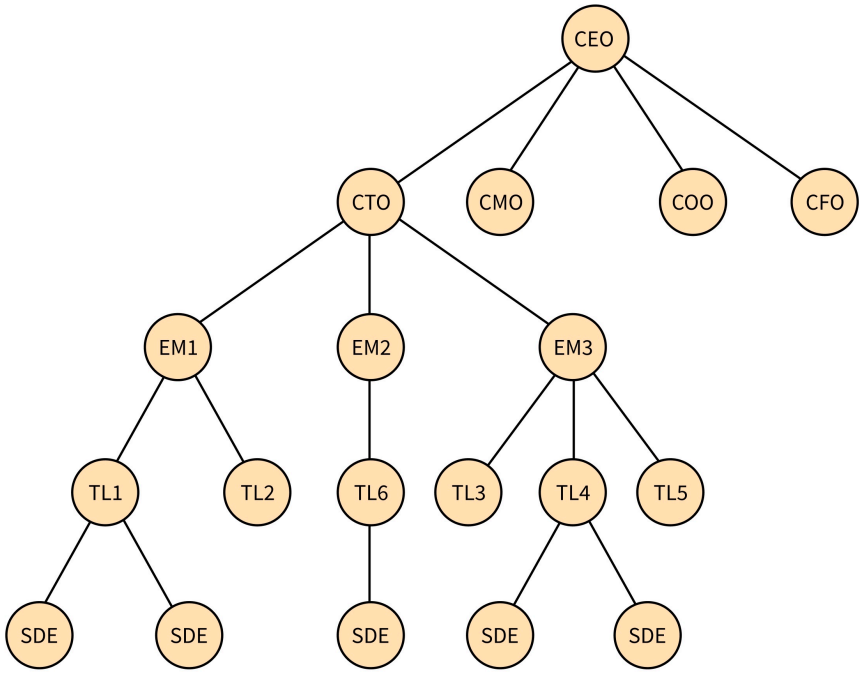
SCALER

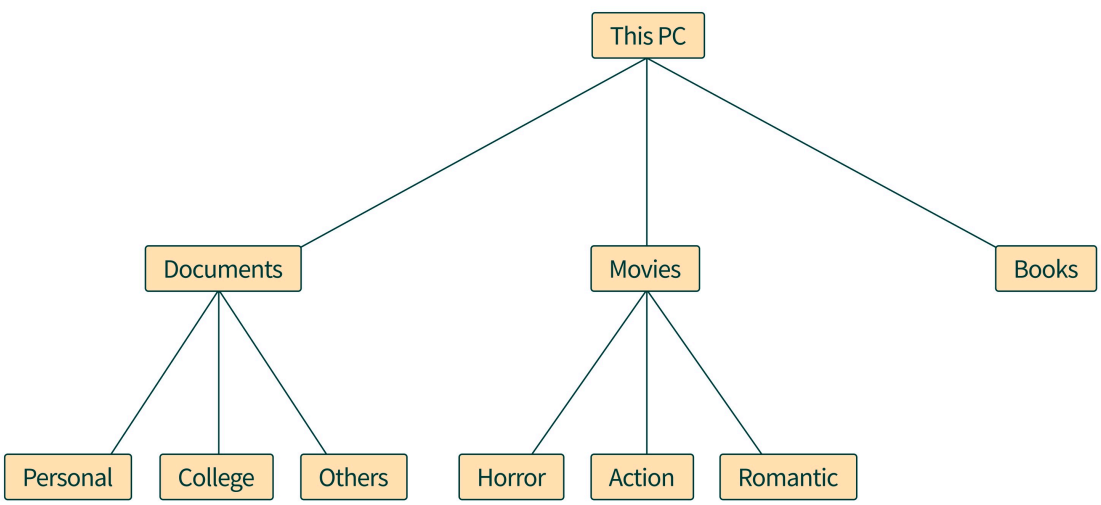
Trees - 1

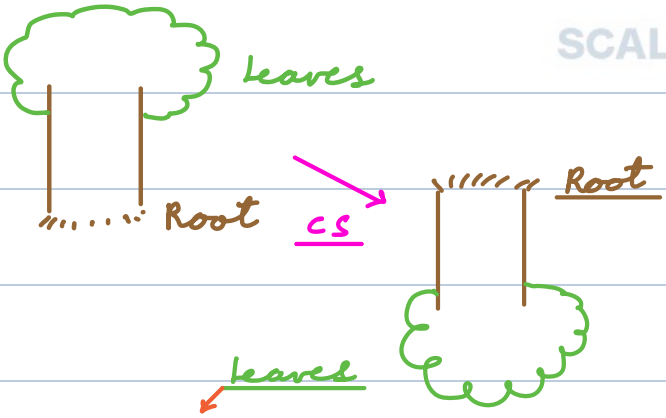
TABLE OF CONTENTS

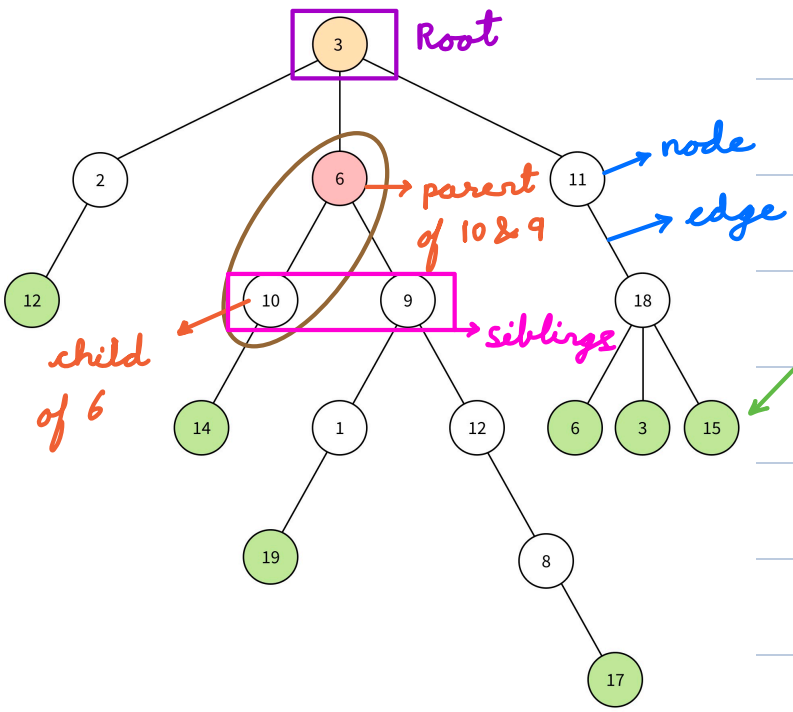
Trees

Hierarchical Data









Nodes without children.

Height of node → count of edges (distance) to travel from current node to farthest leaf.

leaf

height (18) = 1

height (9) = 3

height (leaf) = 0

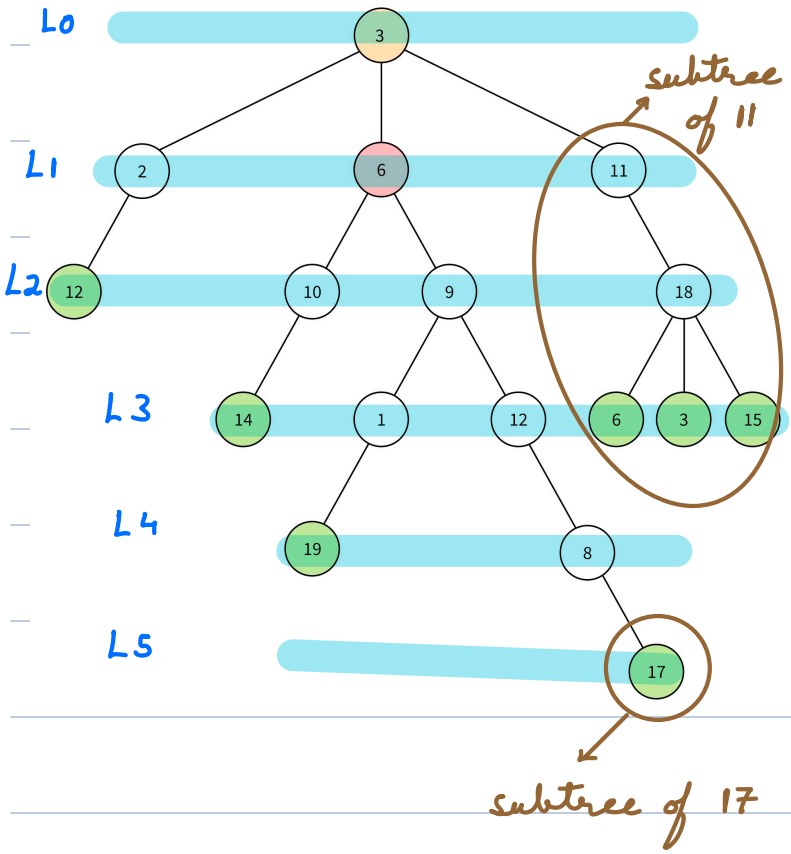
v.imp → Height of tree = Height (root) = 5

Depth / Level of node → count of edges (distance) to travel from root to reach the current node.

root

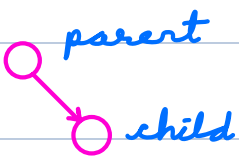
depth (9) = 2

depth (14) = 3



Subtree of a node x →

All the nodes we can travel from node x are part of subtree of x.

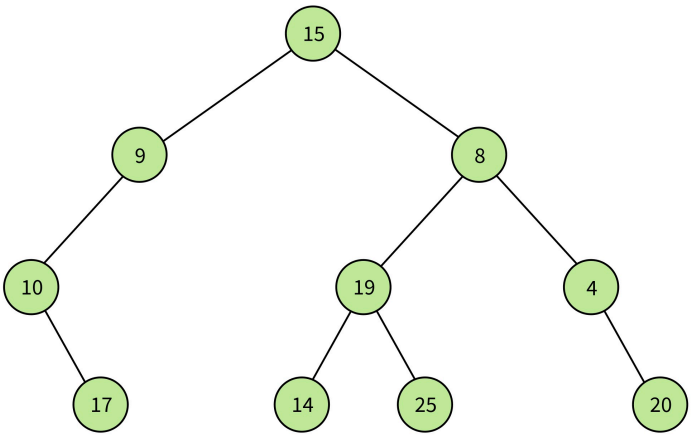


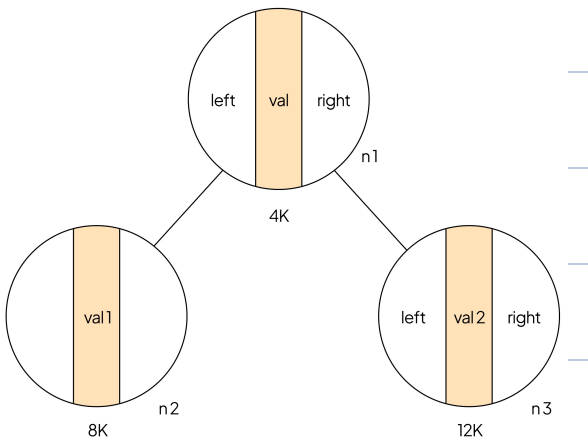
subtree (root) = complete tree

Binary Tree

∀ nodes, max # children = 2

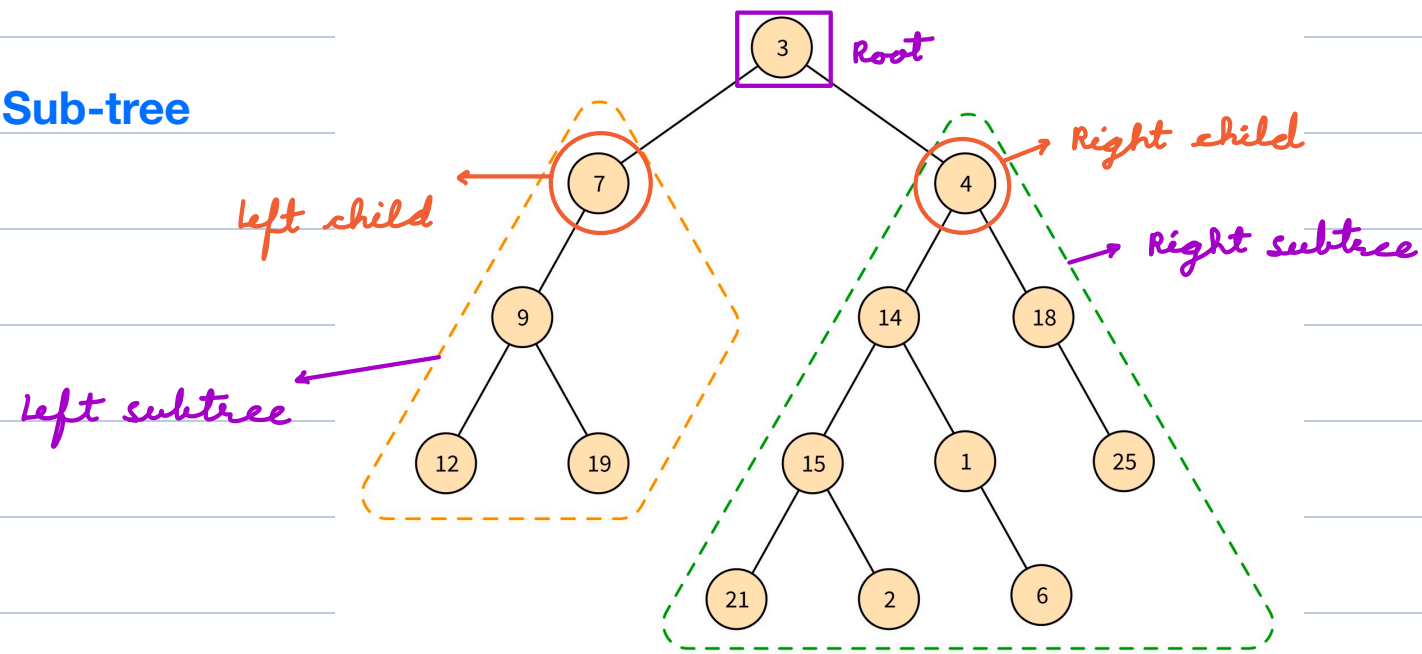
children = {0, 1, 2}






```
class TreeNode {
    int val;
    TreeNode left, right;
}
```

Sub-tree



Preorder N L R

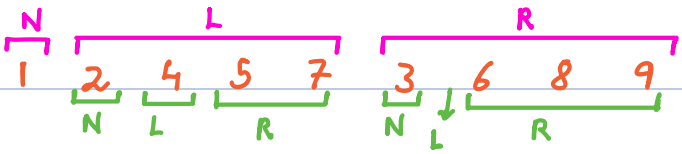
Inorder L N R

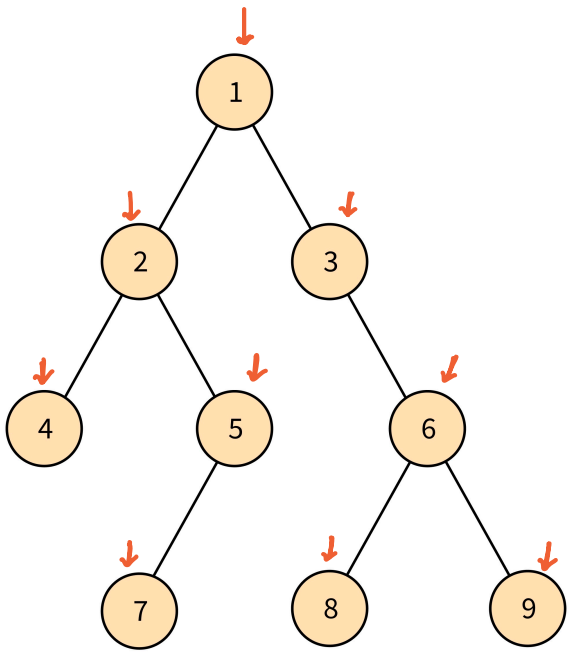
Postorder L R N

Levelorder → Next class

Binary Tree Traversal

1. Pre-order Traversal N L R

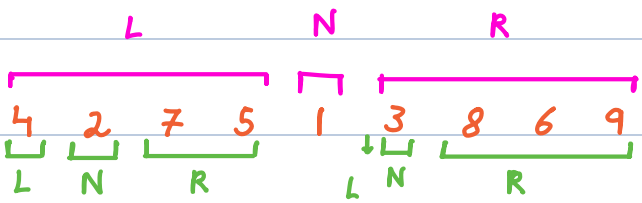


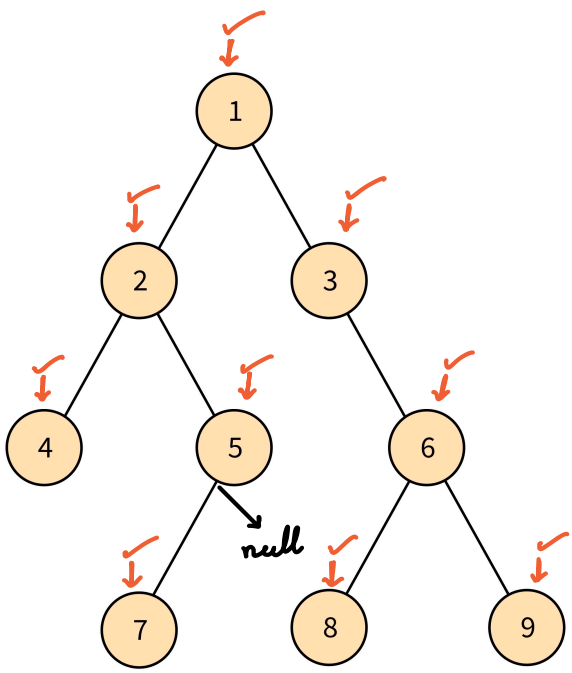



```
void preorder (root) {
    if (root == null) return
    print (root.val)          // Node
    preorder (root.left)      // Left
    preorder (root.right)     // Right
}
```

TC = O(N) SC = O(H)

2. In-order Traversal L N R

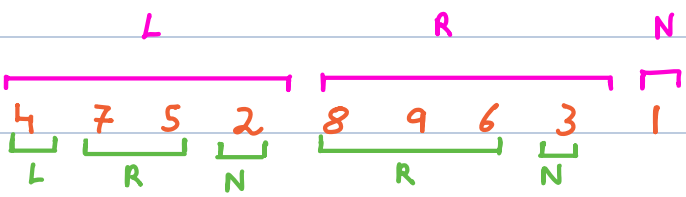


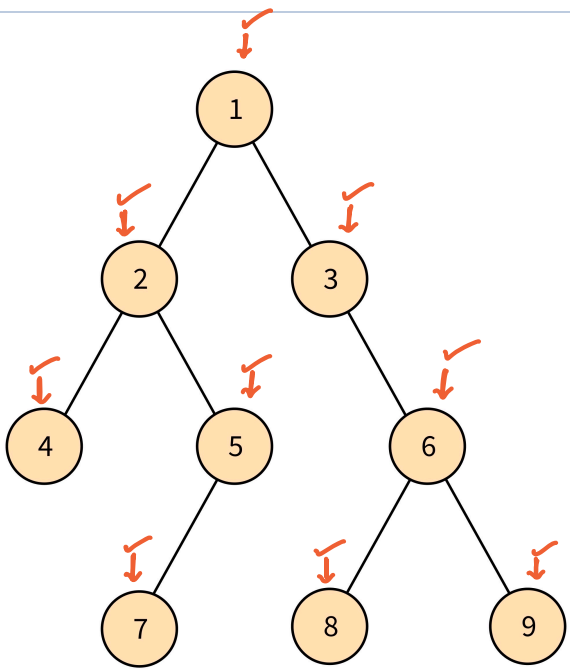


```
void inorder (root) {
    if (root == null) return
    inorder (root.left)  // Left
    print (root.val)     // Node
    inorder (root.right) // Right
}
```

TC = O(N) SC = O(H)

2. Post-order Traversal L R N





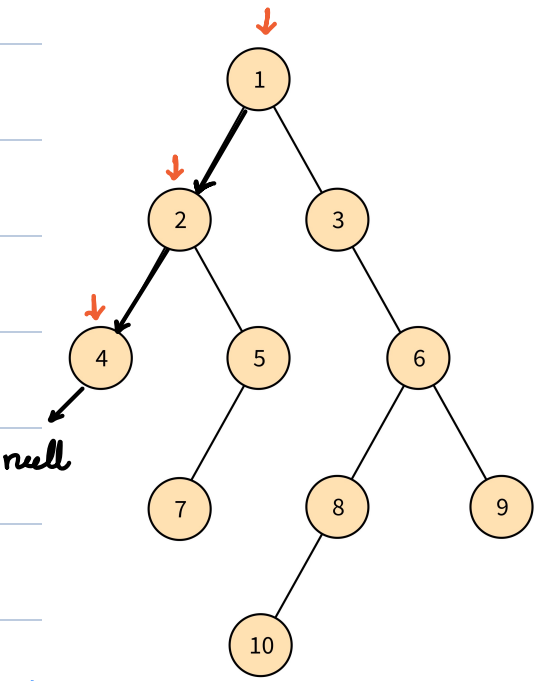

```
void postorder (root) {
    if (root == null) return
    postorder (root.left)  // Left
    postorder (root.right) // Right
    print (root.val)       // Node
}
```

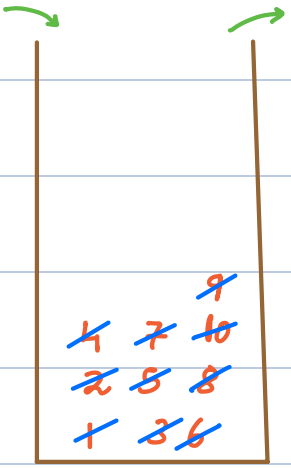
TC = O(N) SC = O(H)

Iterative In-order Traversal

L N R

DS to use → Stack





1) st.push(cur)

cur = cur.left

2) cur = st.pop()

print(cur.val)

cur = cur.right

cur = root

o/p → 4 2 7 5 1 3 10 8 6 9

4 null 2 5 7 null 7 null 5 null 1 3 null 3 6 8 10 null 10 null 8 null 6 9 null 9 null

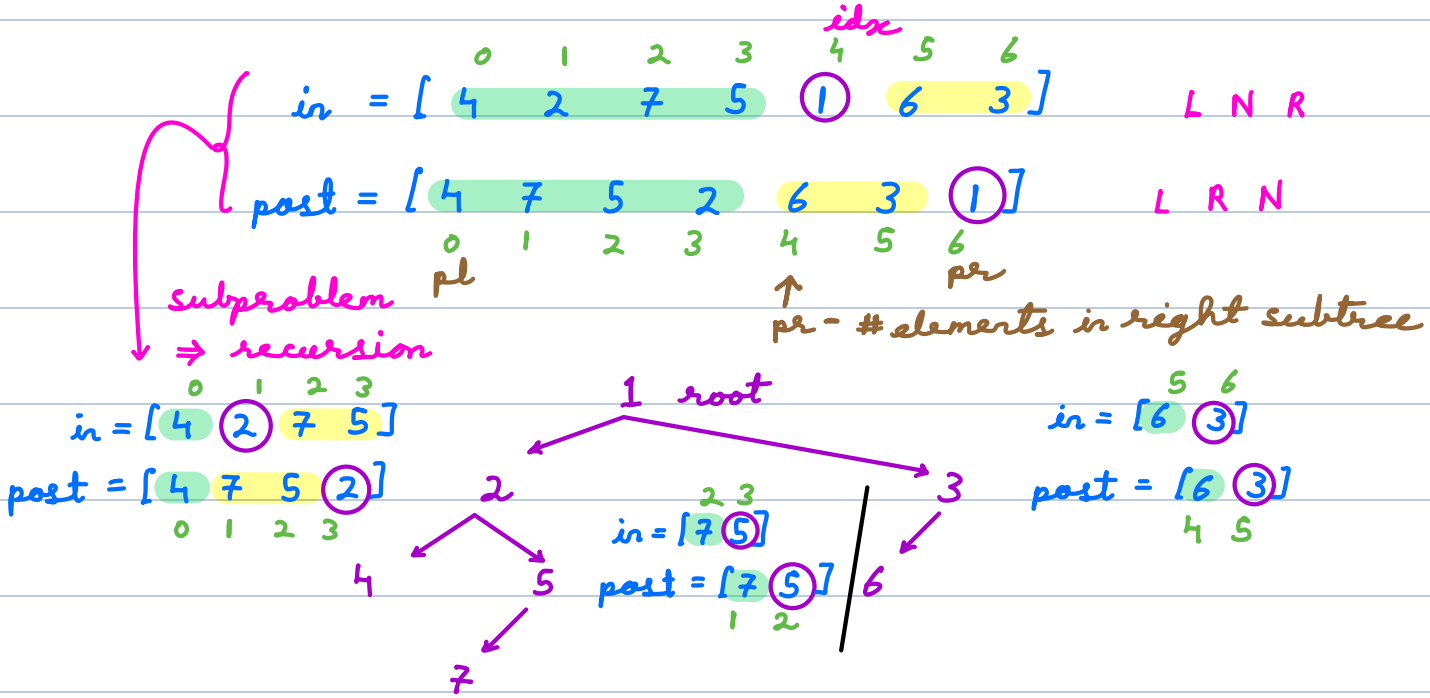
```
cur = root
while (cur != null || !st.isEmpty()) {
    if (cur != null) { st.push(cur)
                       cur = cur.left }
    else { cur = st.pop()
           print(cur.val)
           cur = cur.right }
}
```

TC = O(N) SC = O(H)

H.W → Try iterative code for preorder & postorder. ✓

< Question > : Construct the binary tree from in-order and post-order.

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




```
TreeNode build ( in[], il , ir , post[], pl, pr) {
    if ( ir < il ) return null // empty array
    root = new TreeNode ( post[pr] )
    idx = index ( in , root.val ) // iterate <A[i], i> → hashmap ✓
    // in → left   l = il   r = idx - 1
    //      right  l = idx+1   r = ir
    cnt_r = ir - idx                    // R-L+1  [L R]
    // post → left   l = pl   r = pr - cnt_r - 1
    //        right  l = pr-cnt_r   r = pr - 1
    root.left = build (in, il, idx-1, post, pr-cnt_r-1)
    root.right = build (in, idx+1, ir, post, pr-1)
```

return root

}

TC = O(N) SC = O(N)