",ptr->priority,ptr->info);
     ptr=ptr->link;

    }
  }
}
```

OUTPUT:

```
1 - Insert an element into queue
2 - Delete an element from queue
3 - Display queue elements
4 - Exit
Enter your choice: 1

Enter value to be inserted: 20
 Enter your choice: 1
 Enter value to be inserted: 45
 Enter your choice: 1
 Enter value to be inserted: 89
 Enter your choice: 3
 89 45 20
Enter your choice: 1
 Enter value to be inserted: 56
 Enter your choice: 3
 89 56 45 20
Enter your choice: 2
 Enter value to delete: 45
 Enter your choice: 3
 89 56 20
Enter your choice: 4
```

<u>Program:</u> (b)

```
#include<stdio.h>
#include<conio.h>
#define max 3
int q[max],rear=-1,front=-1;
void main()
{ int ch;
clrscr();
do
{ printf("\nqueue implementation\n");
  printf("1.insert 2.delete 3.display 4.exit\n");
  printf("enter your choice\n");
  scanf("%d",&ch);
  switch(ch)
  {  case  1:insert();  break;
    case  2:delete();  break;
    case 3:display(); break;
    case 4:exit(1);
    default:printf("wrong choice\n"); break;
  }
}while(ch<=4);
getch();
}
insert()
{ int item;
```

```
  if(rear==max-1)
  { printf("queue overflow\n"); }
  else
  { if(front==-1)
    front=0;
    printf("insert the element in queue:");
    scanf("%d",&item);
    rear++;
    q[rear]=item;
  }
}
delete()
{ if(front==-1)
  { printf("queue underflow\n");
  }
  else
  { printf("element deleted from queue is:%d\n",q[front]);
    front++;
    if(front==max)
    front=rear=-1;
  }
}
display()
{ int i;
 if(front==-1)
 printf("queue is empty\n");
 else
 { printf("queue is :\n");
   for(i=front;;i++)
   { printf("%2d",q[i]);
     if(i==rear)
      return;
   }   } }
```

OUTPUT:

```
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice:1
Enter element to cqueue: 10
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 1
Enter element to circular queue: 20
1. Insert
2. Delete
```

```
3. Display
4. Exit
Enter your choice: 2
Deleted element from circular queue is: 10
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 3
Elements from circular queue is: 20
1. Insert
2. Delete
3. Display
4. Exit
Enter your choice: 4
```

5: SINGLY LINKED LIST

Program:

```
#include<stdio.h>
#include<conio.h>
#include<stdlib.h>
#include<alloc.h> struct node
{
int data;
struct node* link; };
typedef struct node* pnode; pnode head=NULL;
void menu(void); void insbeg(int); void delbeg(void); void insend(int); void delend(void);
void insafter(int,int); void delmid(int); void display(void); void main()
{
int ch,x,pos; clrscr(); while(1)
{
menu();
printf("enter ur choice\n"); scanf("%d",&ch);
switch(ch)
{
 case 1: printf("enter element to insert\n");
 scanf("%d",&x);
insbeg(x);
      break;
case 2:delbeg();
break;

case 3: printf("enter element to insert\n");
 scanf("%d",&x);
insend(x);
      break;
      case 4:delend();
break;
 case 5: printf("enter element,pos to insert\n");
scanf("%d%d",&x,&pos);
insafter(x,pos);
break;
 case 6:printf("enter position of element to delete\n"); scanf("%d",&pos);
delmid(pos);
break;
case 7:display();
        break;
case 8:exit(0);
}           }           }
void menu()
{
printf("1.insbeg\n2.delbeg\n3.insend\n4.delend\n");
printf("5.insafter\n6.delmid\n7.display\n8.exit\n");
}
void insbeg(int x)
{
 pnode ptr;
 ptr=(pnode)malloc(sizeof(struct node));
 ptr->data=x;
ptr->link=head; head=ptr;
}
void delbeg()
{
pnode tmp; int x;
if(head==NULL)
{
printf("list is empty\n"); return;
}
tmp=head;
head=tmp->link;
printf("deleted element is %d\n",tmp->data);
 free(tmp);
}
void insend(int x)
{
pnode tmp,ptr;
ptr=(pnode)malloc(sizeof(struct node));
ptr->data=x;
ptr->link=NULL; if(head==NULL)
{
head=ptr;
}
else
{
tmp=head;
while(tmp->link!=NULL)
tmp=tmp->link;
tmp->link=ptr;
```

```
  }          }
  void delend()
  {
  pnodeprev=NULL,ptr=head;
  int x; if(head==NULL)
  {
     printf("listis empty\n"); return;
  }
  while(ptr->link!=NULL)
  {
  prev=ptr; ptr=ptr->link;
  }
  if(prev==NULL)
      head=NULL;
  else
  prev->link=NULL;
printf("deleted node:%d\n",ptr->data);
free(ptr);
  }
  void insafter(intx,intpos)
  {
     pnodetmp=head,ptr;
    int  i;
    for(i=1;i<pos;i++)
  {
      tmp=tmp->link;
     if(tmp==NULL)
  {
  printf("position out of range\n"); return;
  }          }
  ptr=(pnode)malloc(sizeof(struct node));
  ptr->data=x;
  ptr->link=tmp- >link;
   tmp->link=ptr;
  }
  void delmid(intpos)
  {
  pnodeprev=NULL,tmp=head;
 int i;

  if(head==NULL)
  {
  printf("list is empty\n"); return;
  }
  for(i=0;i<pos;i++)
  {
  prev=tmp; tmp=tmp->link;
  if(tmp==NULL)
  { printf("position out of range\n"); return;
```

```
    }          }

      if(prev!=NULL)
         prev->link=tmp->link;
    else
    head=tmp->link;
  printf("deleted element: %d\n",tmp->data);
   free(tmp);
    }
    void display()
    {
    pnodeptr=head;
    while(ptr!=NULL)
    {
    printf("%d-->",ptr->data);
     ptr=ptr->link;
    }
    printf("\n");
    getch();
    }
```

OUTPUT:

```
1.  Ins beg
2.  el beg
3.  Ins end
4.  Del end
5.  Ins after
6.  Del mid
7.  Display
8.  Exit
    Enter your choice: 1
    Enter element to insert: 94
1.  Ins beg
2.  Del beg
3.  Ins end
4.  Del end
5.  Ins after
6.  Del mid
7.  Display
8.  Exit
    Enter your choice: 1
    Enter element to insert: 90
1.  Ins beg
2.  Del beg
3.  Ins end
4.  Del ed
5.  Ins after
6.  Del mid
7.  Display
8.  Exit
```

```
      Enter your choice 5
      Enter element, Pos to insert
55 2

1.  Ins beg
2.  Del beg
3.  Ins end
4.  Del end
5.  Ins after
6.  Del mid
7.  Display
8.  Exit
      Enter your choice 7 90->94->55->
1.  Ins beg
2.  Del beg
3.  Ins end
4.  Del end
5.  Ins after
6.  Del mid
7.  Display
8.  Exit
      Enter your choice 8
```

6: DOUBLY LINKED LIST

Program:

```
#include<stdio.h>
#include<conio.h> struct node
{
struct node *prev; int data;
            struct node *nxt;
             }
*head=NULL,*curr=NULL,*curr1=NULL,*p;

void insert(int pos)
{
int count=1,i; p=head;
while(p->nxt!=NULL)
{
count++; p=p->nxt;
}
p=head;
if(pos<=count+1)
{
curr=(struct node*)malloc(sizeof(struct node));
printf("Enter the node:");
scanf("%d",&curr->data);
curr->nxt=NULL;
curr->prev=NULL;
 if(head==NULL)
{
head=curr; }
else if(pos==1)
{
head->prev=curr;
 curr->nxt=head;
 head=curr;  }
else
{    for(i=1;i<(pos-1);i++) p=p->nxt;
curr->prev=p; curr->nxt=p->nxt;
p->nxt->prev=curr;
p->nxt=curr;
}
printf("\n%d inserted at pos:%d!\n",curr->data,pos);
}
else
printf("\nEnter a valid position!");

}
void deletenode(int data)
{
int found=0; curr=head; if(head->data == data)
{
(head->nxt)->prev=NULL; head=head->nxt;
printf("\n%d deleted!\n",curr->data);
free(curr);
}
else
{
curr=curr->nxt;
while(curr->nxt!=NULL)
{
if(curr->data==data)
{
found=1;
break;
}
else
curr=curr->nxt;
}
if(found==1 || curr->data==data)
{
curr1=curr->prev; curr1->nxt=curr->nxt; (curr->nxt)->prev=curr1; printf("\n%d
deleted!\n",curr->data); free(curr);
}
else
printf("\n%d is not present in the list!\n",data);
}     }
void display()
```

```
{
curr=head;
if(head==NULL)
        printf("\nList is empty!\n"); else {
printf("\nList:\n"); while(curr->nxt!=NULL) {
printf("%d\t",curr->data); curr=curr->nxt;
}
printf("%d\n",curr->data);
}
}
void main()
{
intop,data;
clrscr();
printf("Creation of Doubly Linked List\n");
curr=(struct node*)malloc(sizeof(struct node));
curr->nxt=NULL;
curr->prev=NULL; printf("Enter the first node:"); scanf("%d",&curr->data); head=curr;
head->nxt=NULL; head->prev=NULL; do
{ printf("\nDOUBLY LINKED LIST OPERATIONS:\n1.Insert a node ");
printf("\n2.Delete a node\n3.Display\n4.Exit\n");
printf("\nSelect an operation:"); scanf("%d",&op);
switch(op)
{
case 1:
{   printf("Enter the position where you want to insert the node:"); scanf("%d",&data);
insert(data); break;
 }
case 2:{
printf("Enter the node to be deleted:"); scanf("%d",&data);
deletenode(data); break; }

case 3:        {
display();
break; }
case 4:
break;
default:
printf("\nEnter a valid option!");
}
}while(op!=4);
getch();
}
```

OUTPUT:

Doubly linked list operations

1. Insert node
2. Delete node
3. Display

```
4.  Exit
    Select an operation: 1
    Enter the position to insert node: 1 Enter the node: 59
    59 inserted at pos: 1
1.  Insert node
2.  Delete node
3.  Display
4.  Exit
    Select an operation: 4
```

7: SEARCHING ALGORITHMS

i) Linear search ii) Binary search
iii) Fibonacci search

<u>Program:</u> (i)and (ii)

```
#include <stdio.h>
#include <conio.h>
# include <stdlib.h>
void main()
{
Int a[10],n,flag=0,i,lb,ub,key,mid,ch;
clrscr();
printf("enter the size of the elements\n");
scanf("%d",&n);
printf("enter the elements\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
printf("enter any key element to search\n");
scanf("%d",&key);
printf("menu\n");
printf("\n1.linear search\n2.binary search \n");
printf("enter your choice:\n"); scanf("%d",&ch);
switch(ch)
{ case 1:for(i=0;i<n;i++) if(a[i]==key)
{
flag=1;
break;}
case 2:for(lb=0,ub=n-1;lb<=ub;)
{
mid=(lb+ub)/2;
if(key==a[mid])
{
flag=1;
break;
}
else if(key<a[mid]) ub=mid-1;
else lb=mid+1;
}
break;
default:exit(0);
}
if(flag==1)
printf("seach is successful");
else
printf("search is not successful \n");
 getch();

}
```

OUTPUT:

```
Enter the size of the elements 5
Enter the elements 32 6 3 9 5
Enter any element to search 3
Menu
```

1. Linear search
2. Binary search Enter your choice: 2

```
Search is successful
```

Program:(iii)

```
#include<stdio.h>
void main()
{
  int n,a[50],i,key,loc,p,q,r,m,fk;
  clrscr();
  printf("\nenter number elements to be entered");
  scanf("%d",&n);
  printf("enter elements");
  for(i=1;i<=n;i++)
  scanf("%d",&a[i]);
  printf("enter the key element");
  scanf("%d",&key);
  fk=fib(n+1);
  p=fib(fk);
  q=fib(p);
  r=fib(q) ;
  m=(n+1)-(p+q);
  if(key>a[p])
   p=p+m;
  loc=rfibsearch(a,n,p,q,r,key);
  if(loc==0)
   printf("key is not found");
  else
   printf("%d is found at location %d",key,loc);
  getch();
}
int fib(int m)
{
  int a,b,c;
  a=0;
  b=1;
  c=a+b;
  while(c<m)
  {
   a=b;
   b=c;
   c=a+b;
  }
  return b;
}
int rfibsearch(int a[],int n,int p,int q,int r,int key)
{
  int t;
  if(p<1||p>n)
  return 0;
  else if(key==a[p])
   return p;
  else if(key<a[p])
 {
   if(r==0)
   return 0;
   else
   {
    p=p-r;
    t=q;
    q=r;
    r=t-r;
    return rfibsearch(a,n,p,q,r,key);
   }    }
  else
  {
   if(q==1)
   return 0;
   else
   {
    p=p+r;
    q=q-r;
    r=r-q;
    return rfibsearch(a,n,p,q,r,key);
   }    }    }
```

OUTPUT:

```
Enter the number elements to be entered 8
Enter the elements 1 3 2 5 4 6 7 9
Enter the key element 9
8 is found at location 8
```

8: SORTING ALGORITHMS

Program: Bubble Sort

```
#include<stdio.h>
#include<conio.h>
#define TRUE 1
#define FALSE 0
void bubblesort(int x[],int n); void main()
{
intnum[10],i,n;
clrscr();
printf("Enter the no of elements\n"); scanf("%d",&n);
printf("Enter the elements\n"); for(i=0;i<n;i++) scanf("%d",&num[i]); bubblesort(num,n);
printf("sorted elements are\n"); for(i=0;i<n;i++) printf("%d\t",num[i]);
getch();}
void bubblesort(int x[],int n)
{
inthold,j,pass,K=TRUE;
for(pass=0;pass<n-1&&K==TRUE;pass++)
{
K=FALSE;
for(j=0;j<n-pass-1;j++) if(x[j]>x[j+1])
{
K=TRUE;
hold=x[j];
x[j]=x[j+1];
x[j+1]=hold;}}}
```

OUTPUT:

```
Enter the no of elements 5
Enter the elements 36 23 59 68 2
Sorted elements are 2 23 36 59 68
```

Program: selection sort

```
#include<stdio.h>
#include<conio.h> void main()
{
intn,i,j,a[10],min,t;
clrscr();
printf("enter how many elements\n"); scanf("%d",&n);
printf("enter the elements\n"); for(i=0;i<n;i++) scanf("%d",&a[i]); for(i=0;i<n-1;i++)
{
min=i;
for(j=i+1;j<n;j++)
{
if(a[min]>a[j])
min=j;
}
t=a[i];
a[i]=a[min];
a[min]=t;
}
 printf("the sorted elements are \n"); for(i=0;i<n;i++)
printf("%5d",a[i]);
getch();
}
```

OUTPUT:

```
Enter how many elements 5
Enter the elements 56 48 46 23 35
The sorted elements are 23 35 46 56 98
```

Program: insertion sort

```
#include<stdio.h>
#include<conio.h> void main()
{
intn,i,a[10],t,j;
clrscr();
printf("enter how many elements\n");
 scanf("%d",&n);
printf("enter the elements\n");
for(i=0;i<n;i++)
scanf("%d",&a[i]);
for(i=1;i<n;i++)
{
for(j=i;j>0;j--)
{
if(a[j]<a[j-1])
{
t=a[j]; a[j]=a[j-1]; a[j-1]=t;
}     }  }
printf("the sorted elements are\n"); for(i=0;i<n;i++)
printf("%5d",a[i]);
getch();
}
```

OUTPUT:

```
Enter how many elements 5
Enter the elements 26 36 98 12 5
The sorted elements are 5 12 26 36 98
```

Program:Quick sort

```
#include<stdio.h>
#include<conio.h>
void quick(int a[10],intlb,int n);
void main()
{
intn,i,a[10];
clrscr();
printf("enter how many elements \n"); scanf("%d",&n);
printf("enter the elements \n"); for(i=0;i<n;i++) scanf("%d",&a[i]); quick(a,0,n-1);
printf("the sorted elements are \n"); for(i=0;i<n;i++)
printf("%d \n",a[i]); getch();
}
void quick(int a[],intlb,intub)
{
inti,j,t,key;
if(lb>ub) return; i=lb;
j=ub;
key=lb;
while(i<j)
{
while(a[key]>a[i])
i++;
while(a[key]<a[j]) j--;
if(i<j)
{
t=a[i];
a[i]=a[j];
a[j]=t;
}
}
t=a[j];
a[j]=a[key];
a[key]=t;
quick(a,0,j-1);
quick(a,j+1,ub);
}
```

OUTPUT:

```
Enter how many elements 5
Enter the elements 65 23 89 68 71
The sorted elements are 23 65 68 71 89
```

program:heap sort

```
#include<conio.h>
void maxheap(int [],int,int);
 void buildmaxheap(int a[],int n)
{
int i; for(i=n/2;i>=1;i--)
{
maxheap(a,i,n);
}
}
void maxheap(int a[],inti,int n)
{
intR,L,largest,t;
L=2*i;
R=2*i+1;
if((L<=n) && (a[L]>a[i])) largest=L;
else largest=i;
if((R<=n) && (a[i]>a[largest])) largest=R;
if(largest!=i)
{
t=a[i];
a[i]=a[largest];
a[largest]=t;
maxheap(a,largest,n);
}
}
void heapsort(int a[],int n)
{
inti,temp;
buildmaxheap(a,n);
for(i=n;i>=2;i--)
{
temp=a[1];
a[1]=a[i];
a[i]=temp; maxheap(a,1,i-1);
}
}
vod main()
{
int a[50],i,n; clrscr();
printf("Enter the size of array : "); scanf("%d",&n);
printf("Enter the elements of array \n"); for(i=1;i<=n;i++)
{
scanf("%d",&a[i]);
}
heapsort(a,n);
printf("sorted array is \n"); for(i=1;i<=n;i++)
{
printf("%d\t",a[i]);
}
getch();
}
```

OUTPUT:

```
Enter the size of array: 4
Enter the elements of array: 35 21 95 17
Sorted array is: 17 21 35 95
```

Program: merge sort

```
#include<stdio.h>
#include<conio.h>
void merge(int [],int ,int ,int );
void part(int [],int ,int );
int main()
{
intarr[30];
inti,size;
 printf("\n\t------- Merge sorting method ------ \n\n");
 printf("Enter total no. of elements : "); scanf("%d",&size);
```

```
for(i=0; i<size; i++)
{printf("Enter %d element : ",i+1);
   scanf("%d",&arr[i]);
}
part(arr,0,size-1);
printf("\n\t------- Merge sorted elements -------\n\n"); for(i=0; i<size; i++)

printf("%d ",arr[i]); getch();
return 0;
}
void part(intarr[],intmin,int max)
{
int mid; if(min<max)
{
mid=(min+max)/2;
part(arr,min,mid);

part(arr,mid+1,max);
merge(arr,min,mid,max);
}
}
void merge(intarr[],intmin,intmid,int max)
{
inttmp[30];
inti,j,k,m;
j=min;
m=mid+1;
for(i=min; j<=mid && m<=max ; i++)
{
if(arr[j]<=arr[m])
{
tmp[i]=arr[j];
j++;
}
else
{
tmp[i]=arr[m];
m++;
}
}
if(j>mid)
{
for(k=m; k<=max; k++)
{
tmp[i]=arr[k];
i++;
}
}
else
```

```
{
for(k=j; k<=mid; k++)
{
tmp[i]=arr[k];
i++;
}
}
  for(k=min; k<=max; k++) arr[k]=tmp[k];
}
```

OUTPUT:

```
Merge sorting method
Enter total no of elements: 4
Enter 4 elements:
35
95
17
21
Merge sorted elements: 17 21 35 95
```

9. BINARY TREE TRAVERSALS

i) Preorder ii) Inorder iii) Postorder

Program:

```
#include<stdio.h>
#include<conio.h>
#include<malloc.h>
#include<stdlib.h> struct node
{
int data;
struct node *left; struct node *right; };
typedefstruct node *pnode; pnode root=NULL;
void insert(intval)
{
pnodep,q,t; t=(pnode)malloc(sizeof(struct node)); t->left=t->right=NULL;
t->data=val; if(root==NULL)
{
root=t;
return;
}
p=root;q=NULL;
while(p)
{
if(p->data==val)
{
return;
}
q=p;
if(val<p->data) p=p->left;
else if(val>p->data)
p=p->right;
```

```
}
if(val<q->data) q->left=t; if(val>q->data) q->right=t;
}
int search(int key)
{
pnode p=root;
 while(p)
{
if(p->data==key) return 1;
else if(key<p->data) p=p->left;
else if(key>p->data) p=p->right;
}
return 0;
}
void inorder(pnode p)
{
if(p==NULL)
return; inorder(p->left); printf("%3d",p->data); inorder(p->right);
}
void preorder(pnode p)
{
if(p==NULL)
return;
printf("%3d",p->data);
preorder(p->left);
preorder(p->right);
}
void postorder(pnode p)
{
if(p==NULL)
return;
postorder(p->left);
postorder(p->right);
printf("%3d",p->data);
}
int main()
{
intch,x;
clrscr();
while(1)
{
printf("\n1.insertion");
printf("\n2.inorder");
printf("\n3.preorder");
printf("\n4.postorder");
printf("\n5.search");
printf("\n6.exit");
printf("\nenterur choice\n");
scanf("%d",&ch);
```

```
switch(ch)
{
case 1:printf("enter an elements\n"); scanf("%d",&x);
insert(x);
break;
case 2:inorder(root); break;
case 3:preorder(root); break;
case 4:postorder(root); break;
case 5:printf("enter key elements\n");
 scanf("%d",&x);
if(search(x))
printf("found"); else
printf("not found"); break;
case 6:exit(0);
}         }         }
```

OUTPUT:

```
Tree traversal
Enter the number of terms to add 7 Enter the item 15
Enter the item 7 Enter the item 9 Enter the item 18 Enter the item 6 Enter the item 21 Enter
the item 2
In order traversal 2 6 7 9 15 18 21
Pre order traversal 15 7 6 2 9 18 21
Post order traversal 2 6 9 7 21 18 15
```

10. BALANCE A TREE

Program:

```
#include  <stdio.h>
#include <stdlib.h>
struct btnode
{
   int value;
   struct btnode *l;
   struct btnode *r;
};

typedef struct btnode N;
N* bst(int arr[], int first, int last);
N* new(int val);
void display(N *temp);
 int main()
{
   int arr[] = {10, 20, 30, 40, 60, 80, 90};
   N *root = (N*)malloc(sizeof(N));
   int n = sizeof(arr) / sizeof(arr[0]), i;

   printf("Given sorted array is\n");
   for (i = 0;i < n;i++)
```

```
        printf("%d\t", arr[i]);
    root = bst(arr, 0, n - 1);
    printf("\n The preorder traversal of binary search tree is as follows\n");
    display(root);
    printf("\n");
    return 0;
}
N* new(int val)
{
    N* node = (N*)malloc(sizeof(N));

    node->value = val;
    node->l = NULL;
    node->r = NULL;
    return node;
}

N* bst(int arr[], int first, int last)
{
    int mid;
    N* temp = (N*)malloc(sizeof(N));
    if (first > last)
        return NULL;
    mid = (first + last) / 2;
    temp = new(arr[mid]);
    temp->l = bst(arr, first, mid - 1);
    temp->r = bst(arr, mid + 1, last);
    return temp;
}
void display(N *temp)
{
    printf("%d->", temp->value);
    if (temp->l != NULL)
         display(temp->l);
    if (temp->r != NULL)
        display(temp->r);
}
```

OUTPUT:

```
Given sorted array is
10    20    30    40    60    80    90
The preorder traversal of binary search tree is as follows
40->20->10->30->80->60->90
```