94.Binary Tree Inorder Traversal

Code

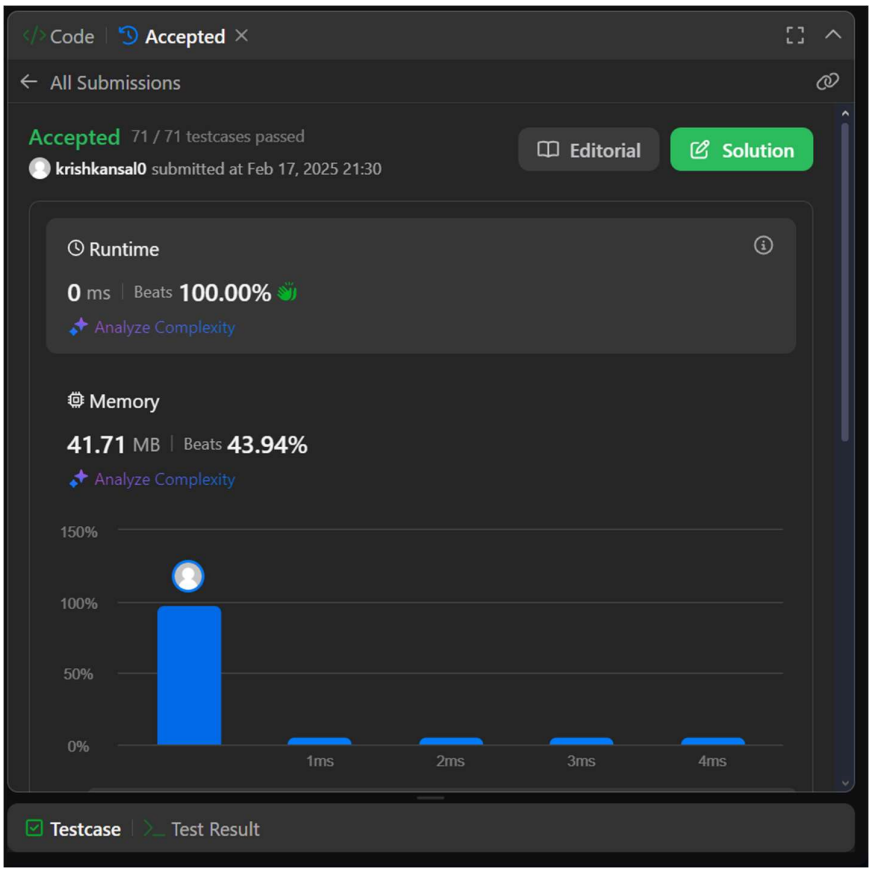
Java Auto

```java
import java.util.*;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}

class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        Stack<TreeNode> stack = new Stack<>();
        while (root != null || !stack.isEmpty()) {
            while (root != null) {
                stack.push(root);
                root = root.left;
            }
            root = stack.pop();
            result.add(root.val);
            root = root.right;
        }
        return result;
    }
}
```

Saved Ln 1, Col 1



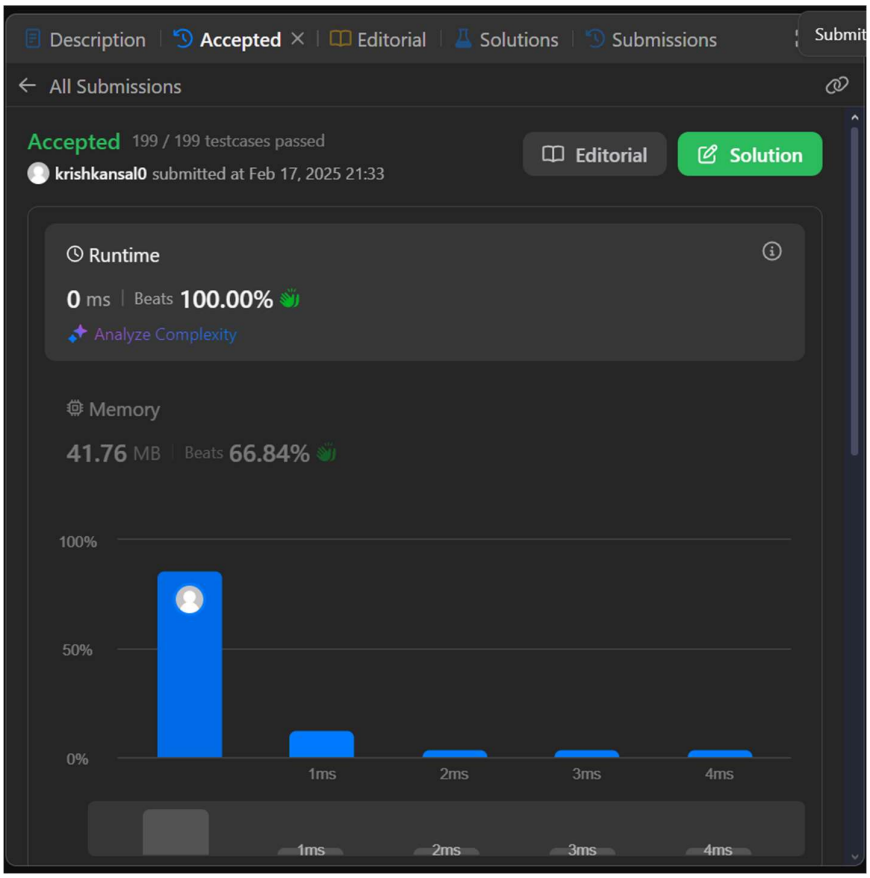
101.Symmetric Tree

```java
import java.util.*;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}

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

    private 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);
    }
}
```



104.Maximum Depth of Binary Tree

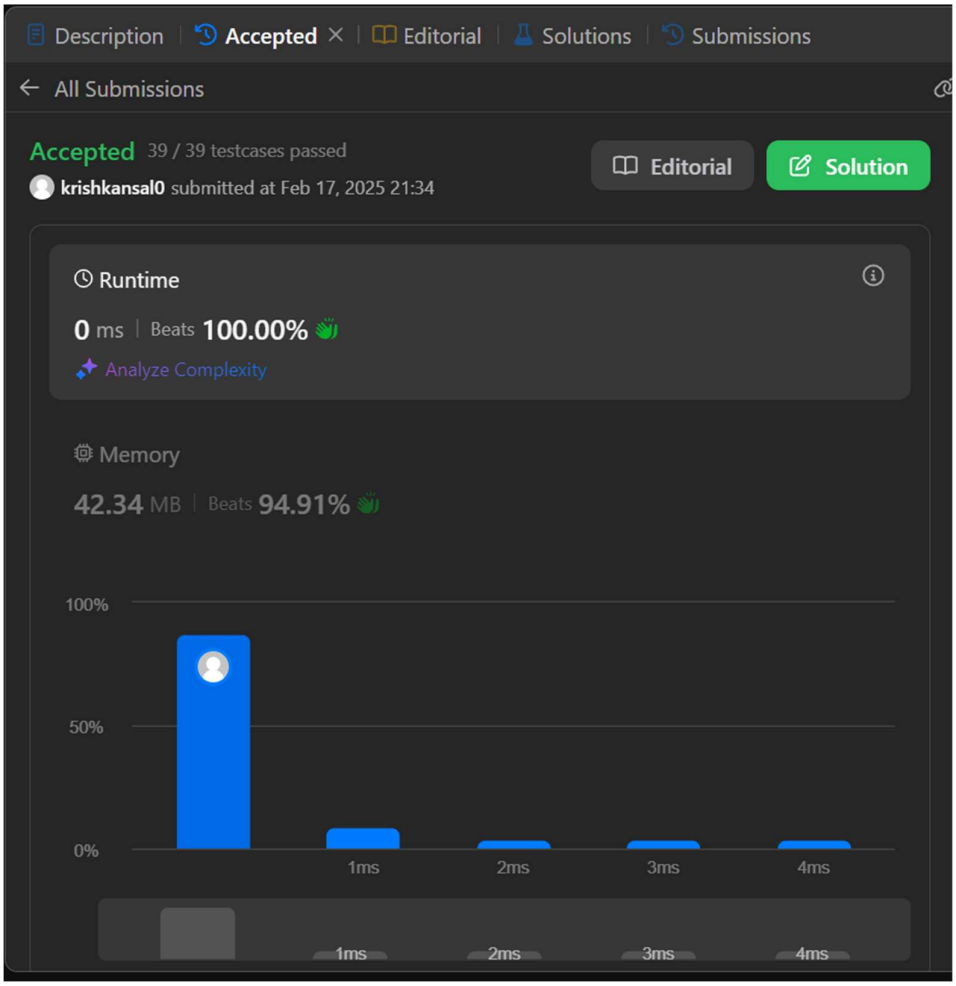
Code

Java Auto

```java
import java.util.*;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}

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



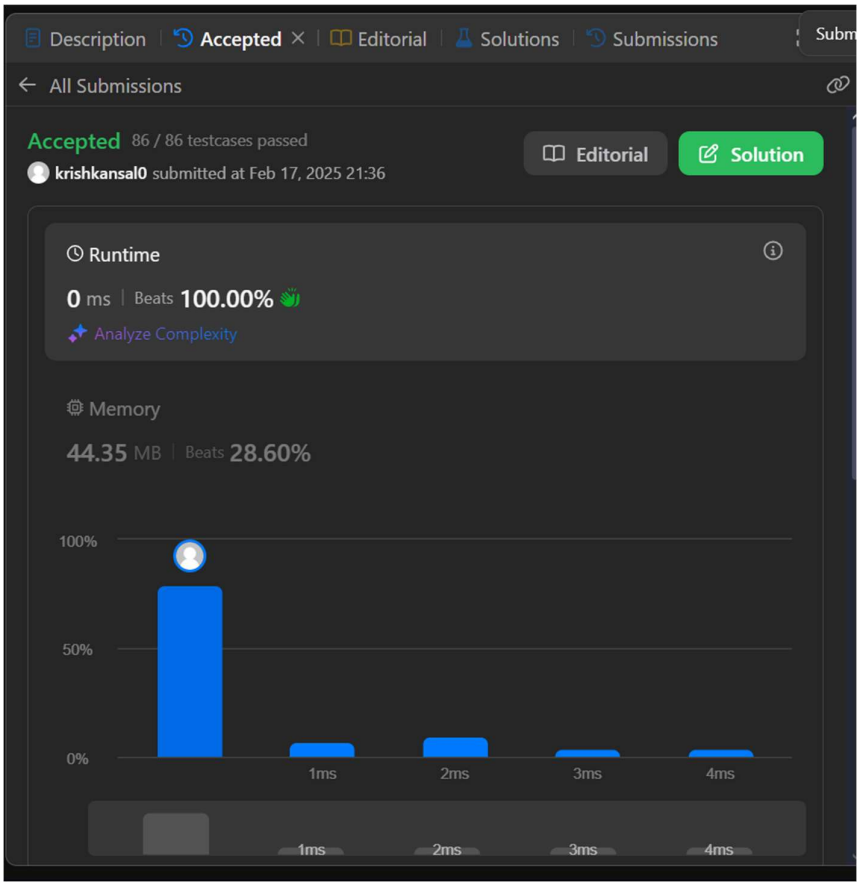
98.[Validate Binary Search Tree]

```java
import java.util.*;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}

class Solution {
    public boolean isValidBST(TreeNode root) {
        return validate(root, null, null);
    }

    private boolean validate(TreeNode node, Integer low, Integer high) {
        if (node == null) return true;
        if ((low != null && node.val <= low) || (high != null && node.val >= high)) return false;
        return validate(node.left, low, node.val) && validate(node.right, node.val, high);
    }
}
```

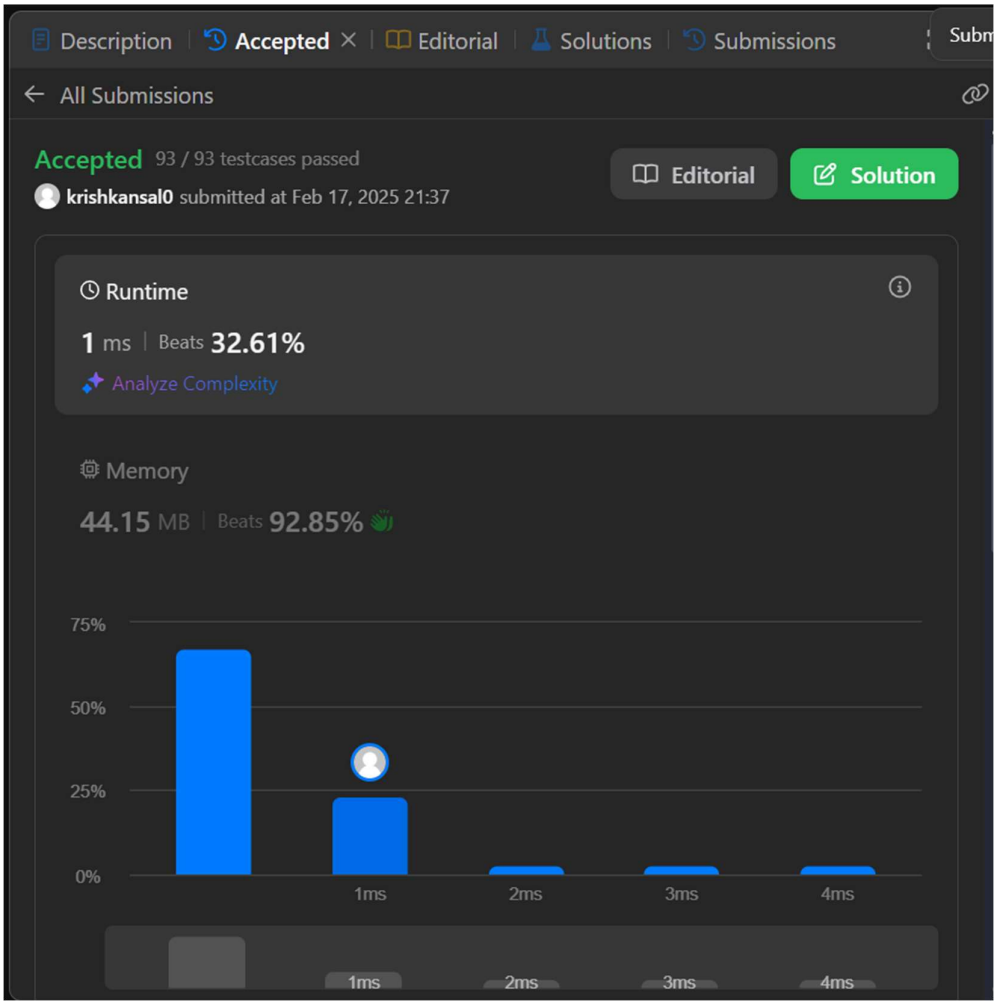


230.[Kth Smallest Element in a BST]

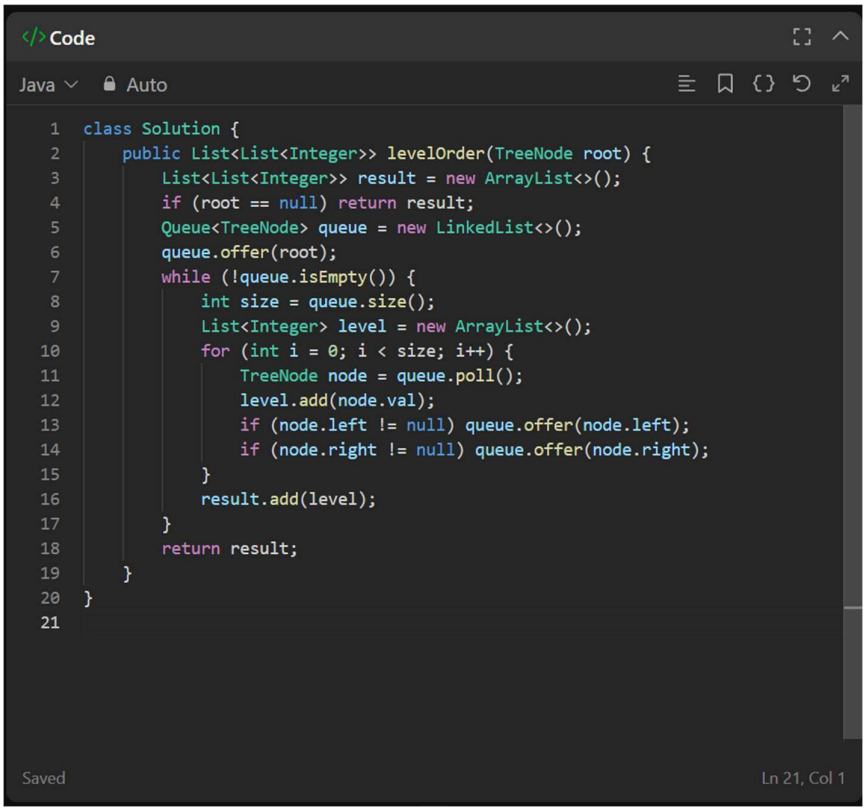
Code

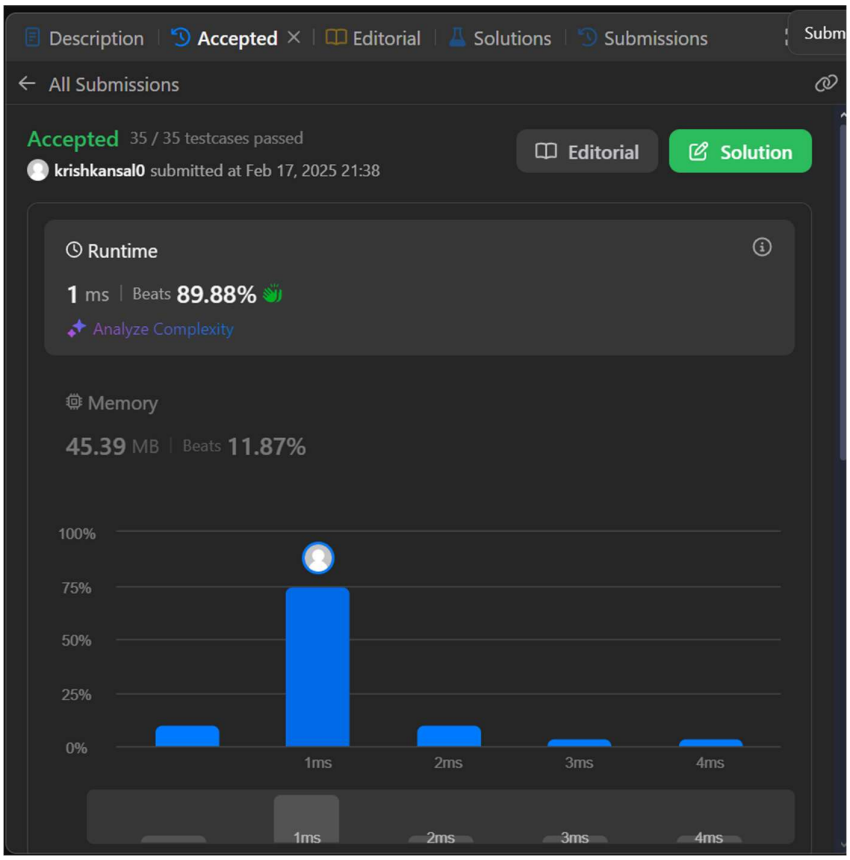
Java Auto

```java
class Solution {
    public int kthSmallest(TreeNode root, int k) {
        List<Integer> elements = new ArrayList<>();
        inorder(root, elements);
        return elements.get(k - 1);
    }
    static void inorder(TreeNode node, List<Integer> elements) {
        if (node == null) return;
        inorder(node.left, elements);
        elements.add(node.val);
        inorder(node.right, elements);
    }
}
```



102. [Binary Tree Level Order Traversal]



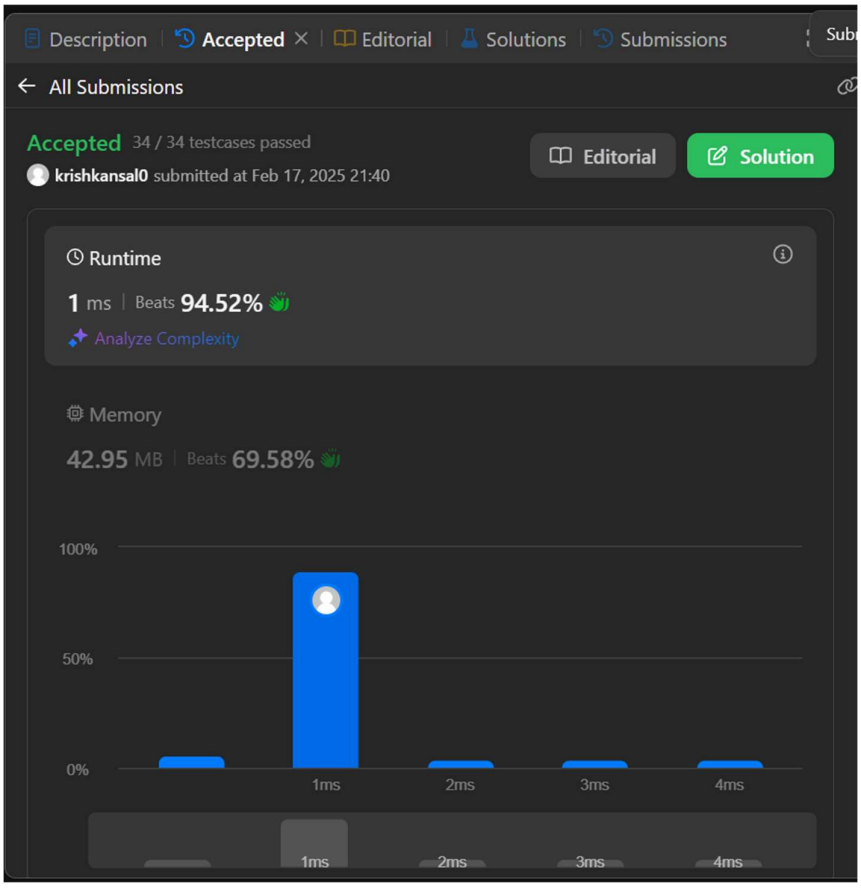


107.[Binary Tree Level Order Traversal II]

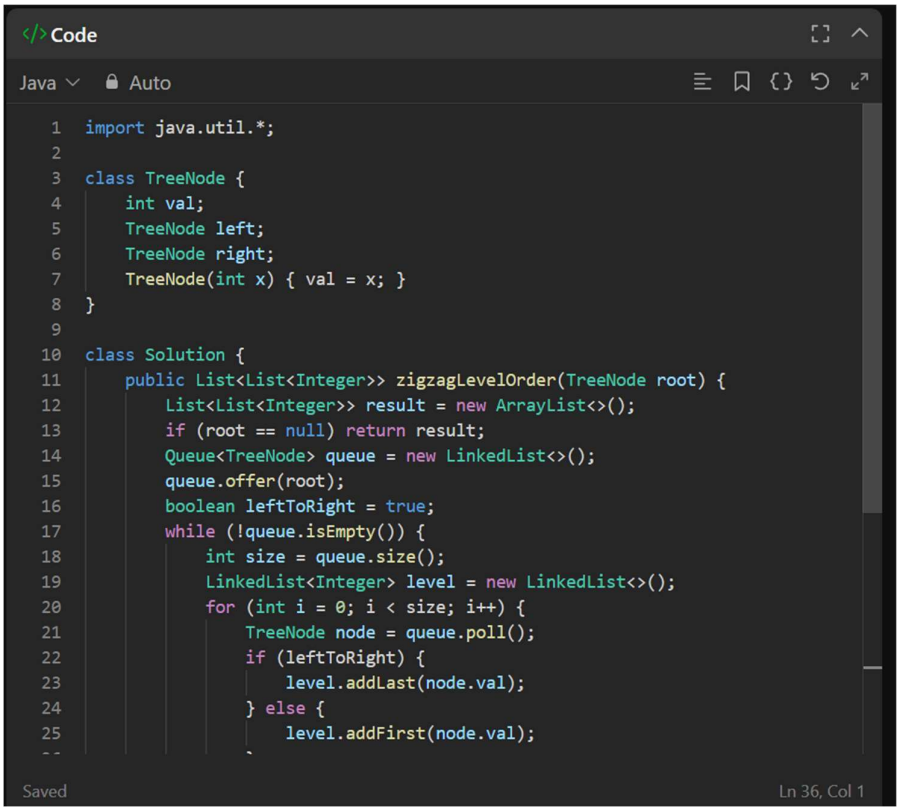
Code

Java Auto

```java
class Solution {
    public List<List<Integer>> levelOrderBottom(TreeNode root) {
        List<List<Integer>> result = new LinkedList<>();
        if (root == null) return result;
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        while (!queue.isEmpty()) {
            int size = queue.size();
            List<Integer> level = new ArrayList<>();
            for (int i = 0; i < size; i++) {
                TreeNode node = queue.poll();
                level.add(node.val);
                if (node.left != null) queue.offer(node.left);
                if (node.right != null) queue.offer(node.right);
            }
            result.add(0, level);
        }
        return result;
    }
}
```



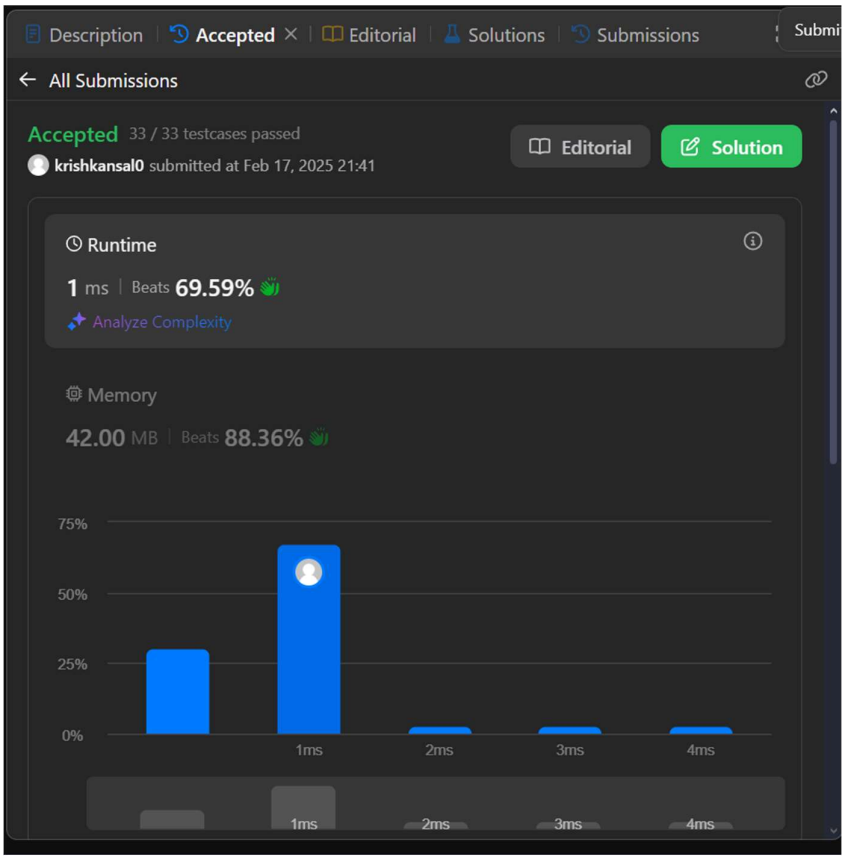
103.[Binary Tree Zigzag Level Order Traversal]



```java
import java.util.*;

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}

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.offer(root);
        boolean leftToRight = true;
        while (!queue.isEmpty()) {
            int size = queue.size();
            LinkedList<Integer> level = new LinkedList<>();
            for (int i = 0; i < size; i++) {
                TreeNode node = queue.poll();
                if (leftToRight) {
                    level.addLast(node.val);
                } else {
                    level.addFirst(node.val);
```

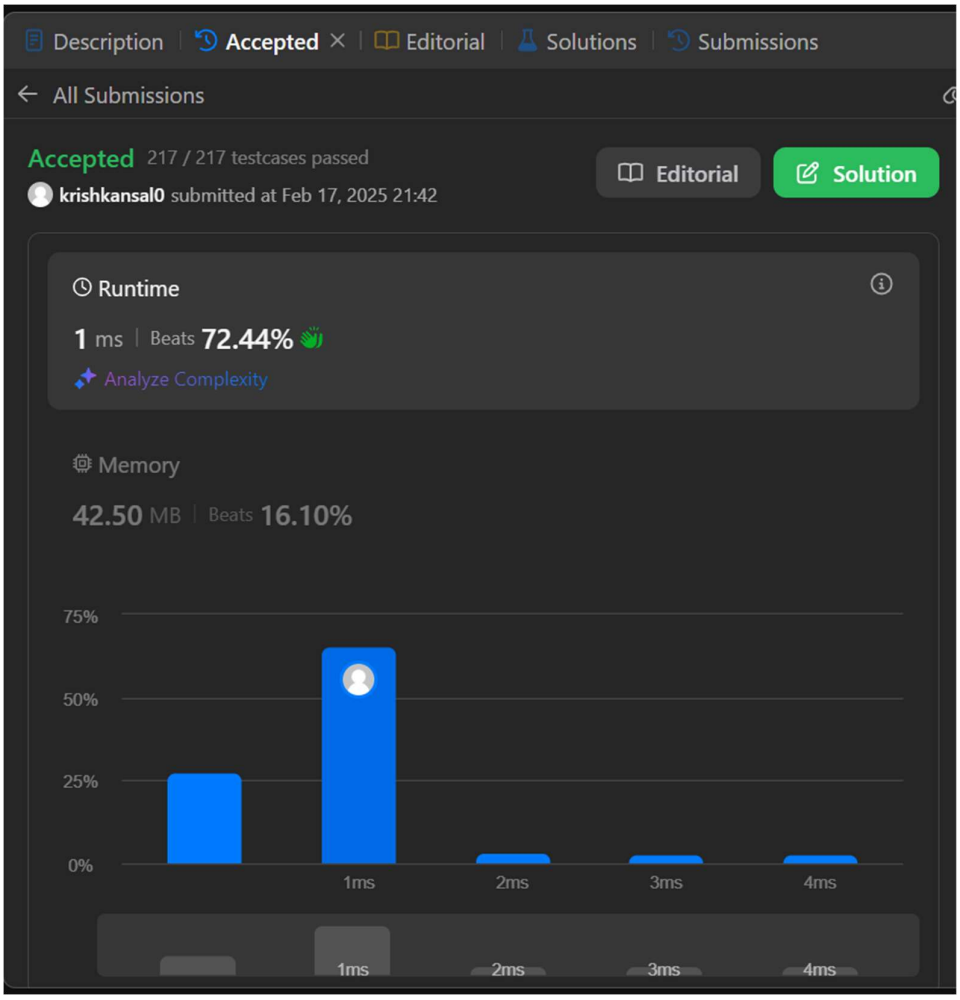


199.[Binary Tree Right Side View]

Code

Java | Auto

```java
class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        if (root == null) return result;
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        while (!queue.isEmpty()) {
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                TreeNode node = queue.poll();
                if (i == size - 1) {
                    result.add(node.val);
                }
                if (node.left != null) queue.offer(node.left);
                if (node.right != null) queue.offer(node.right);
            }
        }
        return result;
    }
}
```



106.[Construct Binary Tree from Inorder and Postorder Traversal]

Code

Java Auto

```java
class Solution {
    private int postIndex;

    public TreeNode buildTree(int[] inorder, int[] postorder) {
        postIndex = postorder.length - 1;
        return build(inorder, postorder, 0, inorder.length - 1);
    }

    private TreeNode build(int[] inorder, int[] postorder, int inStart, int inEnd) {
        if (inStart > inEnd) return null;

        TreeNode root = new TreeNode(postorder[postIndex--]);
        int inIndex = findIndex(inorder, inStart, inEnd, root.val);

        root.right = build(inorder, postorder, inIndex + 1, inEnd);
        root.left = build(inorder, postorder, inStart, inIndex - 1);

        return root;
    }

    private int findIndex(int[] inorder, int start, int end, int value) {
        for (int i = start; i <= end; i++) {
            if (inorder[i] == value) return i;
        }
```