



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## ASSIGNMENT -1 (ADVANCED PROGRAMMING)

### 1. Problem 1: Binary Tree In Order Traversal

### 2. Implementation/Code:

```
class Solution {  
  
    public List<Integer> inorder(TreeNode root, List<Integer> list){  
        if(root == null) return list;  
        inorder(root.left,list);  
        list.add(root.val);  
        inorder(root.right,list);  
        return list;  
    }  
    public List<Integer> inorderTraversal(TreeNode root) {  
        List<Integer> list = new ArrayList<Integer>();  
        return inorder(root,list);  
    }  
}
```

### 3. Output:

The screenshot shows the LeetCode platform interface for problem 94. Binary Tree Inorder Traversal. The problem description is as follows:

**94. Binary Tree Inorder Traversal**  
Given the `root` of a binary tree, return the *inorder traversal* of its nodes' values.

**Example 1:**  
Input: `root = [1,null,2,3]`  
Output: `[1,3,2]`  
Explanation:

**Example 2:**

The code submitted is:

```
Accepted 71 / 71 testcases passed  
Shourya Gupta submitted at Jan 30, 2025 13:11
```

Performance metrics:

- Runtime: 0 ms | Beats 100.00%
- Memory: 10.81 MB | Beats 65.91%

Test Result:

Case 1	Case 2	Case 3	Case 4
<code>root = [1,null,2,3]</code>			



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## 1. Problem 2: Symmetric Tree

### 2. Implementation/Code:

```
class Solution {  
    public boolean isSymmetric(TreeNode root) {  
        if (root == null) {  
            return true;  
        }  
        return isMirror(root.left, root.right);  
    }  
  
    private boolean isMirror(TreeNode node1, TreeNode node2) {  
        if (node1 == null && node2 == null) {  
            return true;  
        }  
        if (node1 == null || node2 == null) {  
            return false;  
        }  
        if (node1.val != node2.val) {  
            return false;  
        }  
        return isMirror(node1.left, node2.right) && isMirror(node1.right,  
node2.left);  
    }  
}
```

### 3. Output:

The screenshot shows a code submission interface for a C++ problem. The top navigation bar includes links for Description, Editorial, Solutions, Accepted (highlighted), Submissions, and a user profile. The main area displays the following information:

- Accepted:** 199 / 199 testcases passed by Shourya Gupta at Feb 14, 2025 12:40.
- Runtime:** 0 ms (Beats 100.00%)
- Memory:** 18.50 MB (Beats 58.65%)
- Complexity Analysis:** 0.48% of solutions used 4 ms of runtime.
- Code:** The source code for the Solution class, identical to the one provided above.
- Test Result:** Accepted, Runtime: 0 ms, with Case 1 and Case 2 both successful.



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## Problem 3: Maximum Depth of Binary Tree

### 1. Implementation/code:

```
public class Solution {

    public int maxDepth(TreeNode root) {
        return height(root);
    }

    private int height(TreeNode node) {
        if (node == null) return 0;
        int leftHeight = height(node.left);
        int rightHeight = height(node.right);
        return 1 + Math.max(leftHeight, rightHeight);
    }
}
```

### 2. Output:

The screenshot shows a code editor interface with a dark theme. On the left, there's a sidebar with navigation links like 'Description', 'Editorial', 'Solutions', 'Accepted' (which is highlighted), and 'Submissions'. Below these are sections for 'Runtime' (0 ms, beats 100.00%) and 'Memory' (19.23 MB, beats 13.08%). A chart below these metrics shows execution time distribution from 0% to 150%. On the right, the main area displays the C++ code for the solution. The code uses a queue to perform a level-order traversal of the binary tree to calculate its depth. At the bottom, there's a 'Testcase' section with two cases and a 'Source' button.

```
int levelSize = q.size();

for (int i = 0; i < levelSize; i++) {
    TreeNode* node = q.front();
    q.pop();

    if (node->left) {
        q.push(node->left);
    }
    if (node->right) {
        q.push(node->right);
    }
}

return depth;
}



```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr), right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr), right(nullptr) {}
 */

```


```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## 1. Problem 4: Validate Binary search Tree

## 2. Implementation/code:

```
class Solution {  
    private long minValue = Long.MIN_VALUE;  
    public boolean isValidBST(TreeNode root) {  
        if (root == null) return true;  
        if (!isValidBST(root.left)) return false;  
        if (minValue >= root.val) return false;  
        minValue = root.val;  
        if (!isValidBST(root.right)) return false;  
        return true;    } }
```

## 3. Output:

The screenshot shows two side-by-side sections of a LeetCode problem page for '98. Validate Binary Search Tree'.  
**Left Side (Problem Statement):**  
The title is '98. Validate Binary Search Tree'. It is marked as 'Solved' with a green checkmark. Below the title, it says 'Medium' and lists 'Topics' and 'Companies'. A description follows: 'Given the `root` of a binary tree, determine if it is a valid binary search tree (BST).'  
A note defines a 'Valid BST': 'A valid BST is defined as follows:

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

'  
An 'Example 1:' shows a binary tree with root 2, left child 1, and right child 3.  
Input: `root = [2,1,3]`  
Output: `true`  
An 'Example 2:' is partially visible.  
**Right Side (Solution Statistics):**  
The title is 'Code | Accepted'. It shows 'Accepted' with 86/86 testcases passed. The author is 'Shourya Gupta' submitted at 'Feb 14, 2025 12:46'. It includes 'Editorial' and 'Solution' buttons.  
Performance metrics:

- Runtime:** 0 ms | Beats 100.00% (green bar)
- Memory:** 22.01 MB | Beats 19.45%

A chart shows memory usage over time, with a single bar reaching 100% at 0ms.  
Test results:

- Testcase:** Accepted Runtime: 0 ms
- Case 1** (green dot)
- Case 2** (green dot)

Input field: `root =`  
Footer stats: 17.4K views, 220 likes, 166 Online users.

## 1. Problem 5: Kth Smallest Element in a BST

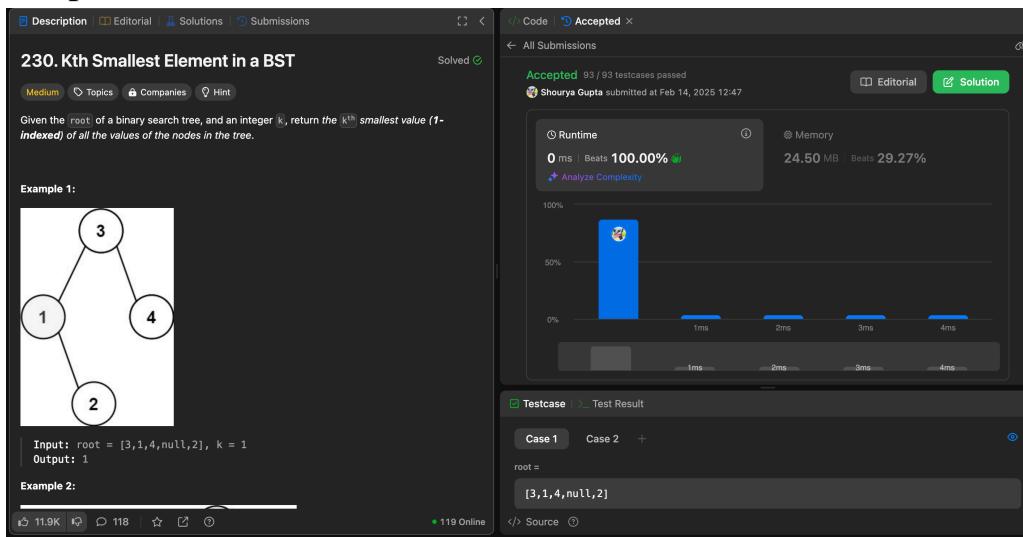
## 2. Implementation/Code:

```
class Solution {
    public int kthSmallest(TreeNode root, int k) {
        int count = countNodes(root.left);
        if (k <= count) {
            return kthSmallest(root.left, k);
        } else if (k > count + 1) {
            return kthSmallest(root.right, k-1-count);
        }
        return root.val;
    }

    public int countNodes(TreeNode n) {
        if (n == null) return 0;

        return 1 + countNodes(n.left) + countNodes(n.right);
    }
}
```

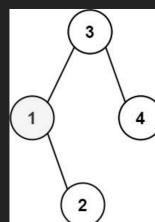
## 3. Output:



230. Kth Smallest Element in a BST

Given the root of a binary search tree, and an integer k, return the  $k^{th}$  smallest value ( $1\text{-indexed}$ ) of all the values of the nodes in the tree.

**Example 1:**



**Input:** root = [3,1,4,null,2], k = 1  
**Output:** 1

**Example 2:**

Runtime: 0 ms | Beats: 100.00%  
Memory: 24.50 MB | Beats: 29.27%

## 1. Problem 6: Binary Tree Level Order Traversal

## 2. Implementation/Code:

```
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        List<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();
            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(level);
        }
        return result;
    }
}
```

## 3. Output:

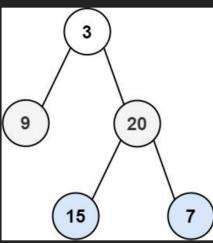
[Description](#) [Editorial](#) [Solutions](#) [Submissions](#)

**102. Binary Tree Level Order Traversal** Solved

[Medium](#) [Topics](#) [Companies](#) [Hint](#)

Given the `root` of a binary tree, return the *level order traversal* of its nodes' values. (i.e., from left to right, level by level).

**Example 1:**



```

graph TD
    3((3)) --> 9((9))
    3((3)) --> 20((20))
    9((9)) --> 15((15))
    9((9)) --> 7((7))
  
```

**Input:** `root = [3,9,20,null,null,15,7]`  
**Output:** `[[3],[9,20],[15,7]]`

**Example 2:**

[Code](#) Accepted

All Submissions

**Accepted** 35 / 35 testcases passed

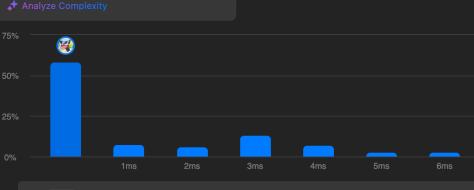
Shourya Gupta submitted at Jan 28, 2025 18:23

[Editorial](#) [Solution](#)

**Runtime** 0 ms | Beats 100.00% 

**Memory** 17.14 MB | Beats 43.81%

Analyze Complexity



**Testcase** [Test Result](#)

Case 1	Case 2	Case 3	+
root =	[3,9,20,null,null,15,7]		

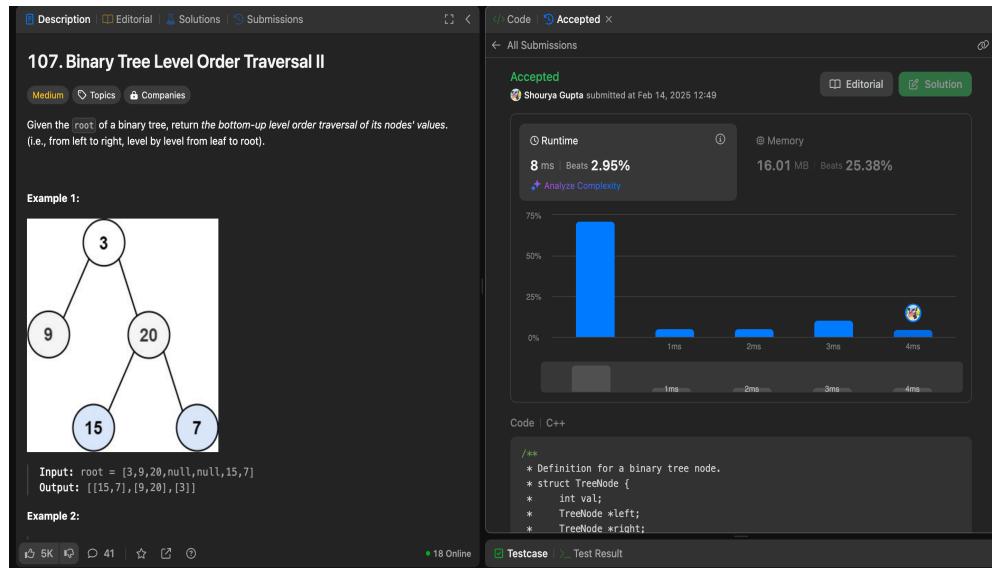
15.9K  116    

183 Online

## 1. Problem 7: Binary Tree Level Order Traversal II

### 2. Implementation/code:

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



The screenshot shows the LeetCode platform interface for problem 107. The problem title is "107. Binary Tree Level Order Traversal II". It describes the task: given the root of a binary tree, return the bottom-up level order traversal of its nodes' values (i.e., from left to right, level by level from leaf to root).

**Example 1:** A binary tree with root 3. Node 3 has children 9 and 20. Node 20 has children 15 and 7.

**Example 2:** Input: root = [3,9,20,null,null,15,7] Output: [[15,7],[9,20],[3]]

**Submission Details:** Shourya Gupta submitted at Feb 14, 2025 12:49. Status: Accepted. Runtime: 8 ms (Beats 2.95%). Memory: 16.01 MB (Beats 25.38%).

**Code (C++):**

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 * };
 */

```

**Testcase:** 5K | 41 | 18 Online

### 3. Output:

## 1. Problem 8: Binary Tree Zig Zag Level Order Traversal

### 2. Implementation/code:

```
class Solution {
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
        List<List<Integer>> res = new ArrayList<>();
        if (root == null) return res;
        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(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);
                if (node.left != null) queue.add(node.left);
                if (node.right != null) queue.add(node.right); }
            res.add(level);
            leftToRight = !leftToRight; }
        return res; }}
```

### 3. Output:

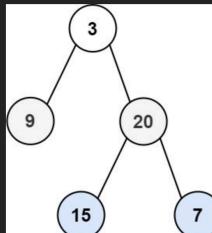
[Description](#) [Editorial](#) [Solutions](#) [Submissions](#)

**103. Binary Tree Zigzag Level Order Traversal** Solved 2

[Medium](#) [Topics](#) [Companies](#)

Given the `root` of a binary tree, return the zigzag level order traversal of its nodes' values. (i.e., from left to right, then right to left for the next level and alternate between).

**Example 1:**



**Input:** root = [3,9,20,null,null,15,7]  
**Output:** [[3],[20,9],[15,7]]

**Example 2:**

[Code](#) [Accepted](#)

All Submissions

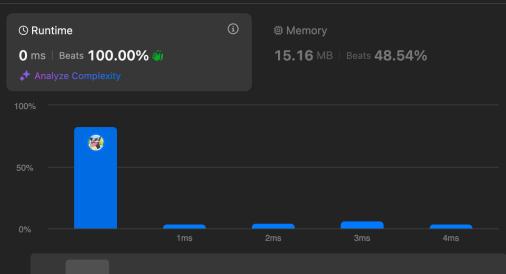
**Accepted** 33 / 33 testcases passed

Shourya Gupta submitted at Feb 14, 2025 12:52

[Editorial](#) [Solution](#)

Runtime
Memory

0 ms   Beats 100.00%	15.16 MB   Beats 48.54%
<a href="#">Analyze Complexity</a>	



**Code | C++**

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 */
```



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## 1. Problem 9: Binary Tree Right Side View

### 2. Implementation/code:

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

### 3. Output:

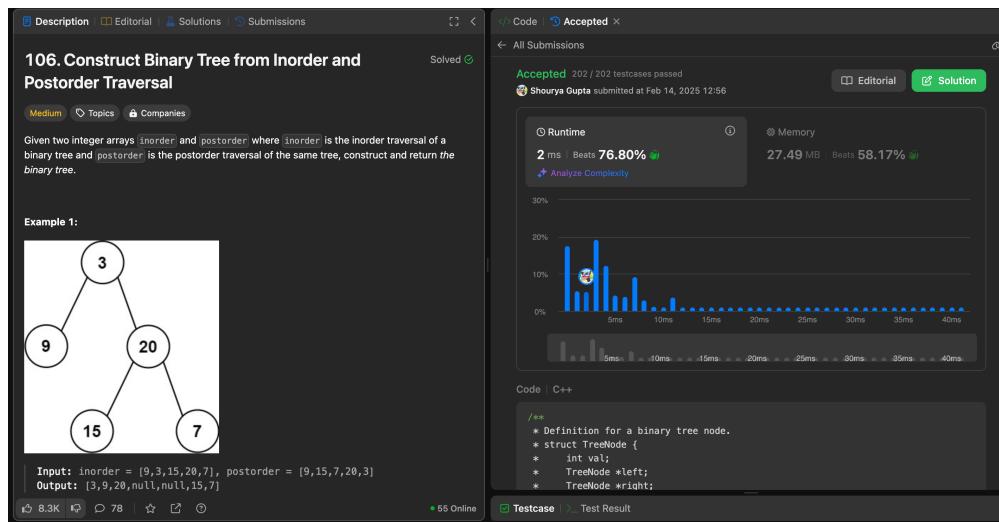
The screenshot shows a programming challenge titled "199. Binary Tree Right Side View". The challenge description states: "Given the root of a binary tree, imagine yourself standing on the right side of it, return the values of the nodes you can see ordered from top to bottom." Example 1 shows input [1,2,3,null,5,null,4] and output [1,3,4]. Example 2 shows input [1,2,3,4,null,null,null,5] and output [1,3,4,5]. The code is written in C++ and is accepted with 100% runtime and memory efficiency. The runtime distribution chart shows a single bar at 0ms. The code includes a diagram of a binary tree with root 1, left child 2, right child 3, left child of 2 is 5, and left child of 3 is 4.

## 1. Problem 10: Construct Binary Tree From Inorder and Postorder Traversal

### 2. Code:

```
import java.util.*;
class Solution {
    int postIndex;
    Map<Integer, Integer> inMap;
    public TreeNode buildTree(int[] inorder, int[] postorder) {
        inMap = new HashMap<>();
        postIndex = postorder.length - 1;
        for (int i = 0; i < inorder.length; i++) {
            inMap.put(inorder[i], i);
        }
        return build(postorder, 0, inorder.length - 1);
    }
    private TreeNode build(int[] postorder, int inStart, int inEnd) {
        if (inStart > inEnd) return null;
        int rootVal = postorder[postIndex--];
        TreeNode root = new TreeNode(rootVal);
        int inIndex = inMap.get(rootVal);
        root.right = build(postorder, inIndex + 1, inEnd);
        root.left = build(postorder, inStart, inIndex - 1);
        return root;
    }
}
```

### 3. Output:



The screenshot shows the LeetCode platform interface. The problem title is "106. Construct Binary Tree from Inorder and Postorder Traversal". The problem statement asks to construct a binary tree from given inorder and postorder traversals. Example 1 shows a tree with root 3, left child 9, right child 20, left child of 20 is 15, and right child of 20 is 7. The accepted submission details show a runtime of 2 ms (76.80% beats) and 27.49 MB memory usage (58.17% beats). The code provided is a C++ implementation of the solution described above.

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 * };
 */
class Solution {
public:
    TreeNode* buildTree(vector<int> &inorder, vector<int> &postorder) {
        if (inorder.size() == 0) return NULL;
        return build(inorder, 0, inorder.size() - 1, postorder, 0, postorder.size() - 1);
    }

    TreeNode* build(vector<int> &inorder, int inStart, int inEnd, vector<int> &postorder, int postIndex) {
        if (inStart > inEnd) return NULL;
        int rootVal = postorder[postIndex];
        int inIndex = find(inorder.begin() + inStart, inorder.begin() + inEnd + 1, rootVal) - inorder.begin();
        TreeNode* root = new TreeNode(rootVal);
        root->left = build(inorder, inStart, inIndex - 1, postorder, postIndex - inIndex);
        root->right = build(inorder, inIndex + 1, inEnd, postorder, postIndex - 1);
        return root;
    }
};
```

## 1. Problem 11: Find Bottom Left Tree Value

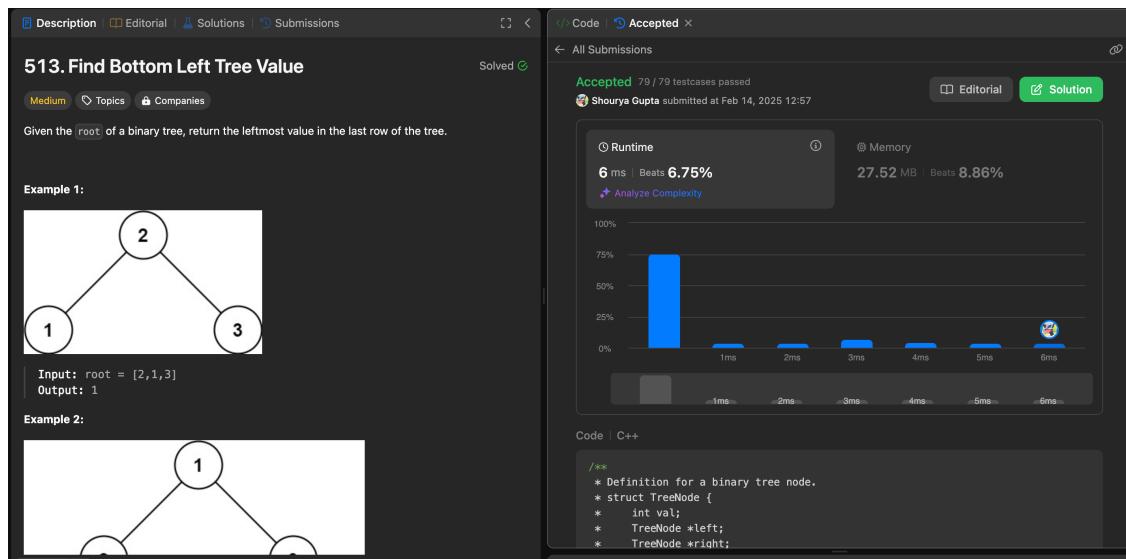
### 2. Code:

```
class Solution {
    public int findBottomLeftValue(TreeNode root) {
        Queue<TreeNode> queue = new LinkedList<>();
        queue.add(root);
        int bottomLeft = root.val;

        while (!queue.isEmpty()) {
            TreeNode node = queue.poll();
            bottomLeft = node.val;

            if (node.right != null) queue.add(node.right);
            if (node.left != null) queue.add(node.left);
        }
        return bottomLeft;
    }
}
```

### 3. Output:



The screenshot shows a LeetCode problem page for "513. Find Bottom Left Tree Value". The problem description asks for the leftmost value in the last row of a binary tree. Example 1 shows a tree with root 2, left child 1, and right child 3. Example 2 shows a tree with root 1, left child null, and right child null. The input is root = [2,1,3] and the output is 1. The submission was made by Shourya Gupta on Feb 14, 2025 at 12:57. It was accepted with 79/79 testcases passed. The runtime is 6 ms (Beats 6.75%) and the memory usage is 27.52 MB (Beats 8.86%). The code is written in C++.

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 * };
 */
class Solution {
public:
    int findBottomLeftValue(TreeNode* root) {
        if (root == NULL) return -1;
        queue.push(root);
        while (!queue.empty()) {
            int size = queue.size();
            for (int i = 0; i < size; i++) {
                TreeNode* curr = queue.front();
                queue.pop();
                if (i == 0) ans = curr->val;
                if (curr->left != NULL) queue.push(curr->left);
                if (curr->right != NULL) queue.push(curr->right);
            }
        }
        return ans;
    }
};
```

## 1. Problem 12: Binary Tree Maximum Path Sum

### 2. Code:

```
class Solution {
    int maxSum = Integer.MIN_VALUE;
    public int maxPathSum(TreeNode root) {
        dfs(root);
        return maxSum;
    }
    private int dfs(TreeNode node) {
        if (node == null) return 0;
        int left = Math.max(0, dfs(node.left));
        int right = Math.max(0, dfs(node.right));
        maxSum = Math.max(maxSum, left + right + node.val);
        return node.val + Math.max(left, right); }}
```

### 3. Output:

Description | Editorial | Solutions | Submissions

**124. Binary Tree Maximum Path Sum**

Solved 

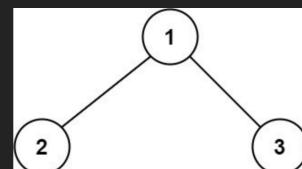
Hard | Topics | Companies

A path in a binary tree is a sequence of nodes where each pair of adjacent nodes in the sequence has an edge connecting them. A node can only appear in the sequence at most once. Note that the path does not need to pass through the root.

The path sum of a path is the sum of the node's values in the path.

Given the `root` of a binary tree, return the maximum path sum of any non-empty path.

**Example 1:**



**Input:** root = [1,2,3]  
**Output:** 6  
**Explanation:** The optimal path is 2 → 1 → 3 with a path sum of 2 + 1 + 3 = 6.

**Example 2:**

Code | Accepted

All Submissions

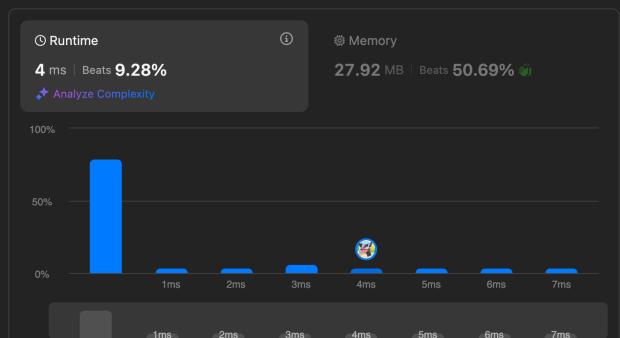
Accepted 96 / 96 testcases passed

Shourya Gupta submitted at Feb 14, 2025 13:00

Editorial | Solution

**Runtime** 4 ms Beats 9.28%  
**Memory** 27.92 MB Beats 50.69%

Analyze Complexity



Code | C++

```
/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 */
```

Testcase | Test Result



# DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

Discover. Learn. Empower.

## 1. Problem 13: Vertical Order Traversal of Binary Tree

### 2. Code:

```
class Solution {  
  
    Map<Integer, TreeMap<Integer, PriorityQueue<Integer>>> map;  
  
    public List<List<Integer>> verticalTraversal(TreeNode root) {  
        if (root == null)  
            return null;  
        map = new TreeMap<>();  
        dfs(root, 0, 0);  
        List<List<Integer>> res = new LinkedList<>();  
        for (int key : map.keySet()) {  
            List<Integer> list = new LinkedList<>();  
            TreeMap<Integer, PriorityQueue<Integer>> tm = map.get(key);  
            for (int k : tm.keySet()) {  
                PriorityQueue<Integer> pq = tm.get(k);  
                while (!pq.isEmpty()) {  
                    list.add(pq.poll());  
                }  
            }  
            res.add(list);  
        }  
        return res;  
    }  
  
    private void dfs(TreeNode root, int index, int level){  
        if (root == null)  
            return;  
    }
```

```

        map.putIfAbsent(index, new TreeMap<>());
        map.get(index).putIfAbsent(level, new PriorityQueue<>());
        map.get(index).get(level).add(root.val);
        dfs(root.left, index - 1, level + 1);
        dfs(root.right, index + 1, level + 1);
    }
}

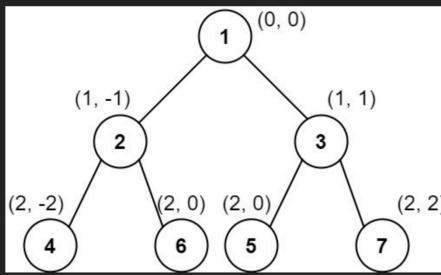
```

### 3. Output:

Description | Editorial | Solutions | Submissions

5 and 6 are at the same position (2, 0), so we order them by their value, 5 before 6.  
 Column 1: Only node 3 is in this column.  
 Column 2: Only node 7 is in this column.

**Example 3:**



**Input:** root = [1,2,3,4,6,5,7]  
**Output:** [[4],[2],[1,5,6],[3],[7]]  
**Explanation:**  
 This case is the exact same as example 2, but with nodes 5 and 6 swapped. Note that the solution remains the same since 5 and 6 are in the same location and should be ordered by their values.

**Constraints:**

8K | 146 | 116 Online

Code | Accepted

All Submissions

Accepted 34 / 34 testcases passed  
 Shourya Gupta submitted at Feb 14, 2025 13:01

Editorial | Solution

Runtime

0 ms | Beats 100.00%

Analyze Complexity

Memory

15.55 MB | Beats 87.19%

Runtime Performance Chart (ms):

Time Range	Percentage
1ms	~5%
2ms	~5%
3ms	~15%
4ms	~5%
5ms	~5%
6ms	~5%

Code | C++

```

/*
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 */

```

Testcase | Test Result