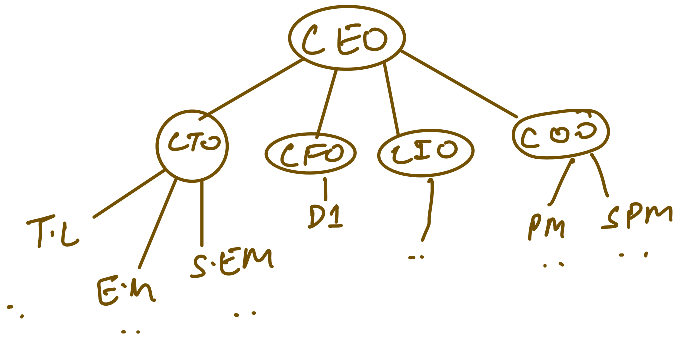
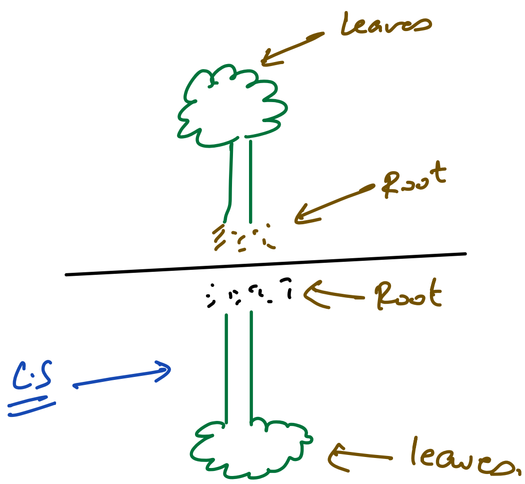
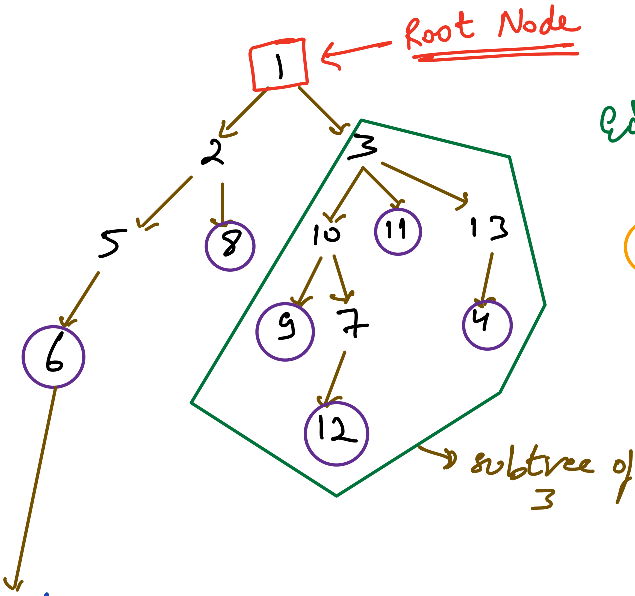
Welcome ☺

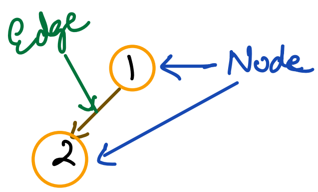
Agenda : Tree D.S
- Nomenclatures
- Traversal
- 1-2

Hierarchical D.S









x ← parent node

↓

y ← child node.

Leaf → Nodes without any children

Subtree → For any node x, all the nodes that can be travelled from x are part of subtree of x

Q Can a root node be a leaf node ?

→ Yes → ◯ single node root / leaf node

Depth → # edges to travel from root node to node X is depth of X

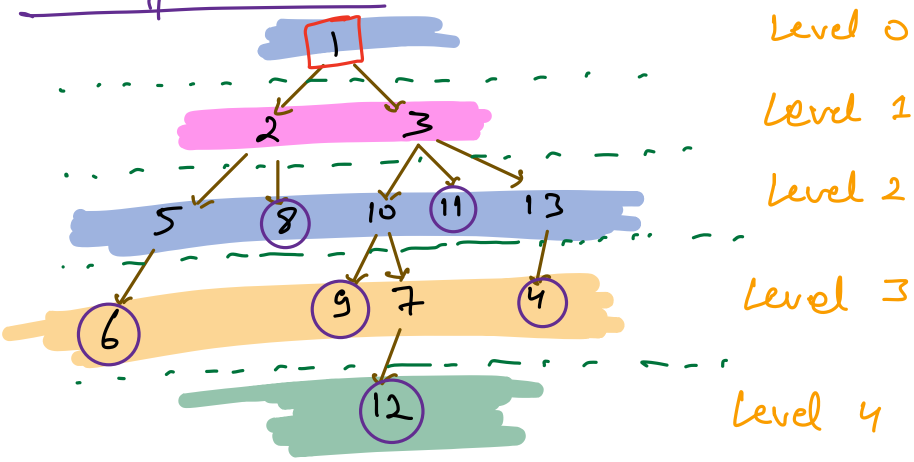
→ depth of root node = 0

Height → # edges to travel from node X to reach farthest leaf node is height of X

height(leaf) = 0

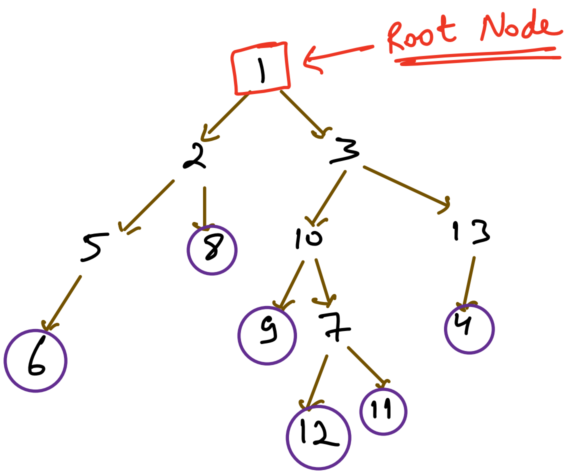
height(tree) = height(root)

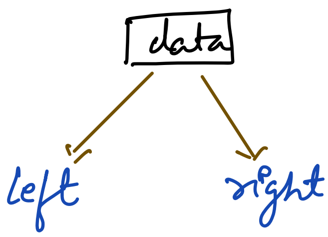
Levels of a Tree



Binary Tree

All the nodes can have ATMOST 2 children.





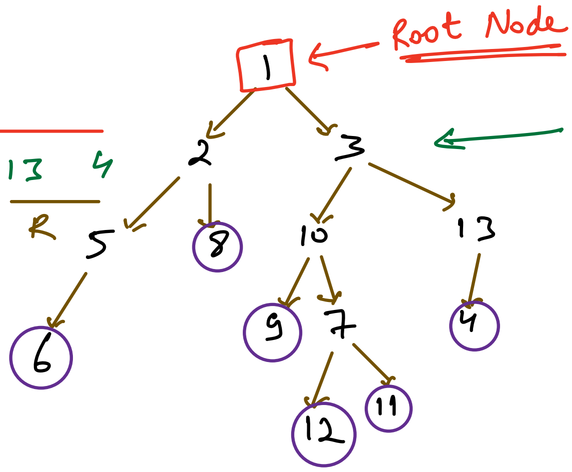

```
class Node {
  int Data;
  Node left, right
}
```

Traversals in Binary Tree.

1. Preorder — Node Left Right
2. Inorder — Left Node Right
3. Postorder — Left Right Node
4. Level order → next class

1) PreOrder Traversal

Node	Left				Right							
1	2	5	6	8	3	10	9	7	12	11	13	4
	N	L		R	N	L					R	



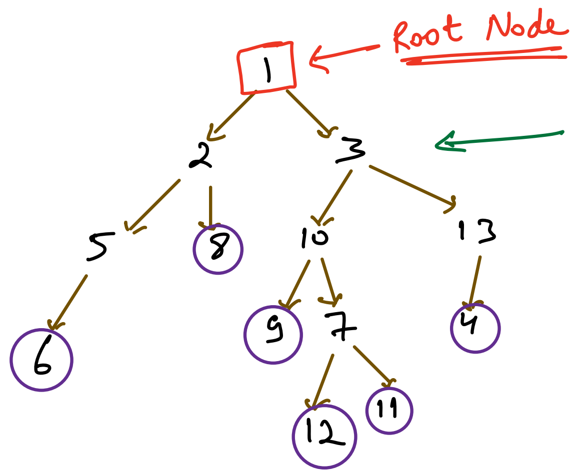
→ It is a depth first traversal

Code

```
void preOrder( root) {
    if( ! root )      return
    print ( root.data ) → Node
    preOrder ( root. left) → Left
    PreOrder ( root. right) → Right
}
```

Inorder Traversal   LNR

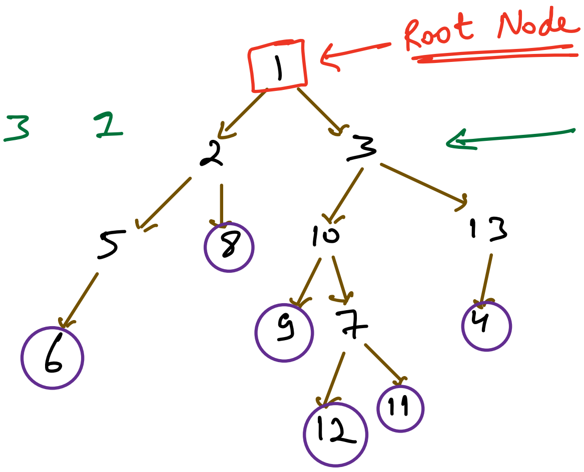
Left				Node	Right							
6	5	2	8	1	9	10	12	7	11	3	4	13
L	N	R			L					N	R	




```
void InOrder ( root) {
    if( ! root )      return
    InOrder ( root. left) → Left
    print ( root.data ) → Node
    InOrder ( root. right) → Right
}
```

Post Order Traversal L R N

6 5 8 2 9 12 11 7 10 4 13 3 1




```
void PostOrder( root) {
    if( !root)   return
    PostOrder( root.left)  → Left
    PostOrder( root.right) → Right
    print( root.data)      → Node
}
```

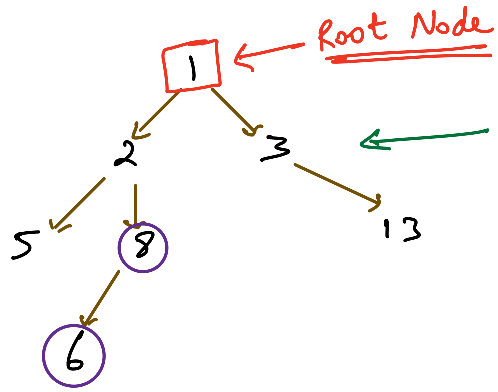
Q → Write iterative code of inorder traversal. (L N R)

~~13~~ ~~3~~ ~~6~~ ~~8~~ ~~5~~ ~~2~~ ~~1~~

curr ~~1 2 5 null 5 null 2 8 6~~
~~null 6 null 8 null 2 3~~
~~null 3 13 null 13~~ null

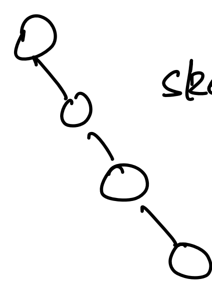
o/p → 5 2 6 8 1 3 13

$\underbrace{5\ 2\ 6\ 8}_{L}\ \underbrace{1}_{N}\ \underbrace{3\ 13}_{R}$




```
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)



skewed

H.W ⇒ Iterative code of preOrder.

Q. Construct binary tree from inorder & postorder (distinct values)

Inorder → $\underbrace{4\ 2\ 7\ 5}_{L}$ 1 $\underbrace{3\ 6}_{R}$

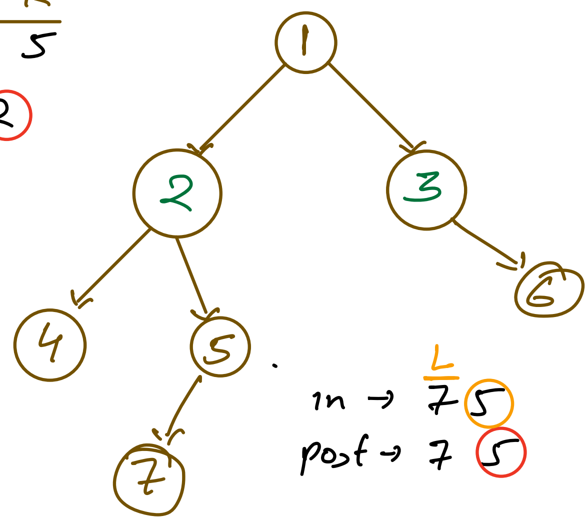
Post Order → 4 7 5 2 6 3 1 ← Root node.

in → $\underbrace{4}_{L}$ 2 $\underbrace{7\ 5}_{R}$

post → $\underbrace{4}_{L}$ $\underbrace{7\ 5}_{R}$ 2

in → 3 $\underbrace{6}_{R}$

post → 6 3



in 4

post 4

in → $\underbrace{7}_{L}$ 5

post → 7 5

Code

```
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] ) // Root Node.
    //2. Find root node in inorder traversal
    idx = getIndex( post[end_p], in, st_in, end_in )
              O(N) → use Hashmap instead ⇒ <value, index>
    // 3. Figure out elements on left subtree & right subtree.
    ~~cnt_L = idx - st_in~~
    cnt_R = end_in - idx                     T.C → O(N+N)
                                             S.C → O(N+N+H)
    root.left = tree( in, post, st_in, idx-1, end_p-cnt_R-1)
    root.right = tree( in, post, idx+1, end_in, end_p-1)
    return root
}
```