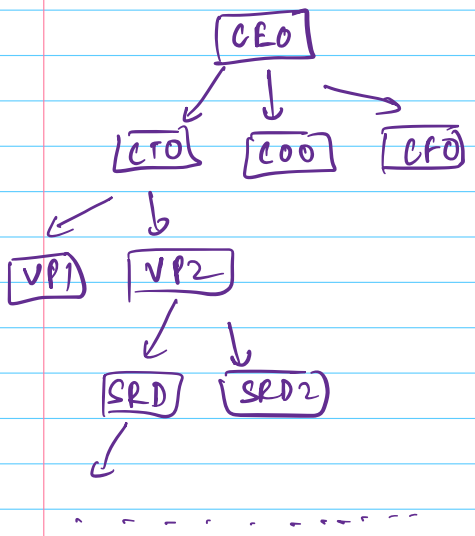
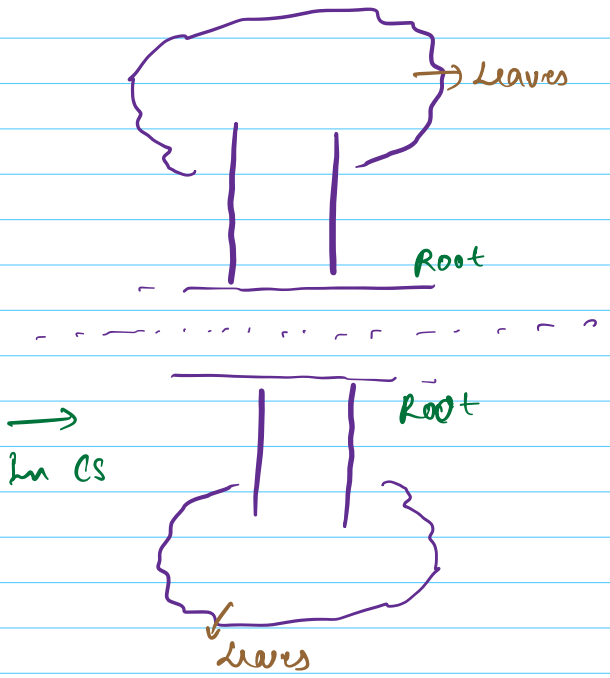
28/7/2023

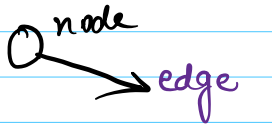
Trees - 1

Hierarchical data structure



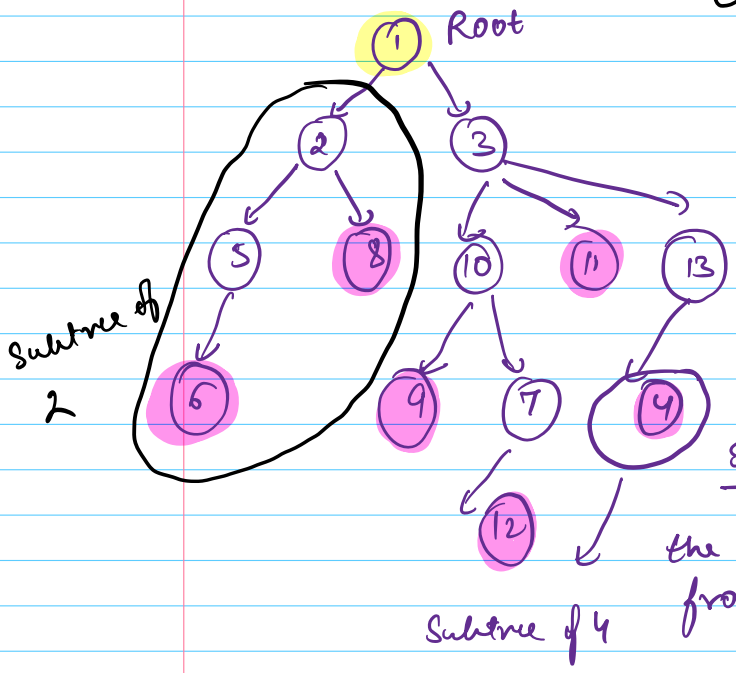


Trees



x → parent of y

y → child of x

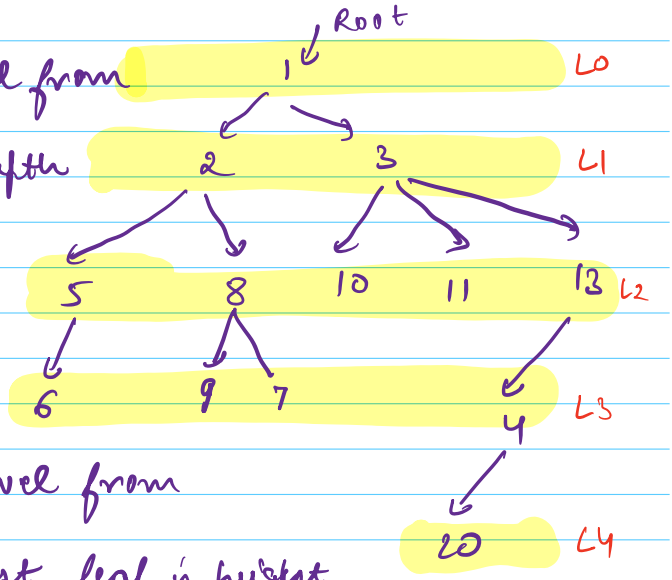


Leaf → Nodes without any children

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

Can root become a leaf node?

Yes, in case of single node. (1 node)



Depth → # edges to travel from root to reach x is depth of x.

depth (root) = 0

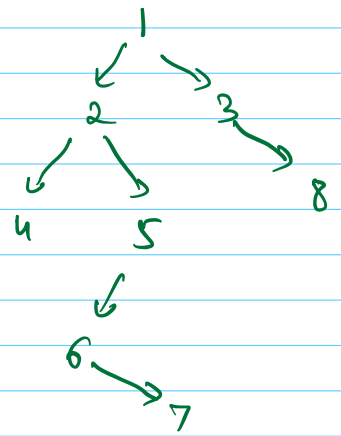
Height → # edges to travel from x to reach the farthest leaf is height of x.

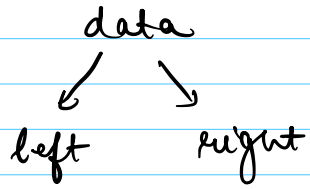
Height of tree = height (root) = 4

height (leaf) = 0

Binary tree :- Max # children for any node = 2.

{0, 1, 2}






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

Traversal in Binary tree

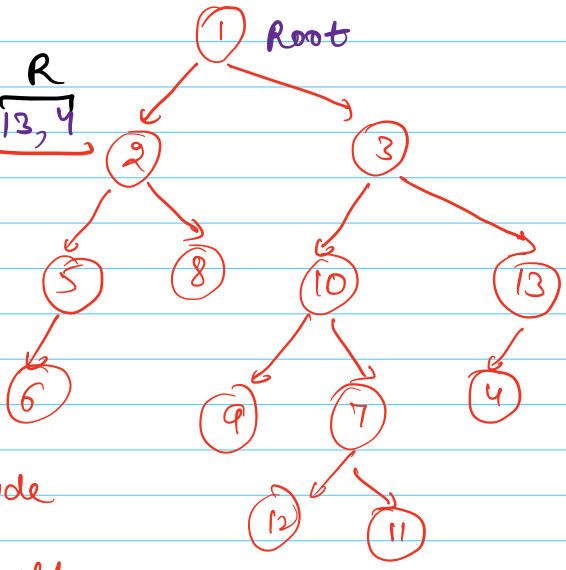
1>	Preorder	Node	Left	Right
2>	Inorder	Left	Node	Right
3>	Postorder	Left	Right	Node.
4>	Level order → next class			

Preorder traversal

N L R N L R

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

Node Left Right

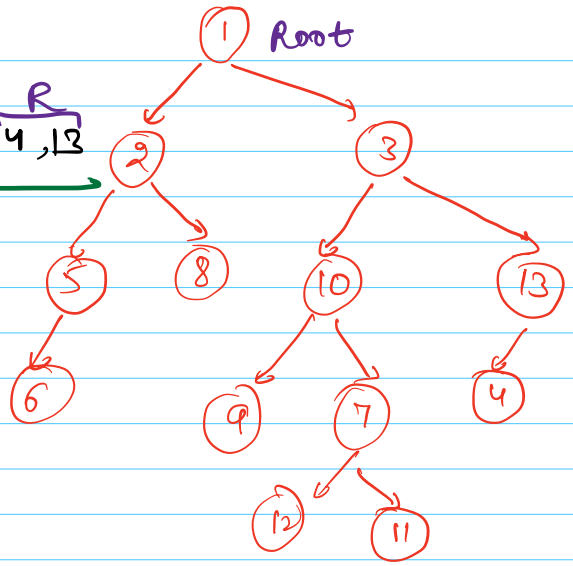



```
void preorder (root) {
    if (root == NULL) return;
    print (root.data);          Node
    preorder (root.left);       Left
    preorder (root.right);      Right
}
```

2> Inorder traversal

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

Left Node Right



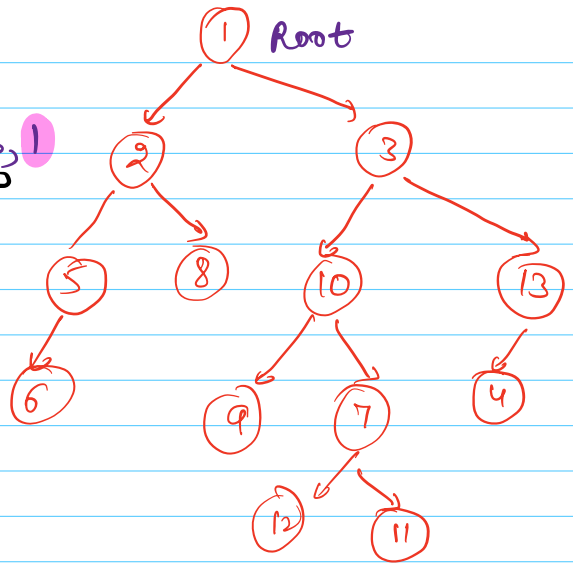

```
void inorder (root) {
    if (root == NULL) return;
    inorder (root.left);      Left
    print (root.data);        Node
    inorder (root.right);     Right
}
```

3) Post order traversal

L L R N L R N N

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

Left Right




```
void postorder (root) {
    if (root == NULL) return;
    postorder (root.left);     Left
    postorder (root.right);    Right
    print (root.data).         Node
}
```

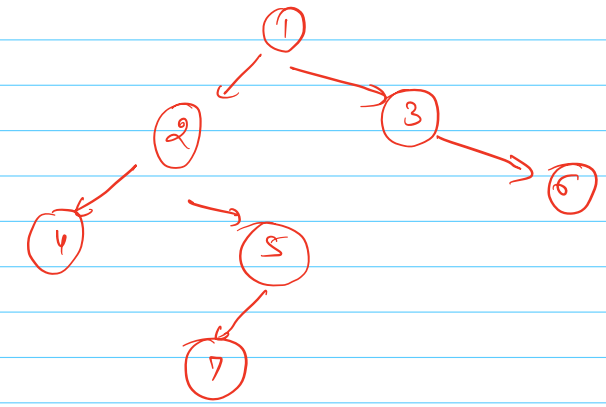
TC: O(N) , SC: O(H) → height of tree.

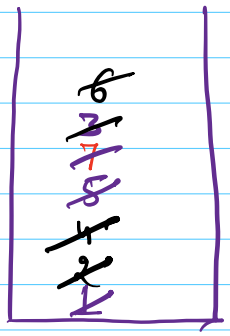
Meet at 8:33 am IST

Q = Write iterative code of inorder traversal.

```
void inorder (root) {
    if (root == NULL) return;
    inorder (root.left);      Left
    print (root.data);        Node
    inorder (root.right);     Right
}
```

Recursion → stack

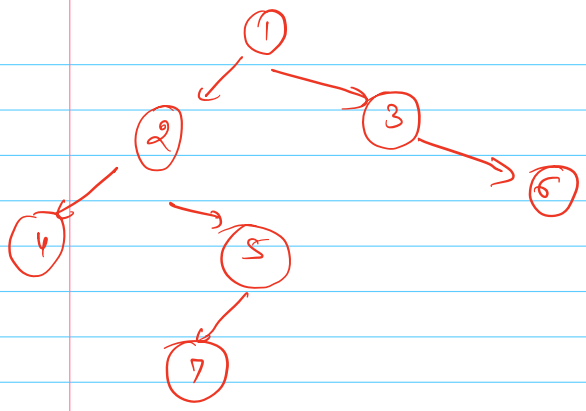


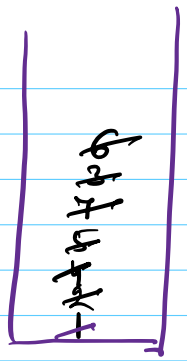


o/p → 4, 2, 7, 5, 1, 3, 6

```
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;
    }
}
```





cur → 1 2 4 null 4 null 2 5 7 null, 7, null 5 null
1 3 null 3 6 null 6 null

o/p → 4, 2, 7, 5, 1, 3, 6

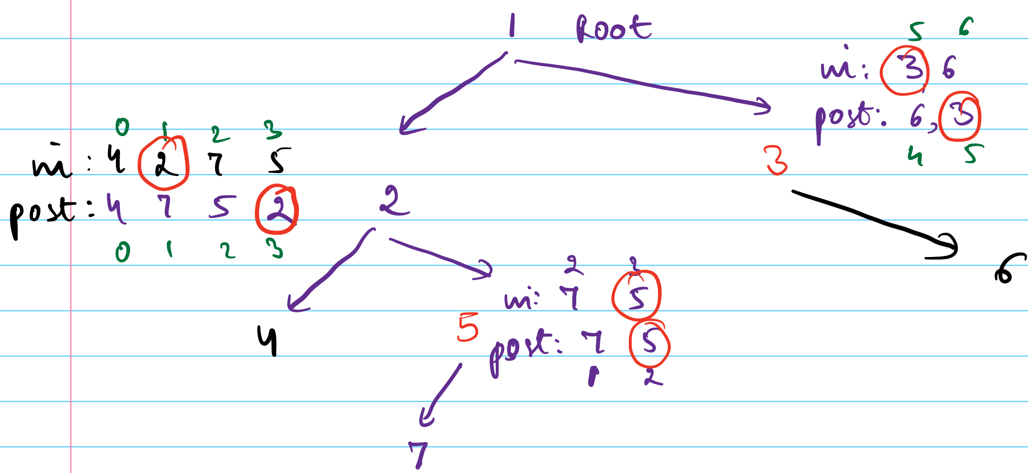
TC: O(N) , SC: O(H)

HW: Iterative code of preorder.

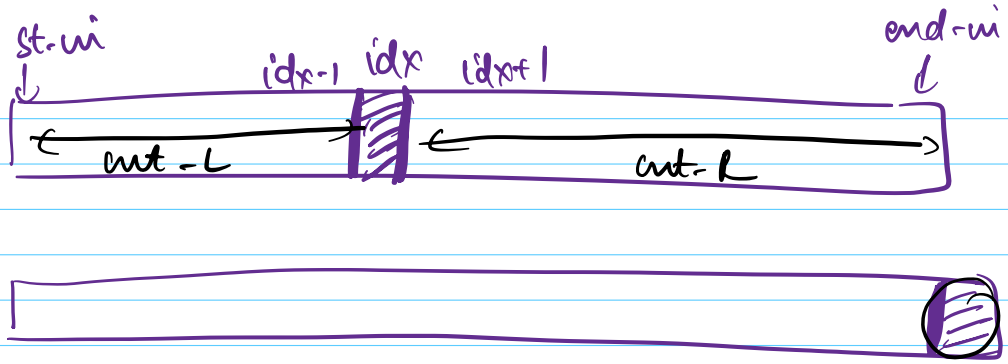
Q Construct binary tree from inorder & postorder (distinct values).

	0	1	2	3	4	5	6	
Inorder →	4	2	7	5	1	3	6	
Postorder →	4	7	5	2	6	3	1	Root
	0	1	2	3	4	5	6	

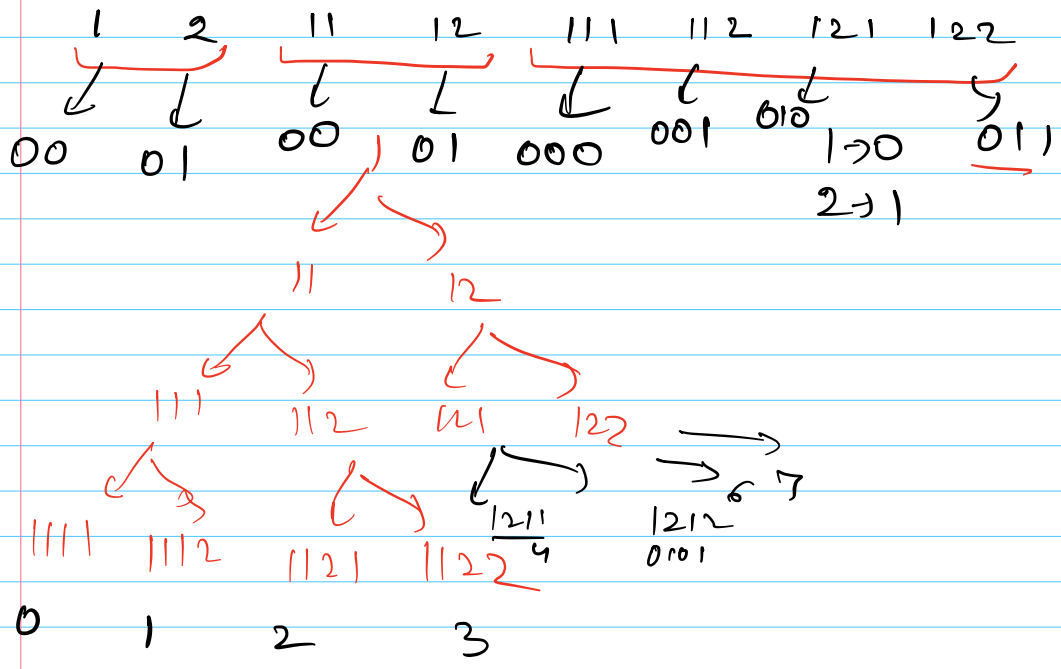
	0	1	2	3	4	5	6
Inorder →	4	2	7	5	1	3	6
Postorder →	4	7	5	2	6	3	1 (Root)




```
Node buildTree(in[], post[], st-in, end-in, st-p, end-p){
                          (→0)   (→N-1)  (→0)  (→N-1)
    if (st-in > end-in) return null;
    root = new Node(post[end-p]);
    idx = getIndex(post[end-p], in[], st-in, end-in);
          TC: O(N) → Hashmap<in[k], k>   (value, index)
    cnt-L = idx - st-in        // st-in → idx-1
    cnt-R = end-in - idx       // idx+1 → end-in
    root.left = buildtree(in, post, st-in, idx-1, end-p-cntR-1)
    root.right = buildtree(in, post, idx+1, end-in, end-p-1);
    return root;
}
```


st.ui
idx-1
idx
idx+1
end-ui
cut-L
cut-R

TC: O(N+N) , SC: O(N+H)


1
2
11
12
111
112
121
122
00
01
00
01
000
001
010
1→0
2→1
011
11
12
111
112
121
122
1211
4
1212
0101
6
7
1111
1112
1121
1122
0
1
2
3