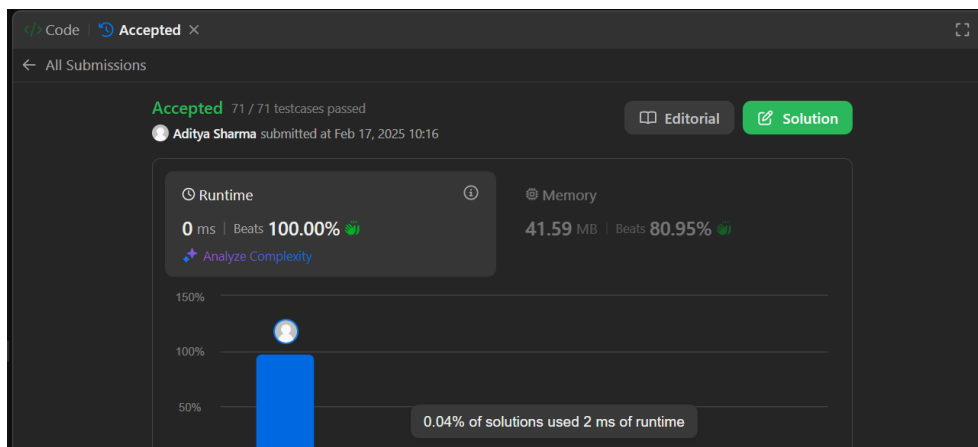


AP: ASSIGNMENT 3

NAME: Aditya Sharma	UID: 22BCS10116
SECTION: 22BSC_FL_IOT-601	GROUP: A

94. [Binary Tree Inorder Traversal](#)

```
/**
 * 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> res = new ArrayList<>();
        traverse(root, res);
        return res;
    }
    void traverse(TreeNode root, List<Integer> res){
        if(root == null) return;
        traverse(root.left, res);
        res.add(root.val);
        traverse(root.right, res);
    }
}
```



101. Symmetric Tree

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

The screenshot displays a coding platform interface for the '101. Symmetric Tree' problem. The 'Description' tab is active, showing the problem statement and the provided Java solution. The 'Runtime' section indicates that the solution passed 200/200 testcases with a runtime of 0 ms and a memory usage of 41.64 MB, both of which are optimal (100.00% and 81.55% respectively). The 'Code' editor shows the Java code for the 'Solution' class, which implements the 'isSymmetric' method by checking if the tree is a mirror of itself. The 'Testcase' section shows the input for a specific test case: 'root = [1,2,2,null,3,null,3]', and the output is 'Accepted' with a runtime of 0 ms.

Accepted 200 / 200 testcases passed
submitted at Mar 03, 2025 09:59

Runtime: 0 ms | Beats 100.00%
Memory: 41.64 MB | Beats 81.55%

Code

```
10 *     this.val = val;
11 *     this.left = left;
12 *     this.right = right;
13 * }
14 * }
15 */
16 class Solution {
17     public boolean isSymmetric(TreeNode root) {
18         return isMirror(root.left, root.right);
19     }
20     private boolean isMirror(TreeNode n1, TreeNode n2){
21         if (n1==null && n2 == null){
22             return true;
23         }
24         if(n1 == null || n2 == null){
25             return false;
26         }
27         return n1.val == n2.val && isMirror(n1.left, n2.right) &&
isMirror(n1.right, n2.left);
28     }
29 }
```

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

root = [1,2,2,null,3,null,3]

104. Maximum Depth of Binary Tree

```
/**
 * 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) {
        return traverse(root);
    }
    int traverse(TreeNode root){
        if(root == null) return 0;
        int r = traverse(root.left);
        int l = traverse(root.right);
        return Math.max(r, l)+1;
    }
}
```

The screenshot displays a coding competition interface. On the left, a sidebar shows the 'Problem List' with a submission status for 'Feb 17, 2025 10:44' at '0 ms'. The main area shows the submission details for a Java solution. The submission is 'Accepted' with '39 / 39 testcases passed'. The user 'Aditya Sharma' submitted it at 'Feb 17, 2025 10:44'. The runtime is '0 ms' and the memory is '42.64 MB'. A bar chart indicates that '87.69% of solutions used 0 ms of runtime'. The test result section shows the input array '[3, 9, 20, null, null, 15, 7]'.

98. [Validate Binary Search Tree](#)

```
/**
 * 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 {
    private long minVal = Long.MIN_VALUE;
    public boolean isValidBST(TreeNode root) {
        if (root == null) return true;
        if (!isValidBST(root.left)) return false;

        if (minVal >= root.val) return false;

        minVal = root.val;

        if (!isValidBST(root.right)) return false;

        return true;
    }
}
```

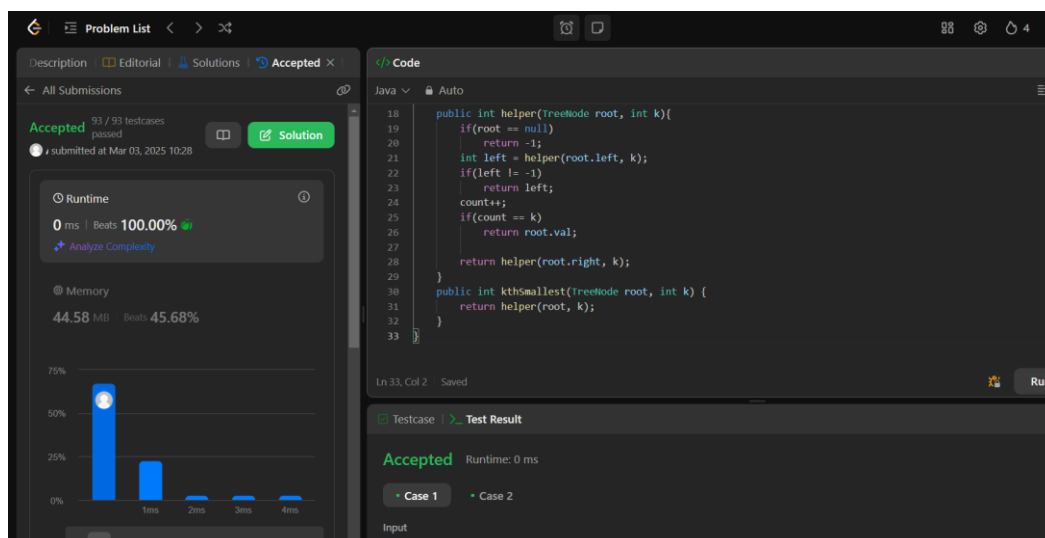
The screenshot displays a coding platform interface with the following components:

- Problem List:** A navigation bar at the top with icons for problem list, editorials, solutions, and accepted submissions.
- Accepted Submissions:** A section showing the submission status as 'Accepted' with 86/86 testcases passed. It includes a 'Solution' button and a timestamp 'submitted at Mar 03, 2025 10:18'.
- Runtime and Memory Performance:**
 - Runtime:** 0 ms, Beats 100.00%.
 - Memory:** 43.23 MB, Beats 79.10%.
- Bar Chart:** A chart showing the distribution of runtime performance across different percentiles, with a peak at 100%.
- Code Editor:** A code editor showing the Java solution for the problem, with line numbers 15 to 30. The code is identical to the one provided in the first block.
- Testcase and Test Result:** A section showing the test result as 'Accepted' with a runtime of 0 ms. It includes buttons for 'Case 1' and 'Case 2'.

230. [Kth Smallest Element in a BST](#)

```
/**
 * 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 {
    int count = 0;
    public int helper(TreeNode root, int k){
        if(root == null)
            return -1;
        int left = helper(root.left, k);
        if(left != -1)
            return left;
        count++;
        if(count == k)
            return root.val;

        return helper(root.right, k);
    }
    public int kthSmallest(TreeNode root, int k) {
        return helper(root, k);
    }
}
```



102. [Binary Tree Level Order Traversal](#)

```
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {

        List<List<Integer>> result=new ArrayList<>();
        if(root==null)
            return result;

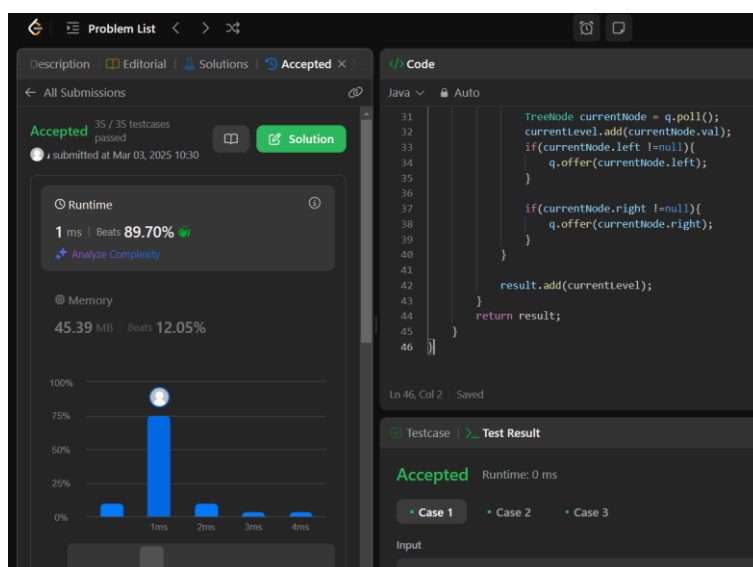
        Queue<TreeNode> q=new LinkedList<>();
        q.offer(root);
        while(!q.isEmpty()){
            int levelSize=q.size();
            List<Integer> currentLevel = new ArrayList<>();

            for(int i=0;i<levelSize;i++){

                TreeNode currentNode = q.poll();
                currentLevel.add(currentNode.val);
                if(currentNode.left !=null){
                    q.offer(currentNode.left);
                }

                if(currentNode.right !=null){
                    q.offer(currentNode.right);
                }
            }

            result.add(currentLevel);
        }
        return result;
    }
}
```



107. Binary Tree Level Order Traversal II

```
/**
 * 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;
 *     }
 * }
 */
public class Solution {
    public List<List<Integer>> levelOrderBottom(TreeNode root) {
        Queue<TreeNode> queue = new LinkedList<TreeNode>();
        List<List<Integer>> wrapList = new LinkedList<List<Integer>>();

        if(root == null) return wrapList;

        queue.offer(root);
        while(!queue.isEmpty()){
            int levelNum = queue.size();
            List<Integer> subList = new LinkedList<Integer>();
            for(int i=0; i<levelNum; i++) {
                if(queue.peek().left != null) queue.offer(queue.peek().left);
                if(queue.peek().right != null) queue.offer(queue.peek().right);
                subList.add(queue.poll().val);
            }
            wrapList.add(0, subList);
        }
        return wrapList;
    }
}
```

The screenshot displays the LeetCode submission interface for the problem 'Binary Tree Level Order Traversal II'. The interface is divided into several sections:

- Problem List:** Located at the top, it shows the problem name and a 'Solution' button.
- Runtime:** A section showing the execution time of the solution. It indicates '1 ms' and 'Beats 94.02%'. A bar chart below this section shows the performance of various submissions, with the current solution being the fastest.
- Memory:** A section showing the memory usage of the solution. It indicates '43.38 MB' and 'Beats 11.98%'. A bar chart below this section shows the memory usage of various submissions, with the current solution being the most memory-efficient.
- Code:** A section showing the source code of the solution. The code is written in Java and implements a breadth-first search (BFS) algorithm to traverse the binary tree level by level, starting from the root and moving downwards. The code is as follows:

```
21 if(root == null) return wrapList;
22
23 queue.offer(root);
24 while(!queue.isEmpty()){
25     int levelNum = queue.size();
26     List<Integer> subList = new LinkedList<Integer>();
27     for(int i=0; i<levelNum; i++) {
28         if(queue.peek().left != null) queue.offer(queue.peek().left);
29         if(queue.peek().right != null) queue.offer(queue.peek().right);
30         subList.add(queue.poll().val);
31     }
32     wrapList.add(0, subList);
33 }
34 return wrapList;
35 }
36 }
```
- Testcase:** A section showing the results of the solution's execution on various test cases. It indicates 'Accepted' and 'Runtime: 0 ms'. Below this section, there are three test cases: 'Case 1', 'Case 2', and 'Case 3', all of which are marked as 'Accepted'.

103. Binary Tree Zigzag Level Order Traversal

```
class Solution {  
  
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {  
        if(root == null) return new ArrayList<>();  
        ArrayDeque<TreeNode> dq = new ArrayDeque<>();  
        dq.offer(root);  
        List<List<Integer>> result = new ArrayList<>();  
        boolean leftToRight = true;  
  
        while(!dq.isEmpty()){  
            List<Integer> currLevel = new ArrayList<>();  
            for(int i = dq.size(); i > 0; i--){  
                TreeNode curr = (leftToRight)?dq.pollFirst():dq.pollLast();  
                currLevel.add(curr.val);  
                if(leftToRight){  
                    if(curr.left != null)  
                        dq.offerLast(curr.left);  
                    if(curr.right != null)  
                        dq.offerLast(curr.right);  
                }  
                else{  
                    if(curr.right != null)  
                        dq.offerFirst(curr.right);  
                    if(curr.left != null)  
                        dq.offerFirst(curr.left);  
                }  
            }  
  
            leftToRight = !leftToRight;  
            result.add(currLevel);  
        }  
        return result;  
    }  
}
```

Problem List < >

Description Editorial Solutions Accepted

All Submissions

Accepted 33 / 33 testcases passed
submitted at Mar 03, 2025 10:34

Runtime 1 ms | Beats 69.88%
[Analyze Complexity](#)

Memory 42.50 MB | Beats 23.80%

75%
50%
25%
0%

1ms 2ms 3ms 4ms

Code

```
Java Auto  
33 dq.offerLast(curr.right);  
34 }  
35 else{  
36     if(curr.right != null)  
37         dq.offerFirst(curr.right);  
38     if(curr.left != null)  
39         dq.offerFirst(curr.left);  
40 }  
41 }  
42  
43 leftToRight = !leftToRight;  
44 result.add(currLevel);  
45 }  
46 return result;  
47 }  
48 }
```

Ln 43, Col 32 Saved

Testcase Test Result

Case 1 Case 2 Case 3 +

root =
[3,9,20,null,null,15,7]

</> Source

199. Binary Tree Right Side View

```
import java.util.*;
// takesoumen collection
class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        if (root == null) return result;

        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);

        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            for (int i = 0; i < levelSize; i++) {
                TreeNode node = queue.poll();
                if (i == levelSize - 1) result.add(node.val);
                if (node.left != null) queue.add(node.left);
                if (node.right != null) queue.add(node.right);
            }
        }

        return result;
    }
}
```


Problem List < > ↺


on | Editorial | Solutions | Accepted × | Submit

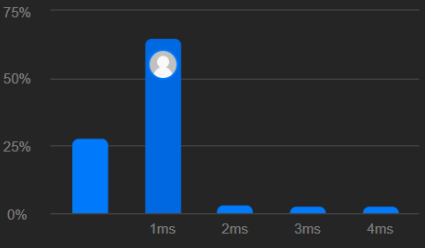
← All Submissions

Accepted 217 / 217 testcases passed **Solution**

submitted at Mar 03, 2025 10:36

Runtime 1 ms | Beats 72.03%  [Analyze Complexity](#)

Memory 42.10 MB | Beats 78.87% 



75%
50%
25%
0%

1ms 2ms 3ms 4ms

1ms 2ms 3ms 4ms

Code Java Auto

```
8 Queue<TreeNode> queue = new LinkedList<>();
9 queue.add(root);
10
11 while (!queue.isEmpty()) {
12     int levelSize = queue.size();
13     for (int i = 0; i < levelSize; i++) {
14         TreeNode node = queue.poll();
15         if (i == levelSize - 1) result.add(node.val);
16         if (node.left != null) queue.add(node.left);
17         if (node.right != null) queue.add(node.right);
18     }
19 }
20
21 return result;
22 }
23
```

Ln 23, Col 2 | Saved

Testcase | **Test Result**

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3 • Case 4

Input

root =

[1,2,3,null,5,null,4]

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

```
class Solution {

    public TreeNode buildTree(int[] inorder, int[] postorder) {
        // Call the recursive function with full arrays and return the result
        return buildTree(inorder, 0, inorder.length - 1, postorder, 0,
postorder.length - 1);
    }

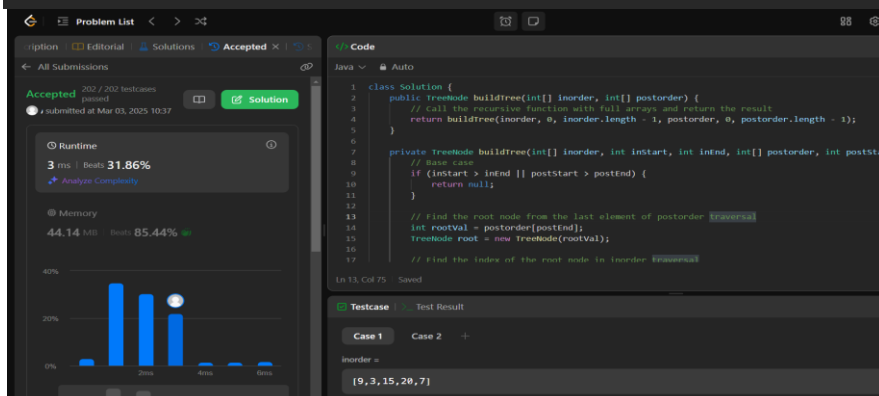
    private TreeNode buildTree(int[] inorder, int inStart, int inEnd, int[]
postorder, int postStart, int postEnd) {
        // Base case
        if (inStart > inEnd || postStart > postEnd) {
            return null;
        }

        // Find the root node from the last element of postorder traversal
        int rootVal = postorder[postEnd];
        TreeNode root = new TreeNode(rootVal);

        // Find the index of the root node in inorder traversal
        int rootIndex = 0;
        for (int i = inStart; i <= inEnd; i++) {
            if (inorder[i] == rootVal) {
                rootIndex = i;
                break;
            }
        }

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

        return root;
    }
}
```



513. Find Bottom Left Tree Value

```
public class Solution {
    public int findBottomLeftValue(TreeNode root) {
        int last = 0;
        Queue<TreeNode> q = new LinkedList<>();
        q.add(root);

        while (!q.isEmpty()) {
            int count = q.size();
            for (int i = 0; i < count; i++) {
                TreeNode curr = q.poll();
                if (i == 0)
                    last = curr.val; // last leftMost val
                if (curr.left != null)
                    q.add(curr.left);
                if (curr.right != null)
                    q.add(curr.right);
            }
        }
        return last;
    }
}
```

The screenshot displays the LeetCode submission page for the problem '513. Find Bottom Left Tree Value'. The interface is divided into several sections:

- Problem List:** Navigation icons for problem list, editorial, solutions, and accepted submissions.
- Description:** Tabs for Description, Editorial, Solutions, and Accepted (selected).
- All Submissions:** A section showing submission status. It indicates 'Accepted' with '79 / 79 testcases passed' and a 'Solution' button. The submission was made on 'Mar 03, 2025 10:39'.
- Runtime:** A box showing '3 ms' and 'Beats 56.19%'. A link to 'Analyze Complexity' is provided.
- Memory:** A box showing '44.64 MB' and 'Beats 57.96%'. Below this is a bar chart showing performance across different time intervals (1ms, 2ms, 3ms, 4ms).
- Code:** A code editor showing the Java solution. The code is as follows:

```
6
7
8     while (!q.isEmpty()) {
9         int count = q.size();
10        for (int i = 0; i < count; i++) {
11            TreeNode curr = q.poll();
12            if (i == 0)
13                last = curr.val; // last leftMost val
14            if (curr.left != null)
15                q.add(curr.left);
16            if (curr.right != null)
17                q.add(curr.right);
18        }
19    }
20    return last;
21 }
```
- Testcase / Test Result:** A section showing 'Accepted' with 'Runtime: 0 ms'. It includes tabs for 'Case 1' and 'Case 2', and an 'Input' field.

The Windows taskbar at the bottom shows the system clock as 17°C Cloudy and various application icons.

124. [Binary Tree Maximum Path Sum](#)

```
class TreeNode {
    int val;
    TreeNode left, right;
    TreeNode(int val) { this.val = val; }
}

class Solution {
    private int ans = Integer.MIN_VALUE;

    public int maxPathSum(TreeNode root) {
        helper(root);
        return ans;
    }

    private int helper(TreeNode root) {
        if (root == null) return 0;

        int left = Math.max(0, helper(root.left));
        int right = Math.max(0, helper(root.right));

        ans = Math.max(ans, root.val + left + right);

        return root.val + Math.max(left, right);
    }
}
```

Problem List < >

Description | **Editorial** | Solutions | Accepted ×

← All Submissions

Accepted 96 / 96 testcases passed **Solution**

submitted at Mar 03, 2025 10:40

Runtime
0 ms | Beats 100.00%
 Analyze Complexity

Memory
44.42 MB | Beats 49.33%

100%
50%
0%
1ms 2ms 3ms 4ms

Code

Java Auto

```
13 }
14
15 private int helper(TreeNode root) {
16     if (root == null) return 0;
17
18     int left = Math.max(0, helper(root.left));
19     int right = Math.max(0, helper(root.right));
20
21     ans = Math.max(ans, root.val + left + right);
22
23     return root.val + Math.max(left, right);
24 }
25
```

Ln 25, Col 2 | Saved

Testcase | **Test Result**

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

root =

[1, 2, 3]

987. [Vertical Order Traversal of a Binary Tree](#)

```
/**
 * 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 {
    Map<Integer, ArrayList<int[]>> map = new TreeMap<>();
    public List<List<Integer>> verticalTraversal(TreeNode root) {
        dfs(root, 0, 0);
        List<List<Integer>> result = new ArrayList<>();
        for(ArrayList<int[]> list: map.values()) {
            Collections.sort(list, (a, b) -> a[0] == b[0] ? Integer.compare(a[1],
b[1]) : Integer.compare(a[0], b[0]));
            ArrayList<Integer> current = new ArrayList<>();
            for(int[] num : list) {
                current.add(num[1]);
            }
            result.add(current);
        }
        return result;
    }

    void dfs(TreeNode root, int index, int dept) {
        if(root == null) {
            return;
        }
        map.putIfAbsent(index, new ArrayList<>());
        map.get(index).add(new int[]{dept, root.val});
        dfs(root.left, index - 1, dept + 1);
        dfs(root.right, index + 1, dept + 1);
    }
}
```

Problem List

Description | Editor | Solutions | Accepted

All Submissions

Accepted

34 / 34 testcases passed

submitted at Mar 03, 2025 10:41

Solution

Runtime

3 ms | Beats 83.09%

Analyze Complexity

Memory

42.50 MB | Beats 80.55%

Time (ms)	Percentage (%)
1ms	2
2ms	15
3ms	45
4ms	30
5ms	5

Code

Java | Auto

```
28     }
29     return result;
30 }
31
32
33
34 void dfs(TreeNode root, int index, int dept) {
35     if(root == null) {
36         return;
37     }
38     map.putIfAbsent(index, new ArrayList<>());
39     map.get(index).add(new int[]{dept, root.val});
40     dfs(root.left, index - 1, dept + 1);
41     dfs(root.right, index + 1, dept + 1);
42 }
43 }
```

Ln 43, Col 2 | Saved

Testcase | Test Result

Case 1 | Case 2 | Case 3 | +

root =

[3,9,20,null,null,15,7]

</> Source