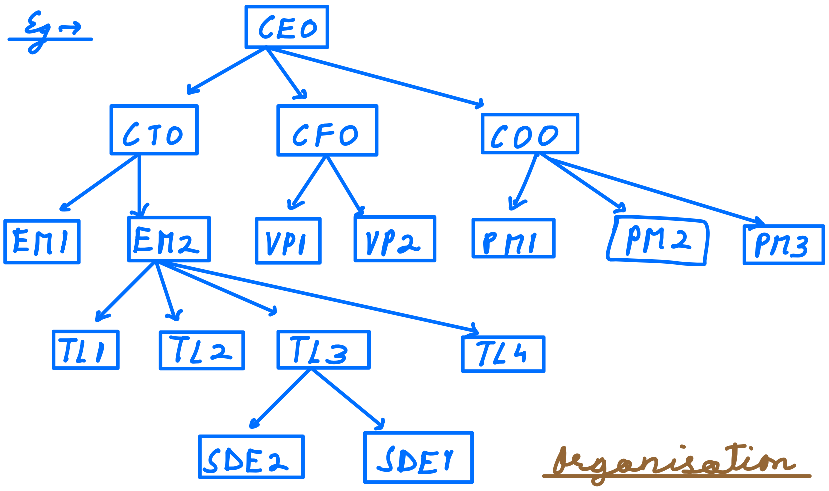
Arrays & Linked list → Linear DS →

Trees → Hierarchical DS

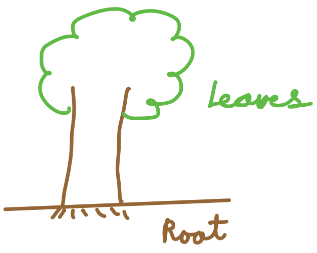
Eg →



Organisation

File System
Family tree
Process Workflow/DB
ARMY
continents → countries → states → cities

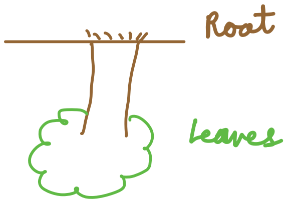
Where is root of tree ?



In Programming →

Tree

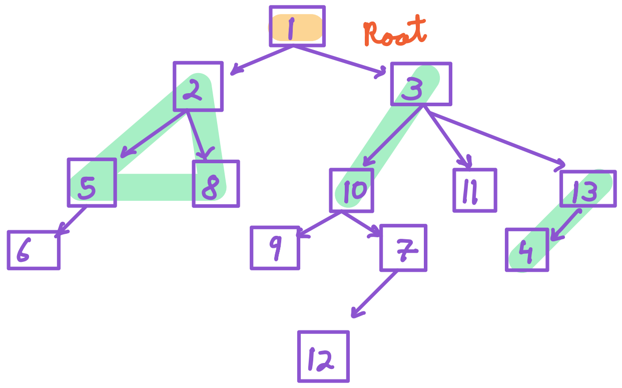
Actual inverted tree



Agenda → 1) Naming & Terms ✓

2) Traversals → a) Preorder
b) Inorder
c) Post Order

} Recursion



Quizzes

1) Parent of 10 → 3

2) children of node 2 → 5 & 8

3) Not a leaf node → 13

4) Can a root become leaf → Yes

Eg → $\boxed{1}$ (only 1 node)

5) Do all nodes have parent → No

Root does not have parent.

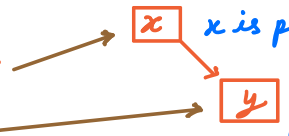
1) Root → Top most node that can be used to Travel the complete tree. (unique)

2) □ node

↘ edge

3) parent

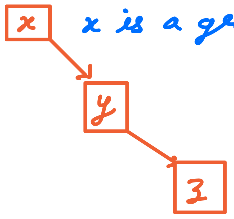
child



x is parent of y

y is child of x.

4)



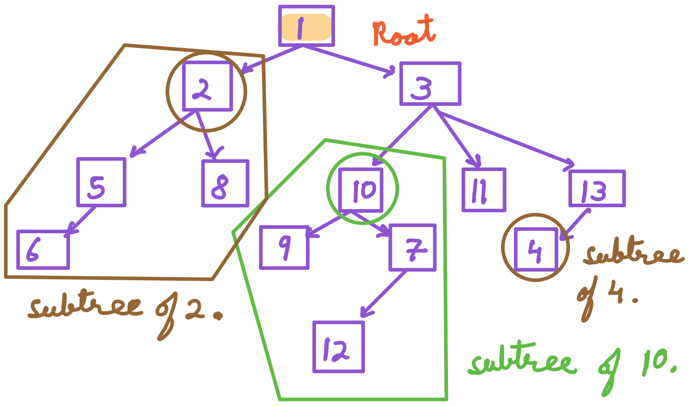
x is a grandparent of z.

z is a grandchild of x.

5) Leaf → Nodes with no children.

Eg → 12 in given tree.

{6, 8, 9, 12, 11, 4}



6) Can a leaf node be subtree → Yes

7) 2 is part of how many subtrees → 2

rooted at 1 & 2.

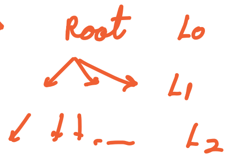
8) Multiple roots → No

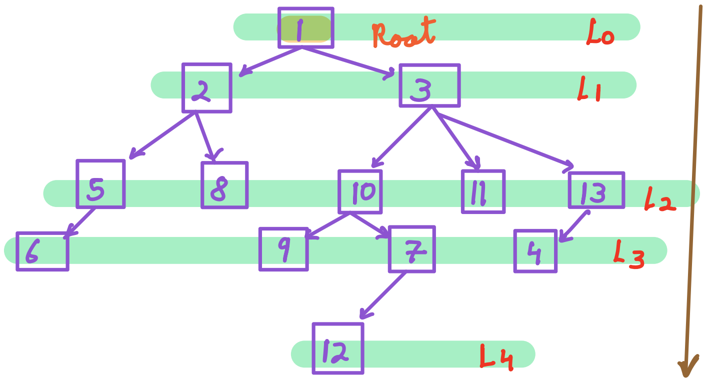
9) Not a part of subtree of 10 → 11

6) Subtree → For any node $\boxed{x}$, all the nodes that can be travelled from $\boxed{x}$ are part of subtree of x.

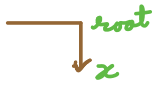
(complete tree is a subtree for the root.)

7) levels →





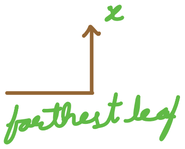
8) Depth of a node x



edges to travel from root to current node x.

Eg → Depth(13) = 2

9) Height of a node x



#edges to travel from current node to farthest leaf.

Eg → Height(2) = 2

10) Height of tree = Height(root)

Given tree = 4

10) Height of leaf = 0
11) Depth of root node = 0
12) Height(8) = 0
13) Depth(8) = 2
14) Depth(1) = 0
15) Last level always leaf node → Yes

Binary Tree → ∀nodes, the nodes can have at max 2 children.
# children {0, 1, 2}.

10:38 PM

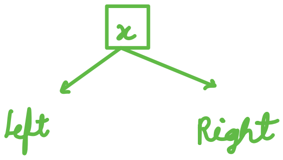
```
class Node {
    int data;
    Node left;
    Node right;
    Node (int x) {
        data = x;
        left = null;
        right = null;
    }
}
```

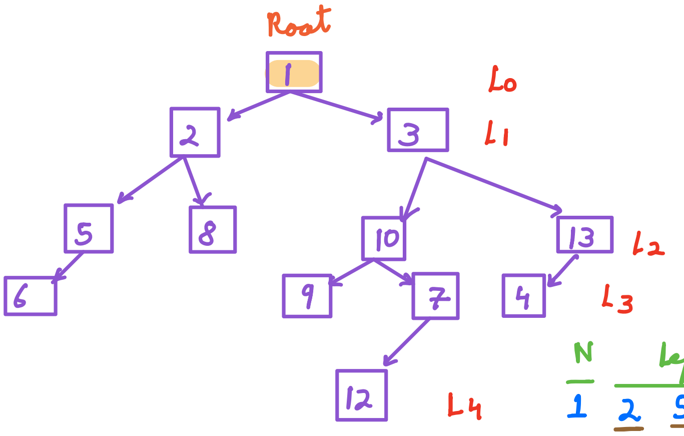
Node a = new Node(1);

data = 1
left = null
right = null

Tree Traversal →

1) Preorder → Node Left Right ∀ nodes
2) Inorder → Left Node Right ∀ nodes
3) Postorder → Left Right Node ∀ nodes





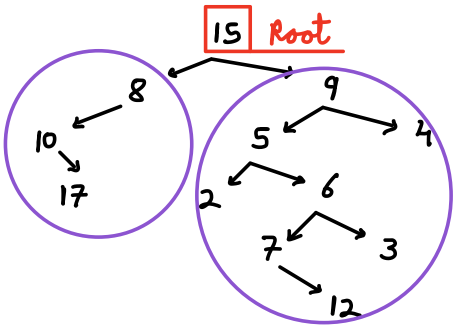
Preorder

Node left Right

N Left Right

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

N L R N L R



15 8 10 17 9 5 2 6 7 12 3 4

Root

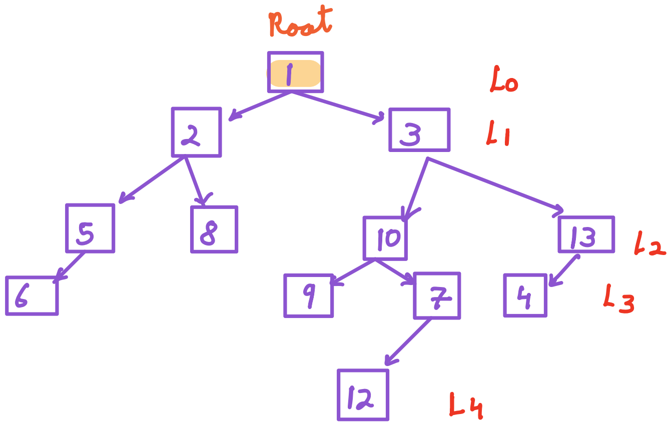
Preorder of left subtree.

Preorder of right subtree.

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

TC = O(N)

SC = O(1) → x → H.W



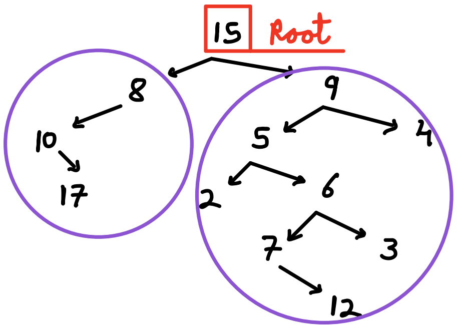
Inorder

left Node Right

left Node Right

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

L N R L N R



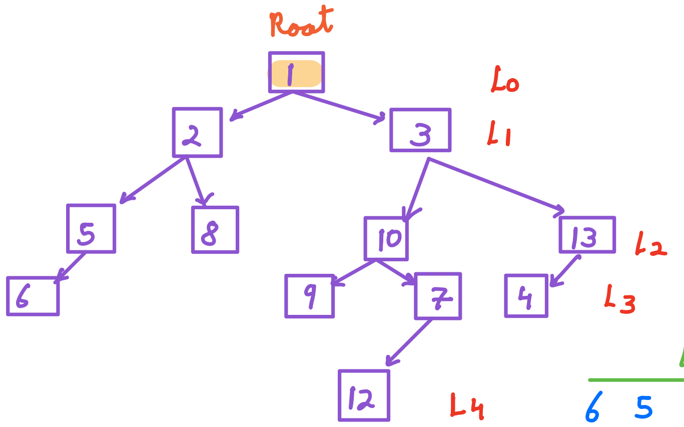
10 17 8 15 2 5 7 12 6 3 9 4

root

Inorder of left subtree.

Inorder of right subtree.

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



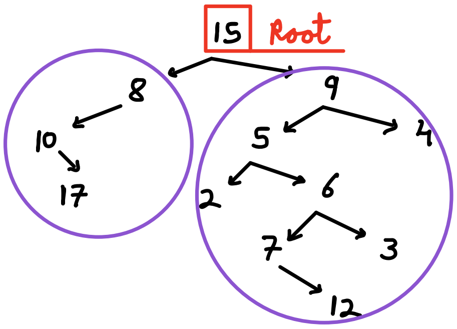
Postorder

left Right Node

L R N

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

L R N L R N



17 10 8 2 12 7 3 6 5 4 9 15

Root

Postorder of left subtree.

Postorder of right subtree.

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