WORKSHEET-6

Student Name: Rahul yadav **UID:** 22BCS13003

Branch: CSE **Section/Group:** NTPP-603-B

Semester: 6th **Date of Performance:** 20/2/25

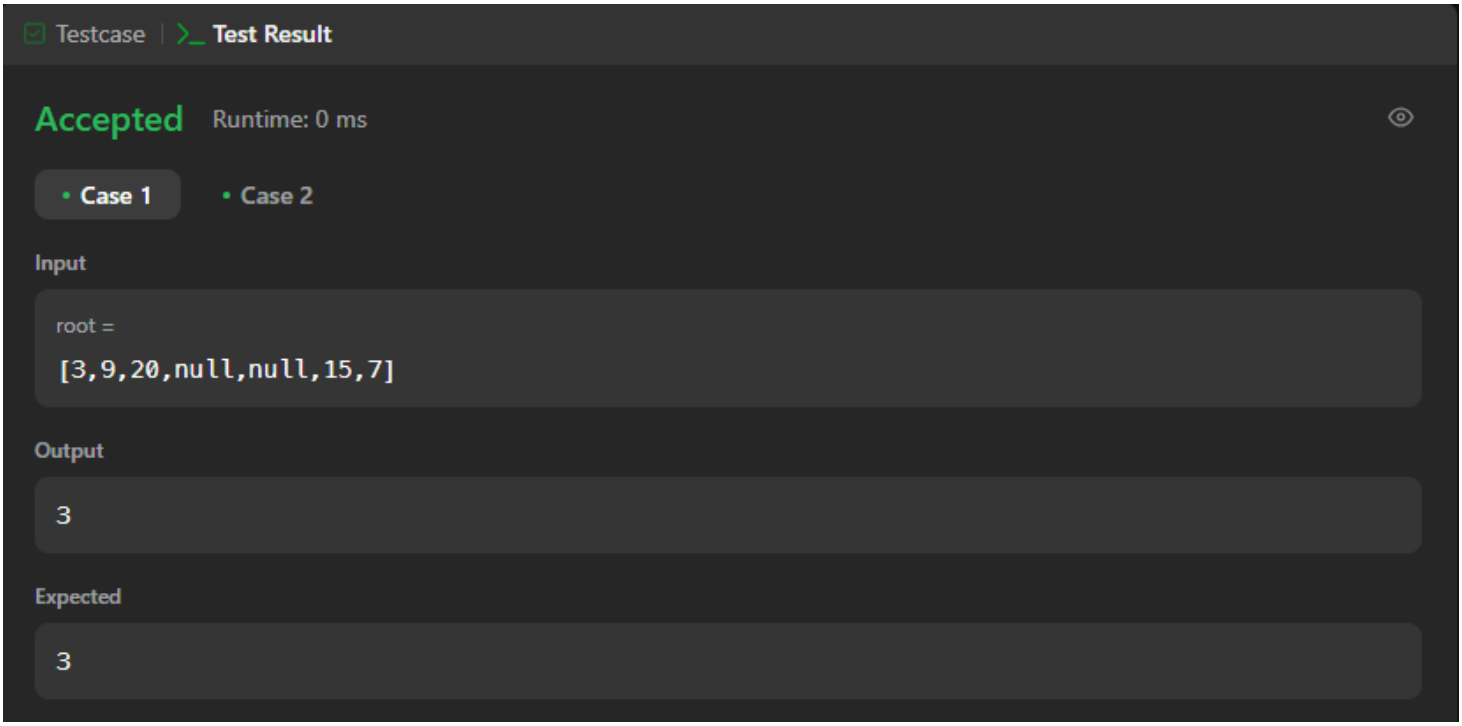
Subject Name: AP-2 **Subject Code:** 22CSP-351

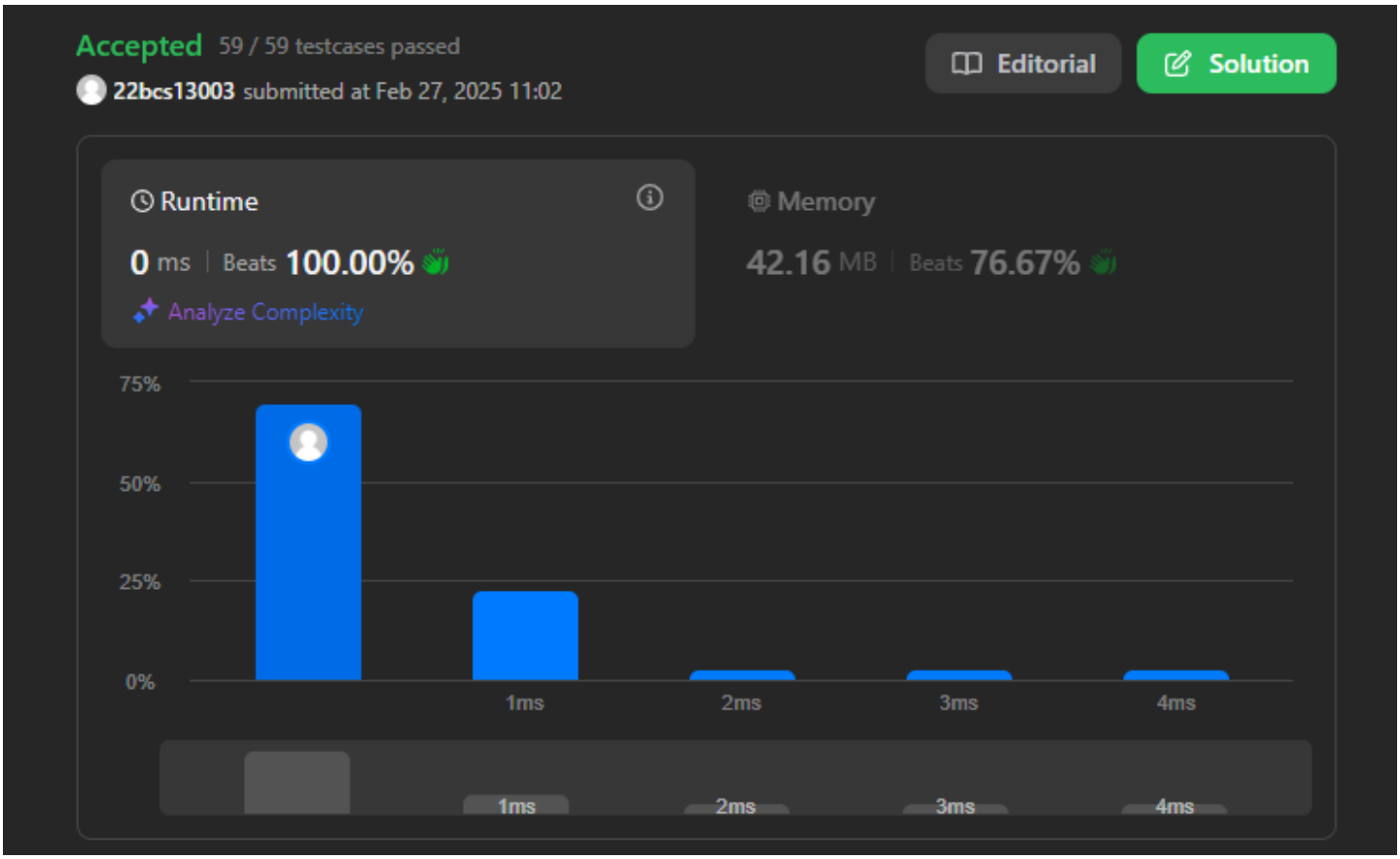
Aim(i): 88. You are given two integer arrays nums1 and nums2, sorted in non-decreasing order, and two integers m and n, representing the number of elements in nums1 and nums2 respectively. Merge nums1 and nums2 into a single array sorted in non-decreasing order.

Source Code:

```
public class Solution {
    public int maxDepth(TreeNode root) {
        // Base case: If the node is null, the depth is 0
        if (root == null) {
            return 0;
        }

        // Recursively get the depth of the left and right subtrees
        int leftDepth = maxDepth(root.left);
        int rightDepth = maxDepth(root.right);

        // The depth of the current node is the max of the left and right subtrees' depths + 1
        return Math.max(leftDepth, rightDepth) + 1;
    }
}
```

OUTPUT:





LEARNING OUTCOME:

1. We learnt Merge Sort.
2. We learnt how to sort Arrays.

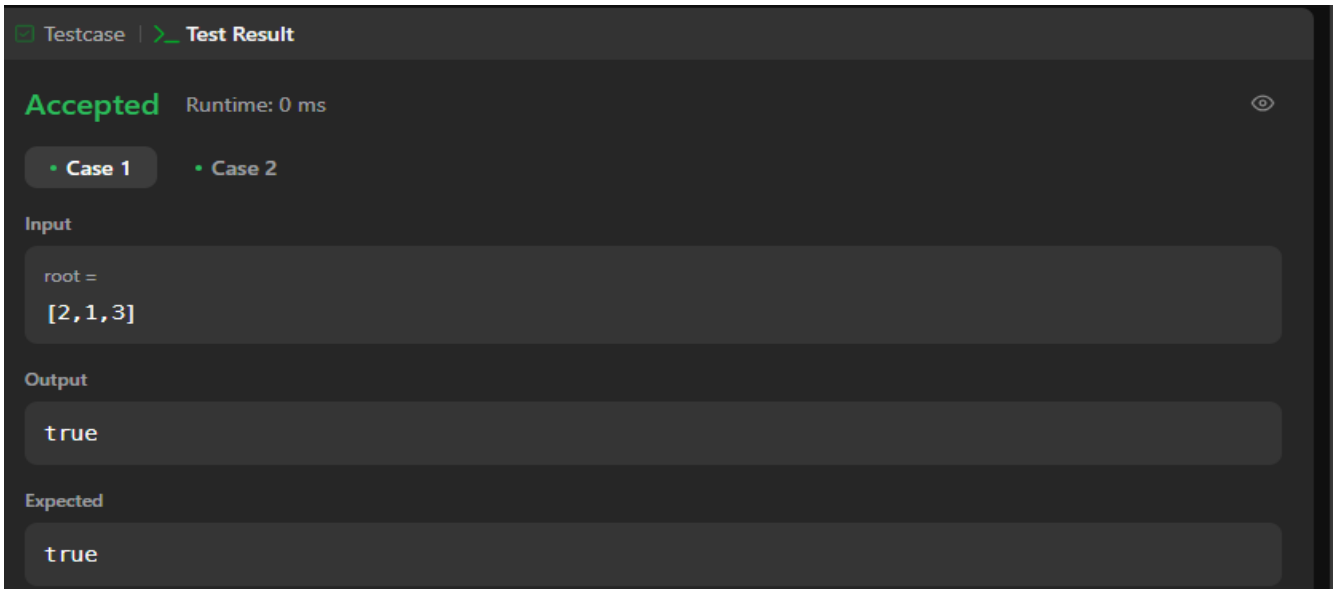
Aim(i): 98 A binary tree is a valid BST if:

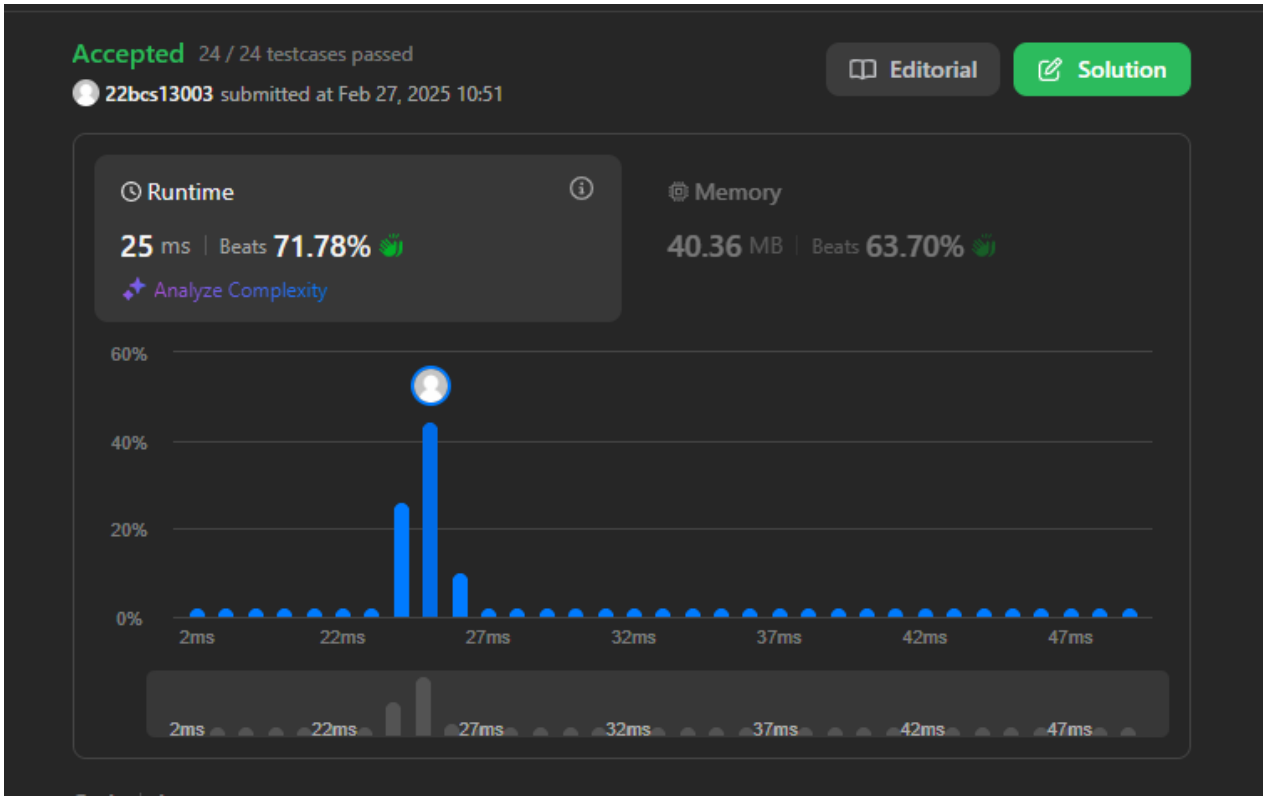
- The left subtree of a node contains only nodes with values **less than** the node's value.
- The right subtree of a node contains only nodes with values **greater than** the node's value.
- Both the left and right subtrees must also be binary search trees.

Source Code:

```
public class Solution {
    public boolean isValidBST(TreeNode root) {
        return isValidBSTHelper(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }

    // Helper function to validate each node's value with its valid range
    private boolean isValidBSTHelper(TreeNode node, long min, long max) {
        // Base case: if the node is null, it is valid
        if (node == null) {
            return true;
        }

        // Check if the current node's value is within the valid range
        if (node.val <= min || node.val >= max) {
            return false;
        }

        // Recursively check the left and right subtrees with updated ranges
        return isValidBSTHelper(node.left, min, node.val) &&
isValidBSTHelper(node.right, node.val, max);
    }
}
```

OUTPUT:


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
root =
[2,1,3]
Output
true
Expected
true


Accepted
24 / 24 testcases passed
22bcs13003 submitted at Feb 27, 2025 10:51
Editorial
Solution
Runtime
25 ms
Beats 71.78%
Analyze Complexity
Memory
40.36 MB
Beats 63.70%
60%
40%
20%
0%
2ms
22ms
27ms
32ms
37ms
42ms
47ms

Aim(iii): Given a binary tree, determine if it is **symmetric** around its center. A symmetric tree is a tree where:

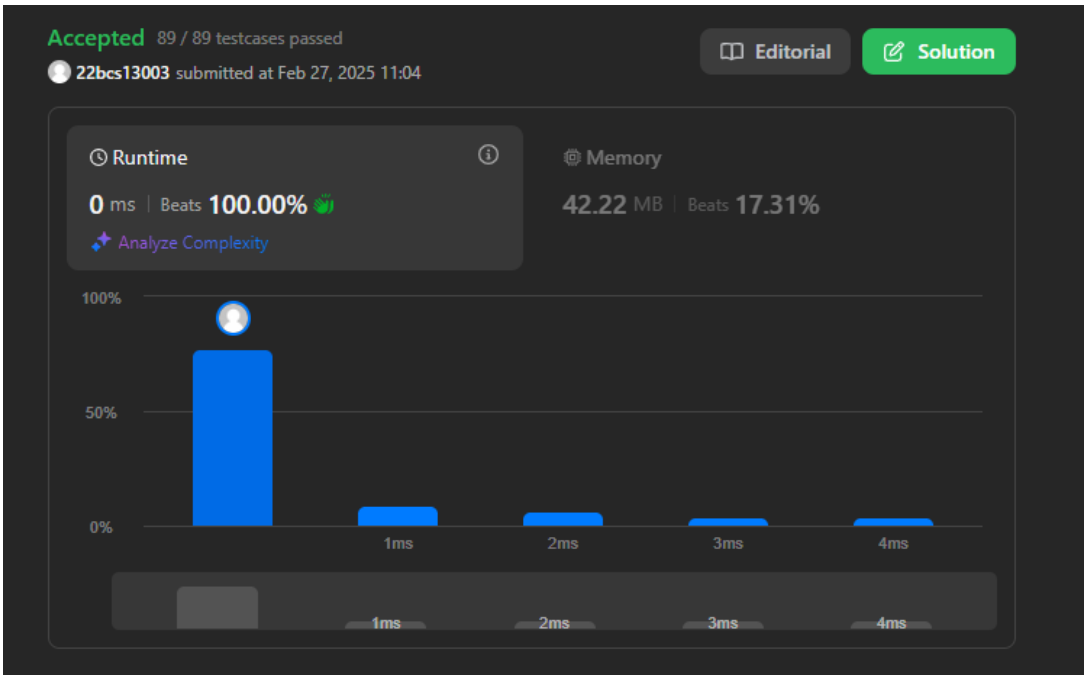
- The left and right subtrees are mirror images of each other.

Source Code:

```
public class Solution {
    public boolean isSymmetric(TreeNode root) {
        // If the root is null, the tree is symmetric (empty tree is symmetric).
        if (root == null) {
            return true;
        }

        // Check if the left and right subtrees are mirror images of each other.
        return isMirror(root.left, root.right);
    }

    // Helper function to check if two trees are mirror images of each other.
    private boolean isMirror(TreeNode left, TreeNode right) {
        // Base case: if both nodes are null, they are symmetric.
        if (left == null && right == null) {
            return true;
        }

        // If only one of them is null, they are not symmetric.
        if (left == null || right == null) {
            return false;
        }

        // The values at the current nodes must be the same and the left subtree of the left node
        // must be a mirror image of the right subtree of the right node, and vice versa.
        return (left.val == right.val)
            && isMirror(left.left, right.right)
            && isMirror(left.right, right.left);
    }
}
```

OUTPUT:


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
root =
[1,2,2,3,4,4,3]
Output
true
Expected
true


Accepted
89 / 89 testcases passed
22bcs13003 submitted at Feb 27, 2025 11:04
Editorial
Solution
Runtime
0 ms | Beats 100.00%
Analyze Complexity
Memory
42.22 MB | Beats 17.31%
100%
50%
0%
1ms
2ms
3ms
4ms

Learning Outcomes

1. We learnt Counting Sort.
2. Usage of a Hash Map.