Worksheet 3

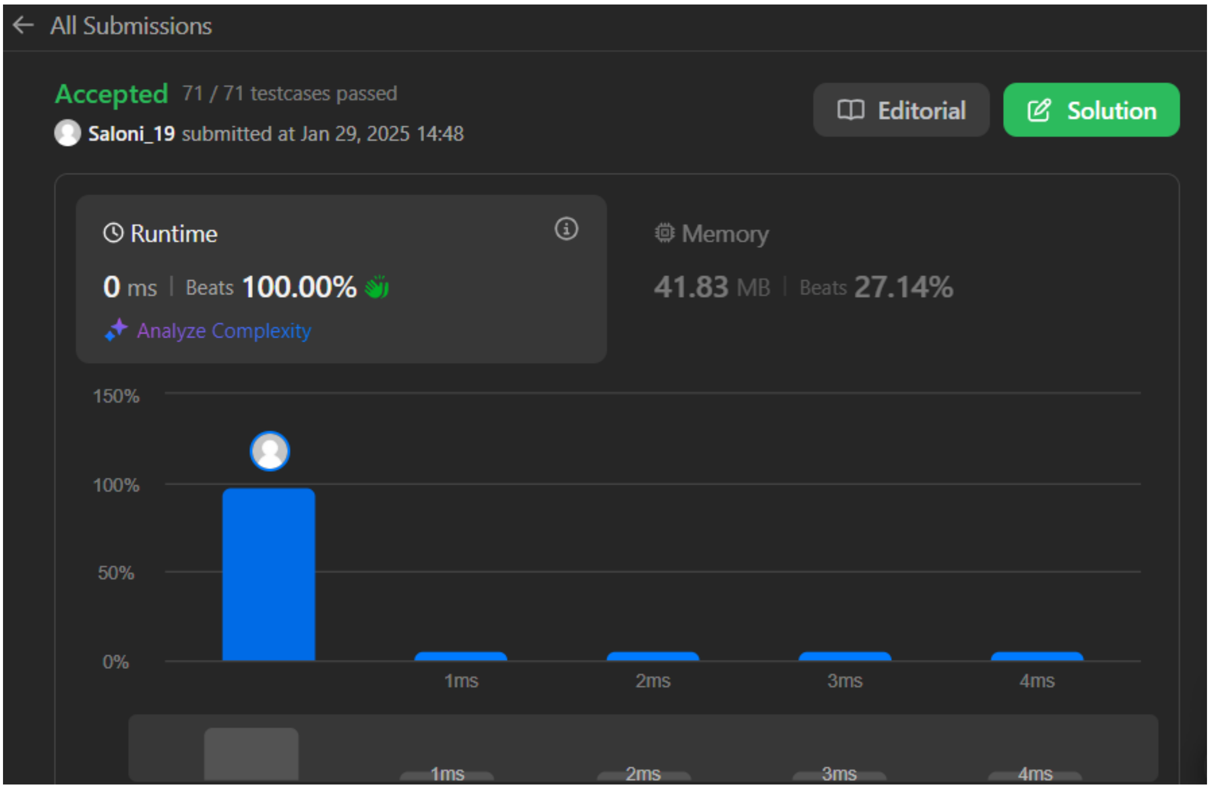
Name: Saloni Gupta

UID: 22BCS16659

94. Binary Tree Inorder Traversal

Code:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<Integer>();

        inorder (root, result);
        return result;
    }
    public void inorder(TreeNode a, List<Integer> res){
        if (a == null) return;

        inorder(a.left, res);
        res.add(a.val);
        inorder(a.right, res);
    }
}
```



Accepted 71 / 71 testcases passed

Saloni_19 submitted at Jan 29, 2025 14:48

Runtime

0 ms | Beats 100.00%

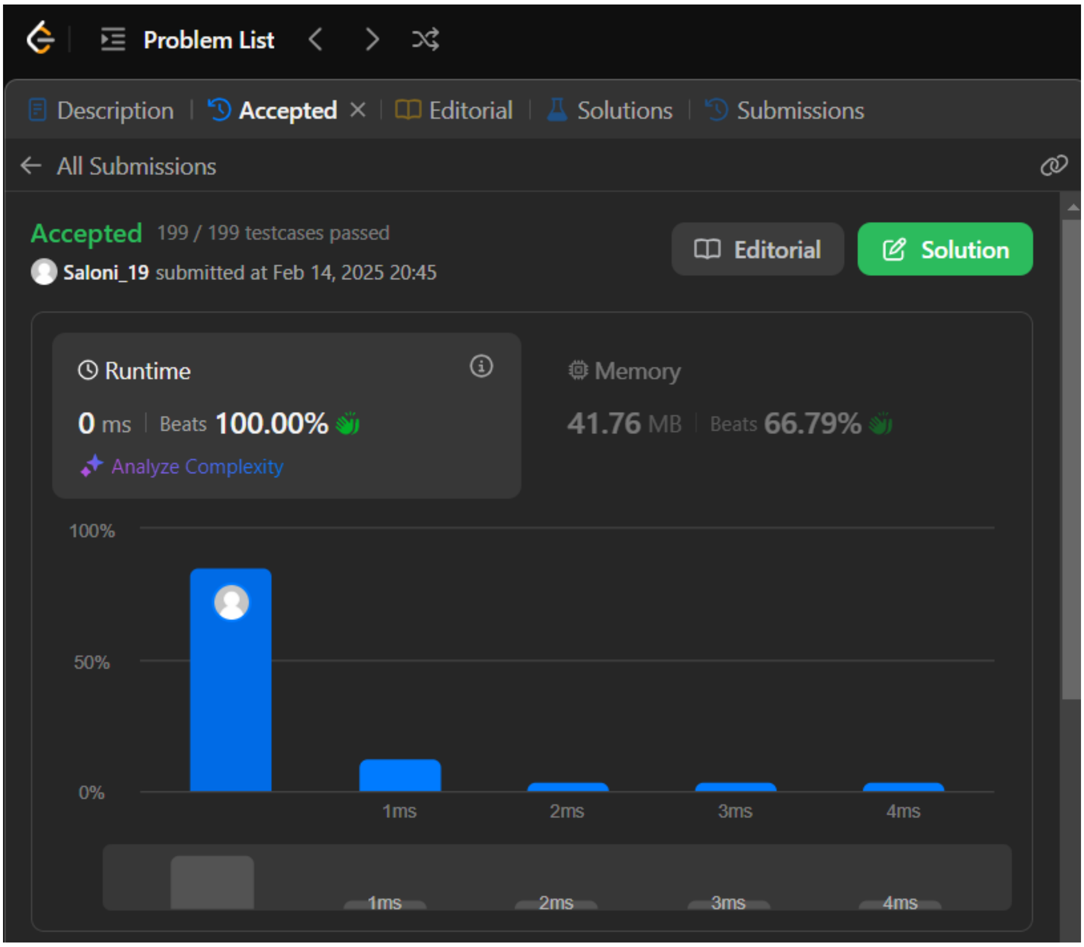
Memory

41.83 MB | Beats 27.14%

101. Symmetric Tree

Code:

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

    private boolean isMirror(TreeNode n1, TreeNode n2) {
        if (n1 == null && n2 == null) {
            return true;
        }

        if (n1 == null || n2 == null) {
            return false;
        }

        return n1.val == n2.val && isMirror(n1.left, n2.right) && isMirror(n1.right, n2.left);
    }
}
```



Accepted 199 / 199 testcases passed

Saloni_19 submitted at Feb 14, 2025 20:45

Runtime

0 ms | Beats 100.00%

Memory

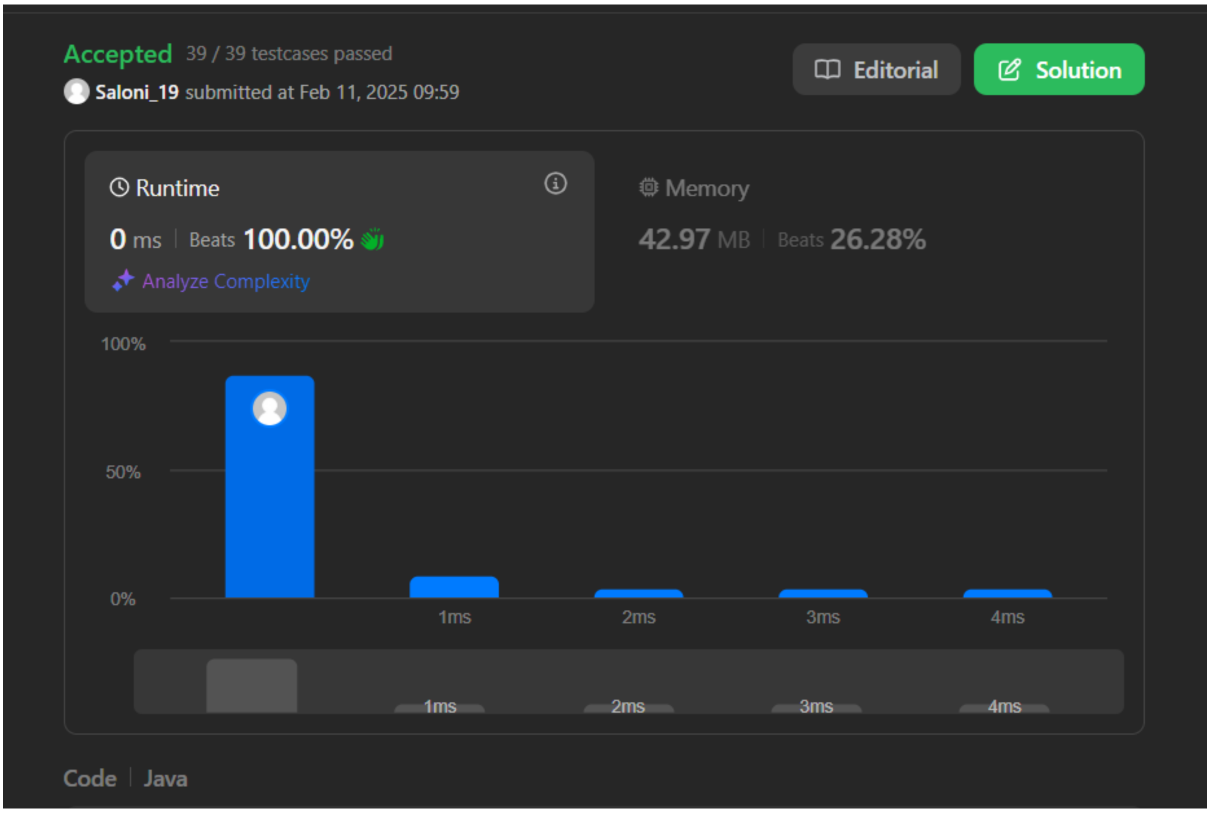
41.76 MB | Beats 66.79%

104. Maximum Depth of Binary Tree

Code :

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
```

```
class Solution {
    public int maxDepth(TreeNode root) {

        if(root == null) return 0;
        int l = maxDepth(root.left);
        int r = maxDepth(root.right);
        return Math.max(l,r) + 1;

    }
}
```



Accepted 39 / 39 testcases passed

Saloni_19 submitted at Feb 11, 2025 09:59

Runtime

0 ms | Beats 100.00%

Memory

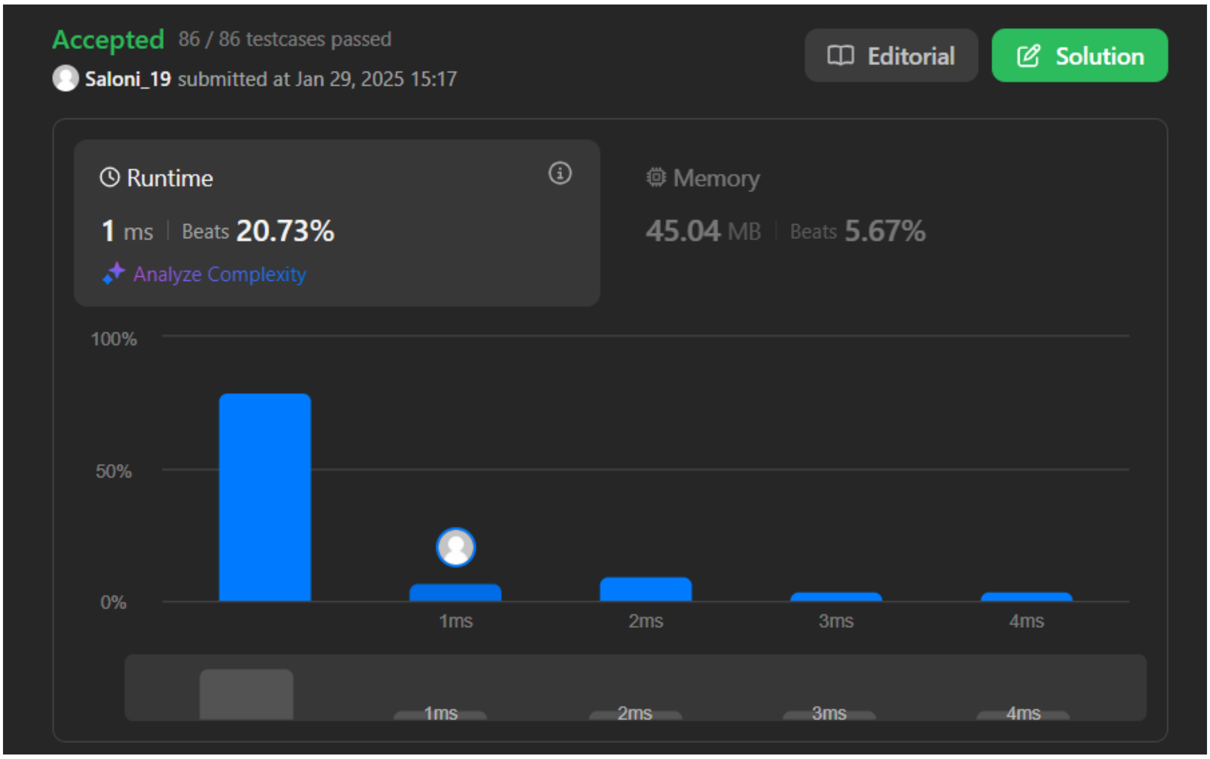
42.97 MB | Beats 26.28%

Code | Java

98. Validate Binary Search Tree

Code:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    public boolean isValidBST(TreeNode root) {
        List<Integer> res = new ArrayList<>();
        inorder(root, res);
        for(int i = 0; i<res.size()-1; i++){

            if(res.get(i) >= res.get(i+1)) return false;

        }
        return true;
    }
    private void inorder(TreeNode a , List<Integer> res){
        if(a == null) return;

        inorder(a.left, res);
        res.add(a.val);
        inorder(a.right, res);
    }

}
```



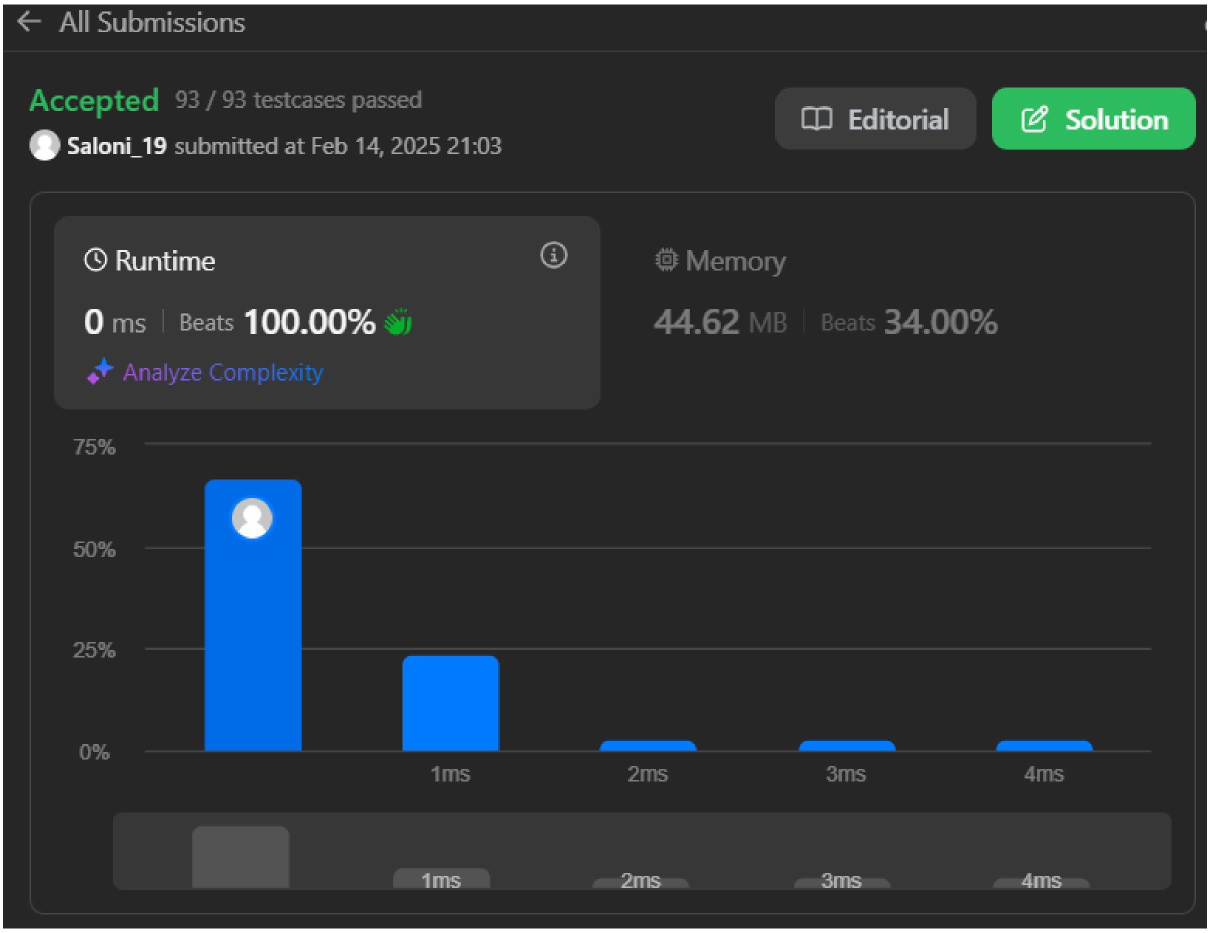
Accepted 86 / 86 testcases passed

Saloni_19 submitted at Jan 29, 2025 15:17

230. Kth Smallest Element in a BST

Code:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left = left;
 *         this.right = right;
 *     }
 * }
 */
class Solution {
    // Helper function to perform inorder traversal and store elements in an ArrayList
    public void inorder(TreeNode root, ArrayList<Integer> arr) {
        // Base case: if the node is null, return
        if (root == null) {
            return;
        }
        // Recursively visit the left subtree
        inorder(root.left, arr);
        // Add the current node's value to the list (in sorted order)
        arr.add(root.val);
        // Recursively visit the right subtree
        inorder(root.right, arr);
    }

    // Function to find the kth smallest element in a BST
    public int kthSmallest(TreeNode root, int k) {
        // Create an ArrayList to store inorder traversal elements
        ArrayList<Integer> fin = new ArrayList<>();
        // Perform inorder traversal (stores elements in sorted order)
        inorder(root, fin);
        // Return the (k-1)th element, as array index starts from 0
        return fin.get(k - 1);
    }
}
```



All Submissions

Accepted 93 / 93 testcases passed

Saloni_19 submitted at Feb 14, 2025 21:03

Runtime

0 ms | Beats 100.00%

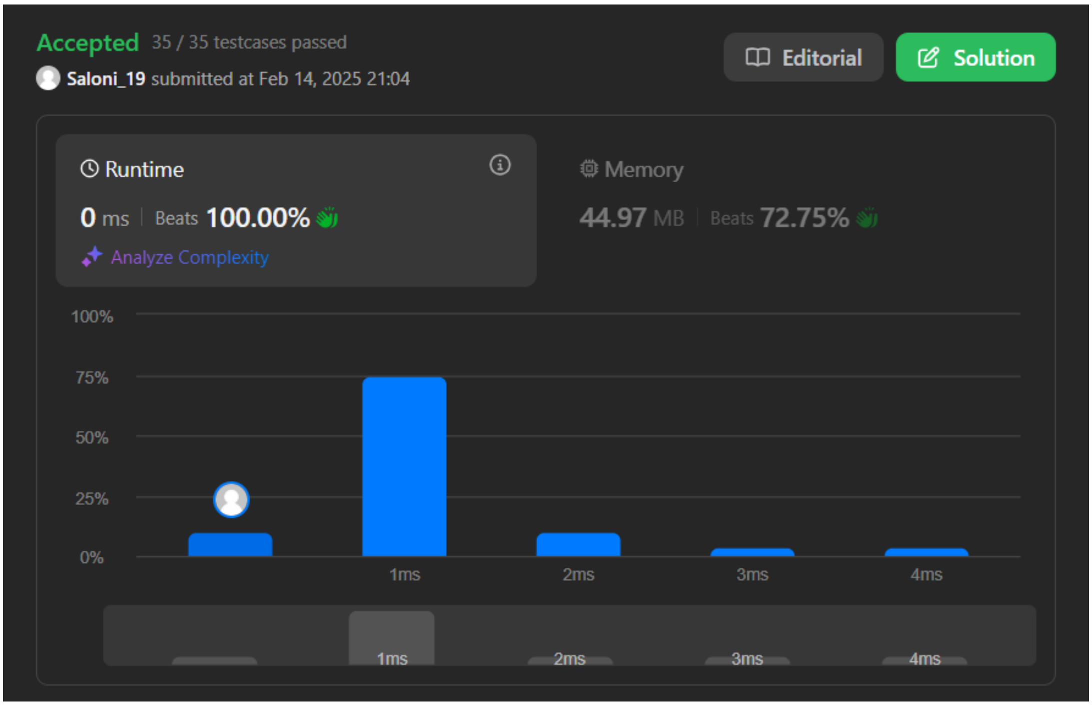
Memory

44.62 MB | Beats 34.00%

102. Binary Tree Level Order Traversal

Code:

```
class Solution {
   public List<List<Integer>> levelOrder(TreeNode root)
   {
      List<List<Integer>>al=new ArrayList<>();
      pre(root,0,al);
      return al;
   }
   public static void pre(TreeNode root,int l,List<List<Integer>>al)
   {
      if(root==null)
         return;
      if(al.size()==l)
      {
         List<Integer>li=new ArrayList<>();
         li.add(root.val);
         al.add(li);
      }
      else
         al.get(l).add(root.val);
      pre(root.left,l+1,al);
      pre(root.right,l+1,al);
   }
}
```



Accepted 35 / 35 testcases passed

Saloni_19 submitted at Feb 14, 2025 21:04

Runtime

0 ms | Beats 100.00%

Memory

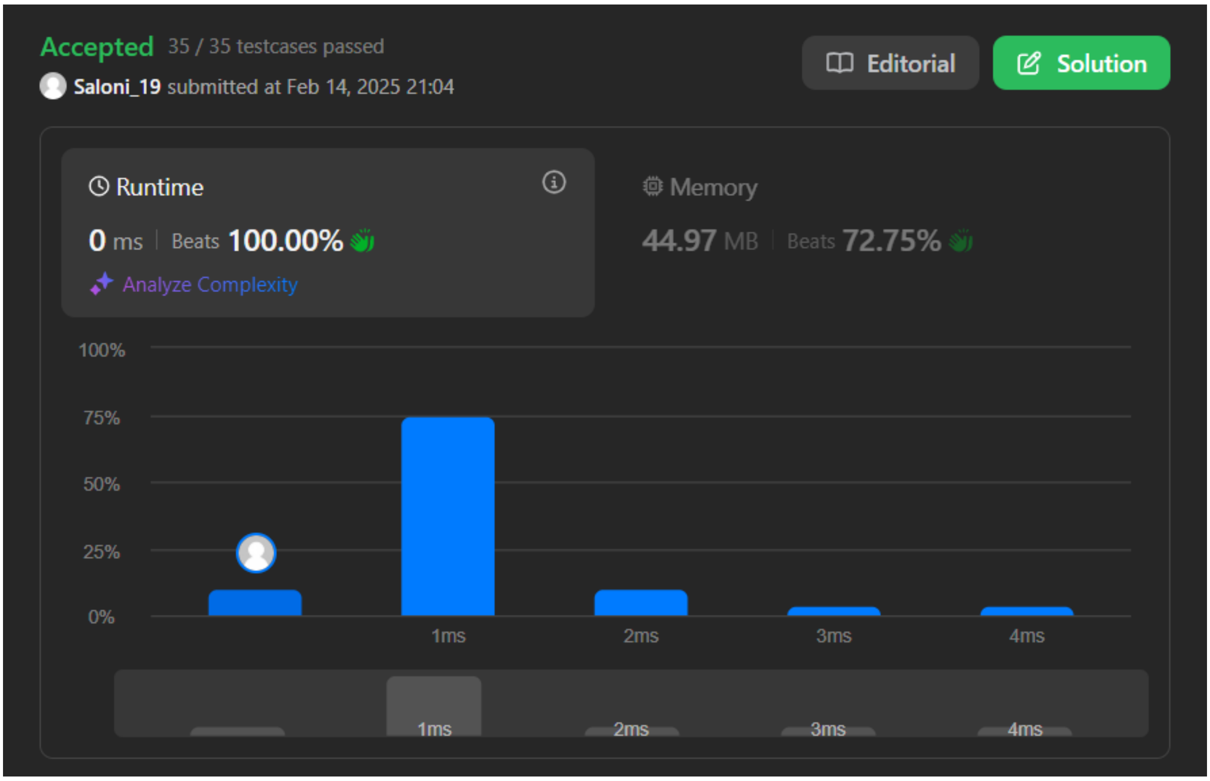
44.97 MB | Beats 72.75%

103. Binary Tree Zigzag Level Order Traversal

Code:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 * int val;
 * TreeNode left;
 * TreeNode right;
 * TreeNode() {}
 * TreeNode(int val) { this.val = val; }
 * TreeNode(int val, TreeNode left, TreeNode right) {
 * this.val = val;
 * this.left = left;
 * this.right = right;
 * }
 * }
 */
class Solution {
  public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
    if (root == null)
      return new LinkedList<>();
    Queue<TreeNode> queue = new LinkedList<>();
    List<List<Integer>> result = new ArrayList<>();
    queue.add(root);
    int lvl = 1;
    while (!queue.isEmpty()) {
      int levelWidth = queue.size();
      List<Integer> lvlItems = new ArrayList<>();
      // Iterate though all the nodes in the level adding them to the solution list.
      for (int i = 0; i < levelWidth; i++) {
        TreeNode node = queue.poll();
        if (lvl % 2 == 0) {
          lvlItems.addFirst(node.val); // Even: right to left
        } else {
          lvlItems.addLast(node.val); // Odd :  left to right
        }

        // Continue adding nodes to traverse
        if (node.left != null)
          queue.add(node.left);
```

```
                if (node.right != null)
                    queue.add(node.right);

            }
            // Add the level list and increment level
            result.add(lvlItems);
            lvl++;

        }
        return result;
    }
}
```



Accepted 35 / 35 testcases passed

Saloni_19 submitted at Feb 14, 2025 21:04

Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

44.97 MB | Beats 72.75%

987. Vertical Order Traversal of a Binary Tree

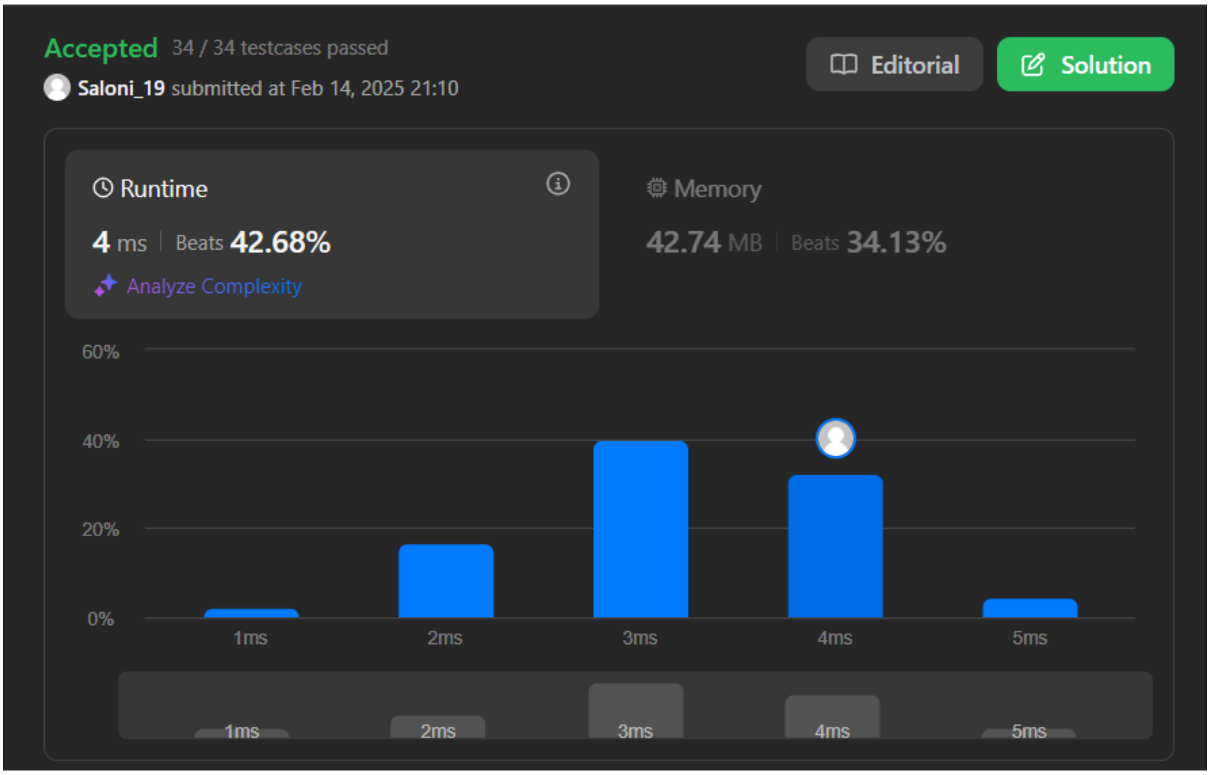
Code:

```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
```

```
 *        this.val = val;
 *        this.left = left;
 *        this.right = right;
 *    }
 * }
 */

 // Create class tuple to store the node and coordinates.
 class Tuple{
    TreeNode node;
    int row;
    int col;
    // Constructor for tuple.
    public Tuple(TreeNode _node, int _row, int _col){
        node = _node;
        row = _row;
        col = _col;
    }
 }
class Solution {

    // We perform Level order trversal to get the output....

    public List<List<Integer>> verticalTraversal(TreeNode root) {

        // We need a treemap to store the vertical values(columns) and PriorityQueue to store
the node values in increasing order.
        // (x,y,node)
        TreeMap<Integer,TreeMap<Integer,PriorityQueue<Integer>>> map = new TreeMap<>();

        // Create a queue for instering each node with respective row(x), column(y) values
during iteration.
        // Initially coordinates of node are...(node,x->(0),y->(0))
        Queue<Tuple> q = new LinkedList<Tuple>();

        // Insert the tuple
        q.add(new Tuple(root,0,0));

        // Loop untill queue is empty.
        while(!q.isEmpty()){
```

```
    // Pop the tuple from stack.
    Tuple tuple = q.poll();

    // Initialize the values inside the tuple.
    TreeNode node = tuple.node;
    int x = tuple.row;
    int y = tuple.col;

    // Insert the values into the treemap.

    // x - > vertical coordinate --> check example test cases.
    if(!map.containsKey(x)) map.put(x,new TreeMap<>());

    // y - > horizontal coordinate --> check example test cases.
    if(!map.get(x).containsKey(y)) map.get(x).put(y,new PriorityQueue<>());

    // Finally insert node value (!!!not node!!!) into map inside PriorityQueue.
    map.get(x).get(y).add(node.val);

    // Check is there exists a left or right node to the node present in the queue.
    // If present, then add it to the queue.
    if(node.left!=null) q.add(new Tuple(node.left,x-1,y+1));
    if(node.right!=null) q.add(new Tuple(node.right, x+1,y+1));
}

// Create a List Of List to store the list of node values.
List<List<Integer>> list = new ArrayList<>();

// Loop through the map and add the values.
// x - > key, (y, nodes) -> values.
for(TreeMap<Integer,PriorityQueue<Integer>> yn : map.values()){
    // Create a sublist to store node values in each vertical.
    list.add(new ArrayList<>());

    // Now iterate in the PriorityQueue.
    for(PriorityQueue<Integer> nodes : yn.values()){
        // Add node into the sublist from
        while(!nodes.isEmpty()){
            list.get(list.size()-1).add(nodes.poll());
        }
    }
```

```
        }return list;
    }
}
```



Accepted 34 / 34 testcases passed

Saloni_19 submitted at Feb 14, 2025 21:10

Runtime

4 ms | Beats 42.68%

Analyze Complexity

Memory

42.74 MB | Beats 34.13%