AP ASSIGNMENT

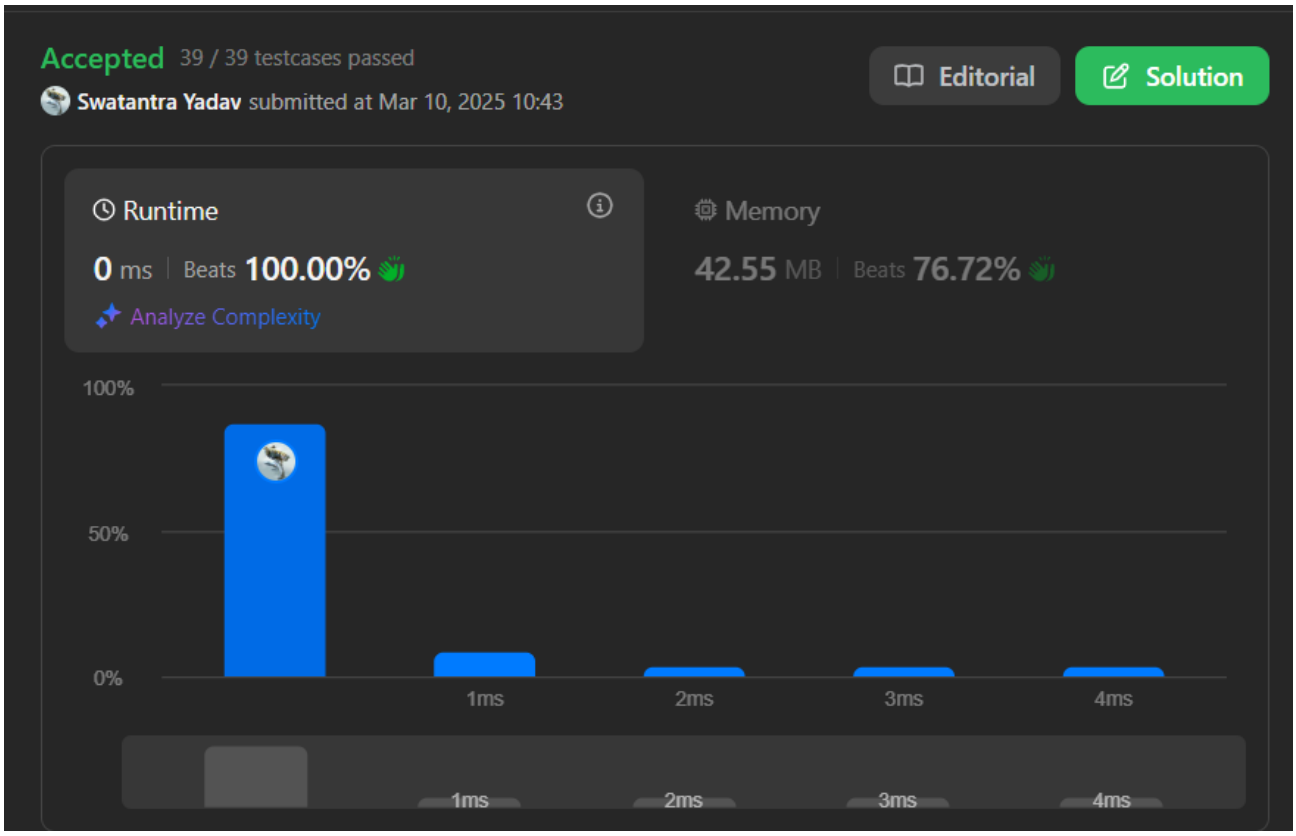
Name: Swatantra

UID: 22BCS12725

Section: 612-“B”

Maximum Depth of Binary Tree

```
class Solution {
    public int maxDepth(TreeNode root) {
        if (root == null) {
            return 0;
        }
        return 1 + Math.max(maxDepth(root.left), maxDepth(root.right));
    }
}
```



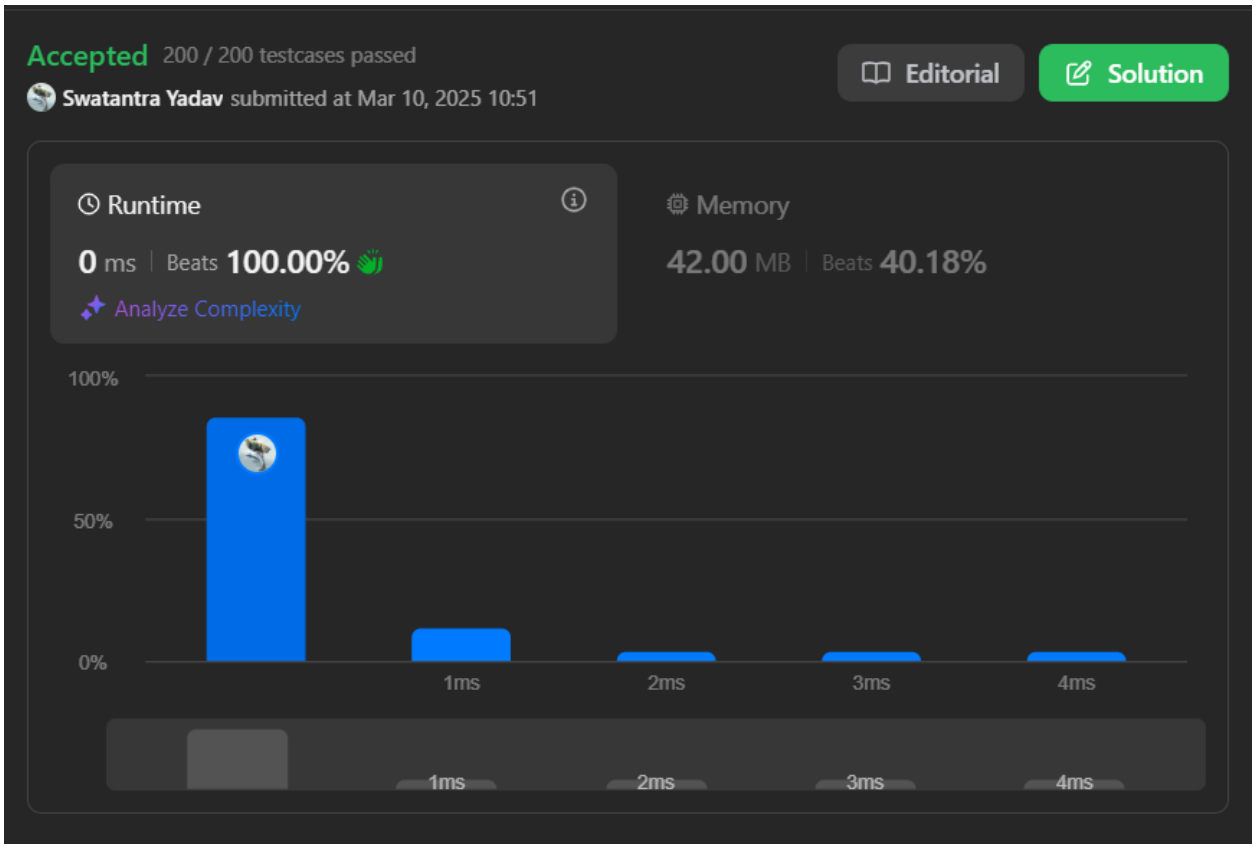
Validate Binary Search Tree

```
class Solution {
    long prev = Long.MIN_VALUE;
    boolean isValid = true;

    public boolean isValidBST(TreeNode root) {
        inorder(root);
        return isValid;
    }

    void inorder(TreeNode root) {
        if (root.left != null) inorder(root.left);

        int val = root.val;
        if (prev >= val) {
            isValid = false;
            return;
        }
        prev = val;

        if (root.right != null) inorder(root.right);
    }
}
```

Symmetric Tree

```
class Solution {
    public boolean isSymmetric(TreeNode root) {
        if (root == null) {
            return true;
        }
        return isMirror(root.left, root.right);
    }

    private boolean isMirror(TreeNode node1, TreeNode node2) {
        if (node1 == null && node2 == null) {
            return true;
        }
        if (node1 == null || node2 == null) {
            return false;
```

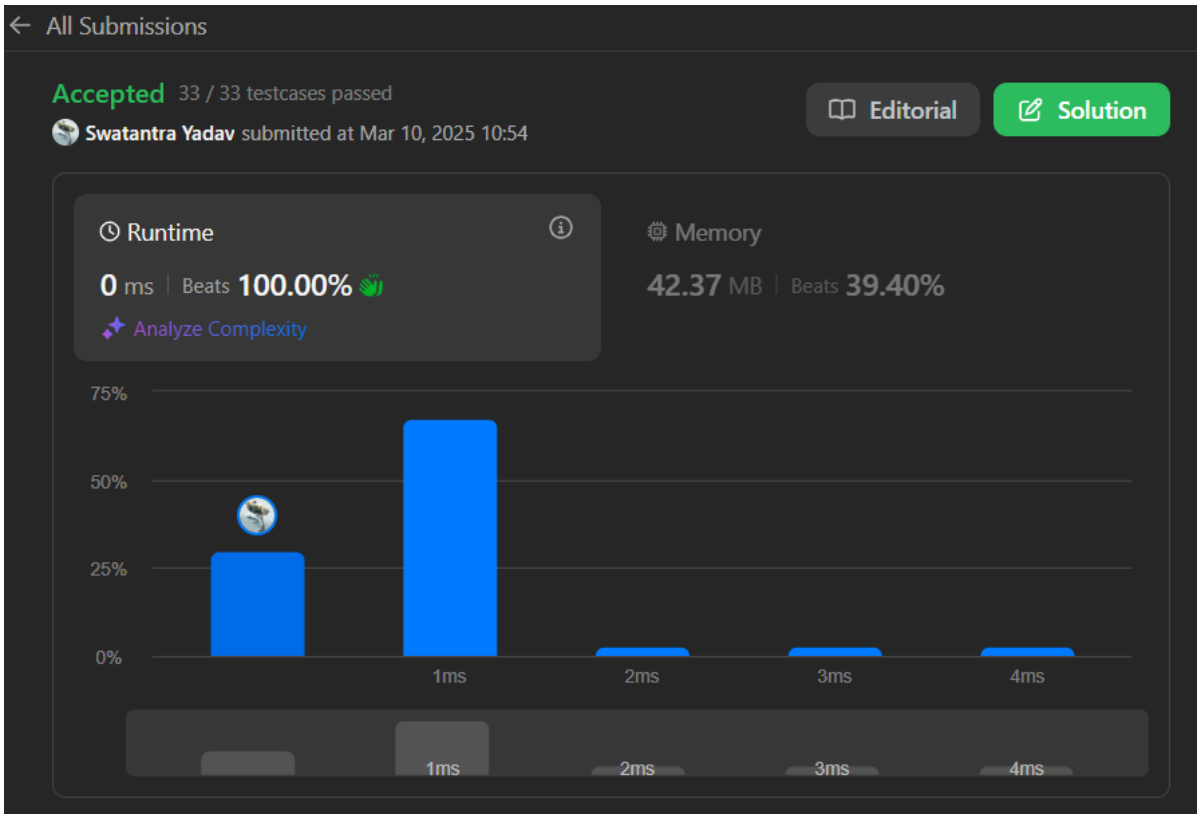
```
        }
        return node1.val == node2.val && isMirror(node1.left, node2.right) &&
isMirror(node1.right, node2.left);
    }
}
```



Binary Tree Zigzag Level Order Traversal

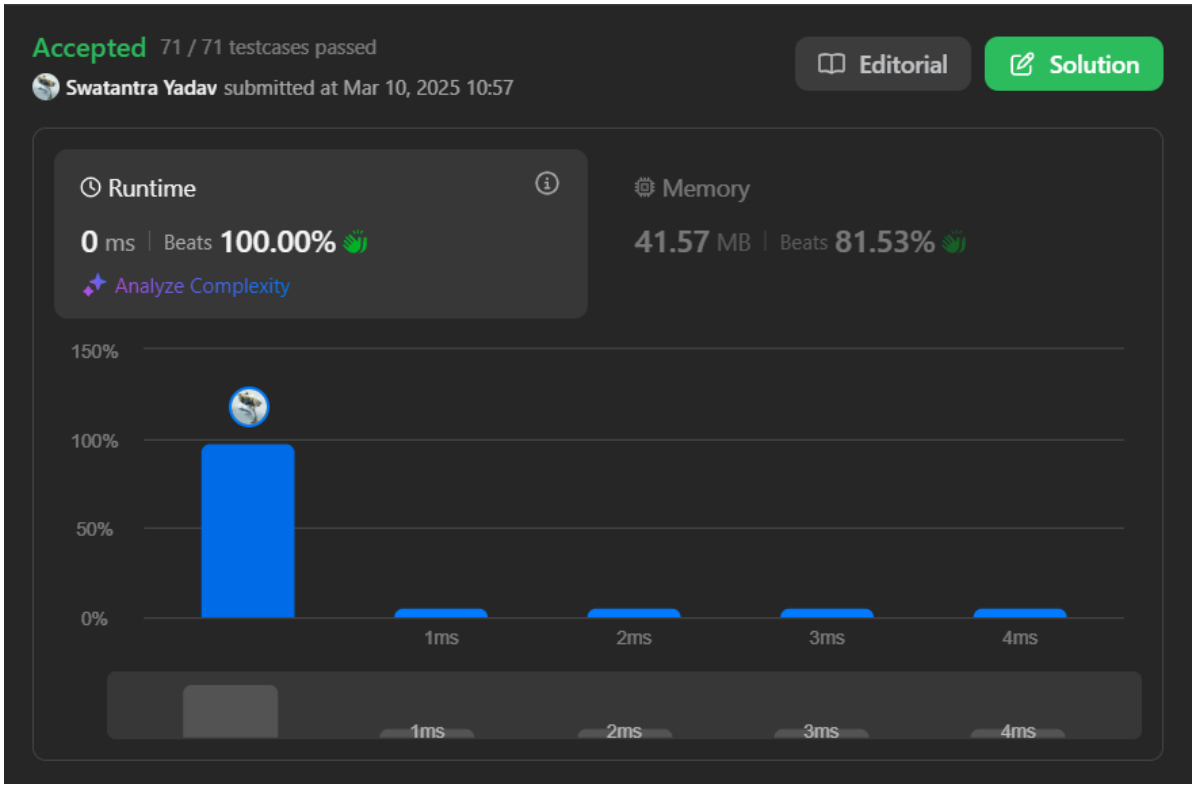
```
public class Solution {
    public List<List<Integer>> zigzagLevelOrder(TreeNode root)
    {
        List<List<Integer>> sol = new ArrayList<>();
        travel(root, sol, 0);
        return sol;
    }
```

```
private void travel(TreeNode curr, List<List<Integer>> sol, int level)
{
    if(curr == null) return;

    if(sol.size() <= level)
    {
        List<Integer> newLevel = new LinkedList<>();
        sol.add(newLevel);
    }

    List<Integer> collection  = sol.get(level);
    if(level % 2 == 0) collection.add(curr.val);
    else collection.add(0, curr.val);

    travel(curr.left, sol, level + 1);
    travel(curr.right, sol, level + 1);
}
```



Binary Tree Inorder Traversal

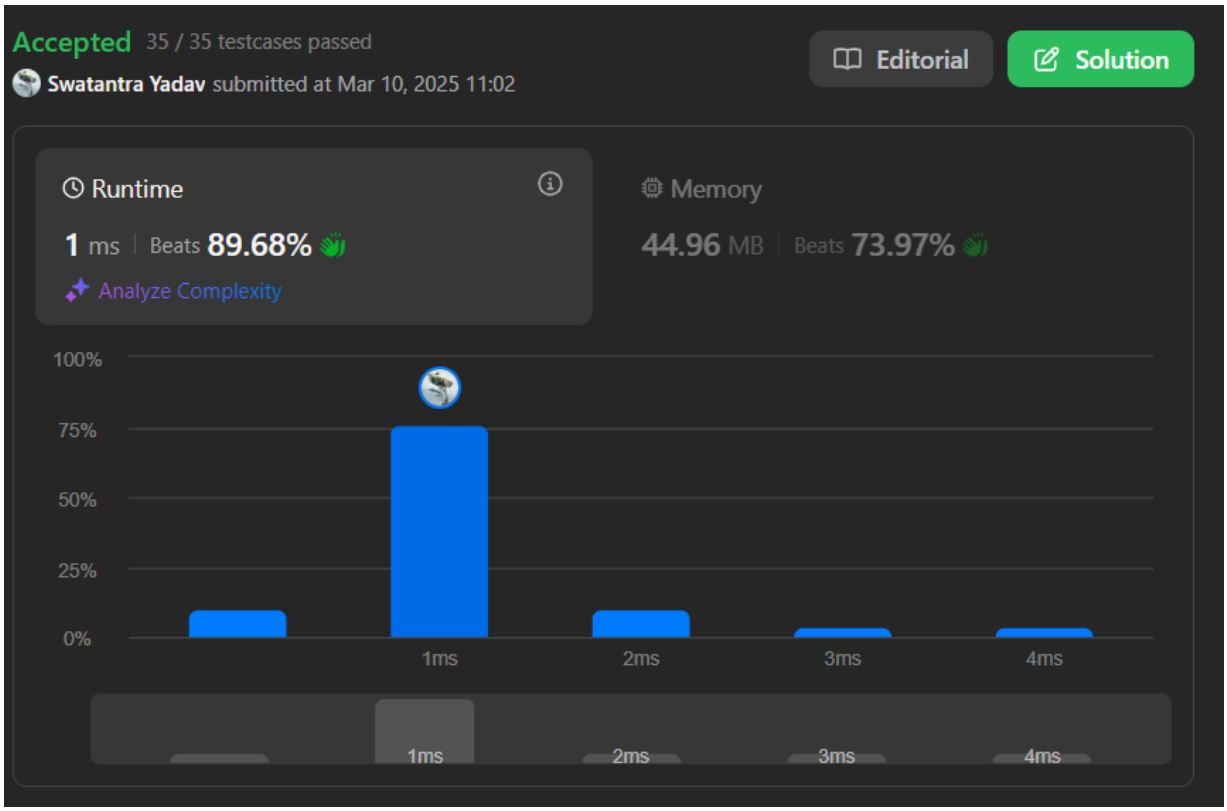
```
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> res = new ArrayList<>();

        inorder(root, res);
        return res;
    }

    private void inorder(TreeNode node, List<Integer> res) {
        if (node == null) {
            return;
        }
        inorder(node.left, res);
        res.add(node.val);
        inorder(node.right, res);
    }
```



Binary Tree Level Order Traversal

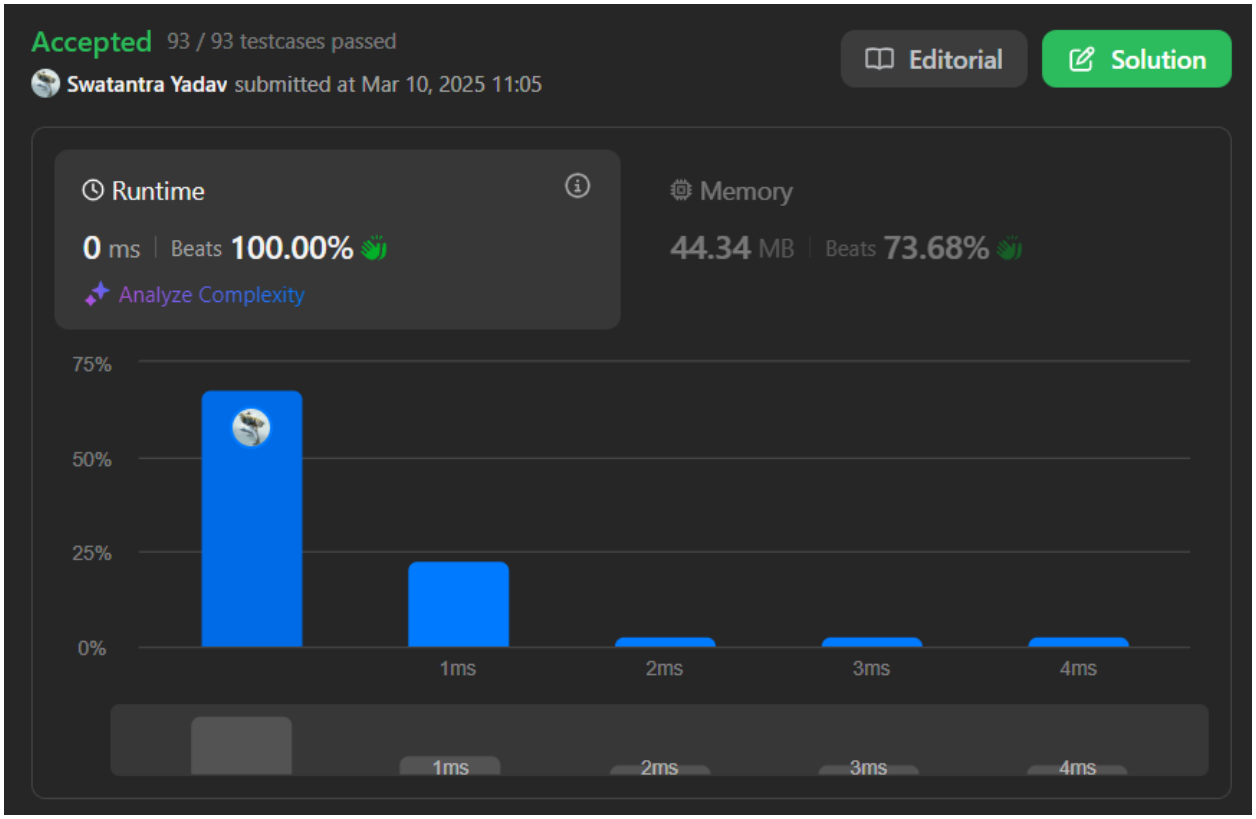
```
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {

        List<List<Integer>> result=new ArrayList<>();
        if(root==null)
        return result;

        Queue<TreeNode> q=new LinkedList<>();
        q.offer(root);
        while(!q.isEmpty()){
            int levelSize=q.size();
            List<Integer> currentLevel = new ArrayList<>();

            for(int i=0;i<levelSize;i++){
```

```
            TreeNode currentNode = q.poll();
            currentLevel.add(currentNode.val);
            if(currentNode.left !=null){
                q.offer(currentNode.left);
            }

            if(currentNode.right !=null){
                q.offer(currentNode.right);
            }
        }

        result.add(currentLevel);
    }
    return result;
  }
}
```



Kth Smallest Element in a BST

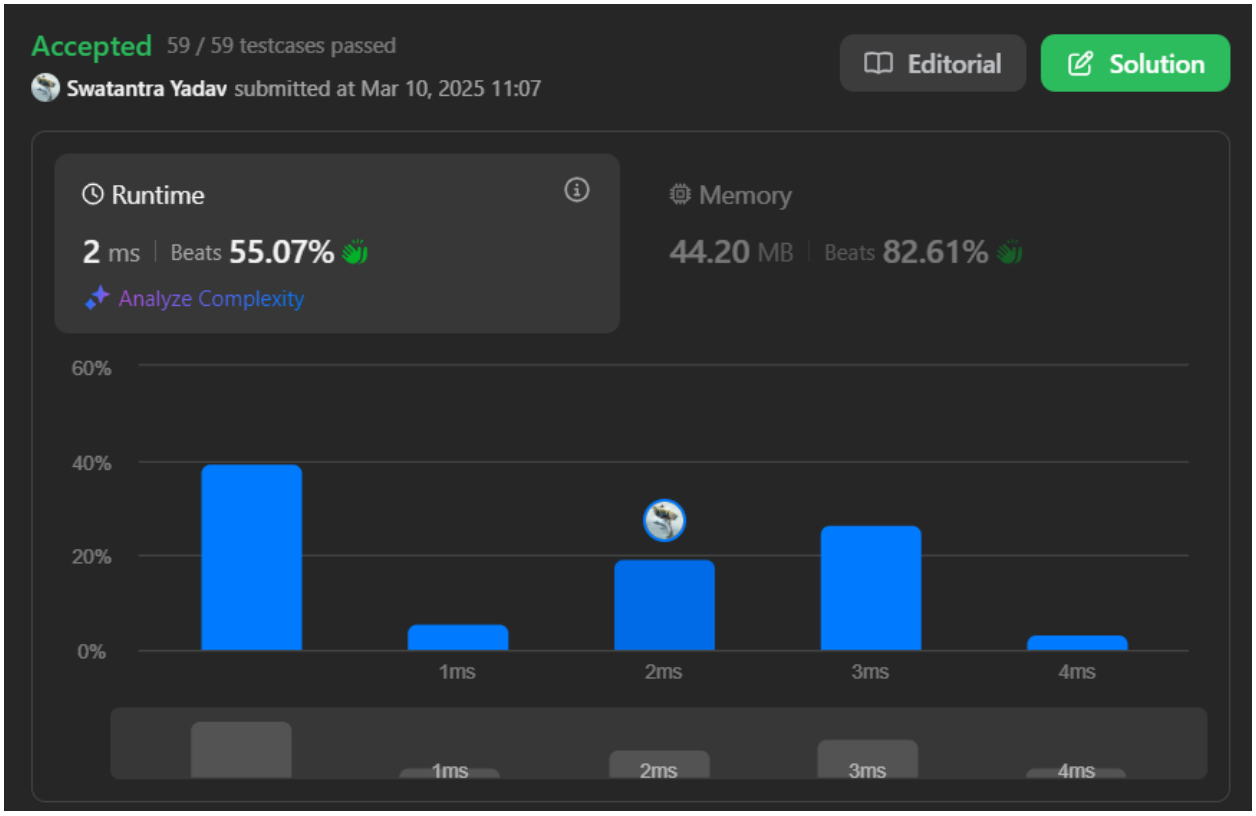
```
class Solution {
    private int count = 0;

    public int kthSmallest(TreeNode root, int k) {
        TreeNode result = helper(root, k);
        return result != null ? result.val : 0;
    }

    private TreeNode helper(TreeNode root, int k) {
        if (root == null) return null;



        TreeNode left = helper(root.left, k);
        if (left != null) return left;

        count++;
        if (count == k) return root;

        return helper(root.right, k);
    }
}
```



Populating Next Right Pointers in Each Node

```
class Solution {
    public Node connect(Node root) {
        if(root == null) return null;
        Queue<Node> q = new LinkedList<>();
        q.offer(root);
        while(!q.isEmpty()) {
            Node rightNode = null;
            for(int i = q.size(); i > 0; i--) {
                Node cur = q.poll();
                cur.next = rightNode;
                rightNode = cur;
                if(cur.right != null) {
                    q.offer(cur.right);
                    q.offer(cur.left);
```

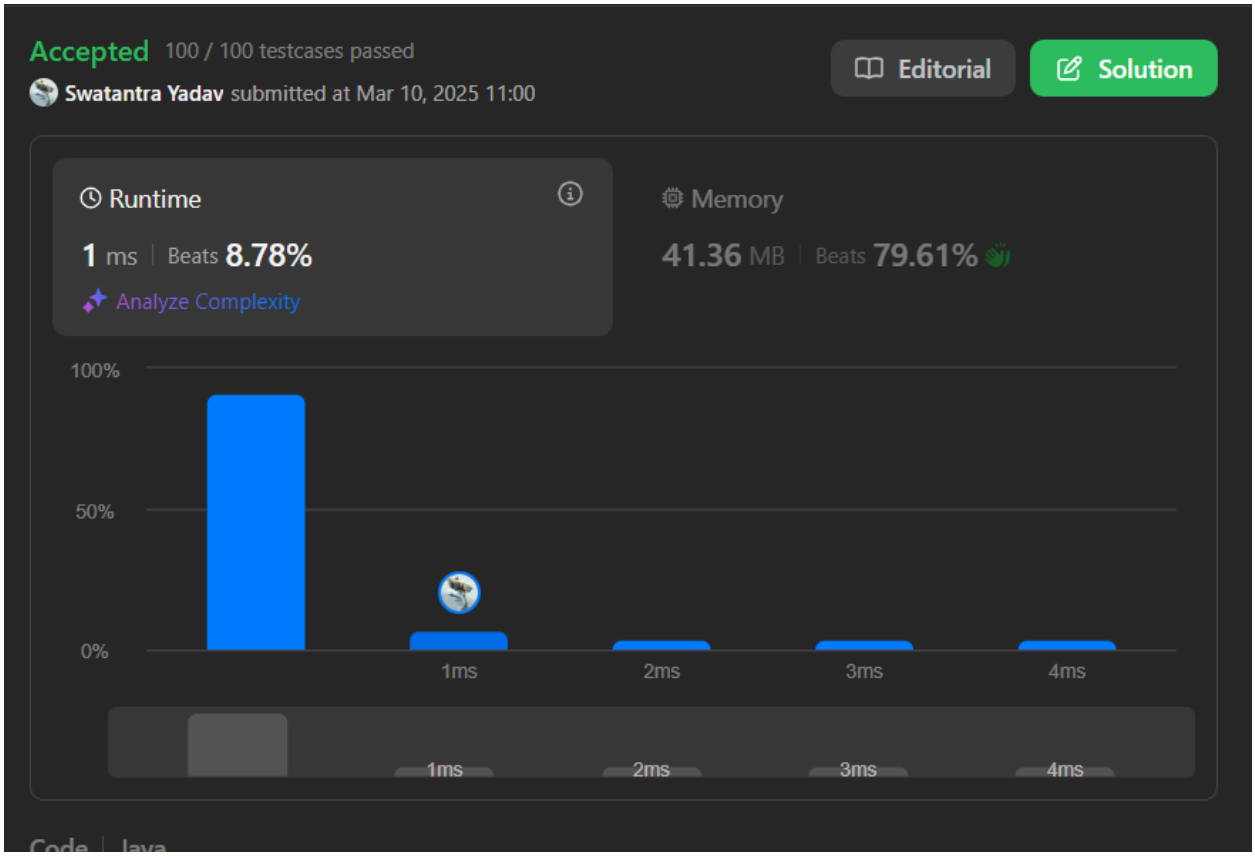
```
            }
        }
    }
    return root;
  }
}
```



Sum of Left Leaves

```
class Solution {
    public int sumOfLeftLeaves(TreeNode root) {
        if (root == null) {
            return 0;
        }

        Queue<Pair<TreeNode, Boolean>> queue = new LinkedList<>();
        queue.offer(new Pair<>(root, false));  // (node, is_left)
        int totalSum = 0;
```

```
    while (!queue.isEmpty()) {
        Pair<TreeNode, Boolean> pair = queue.poll();
        TreeNode node = pair.getKey();
        boolean isLeft = pair.getValue();

        if (isLeft && node.left == null && node.right == null) {
            totalSum += node.val;
        }

        if (node.left != null) {
            queue.offer(new Pair<>(node.left, true));
        }
        if (node.right != null) {
            queue.offer(new Pair<>(node.right, false));
        }
    }

    return totalSum;
}
```


Accepted 100 / 100 testcases passed
Swatantra Yadav submitted at Mar 10, 2025 11:00
Editorial
Solution
Runtime
1 ms | Beats 8.78%
Analyze Complexity
Memory
41.36 MB | Beats 79.61%
100%
50%
0%
1ms
2ms
3ms
4ms
1ms
2ms
3ms
4ms