Trees 1

- Madhan Kumar M S
- Abhishek Sharma
- Akansh Nirmal
- Balaji S K
- Bhaveshkumar
- Burhan
- Gagan Kumar S
- Murali Mudigonda
- Naval Oli
- Nikhil Pandey
- Pankaj Bhanu
- PREETHAM
- Purusharth A
- Rajat Sharma
- Rajendra
- Sanket Giri
- Saurabh Ruikar
- Shani Jaiswal
- sharath r
- Subhashini
- Subhranil Kundu
- Sumit Adwani
- Suyash Gupta
- Vasanth
- Vetrivel H M
- Vishal Mosa
- Yugesh v

AGENDA:

- Introduction
- Traversals
- Iterative Inorder
- Construct tree from inorder & postorder

Important Announcement

Mock Interview

Current PSP

↓

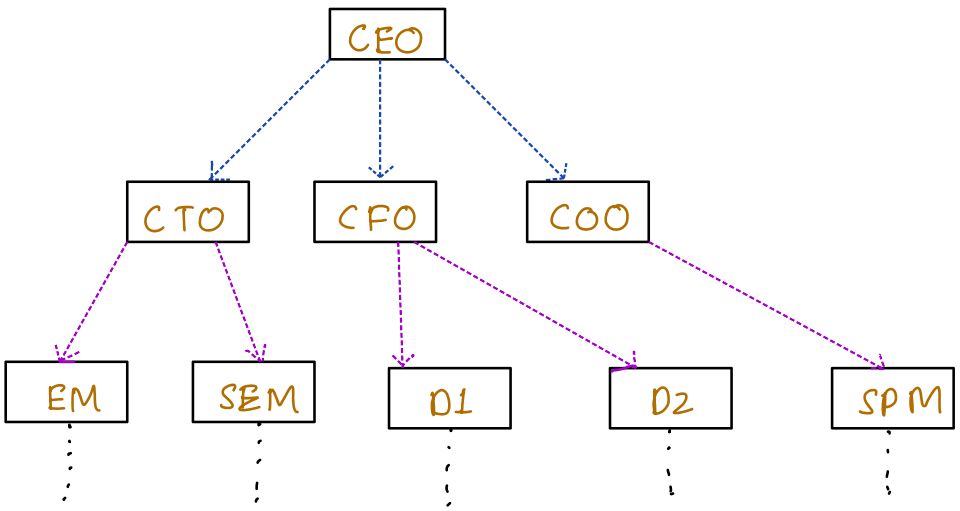
62

~30 ⟶ 100% PSP

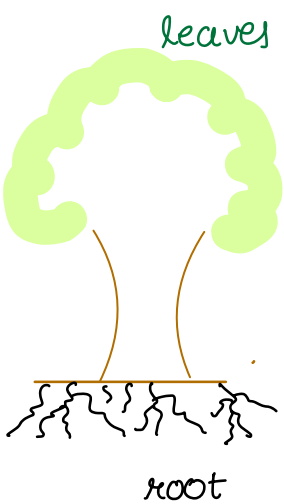
↓

50

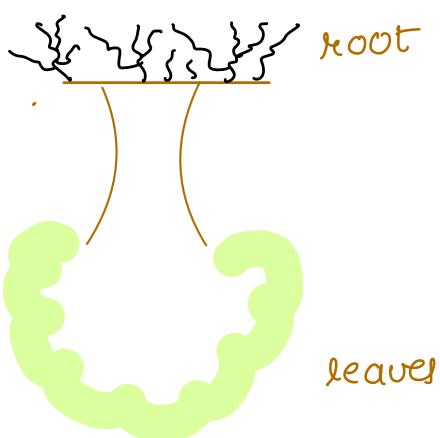
Hierarchical Data structure

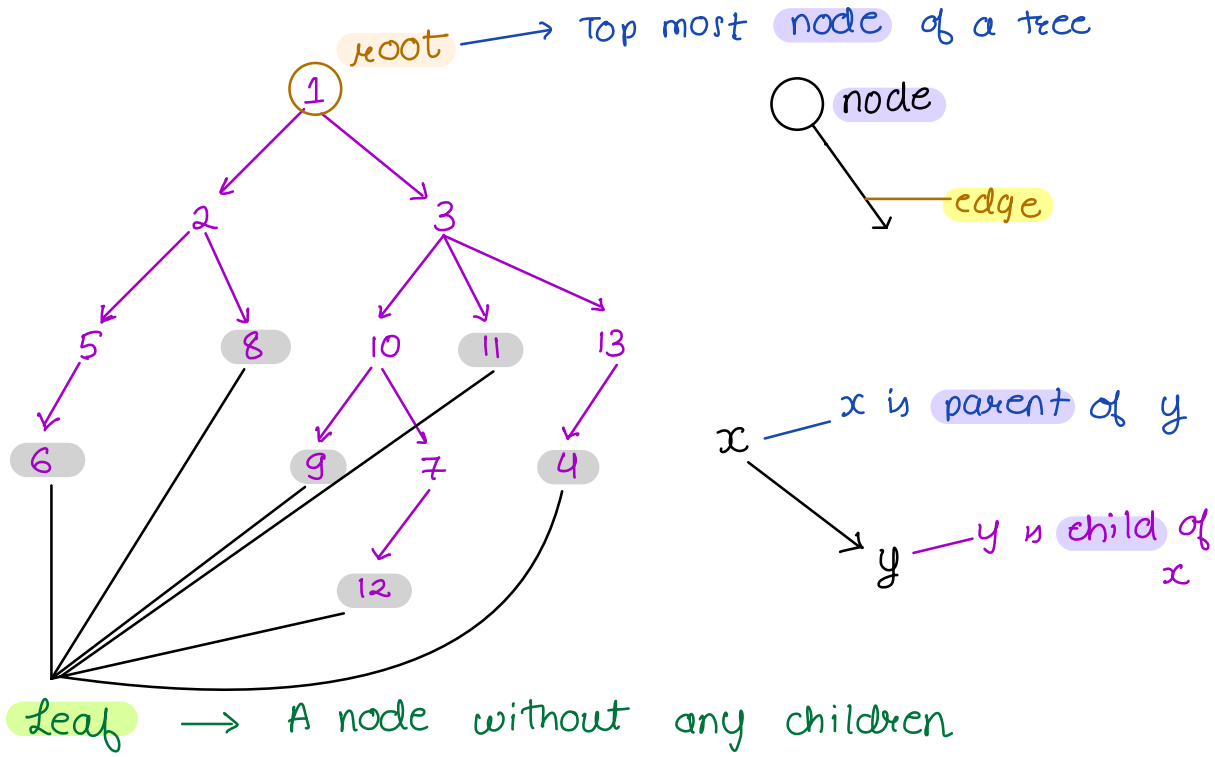


Tree in real life



Tree in computer science.



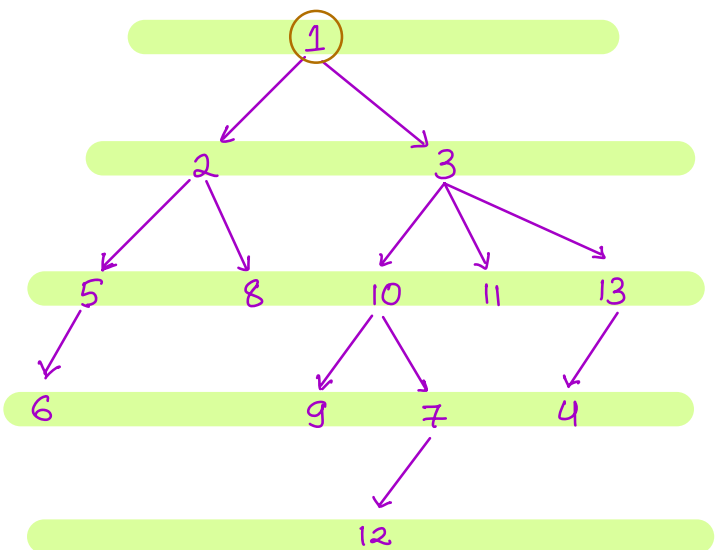


Leaf → A node without any children

Subtree → All the nodes that can be reached from a node.

① → A single node is a root as well as a leaf.

Levels in a Tree



Levels

L0

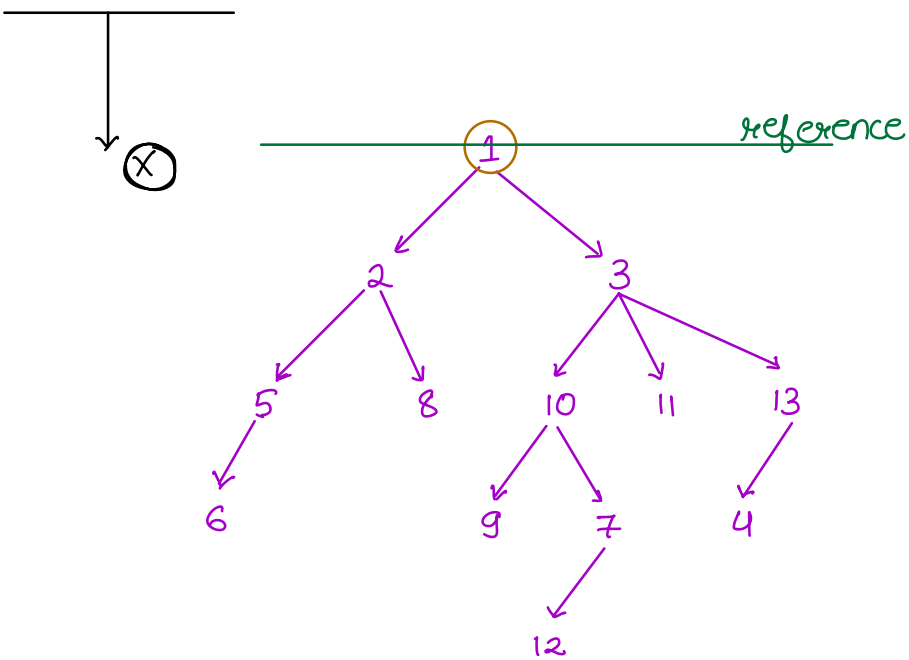
L1

L2

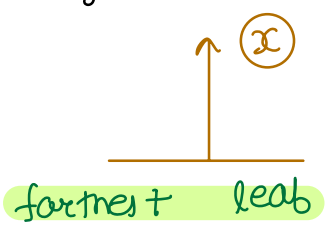
L3

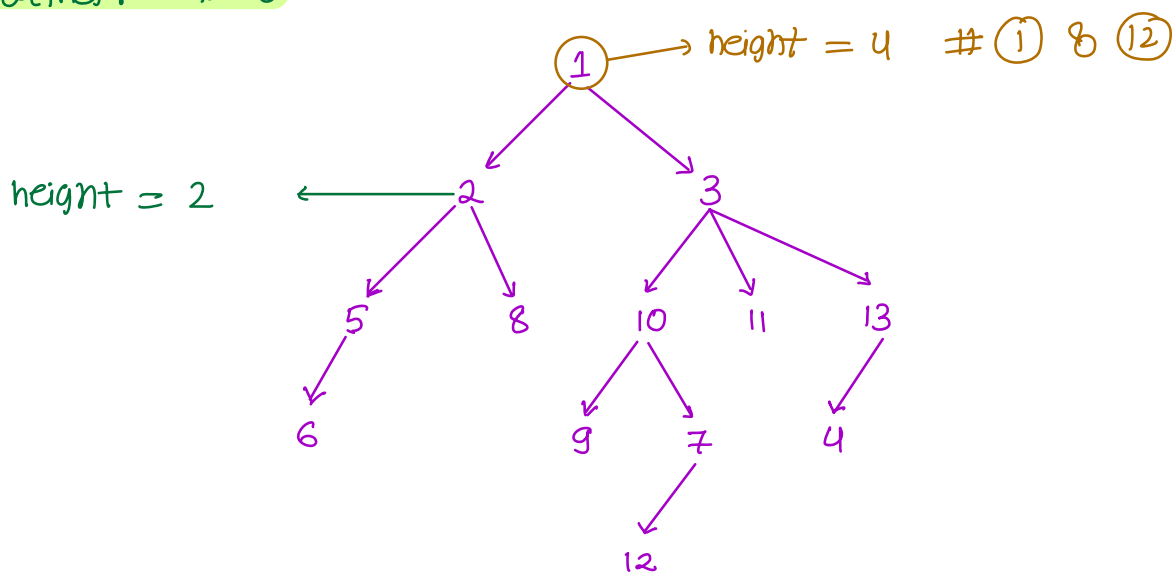
L4

Depth → # edges from root to node x



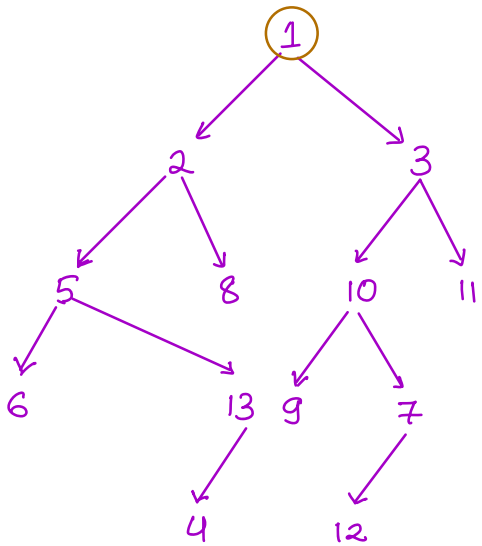
Height # edges b/w node and farthest leaf from node

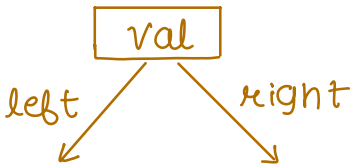




Binary Tree

Every node can have atmost 2 children {0, 1, 2}






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

Traversals in a Binary Tree

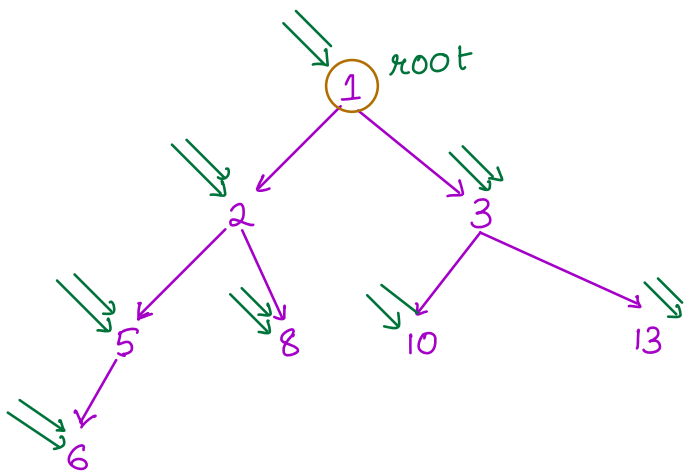
1) Preorder ⟶ Node Left Right

2) Inorder ⟶ Left Node Right

3) Postorder ⟶ Left Right Node

4) Levelorder ⟶ Next Class

Preorder Traversal Node Left Right



Output ⟶ 1 2 5 6 8 3 10 13

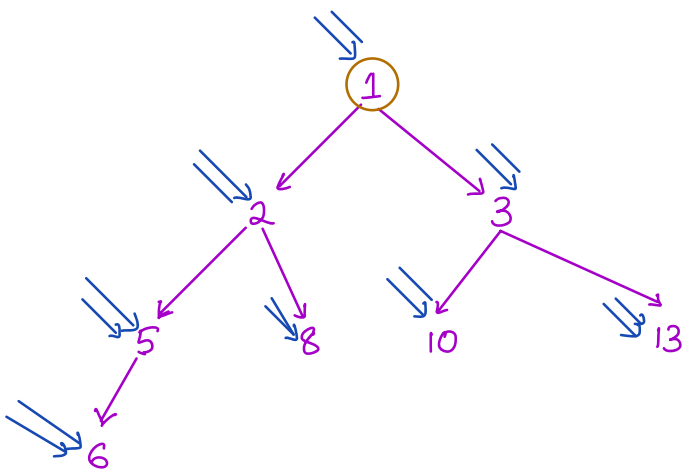
TC: O(N)

SC: O(H)

```
void preorder (root) {
    if (root == null) return          // Base cond"

    print (root.data)                 // Node
    preorder (root.left)              // left subtree
    preorder (root.right)             // right subtree
}
```

Inorder Left Node Right



Output ⟶ L L N R

6 5 2 8 1 10 3 13

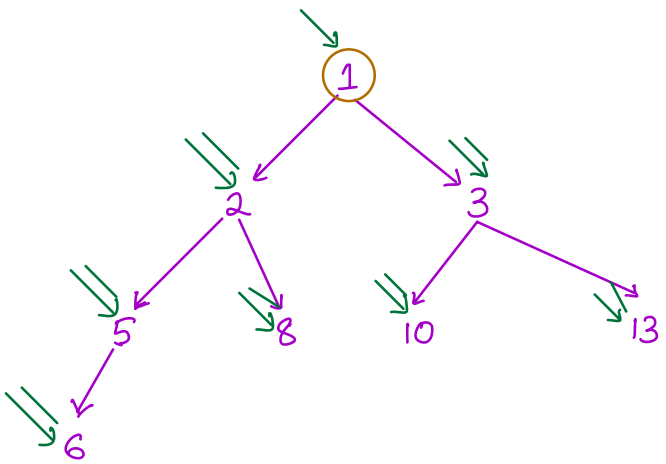
L N R

SC: O(H)

```
void inorder (root) {                                   TC : O(N)
    if (root == null) return        // Base cond"

    inorder (root.left)             // left subtree
    print (root.data)               // Node
    inorder (root.right)            // right subtree
}
```

Post order traversal L R N



Output ⟶ L L R N

6 5 8 2 10 13 3 1

L L R N L R N

TC: O(N)

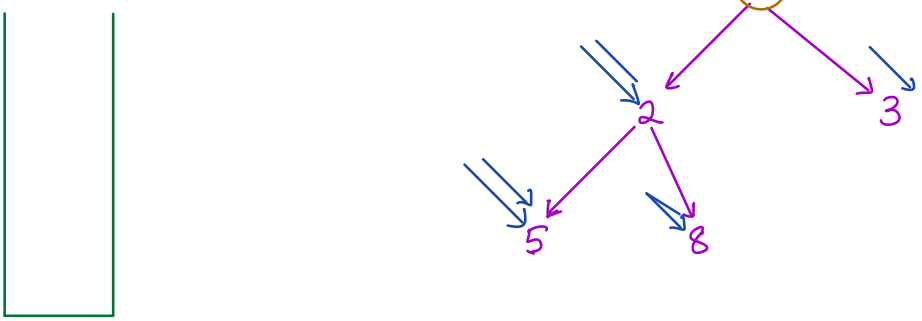
```
if (root == null) return        // Base cond"

postOrder (root.left)           // left subtree
postOrder (root.right)          // right subtree
print (root.data)               // Node
```

SC : O(H)

Q) Write iterative code of inorder traversal

L N R



(5) (2) (8) (1) (3)

Output ⟶ 5 2 8 1 3

```
void iterativeInorder(root) {
    stack = []
    node = root

    while (node != null || !stack.isEmpty())
        if (node != null) {
            stack.push(node)
            node = node.left
        } else {
            node = stack.pop()
            print(node.data)
            node = node.right
        }
}
```

TC: O(N)

SC: O(H)

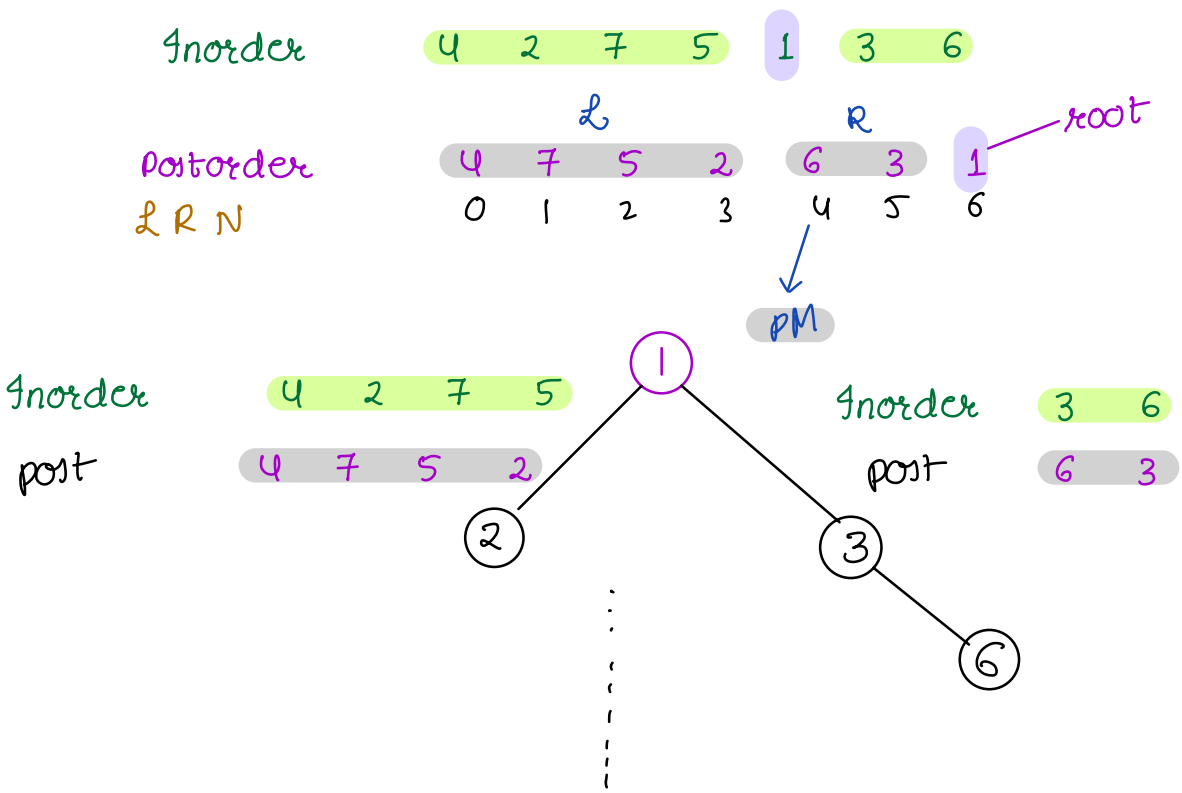
22:45

HW. Iterative version for preorder
postorder

Q> Construct binary tree from inorder & postorder
All the values are distinct

Input

L N R




```
int[] in;
int[] post;                    } declare at class level
HashMap<Int,Int> hm;

TreeNode solve (in[], post[]) {
        this.in = in
        this.post = post
        N = in.length
```

global

global

```
hm = new HashMap<>();
for (i ⟶ 0 to N-1) {
    hm.put(in[i], i)
}

return build(0, N-1, N-1)
}
```

removed ∵ never used

```
TreeNode build(iL, iR, pL, pR) {
    if (iL > iR) return null

    root = TreeNode(post[pR])
    // find the index of root.data
        (1) Linear search
        (2) Create HM { value : index }
                        ↓       ↓
                        K       V

    idx = hm.get(root.data)
    cntR = iR - idx          // [idx+1 .... iR]
    pM = pR - cntR
    root.left = build(iL, idx-1, pM-1)
    root.right = build(idx+1, iR, pR-1)

    return root
}
```

TC: O(N)

SC: O(N)

HashMap

~~pre + post~~

pre + in

in + post

Note : inorder info is needed to build the tree.