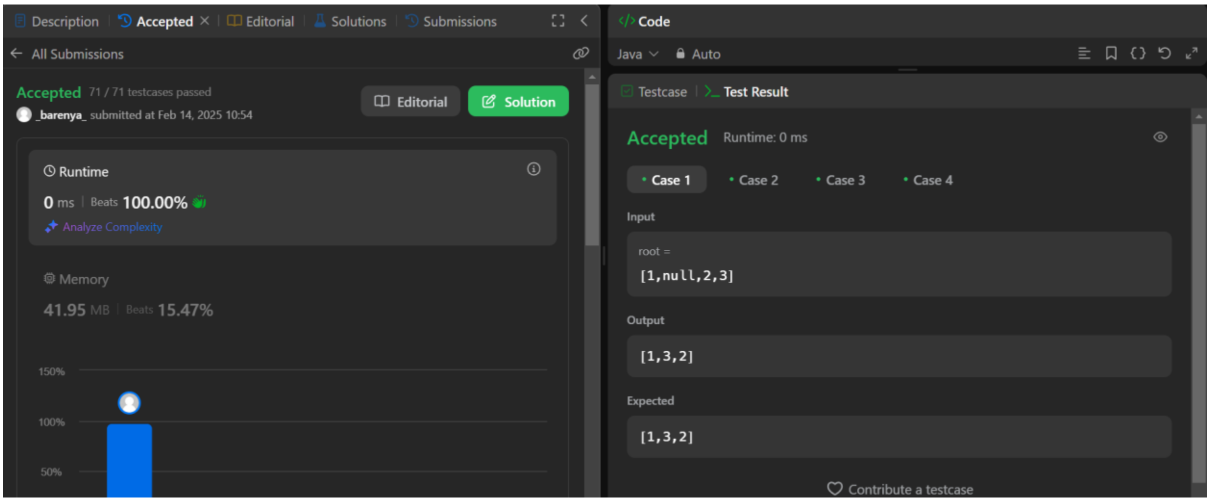
Experiment-3

Student Name: Barenya Behera **UID: 22BCS15121**
Branch: BE-CSE **Section/Group: FL_IOT-602-B**
Semester: 6 **Date of Performance: 14/02/25**
Subject Name: Advanced Programming Lab- 2 **Subject Code: 22CSP-351**

1. Binary Tree Inorder Traversal

```
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        inorderHelper(root, result);
        return result;
    }
    private void inorderHelper(TreeNode node, List<Integer> result) {
        if (node == null) return;
        inorderHelper(node.left, result);
        result.add(node.val);
        inorderHelper(node.right, result);
    }
}
```



Accepted 71 / 71 testcases passed
barenya submitted at Feb 14, 2025 10:54

Runtime
0 ms | Beats 100.00%

Memory
41.95 MB | Beats 15.47%

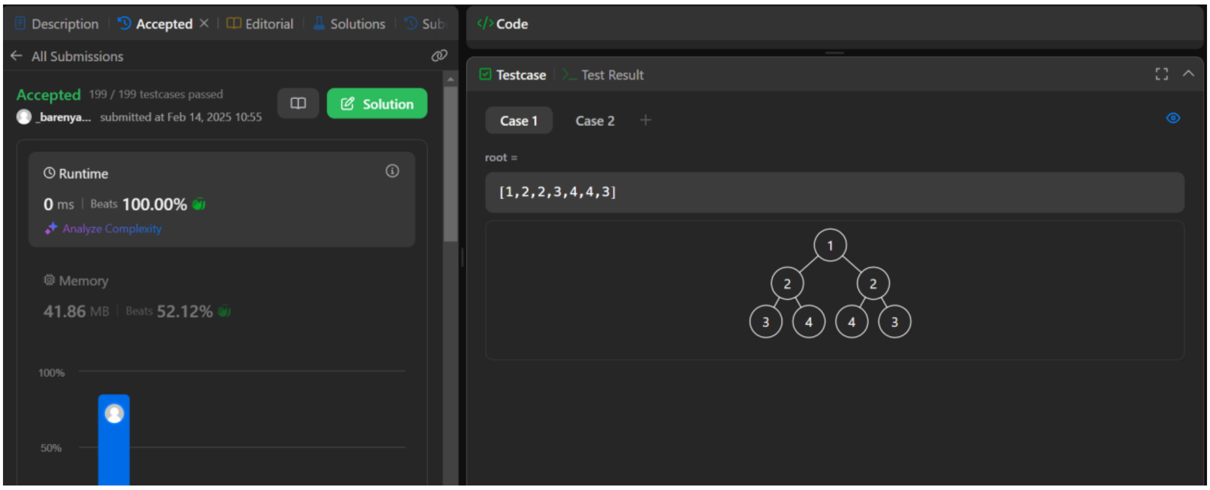
Accepted Runtime: 0 ms

Input
root =
```
[1,null,2,3]
```
Output
```
[1,3,2]
```
Expected
```
[1,3,2]
```

2. Symmetric Tree

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

```
  public boolean isMirror(TreeNode t1,TreeNode t2){

  if(t1==null && t2==null){
      return true;

 }
    if(t1==null || t2==null){
      return false;
    }
    return (t1.val == t2.val)
      && isMirror(t1.left, t2.right)
      && isMirror(t1.right, t2.left);
  }
}
```



Description | Accepted × | Editorial | Solutions | Sub

Code

All Submissions

Accepted 199 / 199 testcases passed

_barenya... submitted at Feb 14, 2025 10:55

Solution

Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

41.86 MB | Beats 52.12%

Testcase | Test Result

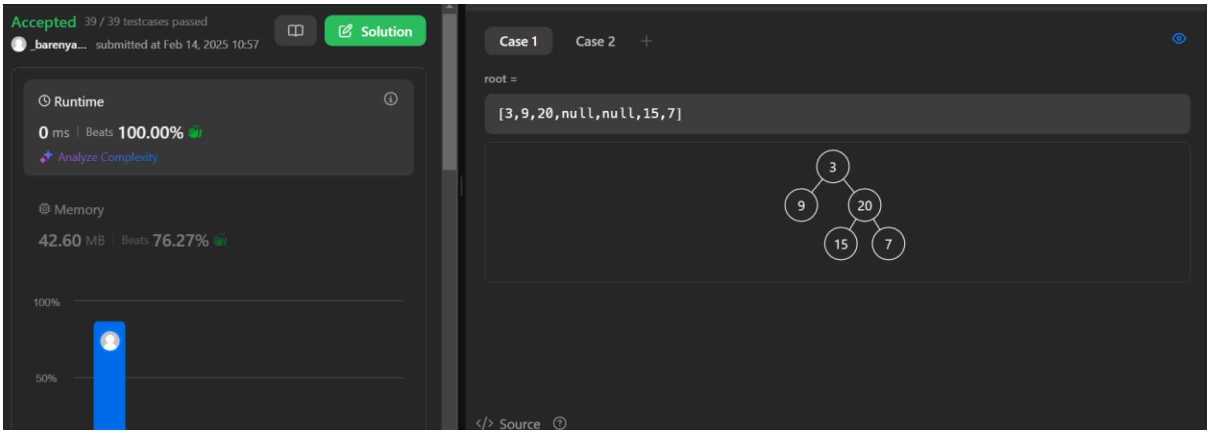
Case 1 Case 2 +

root =

[1,2,2,3,4,4,3]

3. 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));
  }
}
```



Accepted 39 / 39 testcases passed

_barenya... submitted at Feb 14, 2025 10:57

Solution

Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

42.60 MB | Beats 76.27%

Case 1 Case 2 +

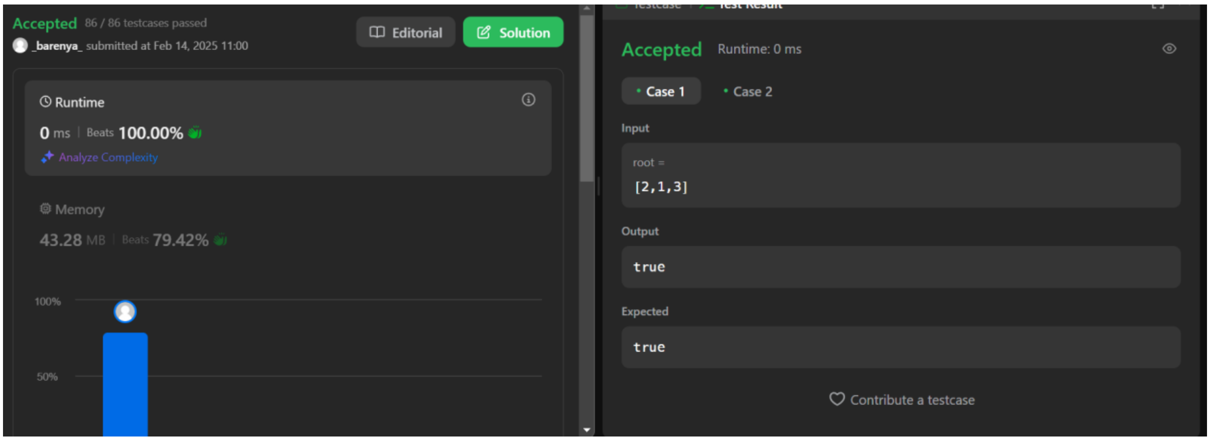
root =

[3,9,20,null,null,15,7]

Source

4. Validate Binary Search Tree

```
 class Solution {
    public boolean isValidBST(TreeNode root) {

 return validate(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }

    private boolean validate(TreeNode node, long min, long max) {
        if (node == null) return true;
        if (node.val <= min || node.val >= max) return false;
        return validate(node.left, min, node.val) && validate(node.right, node.val, max);
    }
}
```



Accepted 86 / 86 testcases passed
barenya submitted at Feb 14, 2025 11:00

Runtime
0 ms | Beats 100.00%

Memory
43.28 MB | Beats 79.42%

Accepted Runtime: 0 ms

Case 1 · Case 2

Input
root =
[2,1,3]

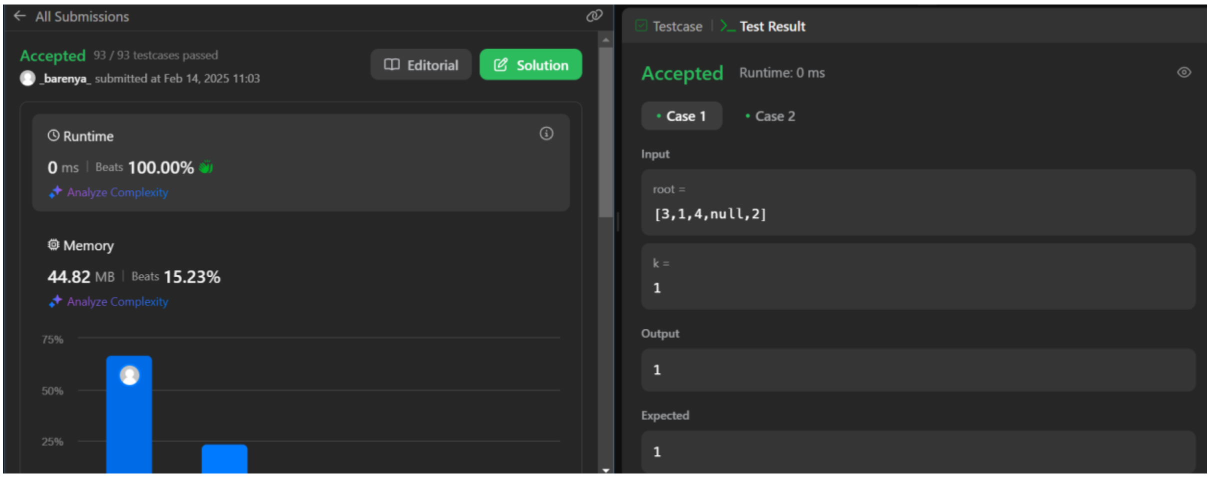
Output
true

Expected
true

5. Kth Smallest Element in a BST

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

    public int kthSmallest(TreeNode root, int k) {
        inorder(root, k);
        return result;
    }

    private void inorder(TreeNode node, int k) {
        if (node == null) return;

        inorder(node.left, k);
        count++;
        if (count == k) {
            result = node.val;
```

```
      return;
    }
    inorder(node.right, k);
  }
}
```



All Submissions

Accepted 93 / 93 testcases passed

barenya submitted at Feb 14, 2025 11:03

Runtime

0 ms | Beats 100.00%

Memory

44.82 MB | Beats 15.23%

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

root =

```
[3,1,4,null,2]
```

k =

```
1
```

Output

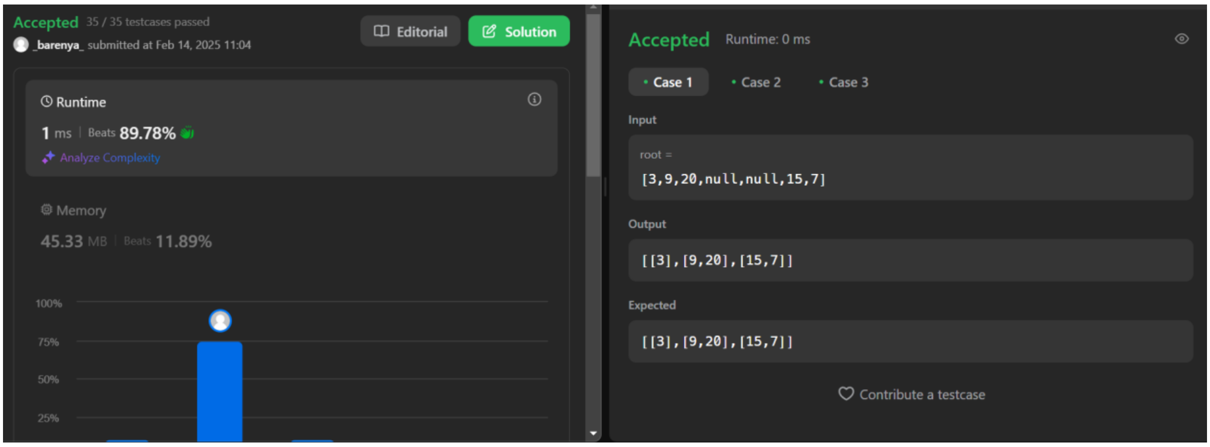
```
1
```

Expected

```
1
```

6. Binary Tree Level Order Traversal I

```
class Solution {
  public List<List<Integer>> levelOrder(TreeNode root) {
    List<List<Integer>> result = new ArrayList<>();
    if (root == null) return result;

    Queue<TreeNode> queue = new LinkedList<>();
    queue.add(root);

    while (!queue.isEmpty()) {
      int levelSize = queue.size();
      List<Integer> level = new ArrayList<>();

      for (int i = 0; i < levelSize; i++) {
        TreeNode current = queue.poll();
        level.add(current.val);

        if (current.left != null) queue.add(current.left);
        if (current.right != null) queue.add(current.right);
      }

      result.add(level);
    }

    return result;
    }

}
```



Accepted 35 / 35 testcases passed
barenya submitted at Feb 14, 2025 11:04

Runtime
1 ms | Beats 89.78%
Memory
45.33 MB | Beats 11.89%

Accepted Runtime: 0 ms

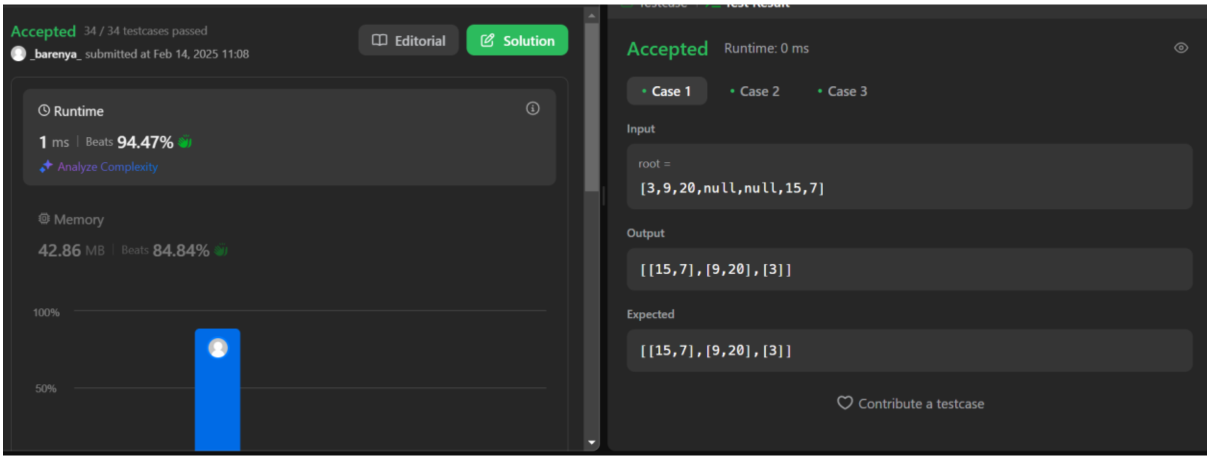
Input
root =
[3,9,20,null,null,15,7]

Output
[[3],[9,20],[15,7]]

Expected
[[3],[9,20],[15,7]]

7. Binary Tree Level Order Traversal II

```
class Solution {
   public List<List<Integer>> levelOrderBottom(TreeNode root) {
       LinkedList<List<Integer>> result = new LinkedList<>();
       if (root == null) return result;

       Queue<TreeNode> queue = new LinkedList<>();
       queue.add(root);

       while (!queue.isEmpty()) {
           int levelSize = queue.size();
           List<Integer> level = new ArrayList<>();

           for (int i = 0; i < levelSize; i++) {
               TreeNode current = queue.poll();
               level.add(current.val);

               if (current.left != null) queue.add(current.left);
               if (current.right != null) queue.add(current.right);
           }
           result.addFirst(level);
       }

       return result;
   }
}
```



Accepted 34 / 34 testcases passed
barenya submitted at Feb 14, 2025 11:08

Runtime
1 ms | Beats 94.47%

Memory
42.86 MB | Beats 84.84%

Accepted Runtime: 0 ms

Case 1 Case 2 Case 3

Input

```
root =
[3,9,20,null,null,15,7]
```

Output

```
[[15,7],[9,20],[3]]
```

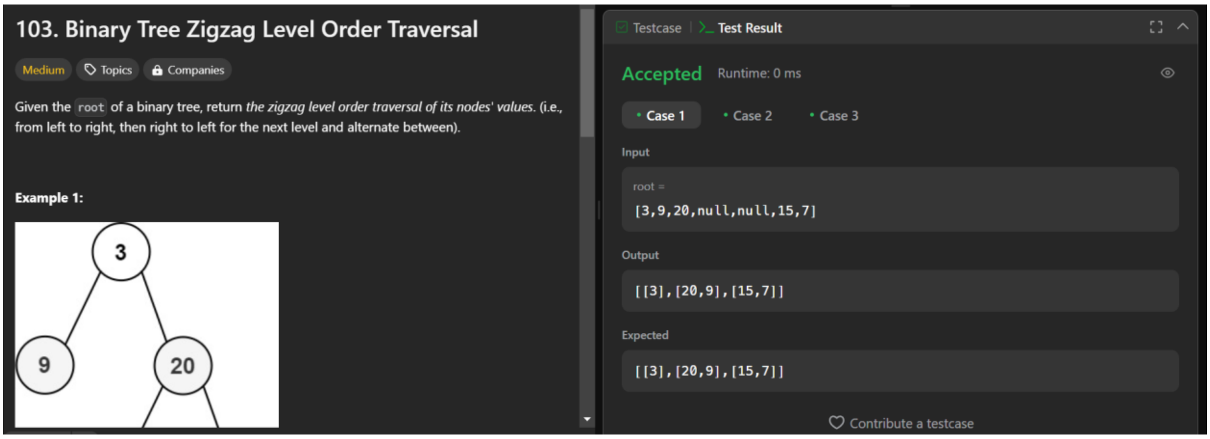
Expected

```
[[15,7],[9,20],[3]]
```

8. **Binary Tree ZigZag Level Order Traversal**

```
class Solution {
  public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
    List<List<Integer>> result = new ArrayList<>();
    if (root == null) return result;

    Queue<TreeNode> queue = new LinkedList<>();
    queue.add(root);
    boolean leftToRight = true;

    while (!queue.isEmpty()) {
      int levelSize = queue.size();
      LinkedList<Integer> level = new LinkedList<>();

      for (int i = 0; i < levelSize; i++) {
        TreeNode current = queue.poll();

        // Add elements based on the traversal direction
        if (leftToRight) {
          level.addLast(current.val);
        } else {
          level.addFirst(current.val);
        }

        if (current.left != null) queue.add(current.left);
        if (current.right != null) queue.add(current.right);
      }
```

```
             result.add(level);
            leftToRight = !leftToRight; // Toggle direction
        }

        return result;
    }
}
```



103. Binary Tree Zigzag Level Order Traversal

Medium | Topics | Companies

Given the `root` of a binary tree, return *the zigzag level order traversal of its nodes' values*. (i.e., from left to right, then right to left for the next level and alternate between).

Example 1:

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 | Case 2 | Case 3

Input

```
root =
[3,9,20,null,null,15,7]
```

Output

```
[[3],[20,9],[15,7]]
```

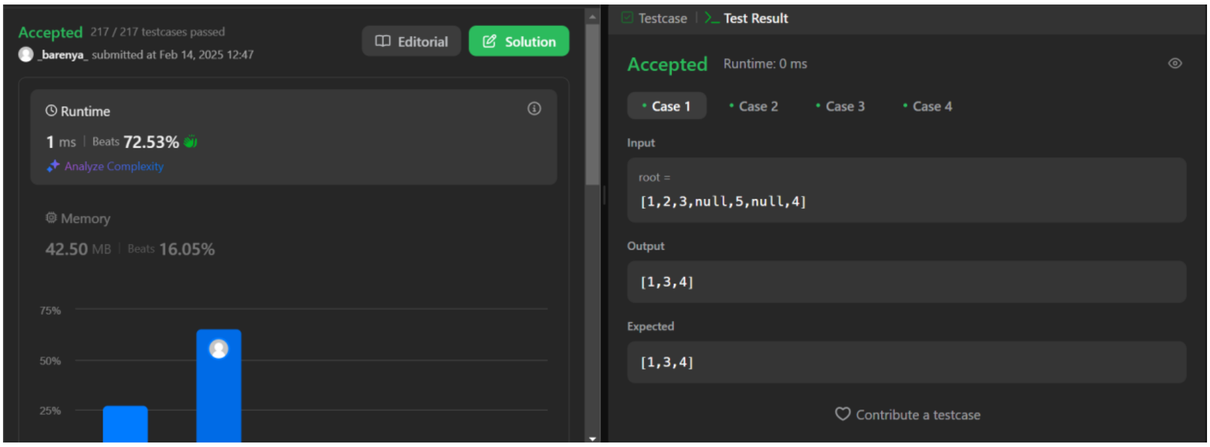
Expected

```
[[3],[20,9],[15,7]]
```

Contribute a testcase

9. Binary Tree Right Side View

```
class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> result = new LinkedList<>();
        if (root == null) {
            return result;
        }

        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);

        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            int rightmostValue = 0;

            for (int i = 0; i < levelSize; i++) {
                TreeNode current = queue.poll();
                rightmostValue = current.val;

                if (current.left != null) {
                    queue.offer(current.left);
                }
                if (current.right != null) {
                    queue.offer(current.right);
                }
            }

            result.add(rightmostValue);
        }

        return result;
    }
}
```



Accepted 217 / 217 testcases passed
barenya submitted at Feb 14, 2025 12:47

Runtime
1 ms | Beats 72.53%

Memory
42.50 MB | Beats 16.05%

Testcase | Test Result

Accepted Runtime: 0 ms

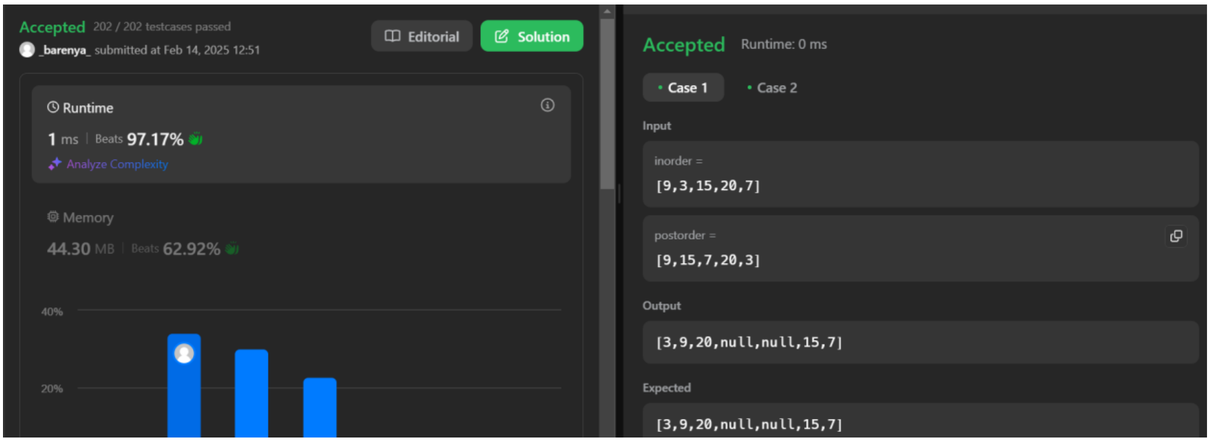
Input
root =
[1,2,3,null,5,null,4]

Output
[1,3,4]

Expected
[1,3,4]

10. Construct Binary Tree from Inorder and Postorder Traversal

```
class Solution {
    private HashMap<Integer, Integer> inorderMap;
    private int postIndex;

    public TreeNode buildTree(int[] inorder, int[] postorder) {
        inorderMap = new HashMap<>();
        postIndex = postorder.length - 1;

        for (int i = 0; i < inorder.length; i++) {
            inorderMap.put(inorder[i], i);
        }

        return constructTree(postorder, 0, inorder.length - 1);
    }

    private TreeNode constructTree(int[] postorder, int left, int right) {
        if (left > right) return null;

        int rootVal = postorder[postIndex--];
        TreeNode root = new TreeNode(rootVal);
        int inorderIndex = inorderMap.get(rootVal);

        root.right = constructTree(postorder, inorderIndex + 1, right);
        root.left = constructTree(postorder, left, inorderIndex - 1);

        return root;
    }
}
```



Accepted 202 / 202 testcases passed
barenya submitted at Feb 14, 2025 12:51

Runtime
1 ms | Beats 97.17%

Memory
44.30 MB | Beats 62.92%

Accepted Runtime: 0 ms

Input

inorder =
```
[9,3,15,20,7]
```

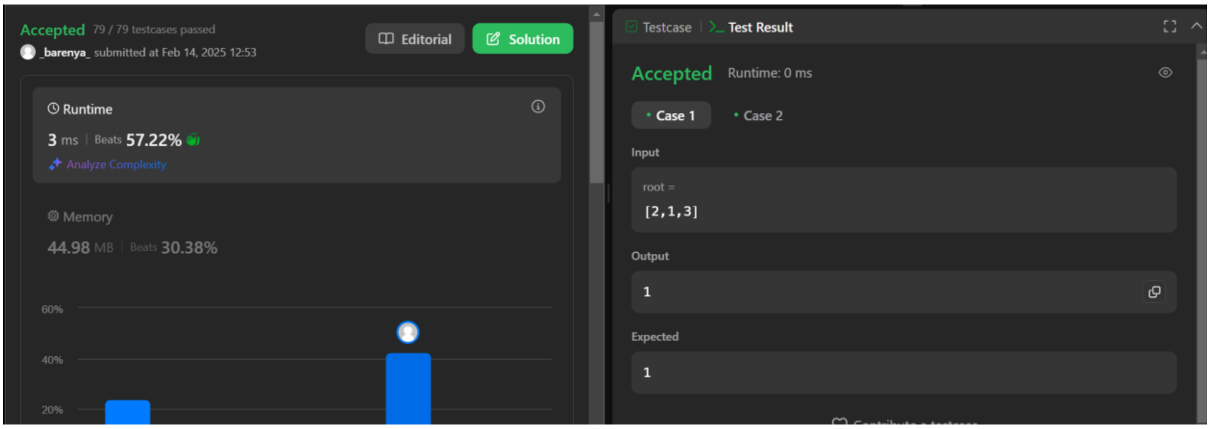
postorder =
```
[9,15,7,20,3]
```

Output
```
[3,9,20,null,null,15,7]
```

Expected
```
[3,9,20,null,null,15,7]
```

11. Find Bottom Left Tree Value

```
class Solution {
    public int findBottomLeftValue(TreeNode root) {
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        int leftMostValue = root.val;

        while (!queue.isEmpty()) {
            int size = queue.size();
            leftMostValue = queue.peek().val;

            for (int i = 0; i < size; i++) {
                TreeNode node = queue.poll();

                if (node.left != null) queue.offer(node.left);
                if (node.right != null) queue.offer(node.right);
            }
        }
        return leftMostValue;
    }
}
```



Accepted 79 / 79 testcases passed
barenya submitted at Feb 14, 2025 12:53

Runtime
3 ms | Beats 57.22%

Memory
44.98 MB | Beats 30.38%

Accepted Runtime: 0 ms

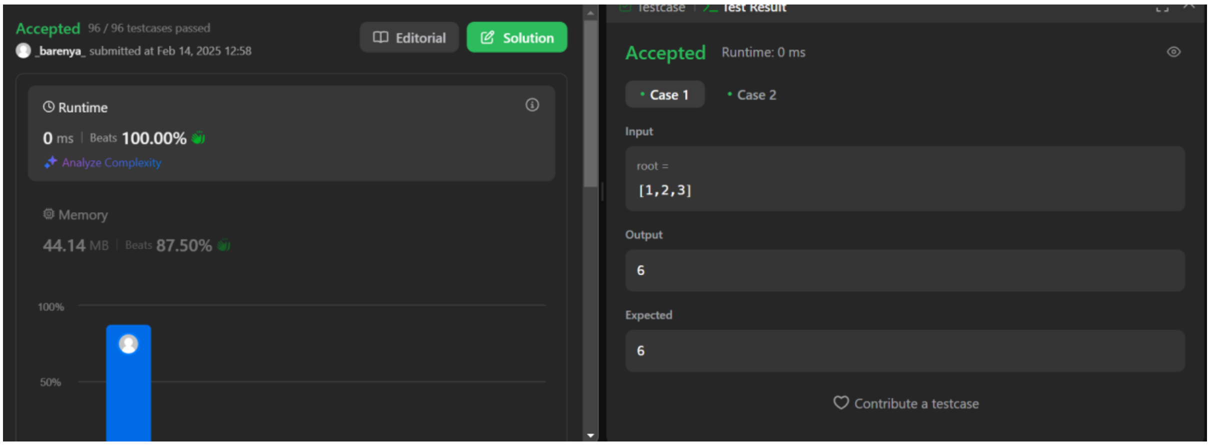
Input
root =
[2,1,3]

Output
1

Expected
1

12. Binary Tree Maximum Path Sum

```
class Solution {
  private int maxSum = Integer.MIN_VALUE;

  public int maxPathSum(TreeNode root) {
    findMaxPath(root);
    return maxSum;
  }

  private int findMaxPath(TreeNode node) {
    if (node == null) return 0;

    int left = Math.max(0, findMaxPath(node.left));
    int right = Math.max(0, findMaxPath(node.right));

    maxSum = Math.max(maxSum, left + right + node.val);

    return Math.max(left, right) + node.val;
  }
}
```



Accepted 96 / 96 testcases passed
barenya submitted at Feb 14, 2025 12:58

Runtime
0 ms | Beats 100.00%

Memory
44.14 MB | Beats 87.50%

Accepted Runtime: 0 ms

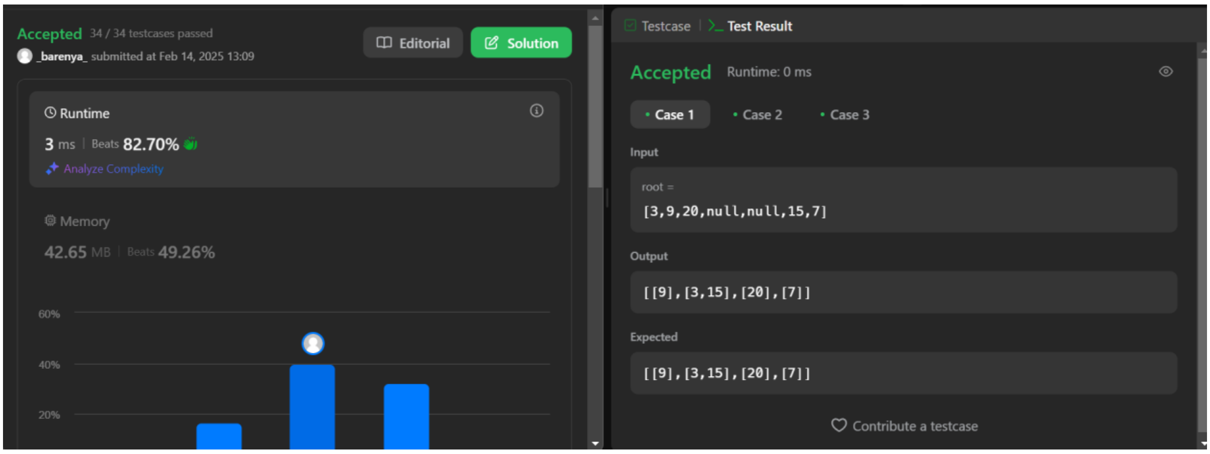
Input
root =
[1,2,3]

Output
6

Expected
6

13. Vertical Order Traversal of Binary Tree

```
 class Solution {
  public List<List<Integer>> verticalTraversal(TreeNode root) {
    TreeMap<Integer, PriorityQueue<NodeInfo>> map = new TreeMap<>();
    Queue<NodeInfo> queue = new LinkedList<>();
    queue.offer(new NodeInfo(root, 0, 0));

    while (!queue.isEmpty()) {
      NodeInfo curr = queue.poll();
      map.putIfAbsent(curr.col, new PriorityQueue<>());
      map.get(curr.col).offer(curr);

      if (curr.node.left != null) queue.offer(new NodeInfo(curr.node.left, curr.row + 1,
curr.col - 1));
      if (curr.node.right != null) queue.offer(new NodeInfo(curr.node.right, curr.row + 1,
curr.col + 1));
    }

    List<List<Integer>> result = new ArrayList<>();
    for (PriorityQueue<NodeInfo> pq : map.values()) {
      List<Integer> columnList = new ArrayList<>();
      while (!pq.isEmpty()) columnList.add(pq.poll().node.val);
      result.add(columnList);
    }
    return result;
  }

  private static class NodeInfo implements Comparable<NodeInfo> {
    TreeNode node;
    int row, col;
    NodeInfo(TreeNode node, int row, int col) {
      this.node = node;
      this.row = row;
      this.col = col;
    }

    @Override
    public int compareTo(NodeInfo other) {
      if (this.row == other.row) return Integer.compare(this.node.val, other.node.val);
      return Integer.compare(this.row, other.row);
    }
  }
}
```



Accepted 34 / 34 testcases passed

barenya submitted at Feb 14, 2025 13:09

Runtime

3 ms | Beats 82.70%

Memory

42.65 MB | Beats 49.26%

Accepted Runtime: 0 ms

Input

root =

```
[3,9,20,null,null,15,7]
```

Output

```
[[9],[3,15],[20],[7]]
```

Expected

```
[[9],[3,15],[20],[7]]
```