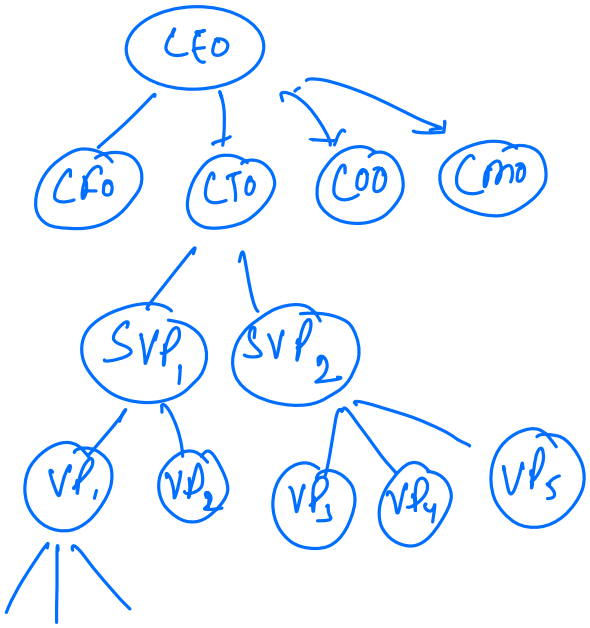
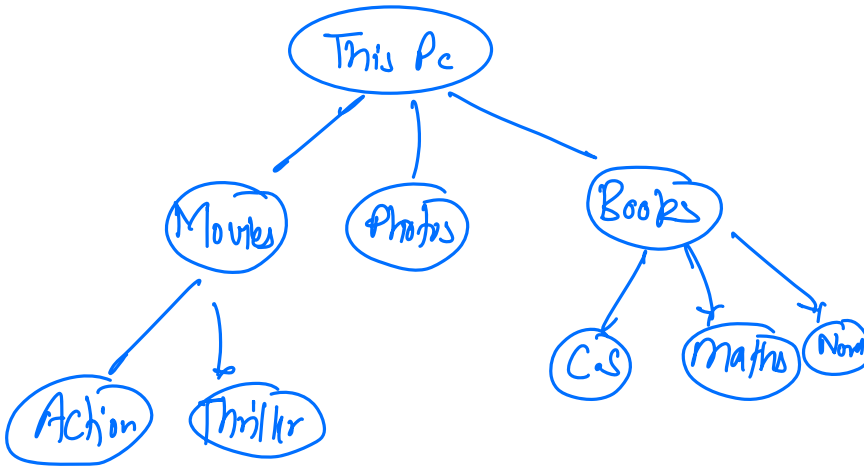
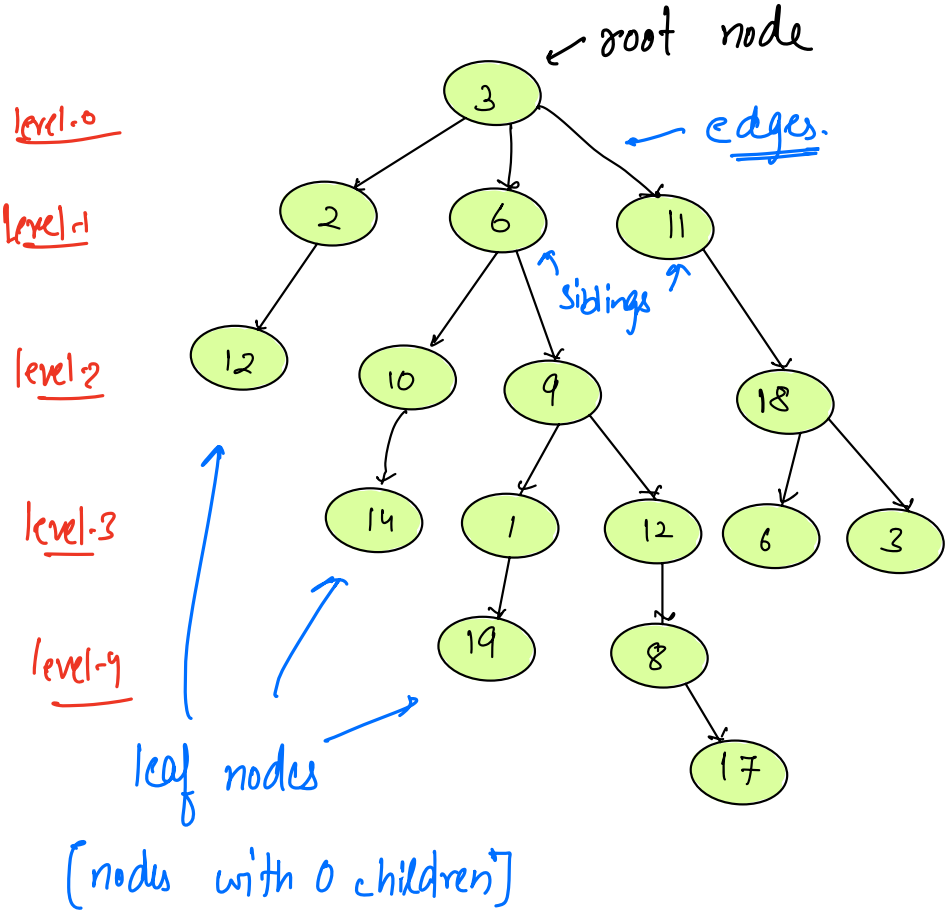
Tree → data structure which stores non-linear info.




12 is descendent of 3

3 is ancestor of 12.

Every node is having exactly one parent node except the root node.

If total no. of nodes are N

no. of edges → Exactly N-1

Height of a node →

Maximum distance [in terms of edges] between node and its descendent leaf node.

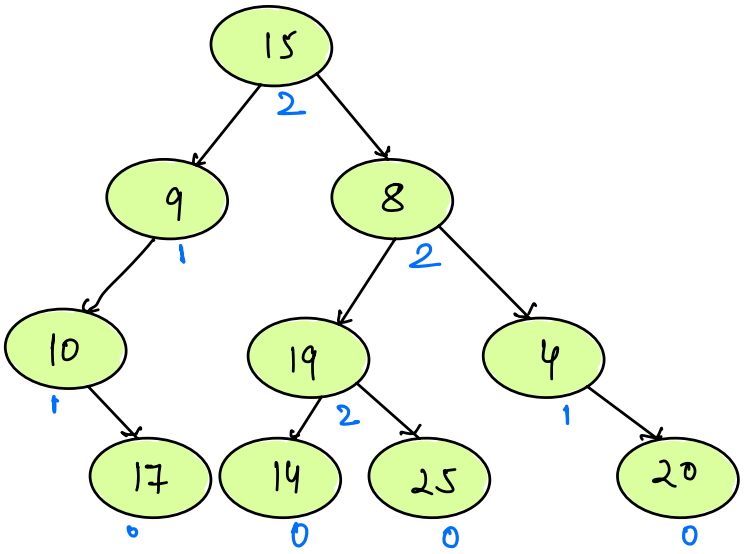
ht(9) → 3.

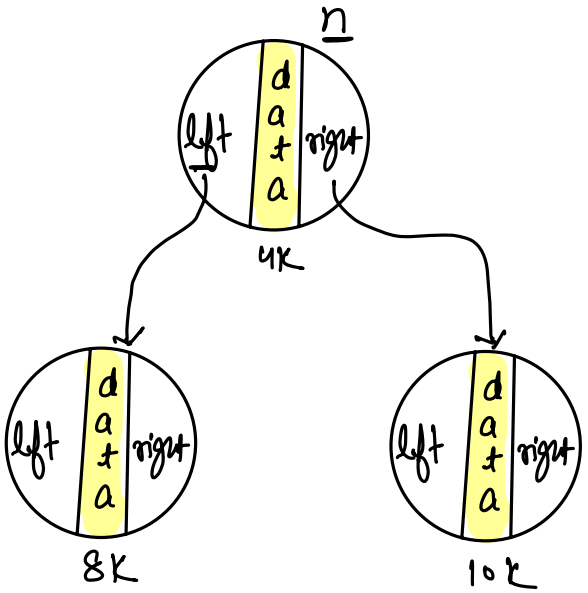
Height of a tree → Height of the root node.

Depth of a node → no. of edges node is away from the root node.

**Binary Tree** → Every node can have **maximum two** children.

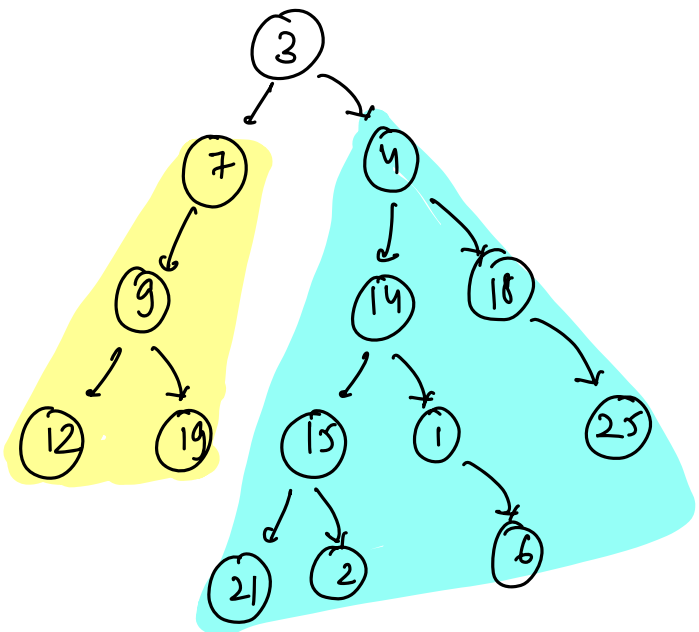
[0, 1, 2]




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

**Sub-tree.**



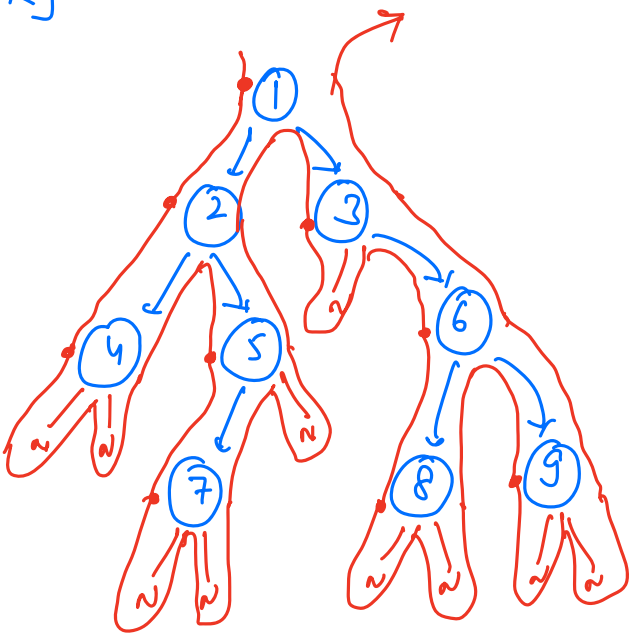

[node + all its descendents]

# Binary Tree Traversals

## ① Pre-order Traversal [N L R]

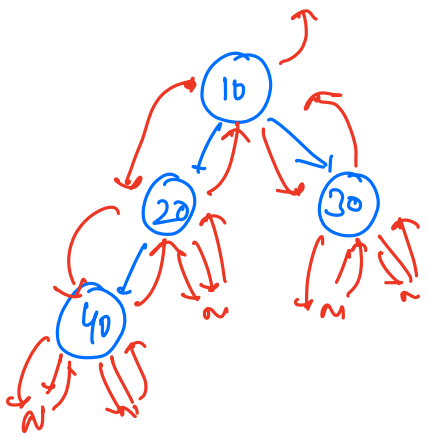
o/p → [1, 2, 4, 5, 7, 3, 6, 8, 9]

- 1 → Node
- 2, 4, 5, 7 → pre-order of l.s.t
- 3, 6, 8, 9 → pre-order of r.s.t




pre-order of tree = print(root) + pre-order of left sub-tree + pre-order of right sub-tree

```
void pre-Order(Node root){
    if (root == NULL) { return }
    print(root.value);
    pre-order(root.left);
    pre-order(root.right);
}
```



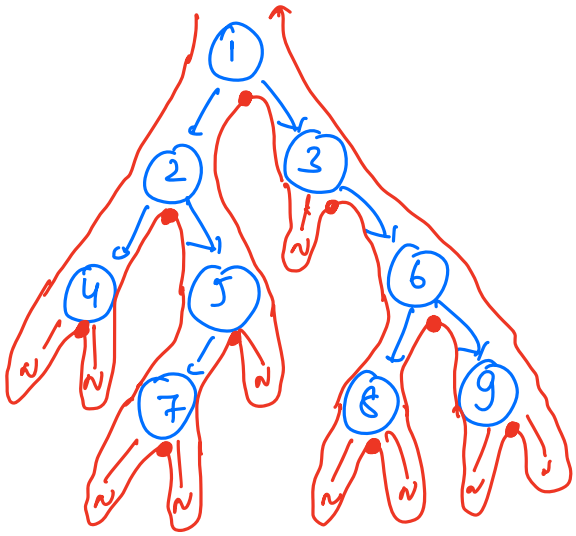

T.C → O(N)
S.C → O(Ht. of tree)

o/p → [10, 20, 40, 30]

## ② In-order Traversal (left node right)

in[7-[4, 2, 7, 5, 1, 3, 8, 6, 9]

- inorder of left sub-tree
- print root
- inorder of right sub-tree

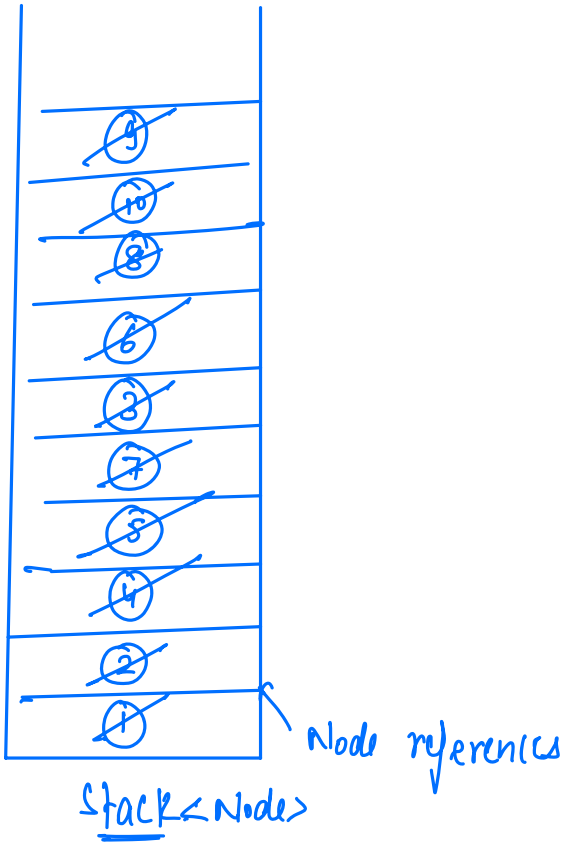


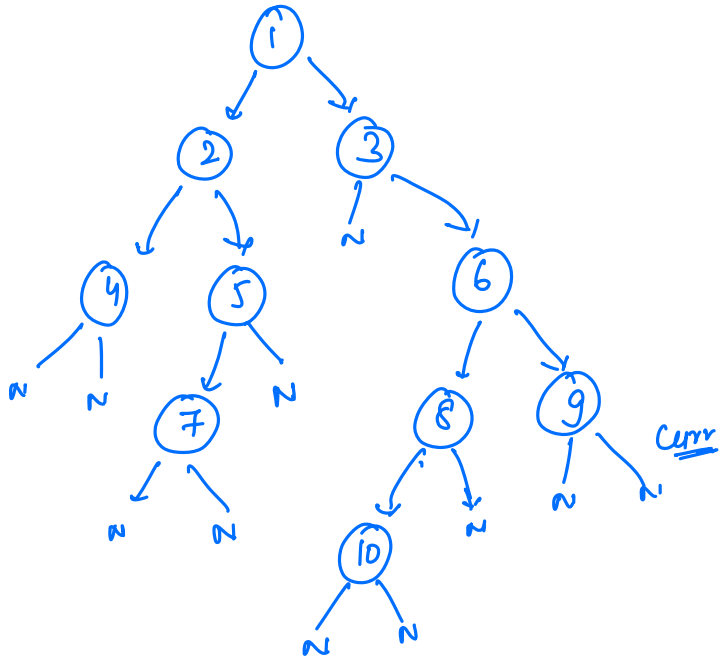

```
void inorder (Node root){
    if (root == NULL) {return}

    inorder (root.left);
    print(root.value);
    inorder(root.right);
}
```

T.C → O(N)
S.C → O(Ht. of tree)

## POST-ORDER Traversal (left right node)

# Iterative In-Order Traversal




o/p. → 4, 2, 7, 5, 1, 3, 10, 8, 6, 9

# code. →

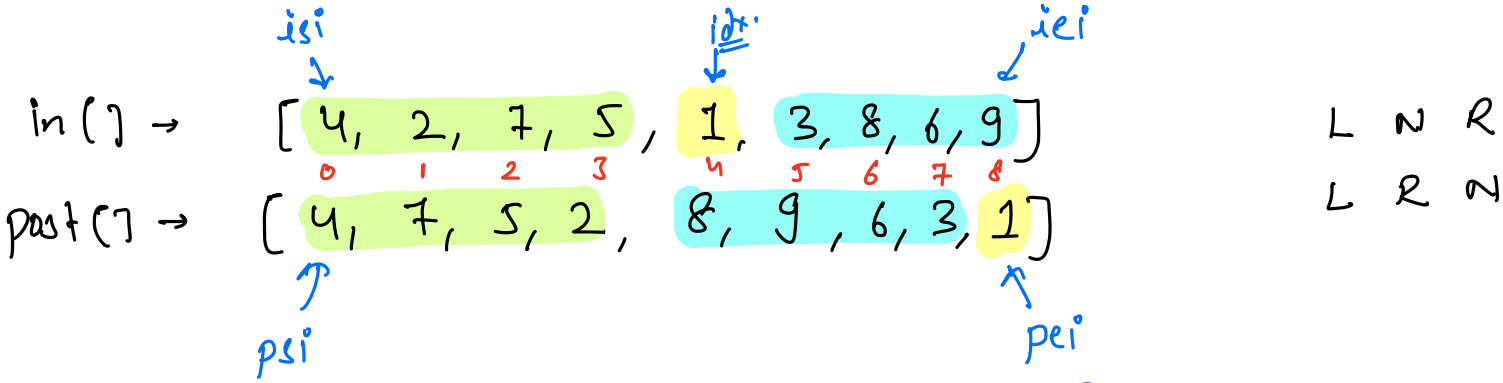
```
Stack<Node> st;
Node curr = root;
while( st.size() != 0 || curr != NULL ){
    while( curr != NULL){
        st.push(curr);
        curr = curr.left;
    }
    curr = st.pop();
    print( curr.value);
    curr = curr.right;
}
```

T.C → O(N)
S.C → O(Ht. of tree)

# pre-order, post-order - todo

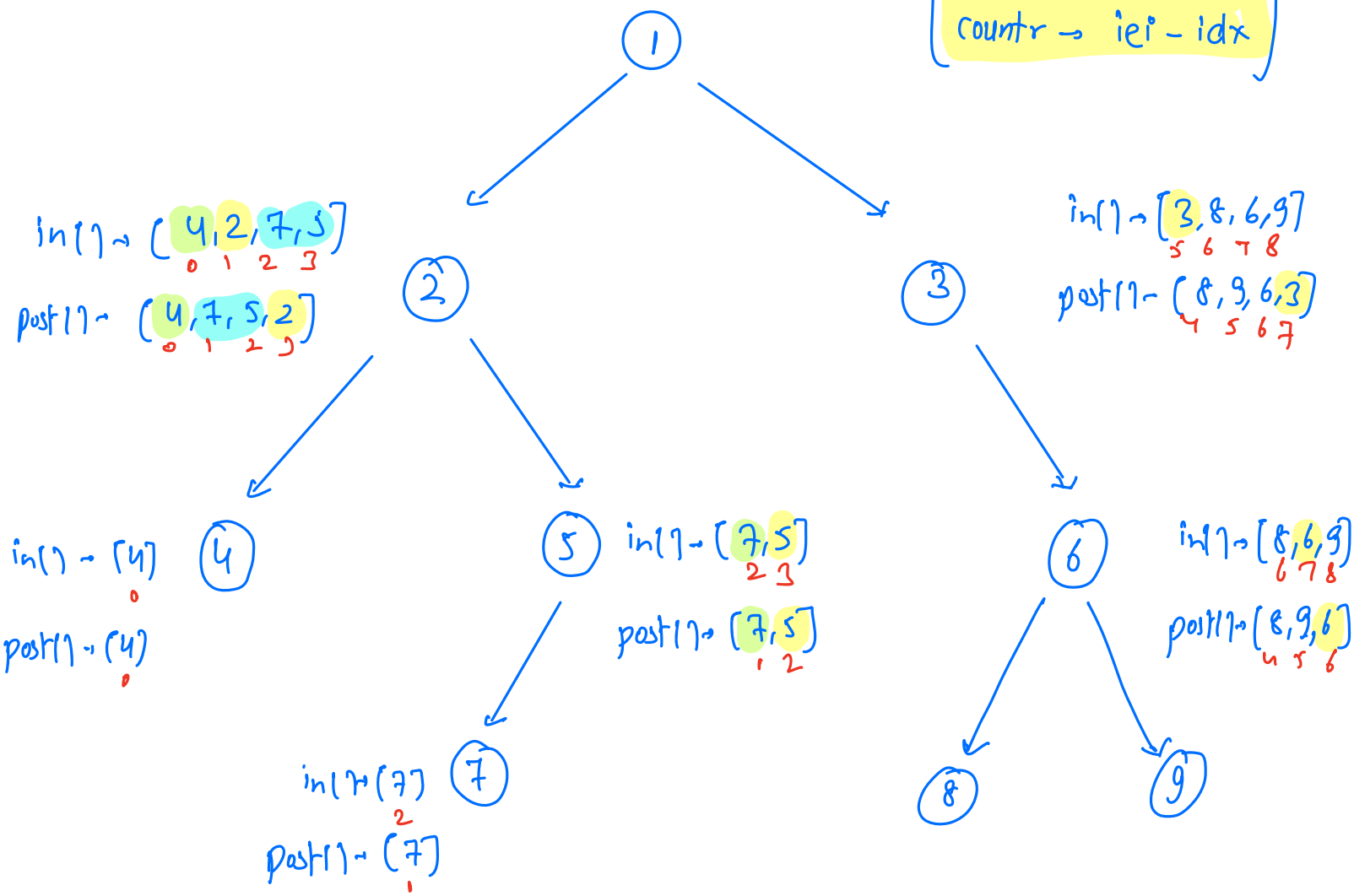
Q: Construct the binary tree from inorder & postorder.

Given two arrays in[], post[]. (unique elements)




countL → idx - isi

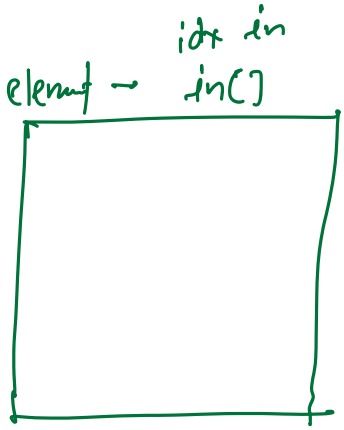
countR → iei - idx



# code →

```
                                  0      N-1     0      N-1
                                  ↑       ↑      ↑       ↑
Node construct( in[], post[], isi, iei, psi, pei){
    if(isi > iei){ return NULL};
    Node root = new Node(post[pei]);
    //find the position of post[pei] in in[]
     int idx = -1;
     for(i = isi; i <= iei; i++){
        if(in[i] == post[pei]){
            idx = i , break;
        }
     }
    countL = idx - isi;
   root.left = construct(in[], post[], isi, idx-1, psi, psi+countL-1);
   root.right = construct(in[], post[], idx+1, iei, psi+countL, pei-1);
   return root;
}
```

Hashmap.




T.C → O(N)
L.C → O(N)

pre[] →
in[] →

---

(pre[] →
post[] →) ✗ not valid

Level order Traversal → Trees - 2.

Left View / Right View

Top View / Bottom View

Vertical Order Traversal

4th October

Plase Plase Plase utilise this time efficiently!!