AGENDA

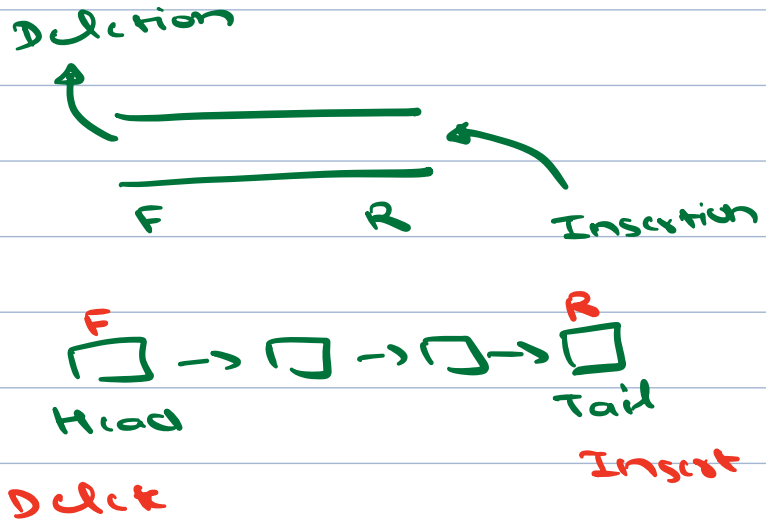
- What is Tree?
- Terminology
- Traversal
- Inorder Traversal (Iterative)
- Equal Tree Partition

Revision Quiz 1

Queue example : People standing in line at ticket counter

Revision Quiz 2

Queue using LL :



Linear Data structures

Access data in sequential order

↓

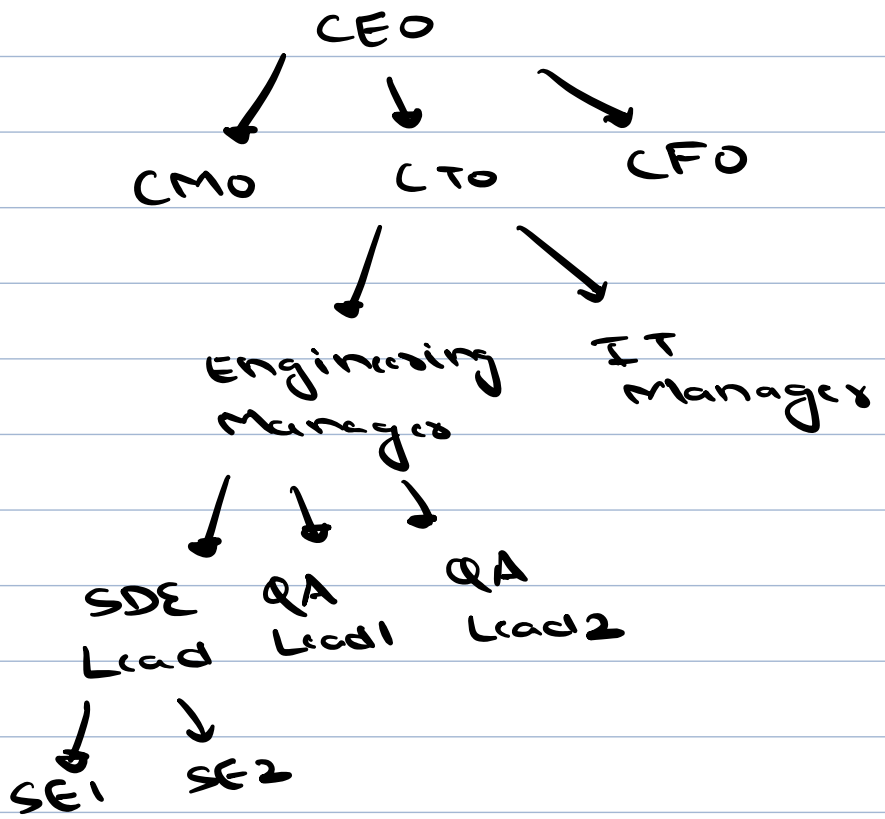
Array, LL, stack, Queues etc.

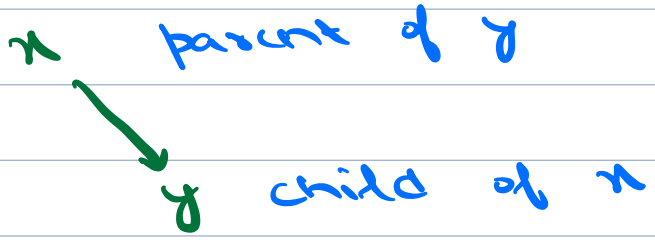
Tree : Fav topic among interviewers

Why ? 1. Requires solid recursion understanding
2. Ability to visualize concepts

Tree is a non-linear data structure used to represent hierarchy in information.

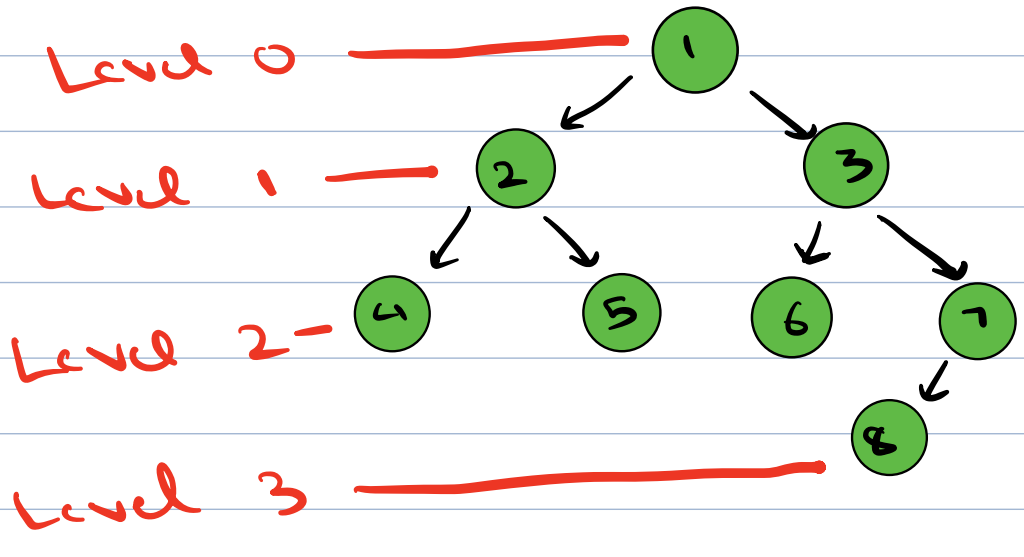
For eg. Family Tree, Org Structure, Folder structure

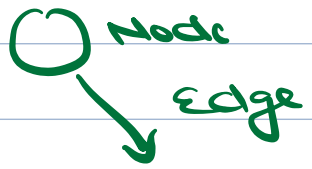




Tree Naming Convention

Node: element of tree with data and nodes are connected to it.





Root: Topmost node of a tree from which all other nodes descend. It has no parent.
1 is root node

Parent: A node that has child node connected to it. Parent Node → 1, 2, 3, 7

Child: A node that has a parent node connected to it. Every node except root

Leaf: Node with 0 children. It's a terminal node. Node 4, 5, 6 and 8

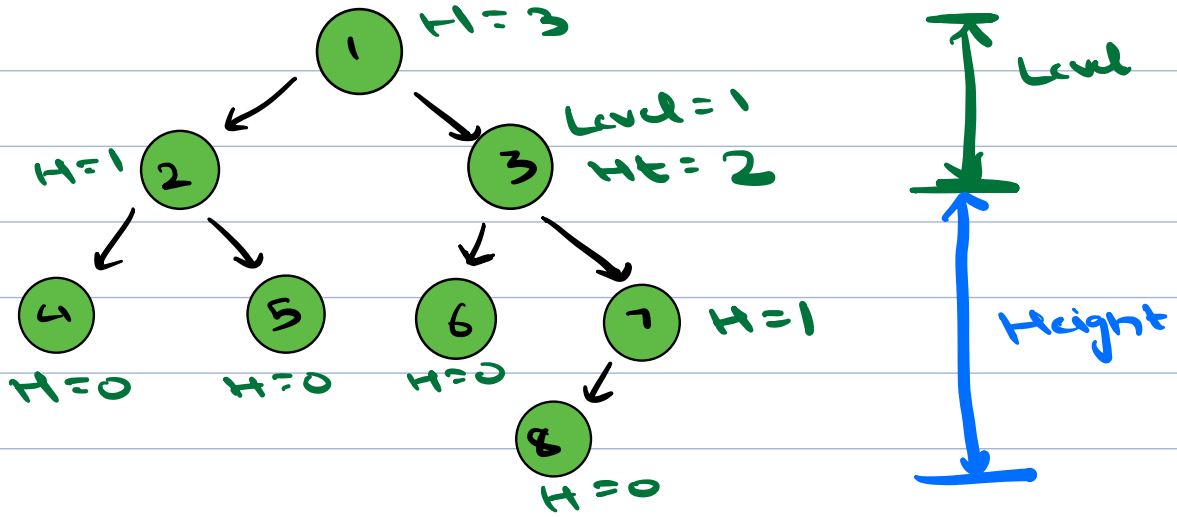
Depth / level: Distance from root node (in terms of edges)

Height: Length of longest path from a node to a leaf.

Height of tree = Height of root node

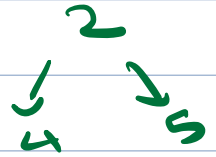
1 child
↓
1 par

1 par
↓
Multiple children

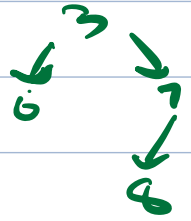


Subtree: A tree structure which is part of a larger tree

Subtree rooted at 2



Subtree rooted at 3



Siblings: Nodes that share same parent node.
P → 2; 4 and 5 are siblings.

Ancestor: All nodes from parent to root upwards are ancestors of a node.
8 : 1, 3, 7 are ancestors.

Descendant: All nodes from child to leaf node along all the paths.

Can a leaf node also be a subtree?

YES

Do all nodes have a parent node?

NO (Everyone has parent except root)

Height of leaf node

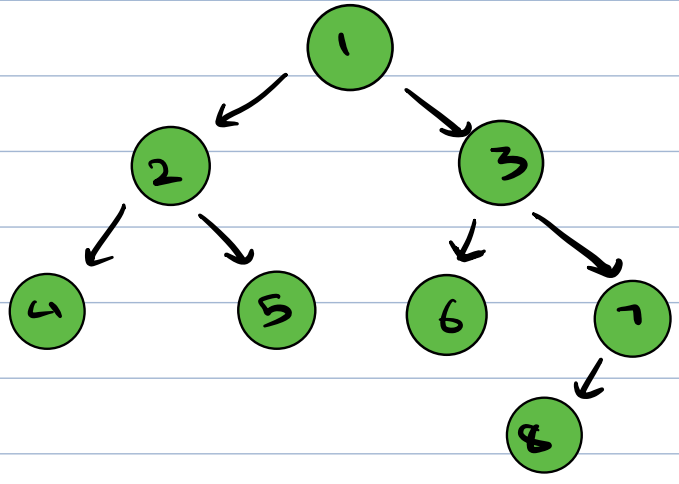
Length of path from leaf to leaf → 0

H → 0

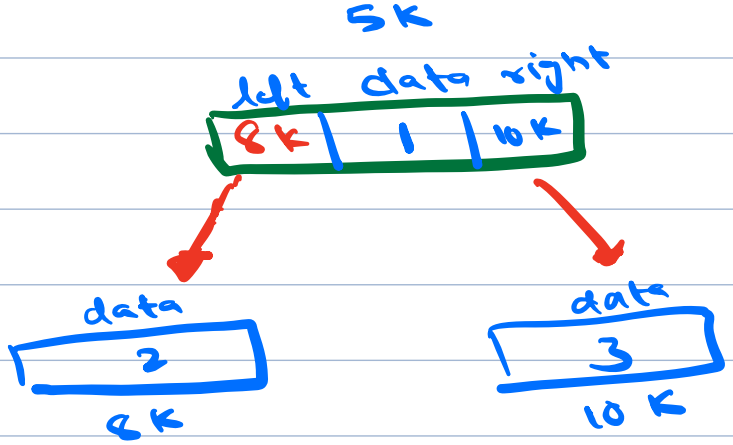
Binary Tree

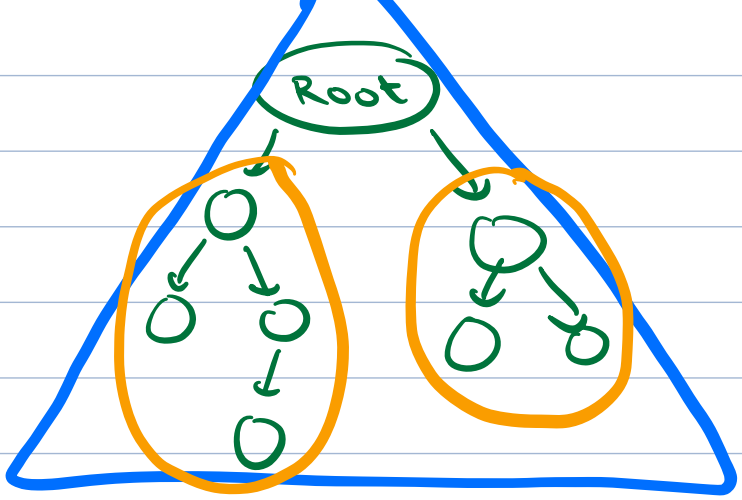
A type of tree in which each node can have atmost 2 children i.e. either 0, 1, 2.

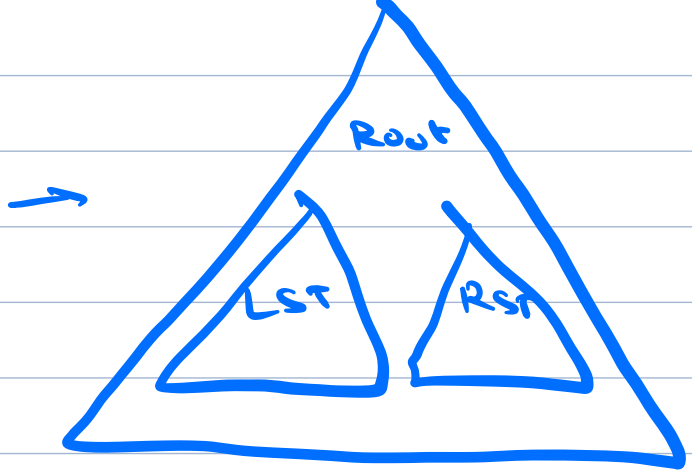
They are referred as left and right child




```
class TreeNode {
    int data
    TreeNode left
    TreeNode right

    TreeNode(int x) {
        data = x
        left = NULL
        right = NULL
    }
}
```







How can we traverse a tree? Depth First Search (DFS)

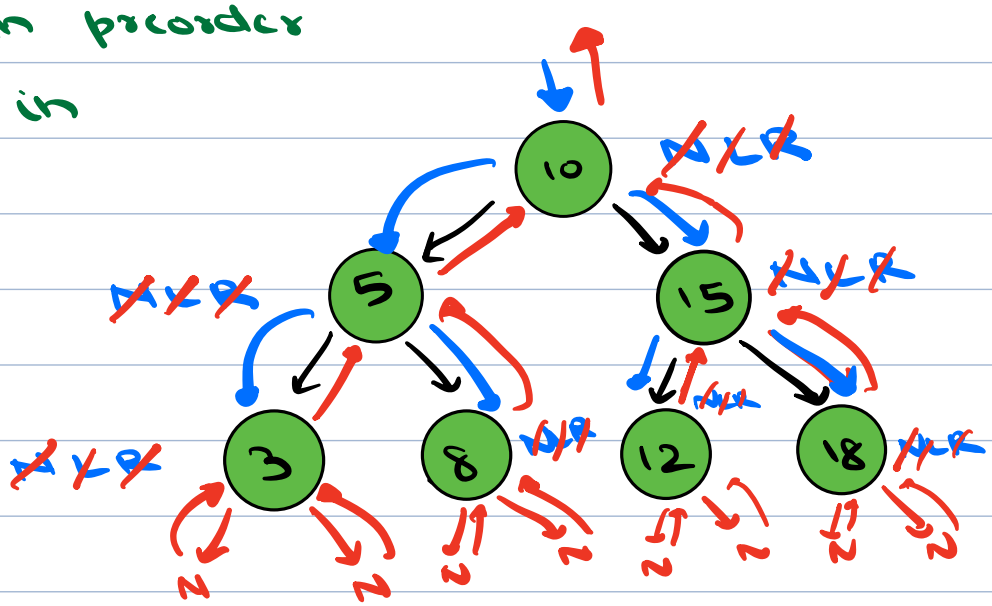
L → Left subtree

R → Right subtree

N → Node

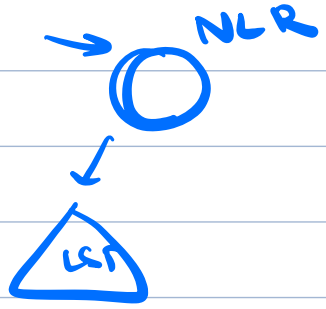
Preorder → NLR

1. Visit the node
2. Left subtree in preorder
3. Right subtree in preorder



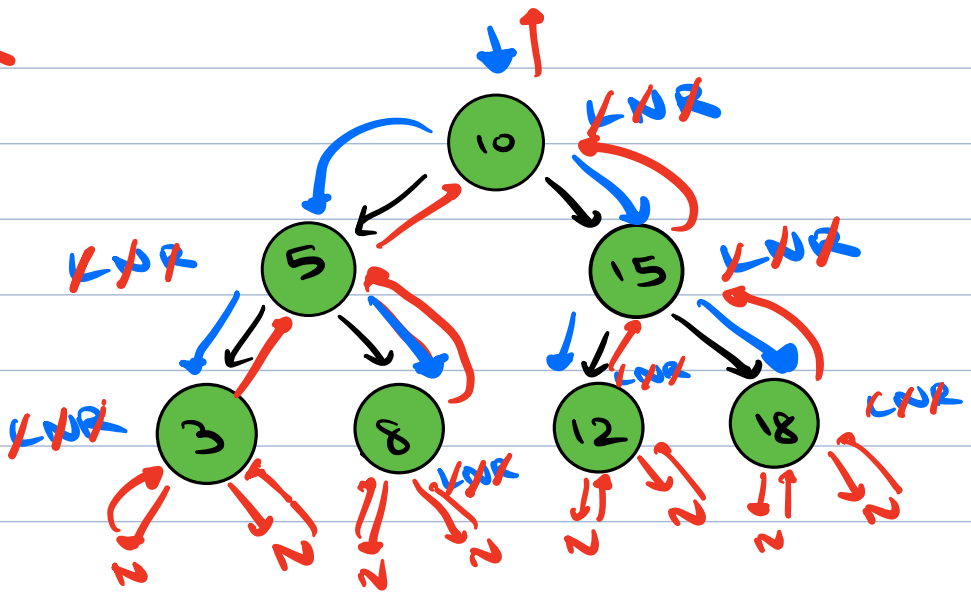
O/P: 10 5 3 8 15 12 18

// Given root node as I/P, fn will visit nodes in preorder




```
void preorder (TreeNode root) {
    if (root == = NULL)  return
    print (root.data)
    preorder (root.left)
    preorder (root.right)
}
```

Inorder : L N R

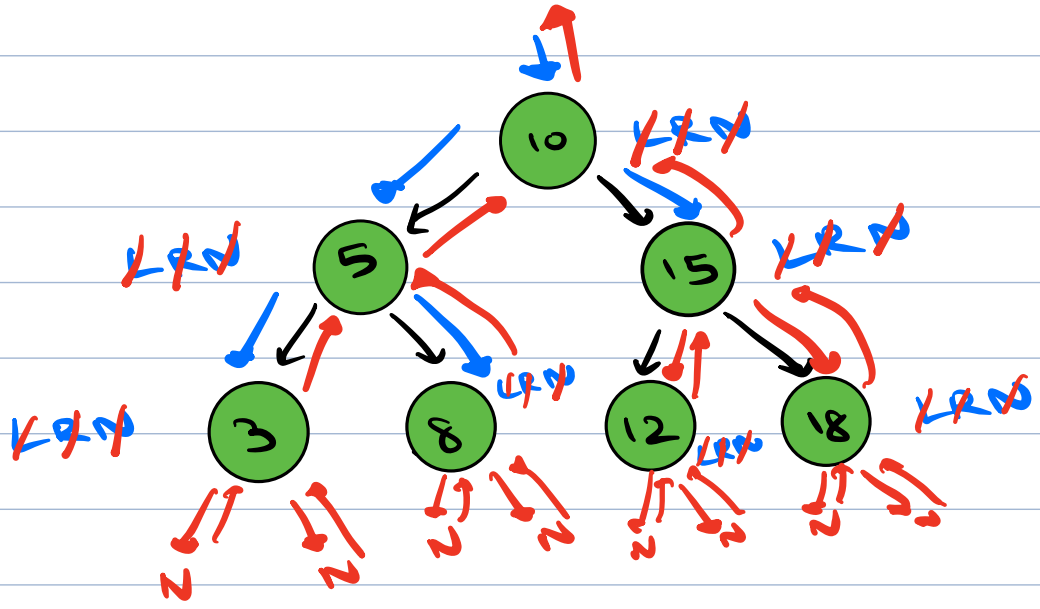


O/P : 3 5 8 10 12 15 18

```
void inorder (TreeNode root) {
    if (root == = NULL)  return
    inorder (root.left)
    print (root.data)
    inorder (root.right)
}
```

L N R

Postorder : L R N

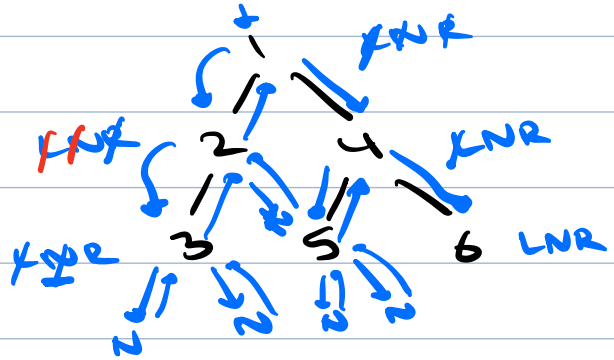


O/P : 3 8 5 12 18 15 10

```
void postorder (TreeNode root) {
    if (root == NULL) return

    postorder (root.left)
    postorder (root.right)
    print (root.data)
}
```

LRN

Inorder : LNR



Preorder : N L R

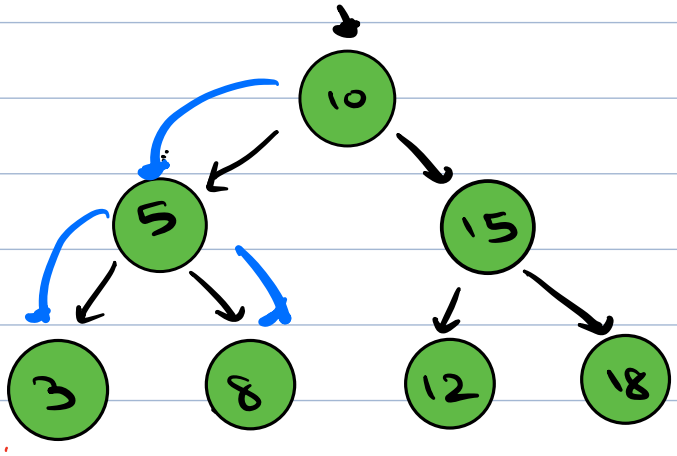
Postorder : L R N

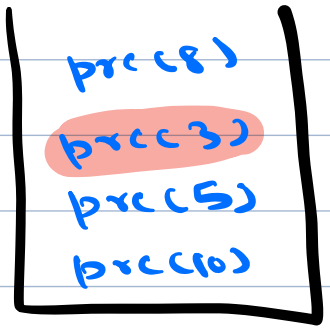
Inorder : L N R

O/P: 3 2 1 5 4 6

10:39

Preorder : NLR



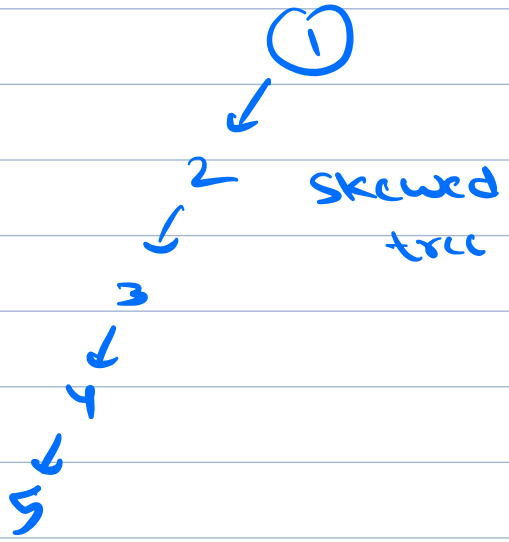


TC : O(N)

At any point, nodes of 1 path are on the stack. We are exploring node path by path.

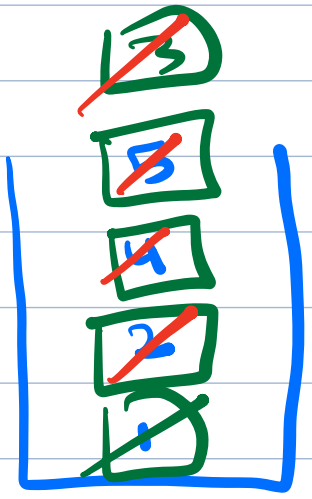
SC : O(H) → height of tree

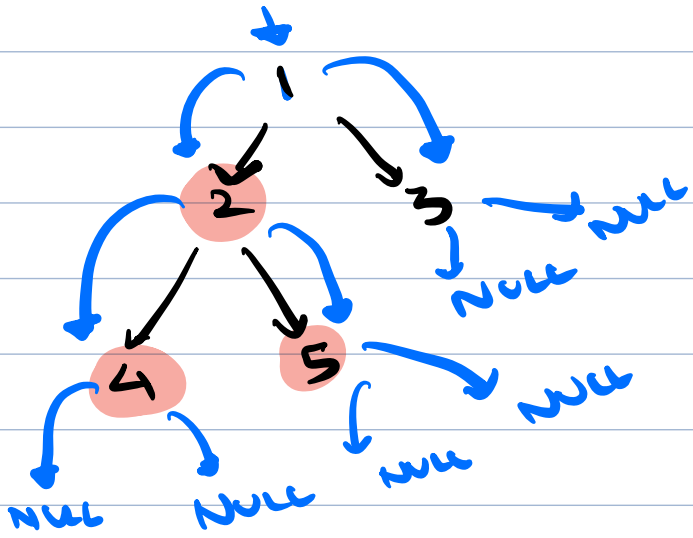
$\log_2 N$ N



Iterative Inorder Traversal

LNR





NULL : Take out a node from stack

1st : Push in stack, go to left

o/p : 4 2 5 1 3

STOP : cur == NULL and st.size() == 0

```
Stack <Node> st
Node cur = root

while ( cur != NULL || !st.isEmpty()) {
    if (cur != NULL) {
        st.push(cur)
        cur = cur.left
    }
    else {
        Node t = st.top()      st.pop()
        print (t.data)
        cur = t.right
    }
}
return
```

TC : O(N)

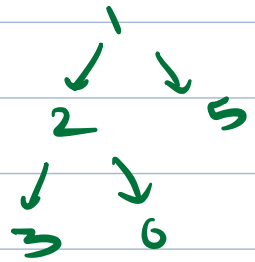
SC : O(H)

↓

Stack

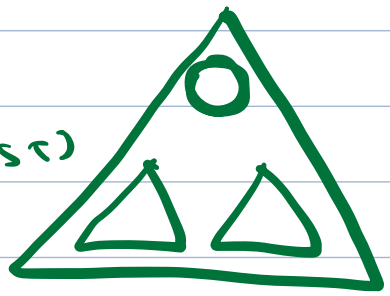
Iterative Preorder and Postorder

Q. Given a tree, find sum of tree



ans = 17

Sum (tree) = Sum (LST) + Sum(RST) + root.data



//Given a root node, this fn will sum up all the nodes in tree rooted at 'root'

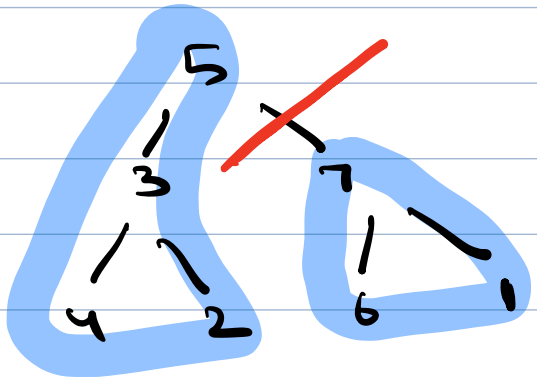
```
int Sum (TreeNode root) {
    if (root == NULL) return 0
    return Sum(root.left) + Sum(root.right)
           + root.data
}
```

TC : O(N)

SC : O(H)

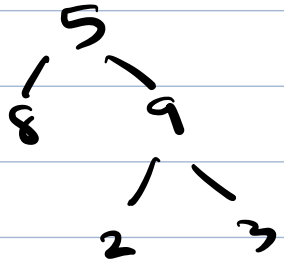
Equal Tree Partition

Q. Given a binary tree, check if you can partition the tree into 2 trees of equal sum.



total = 28

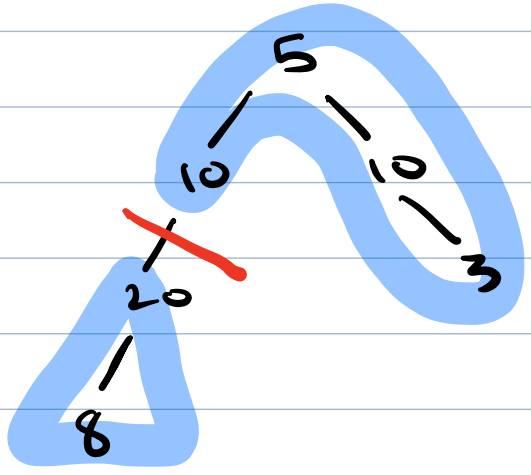
ans = true



total = 27

ans = false

If totalsum is odd, ans = false



totalsum = 56

ans = true

1. Calculate totalsum
2. If (totalsum % 2 == 1) return false
3. Look for a piece of tree →

$$sum = \frac{totalsum}{2}$$

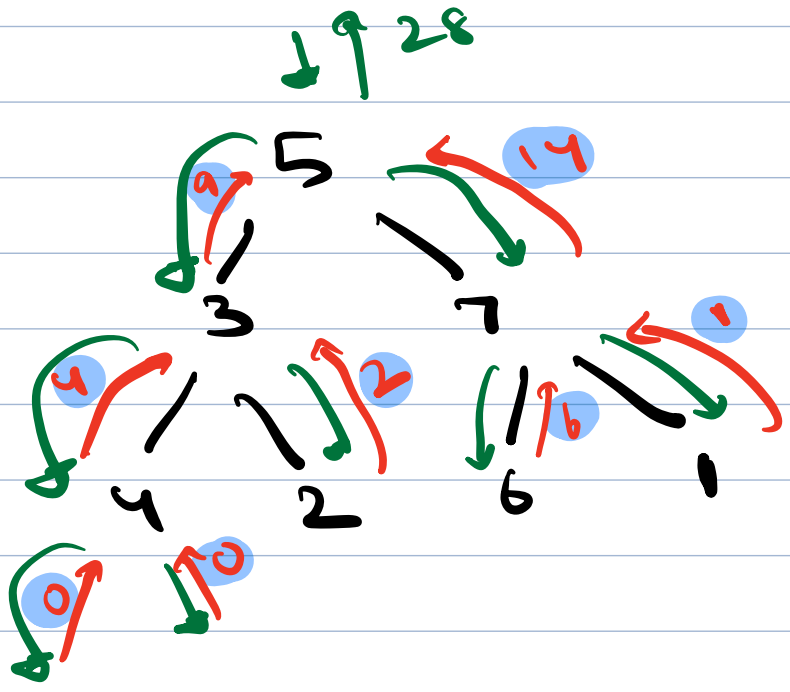
↓

Look for any subtree with $sum = \frac{totalsum}{2}$

```
int desiredSum = INT_MIN
bool ans = false

int Sum (TreeNode root) {
    if (root == NULL) return 0

    int subtree_sum = sum(root.left)
                    + sum(root.right) + root.data

    if (subtree_sum == desiredSum) {
        ans = true
    }

    return subtree_sum
}
```

main() {

```
int totalSum = Sum(root)
if (totalSum % 2 == 1)
        return false

desiredSum = totalSum/2
   Sum(root)
   return ans
}
```



TC: O(N)

SC: O(H)

↓

height of tree

desiredsum = ~~6~~ 14

ans = ~~F~~ T

totalsum = 28