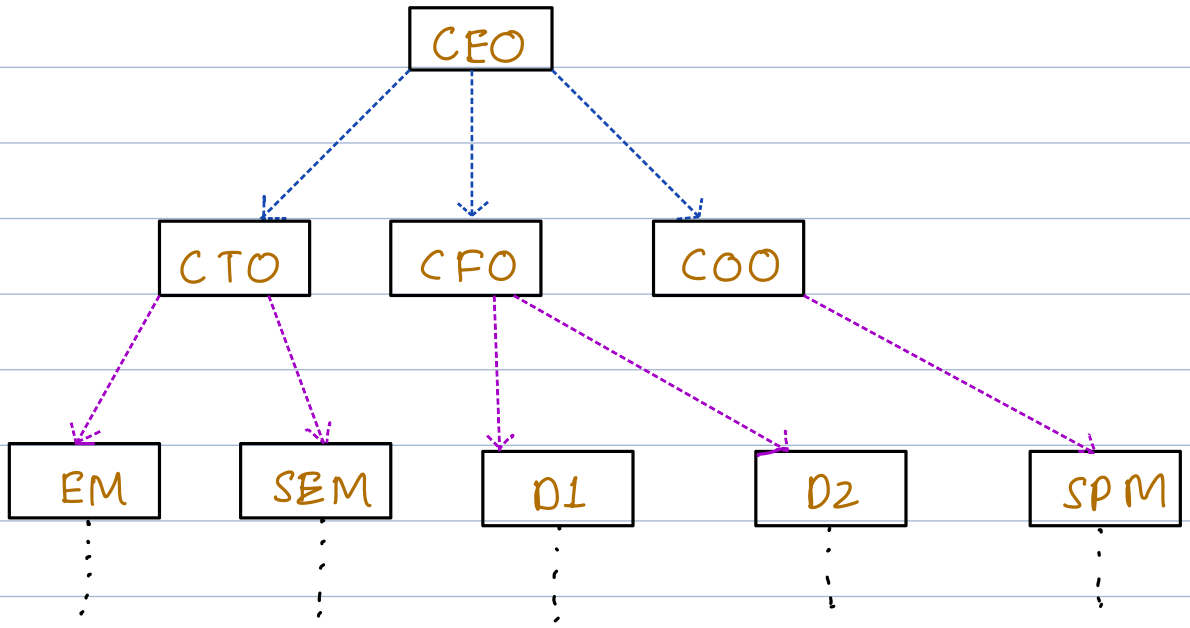
Trees 1

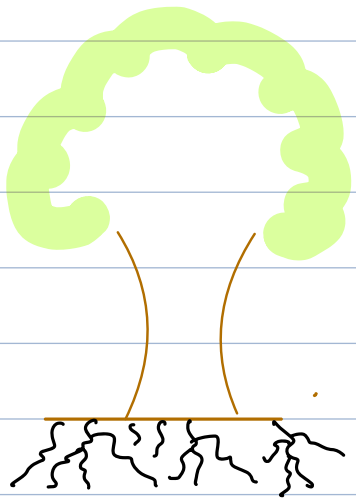
content

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

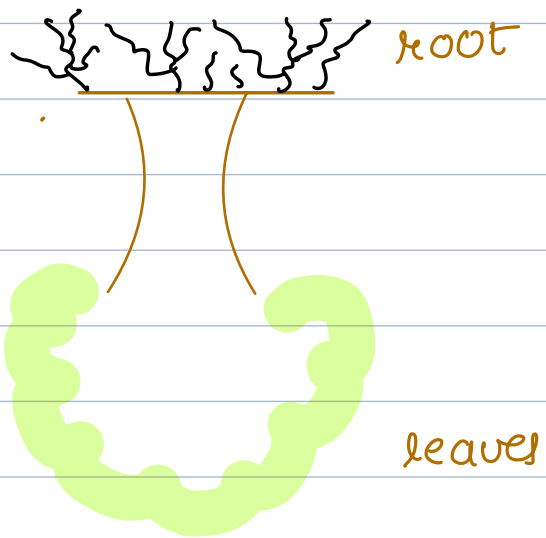
Hierarchical Data structure

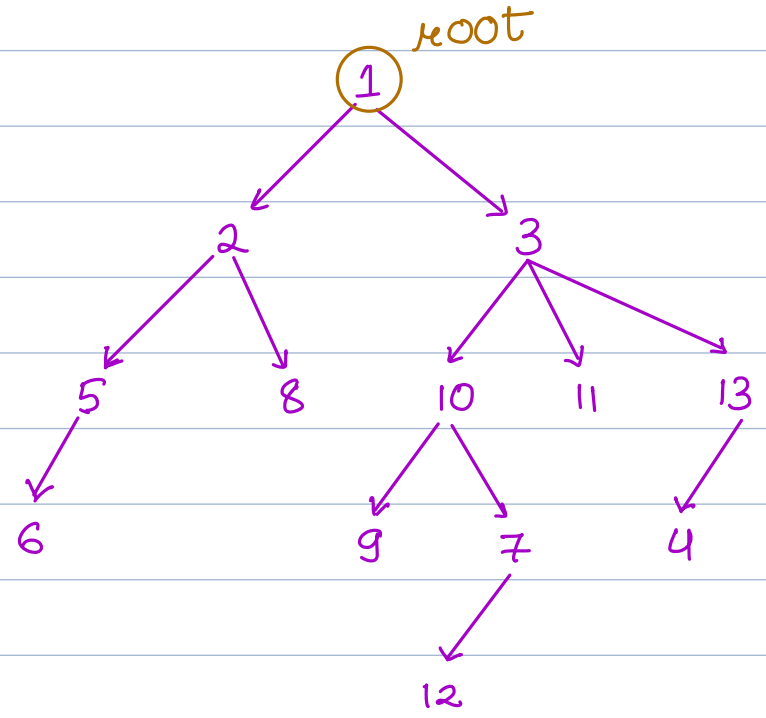


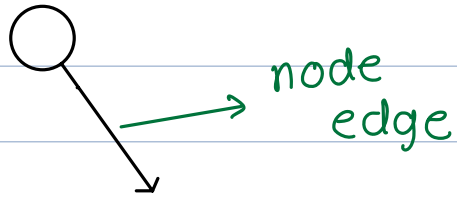
Tree in real life

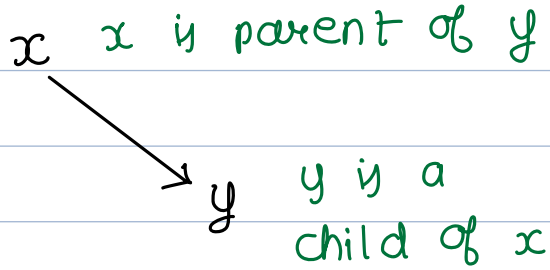


Tree in computer science.







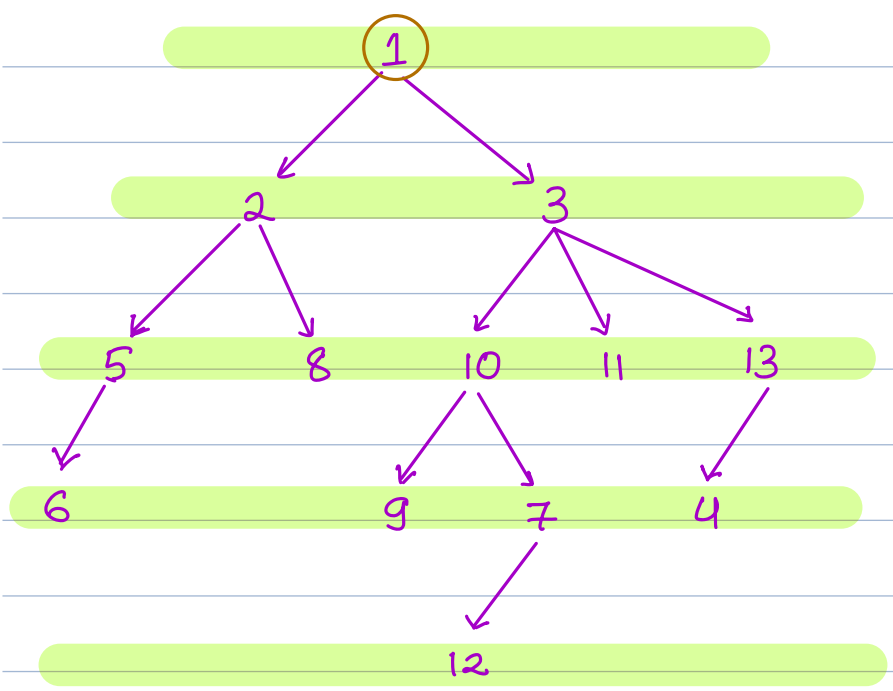


Leaf → A node without any children

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

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

Levels in a Tree



Levels

L0

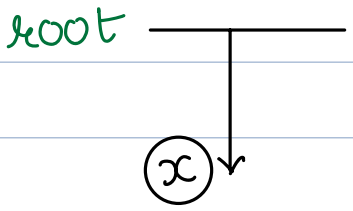
L1

L2

L3

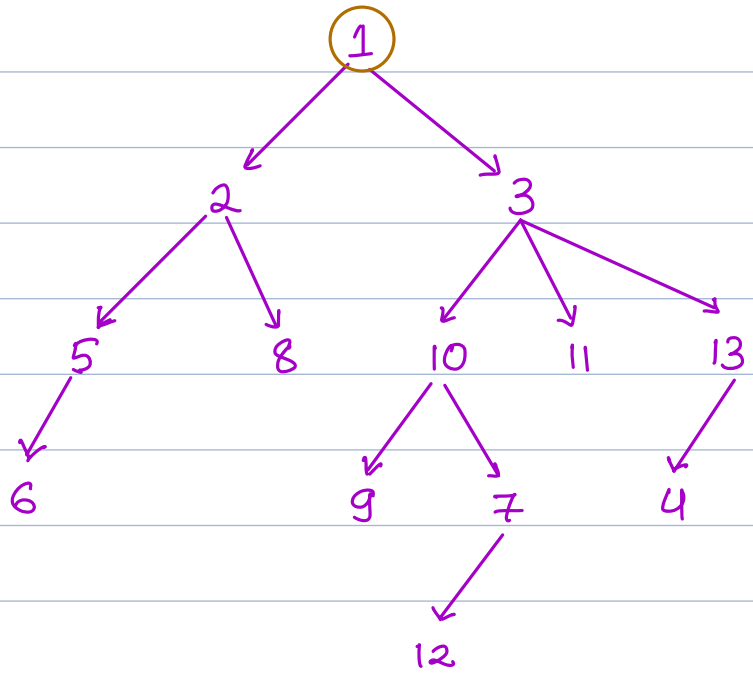
L4

Depth → # edges to traverse from root to reach x

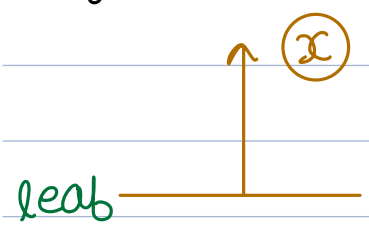


depth of (7) → 3

depth (root) = 0



Height # edges travelled to reach the furthest leaf



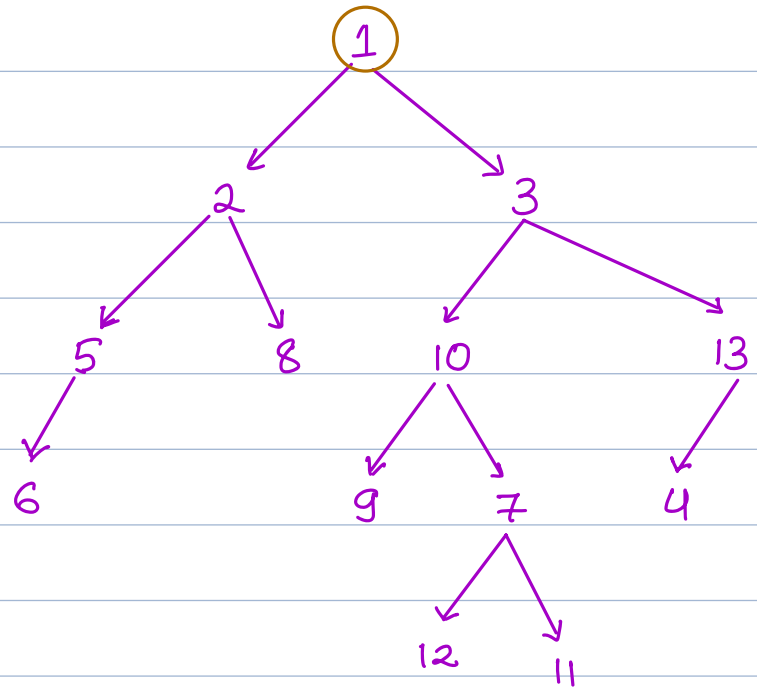
Height (①) = 4

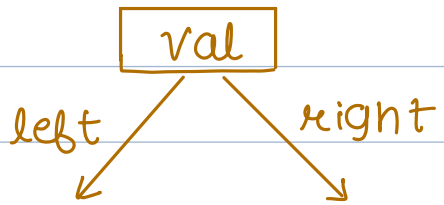
Height (any leaf) = 0

Binary Tree

Every node can have atmost 2 children

{0, 1, 2}






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

Traversals in a Binary Tree

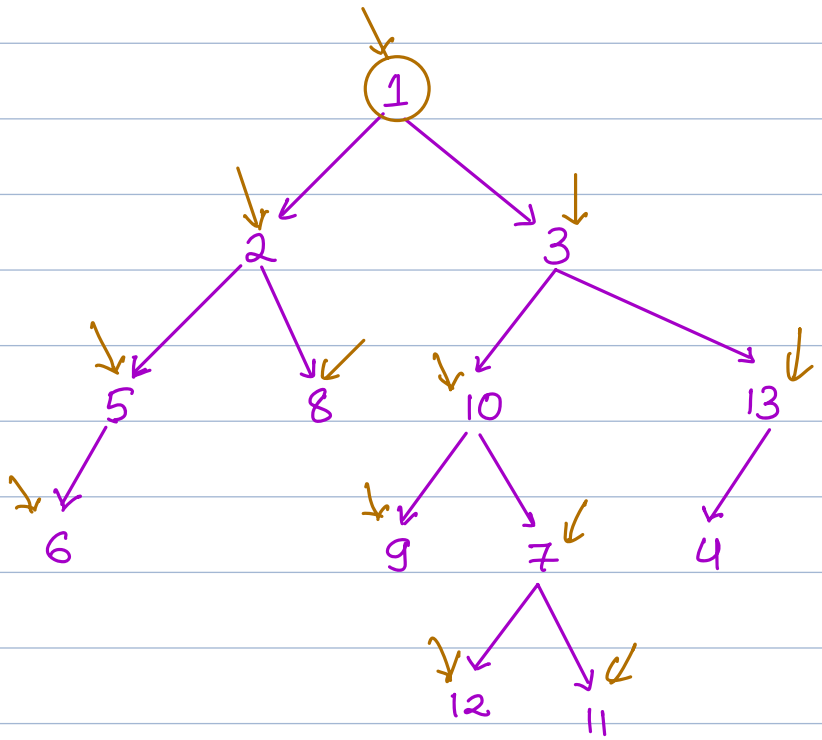
1) Preorder ⟶ Node Left Right

2) Inorder ⟶ Left Node Right

3) Postorder ⟶ Left Right Node

4) Levelorder ⟶ Next Lecture

Preorder Traversal

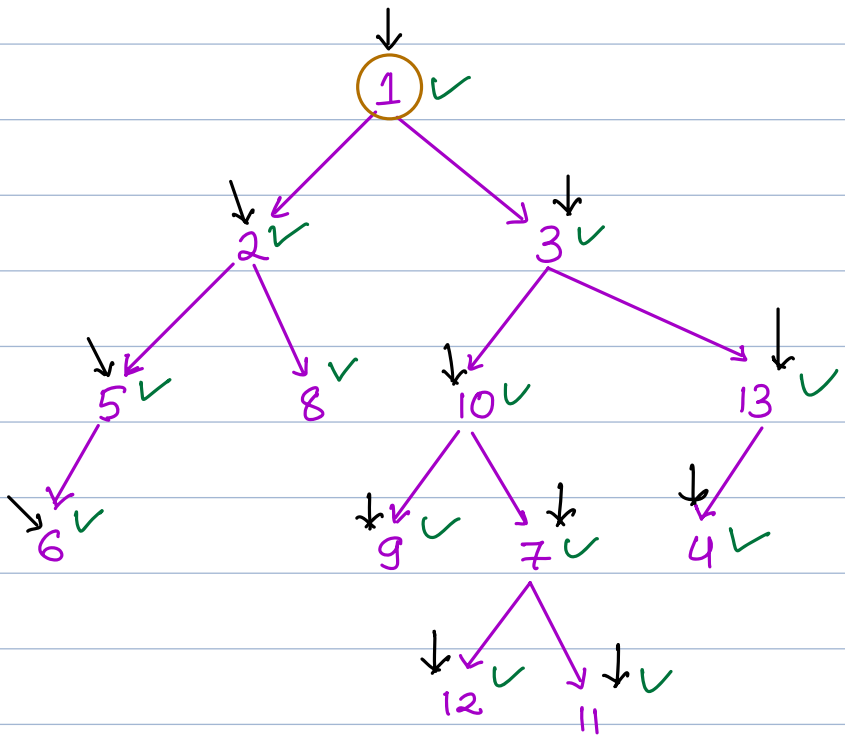


Node	Left				Right							
1	2	5	6	8	3	10	9	7	12	11	13	4
	N	L		R	N	L					R	

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

    print (root.val)
    preorder (root.left)
    preorder (root.right)
}
```

Inorder Left Node Right

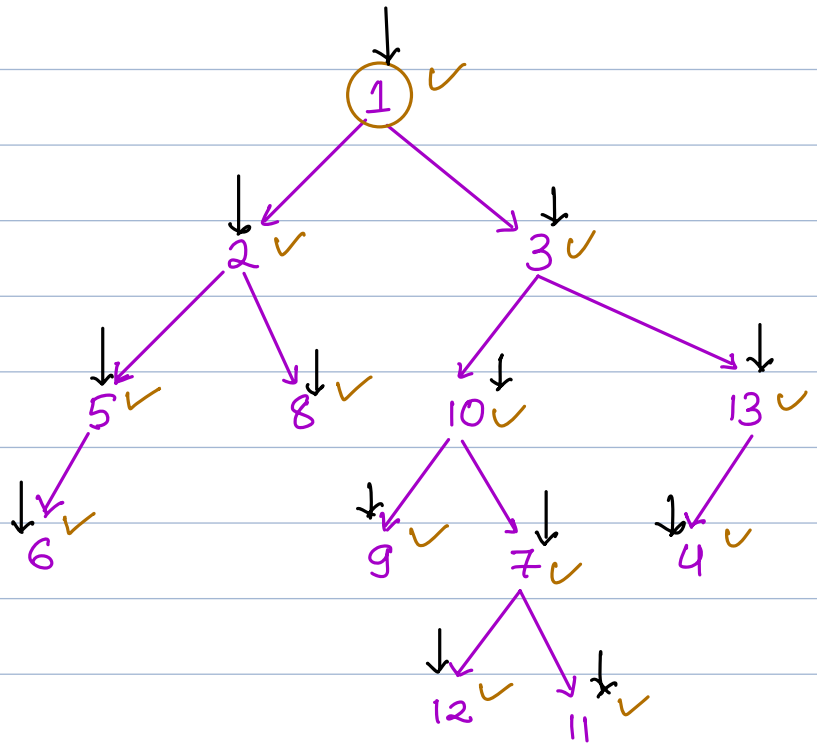


Left				Node	Right							
6	5	2	8	1	9	10	12	7	11	3	4	13
L		N	R		L					N	R	

```
void inorder (root) {
    // Base condition
    if (root == null) return

    in order (root.left)      // Left
    print (root.val)          // Node
    in order (root.right)     // Right
}
```

Post order traversal

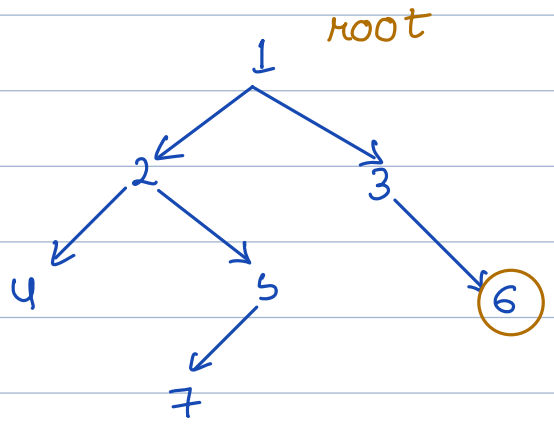


L				R								Node
6	5	8	2	9	12	11	7	10	4	13	3	1
L		R	N	L					R		N	

```
postorder (root.left)
postorder (root.right)
print (root.val)
```

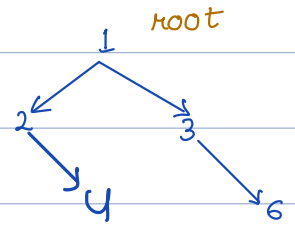
Write iterative code of inorder traversal

```
in order ( root. left )
print ( root. val )
in order  (root. right )
```



stack can help us simulate recursion

4 2 7 5 1 3 6



2 4 1 3 6

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

    while (node != null || !stack.isEmpty()) {
        if (node != null) {
            stack.push(node)
            node = node.left
        }
        else {
            // Restore back to last node
            node = stack.pop()
            print(node.val)
            node = node.right
        }
    }
}
```

Break : 8:49

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

count L = 4 ← | → count R = 2

Inorder	4	2	7	5	1	3	6
Postorder	4	7	5	2	6	3	1

From the post order we can identity 1 is my root

4	2	7	5	3	6
4	7	5	2	6	3

4		7	5
4		7	5

```
TreeNode solve (inorder[], postorder[]) {
    N = inorder.length

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

// Assuming inorder & postorder are global.
// Build value : index mapping for inorder and
   call it indices


TreeNode build (stin, endin, stpost, endpost)
                                  Not used
    // base
    if (stin > endin) {
        return null
    }

    val = postorder[endpost]
    root = new TreeNode (val)

    idx = indices.get(val)
    // this will give us index of root val in inorder

    countLeft = idx - stin   // stin ..... idx-1
    countRight = endin - idx // idx+1 .... endin

    root.left = build (stin, idx-1, postEnd -
                                    countRight - 1)
```

0	1	2	3	4	5	6
4	2	7	5	1	3	6
4	7	5	2	6	3	1

```
        root.right = build(idx+1, endIn, postEnd-1)

        return root
    }
```

Tc : O(N) SC : O(N)