SCALER

Trees 2

TABLE OF CONTENTS

Arunava Basak

Manohar A N

ANUBHAV RAMNANI

Uddeepta Saikia

Surabhi Kumari

Pranjul Kesharwani

SHIVAM SHIV

M S Haseeb Khan

Ravi Dalal

Mahesh Baswaraj

Amreshwar

Subhash

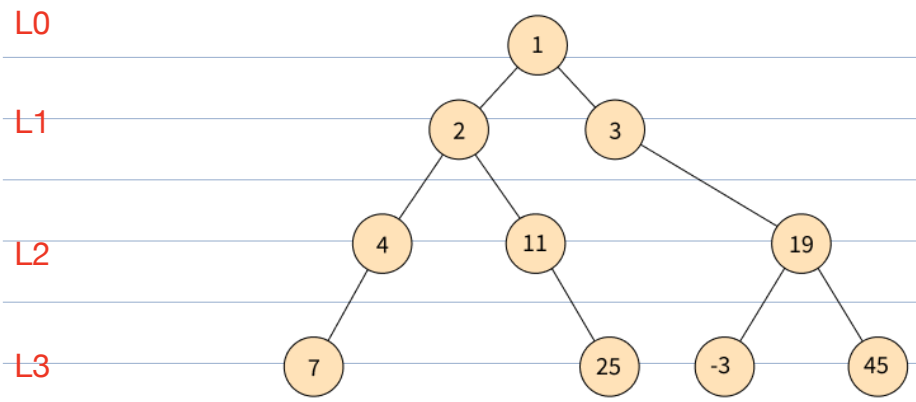
Rules

Q ⟶ QT

A ⟶ PC

100% active

Level Order Traversal



Output

1

2 3

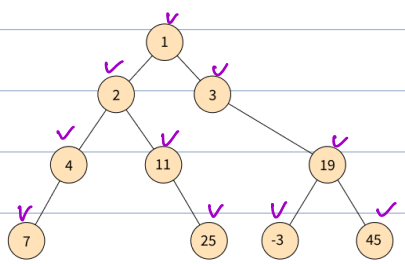
4 11 19

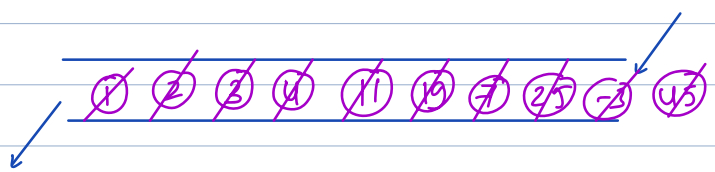
7 25 -3 45

Array List of Array List

HM of Array List

NOTE: Since we want to print in a FIFO manner we can use queue DS





1 2 3 4 11 19 7 25 -3 45

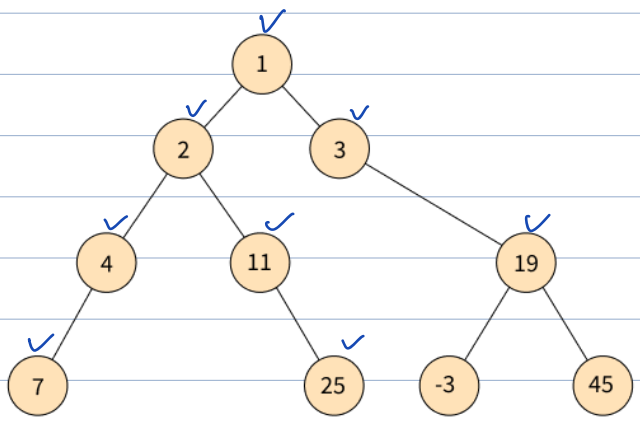
NOTE: Alongside node we also store the depth/level info

</> Code

```
// Instead of printing List of List<Integer>

class pair {
    Node node;
    int depth;
}

// 0 ---> [1 2 3]   (depth or level -> values in that level)

Map<Integer, ArrayList<Integer>> levels //

queue //          depth   node
queue.enqueue([0, root])
maxdepth = 0
while (!queue.isEmpty()) {
    Pair p = queue.dequeue()
    node = p.node
    depth = p.depth                          //Java
    maxdepth = max(maxdepth, depth)          new pair(..., ...)
    levels[depth].add(node.val)
    if (node.left != null) {
        queue.enqueue([depth+1, node.left])
    }
    if (node.right != null) {
        queue.enqueue([depth+1, node.right])
    }
}
```

TC: O(N) SC: O(N)

```
for d ⟶ 0 to max depth
    al = Levels.get(d)
    print(al)
}
```




```
{
0 : [1]
1 : [2, 3]
2 : [4 11 19]
3 : [7 25 -3 45]
}
```

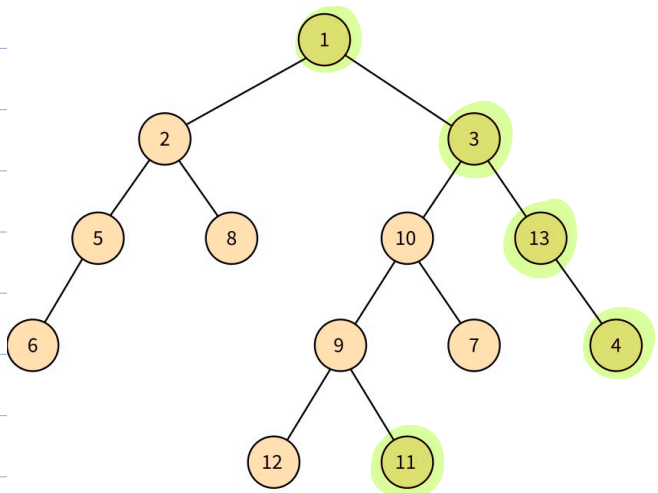
3

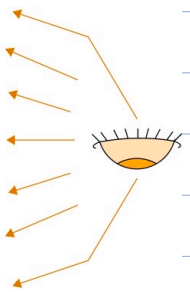
queue

[0, 1] [1 2] [1, 3] [2 4] [2 11] [2, 19]

[3 7] [3 25] [3 -3] [3 45]

< *Question* > : Find right view of binary tree.



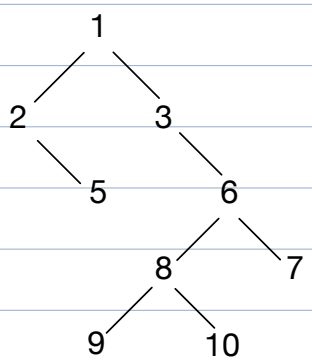


O/p = 1 3 13 4 11

last node value of each level in me ary

1
2 3
5 8 10 13
6 9 7 4
12 11

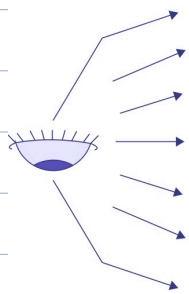
Quiz :

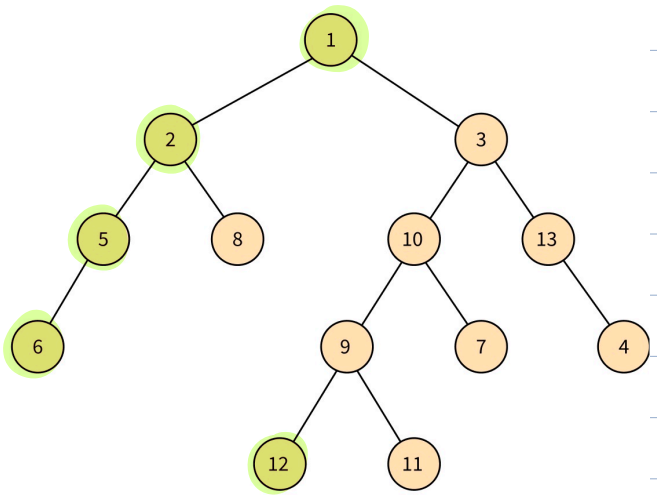


1
2 3
5 6
8 7
9 10

1 3 6 7 10

< Question > : Find left view of binary tree.





Op 1 2 5 6 12

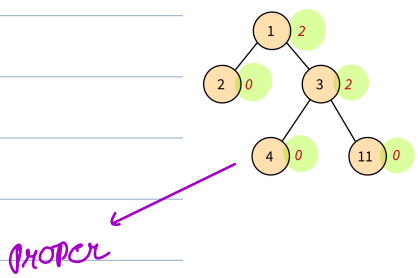
Idea DO LOT { level order traversal } and first node value at each level is the ans.

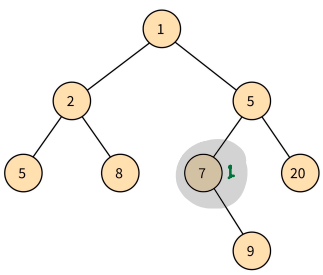
HW Try to think of top and bottom view.

Types of Binary Tree [Structure]

1. Proper/ Full Binary Tree

Every node has either 0 or 2 children.



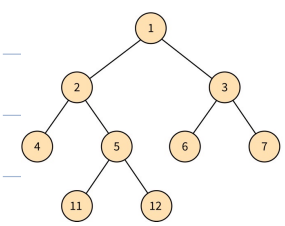


X

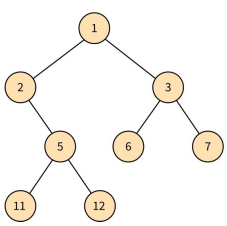
2. Complete Binary Tree (C.B.T)

All levels must be completely filled except possibly the last level

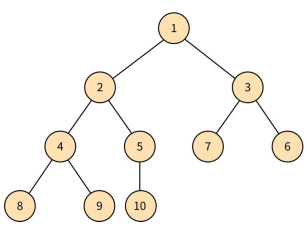
and the last level must be filled from left to right.



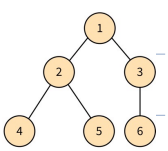
X



X



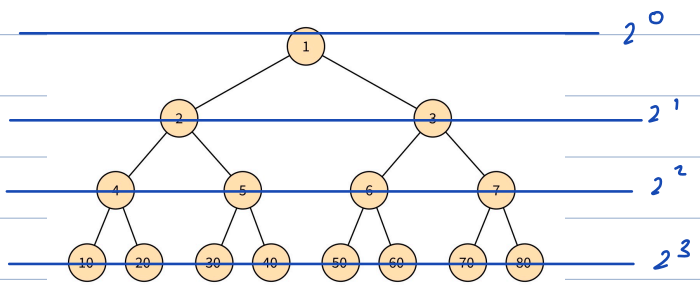
✓

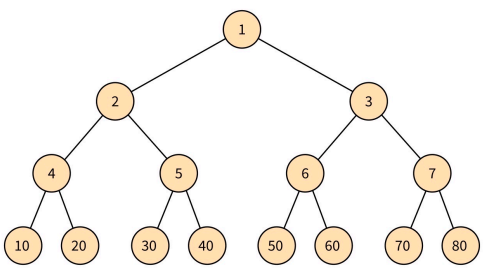


✓

3. Perfect Binary Tree

All levels are completely filled.





Is my complete ? ✓

Is this proper ? ✓

0 or 2 children

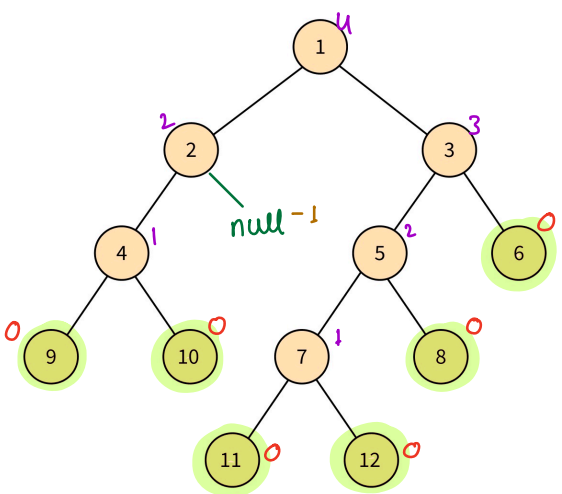
Both

Balanced Binary Tree

For all nodes

ht. of left child	-	*ht. of right child*	≤ 1

< *Question* > : Given a binary tree, check if it is balanced or not.



L R N

post order

Height = distance or no. of edges to the fortnest leaf.

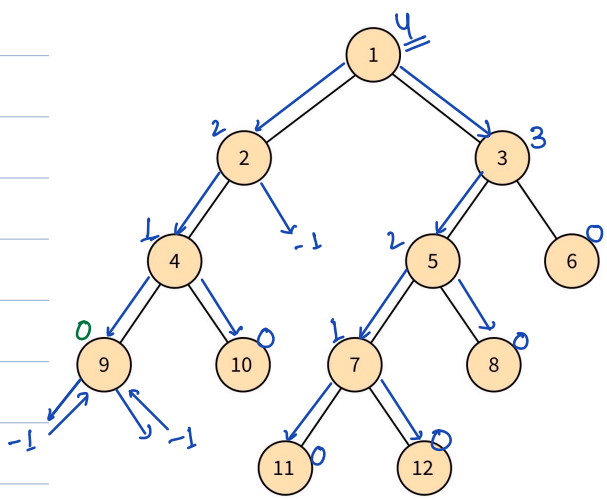
< / > Code

```
isBalanced = true        // global variable
int getTreeHeight ( TreeNode root ) {
    if ( root == null ) return -1

    int leftH = getTreeHeight ( root.left)
    int rightH = getTreeHeight ( root.right)

    if ( abs ( leftH - rightH) > 1 ) { isBalanced = false }

    return max ( leftH, rightH) + 1
}
```

TC: O(N)

SC: O(H)

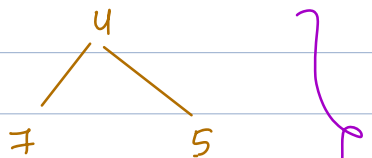
Break : 22:43



isBalanced = false

Q> Construct binary tree from inorder & postorder
All the values are distinct

Input



Inorder 7 4 5
Post 7 5 4

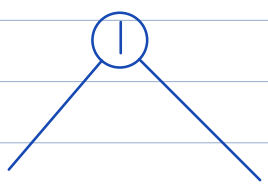
Last element in postorder will give us the root node val.

L N R Inorder

4 2 7 5 | 1 | 3 6
N
L R

L R N Postorder

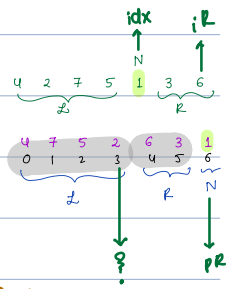
4 7 5 2 | 6 3 | 1
0 1 2 3 | 4 5 | 6
L R N



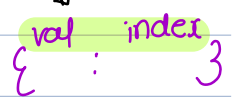
4 2 7 5 | 3 6

4 7 5 2 | 6 3
0 1 2 3 | 4 5

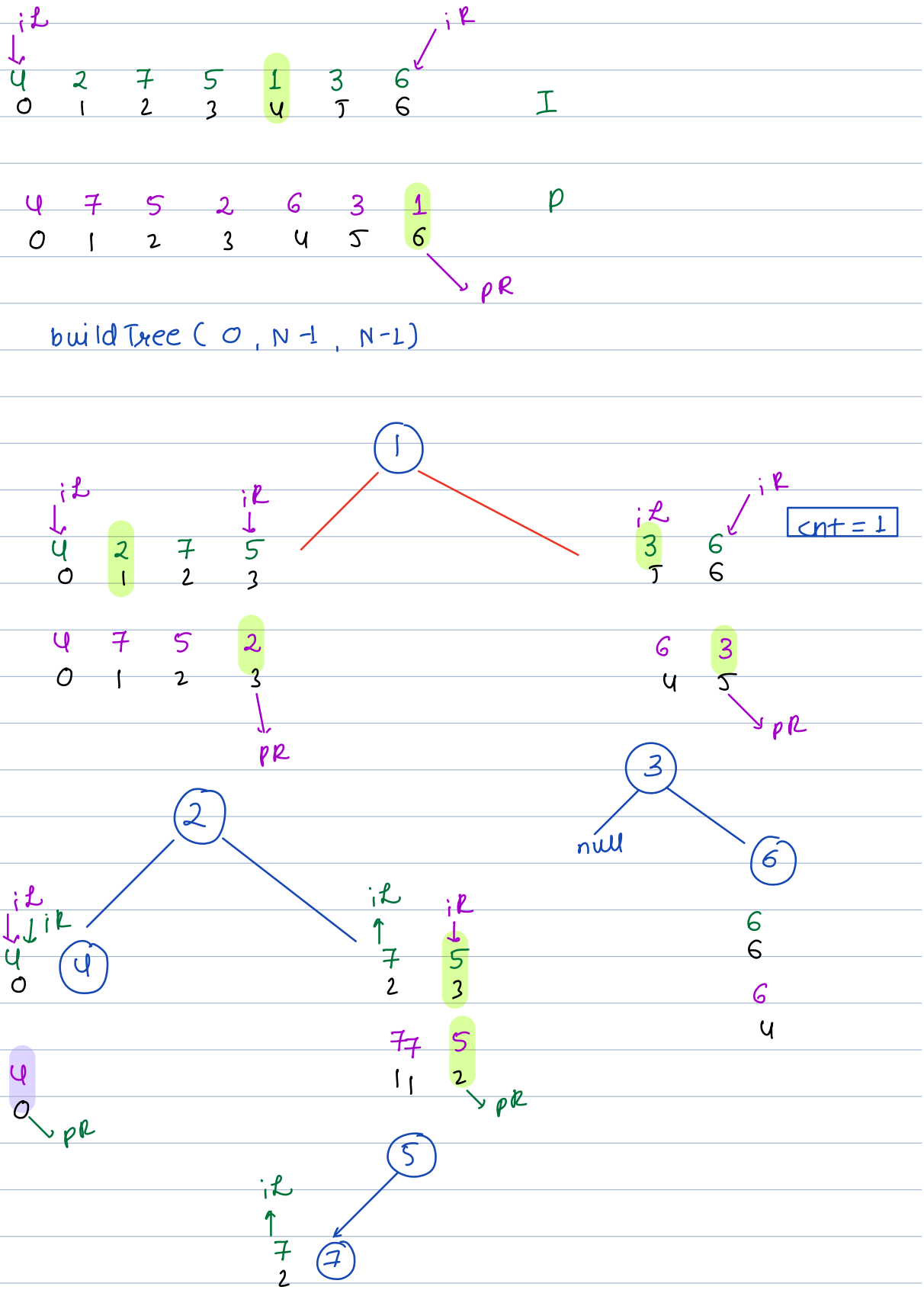
```
I[]    // inorder
P[]    // postorder
TreeNode buildTree ( iL , iR , pR ) {
    if ( iL > iR ) return null

    val = P[pR]
    TreeNode root = new TreeNode ( val )

    idx = search ( I , val ) // O(1) with HM
    cnt = iR - idx                    // idx+1 ..... iR

    root.left  = buildTree ( iL , idx-1 , pR-cnt-1 )
    root.right = buildTree ( idx+1 , iR , pR-1 )

    return root
}
```



TC: O(N)
SC: O(N)




```
main ( I , P ) {
    this.I = I
    this.P = P
    // Build HM with key, val for inorder
    return buildTree ( 0 , N-1 , N-1 )
}
```


iL
iR
4 2 7 5 1 3 6
0 1 2 3 4 5 6
I
4 7 5 2 6 3 1
0 1 2 3 4 5 6
P
pR
buildTree (0 , N-1 , N-1)
1
4 2 7 5
0 1 2 3
3 6
5 5
cnt = 1
4 7 5 2
0 1 2 3
6 3
4 5
pR
pR
3
null
6
2
4
0
pR
7 5
2 3
77 5
11 2
pR
6
6
6
4
5
7
7
2

7