

## Assignment - 7

**Name: Aanchal Negi**

**Branch: BE-CSE**

**Semester: 6<sup>th</sup>**

**Subject Name: AP Lab-2**

**UID: 22BCS14969**

**Section/Group: IOT-609/B**

**Date :02/04/25**

**Subject Code: 22CSP-351**

### ➤ Maximum Depth of Binary Tree

**Code:**

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

        // Recursive case: Find the maximum depth of left and right subtrees
        int leftDepth = maxDepth(root.left);
        int rightDepth = maxDepth(root.right);

        // Return the maximum depth plus one for the current node
        return Math.max(leftDepth, rightDepth) + 1;
    }
}
```

**Output:**

The screenshot displays a code editor interface with a dark theme. The top navigation bar includes a 'Problem List' tab and a 'Premium' badge. The main editor area shows the Java code for the 'Maximum Depth of Binary Tree' problem. The code is as follows:

```
1 class Solution
2 {
3     public int maxDepth(TreeNode root)
4     {
5         // Base case: If the tree is empty, return 0
6         if (root == null)
7         {
8             return 0;
9         }
10
11         // Recursive case: Find the maximum depth of left and right subtrees
12         int leftDepth = maxDepth(root.left);
13         int rightDepth = maxDepth(root.right);
14
15         // Return the maximum depth plus one for the current node
16         return Math.max(leftDepth, rightDepth) + 1;
17     }
18 }
```

On the left side, the 'Runtime' section shows '0 ms' and '100.00%' with a green checkmark. The 'Memory' section shows '42.69 MB' and '61.32%' with a green checkmark. Below these, a progress bar indicates 100% completion. The 'Testcase' section shows 'Accepted' with a green checkmark and 'Runtime: 0 ms'. The 'Input' section is empty.

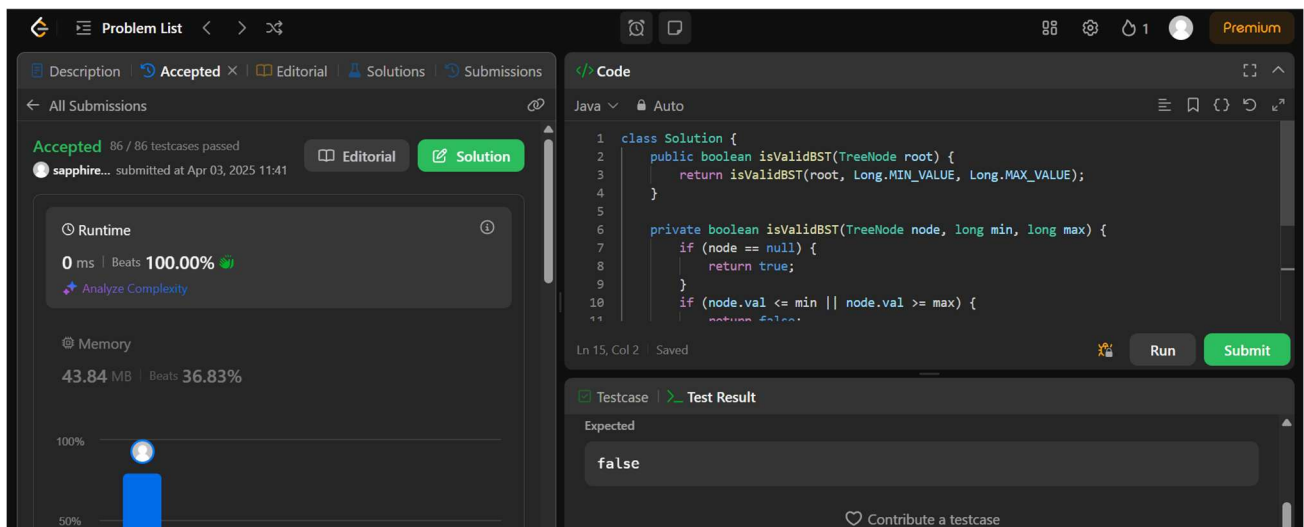
## ➤ Validate Binary Search Tree

### Code:

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

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

### Output:



The screenshot displays the LeetCode interface for the 'Validate Binary Search Tree' problem. The left sidebar shows the problem status as 'Accepted' with 86/86 testcases passed. The user 'sapphire...' submitted the solution on April 03, 2025, at 11:41. The runtime is 0 ms, beating 100.00% of solutions, and the memory usage is 43.84 MB, beating 36.83% of solutions. The right panel shows the Java code for the solution, which is a recursive function that checks if a binary tree is a valid BST by comparing node values with a minimum and maximum range. The bottom right panel shows the test result for a specific test case, where the expected output is 'false'.

Accepted 86 / 86 testcases passed  
sapphire... submitted at Apr 03, 2025 11:41

Runtime  
0 ms | Beats 100.00%  
Analyze Complexity

Memory  
43.84 MB | Beats 36.83%

Code

```
1 class Solution {
2     public boolean isValidBST(TreeNode root) {
3         return isValidBST(root, Long.MIN_VALUE, Long.MAX_VALUE);
4     }
5
6     private boolean isValidBST(TreeNode node, long min, long max) {
7         if (node == null) {
8             return true;
9         }
10        if (node.val <= min || node.val >= max) {
11            return false;
12        }
13        return isValidBST(node.left, min, node.val) && isValidBST(node.right, node.val, max);
14    }
15 }
```

Testcase | Test Result

Expected  
false

Contribute a testcase

## ➤ Symmetric Tree

### Code:

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

    private boolean isMirror(TreeNode t1, TreeNode t2) {
        if (t1 == null && t2 == null) {
            return true; // Both subtrees are null, so they're symmetric
        }
        if (t1 == null || t2 == null) {
            return false; // One of the subtrees is null, so not symmetric
        }
        // Check if the current nodes are equal and their subtrees are mirrors of each other
        return (t1.val == t2.val) && isMirror(t1.left, t2.right) && isMirror(t1.right, t2.left);
    }
}
```

### Output:

The screenshot displays a web-based code editor interface. On the left, the 'Problem List' tab is active, showing the 'Symmetric Tree' problem. The 'Accepted' status is confirmed with '200 / 200 testcases passed'. The submission details show it was submitted by 'sapphire...' on April 04, 2025, at 20:09. The 'Runtime' section indicates 0 ms execution time, beating 100.00% of other submissions. The 'Memory' section shows 42.12 MB usage, beating 20.36% of other submissions. The 'Code' tab on the right shows the Java solution, which is identical to the code provided in the previous block. The 'Testcase' tab shows the result for 'Case 1' as 'Accepted' with a runtime of 0 ms.

## ➤ Binary Tree Level Order Traversal

### Code:

```
import java.util.*;

class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        if (root == null) {
            return result; // Return an empty list if the tree is empty
        }

        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root); // Start with the root node

        while (!queue.isEmpty()) {
            int levelSize = queue.size(); // Number of nodes in the current level
            List<Integer> level = new ArrayList<>();

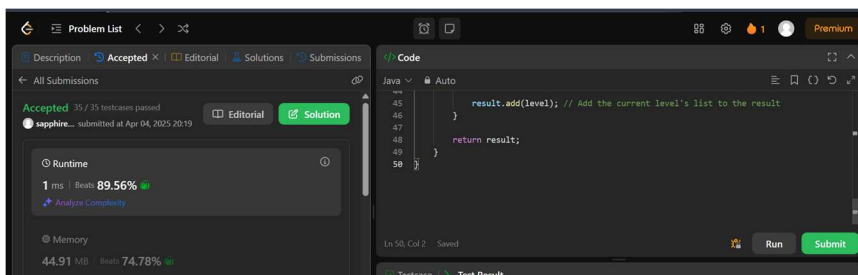
            for (int i = 0; i < levelSize; i++) {
                TreeNode currentNode = queue.poll(); // Remove the front node from the queue
                level.add(currentNode.val); // Add its value to the current level's list

                // Add left and right children to the queue if they exist
                if (currentNode.left != null) {
                    queue.add(currentNode.left);
                }
                if (currentNode.right != null) {
                    queue.add(currentNode.right);
                }
            }

            result.add(level); // Add the current level's list to the result
        }

        return result;
    }
}
```

### Output:



## ➤ Convert Sorted Array to Binary Search Tree

### Code:

```
class Solution {
    public TreeNode sortedArrayToBST(int[] nums) {
        return helper(nums, 0, nums.length - 1);
    }

    private TreeNode helper(int[] nums, int left, int right) {
        // Base case: If the range is invalid
        if (left > right) {
            return null;
        }

        // Find the middle element to make it the root
        int mid = left + (right - left) / 2;
        TreeNode root = new TreeNode(nums[mid]);

        // Recursively construct the left and right subtrees
        root.left = helper(nums, left, mid - 1);
        root.right = helper(nums, mid + 1, right);

        return root;
    }
}
```

### Output:

The screenshot displays a code editor interface for a problem titled "Convert Sorted Array to Binary Search Tree". The left sidebar shows the problem status as "Accepted" with 31/31 testcases passed, submitted by "sapphire..." on April 04, 2025. The runtime is 0 ms (Beats 100.00%) and memory is 43.36 MB (Beats 43.71%). The main editor shows the Java code for the solution, which is a recursive function to construct a balanced BST from a sorted array. The right sidebar shows the "Test Result" section, indicating the solution is "Accepted" with a runtime of 0 ms for Case 1.

```
30
31
32 // Recursively construct the left and right subtrees
33 root.left = helper(nums, left, mid - 1);
34 root.right = helper(nums, mid + 1, right);
35
36 return root;
37 }
```

Ln 37, Col 2 Saved Run Submit

Testcase Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

## ➤ Binary Tree Inorder Traversal

### Code:

```
import java.util.ArrayList;
import java.util.List;

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; // Base case: if node is null, do nothing
        }
        inorderHelper(node.left, result); // Visit left subtree
        result.add(node.val); // Visit current node
        inorderHelper(node.right, result); // Visit right subtree
    }
}
```

### Output:

The screenshot displays the LeetCode submission interface for the problem "Binary Tree Inorder Traversal". The interface is divided into several sections:

- Problem List:** Shows the problem name and a "Premium" badge.
- Accepted:** Indicates that 71 out of 71 testcases passed. The submission is by "sapphire..." and was submitted on Apr 04, 2025 at 20:36.
- Runtime:** Shows a runtime of 0 ms, which beats 100.00% of other submissions. A link to "Analyze Complexity" is provided.
- Memory:** Shows a memory usage of 41.32 MB, which beats 96.69% of other submissions.
- Code:** Displays the Java code for the solution, which is the same as the code provided in the "Code" section above. The code is in a dark-themed editor with line numbers 17 to 27.
- Testcase:** Shows the test results for the submission. The status is "Accepted" with a runtime of 0 ms. There are four test cases listed: Case 1, Case 2, Case 3, and Case 4, all of which passed.

## ➤ Construct Binary Tree from Inorder and Postorder Traversal

**Code:**

**Code:**

```
class Solution {
    public TreeNode buildTree(int[] inorder, int[] postorder) {
        return buildTreeHelper(inorder, 0, inorder.length - 1, postorder, 0, postorder.length - 1);
    }

    private TreeNode buildTreeHelper(int[] inorder, int inStart, int inEnd,
                                     int[] postorder, int postStart, int postEnd) {
        // Base case: if no elements to construct the tree
        if (inStart > inEnd || postStart > postEnd) {
            return null;
        }

        // The root is the last element of the postorder traversal
        TreeNode root = new TreeNode(postorder[postEnd]);

        // Find the index of the root in the inorder array
        int rootIndex = inStart;
        while (rootIndex <= inEnd && inorder[rootIndex] != root.val) {
            rootIndex++;
        }

        // Calculate the size of the left subtree
        int leftSize = rootIndex - inStart;

        // Recursively construct the left and right subtrees
        root.left = buildTreeHelper(inorder, inStart, rootIndex - 1, postorder, postStart, postStart + leftSize - 1);
        root.right = buildTreeHelper(inorder, rootIndex + 1, inEnd, postorder, postStart + leftSize, postEnd - 1);

        return root;
    }
}
```

**Output:**

The screenshot displays a coding platform interface with the following components:

- Problem List:** A navigation bar at the top with icons for problem list, accepted solutions, editorial, solutions, and submissions.
- Submissions:** A section showing the submission status as "Accepted" with 202 / 202 testcases passed. It includes the username "sapphire..." and the submission time "Apr 04, 2025 20:44".
- Runtime and Memory:** Performance metrics are shown: Runtime is 3 ms (Beats 28.64%) and Memory is 44.16 MB (Beats 87.76%).
- Code Editor:** The Java code for the solution is displayed, matching the code provided in the previous block. The editor includes line numbers and a "Run" button.
- Test Results:** A section at the bottom showing the test results as "Accepted" with a runtime of 0 ms. It lists "Case 1" and "Case 2" as passed.

## ➤ Kth Smallest element in a BST

### Code:

```
class Solution {
    public int kthSmallest(TreeNode root, int k) {
        // Initialize a stack for iterative inorder traversal
        Stack<TreeNode> stack = new Stack<>();
        TreeNode current = root;

        while (current != null || !stack.isEmpty()) {
            // Traverse to the leftmost node
            while (current != null) {
                stack.push(current);
                current = current.left;
            }

            // Process the node
            current = stack.pop();
            k--;

            // If k becomes 0, we have found the kth smallest element
            if (k == 0) {
                return current.val;
            }

            // Move to the right subtree
            current = current.right;
        }

        // If k is invalid, return -1 (though this shouldn't happen for valid input)
        return -1;
    }
}
```

### Output:

The screenshot displays a code editor interface for a problem titled "Kth Smallest element in a BST". The left sidebar shows the "Problem List" and "Submissions" tabs. The "Submissions" tab is active, showing a submission by "sapphire..." that is "Accepted" with 93/93 testcases passed. The submission was made on Apr 04, 2025 at 20:52. The "Runtime" section shows 0 ms and 100.00% beats, with a link to "Analyze Complexity". The "Memory" section shows 45.04 MB and 5.72% beats. A blue bar chart shows the submission's performance relative to others. The main editor area shows the Java code for the solution, which is the same code as provided in the "Code" section. The bottom right shows the "Testcase" and "Test Result" section, indicating the solution is "Accepted" with a runtime of 0 ms. The "Input" field is empty.



## ➤ Populating Next Right Pointers in Each Node

### Code:

```
class Solution {
    public Node connect(Node root) {
        if (root == null) {
            return null; // If the tree is empty
        }

        // Start with the root node
        Node levelStart = root;

        while (levelStart.left != null) { // Traverse levels until leaf nodes
            Node current = levelStart;

            while (current != null) {
                // Connect left child to right child
                current.left.next = current.right;

                // Connect right child to the left child of the next node
                if (current.next != null) {
                    current.right.next = current.next.left;
                }

                // Move to the next node in the level
                current = current.next;
            }

            // Move to the next level
            levelStart = levelStart.left;
        }

        return root;
    }
}
```

### Output:

The screenshot displays the LeetCode submission interface for the problem "Populating Next Right Pointers in Each Node". The interface is divided into several sections:

- Submission Status:** Shows "Accepted" with 59/59 testcases passed. The user "sapphire..." submitted on Apr 04, 2025 at 21:01.
- Runtime and Memory:** The runtime is 0 ms, beating 100.00% of submissions. The memory usage is 44.52 MB, beating 42.93% of submissions.
- Code Editor:** Displays the Java code for the solution, which is the same as the code provided in the previous block. The code is in Java and uses a while loop to traverse the tree level by level, connecting nodes.
- Test Result:** Shows the result of the test cases, indicating that the solution is "Accepted" with a runtime of 0 ms. It also lists "Case 1" and "Case 2" as test cases.