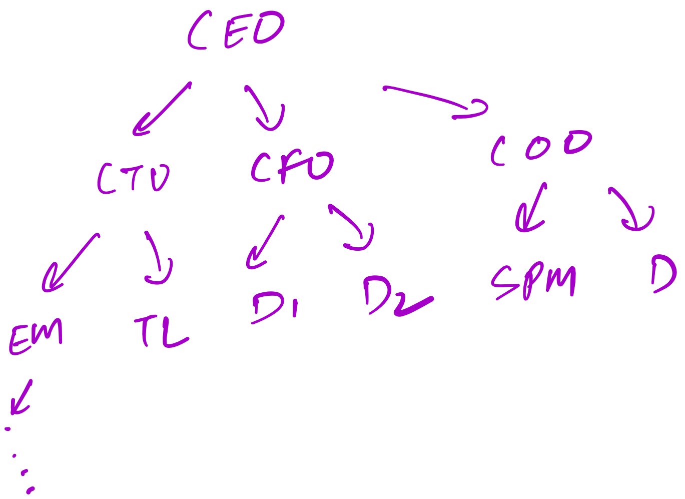
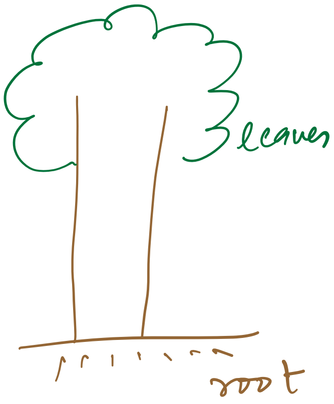
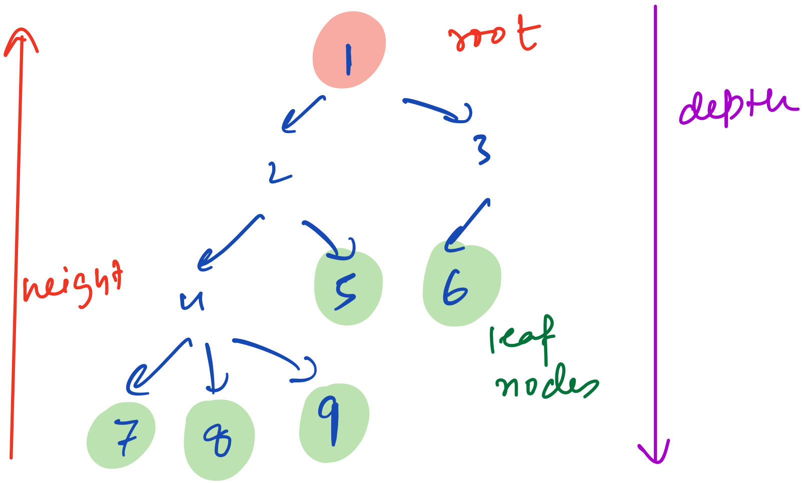
Trees I : Structure & Traversal

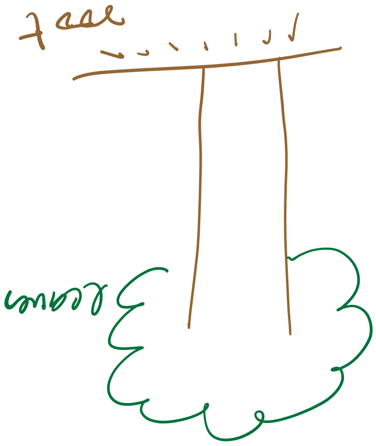
Heirarchial DS

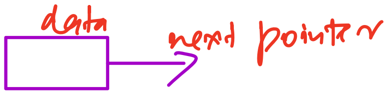


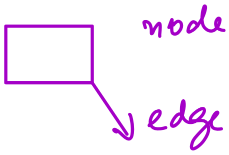


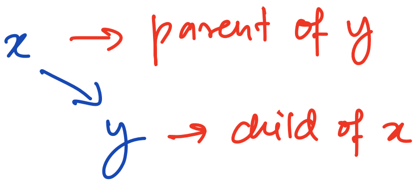
Tree in CS → Inverted tree







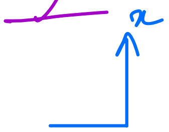




Root → topmost node of a tree, it is the tree representative.
↳ only node without parent.

leaf → node without children

Height → # edges to travel from node x to farthest leaf.

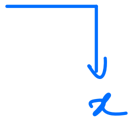


height(2) = 2

height(leaf) = 0

Height of tree = Height(root) = 3

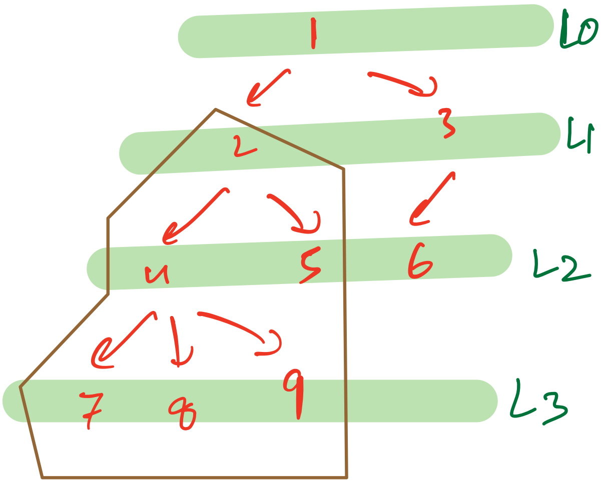
Depth / Level → # edges to travel from root to current node x.



depth(2) = 1

depth(root) = 0

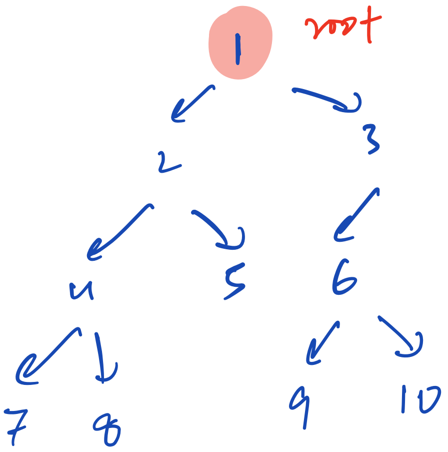
Subtree → subtree of a node x is the part of the tree which includes all the nodes that can be travelled from x.



Can leaf node be a subtree → Yes

Do all nodes have parent → No (eg ⇒ root node)

Binary tree → A tree in which ∀ nodes, # children = {0,1,2}



```
class Node {
    int data;
    Node left, right;
    Node(x){
        data = x
        left = right = null
    }
}
```

Tree traversal

1. Pre order traversal — Node Left Right
2. Inorder traversal — Left Node Right
3. Post order traversal — Left Right Node
4. Level order traversal → next class

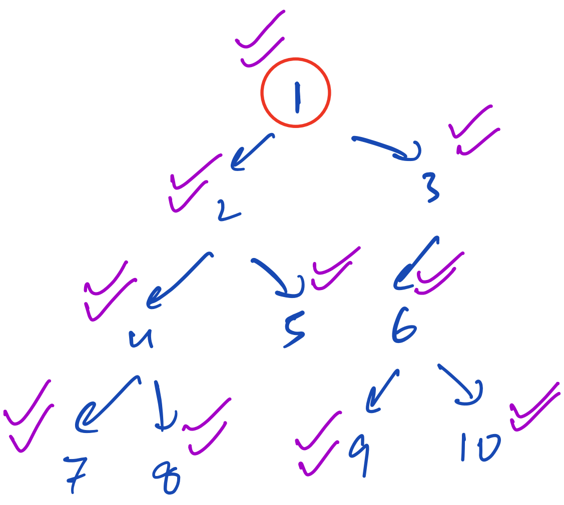
1. Pre order traversal

N L R

Node — 1
Left — 2 4 7 8 5
Right — 3 6 9 10

1 2 4 7 8 5 3 6 9 10

2 (N) 4 7 8 (L) 5 (R) 3 (N) 6 9 10 (L)



N → # nodes

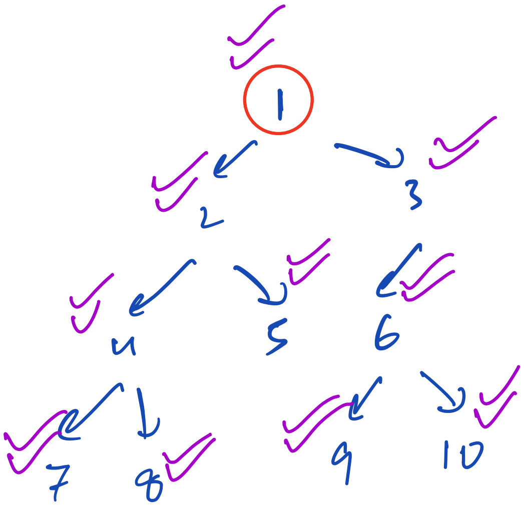
H → height of tree
= O(N)

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

TC = O(N)

SC = O(H)

2. Inorder traversal   L N R



Left | Node | Right

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

L N R | | L N

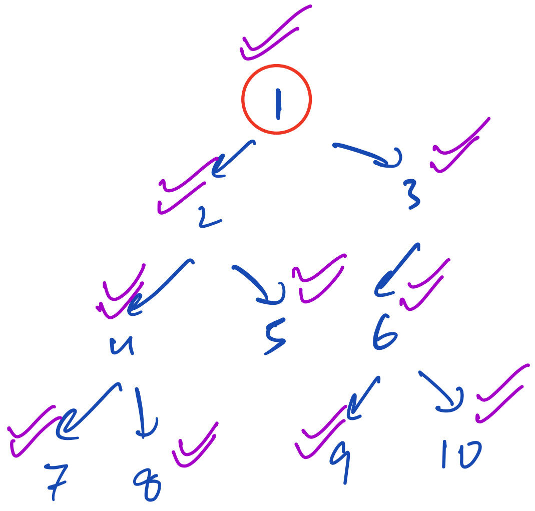
```
void inorder(root) {
    if(root == null) return
    inorder (root.left)       Left
    print(root.data)          Node
    inorder (root.right)      Right
}
```

TC = O(N)

SC = O(H)

3. Post order traversal   L R N

$$\overbrace{\underbrace{7\ 8\ 4}_{L}\ \underbrace{5}_{R}\ \underbrace{2}_{N}}^{Left}\ \overbrace{\underbrace{9\ 10\ 6}_{L}\ \underbrace{3}_{N}}^{Right}\ \overbrace{1}^{Node}$$



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

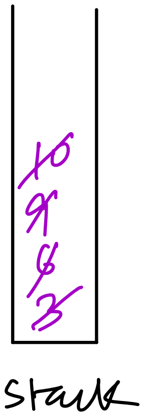
TC = O(N)

SC = O(H)

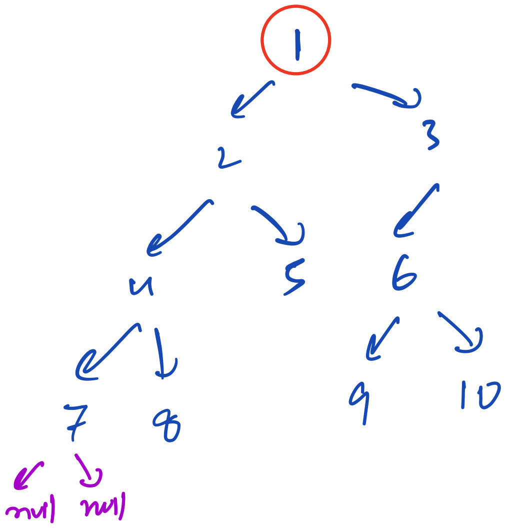
Ques → Iterative inorder traversal

recursion → iterative
(stack)

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



curr = ~~1~~ ~~2~~ ~~4~~ ~~7~~ ~~null~~
~~7~~ ~~null~~ ~~4~~ ~~8~~ ~~null~~
~~8~~ ~~null~~ ~~2~~ ~~5~~ ~~null~~
~~5~~ null ~~1~~ ~~3~~ ~~6~~ ~~9~~ ~~null~~
~~9~~ ~~null~~ ~~6~~ ~~10~~ ~~null~~ ~~10~~
~~null~~ 3 ~~null~~



o/p → 7 4 8 2 5 1 9 6 10 3

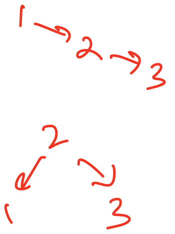
```
curr = root
while( curr != null || !st.isEmpty()) {
    if ( curr != null) {
        st.push (curr)   // store complete node
        curr = curr.left   // left
    }
    else {
        curr = st.pop()
        print (curr.data)   // Node
        curr = curr.right   // Right
    }
}
```

HW → 1. Iterative preorder
2. Iterative postorder

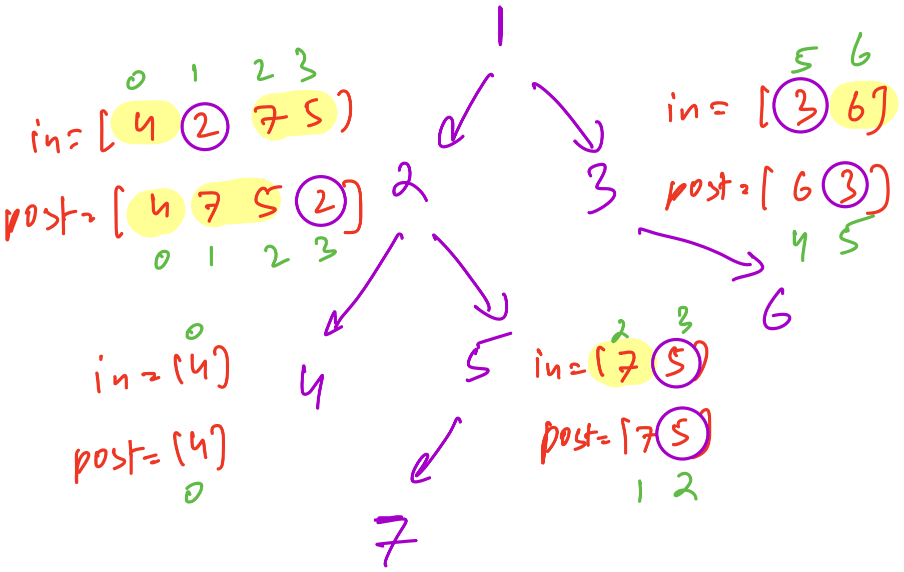
Ques → Construct binary tree from the given inorder & post order traversal. (distinct nodes)

eg →

```
         0  1  2  3  4  5  6
in  = [  4  2  7  5  1  3  6 ]
post= [  4  7  5  2  6  3  1 ]
         0  1  2  3  4  5  root 6
```



1 2 3
same inorder
1 2 3



```
                                   0        N-1      N-1
Node build(in[], post[], inL, inR, postR){
    if(inL > inR) return null

    root = new Node(post[postR])
    // find index of root in inorder array?
     1. travel inorder array
     2. Hashmap (value → index) for in[]  → TC = O(N) } in
                                            SC = O(N) } total
    idx = mp.get(root.data)
    cntR = inR - idx          [idx+1, inR] // how many nodes in right
    root.left = build(in, post, inL, idx-1, postR-cntR-1)
    root.right = build(in, post, idx+1, inR, postR-1)

    return root

}
```

$$TC = O(\underset{\uparrow \atop \text{hashmap}}{N} + \underset{\uparrow \atop \text{main logic}}{N}) = O(N)$$

$$SC = O(\underset{\uparrow \atop \text{hashmap}}{N} + \underset{\uparrow \atop \text{main logic}}{H}) = O(N)$$