

12/02/24
MON.

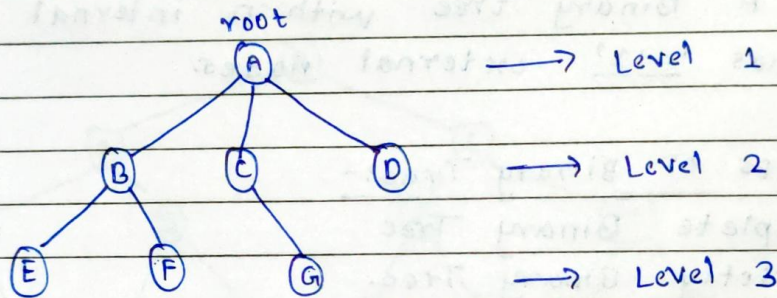
UNIT: NO: 2: (TWO): - TREES.

o TREE:-

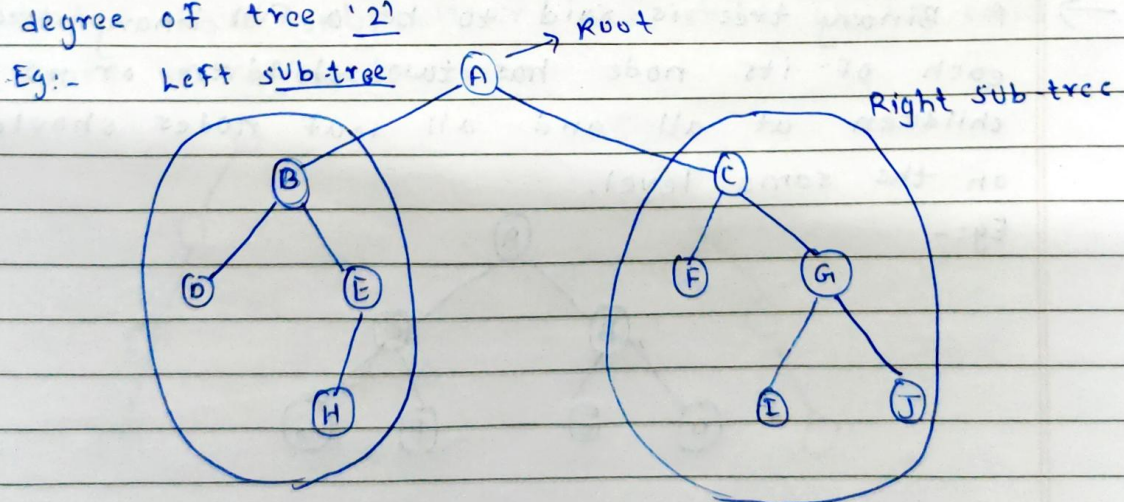
A Tree is an abstract model of a hierarchical structure that consists of nodes with a parent-child relationship.

→ Tree is a sequence of Nodes

- o There is a starting node known as a root node.
- o Every node other than root has a parent node

# Binary Tree:-

→ A Binary Tree (T) is an m -array tree with $m=2$ which has a left subtree and right subtree with degree of tree '2'



D PROPERTIES OF BINARY TREE:-

① A Tree with n nodes has exactly $(n-1)$ edges or branches.

② For m nodes at level L , it contains at most ' $2m$ ' nodes at level $L+1$.

③ Max. no. of Nodes in binary tree = $2^h - 1$
($h \rightarrow$ height of tree)

④ The minimum height of a binary tree with n nodes is $\log_2(n+1) - 1$

⑤ A Binary tree with n internal nodes has ' $n+1$ ' external nodes.

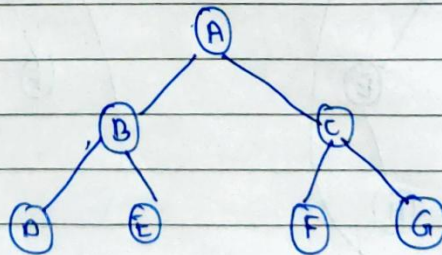
\Rightarrow TYPES OF Binary Tree:-

1. Complete Binary Tree
2. strictly Binary Tree.
3. Full Binary Tree
4. skewed Binary Tree.

o Full Binary Tree:-

\rightarrow A Binary tree is said to be a Full binary tree if each of its node has two children or no children at all and all leaf nodes should be on the same level.

Eg:-



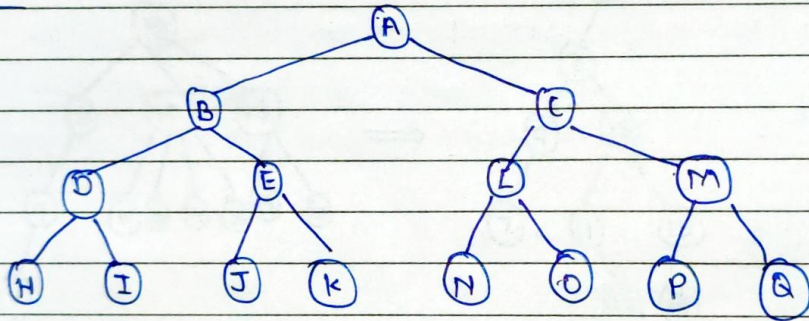
◦ Complete Binary Tree:-

→ A Complete Binary Tree is a binary tree in which every level, except possibly the last, is completely filled, and all nodes are as far left as possible.

◦ A complete binary tree of depth d is called strictly binary tree if all of whose leaves are at level d .

◦ A complete binary tree has 2^d nodes at every depth d and $2^d - 1$ non leaf nodes.

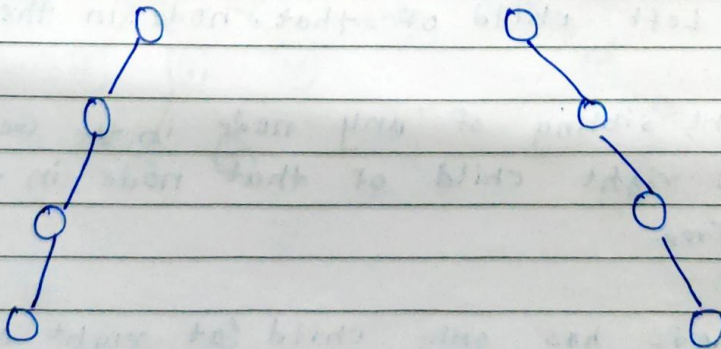
Eg:-



◦ Skewed Binary Tree:-

→ A skewed binary tree is the one in which all nodes have only either one child or no child.

Types of skewed tree:-



◦ Strictly Binary Tree:-

- ① IF every non-leaf node in a binary tree has non-empty left and right sub-trees, then such a tree is called a strictly Binary Tree.
- ② Or, to put in another way, all of the nodes in a strictly binary tree are of degree zero or two, never degree-one.

CONVERSION:- General Tree to Binary Tree

◦ Algorithm:-

- ① The root of the Binary Tree is the root of the General Tree.
- ② The left child of a node in the General tree is the left child of that node in the Binary Tree
- ③ The right sibling of any node in the General Tree is the right child of that node in the Binary Tree.

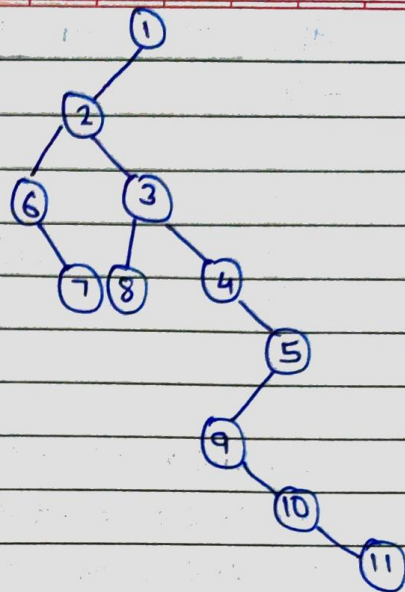
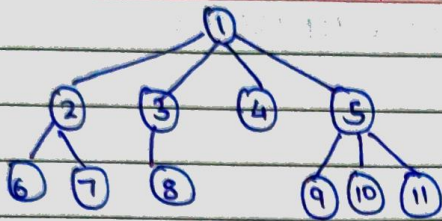
→ For Eg:-

IF a node has only child (at right or left), then it will be attached to left in binary Tree (BT)

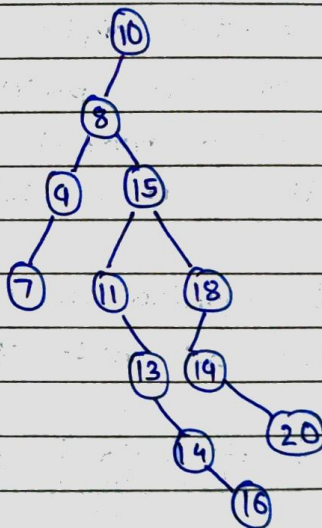
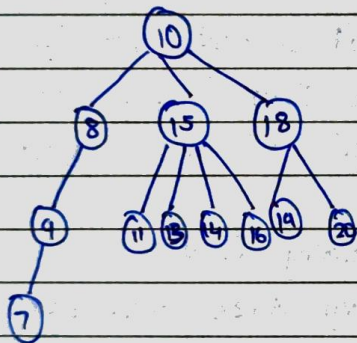
GENERAL TREE

BINARY TREE

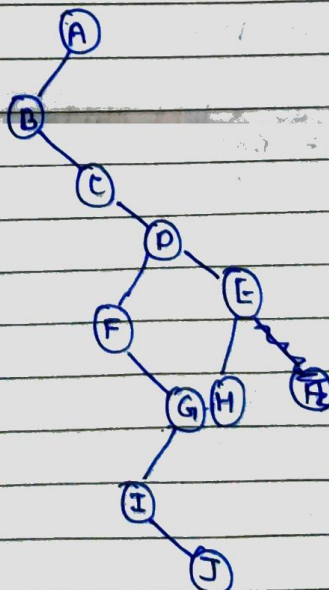
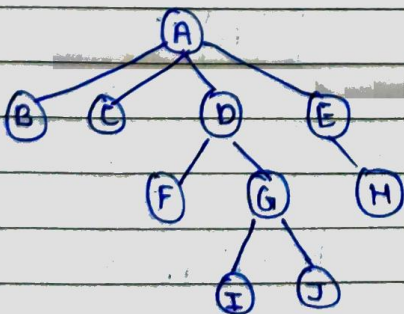
1



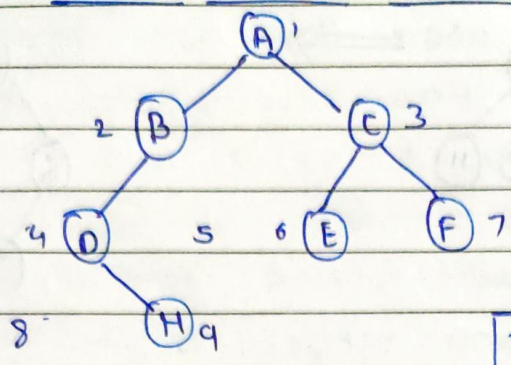
2



3



TREE AND ITS ARRAY REPRESENTATION :-



0	1	2	3	4	5	6	7	8	9
7	A	B	C	D	-	E	F	-	H

- Index of left child of a node $i = 2i$
- Index of the right child of a node $i = 2i + 1$
- Index of the parent of a node $i = i/2$.
- Sibling of a node i will be found at the location $i+1$, if i is a left child of its parent.

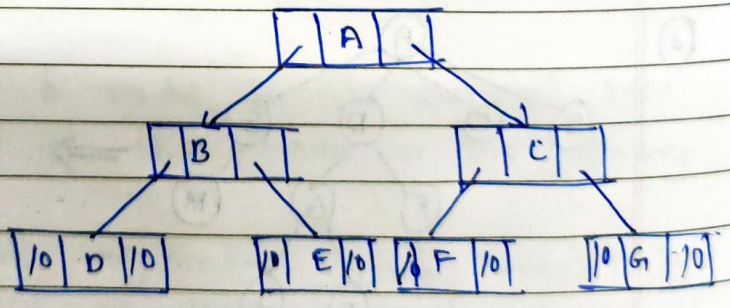
Linked Organization of a Binary Tree:-

→ To represent node of binary tree, the structure is:

Left child Address	Data	Right child Address
--------------------	------	---------------------

```

struct Node
{
    Node *left_child;
    int data;
    Node *right_child;
};
  
```



Tree Traversal:-

(1) Pre-Order

Root → Left → Right

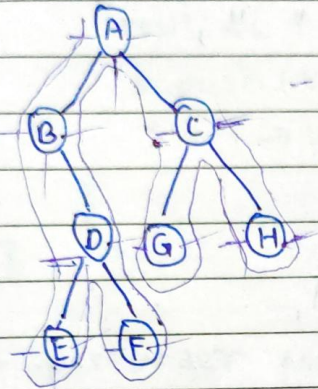
(2) In post Order

Left \rightarrow Root \rightarrow Right.

(3) Post Order.

Left \rightarrow Right \rightarrow Root.

(1)

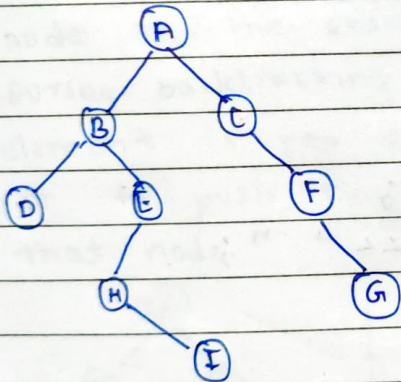


o In Order : BEDFAGCH

o Pre Order : ABDEFCGH

o post Order : EFDBGHCA

(2)



o In-Order : D B H I E A C F G.

o Pre-Order : A B D E H I C F G

o Post-Order : D I H E B G F C A

Depth first search.

Recursive pre-order traversal.

\rightarrow void preorder (node * root)

{

if (root != NULL)

{

cout << root \rightarrow data << " " ;

preorder (root \rightarrow left);

preorder (root \rightarrow right);

}

}

Recursive in-order traversal

→ void inorder (node *root)

{

if (root != NULL)

{

inorder (root → left);

cout << root → data << " ";

inorder (root → right);

}

}

Recursive post-order traversal,

→ void postorder (node *root)

{

if (root != NULL)

{

postorder (root → left);

postorder (root → right);

cout << root → data << " ";

}

}

Non-recursive pre-order traversal,

→ void preorder_nonrec (node *T)

{

stack s;

while (T != NULL)

{

cout << T → data << " ";

s.push(T);

T = T → left;

}


```
while (!s.empty())
```

```
{
```

```
    T = s.pop();
```

```
    T = T → right;
```

```
    while (T != NULL)
```

```
    {
```

```
        cout << T → data << " ";
```

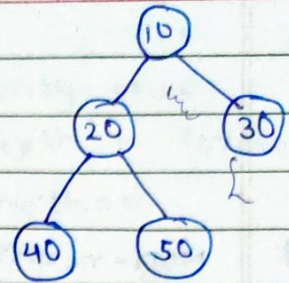
```
        s.push(T);
```

```
        T = T → left;
```

```
    }
```

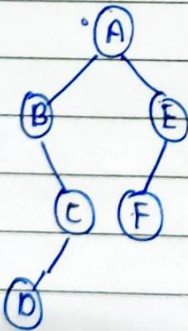
```
}
```

```
}
```



→ For postorder non recursive traversal:-

- While going to the left subtree push all the left side node on the stack with the flag value 0.
- During backtracking traverse to the right sub-tree, an element is pop and if the flag value is 0, then it is push flag value with 1. and print data of that node.



				D	0
				E	0
	B	0	B	1	
	A	0	A	0	

Non-recursive Postorder traversal.

→ 1) while (T != NULL)

{

s.push(T); st.push(NULL)

T = T → ptr;

}

2) while (!s.empty())

{

T = s.pop();

3) check the flag is '0'. if (st.pop() = NULL)
then push that element again with flag 1

flag = 1

{ & s.push(T); st.push((node*)1)

T = T → right.

while (T != NULL)

{ // flag = 0

st.push(NULL);

s.push = (T)

T = T → ptr;

}

4) else if ^{when} flag = 1

p.pop() and print the data.

5) end.

#

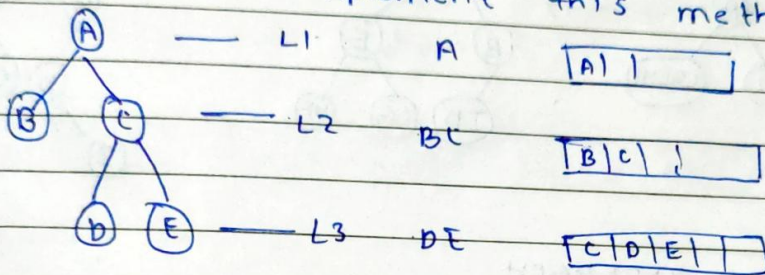
Ex:

①

②

Breadth First search (BFS)

- It is levelwise traversing the node only once.
 • In Breadth first search (BFS), no depth is formed.
 • Queue is used to implement this method.



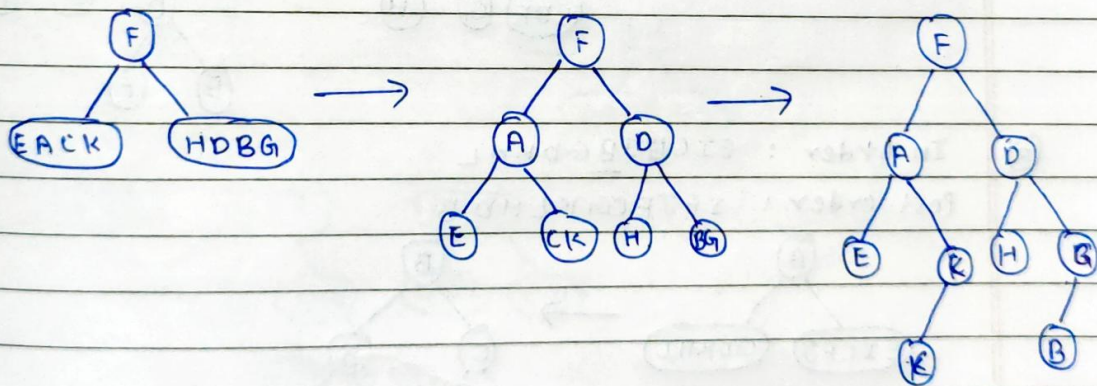
Creation of Binary Tree Inorder, Pre-order and Postorder

Ex) With Inorder and preorder.

① i) Inorder : EACKFHDBG.

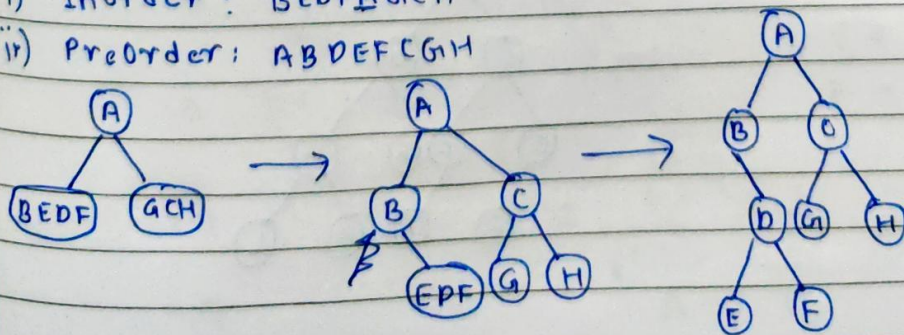
ii) Preorder : FAEKCDHGB.

↳ root

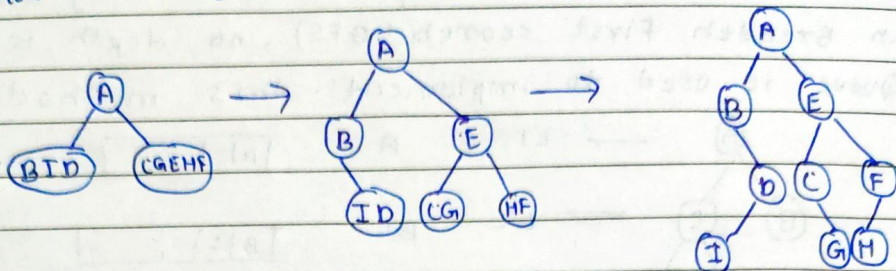


② i) Inorder : BEDFAGCH

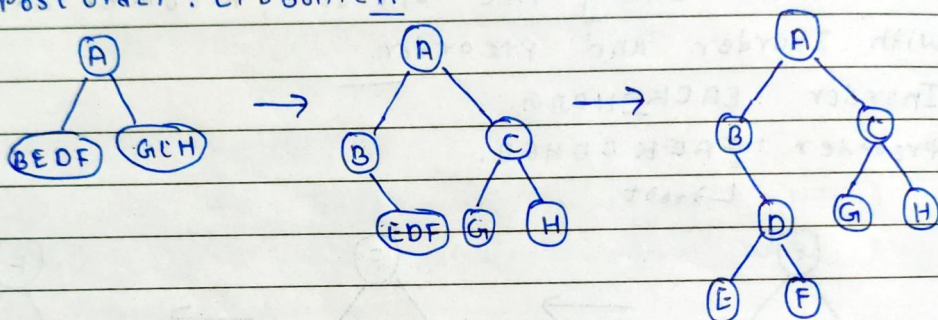
ii) Preorder : ABDEFCGH



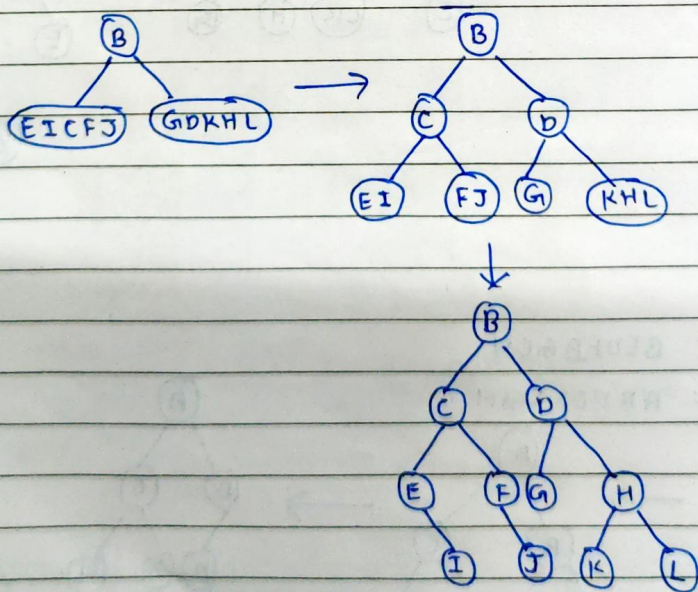
③ In order : B I B A C G E H F
PostOrder : I D B G C H F E A.



④ In Order : B E D F A G C H
Post Order : E F D B G H C A

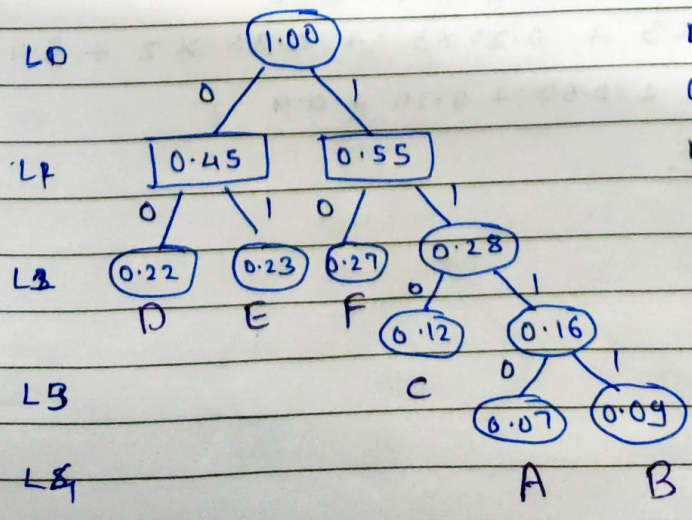
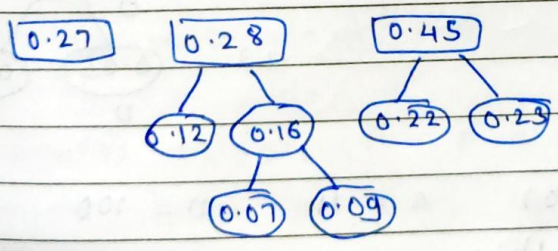
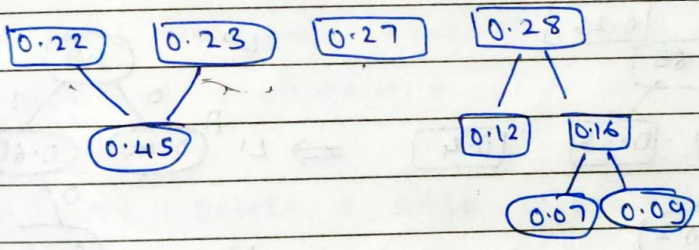
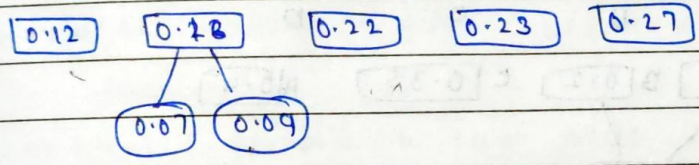
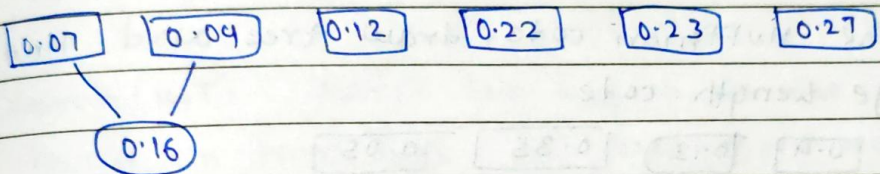


⑤ In Order : E I C F J B G D K H L
Post order : I E J F G K L H D B



Huffman Coding:-

Suppose character A, B, C, D, E, F have probabilities 0.07, 0.09, 0.12, 0.22, 0.23 and 0.27 respectively. Find an optimal Huffman code and draw Huffman tree - What is the average code length.



- A = 1110
- B = 1111
- C = 110
- D = 00
- E = 01
- F = 10

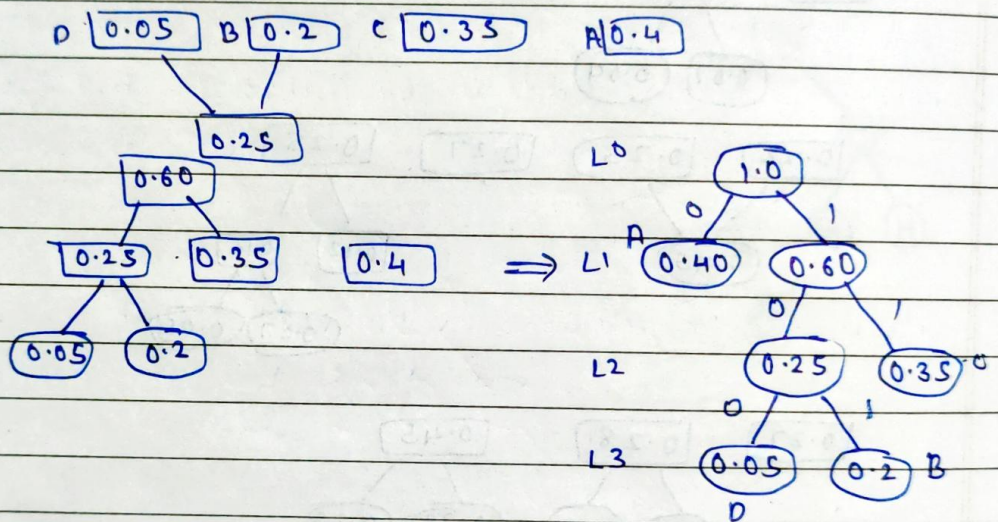
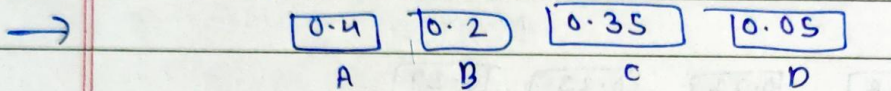
L0
L1
L2
L3
L4

$$\text{Average length code} = (0.07 \times 4) + (0.09 \times 4) + (0.12 \times 3) + (0.22 \times 2) + (0.23 \times 2) + (0.27 \times 2)$$

$$= 0.28 + 0.36 + 0.36 + 0.44 + 0.46 + 0.54$$

$$= \underline{\underline{2.44}}$$

Q.] Find the Huffman code, draw tree and find the average length code.



$$A \equiv 0 \quad B \equiv 101 \quad C \equiv 11 \quad D \equiv 100$$

Average code length :- ACL

$$= 0.05 \times 3 + 0.20 \times 3 + 0.35 \times 2 + 0.4 \times 1$$

$$= 0.15 + 0.60 + 0.70 + 0.4$$

$$= \underline{\underline{1.85}}$$

Binary Search Tree:-

	Array	LinkedList	BST
Search	$O(n)$	$O(n)$	$O(\log n)$
Insert	$O(1)$	$O(1)$	$O(\log n)$
Delete	$O(n)$	$O(n)$	$O(\log n)$

Search(k, T): Search for key k in the tree T . If k is found in some node of tree then return true otherwise return false

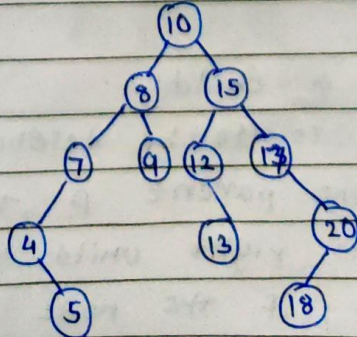
Insert(k, T): Insert a new node with value k in the info field in the tree T such that the property of BST is maintained.

Delete(k, T): Delete a node with value k in the info field from the tree T such that the property of BST is maintained.

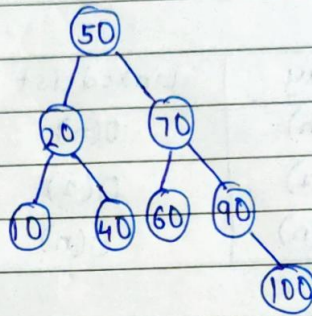
Find Min(T), Find Max(T): Find minimum and maximum element from the given non-empty BST.

Construct BST for following number.

10 8 15 12 13 7 9 17 20 18 4 5



② 50 70 60 20 40 10 40 100.

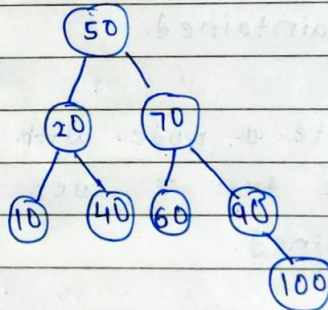


DELETION:- (in BST)

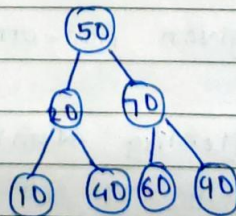
(i) When node is a leaf node

→ IF the node to be deleted is leaf node, it can be deleted directly by setting current parent pointer pointing to NULL.

For Eg:-



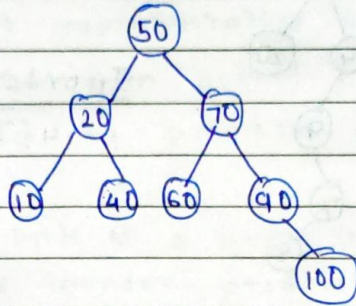
→ Delete 100.



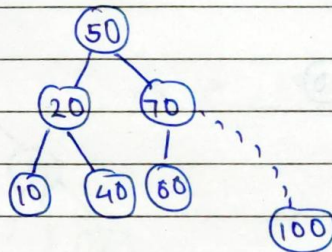
(2) When node has a child:-

→ If a node q is to be deleted and it is right child of the parent p , the only child of q will become the right child of p after deletion of q . Similarly, if the node q is to be deleted and it is the left child of parent node p , then only child of q will become the left child of p .

after deletion of 4.



→ delete 90:-



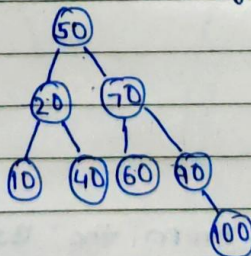
(3) A node have 2 (two) child:-

→ The general strategy is to replace the data of this node with smallest data of right side or inorder function.

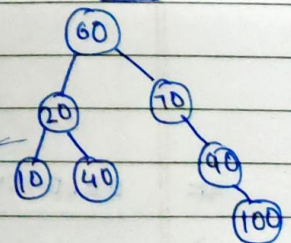
The smallest node in right-sub tree will either be a leaf node or node of degree one.

Then, delete that node using 1

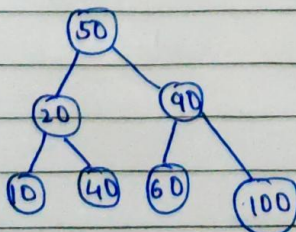
→ Example :-



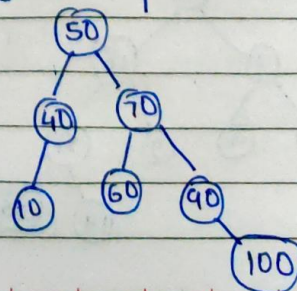
Delete 50



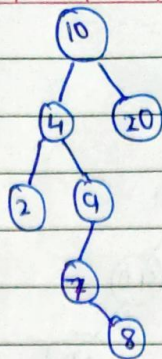
Delete 70:



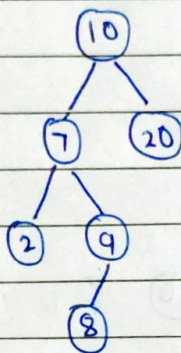
Delete 20



→

In order:-

2, 4, 7, 8, 9, 20

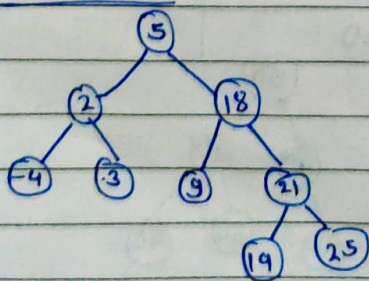
Delete 4:-# SEARCHING THROUGH the BST.

1. Compare the target value with the element in the root node.

-
- IF the target value is equal, the search is successful.
 - IF the target value is less, search the left subtree.
 - IF the target value is greater, search the right subtree.

DELETING a node from the BST.

→

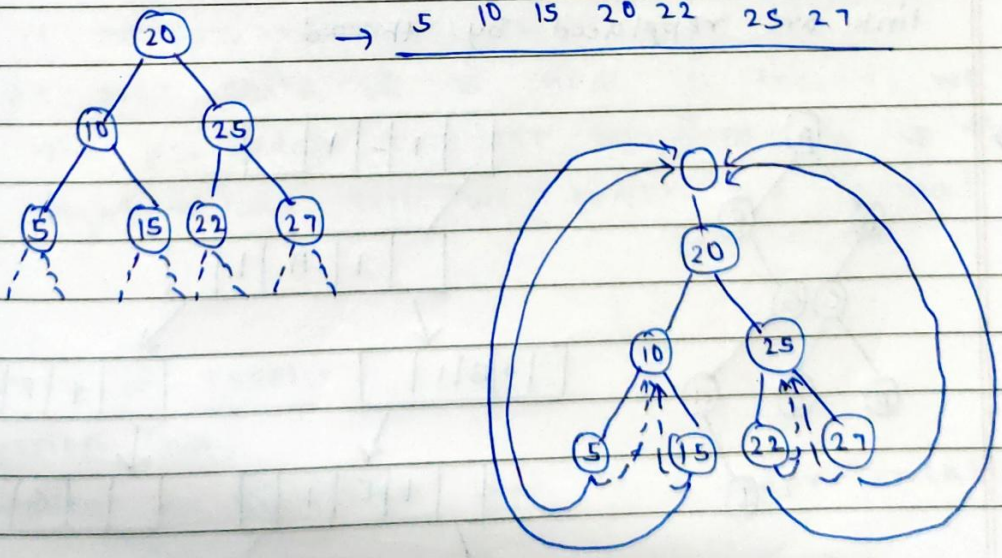
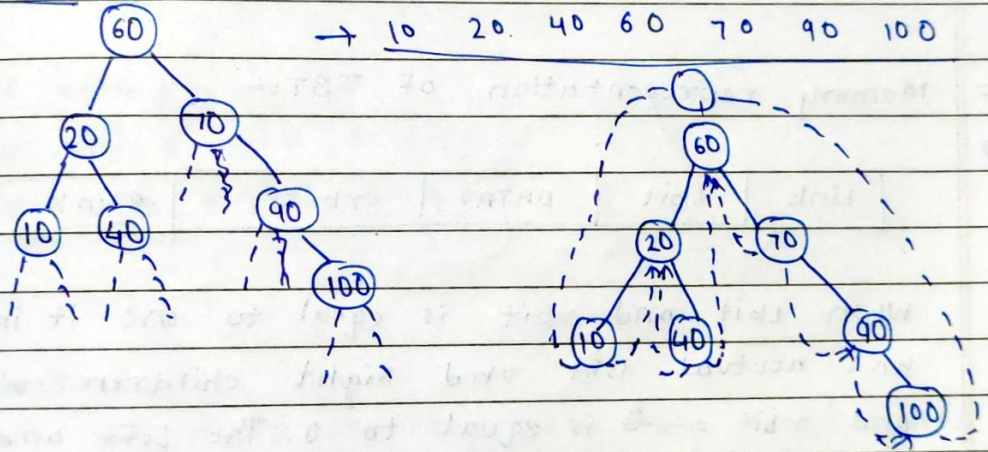
Delete 18:-

Threaded Binary Tree: - (TBT)

In a linked list representation of a binary tree, there are more null links than actual pointers. These null links can be replaced by pointers called threads to other nodes.

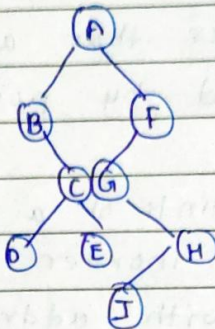
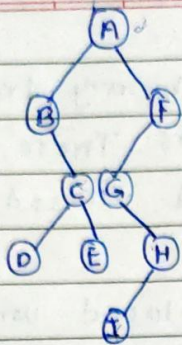
A left null link of a node is replaced with the address of its inorder predecessor and right null link is replaced with address of its inorder successor.

Example:-



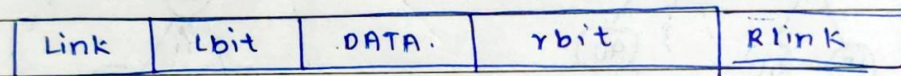
LRootR

InOrder: BDCEAGIHF



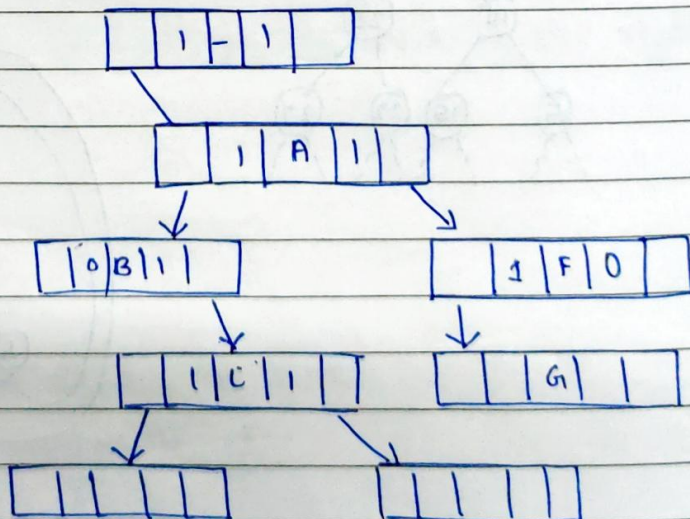
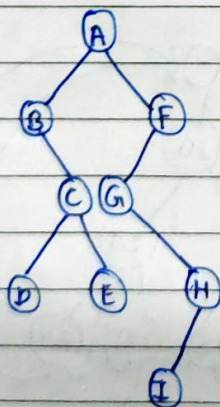
Memory representation of TBT:-

→



When lbit and rbit is equal to one it indicates it has actual left and right children when lbit and rbit is equal to 0. The left and right link are replaced by thread.

→



Advantage: The time required is reduced due to backtracking using threads.

- No need to use stack. (No requirement)
- We can directly add extra datastructure.

Disadvantage :- ① Complex structure. (class TBT node)
② Insertion and deletion is difficult.

```
class TBT_node
{
    int data;
    TBT_node * lptr, * rptr;
    TBT_node * lbit, * rbit;
}
```

TBT Traversals:-

→ If the left child of the node is actual child then left is the preorder successor.

If the left child of a node is thread, we can locate the pre-order successor by climbing up the tree using right thread till we reach the normal right child.

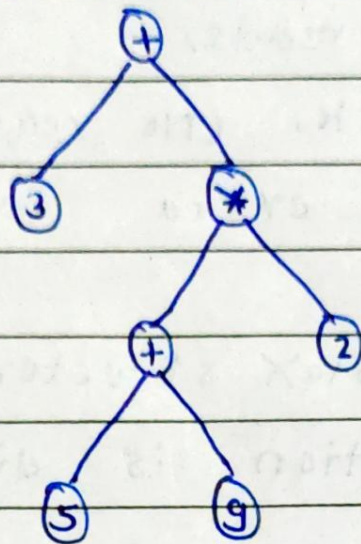
Application of TREES:-

(i.) Expression Tree

Definition: An Expression tree is a representation

The traversals in the expression tree will prefix, infix, postfix expressions.

• Expression tree for $3 + ((5 + 9) * 2)$ would be :-



(2) Decision Tree :-

→

