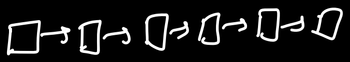
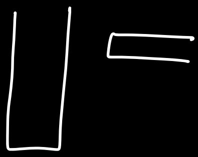
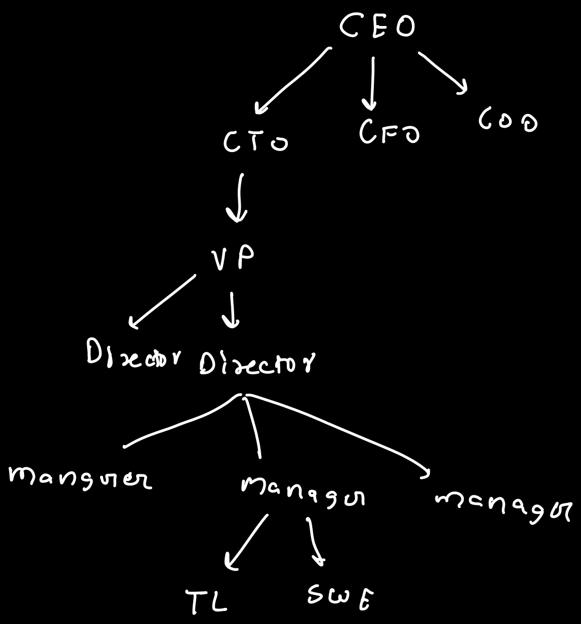
Arrays

LL

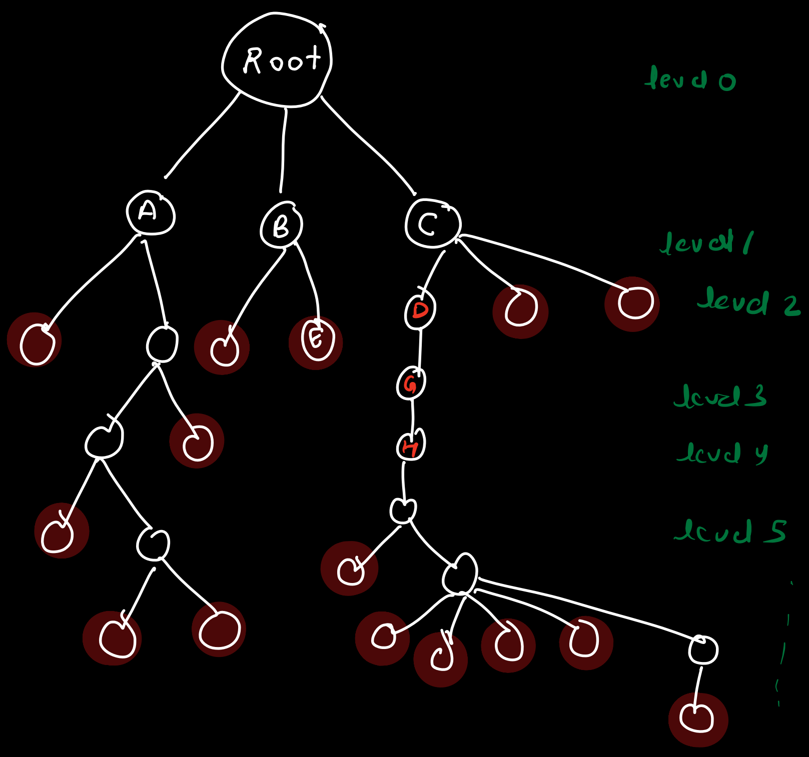




Hierachy Data



Tree → hiearchy data structure



B is parent of E

E is child of B

Naming conventions

1. Parent / child

2. Ancestor / Descendent

3. sibling Nodes → same parent
→ Same level (cousins)

4 Leaf Node : Node without children ●

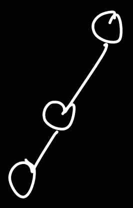
5. Root Node : Node without parent

1. Can only be one root

2. every node can have atmost one parent.

Height (Node) : len of longest path from Node to any of its descentont leaf.

length of path → No of Nodes
→ No of edges

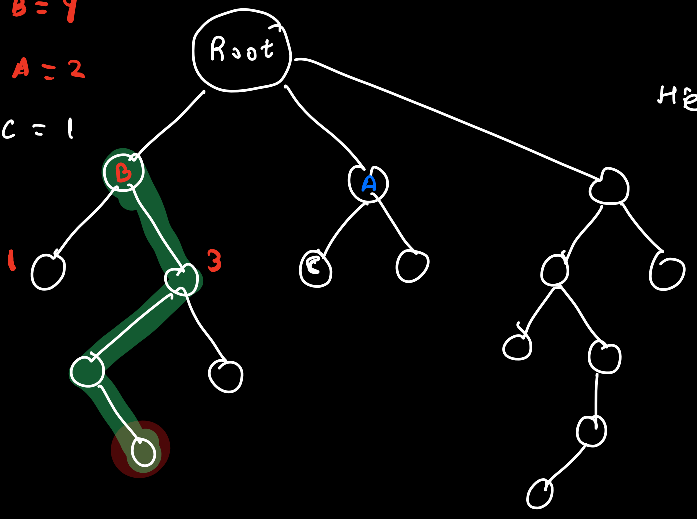


(No of Nodes)

height of B = 4

height of A = 2

heighth of C = 1

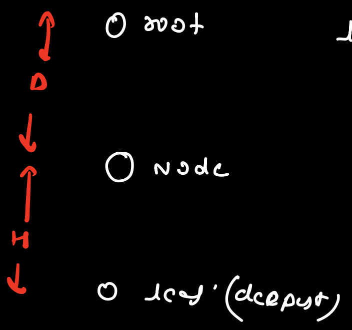


Height (leaf) = 1

height of Node = 1 + max (height of any child)

Height of Tree = Height of root

Depth (Node)

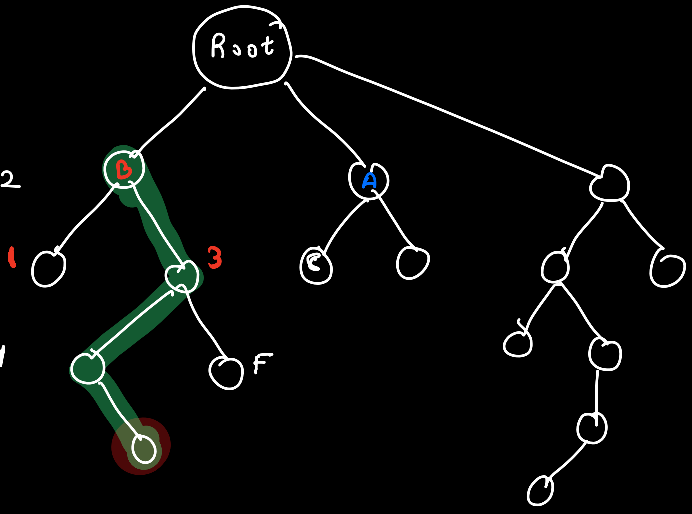


length of root to that Node

Depth of B = 2

Depth of F = 4

Depth (root) = 1



Depth (Node) = K

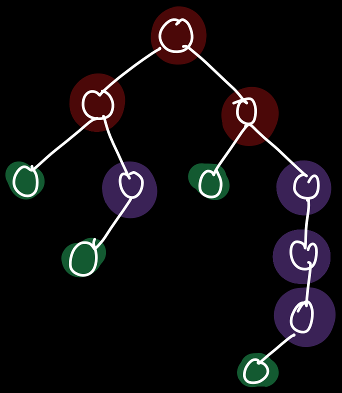
Depth (its child) = K + 1

Binary Tree

↓

Every Node can atmost have 2 child

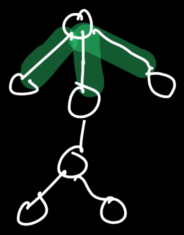
Child = 0, 1, 2



🟩 Leaf nodes 0 Chid

🟪 1 child

🟥 2 chids



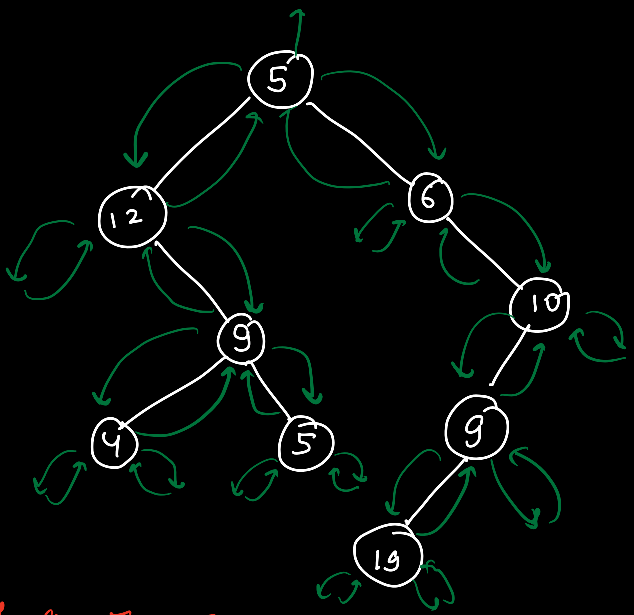
X 3child

```
Class Node
{
        int data
        Node left
        node right

        Node (int x)
        {
            this.data = x
            this.left = NULL
            this.right = NULL
        }
}
```

3 traversals.

1. Pre order NLR
2. Inorder LNR ✓
3. Post order LRN

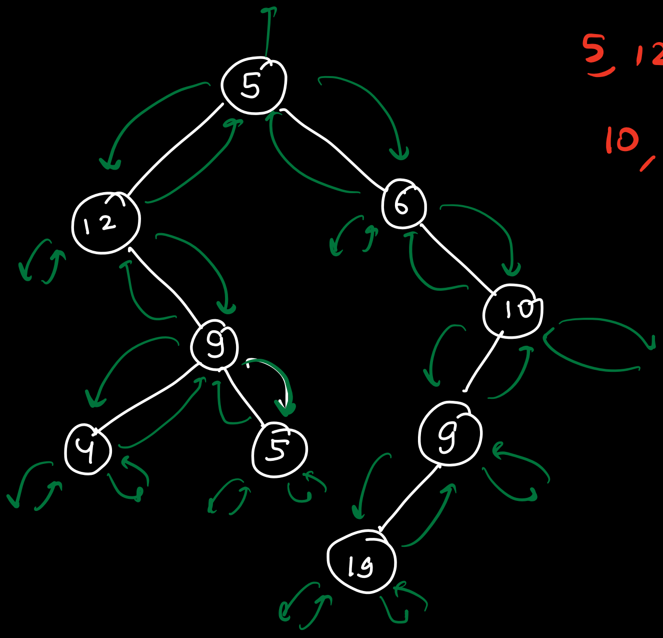
L N R



Inorder

: 12 , 4 , 9 , 5 , 5 , 6 ,19 , 9 , 10 ,

Pre order N L R

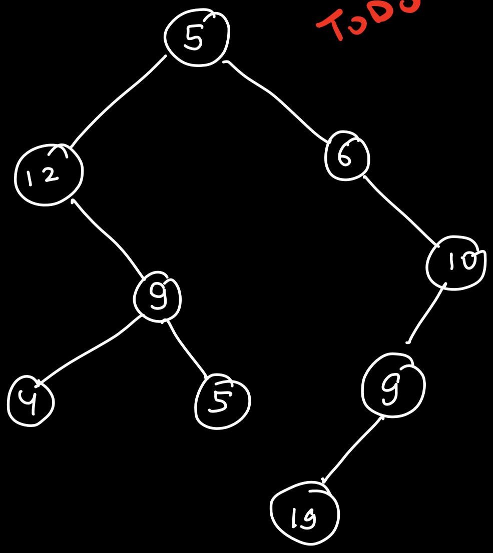


5, 12, 9, 4, 5, 6

10, 9, 19

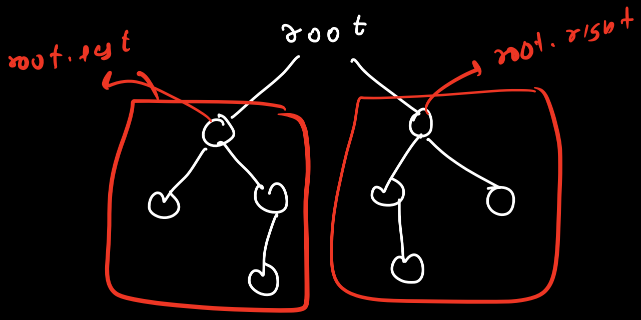
Post order   L R N

4, 5, 9, 12, 19, 9, 10, 6, 5



TODO

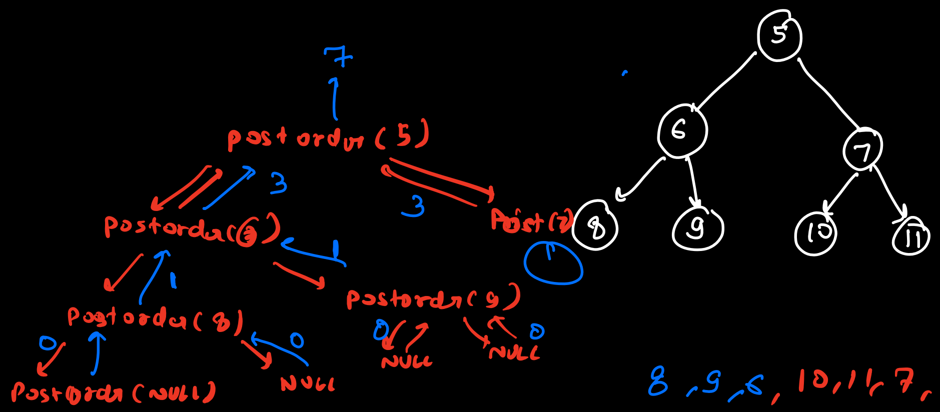
Most used Technique is Tree : Recursion



```
void Inorder ( Node root )
{
        if ( root == NULL)   return

        Inorder ( root. left)
        print( root. data)
        Inorder ( root. right)
}
```

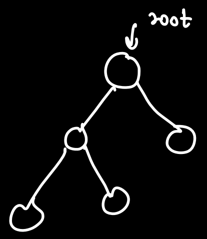
```
void preorder ( Node root )
{
        if ( root == NULL)   return
        print( root. data)
        preorder ( root. left)

        preorder ( root. right)
}
```

```
void postorder ( Node root )
{
        if ( root == NULL)   return

        postorder ( root. left)

        postorder ( root. right)
        print( root. data)
}
```



8 ,9 ,6, 10,11, 7, 5

Q. Given root node, Find no of nodes in the tree.



ans = 5.

return the sz of root

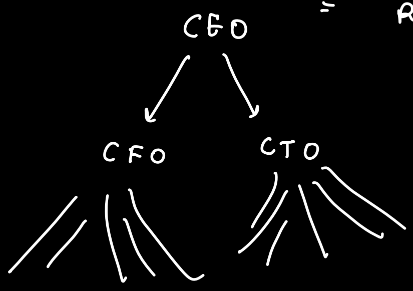
```
int sz( Node root )
{
    if( root == NULL )   return 0

    int L = sz( root.left )
    int R = sz( root.right )

    return L+R+1
}
```

Postorder.

Both correct

```
if( root.left == NULL && root.right == NULL )
    return 1
```

Reports of (CEO)



= Reports of (CFO)

+

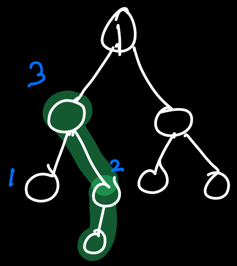
Reports of (CTO)

+

1

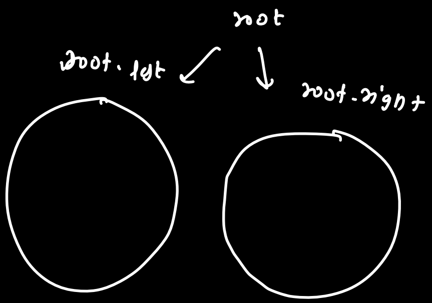
Q. Find height of Binary Tree. (Given).

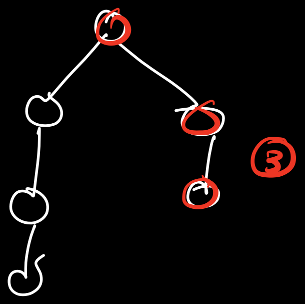
return the hg of root
↑
```
int hg ( Node root )
{
    If ( root == NULL )   return 0

    int L = hg ( root.left )
    int R = hg ( root.right )

    return 1 + max ( L , R )
}
```



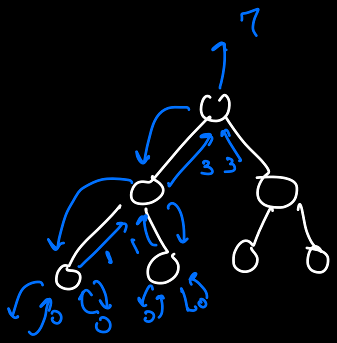
Q. Find sum of all nodes. data

return the Sum of root Including root.
↑
```
int sum ( Node root )
{
    If ( root == NULL )   return 0

    int L = sum ( root.left )
    int R = sum ( root.right )

    return L + R + root.data
}
```

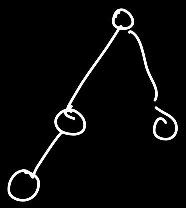




4



sz of tree.



```
sz of tre
↑
int   sz( )
{
    if( root = NULL)   return 0
      L = sz( left)
     R = sz( right)
      L+R+1
}
```