WORKSHEET 6

Student Name: Shubham Chauhan **UID:22BCS11888**

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

Semester: 06 **Date of Performance: 27/02/2025**

Subject Name: AP Lab II **Subject Code: 22CSP-351**

1. **Aim**:

A. Maximum Depth of Binary Tree
B. Validate Binary Search Tree
C. Symmetric Tree

2. **Source Code:**

a.

```
/**
 * 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;
 *     }
 * }
```

```java
 */
// Definition for a binary tree node.
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;
    }
}

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

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

        // Return the maximum depth plus 1 (to count the current node)
```

```
        return Math.max(leftDepth, rightDepth) + 1;
    }
}
```

b.

```
/**
 * 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;
 *     }
 * }
 */
// Definition for a binary tree node.
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;
    }
}

public class Solution {
    public boolean isValidBST(TreeNode root) {
        // Call the helper function with the full valid range for the root
        return isValidBST(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }

    // Helper function to check the validity of the BST with valid range
    private boolean isValidBST(TreeNode node, long min, long max) {
        // Base case: if the node is null, it's valid (an empty tree is a valid BST)
        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 subtree and right subtree
```

```
        return isValidBST(node.left, min, node.val) && isValidBST(node.right, node.val, max);
    }
}
```

C.

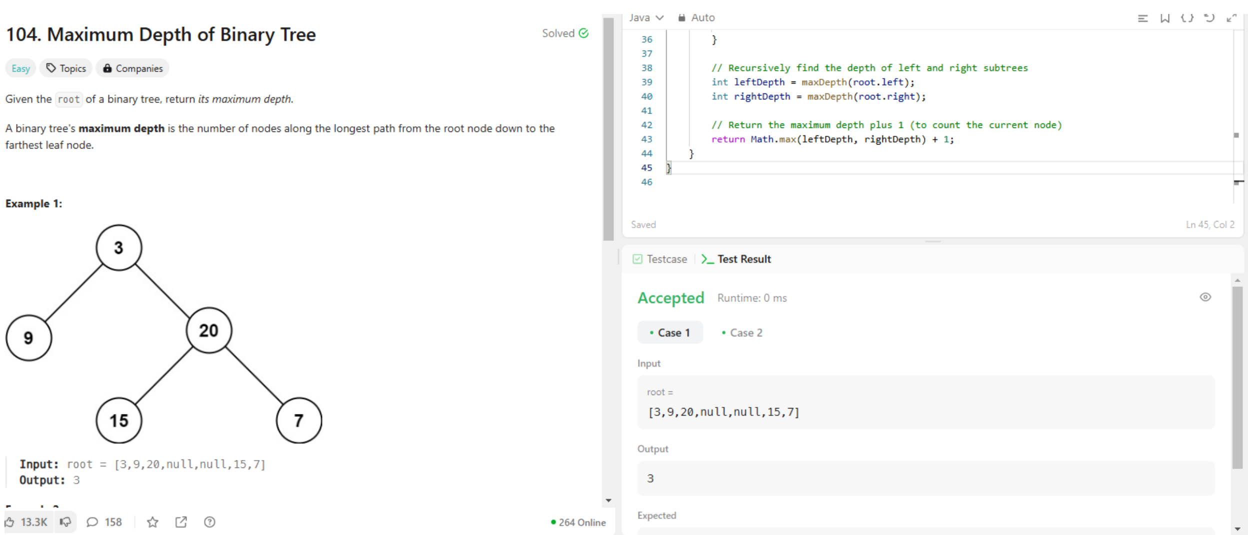
```
/**
 * Definition for a binary tree node.
 * public class TreeNode {
 *     int val;
 *     TreeNode left;
 *     TreeNode right;
 *     TreeNode() {}
 *     TreeNode(int val) { this.val = val; }
 *     TreeNode(int val, TreeNode left, TreeNode right) {
 *         this.val = val;
 *         this.left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

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

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

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

        // If one of them is null or the values are different, they are not mirrors
```

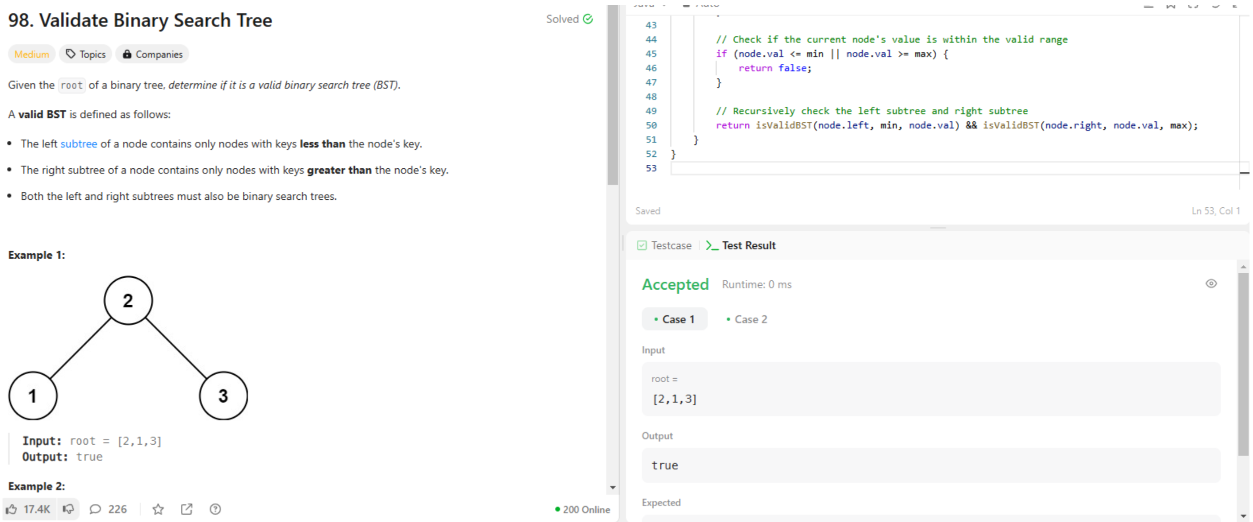
```java
        if (left == null || right == null || left.val != right.val) {
            return false;
        }

        // Recursively check if the left child of the left tree and the right child of the right tree
        // are mirrors, and if the right child of the left tree and the left child of the right tree are mirrors
        return isMirror(left.left, right.right) && isMirror(left.right, right.left);
    }
}
```

3. Screenshot of Outputs:

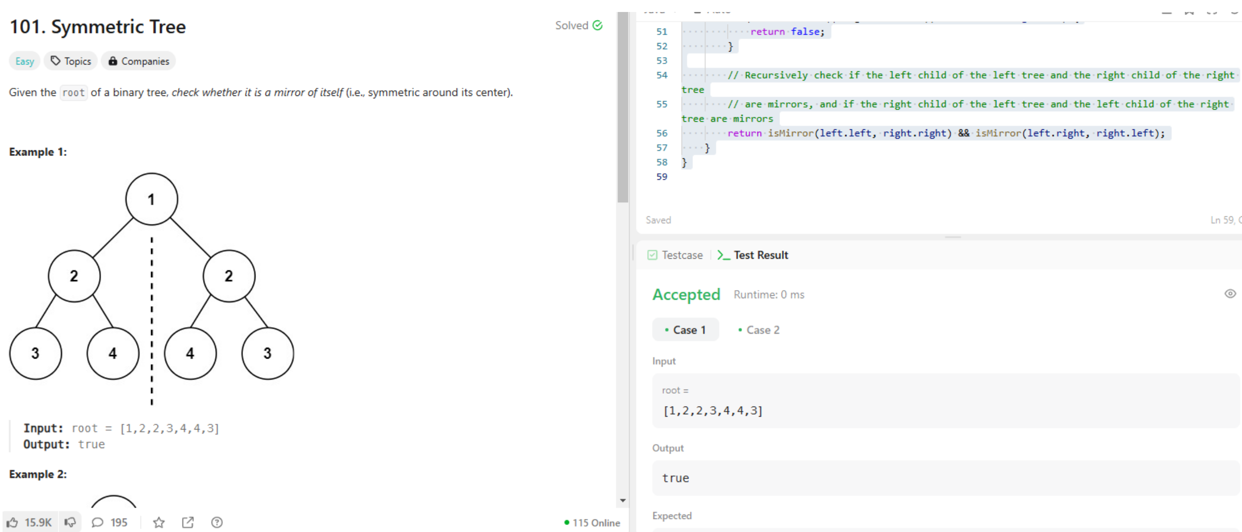
a.



b.



c.



4. Learning Outcomes

(i) Learned about trees .

(ii) Learned about types of trees.

(iii) Learned about tree traversal methods.