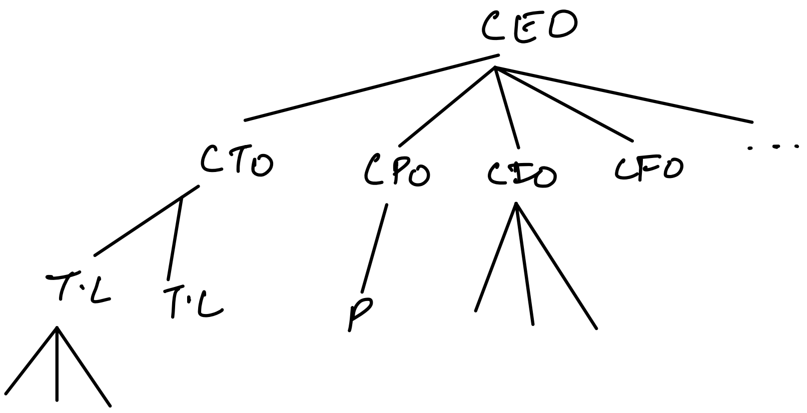
Welcome ☺

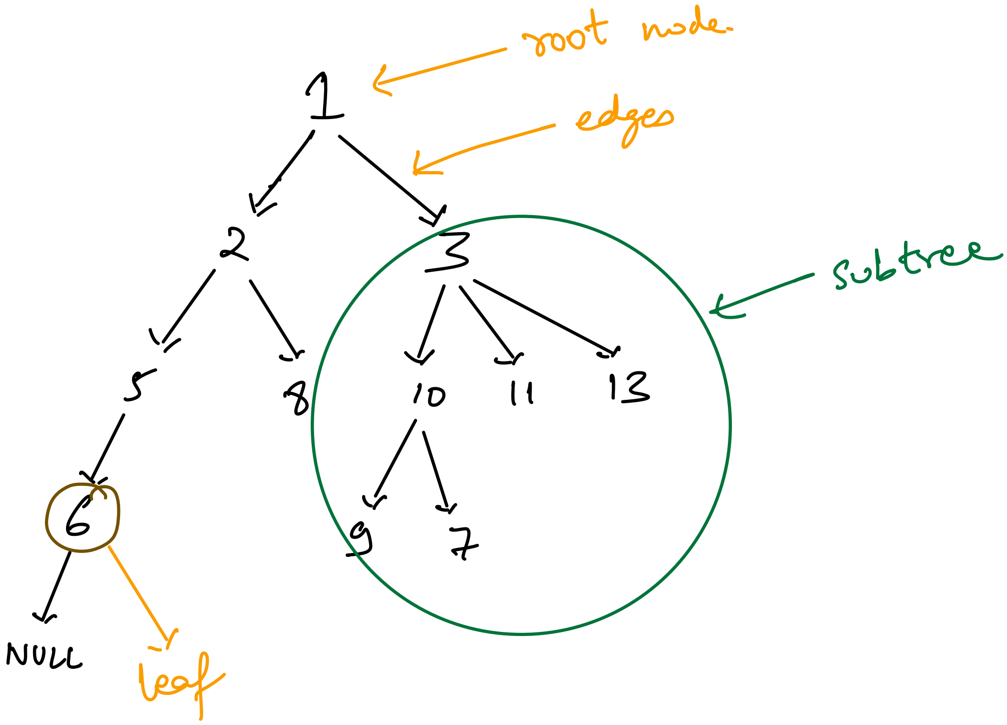
Agenda : Tree D.S
Terms
Traversal
1-2 ques"s.

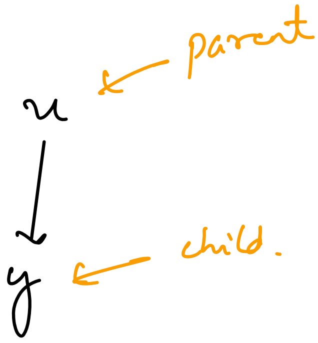
Tree D.S

→ non linear D.S / heirarchical D.S

↳ traverse all the data in single run.







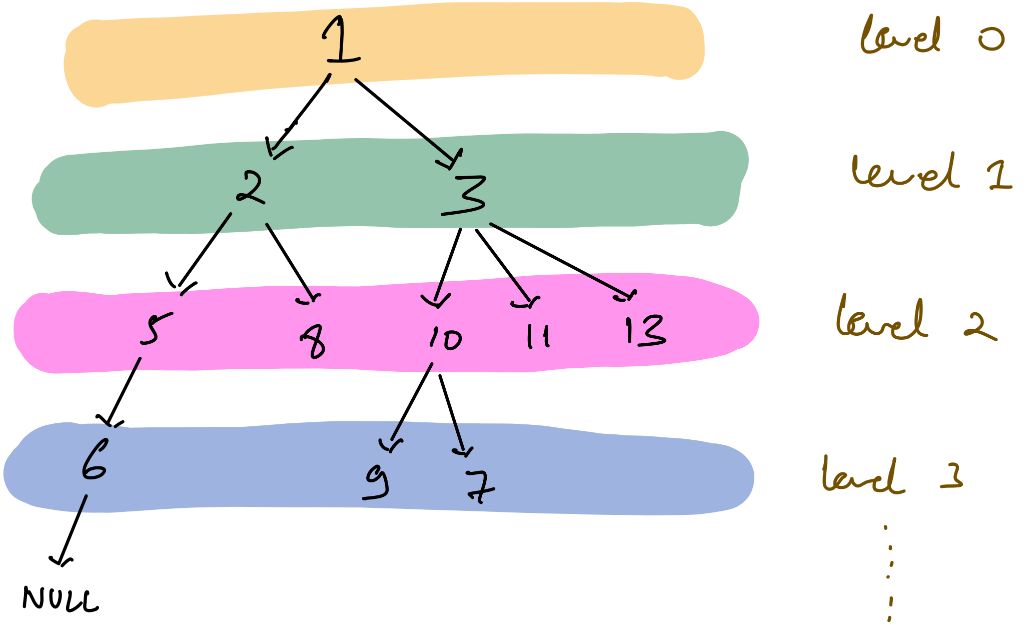
leaf node → Nodes without any child

Siblings → Nodes which have same parent

Depth → # edges to travel from root node to node X

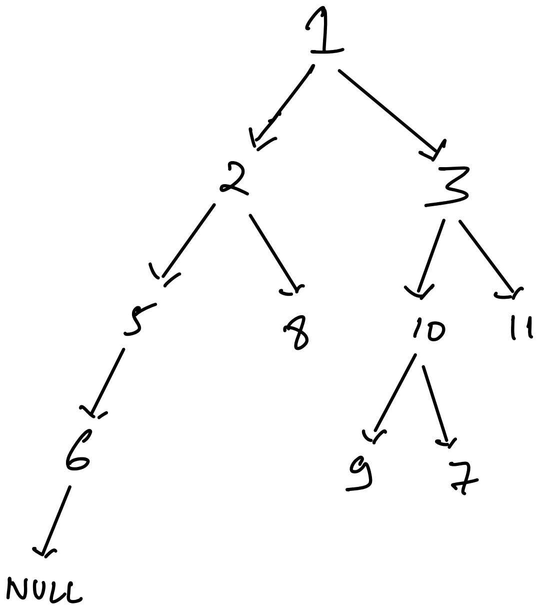
Height → # edges to travel from node X to farthest leaf node.

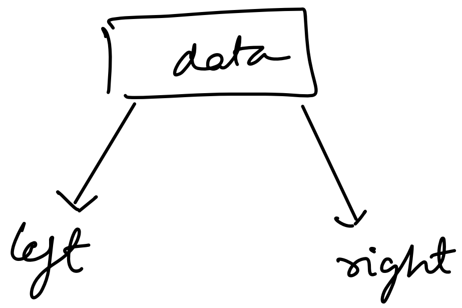
Levels



Binary Tree

⇒ All the nodes can have atmost 2 children





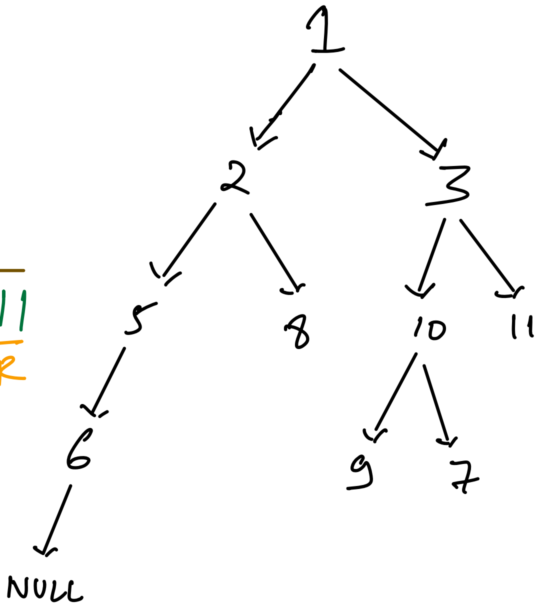
```
class Node
{
    int data ;
    Node* left ;
    Node* right;
}
```

Traversal

1.	Preorder	Node	Left	Right
2.	Inorder	left	Node	Right
3.	Postorder	left	Right	Node.
4.	Level Order			

1. Pre-order traversal

Node	left				Right					
1	2	5	6	8	3	10	9	7	11	
	N	L		R	N	L			R	



Code

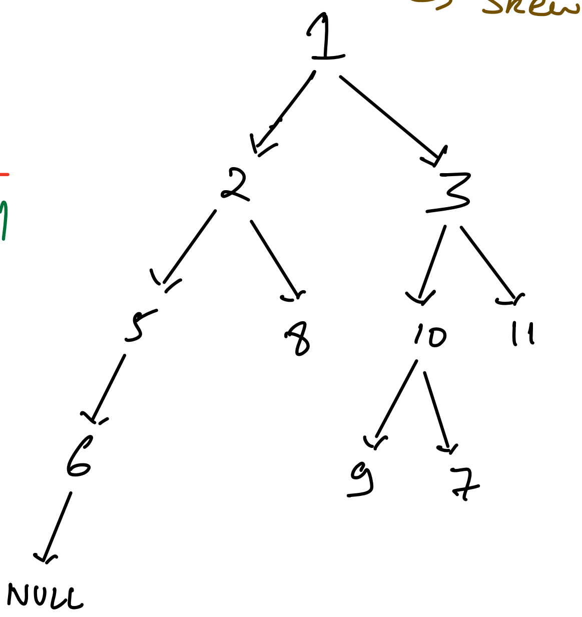
```
void   preOrder ( root) // prints in pre-order.
{
       if ( ! root )    return ;
       print ( root. data )   // Node
       preOrder ( root.left )  // left subtree
       preOrder ( root. right // Right subtree.
}
```

T.C ⇒ $O(N)$

S.C ⇒ $O(H)$ → height of tree.
→ $O(N)$
↳ skewed tree.

2. Inorder traversal   LNR

Left				N	Right				
6	5	2	8	1	9	10	7	3	11

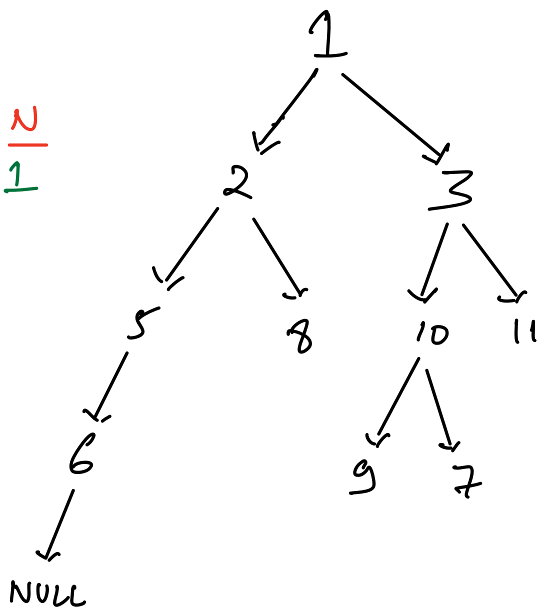


```
void   InOrder ( root) // prints in pre-order.
{
    if ( ! root )    return ;
     InOrder ( root.left )  // left subtree
    print ( root. data )   // Node
    InOrder ( root. right // Right subtree.
}
```

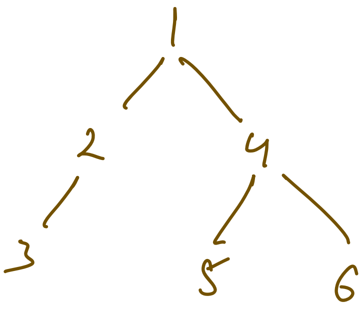
3. Post Order Traversal.

left Right N

6 5 8 2 9 7 10 11 3 1



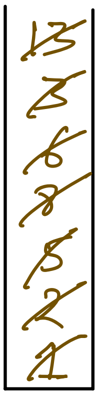
```
void PostOrder ( root) // prints in pre-order.
{
    if ( ! root )   return ;
    PostOrder ( root. left) // left subtree
    PostOrder ( root. right // Right subtree.
    print ( root. data)   // Node
}
```



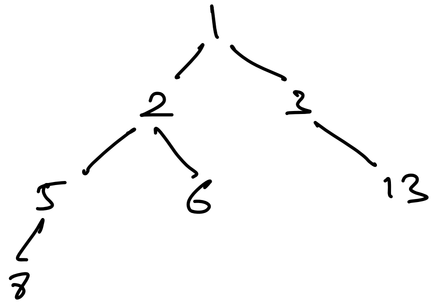
LNR

Inorder → 3 2 1 5 4 6

Q Write iterative code for inorder traversal.



8 5 2 6 1 3 13



curr →

Code

```
curr = root

while ( curr != NULL || !st. isEmpty())
{
    if ( curr != NULL)
    {
        st.push ( curr)
        curr = curr.left
    }
    else
    {
        curr = st.pop()
        print ( curr.data)
        curr = curr.right
    }
}
```

T.C ⇒ O(N)

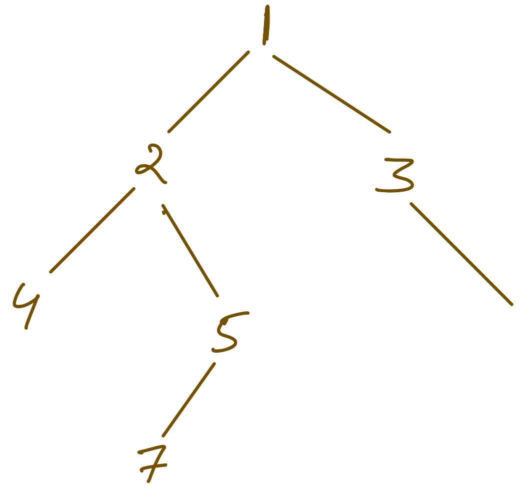
S.C ⇒ O(H)

H.W Iterative for pre order

Q Construct a binary tree using inorder & post order traversal. (distinct values)

LNR
Inorder ⇒ 4 2 7 5 | 1 | 3 6
(L: 4 2 7 5; N: 1; R: 3 6)

PostOrder ⇒ 4 7 5 2 | 6 3 | 1 ← root node.
LRN
(L: 4 7 5 2; R: 6 3; N: 1)
(within L: L 4, R 7 5, N 2)



```
Node tree ( in[] , post [ ], st_in, end_in , st_p , end_p)
{
    If ( st_in > end_in ) return NULL
    root = new Node ( post [ end_p ] )
    //2. Find root node in inorder traversal.
    idx = getIndex ( post [end_p] , st_in , end_in )
              → ~~O(N)~~ → O(1) hashmap < value , index >
    //3. Figure out elements in left & right subtree
    ~~cnt_L = idx - st_in~~
    cnt_R = end_in - idx
    root.left = tree ( in[], post[], st_in, idx-1, end_p - cnt_R - 1)
    root.right = tree ( in[], post[], idx+1 , end_in , endp - 1 )
    return root
}
```