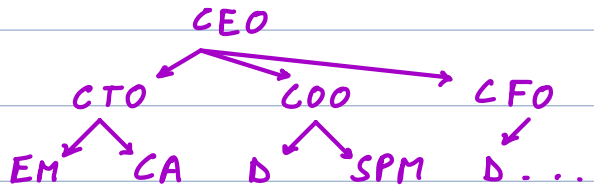
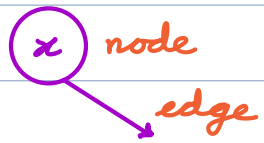
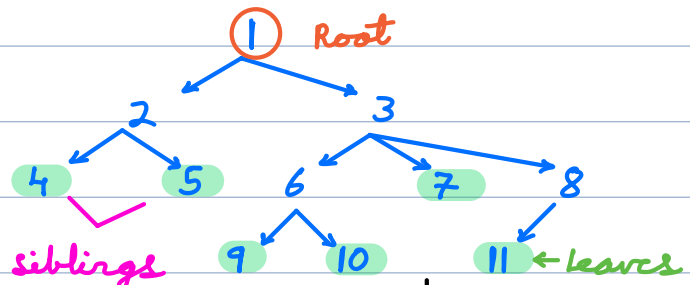
Next week → Tue & Thu (only 2 lectures)

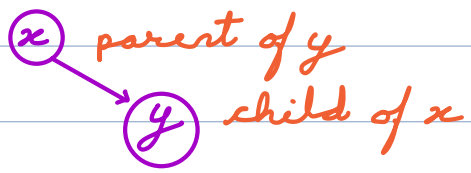
Next to next week → Mon + Tue + Friday

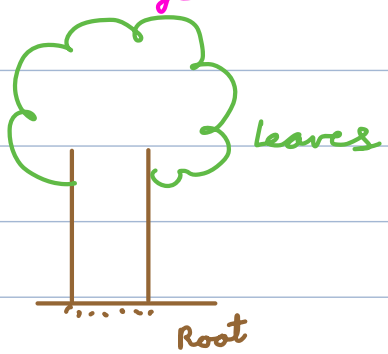
Trees (Hierarchical DS)

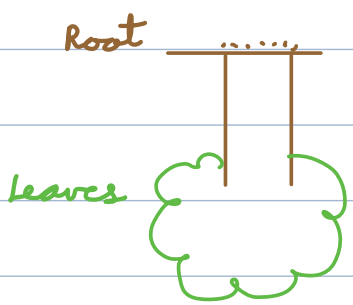






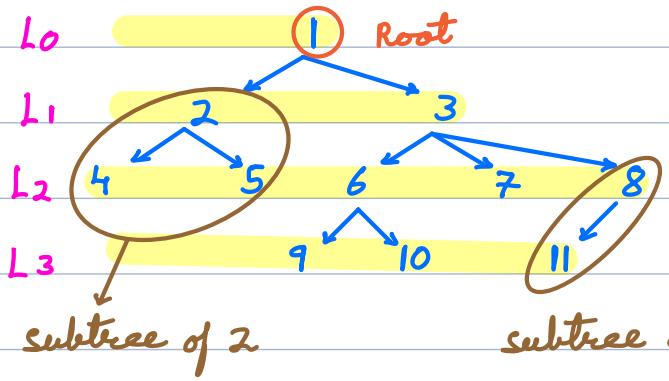




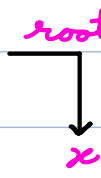


Root → Top most node without parent.
Reference node for a tree.

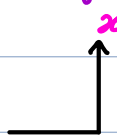
Leaf → Node without any children.



Level / Depth of a node → The # of edges (distance) to travel from root to current node (x).



Height of a node → The # edges to travel from current node (x) to the farthest leaf.



farthest leaf

Height of tree = Height (root)

Subtree of a node (x) → All the nodes that can be travelled from (x) are part of subtree of x.

Can a leaf node be subtree → Yes

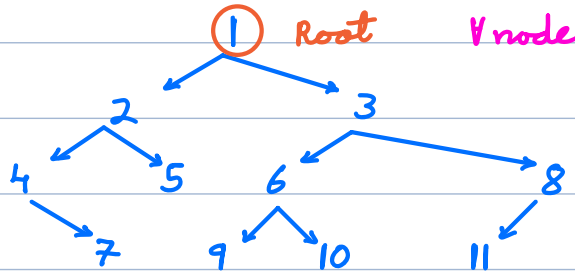
leaf node is subtree of size 1.

Do all nodes have parent → No (root does not have parent)

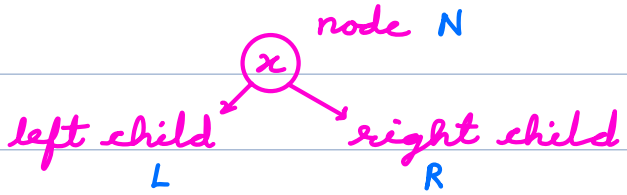
Height (leaf) = 0

Binary Tree → Tree where ∀ nodes, max # children = 2.

∀ nodes, # children → {0, 1, 2}

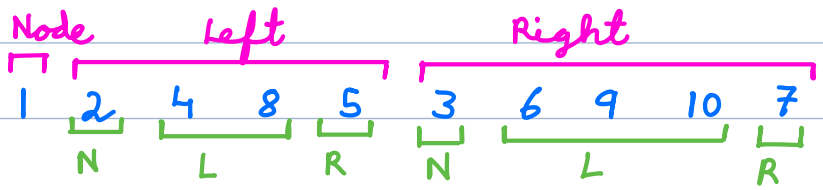


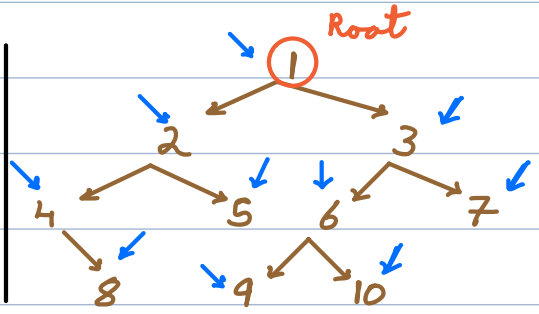
Tree Traversals



1) Pre order N L R
2) In order L N R
3) Post order L R N
4) Level order traversal → Next lecture

Preorder Traversal



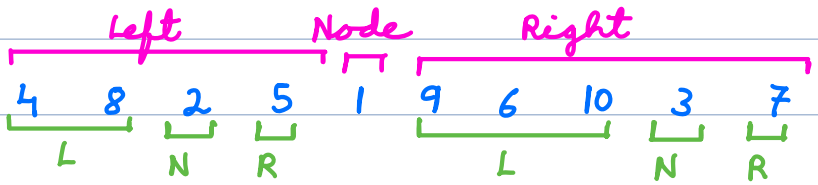


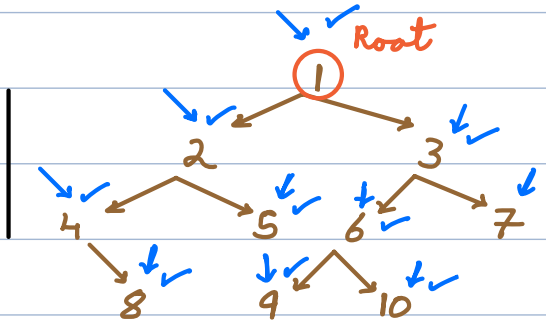

```
void preorder(root) {
    if (root == null) return
    print(root.data)          N
    preorder(root.left)       L
    preorder(root.right)      R
}
```

TC = O(N)

SC = O(H) → height of tree

Inorder Traversal



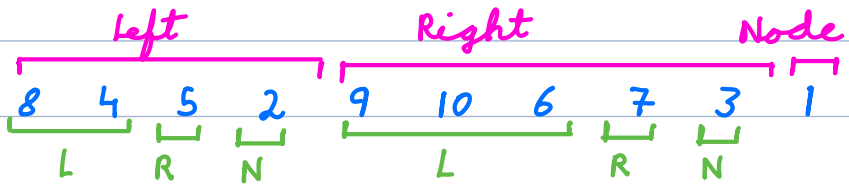


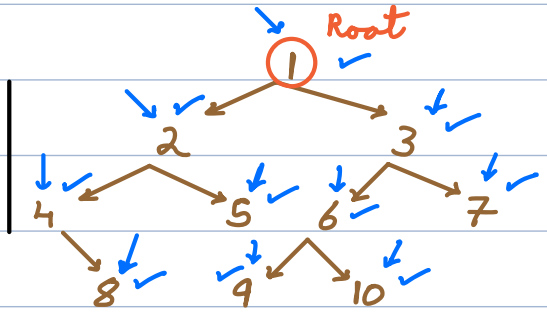
```
void inorder (root) {
    if (root == null) return
    inorder (root.left)     L
    print (root.data)       N
    inorder (root.right)    R
}
```

TC = O(N)

SC = O(H) → height of tree

Post order traversal

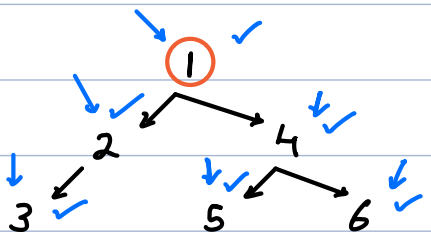





```
void postorder (root) {
    if (root == null) return
    postorder (root.left)     L
    postorder (root.right)    R
    print (root.data)         N
}
```

TC = O(N)

SC = O(H) → height of tree

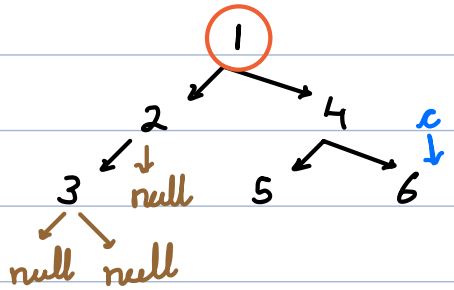


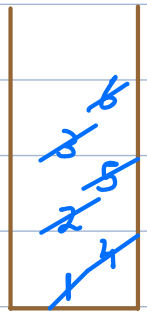
Inorder L N R

3 2 1 5 4 6

Q → For a given tree, find inorder traversal without recursion.

use stack





cur = root

cur 1 2 3 null
3 null 3 null
cur 2 4 5 null
5 null 4 6
cur null 6 null

Inorder L N R

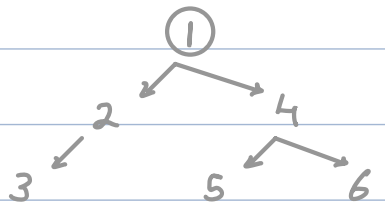
3 2 1 5 4 6

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

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

HW → Iterative sol. for preorder & postorder.

Q→ construct binary tree from given inorder & postorder traversal.



id
0 1 2 3 4 5
in = [3 2 1 5 4 6] L N R
post = [3 2 5 6 4 1] L R N
pL
pR
root

in = [3 2] post = [3 2]
in = [5 4 6] post = [5 6 4]

(inorder & preorder) ✓

```
TreeNode build ( in[], post[], inL, inR, ~~pL~~, pR ) {
    if ( inL > inR ) return null        // not req
    root = new TreeNode ( post[pR] )
    id = findIndex ( in[], root.data )   → travel  TC = O(N)
                                         → precompute & store
    cntR = inR - id          // [(id+1) inR ]   TC = O(1) ✓
    root.left = build ( in, post, inL, id-1, pR-cntR-1 )
    root.right = build ( in, post, id+1, inR, pR-1 )
} return root
```

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