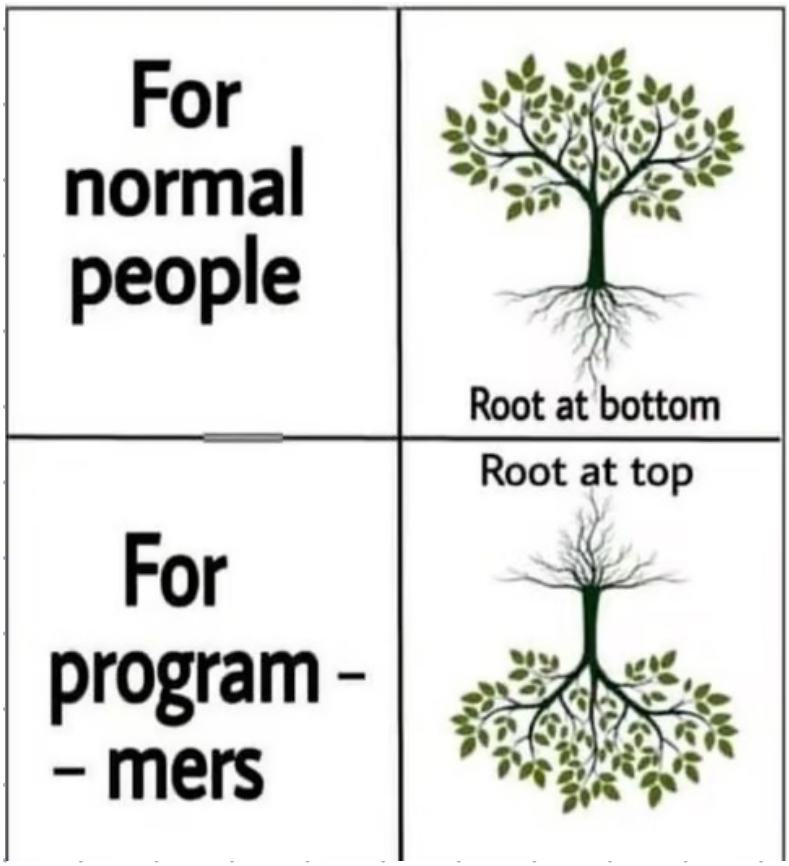
SCALER

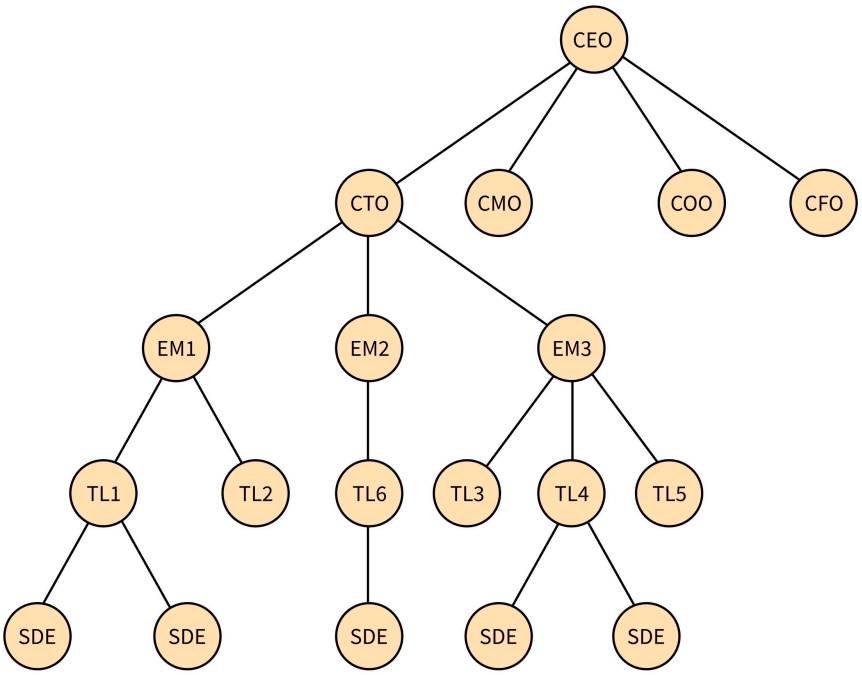
Trees - 1

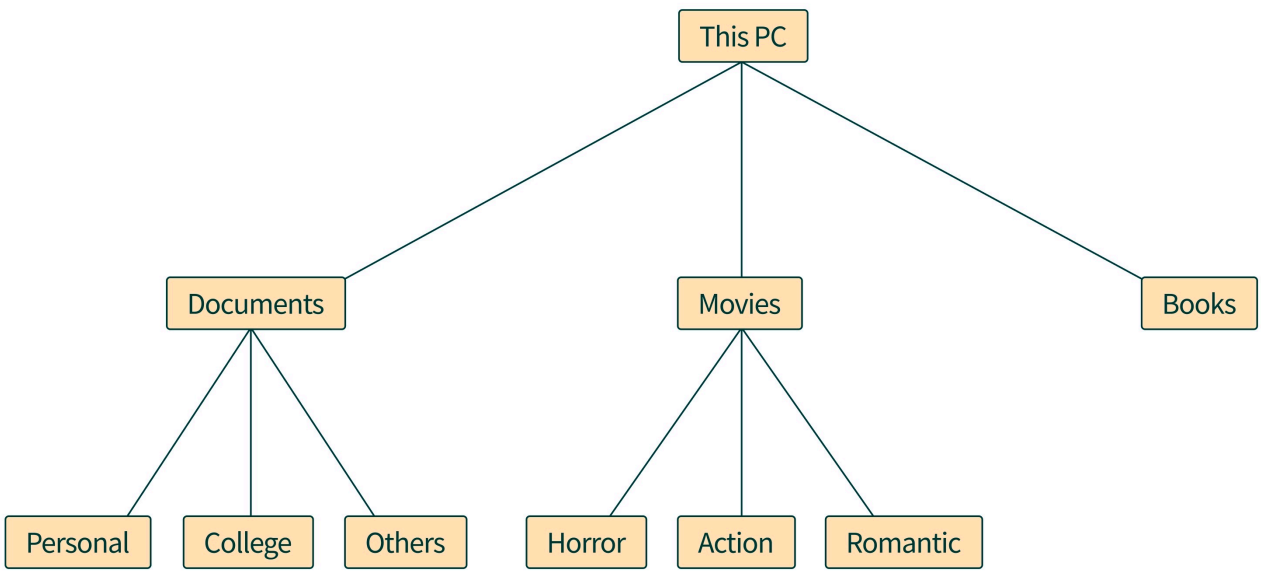
TABLE OF CONTENTS

1. Introduction to Trees
2. Binary Tree
3. Traversal in Binary Tree
   - Pre-order
   - In-order
   - Post-order
4. Iterative In-order
5. Equal Tree Partition



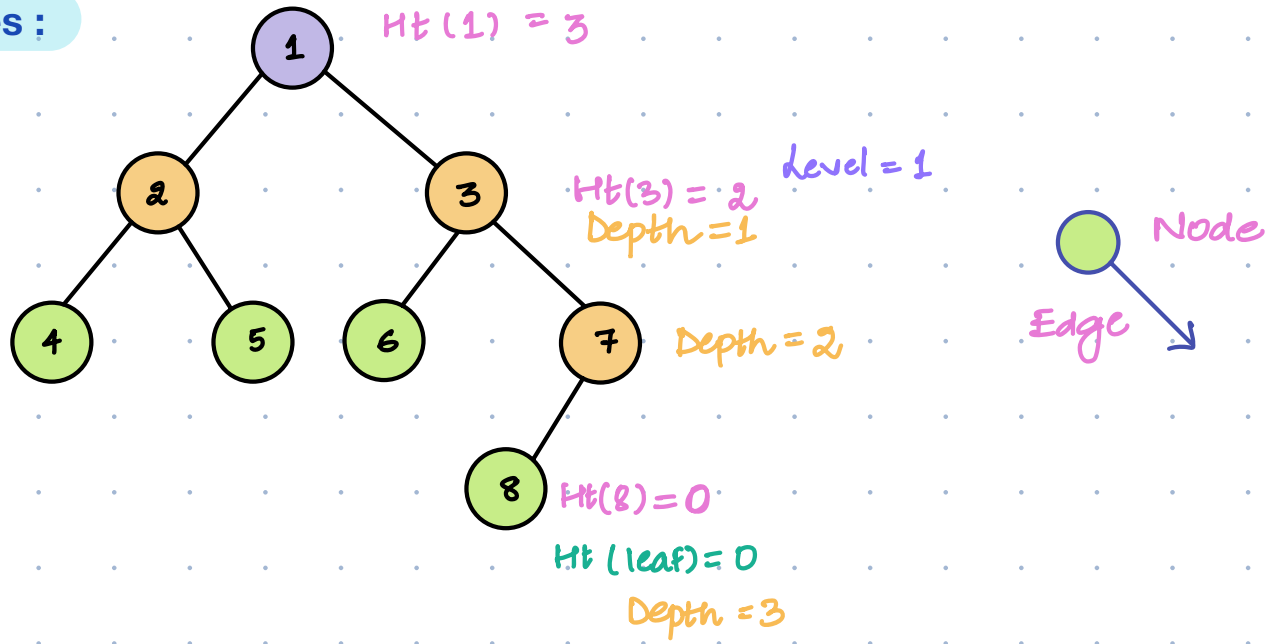
Introduction to Trees :





Trees Data structure which store hierarchical information.

Terminologies :



Node : An element in a tree that contains data and may have child nodes connected to it. 1, 2, 3, 4, 5, 6, 7, 8.

Root : The topmost node in a tree from which all other nodes descend. It has no parent. Node 1 is the root.

Parent : A node that has child nodes connected to it. Nodes 1, 2, 3 and 7.

Child : A node that has a parent node connected to it. Nodes 2, 3, 4, 5, 6, and 7 are children.

Leaf : A node that has no child nodes. It's a terminal node. Nodes 4, 5, 6, and 8 are leaves.

Ancestor : All nodes from parent to the root node upwards are the ancestors of a node.

Nodes 1, 3, 7 are ancestors of node 8.

Descendant : All nodes from child to the leaf node along that path. Nodes 4 and 5 are descendants of node 2.

Siblings : Nodes that share the same parent node. Nodes 2 and 3 are sibling

Height of a Node : Maximum distance (in terms of edges) between node and its descendant leaf node.

Ht: Node to leaf ↓

Depth: Node to Root ↑

Height of Tree : Height of root node.

Height of the tree is 3, which is the number of edges from the root to a farthest leaf (8).

Depth : No of edges node is away from its root note.

The root is at depth 0. Depth of node 1 is 0, 2 and 3 are at depth 1, 4, 5, 6, and 7 are at depth 2, and 8 is at depth 3.

Subtree : Node along with all its descendant, which is part of the original tree.

Subtree rooted at node 2 consists of nodes 2, 4, and 5.

QUIZ Can a leaf node also be a subtree? Yes

QUIZ Do all nodes have a parent node? No

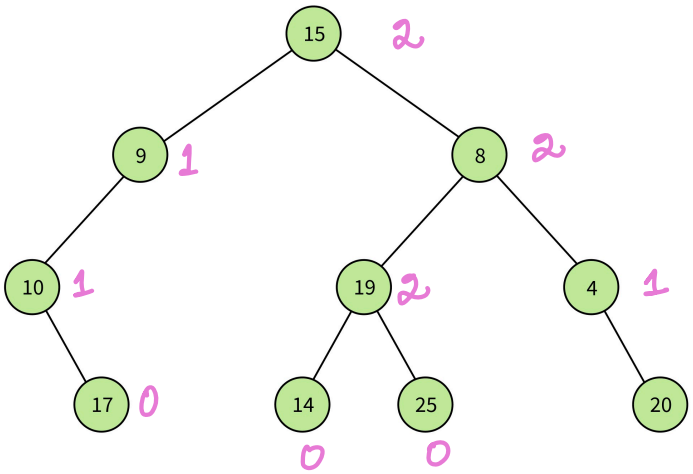
QUIZ What is the height of the leaf node in any tree?

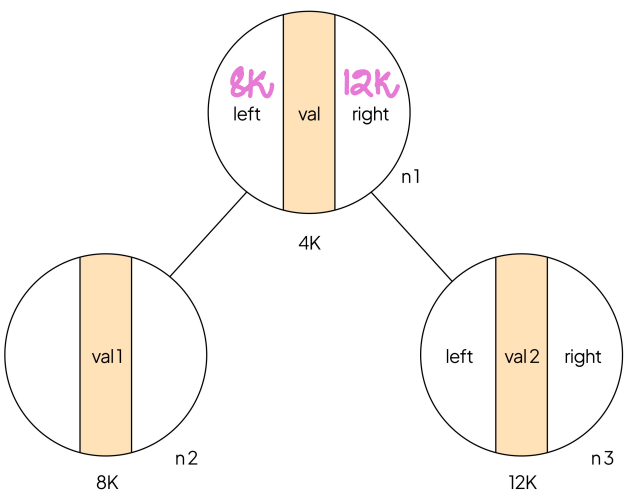
ht(leaf) = 0.

Binary Tree

every node can have atmax 2 children

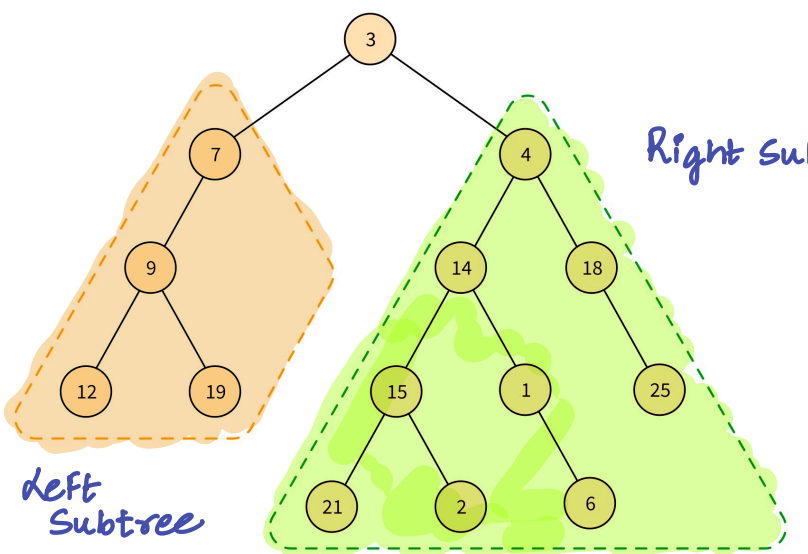
[0, 1, 2]






```
class Node {
    int val;
    Node left;
    Node right;
    Node(int x) {
        val = x;
        left = NULL;
        right = NULL;
    }
}
```

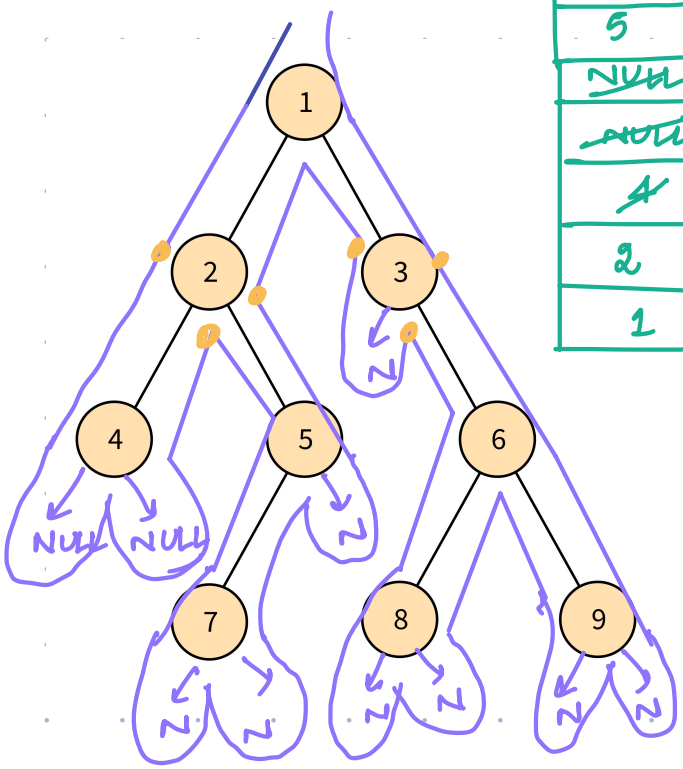
Sub-tree



Binary Tree Traversal

1. Pre-order Traversal

N L R
↓
print



o/p: [1 2 4 5 7 3 6 8 9]

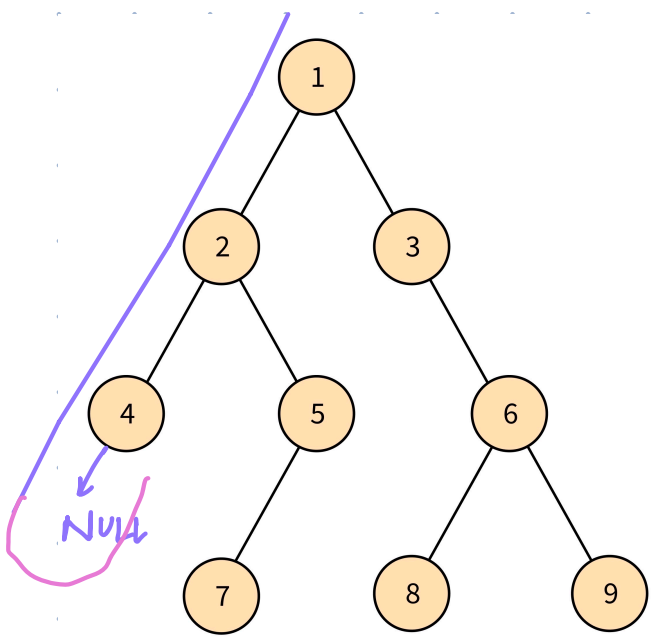
1 → Node

2 4 5 7 → Preorder of LST

3 6 8 9 → Preorder of RST.

Preorder of a tree = Print(Node.val) + Preorder of LST + Preorder of RST.

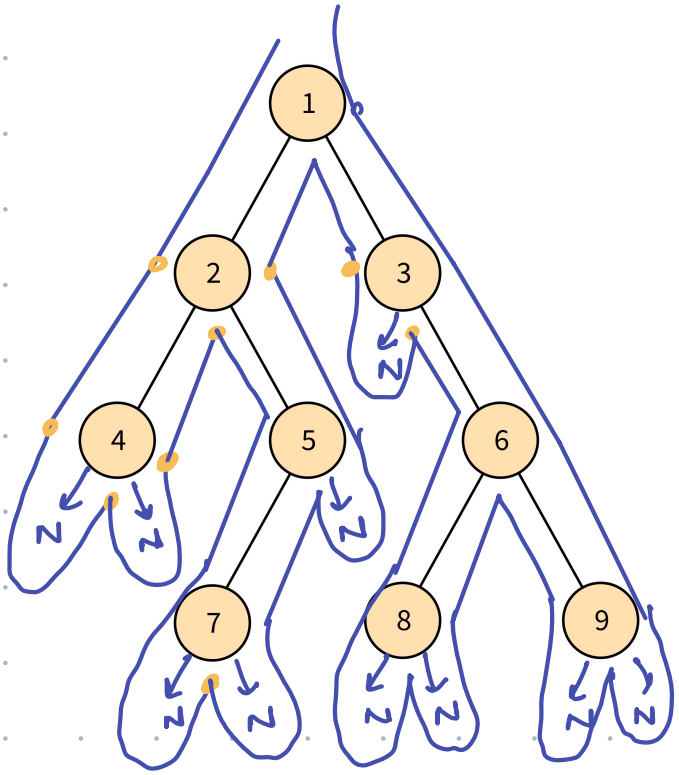
```
void Preorder(Node root) {
    if (root == NULL) return;
    print(root.val);
    Preorder(root.left);
    Preorder(root.right);
}
```



T.C : O(N)
S.C : O(ht of tree)

2. In-order Traversal

L N R
↓
print



o/p : [4 2 7 5 1 3 8 6 9]

Inorder of LST — Node — Inorder of RST

Inorder of a tree = Inorder of LST + Print (root.val) + Inorder of RST.

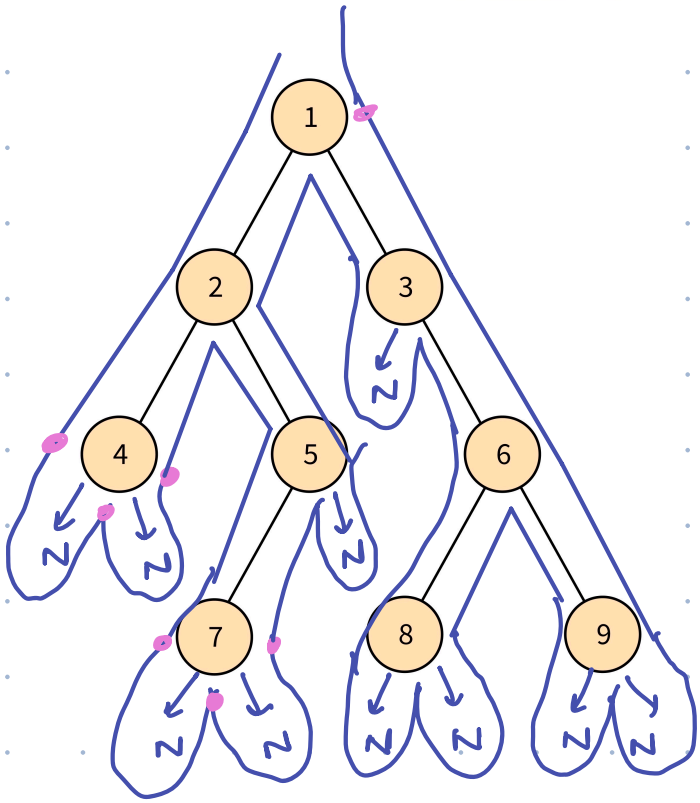
```
void Inorder(Node root) {
    if (root == NULL) return;
    Inorder (root.left);
    print (root.val);
    Inorder (root.right);
}
```

T.C : O(N)

S.C : O(ht of tree)

3. Post order Traversal

L R N → print



o/p : [4 7 5 2 | 8 9 6 3 | 1]

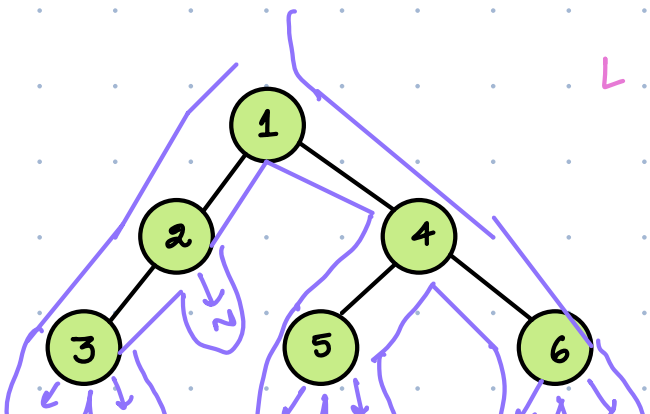
- 4 7 5 2 : Postorder of LST
- 8 9 6 3 : Postorder of RST
- 1 : Node

Postorder of a tree = Postorder of LST + Postorder of RST + print(node.val)

```
void Postorder(Node root) {
    if (root == NULL) return;
    Postorder(root.left);
    Postorder(root.right);
    print(root.val);
}
```

T.C : O(N)
S.C : O(ht of tree)

QUIZ

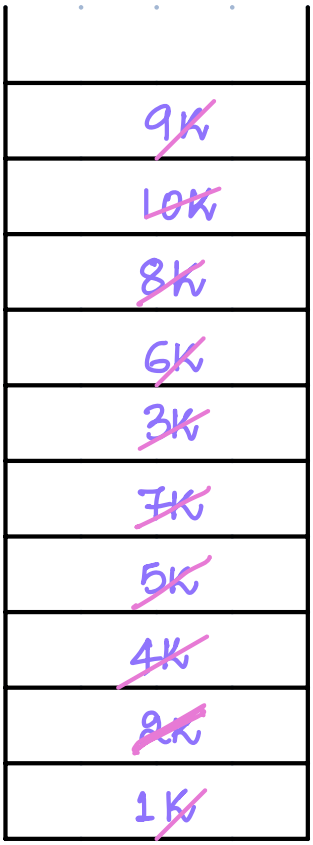


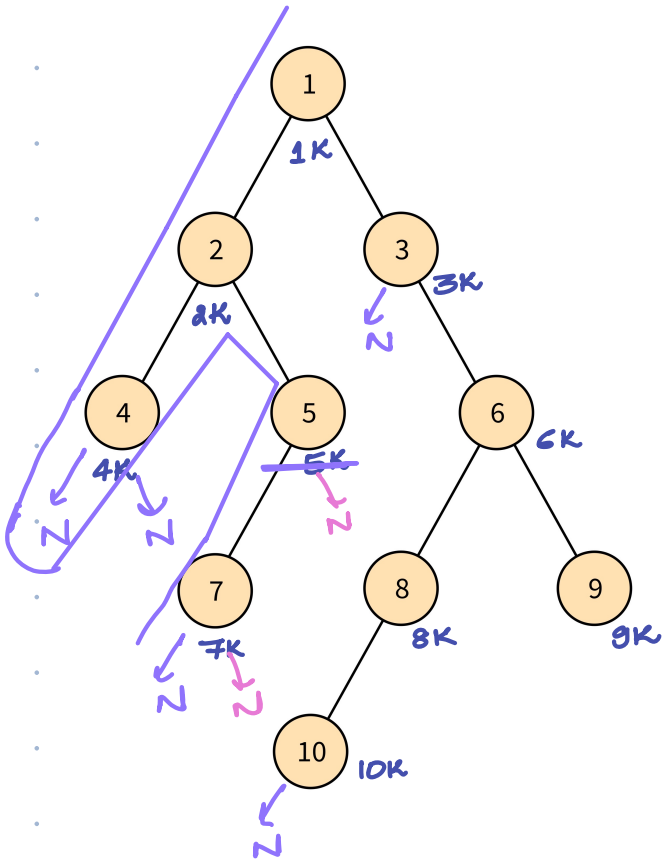
L N R

o/p : [3 2 1 5 4 6]

Iterative In-order Traversal

L N R






```
Node curr = root;
Stack<Node> st;
while(st.size()>0 || curr!=NULL){

    while(curr!=NULL){
        st.push(curr);
        curr = curr.left;
    }

    curr = st.pop();
    print(curr.val);
    curr = curr.right;
}
```

if (curr == NULL)

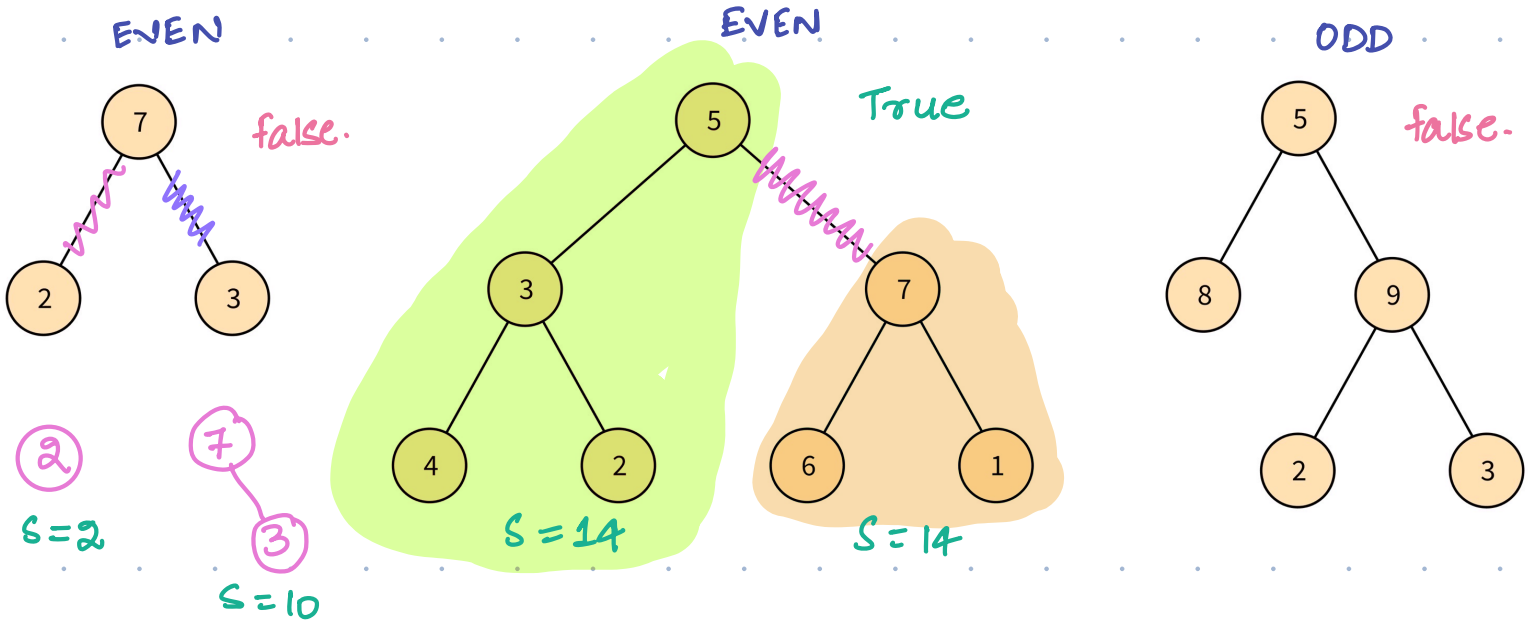
yes
- Pop from Stack
- Print value of that node
- Go to right child.

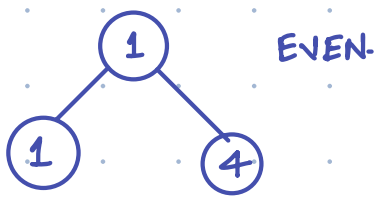
No
- Push it to stack
- Go to left child

T.C : O(N)
S.C : O(ht of tree)

Equal tree Partition

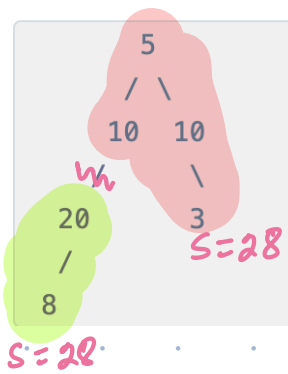
- Check if it is possible to remove an edge from the given binary tree such that sum of resultant two trees is equal.





QUIZ

Check whether the given tree can be split into two non-empty subtrees with equal sums or not.



True.

Observations :

- If total sum of BT is ODD → ans : false
  " " " " " " EVEN → May or Mayn't be possible
  Need to check.

- One of the tree out of 2 resultant trees will be a SUBTREE.

  If total sum = t_s
  Sum of each tree = $t_s/2$
  ∴ We are looking for a subtree with sum = $t_s/2$

</> Code

```
bool EqualPartition (Node root) {
    if (root == NULL) return false;

    // Calculate total sum of tree.
    int ts = TreeSum (root);
    if (ts % 2 != 0) return false;

    // check if there is any subtree with sum = ts/2.
    bool ans = false;
    Sum (root, ans, ts);
    return ans;
}
```

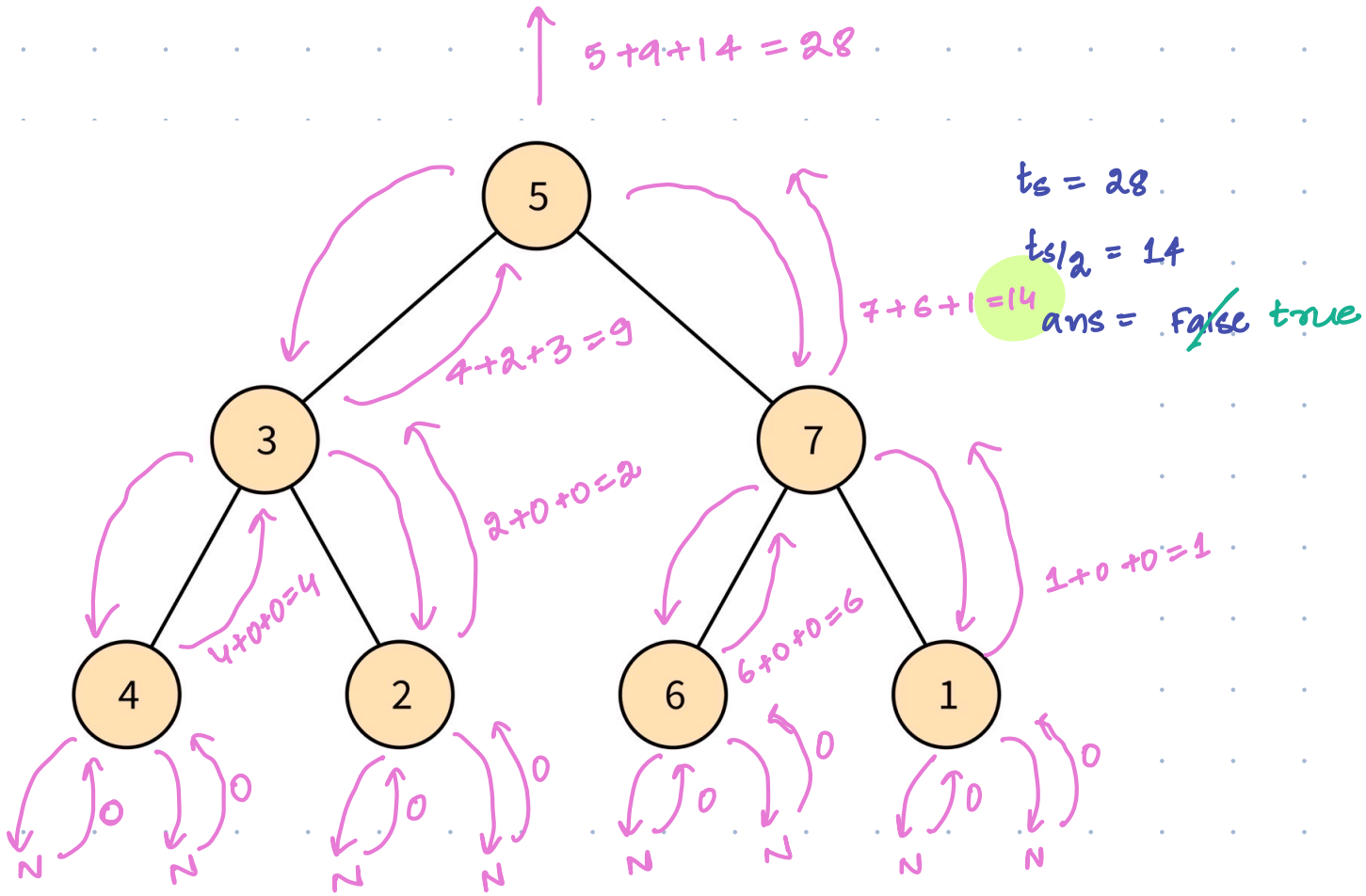
```
// Calculate total sum of tree
int TreeSum(Node root){
    if(root == NULL) return 0;
    return root.val + TreeSum(root.left) + TreeSum(root.right);
}

// Calculating sum of subtree rooted at Node = root.
int Sum(Node root, bool &ans, int totalsum){
    if(root == NULL) return 0;

    int lsum = Sum(root.left, ans, totalsum);
    int rsum = Sum(root.right, ans, totalsum);

    // Check if any subtree has sum = totalsum/2.
    if(lsum == totalsum/2 || rsum == totalsum/2){
        ans = true;
    }
    return root.val + lsum + rsum;
}
```

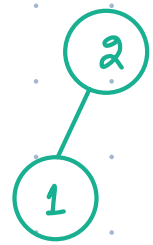
T.C : O(N)
S.C : O(ht of tree)



$t_s = 28$

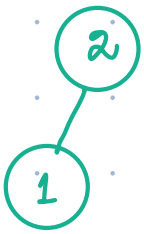
$t_s/2 = 14$

ans = ~~false~~ true



Ht in terms of Nodes :

Ht = 1



Ht in terms of edges :

Ht = 0