Trees

Arrays

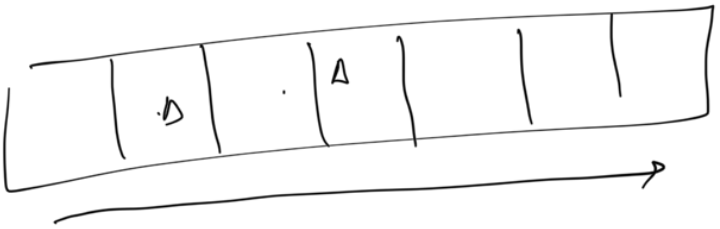
LL

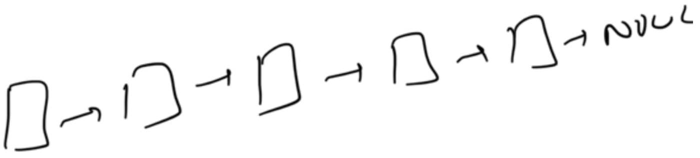
Queus

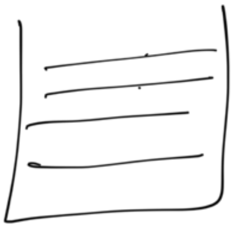
Stack

Hash Map Hash Sets

Linear

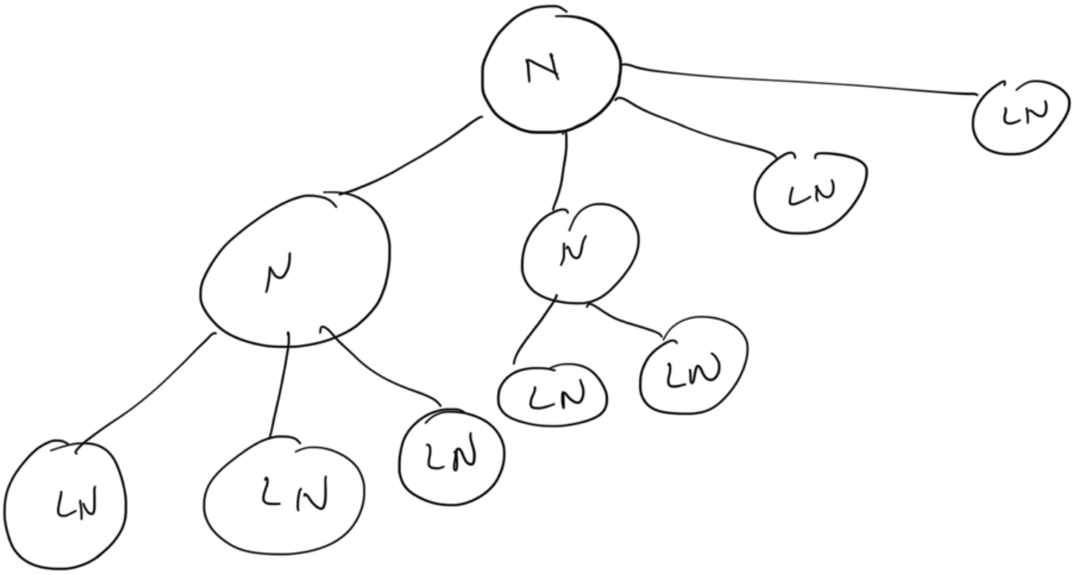


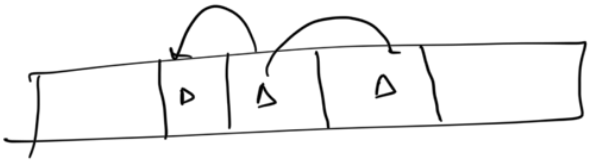


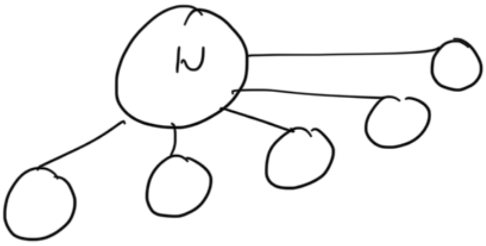


Hirarchial DS

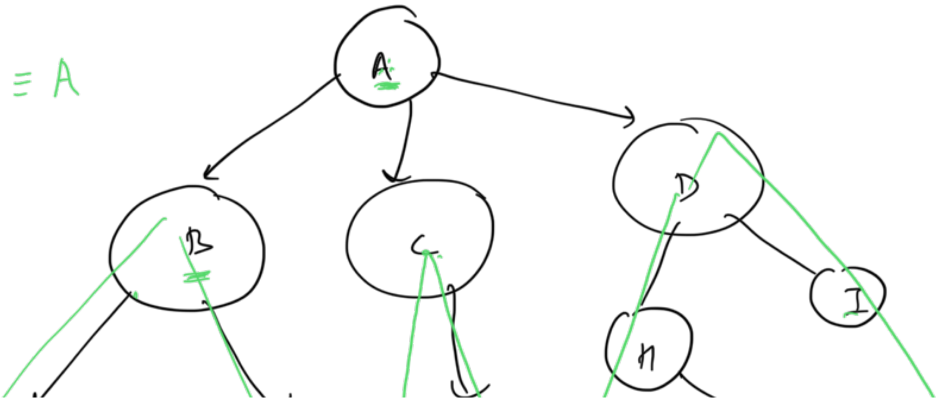
Tree

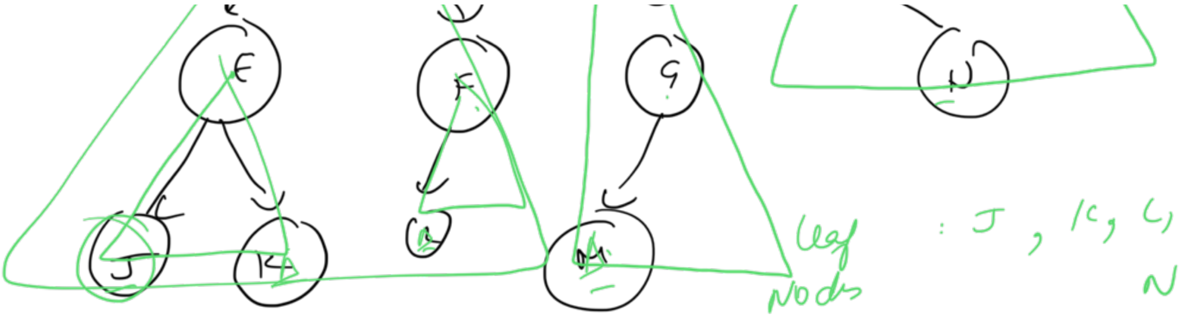




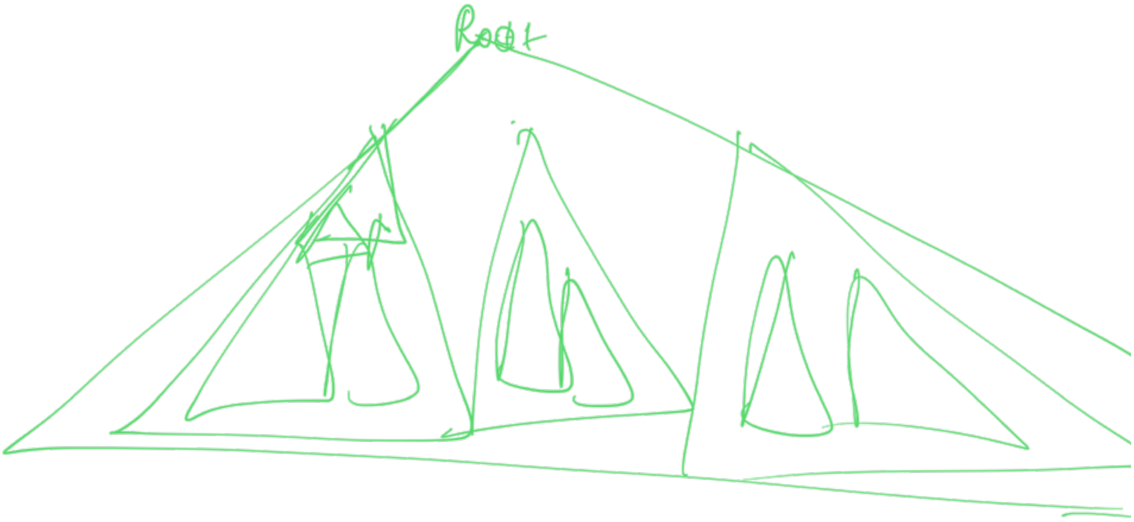


Root ≡ A



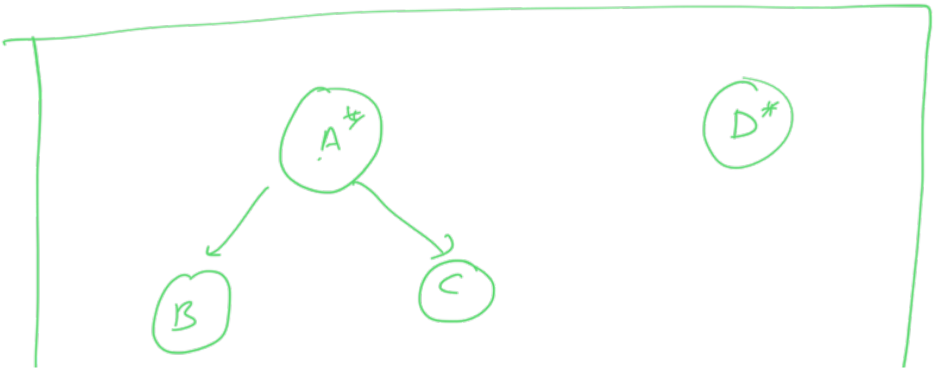


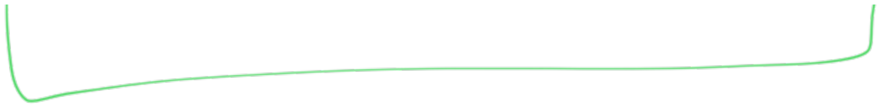
leaf Nodes : J, K, L, M
N

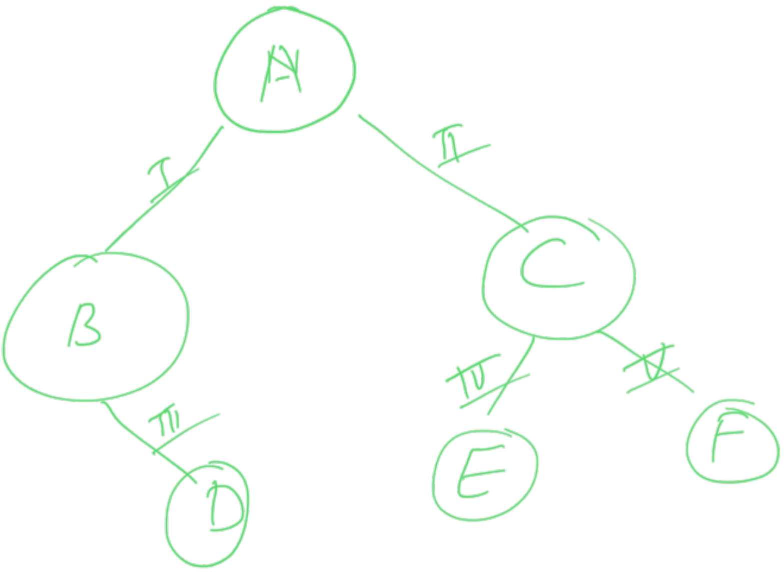


Tree ≡ Recursion

Tree : a collection of nodes, N
in which all of the nodes are connected
and #edges = N-1



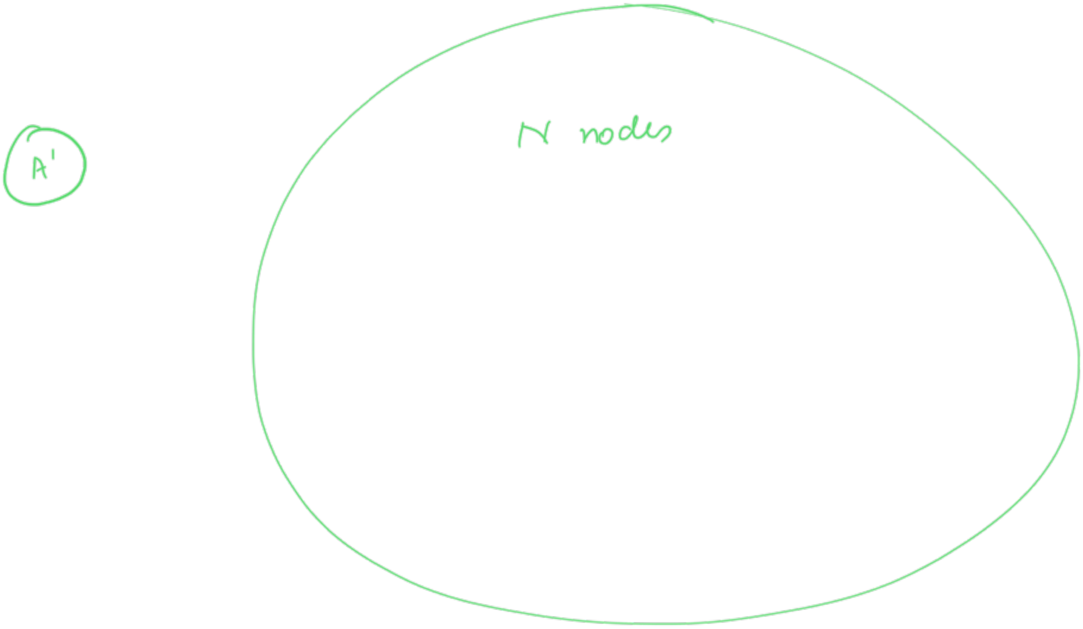


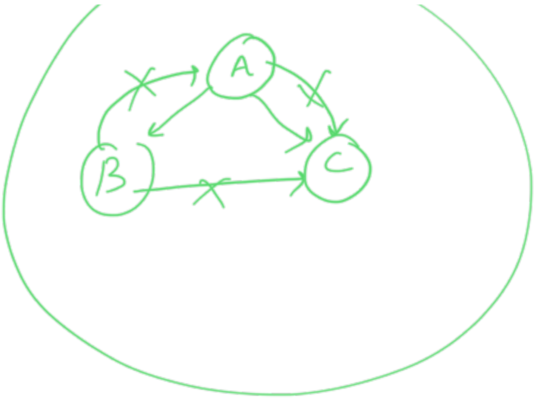


6 Nodes

5 edge

Tree ≡ collection of Nodes





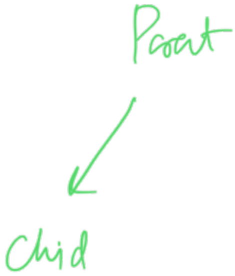
N nodes

0 edge 1st node

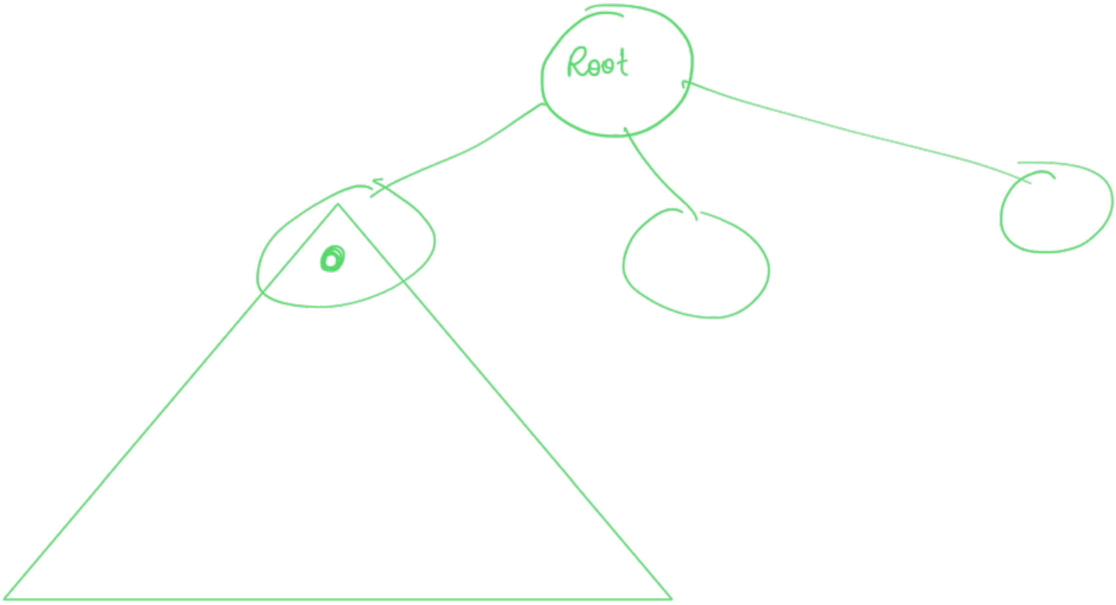
1 edge 2nd node

1 edge 3rd

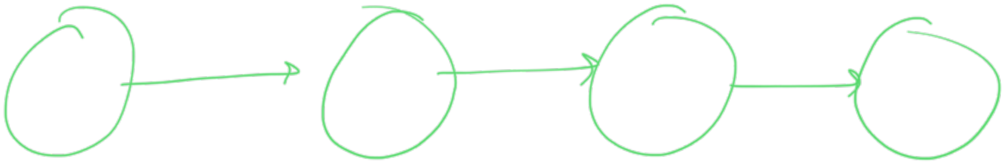
atleast N-1 edges

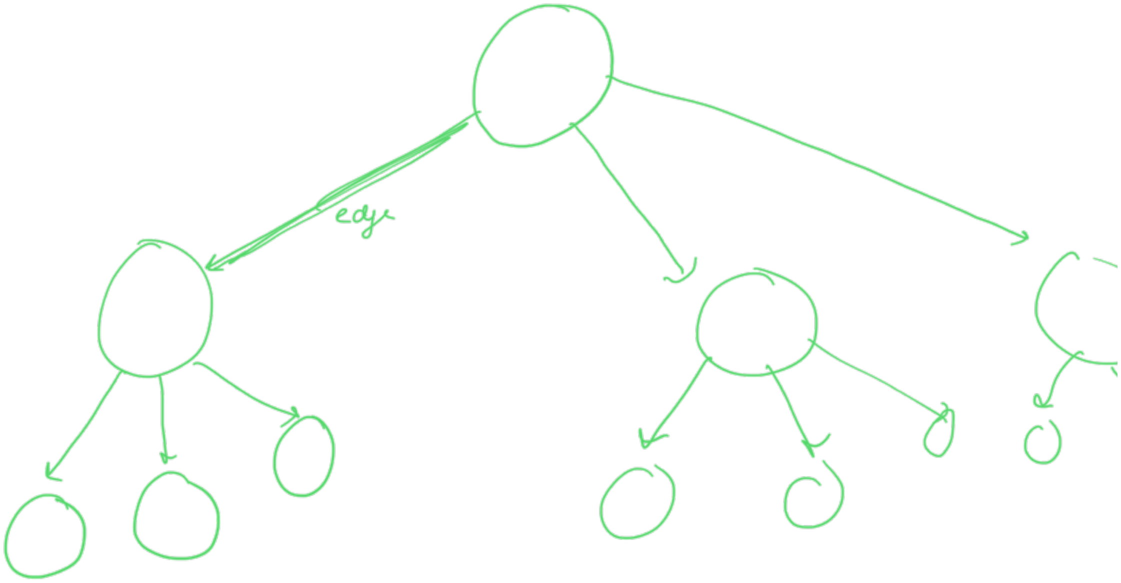


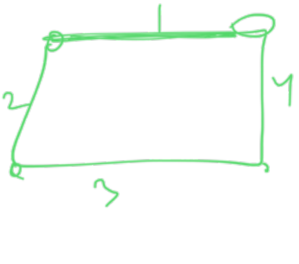
$$\boxed{N \text{ nodes} \equiv N-1 \text{ edges}}$$



L.L



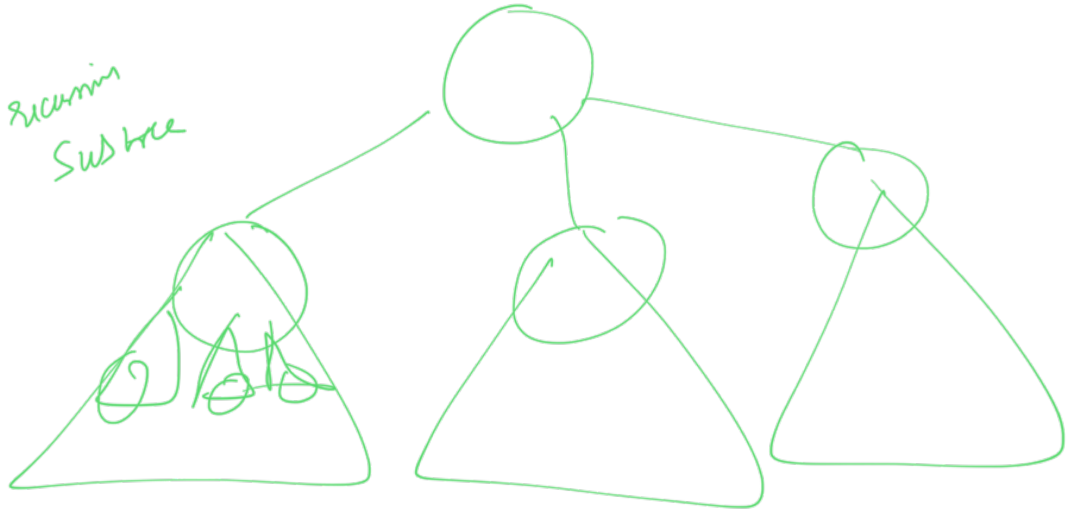


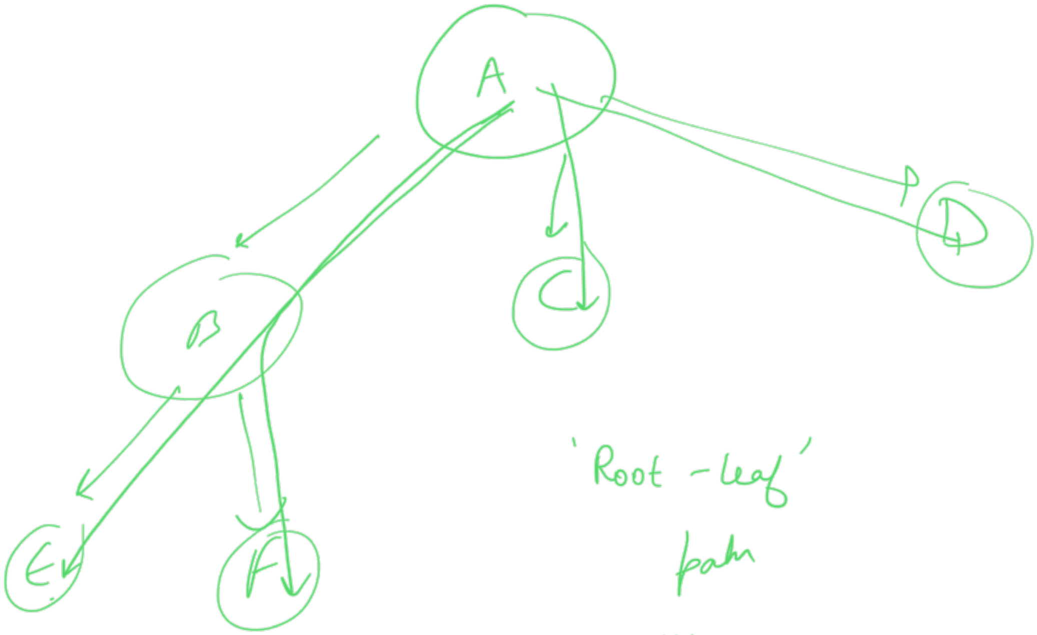


nodes ≡ data member

Tree

edges → structure

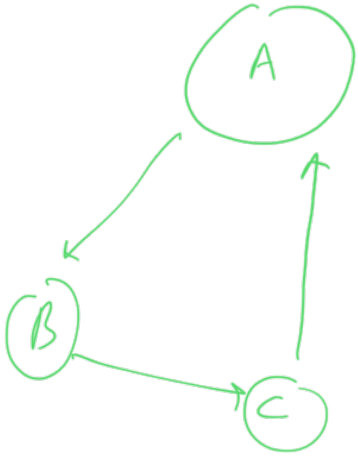




'Root - leaf'
path
|||
LL

NOT A TREE

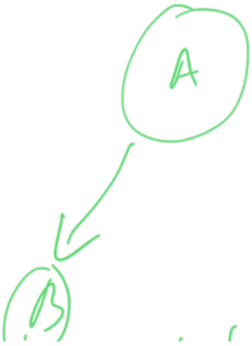
every node other than the root must have 1 parent



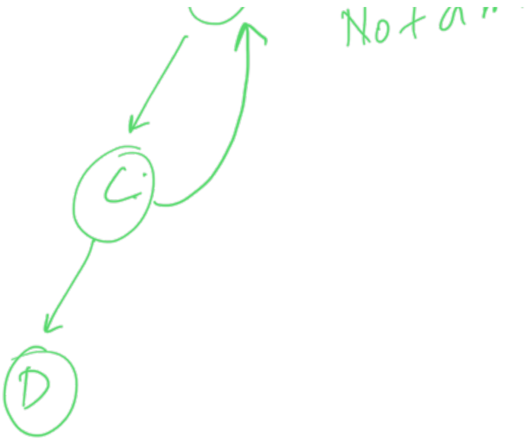
NOT a tree

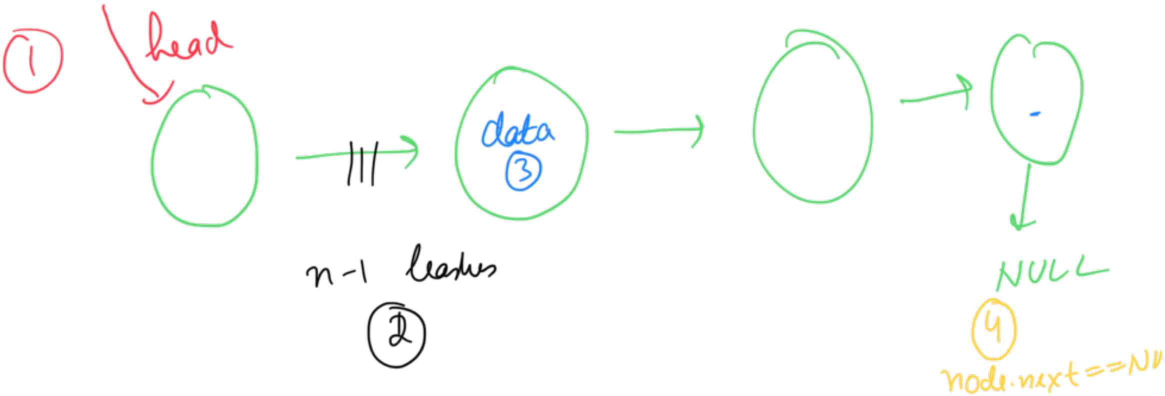
loop X

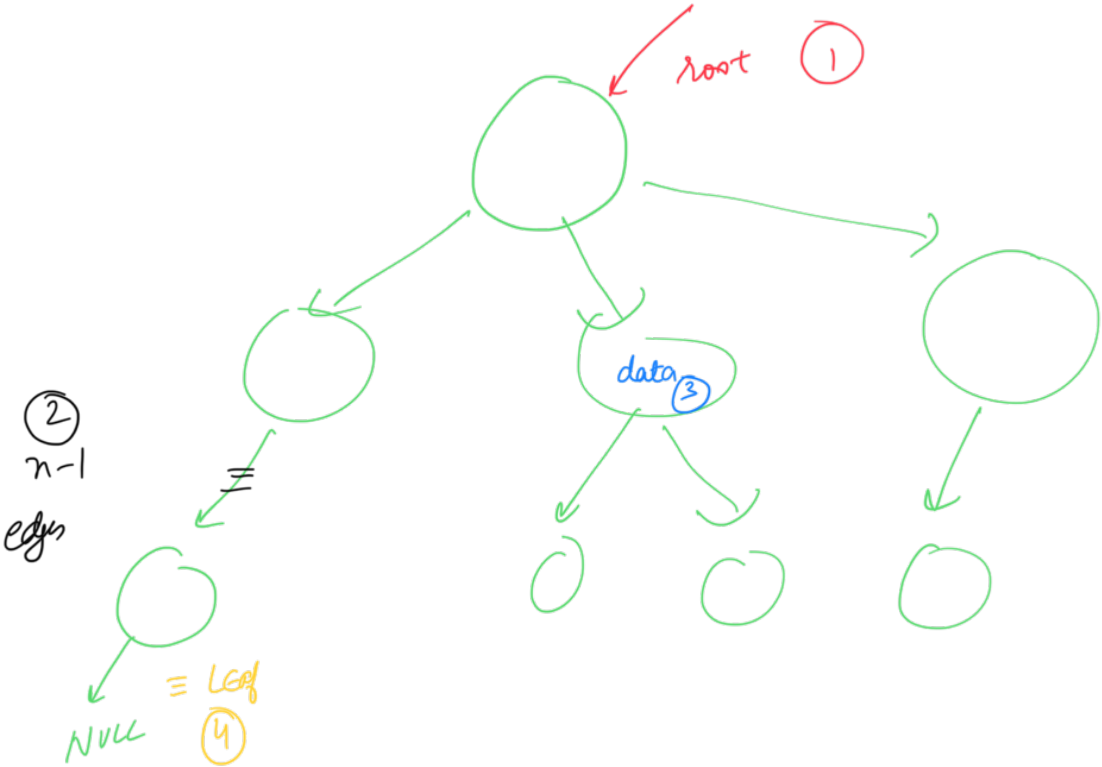
N edges X



a tree


Not a
C
D

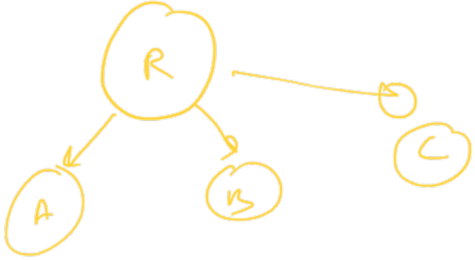

1
head
data
3
n-1 leaves
2
NULL
4
node.next==NU


root
1
data
3
2
n-1
edges
≡
NULL
≡ Leaf
4

Tree is basically hierarchical L.L

L.L is basically a special type of a Tree

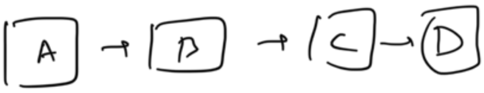
= Tree is a more generic d.s as compared to a LL



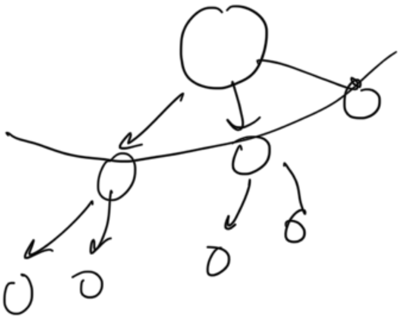
Path Root to Leaf = L.L

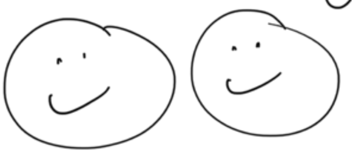
L.L is a Tree with every node == 1 child

```
class LinkedlistNode
{
    int data ✓
    LinkedlistNode next
}
```



```
class TreeNode
{
    int data; ✓
    TreeNode [] children;
}
```





Binary Tree

k-ary Tree : max number of children one node can have = k n -ary Tree with n = 15

Binary Tree

k = 2

every node in the binary tree will have atmost 2 children

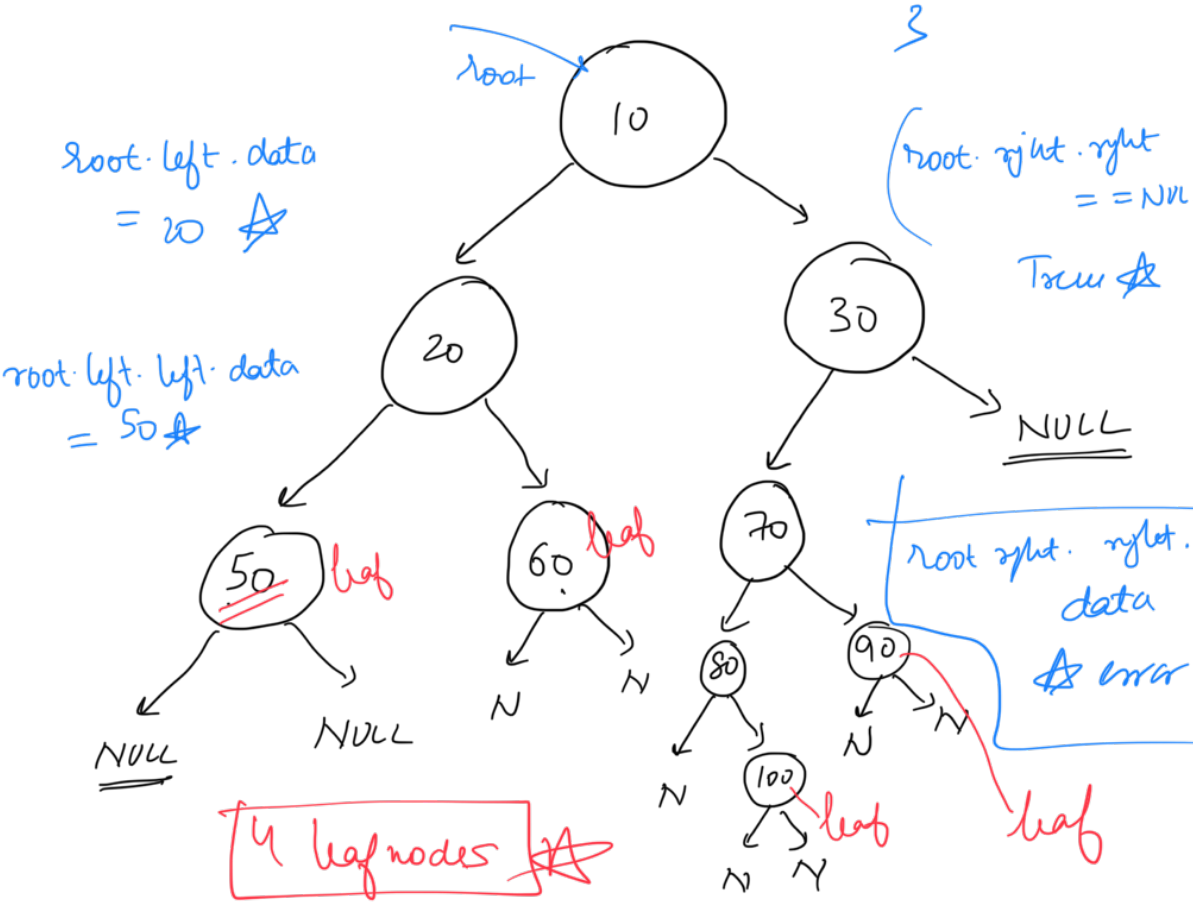
0 child or 1 child or 2 children

```
class TreeNode
{
    int data ;
    TreeNode[] children;
}
```

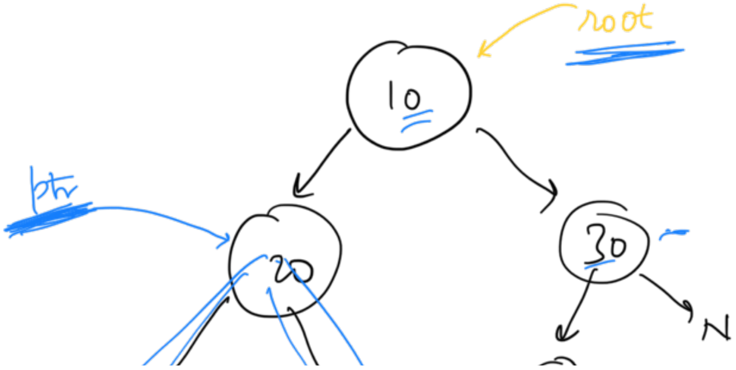
```
class BinaryTreeNode{
        int data;
        TreeNode leftChild
```

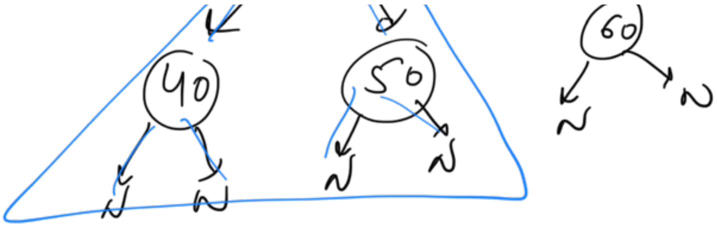
```
TreeNode createNode(i
{     //mem alloc
  TreeNode temp
        = new TreeNo
        TreeNode rightchild
    }
```

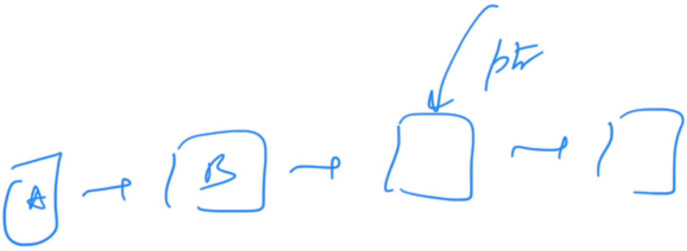
```
temp.data = v;
temp.leftchild = NU
temp.rightchild = N
return temp
}
```



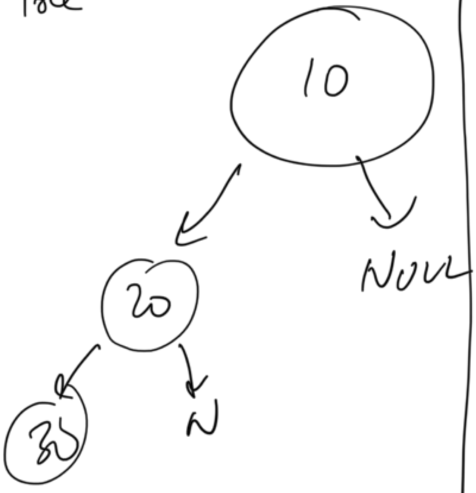
Leaf ≡ node with 0 children

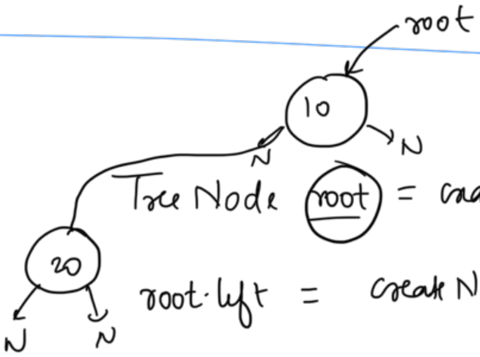






Tree

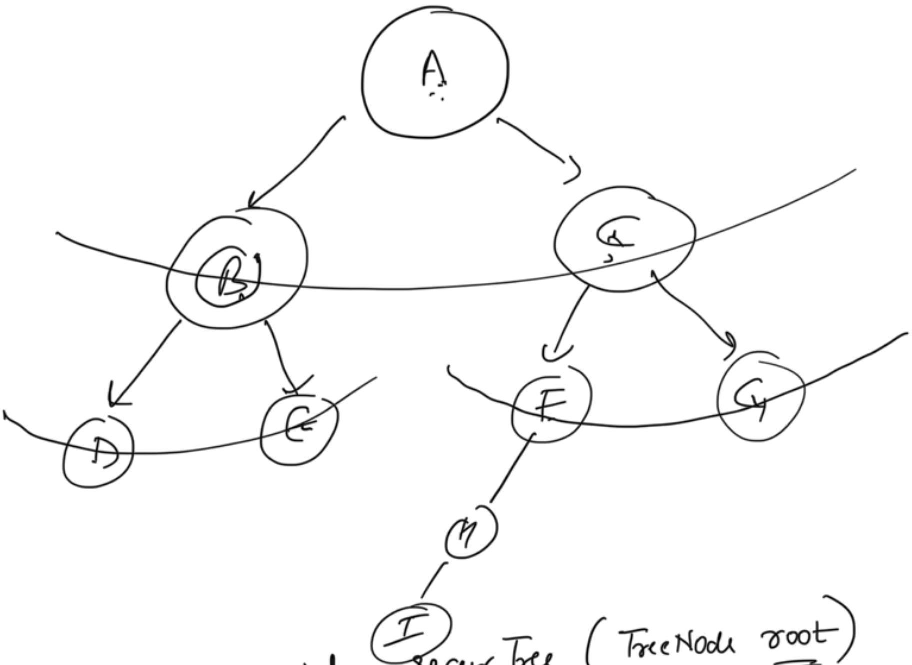




```
Tree Node root = create Node (10);
root.left = create Node (20);
root.left.left = createNode (30);
```

```
void traverseLL (LLNode head)
{
        LLNode temp = head;
        while (temp.next != NULL)
        {
                prnt(temp.data)
                temp = temp.next
        }
```

}

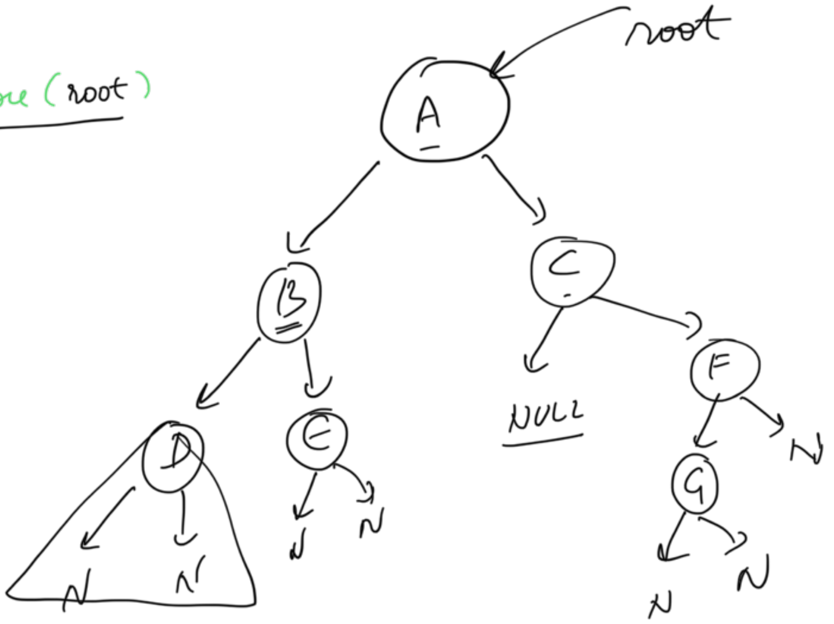


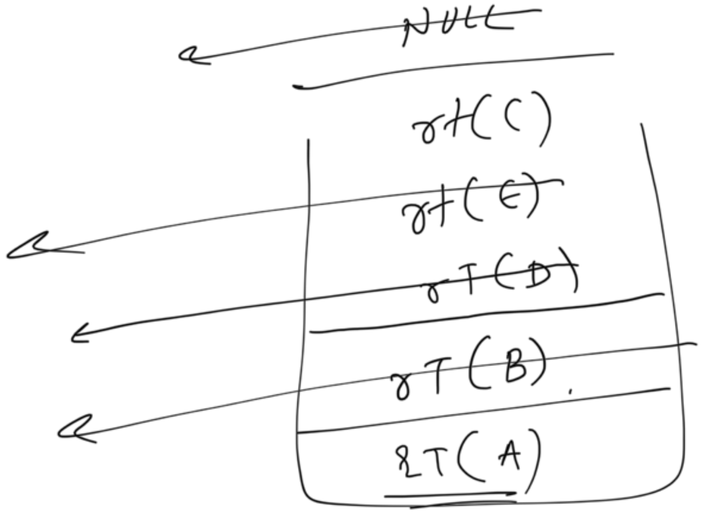
```
void recurTree (TreeNode root)
{
        if (root == NULL)   //sanity
            return;

Base Case   if (root.left == NULL AND
                root.right == NULL)
                return

        // recur
                    (I)
                → recurTree (root.leftchild)
print (root.data)   (II)
                → recurTree (root.rightchild)
                    (III) →
}
```

recursTree (root)





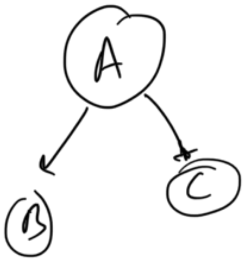
recursTree (A)

I

print(root.data) = A ✓

recursTree(root. leftchild) .

recursTree (roo.~~left~~ child) .



A

new func

|0| recurT(root. leftchild)

II

II

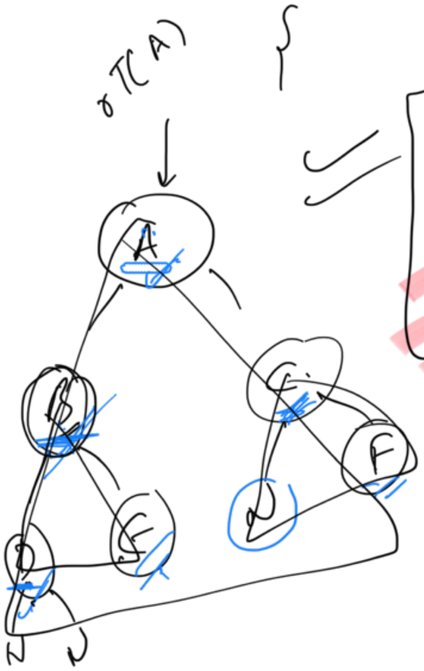
```
102 → print ( root . data) ≡ A ↓ ☺
103 recur T ( root · right child)
```

III

```
recur Tree ( root leftchild)
recur T ( root · right child)
print ( root . data); ≡ A
```

```
void   recurTree ( TreeNode root )
{
    // sc + b.c
    if(root == NULL)  return
    // bc
    if (root.left == NULL  AND  root.right == NULL
            return

    // recursive
    print ( root . data)
    recurTree ( root . leftchild)
    recur Tree ( root . right child).   noly par
}
```

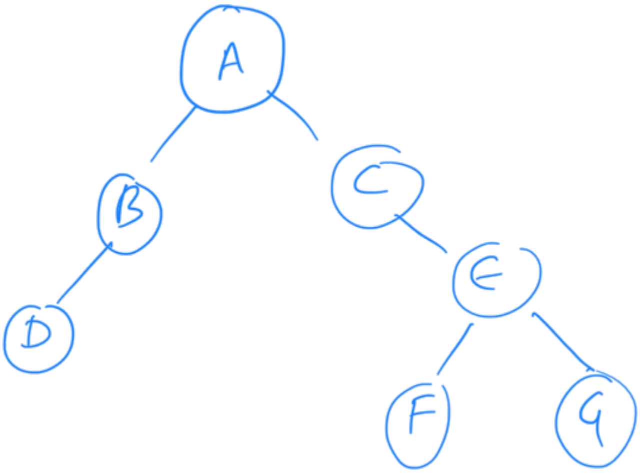


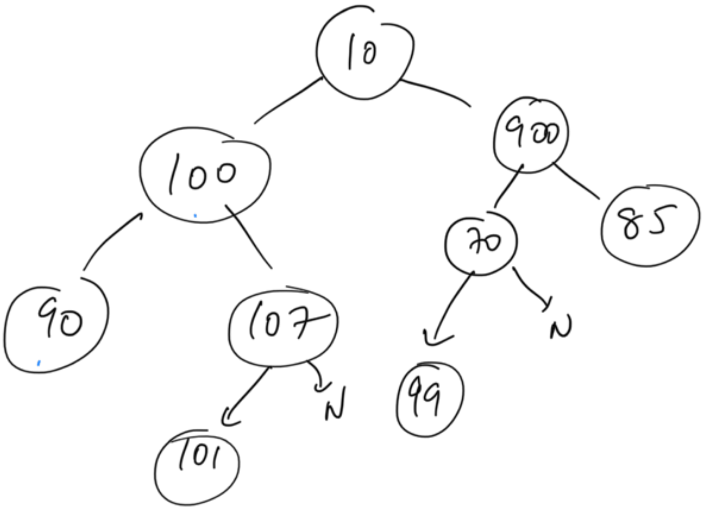
A|B|D|E|C|F

Preorder Traversal

(recursively print yourself first)

A
B
D
C
E
F
G





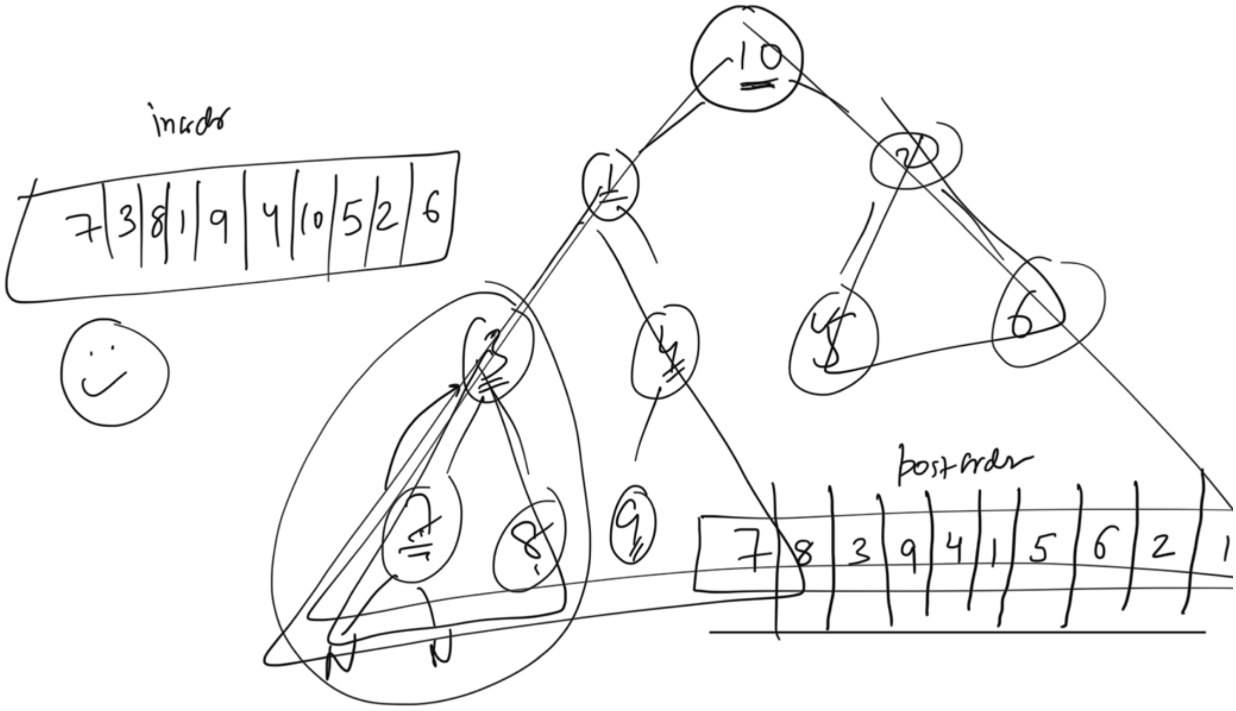
10
100
90

40
107
101
900
70
99
85

Inorder

// recursion

```
101  recurTree ( root. leftchild)
102  print (root.data)          ⟶ A
103  recurTree ( root. rightchild)  ⟶
```



inorder

7	3	8	1	9	4	10	5	2	6

postorder

7	8	3	9	4	1	5	6	2	1

postorder

//recursive

A'

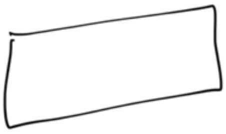
recurTree(root.leftChild) ✓

A''

recurTree(root.rightChild) ✓

A'''

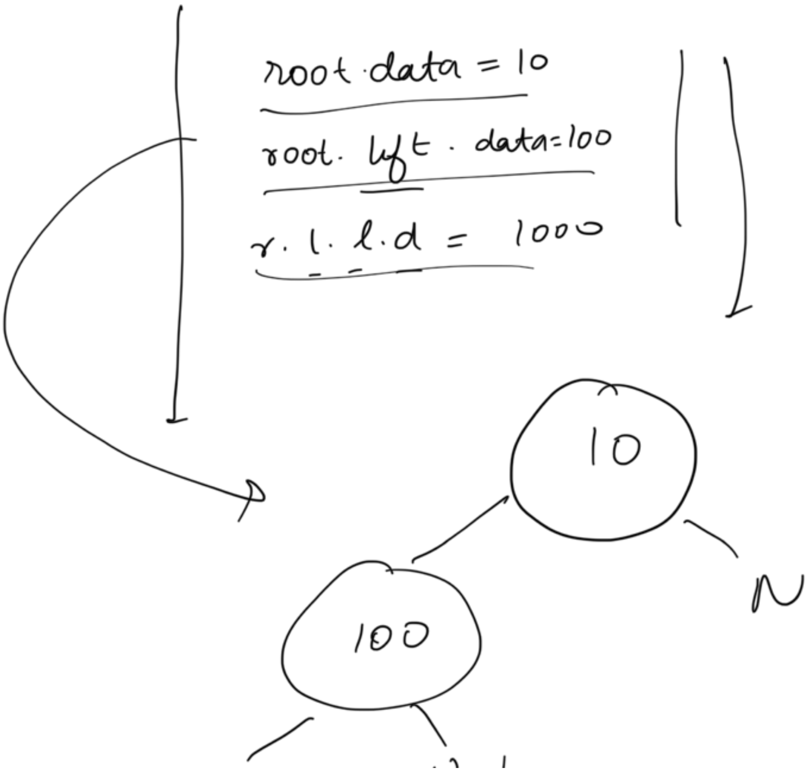
print(root.data) ✓✓



$TC = O(N)$

recur $S.C$ = (depth of Tree)

= longest path from root to leaf



root.data = 10

root.left.data=100

r.l.l.d = 1000