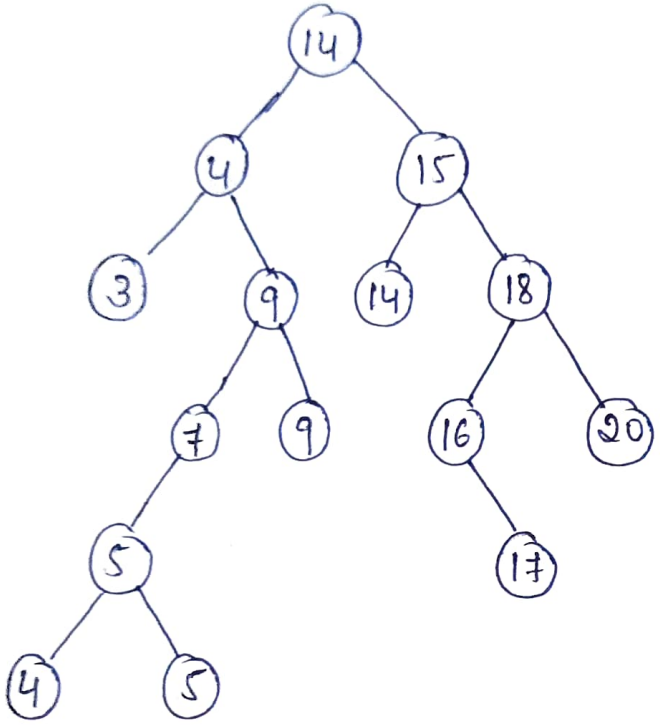
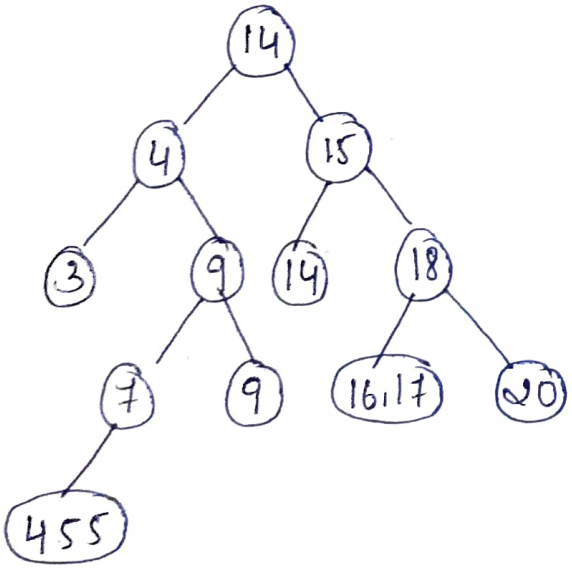
① Draw the Binary Search tree (BST) for the following data and traverse the tree using Inorder, preorder, postorder techniques 14, 15, 4, 9, 7, 18, 3, 5, 16, 4, 20, 17, 9, 14, 5.

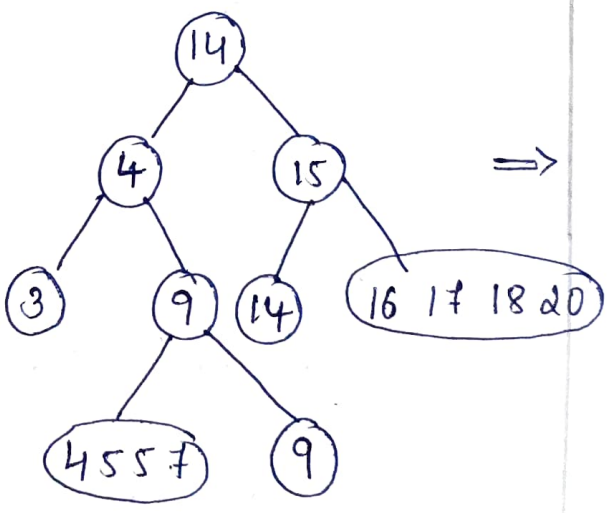
⟹



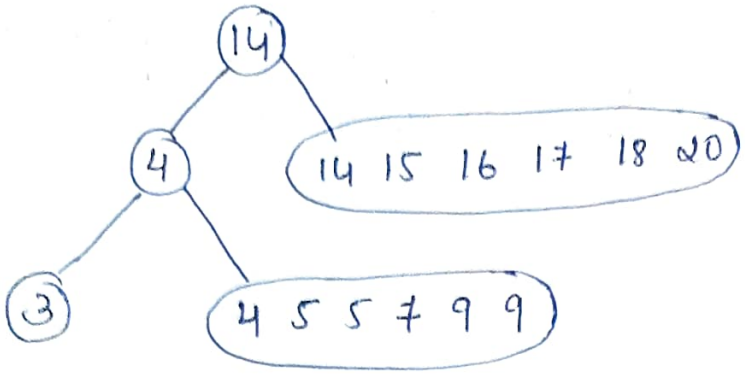
Inorder = LNR

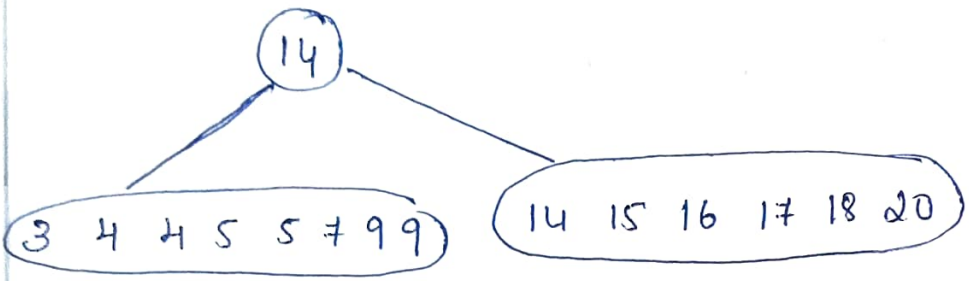


⟹



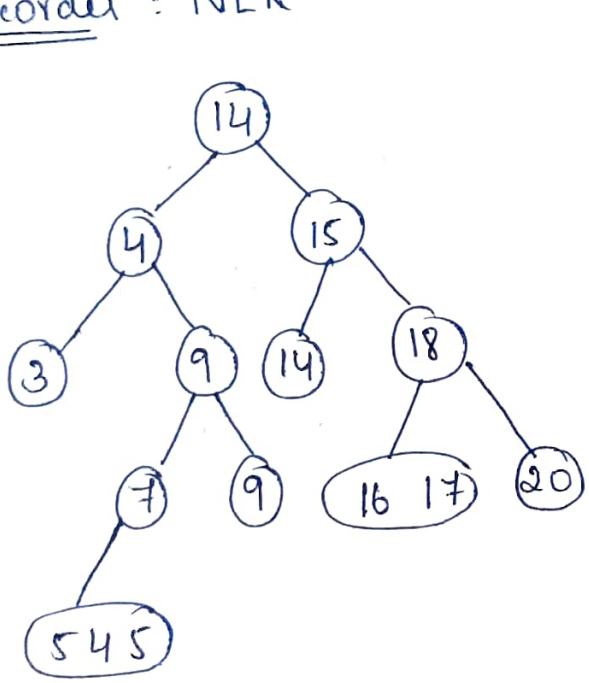
⟹



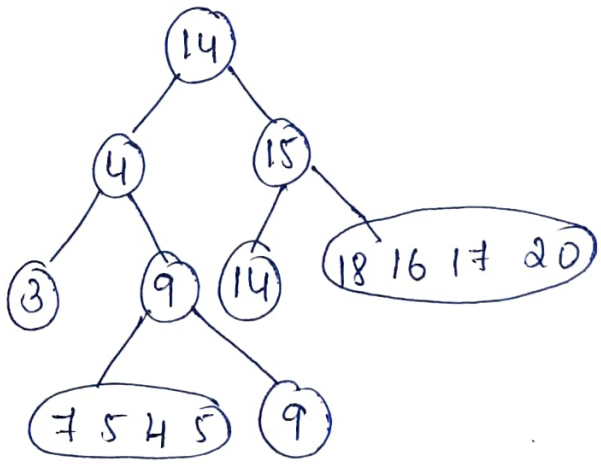


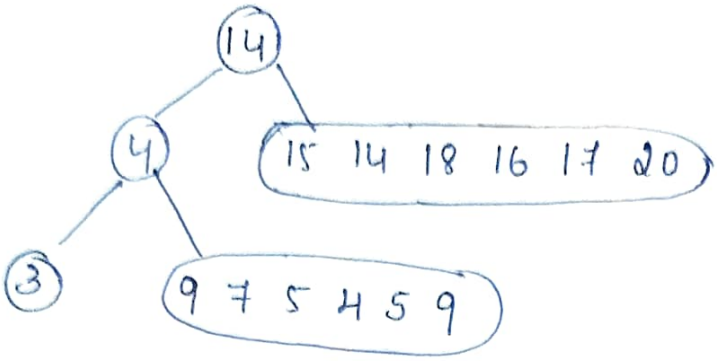
⟹ The Inorder traversal is
3 4 4 5 5 7 9 9 14 14 15 16 17 18 20

Preorder : NLR

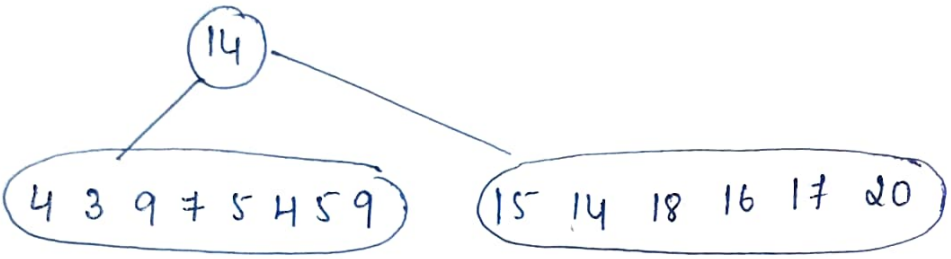


⟹



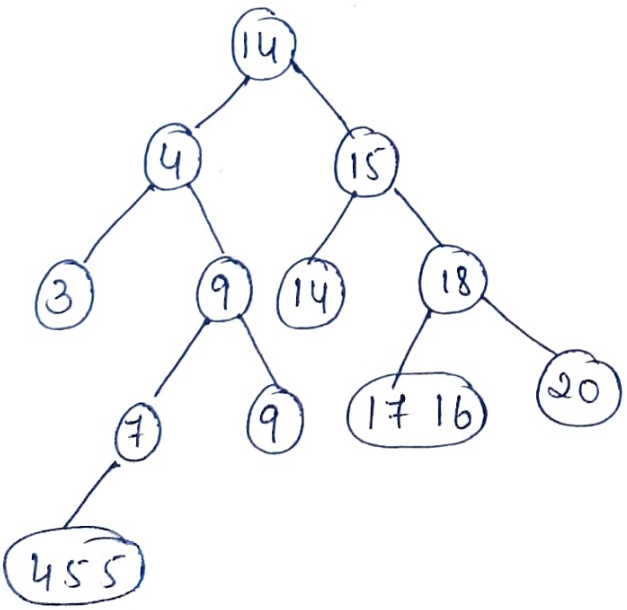


⇓

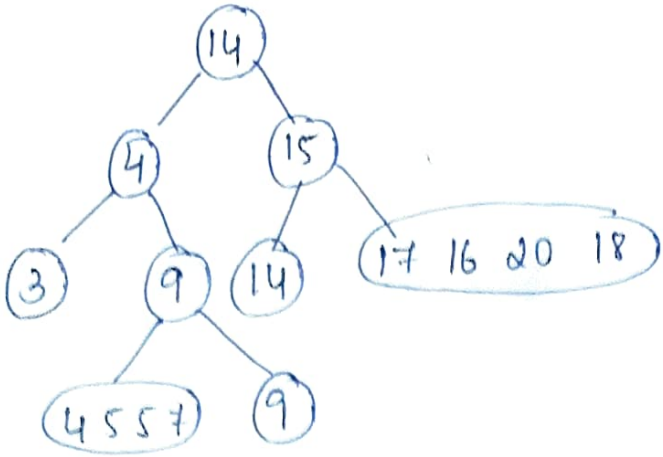


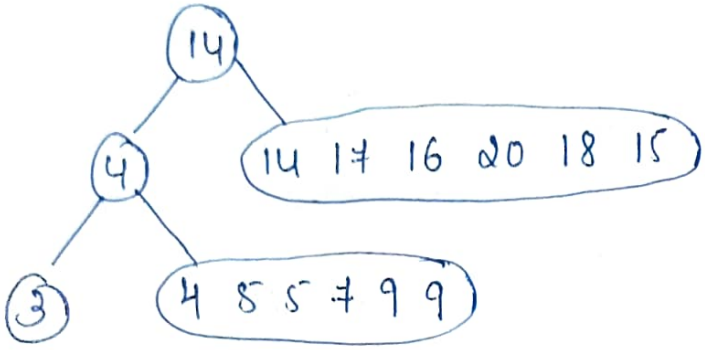
⟹ The preorder traversal is..
14 4 3 9 7 5 4 5 9 15 14 18 16 17 20

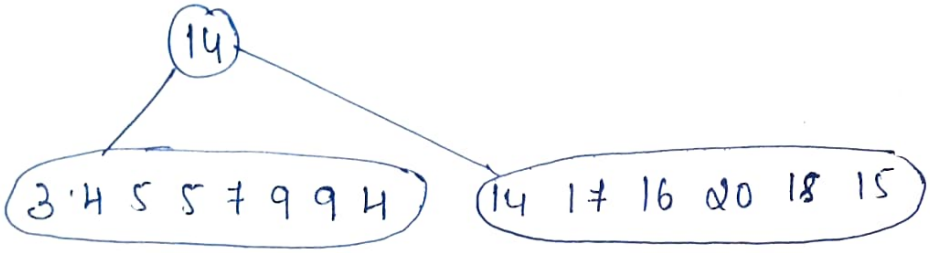
Post Order : LRN



⟹







⇒ The postorder traversal is

3 4 5 5 7 9 9 4 14 17 16 20 18 15 14

Conversion of preorder and Inorder into Binary tree

WKT, the preorder traversal is NLR and the Inorder traversal is LNR.

To find root node we have to go for preorder. In given preorder, the first element will be the root node And we have to scan the traversal from left to right.

Next to find the left and right child of the root node, we have to go for Inorder. First we have to locate the parent node in the Inorder and see for left and right members of the tree.

* If there are more than one members in left and right to the parent node in Inorder then to decide which is the parent node out of that we have to scan preorder traversal that is from left to right. The element which comes first while scanning it will becomes the parent node.
* If there is only one member present either left to right of the parent in the Inorder then itself will be the node to be written.
* These steps repeat until all the elements given in traversals are written in binary tree

Conversion of Postorder and Inorder into Binary tree

WkT, the postorder traversal is LRN and the inorder traversal is LNR.

To find root node, we have to go for postorder. In given Postorder, the last element will be the root node. And we have to scan the traversal from right to left.

Next to find the left and right child of the root node, we have to go for inorder. First we have to locate the parent node in the Inorder and see for left and right members of the tree.

* If there are more than one members in left and right to the parent node in Inorder then to decide which is the parent node Out of that we have to scan postorder traversal that is from right to left. The element which comes first while scanning it will becomes the parent node.
* If there is only one member present either left or right of the parent in the Inorder then itself will be the node to be written.
* These steps repeat until all the ~~nodes are completed~~.

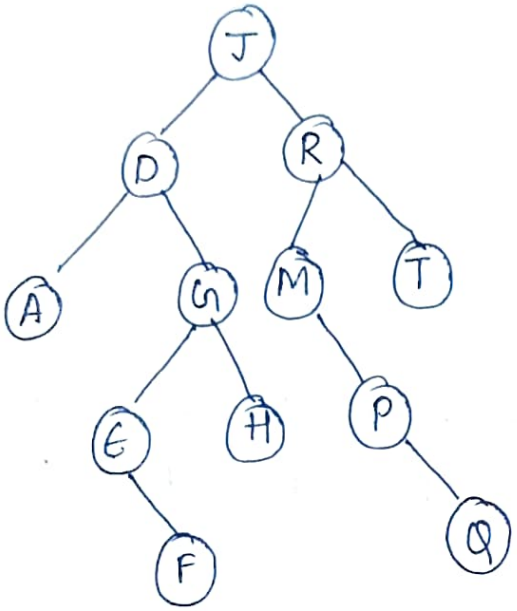
All the elements given in traversals are written in the binary tree.

(2) Construct the Binary Search tree using given Inorder and Preorder Sequence and Inorder and postorder Sequence

i) Inorder : A, D, E, F, G, H, J, M, P, Q, R, T (LNR)

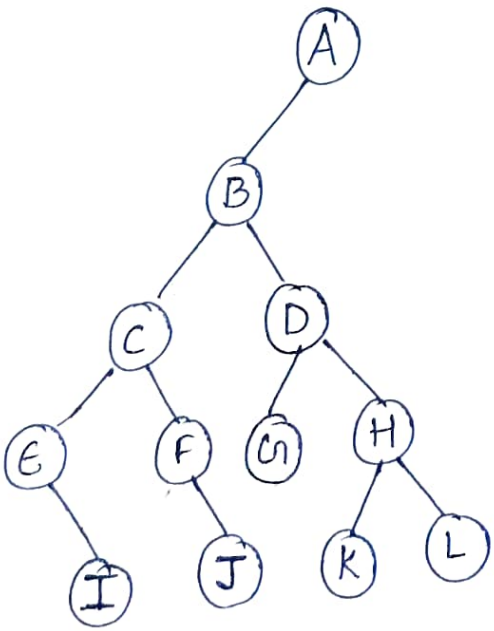
Preorder : J, D, A, G, E, F, H, R, M, P, Q, T (NLR)

$\Rightarrow$

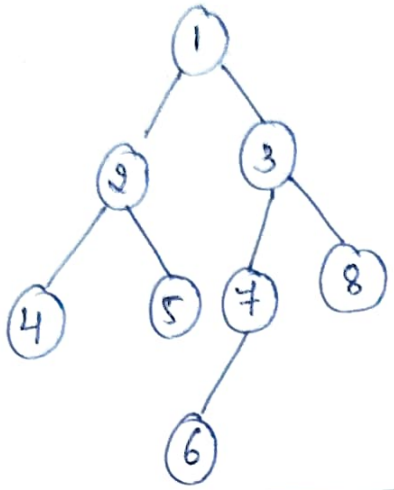


ii) Preorder: A, B, C, E, I, F, J, D, G, H, K, L (NLR)

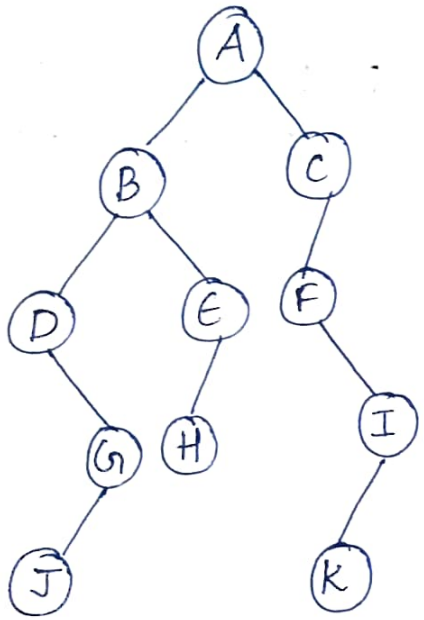
Inorder: E, I, C, F, J, B, G, D, K, H, L, A (LNR)



iii) Inorder: 4, 2, 5, 1, 6, 7, 3, 8 (LNR)
Postorder: 4, 5, 2, 6, 7, 8, 3, 1 (LRN)



iv) Inorder: D, J, G, B, H, E, A, F, K, I, C (LNR)
Postorder: J, G, D, H, E, B, K, I, F, C, A (LRN)



(3) Define Binary Search tree. Draw the BST for the following input:

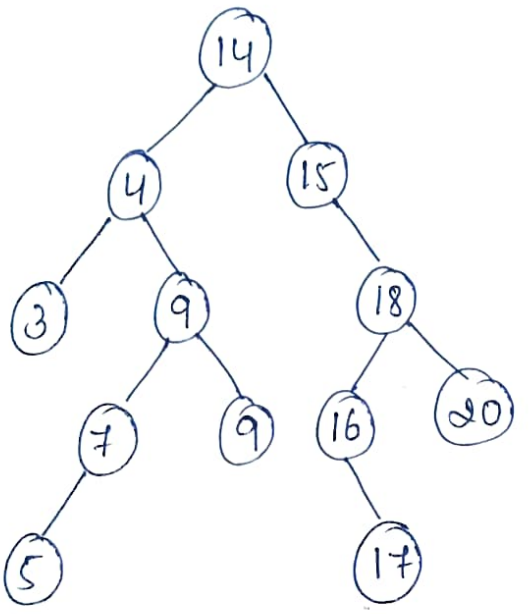
14 15 4 9 7 18 3 5 16 20 17 9

Give recursive search function to search an element in that tree.

⇒ **Binary search tree :**

A Binary search tree is a binary tree in which for each node say X in the tree, elements in the left subtree are less than Info(x) and elements in the right subtree are greater than Info(x). Every node in the tree should satisfy this condition if left subtree or right subtree exists.

14, 15, 4, 9, 7, 18, 3, 5, 16, 20, 17, 9

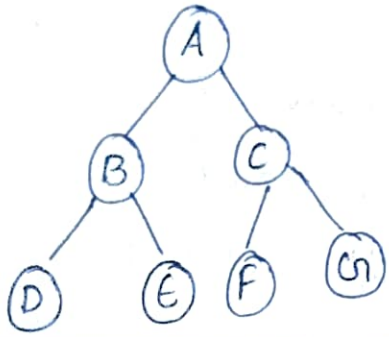


(4) Define Binary tree with an example. Write C recursive routine to traverse the given tree using inorder, preorder and postorder.

⇒ **Binary tree :** A binary tree is a tree which has finite set of nodes that is either empty or consist of a root and two subtrees called left subtree and right subtree.

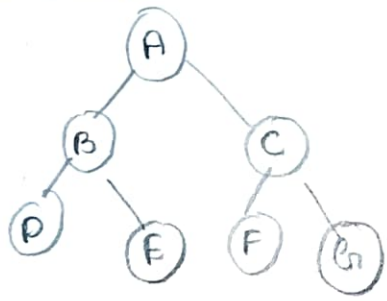
A binary tree can be partitioned into three subgroups namely, root, left subtree and right subtree.

Ex:-



Recursive function to traverse the tree in preOrder:

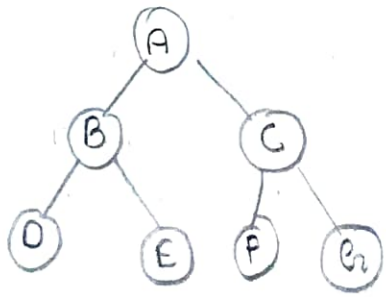
```
Void PreOrder (NODE root)
{
    if (root == NULL)
     return;
    Printf ("%d", root -> Info);
    PreOrder (root -> llink);
    PreOrder (root -> rlink);
}
```



A B D E C F G

Recursive function to traverse the tree in Order

```
Void InOrder (NODE root)
{
    if (root == NULL)
     return;
    InOrder (root -> llink);
    Printf ("%d", root -> Info);
```

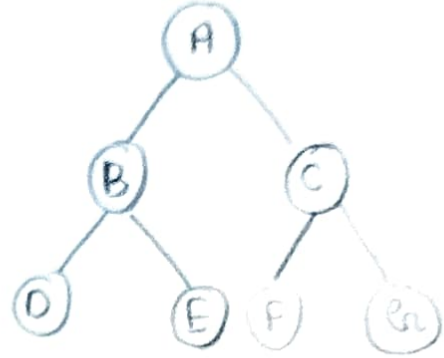


D B E A F C G

```
Inorder (root -> rlink);
}
```

Recursive function to traverse the tree in postorder.

```
Void postorder (NODE root)
{
    if (root == NULL)
    return;
    Postorder (root -> llink);
    Postorder (root -> rlink);
    Printf ("%d", root -> info);
}
```

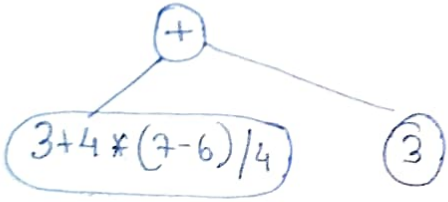


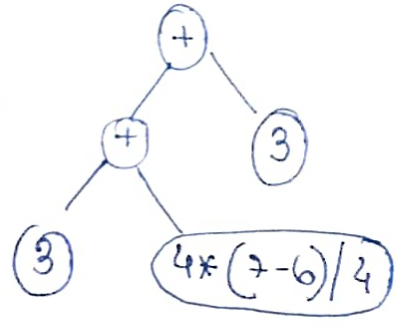
DEBFGCA

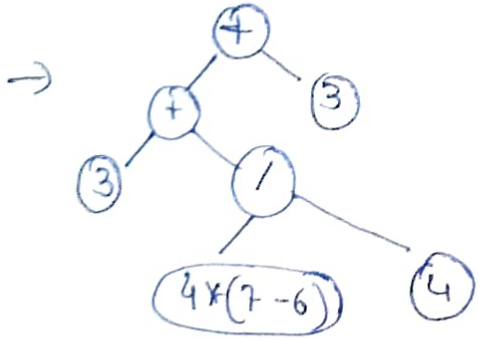
---

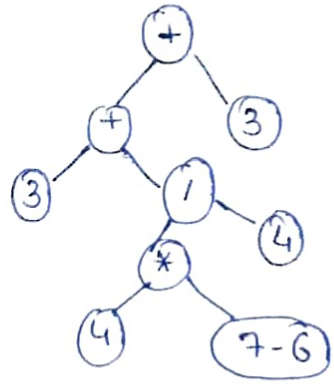
(5) i) Draw the binary tree for the following expression $3 + 4 * (7 - 6) / 4 + 3$. Traverse the generated tree using Inorder, postorder and preorder.

3+4 * (7-6)/4 +3

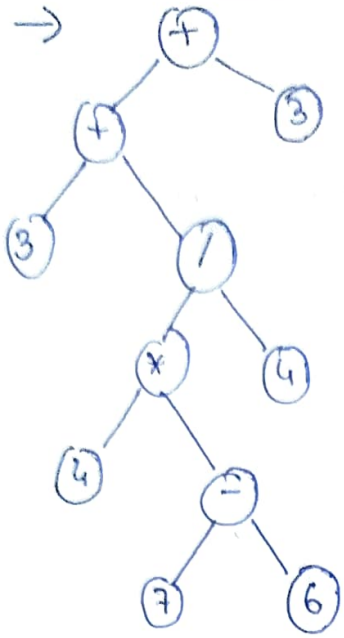




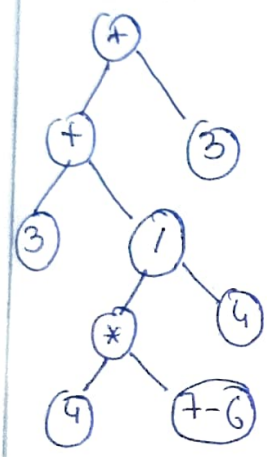




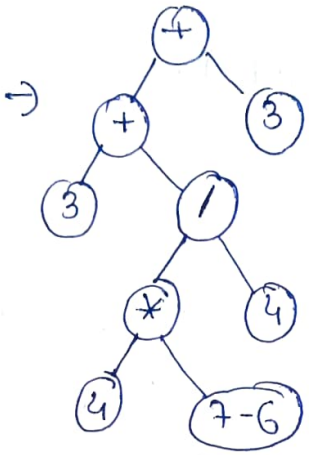
→



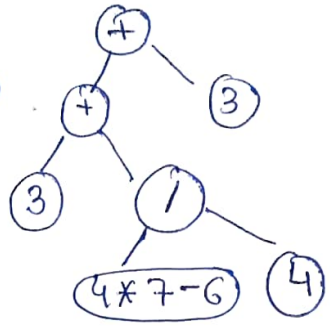
Inorder :- LNR



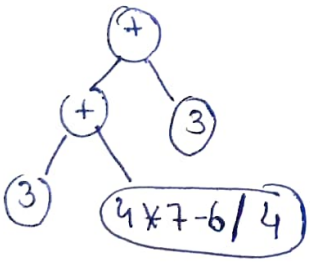
→



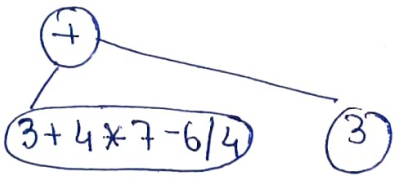
→



→

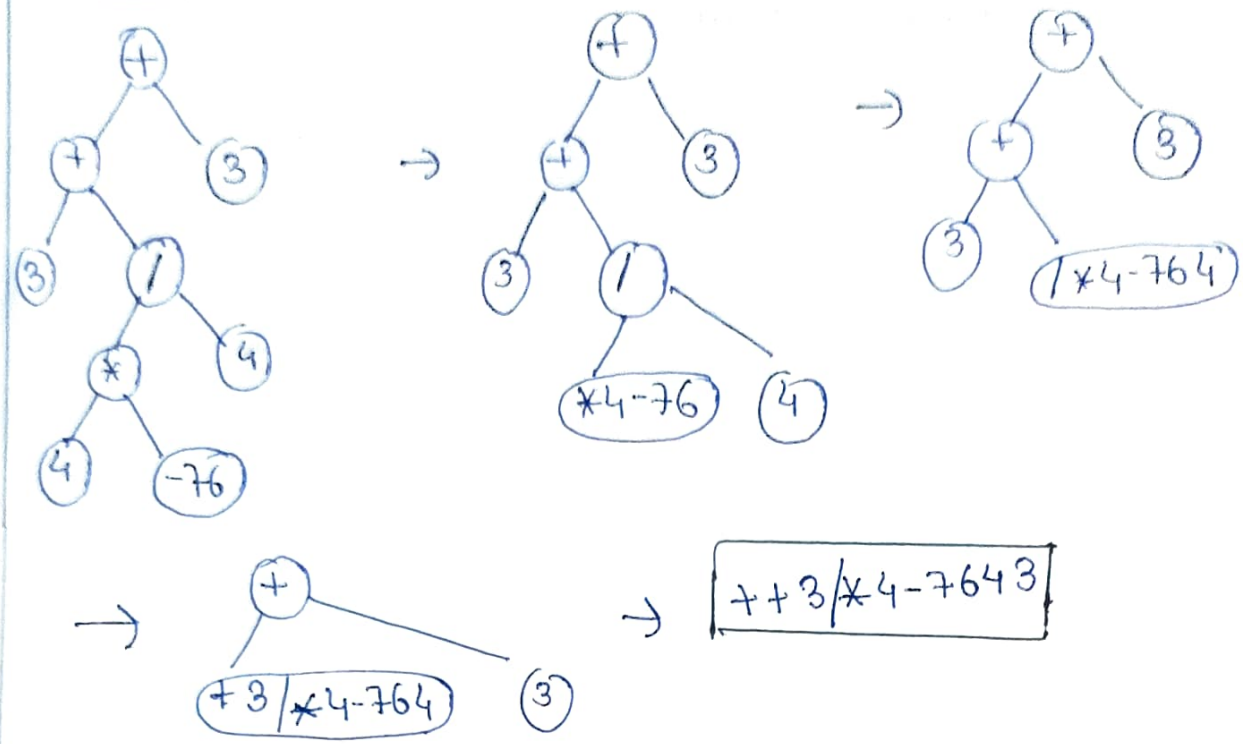


→

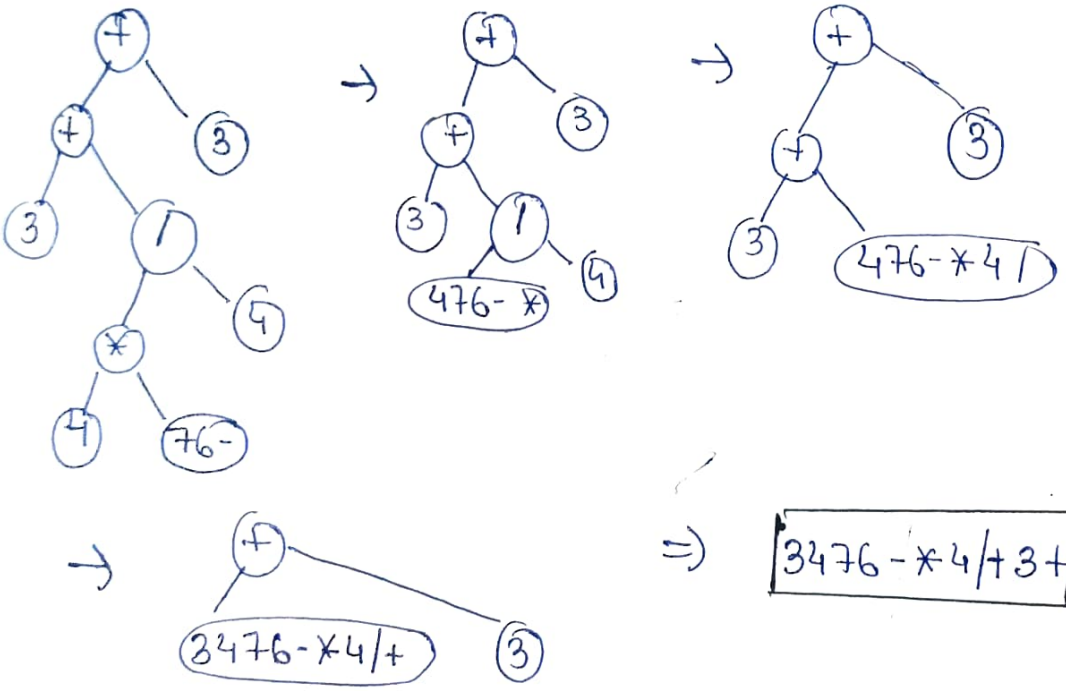


= 3+4*7-6/4+3

Preorder (NLR)

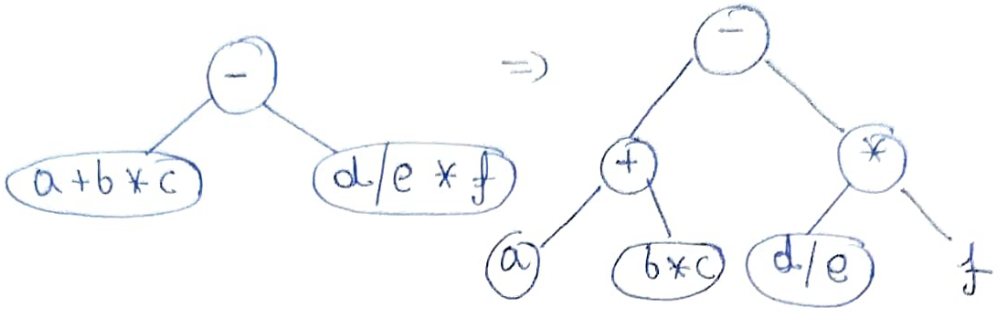


Postorder (LRN)

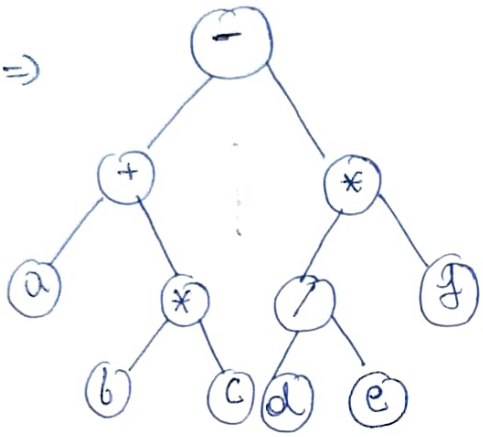


ii) Create an Expression tree for a given expression

a + b * c - d / e * f



=>



★ construction of expression tree using an example

((6 + (3-2) * 5) ^ 2 + 3)

Mainly we have to consider precedence & associativity

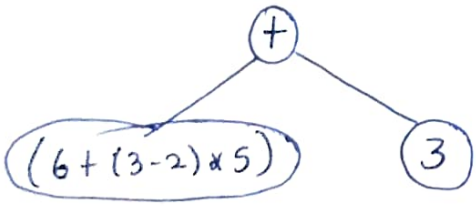
- ^, $ have highest precedence & right to left associativity
- *, / have precedence next to ^ and $ and have associativity from left to right
- +, - have lowest precedence & have associativity from left to right.

step 1 : firstly consider the operator with lowest precedence and consider the associativity. in the above example. we have + and - operator with lowest precedence & its associativity is from left to right.

Hence + operator is the root node.

Hence in the first step we have : + as the root node

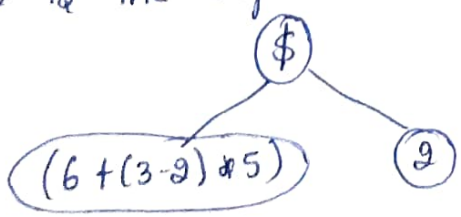
(6 + (3-2) * 5) ^ 2 as left node & 3 as right node



step 2 : again in left node we can split them as $ root node and

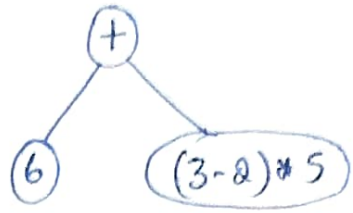
1) (6 + (3-2) * 5) as left node

2) 2 is the right node



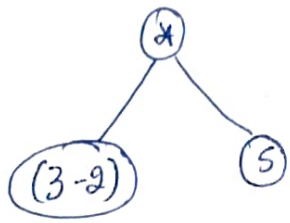
Step 3: Again in left node we have (6 + (3-2) * 5) we have to expand the bracket. Again we have to go with lowest precedence operator, & consider it as root node.

1) Here + is the root node
2) 6 is the left node
3) (3-2) * 5 is the right node



Step 4: in the right node we have (3-2) * 5

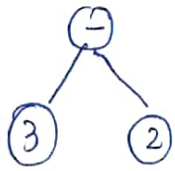
1) here we have to consider the operator outside the bracket
2) hence * is the root node.
3) (3-2) is the left node



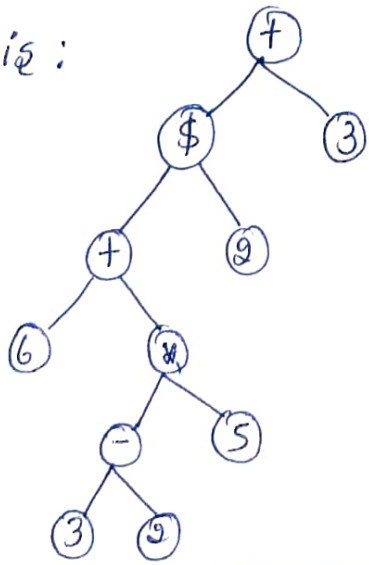
Step 5:

In the left node we have (3-2)

1) we have the operator as - which will be as root node
2) 3 is the left node
3) 2 is the right node



Final tree is:



(6) Write a recursive function to search key value in a Binary search tree. Construct a BST for the given set of values.

14, 15, 4, 9, 7, 18, 3, 5, 16, 20, 17, 9 & peform traverse on it.

=> Recursive function search function to search an element in that tree:

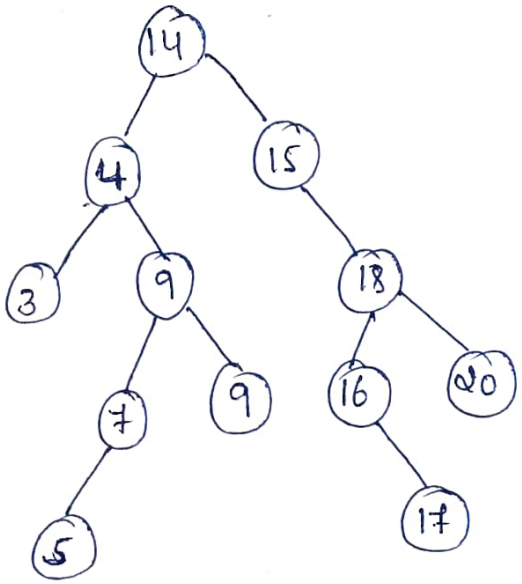
```
NODE search (int item, NODE root)
{
    if (root == NULL)
    return root;
    if (item == root->Info)
    return root;
    if (item < root->Info)
    return search (item, root->llink);
   return search (item, root->rlink);

}
```

or

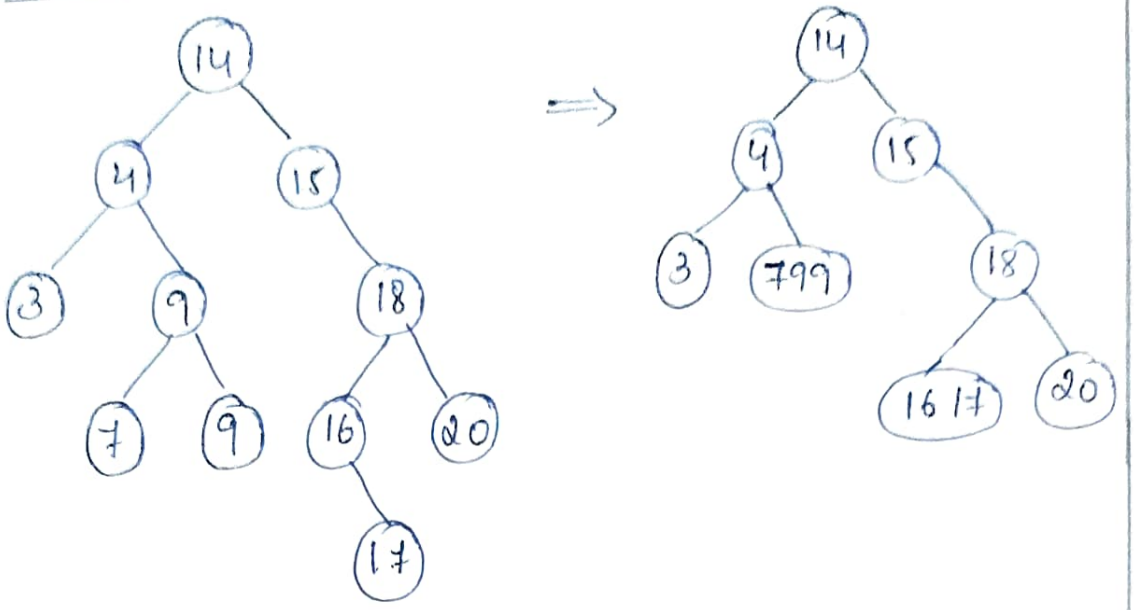
```
Void search (int item, NODE root)
{
    if (root == NULL)
    {
        Printf ("Item not found, search Unsuccessfull\n");
        return;
    }
```

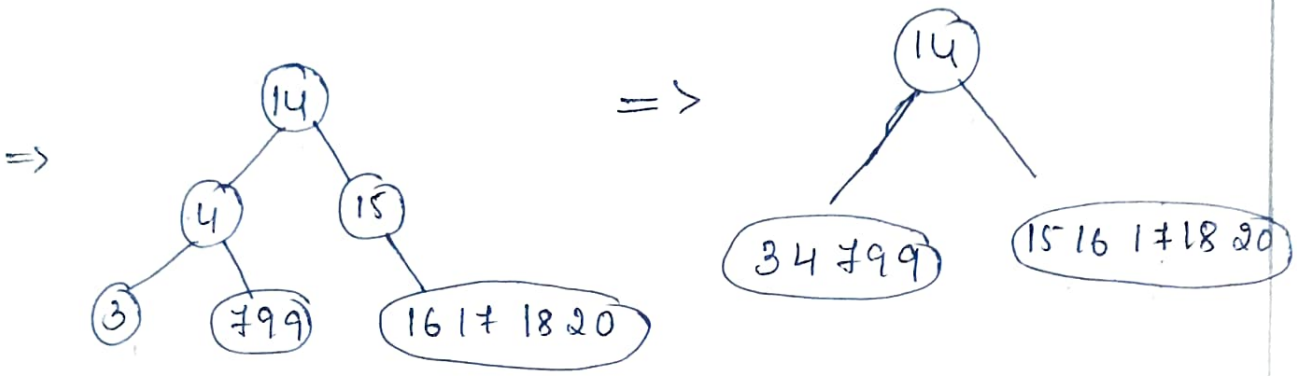
```
if (item == root->info)
{
    Printf ("Search is succesfull \n");
    return;
}
if (item < root->info)
{
    Search (item, root->llink);
}
else
    Search (item, root->rlink);
}
```

14, 15, 4, 9, 7, 18, 3, 5, 16, 20, 17, 9



Inorder traversal : LNR

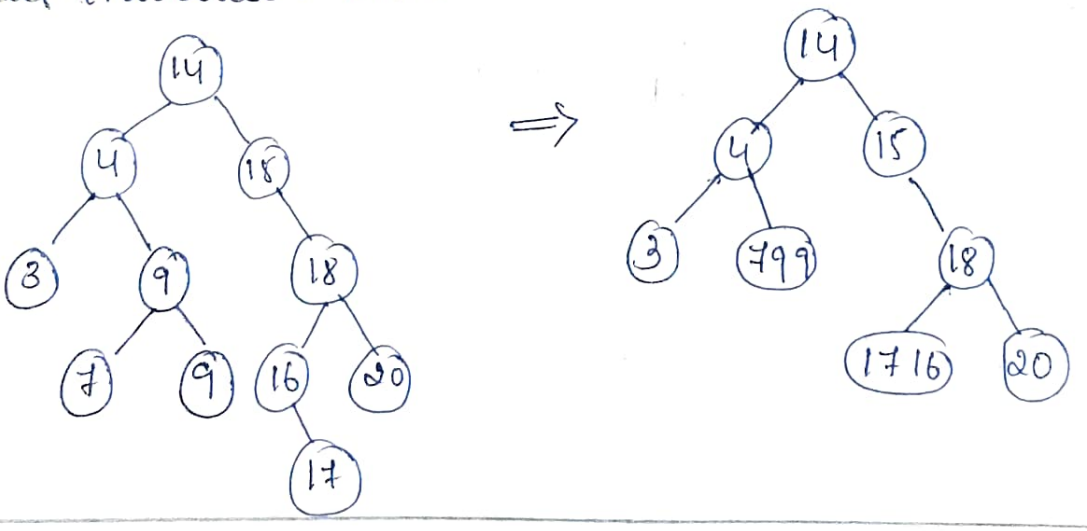




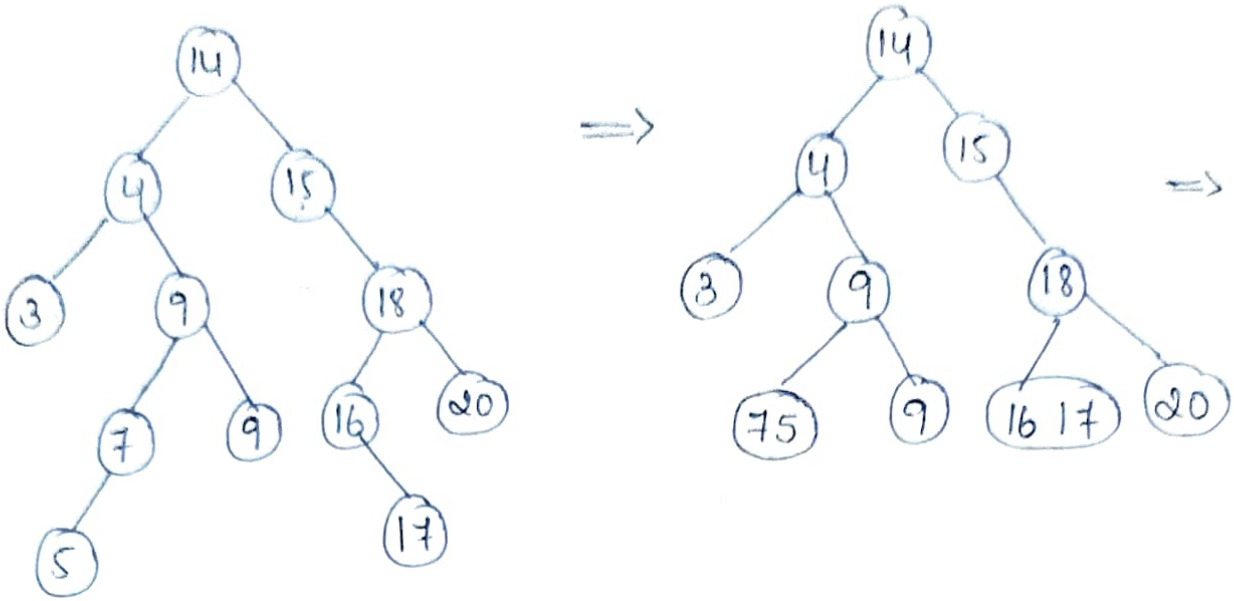
The Inorder traversal is.

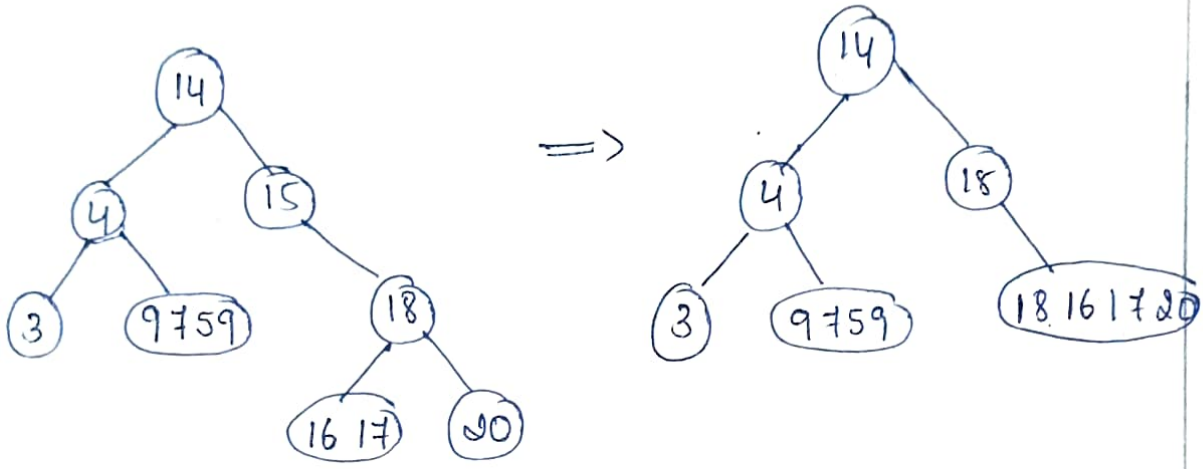
=> 3 4 7 9 9 14 15 16 17 18 20 .

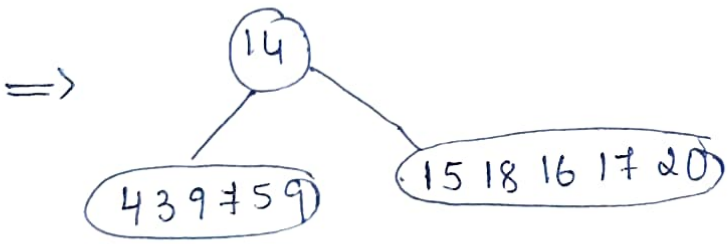
Post order traversal :- LRN



Preorder traversal : NLR

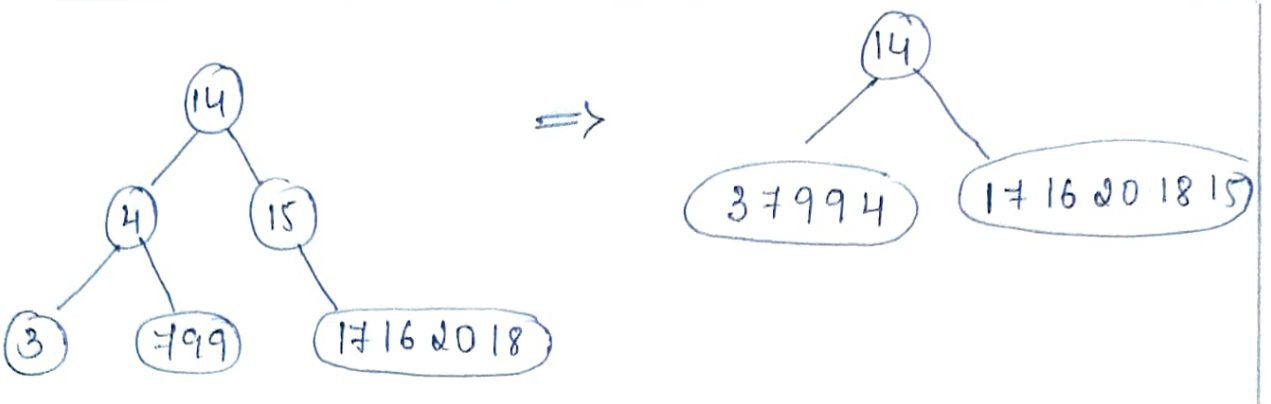






=> The preorder traversal is

14 4 3 9 7 5 9 15 18 16 17 20

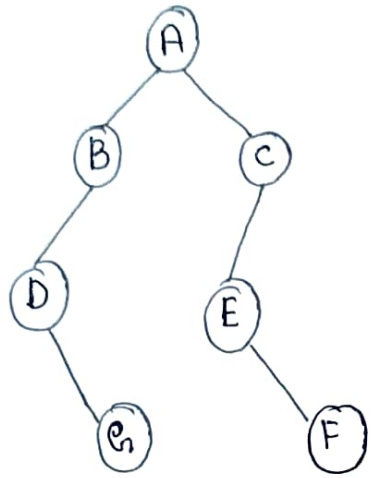


→ The post order traversal is :

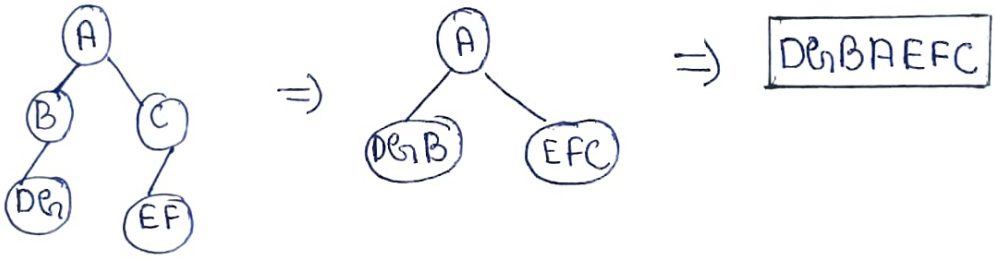
3 7 9 9 4 17 16 20 18 15 14

(7) Find the Inorder, Preorder and postorder for the following:

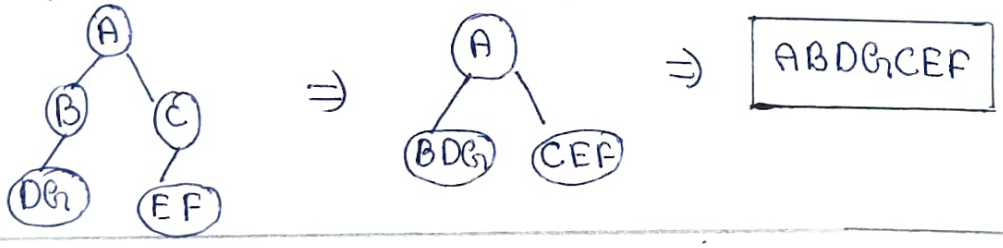
(a)



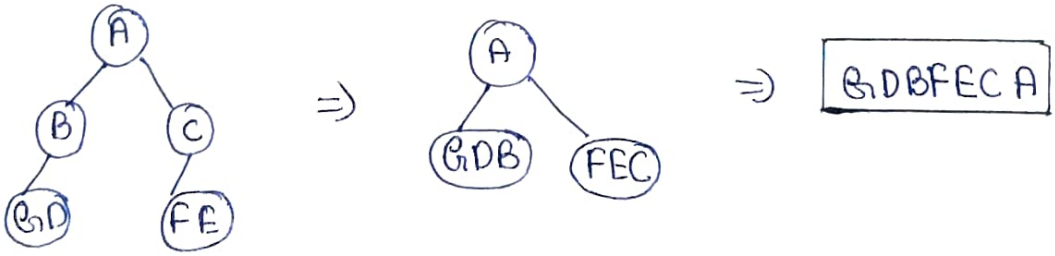
Ans:- i) Inorder traversal (LNR)



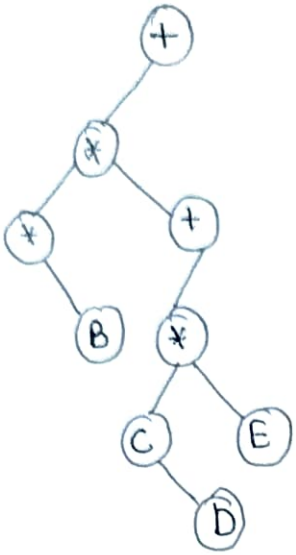
ii) Preorder traversal (NLR)



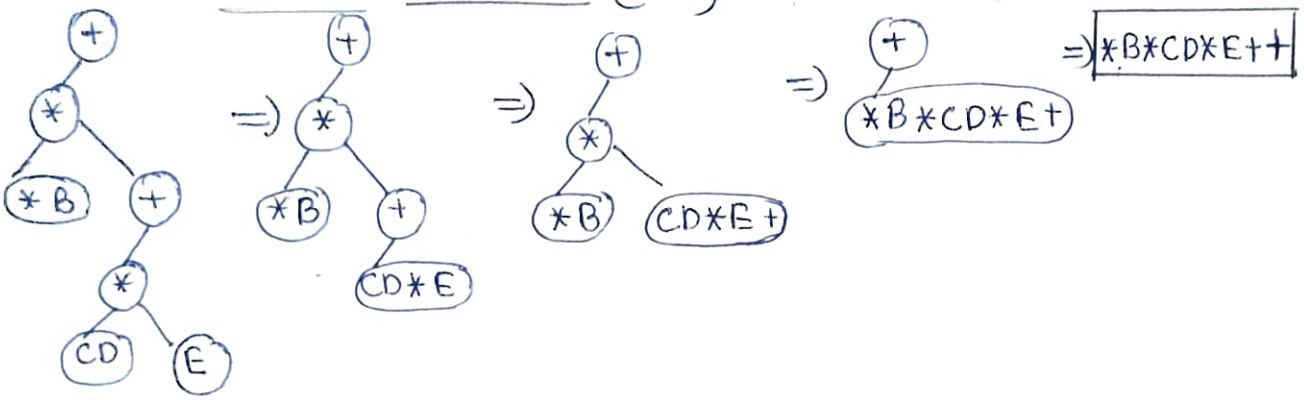
iii) Postorder traversal: (LRN)



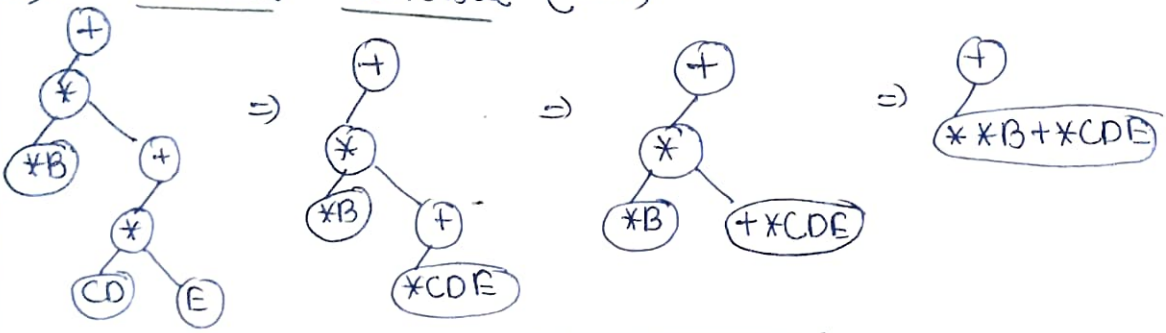
(b)



Ans: i) Inorder traversal (LNR)

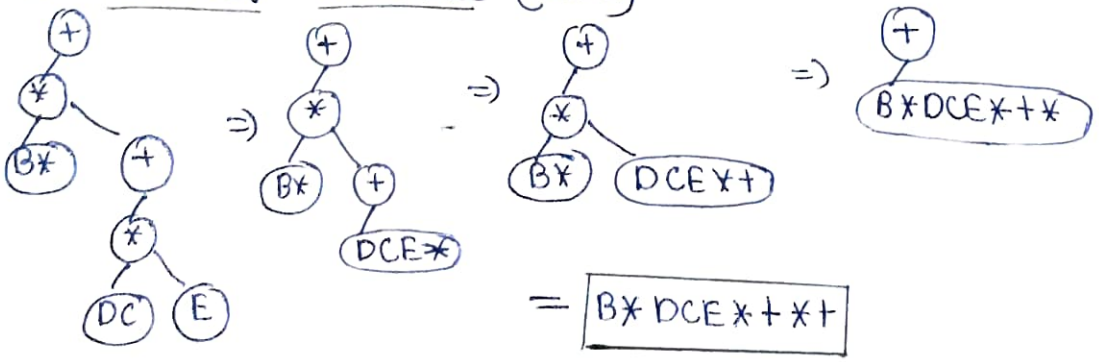


ii) Preorder traversal (NLR)



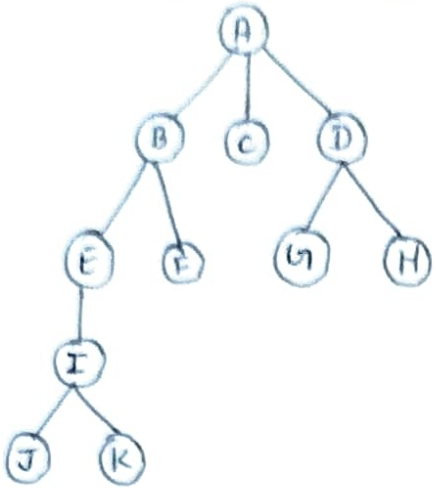
= **++*B+*CDE**

iii) Postorder traversal (LRN)

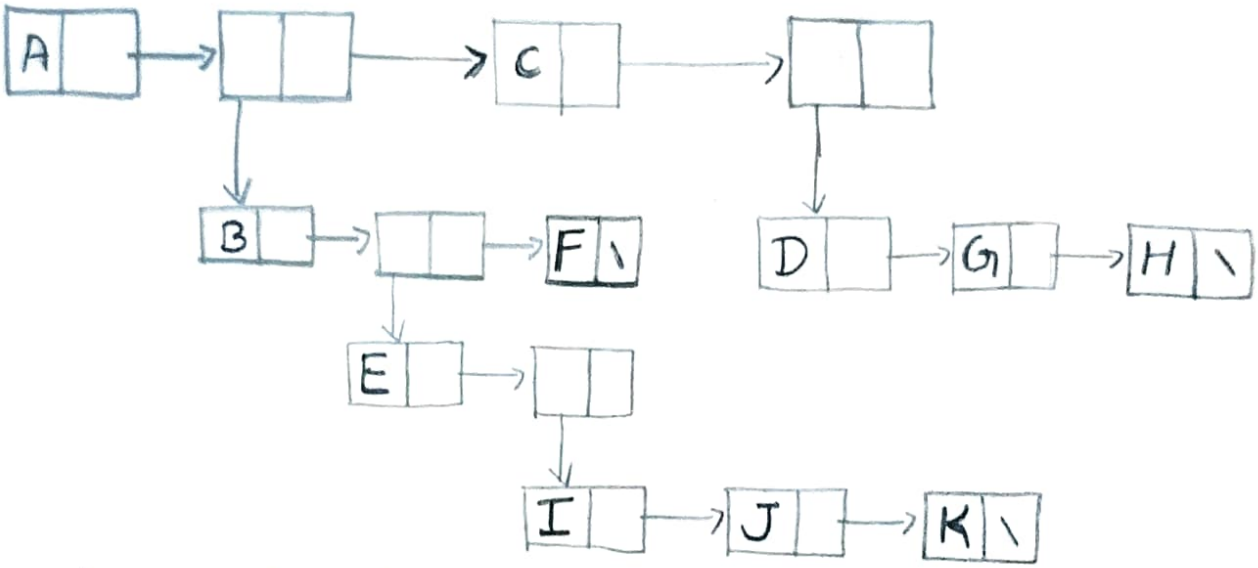


= **B*DCE*+*+**

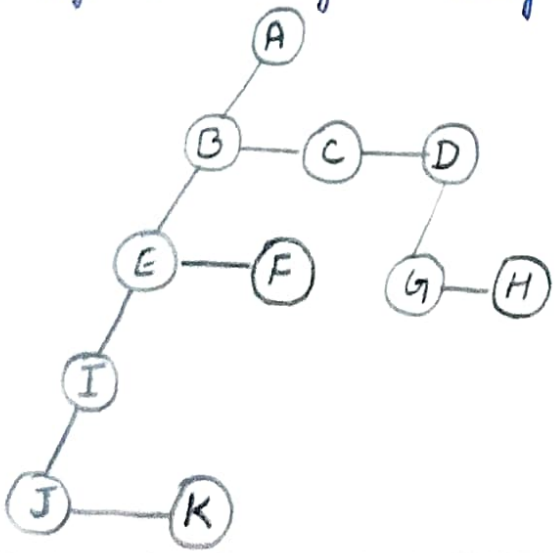
(8) (a) Represent the below given tree using
i) linked list representation
ii) Left child right sibling representation.



=> i) linked list representation.

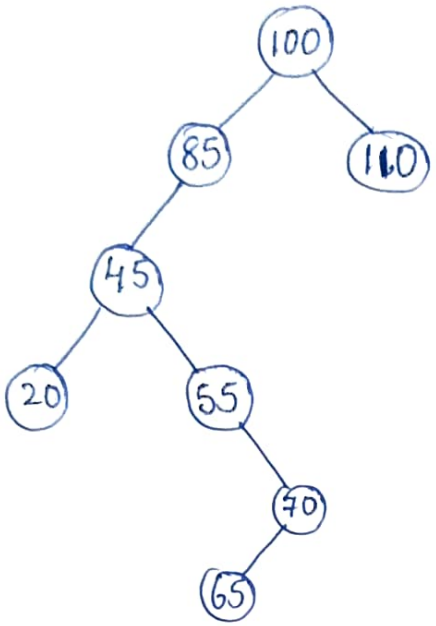


ii) left child right sibling representation.



(b) For the given data, draw a binary search tree and show the array and linked representation of the same: 100, 85, 45, 55, 110, 20, 70, 65

=> BST representation of above data

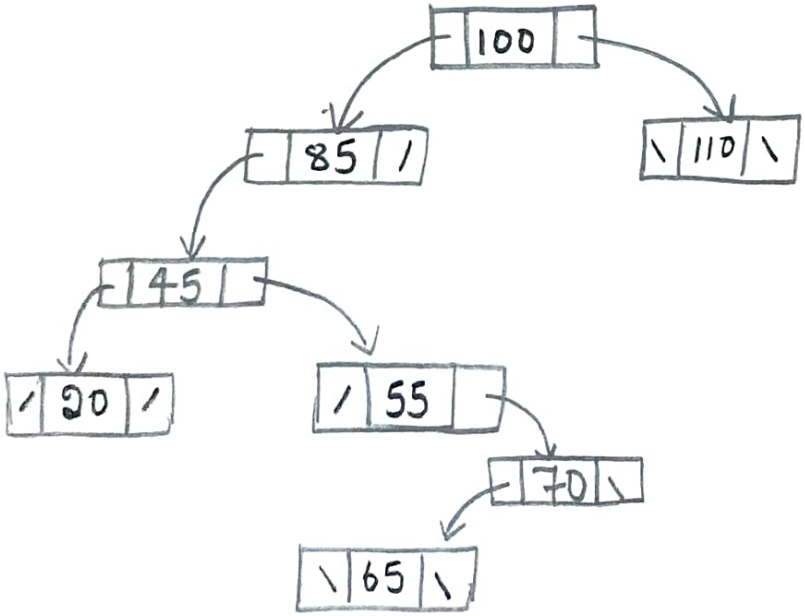


Array representation:

Array

100	85	110	45	-	-	-	20	55	-	-	-	70	65	-

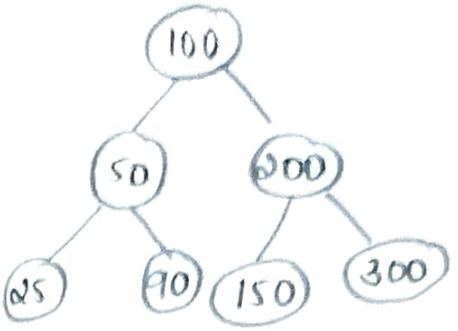
linked representation:



(9) a) write a c-function to search an element in a Binary Search tree (BST).

=>
```
void search(int item, NODE root)
{
    int key;
    NODE cur;
    printf("enter the Element to be Searched \n");
    scanf("%d", &key);
    if (root == NULL)
    {
        printf("tree is empty \n");
        return;
    }
    cur = root;
    while (cur != NULL)
    {
        if (cur->data == key)
        {
            printf("Number found \n");
            return;
        }
        if (key < cur->data)
            cur = cur->lchild;
        else
            cur = cur->rchild;
    }
    Printf("Number not found \n");
}
```

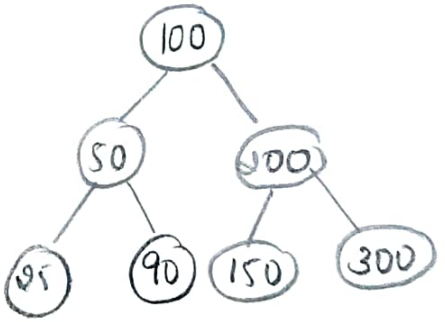
b) Write a c function to find minimum element in a Binary Search tree.



∴ Minimum value in BST → 25

⇒
```
void min (NODE root)
{
    NODE cur;
    if (root == NULL)
    {
        printf (" tree is empty \n");
        return;
    }
    cur = root;
    while(cur->llink != NULL)
    {
        cur = cur->llink;
    }
    printf("The minimum value of node is %d \n", cur->info);
    return;
}
```

c) write a c function to count number of leaf node in Binary tree.



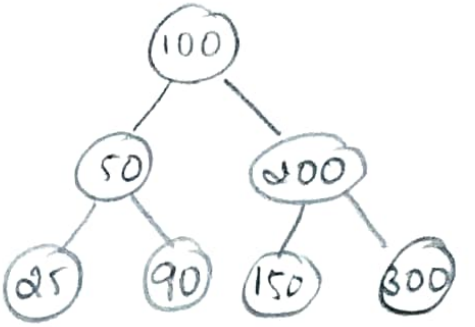
∴ The total leaf node are 4
25, 90, 150, 300

⇒
```
void count_leaf (NODE root)
{
    if (root == NULL)
    {
        printf (" tree is empty \n");
        return;
    }
    count_leaf (root->llink);
    if (root->llink == NULL && root->rlink == NULL)
    count ++;
```

```
    count_leaf (root->rlink);
    Printf (" The no^r, of nodes in a tree is %d \n",
                                        count);
}
```

---

d) write a c-function to find maximum element in a Binary search tree.



Maximum value in BST → 300

```
=> Void MAX (NODE root)
{
    NODE cur ;
    if (root == NULL)
    {
        printf (" tree is empty \n");
        return;
    }
    cur = root ;
    while (root -> rlink != NULL)
    {
        cur = cur -> rlink ;
    }
    Printf (" The max. no^r. of nodes is %d \n", cur -> info);
    return;
}
```

---

(10)(a) write a function to insert an item into an ordered binary search tree (Duplicate item not allowed).

```
void insert()
{
    NODE newnode, prev, cur;
    newnode = (NODE) malloc (size of (struct node));
    Printf ("enter the element to be inserted in tree \n");
    Scanf ("%d", & newnode -> data);
    newnode -> lchild = newnode -> rchild = NULL;
    if (root == NULL)
    {
        root = newnode;
        return;
    }
    prev = NULL;
    cur = root;
    while (cur != NULL)
    if (cur -> data == newnode -> data)
    {
        printf ("Duplicate is not possible \n");
        free (newnode);
        return;
    }
    prev = cur;
    if (newnode -> data < cur -> data)
        cur = cur -> lchild;
    else
        cur = cur -> rchild;
    if (newnode -> data < prev -> data)
        prev -> lchild = newnode;
    else
        prev -> rchild = newnode;
    return;
}
```

(b) with a diagramatic explaination, how do you create and construct a Binary search tree.

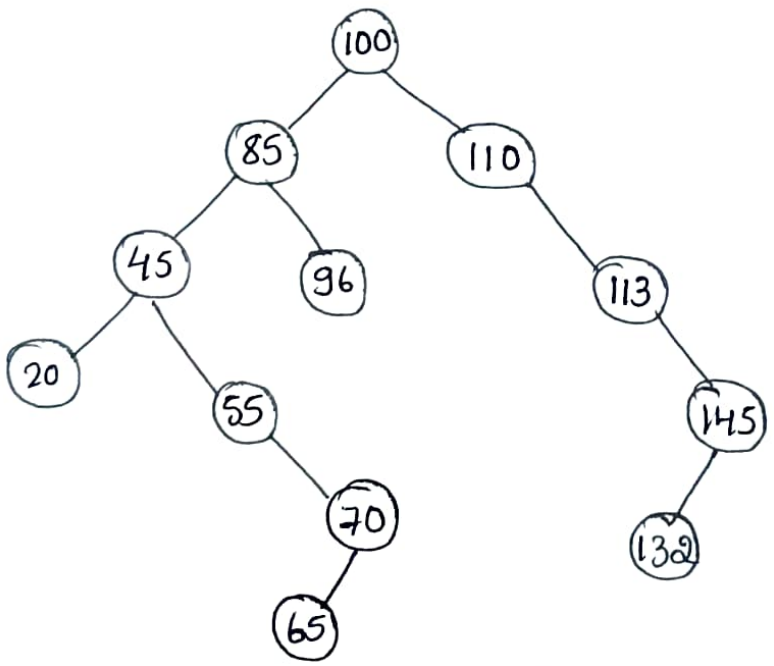
=> 1. First pick the first element of the array and make it root node, which means whenever an element is to be inserted, first locate its proper location.

2. pick the second element, then start searching from root node, then if the key value is less than the root node, search for the empty location in the left subtree and insert the data

3. otherwise, search for the empty location in the right subtree and insert the data.

diagramatic representation

100 85 45 55 110 80 70 65 113 145 132 96



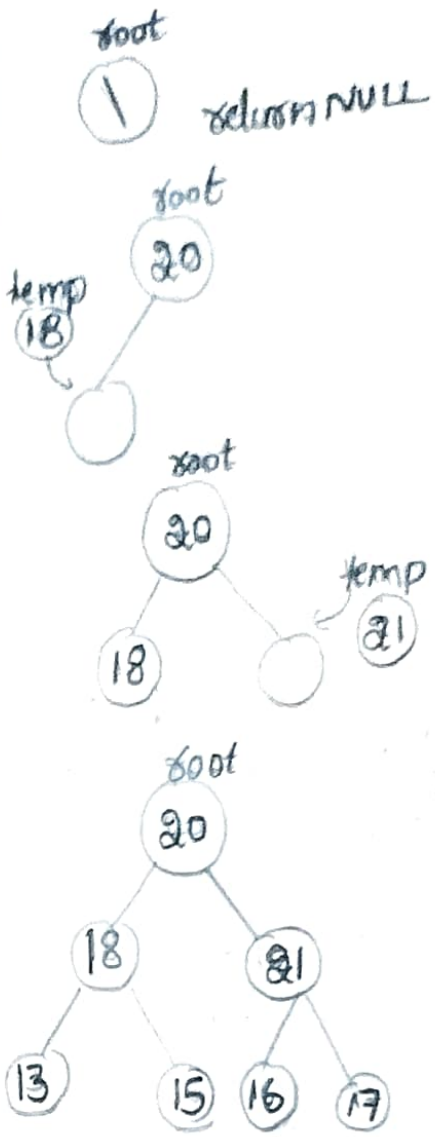
(11) write the routines for :

i) copying binary trees

ii) testing for equality of binary trees.

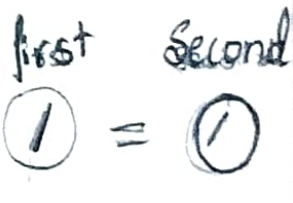
⇒ i) copying binary trees :

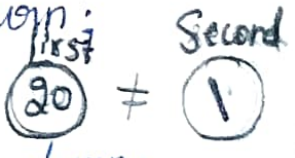
```
NODE copy(NODE root)
{
    NODE temp;
    if(root == NULL)
        return NULL;
    temp = getnode();
    temp->info = root->info;
    temp->lptr = copy(root->lptr);
    temp->rptr = copy(root->rptr);
    return temp;
}
```



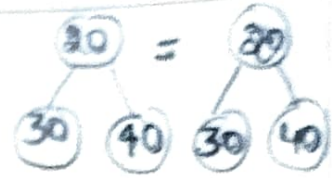
⇒ ii) Testing for equality of binary trees :

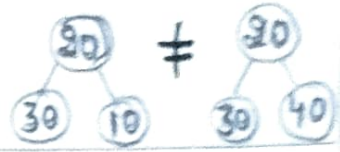
```
void equal(NODE first, NODE second)
{
    if(first == NULL && second == NULL)
        printf("Trees are equal \n"); return;
    if(first == NULL || second == NULL)
        printf("Trees are not equal \n"); return;
    if(equal(first->lptr, second->lptr) &&
       equal(first->rptr, second->rptr) &&
       equal(first->data, second->data))
```





```
    printf("Trees are equal \n"); return;
  else
    printf("Trees are not equal \n"); return;
}
```





(12) Define expression tree using a c-function, explain how do you construct a expression tree. construct an expression tree for : a + b * c/f ^g-h.

~~An infix expressi~~

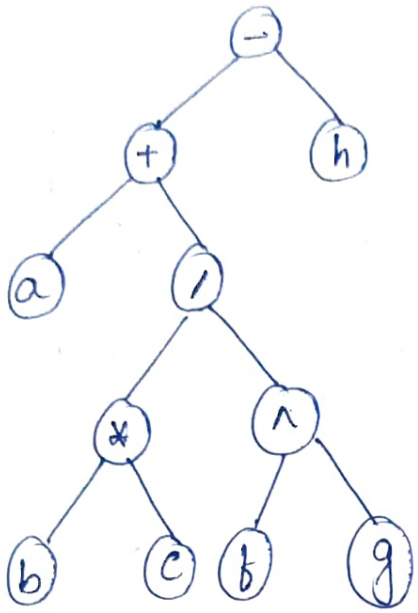
~~A Sequence of Operators and operands that reduce to a single value it~~ x

An expression tree is a binary tree that satisfy the following Properties :

* Any leaf is an operand
* The root and Internal nodes and operators
* The subtrees represent sub expressions with root of the subtree as an Operator.

An Infix expression consisting of operators and Operands can be represented using a binary tree with root as the Operator. The left and right subtrees are the left and right Operands of that operator. A node containing an operator is not a leaf where as a node containing an operand is a leaf.

a + b * c / f ^ g - h

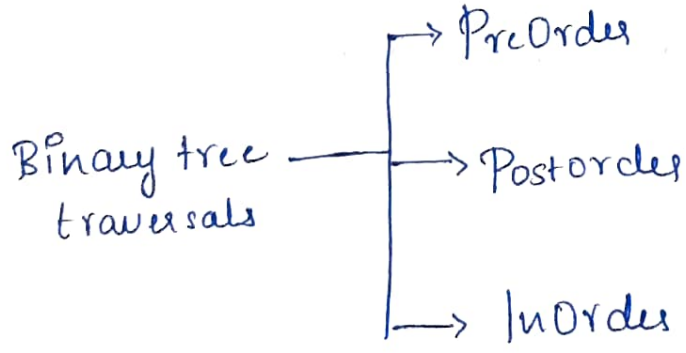


---

(13) Define Binary tree traversal method. List and explain the different binary tree traversal methods along with c-funtions.

Traversing is a method of visiting each node of a tree exactly once in a systematic order. During traversal, we may print the info field of each node visited.

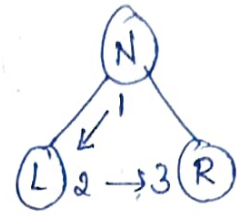
Different traversal techniques of a binary tree: The various traversal techniques of a binary tree as shown below:



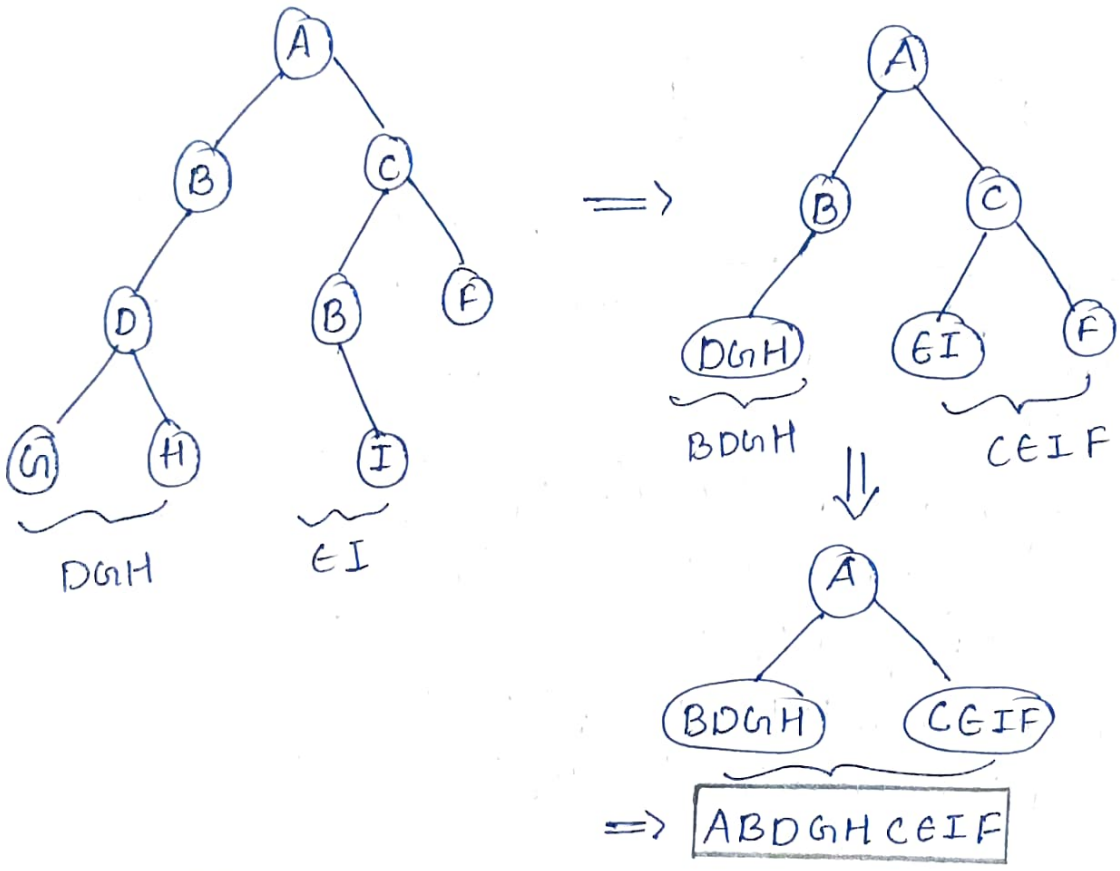
Preorder Traversal

The preorder traversal of a binary tree can be recursively defined as follows:

1. Process the root Node.
2. Traverse the left subtree in preorder.
3. Traverse the Right subtree in preorder.



For example, let us traverse the following tree in preorder.



Function to traverse the tree in preorder

```
Void preorder (NODE root)
{
```

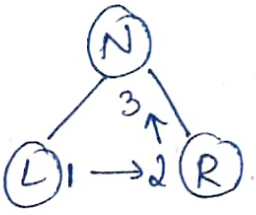
```
if (root == NULL)
return;
Printf ("%d", root -> Info);
Preorder (root -> llink);
Preorder (root -> rlink);
```

}

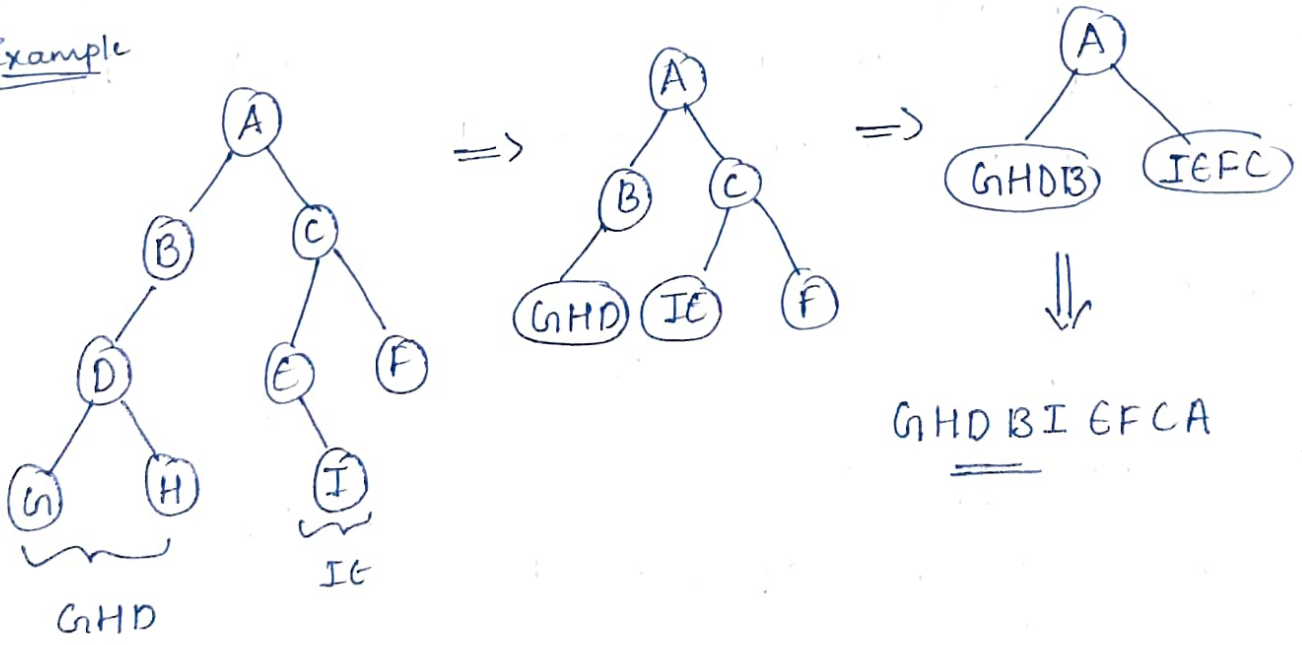
Postorder Traversal

The postorder traversal of a binary tree can be recursively as follows:

1. Traverse the Left Subtree in postorder [L]
2. Traverse the Right Subtree in postorder [R]
3. Process the root Node [N].



Example



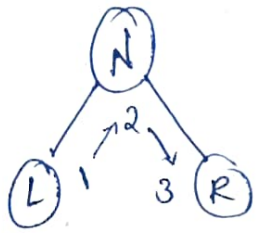
Function to traverse the tree in postorder

```
Void postorder (NODE root)
{
    if (root == NULL)
    return;
    Postorder (root -> llink);
    Postorder (root -> rlink);
    Printf ("%d", root -> info);
}
```

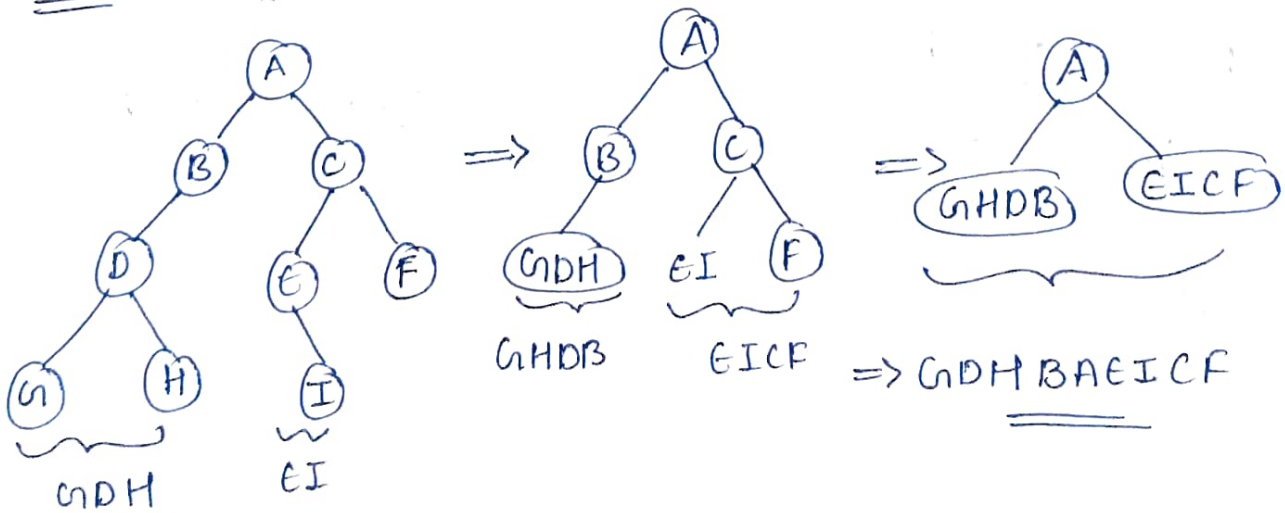
Inorder

The Inorder traversal of a binary tree can be recursively defined as follows:

1. Traverse the Left Subtree in Inorder [L].
2. Process the root Node [N]
3. Traverse the Right Subtree in Inorder [R].



Example



Function to traverse the tree in Inorder

```
Void Inorder (NODE root)
{
    if (root == NULL)
    return;
    Inorder (root -> llink);
    Printf ("%d", root -> Info);
    Inorder (root -> rlink);
}
```

(14) Define binary search tree. Write the recursive search and Iterative search algorithm for a binary search tree.

A binary search tree is a binary tree in which for each node say X in the tree, elements in the left-subtree are less than Info(X) and elements in the right subtree are greater than Info(X). Every node in the tree should satisfy this condition, if left subtree or right subtree exists.

Recursive Search algorithm

NODE Search (int item, NODE root)

A binary search tree T is in memory and item of information is given.

This algorithm find item from the tree recursion by using

Step 1 : If root == NULL, return NULL /* empty tree */

Step 2 : If item == cur -> Info, then it return cur
/* If item found */

Step 3 : If item < root -> Info return Search (item, root -> llink)
/* Search recursively left side */
else return search (item, root -> rlink);
/* Search recursively right side */

Iterative Search algorithm :

Algorithm of binary search tree using iteration

NODE Search (int item, NODE root)

A binary search tree T is in memory, and item of information is given this algorithm find item from the tree by using iteration

Step 1 : We declare NODE as a cur

Step 2 : If root == NULL, then return NULL /* empty tree */

Step 3 : root == cur, cur is point to root

Step 4 : While cur != NULL for search for the item.

Step 5 : If item == cur -> Info, then it return cur
/* If found return the node */

Step 6 : If item < cur -> Info then cur = cur -> llink
/* Search towards left */

Step 7 : Repeat the steps 5 and 6
until while cur != NULL

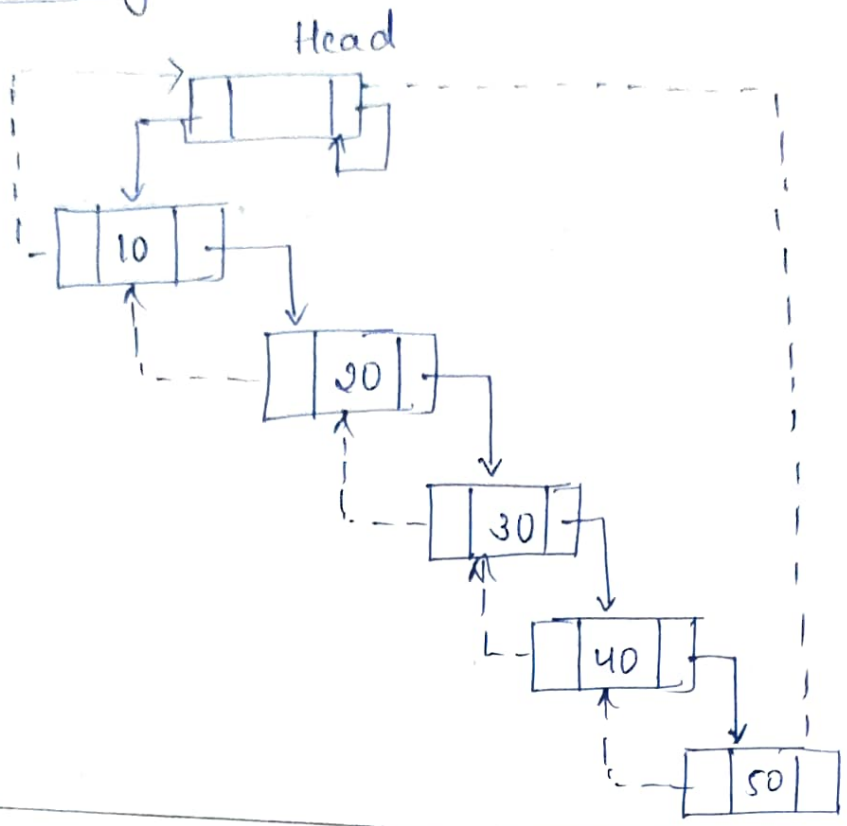
Step 8 : If item is not found return the NULL.

(15) What is the advantages of the threaded binary tree, over binary tree? Explain the construction of threaded binary tree for 10, 20, 30, 40 and 50.

⟹ The various advantages of the binary tree :

* In a binary tree, more than 50% of link field have to (null) values and more memory space is wasted by storing to (null) values. The wastage of memory space is avoided by storing address of some nodes.
* Traversing of a threaded binary tree is very fast. This is because it does not use Implicit or explicit stack.
* Computations of predecessor and successor of given nodes is very easy and efficient.
* Any node can be accessed from any other node using threads, upwards movement is possible and using links downward movement is possible. Thus, in a a threaded binary tree, we can move in either directions. This is not possible in un-threaded binary tree.
* Even though Insertion into a threaded binary tree and deletion from a threaded binary are time consuming operations, they are very easy to Implement.

In threaded binary tree



List the rules to construct the threads. Write the routines for Inorder traversal of a threaded binary tree

⇒
- A binary tree is threaded by making all right teen pointers that would normally be null point to the in-order beneficiary of the node (if it exists), and all left child pointers that would normally be null point to the in-order precursor of the node.
- The idea of threaded binary trees is to make in order traversal quicker & do it without stack & without recursion.

Inorder traversal of Right Inthreaded Binary Tree

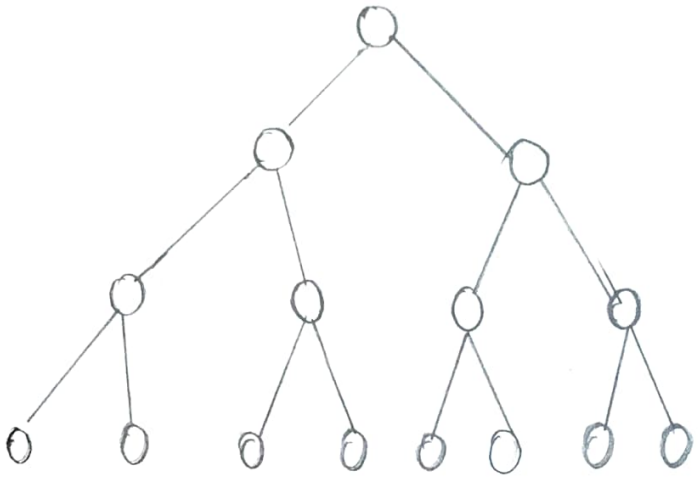
```
Void Inorder (NODE head)
{
    NODE temp;
    if (head -> llink == head)
    {
        Printf ("Tree is Empty\n");
        return;
    }
    Printf ("Inorder traversal of tree is \n");
    temp = head;
    for(;;)
    {
        temp = Inorder_successor (temp);
        if (temp == head)
        return;
        Printf ("%d", temp -> Info);
    }
}
```

To find Successor of Right Inthreaded Binary tree

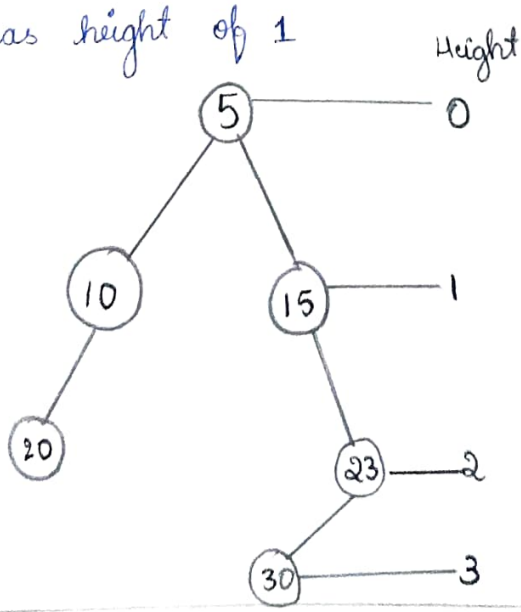
```
NODE InOrder_Successor(NODE x)
{
    NODE temp;
    temp = x->rlink;
    if (x->rthread == 1)
      return temp;
      while (temp->llink != NULL)
        temp = temp->llink;
        return temp;
}
```

16 Explain the following with an example

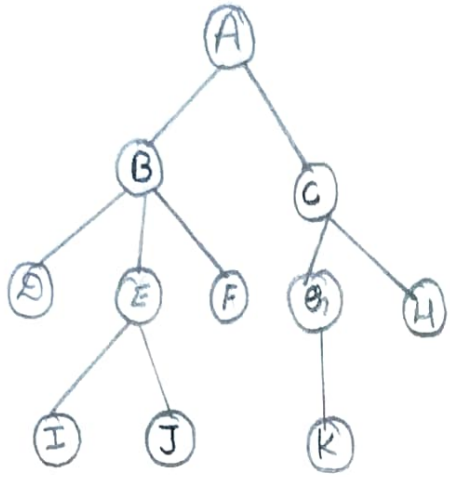
i) Complete binary tree : A binary tree is a complete binary tree if all the levels are completely filled except possibly the last level and the last level has all keys as left as possible.



ii) Height of the tree : Height of the tree is the length of the path from root of that tree to its farthest tree. A tree with only root node has height 0 and a tree with zero nodes would be considered as empty. An empty tree has height of 1



iii) Leaf node : A leaf node is a node with no child.



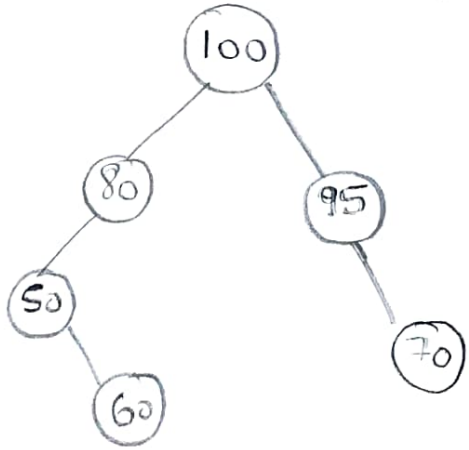
Here, D, I, J, F, K & H are leaf nodes

iv) out degree of a node :

The number of subtrees of a node is called its degree

For example :

* The node 100 has two subtrees. So, degree of node 100 is 2.
* The node 80 has one subtree. So, degree of node 80 is 1.
* The node 70 has no subtree. so, degree of node 70 is 0.



17 What is tree? with suitable example define

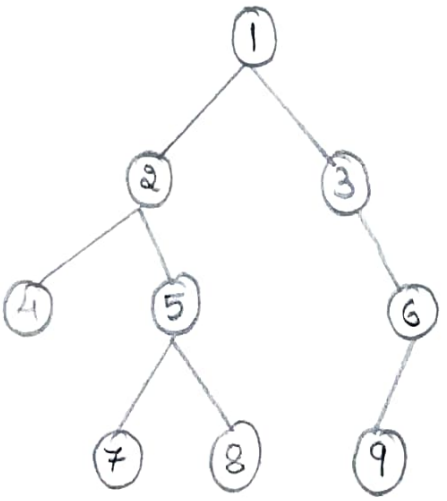
i) Binary tree
ii) Level of tree
iii) ~~complete binary tree~~
iv) Degree of the tree
v) ~~Siblings~~

A tree is a representation of the non-linear data structure. It is a finite set of one or more nodes such that;

i) There is a specially designated node called the root
ii) The remaining nodes are partitioned into $n \geq 0$ disjoint sets $T1 \ldots Tn$

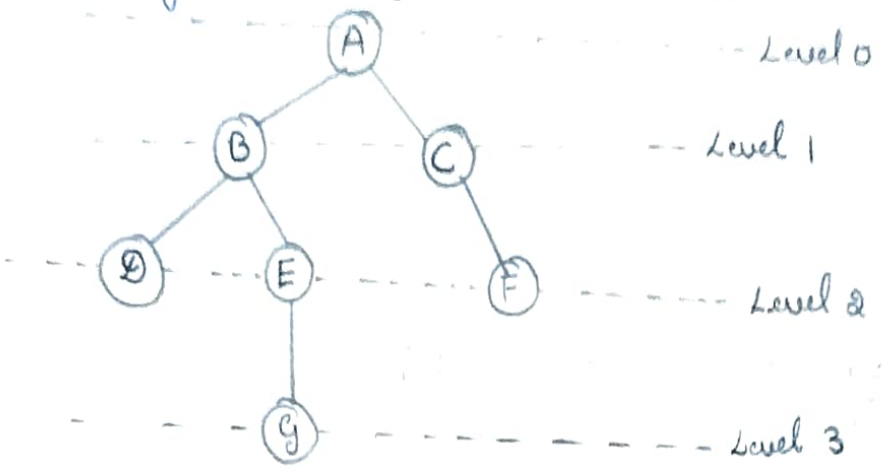
---

i) Binary tree: A binary tree is a finite set of nodes that is either empty or consists of a root and two disjoints binary trees called the left subtree and the right subtree

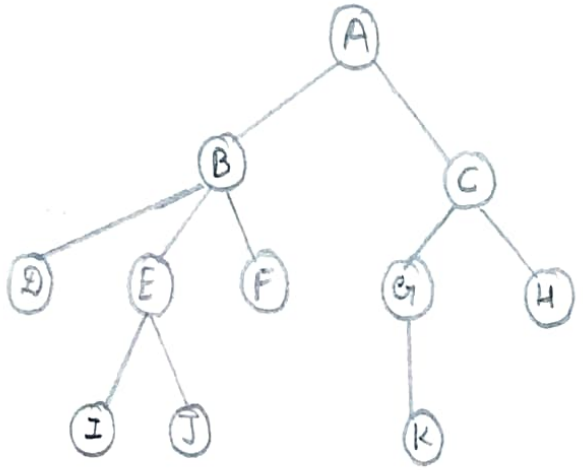


Binary tree

ii) **Level of tree :** A level is the number of parent nodes corresponding to a given a node of the tree.

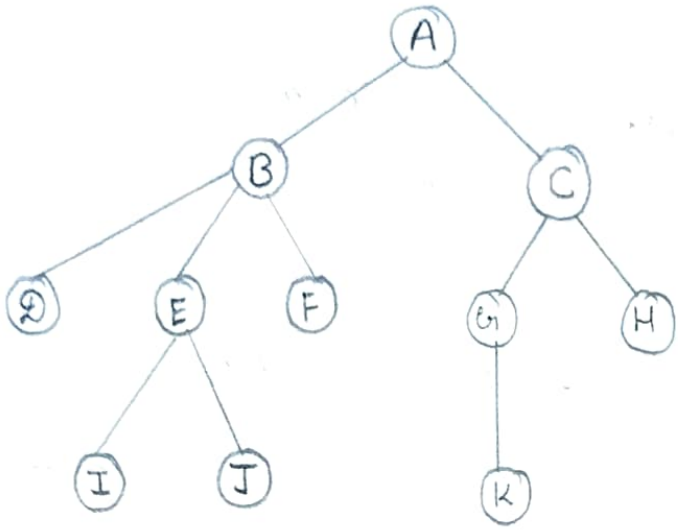


iii) **Degree of the tree :** It is the maximum degree of the nodes in the tree.



Here degree of B is 3
Here degree of A is 2
Here degree of F is 0

vi) **Siblings :** In a tree data structure, nodes which belong to same parent are called as siblings.



Here B & C are Siblings

Here D, E, F are siblings

Here G & H are siblings

Here I & J are siblings

(18) Properties of Binary tree?

Lemma:

(a) The maximum number of nodes on level i of a binary tree $= 2^i$ for $i \geq 0$

(b) The maximum number of nodes in a binary tree of depth $k = 2^k - 1$

Proof: consider, the following complete Binary tree & observe the following factors from the complete binary tree.

Number of nodes at level 0 = 1 = 2^0

Number of nodes at level 1 = 2 = 2^1

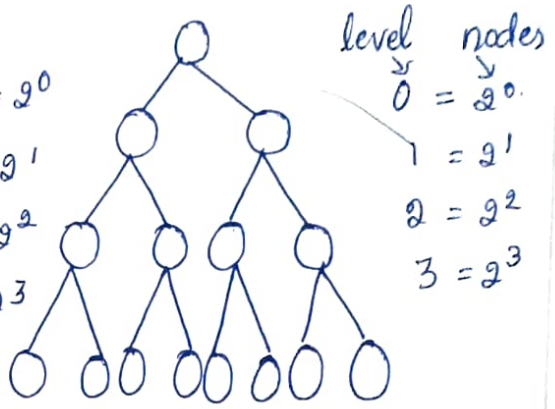
Number of nodes at level 2 = 4 = 2^2

Number of nodes at level 3 = 8 = 2^3

- - - - - - - -

- - - - - - - - -

Number of nodes at level $i = 2^i$.



Total number of nodes in the full binary tree of level $i = 2^0 + 2^1 + 2^2 + \cdots 2^i$.

The above series is a geometric progression whose sum is given by $S = a(r^n - 1)/(r-1)$

where $a = 1$, $n = i+1$ & $r = 2$.

So, total number of nodes $n_t = a(r^n - 1)/(r-1)$

$= 1(2^{i+1} - 1)/(2-1)$

$= 2^{i+1} - 1$

Therefore total number of nodes $\boxed{n_t = 2^{i+1} - 1}$ substituting $i = 3$

we get $n_t = 15$ which is total number of nodes in a fully binary tree

The depth of the tree k = max. level + 1

= $i + 1$

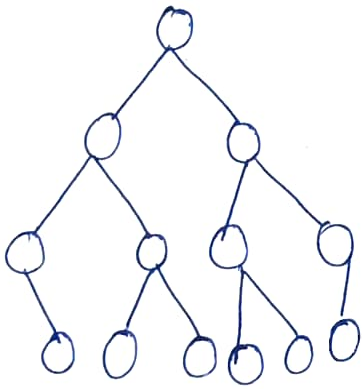
Substituting this value in above equation we get

$$\boxed{n_t = 2^k - 1}$$

Lemma: The number of leaf nodes is equal to number of nodes of degree 2.

Now, lets see the relation b/w the number of leaf nodes & degree-2 nodes.

consider, a binary tree of degree-2 nodes. That is, each node in the tree should have maximum of two children. A node need not have any children, or it can have only one child or it can have two children. But, a node cannot have more than two children.



let no^r. of nodes of degree-0 = n_0

let no^r. of nodes of degree-1 = n_1

let no^r. of nodes of degree-2 = n_2

let total no^r. of nodes in the tree = $n = n_0 + n_1 + n_2$ — (1)

observe, from the tree that total number of nodes is equal to the total no^r. of branches (B) plus one. i.e,

$n = B + 1$ — (2)

If there is node with degree 1, no^r. of branches = 1

So, for n_1 number of nodes of degree 1, no^r. of branches = n_1 — (3)

If there is node with degree 2, no^r. of branches = 2

so, for n_2 no^r. of nodes of degree 2, no^r. of branches = $2n_2$ — (4)

add (3) + (4) $B = n_1 + 2n_2$ — (5)

substitute (5) in (2) we get, $n = n_1 + 2n_2 + 1$ — (6)

The relation b/w no^r. of leaf nodes & no^r. of nodes of degree-2 can be obtained by sub (6) from (1).

i.e. $n = n_0 + n_1 + n_2$

$- \; n = n_1 + 2n_2 + 1$

$0 = n_0 - n_2 - 1$

By re-arranging we get, $\boxed{n_0 = n_2 + 1}$