

## Binary Tree Inorder Traversal

```
class Solution {
    public List<Integer> inorderTraversal(TreeNode root) {
        List<Integer> result = new ArrayList<>();
        helper(root,result);
        return result;
    }

    public static void helper(TreeNode root, List<Integer> result){
        if (root == null) return;
        helper(root.left,result);
        result.add(root.val);
        helper(root.right,result);
    }
}
```

The screenshot displays the LeetCode interface for the 'Binary Tree Inorder Traversal' problem. The top section shows the problem description and the user's submission status: 'Accepted' with 71/71 test cases passed. The submission was made by Shiv Chauhan on Feb 14, 2025, at 14:49. The performance metrics are: Runtime 0 ms (Beats 100.00%) and Memory 41.84 MB (Beats 27.14%). A bar chart shows the runtime performance across different test cases, with the first case being the most time-consuming.

The code editor shows the following Java solution:

```
1 class Solution {
2     public List<Integer> inorderTraversal(TreeNode root) {
3         List<Integer> result = new ArrayList<>();
4         helper(root,result);
5         return result;
6     }
7
8     public static void helper(TreeNode root, List<Integer> result){
9         if (root == null) return;
10        helper(root.left,result);
11        result.add(root.val);
12        helper(root.right,result);
13    }
14 }
```

The test results section shows a single test case (Case 1) with the input root = [1,null,2,3]. The output is a list of integers: 1, 2, 3. A diagram of the binary tree is shown, with root 1, left child 2, and right child 3.

## Symmetric Tree

```
class Solution {
    public boolean isSymmetric(TreeNode root) {
        if (root == null) return true;
        return isMirror(root.left, root.right);
    }

    private boolean isMirror(TreeNode t1, TreeNode t2) {
        if (t1 == null && t2 == null) return true;
        if (t1 == null || t2 == null) return false;
        return (t1.val == t2.val) && isMirror(t1.left, t2.right) &&
isMirror(t1.right, t2.left);
    }
}
```

The screenshot displays the LeetCode submission interface for the 'Symmetric Tree' problem. The top navigation bar includes 'Personal', 'Problem List', 'Run', 'Submit', and 'Premium' status. The left sidebar shows 'Description', 'Editorial', 'Solutions', 'Submissions', and 'Accepted' tabs. The main content area is divided into three sections:

- Accepted:** Shows '199 / 199 testcases passed' and 'Shiv Chauhan submitted at Feb 14, 2025 14:50'. It includes a performance graph with 'Runtime: 0 ms' and 'Memory: 42.05 MB'.
- Code:** Displays the submitted Java code for the 'isSymmetric' method.
- Testcase:** Shows 'Case 1' with the input array '[1,2,2,3,4,4,3]' and a corresponding tree diagram. The tree has a root node '1', which has two children '2' and '2'. The left '2' has children '3' and '4', and the right '2' has children '4' and '3'.

At the bottom, there are 'More challenges' listed: '1123. Lowest Common Ancestor of Deepest Leaves', '3283. Maximum Number of Moves to Kill All Pawns', and '2005. Subtree Removal Game with Fibonacci Tree'.

## Maximum Depth of Binary Tree

```
class Solution {  
    public int maxDepth(TreeNode root) {  
        if (root == null){  
            return 0;  
        }  
        int left = maxDepth(root.left);  
        int right = maxDepth(root.right);  
        return Math.max(left, right) + 1;  
    }  
}
```

The screenshot shows a web browser window with the LeetCode website. The main content area displays the submission results for the problem "Maximum Depth of Binary Tree". The submission is marked as "Accepted" and was submitted by "Shiv Chauhan" on Feb 14, 2025, at 14:51. The runtime is 0 ms, which is 100.00% faster than other submissions, and the memory usage is 42.67 MB, which is 60.81% less than other submissions. A bar chart shows the runtime distribution, with the majority of submissions falling into the 0-1ms range. The code editor on the right shows the Java code for the solution, which is a recursive function that calculates the maximum depth of a binary tree. The test case section shows a single test case with the input [3, 9, 20, null, null, 15, 7] and a corresponding binary tree diagram. The tree has a root node 3, which has a left child 9 and a right child 20. Node 20 has a left child 15 and a right child 7. The output of the solution is 4, which is the maximum depth of the tree.

Accepted 39 / 39 testcases passed  
Shiv Chauhan submitted at Feb 14, 2025 14:51

Runtime: 0 ms, Beats 100.00%  
Memory: 42.67 MB, Beats 60.81%

Code (Java)

```
class Solution {  
    public int maxDepth(TreeNode root) {  
        if (root == null){  
            return 0;  
        }  
        int left = maxDepth(root.left);  
        int right = maxDepth(root.right);  
        return Math.max(left, right) + 1;  
    }  
}
```

Testcase

Case 1

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

Diagram of the binary tree:

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

## Validate Binary Search Tree

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

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

**98. Validate Binary Search Tree**

Medium

Given the `root` of a binary tree, determine if it is a valid binary search tree (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.

**Example 1:**

Input: `root = [2,1,3]`  
Output: `true`

**Example 2:**

Input: `root = [5,1,4]`  
Output: `false`

**Testcase** **Test Result**

**Accepted** Runtime: 0 ms

**Case 1** **Case 2**

Input: `root = [2,1,3]`

Output: `true`

Expected: `true`

Contribute a testcase

## 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> queue = new LinkedList<>();
        queue.offer(root);

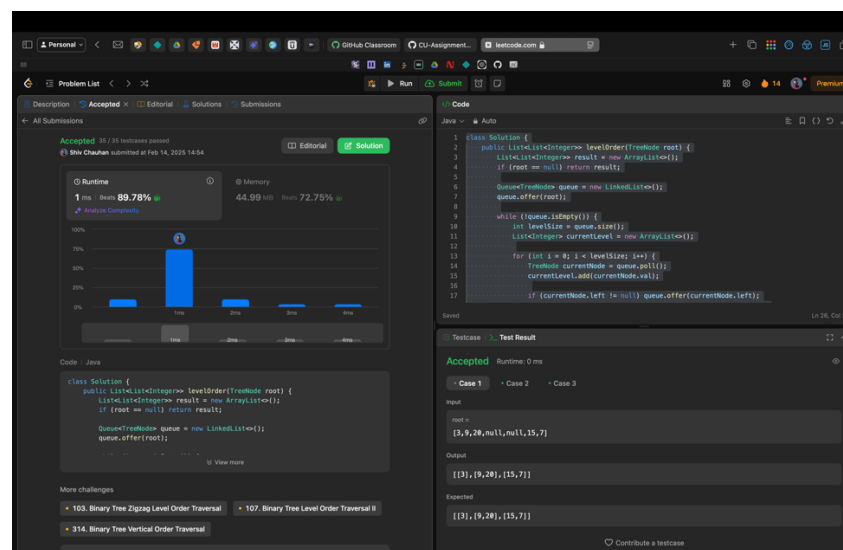
        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            List<Integer> currentLevel = new ArrayList<>();

            for (int i = 0; i < levelSize; i++) {
                TreeNode currentNode = queue.poll();
                currentLevel.add(currentNode.val);

                if (currentNode.left != null) queue.offer(currentNode.left);
                if (currentNode.right != null) queue.offer(currentNode.right);
            }

            result.add(currentLevel);
        }

        return result;
    }
}
```



## Binary Tree Level Order Traversal II

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

The screenshot displays the LeetCode submission interface for the problem "Binary Tree Level Order Traversal II". The left sidebar shows the problem description and a performance graph. The main area shows the code editor with the following Java 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.offer(root);  
  
        while (!queue.isEmpty()) {  
            int levelSize = queue.size();  
            List<Integer> currentLevel = new ArrayList<>();  
  
            for (int i = 0; i < levelSize; i++) {  
                TreeNode currentNode = queue.poll();  
                currentLevel.add(currentNode.val);  
  
                if (currentNode.left != null) queue.offer(currentNode.left);  
                if (currentNode.right != null) queue.offer(currentNode.right);  
            }  
  
            result.add(0, currentLevel);  
        }  
  
        return result;  
    }  
}
```

The test result shows "Accepted" with a runtime of 0 ms. The input is [3,9,20,null,null,15,7] and the output is [[15,7],[9,20],[3]].

## Binary Tree ZigZag Order Traversal

```
class Solution {
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        if (root == null) return result;

        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        boolean leftToRight = true;

        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            LinkedList<Integer> currentLevel = new LinkedList<>();

            for (int i = 0; i < levelSize; i++) {
                TreeNode currentNode = queue.poll();

                if (leftToRight) {
                    currentLevel.addLast(currentNode.val);
                } else {
                    currentLevel.addFirst(currentNode.val);
                }

                if (currentNode.left != null) queue.offer(currentNode.left);
                if (currentNode.right != null) queue.offer(currentNode.right);
            }

            result.add(currentLevel);
            leftToRight = !leftToRight;
        }

        return result;
    }
}
```

The screenshot shows a LeetCode submission interface. On the left, the 'Problem List' tab is active, displaying the submission status 'Accepted' for 33/33 testcases. The user 'Shiv Chauhan' submitted the solution on Feb 14, 2025, at 14:59. Performance metrics show a runtime of 0 ms (beats 100.00%) and memory usage of 42.26 MB (beats 57.42%). A bar chart shows the runtime distribution across different time intervals. The 'Code' tab on the right displays the Java code for the zigzagLevelOrder method. The 'Testcase' tab shows the input [3, 9, 20, null, null, 15, 7] and the output [[3], [20, 9], [15, 7]].

**Accepted** 33 / 33 testcases passed  
Shiv Chauhan submitted at Feb 14, 2025 14:59

**Runtime** 0 ms | Beats 100.00%  
**Memory** 42.26 MB | Beats 57.42%

**Code** Java

```
class Solution {
    public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
        List<List<Integer>> result = new ArrayList<>();
        if (root == null) return result;

        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        boolean leftToRight = true;

        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            LinkedList<Integer> currentLevel = new LinkedList<>();

            for (int i = 0; i < levelSize; i++) {
                TreeNode currentNode = queue.poll();

                if (leftToRight) {
                    currentLevel.addLast(currentNode.val);
                } else {
                    currentLevel.addFirst(currentNode.val);
                }

                if (currentNode.left != null) queue.offer(currentNode.left);
                if (currentNode.right != null) queue.offer(currentNode.right);
            }

            result.add(currentLevel);
            leftToRight = !leftToRight;
        }

        return result;
    }
}
```

**Testcase** **Test Result**

**Accepted** Runtime: 0 ms

**Case 1** • Case 2 • Case 3

**Input**

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

**Output**

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

**Expected**

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

Contribute a testcase

## Binary Tree Right Side View

```
class Solution {
    public List<Integer> rightSideView(TreeNode root) {
        List<Integer> rightSide = new ArrayList<>();
        if (root == null){
            return rightSide;
        }

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

        while(!queue.isEmpty()){
            int level = queue.size();
            for (int i=0;i<level;i++){
                TreeNode current = queue.poll();

                if (i == level-1){
                    rightSide.add(current.val);
                }

                if (current.left != null){
                    queue.add(current.left);
                }

                if (current.right != null){
                    queue.add(current.right);
                }
            }
        }
        return rightSide;
    }
}
```

The screenshot displays the LeetCode submission interface for the 'Binary Tree Right Side View' problem. The submission is accepted, with a runtime of 1 ms and 72.53% of test cases passed. The Java code is shown in the 'Code' tab, and the 'Testcase' tab shows a test case with the input [1,2,3,null,5,null,4] and a corresponding tree diagram.

**Runtime Performance:**

Runtime	Beats
1 ms	72.53%

**Testcase:**

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

Tree Diagram:

```
graph TD
    1((1)) --- 2((2))
    1 --- 3((3))
    2 --- 5((5))
    3 --- 4((4))
```



## Construct Binary Tree from Inorder and Postorder Traversal

```
class Solution {
    private Map<Integer, Integer> inorderIndexMap;
    private int postIndex;

    public TreeNode buildTree(int[] inorder, int[] postorder) {
        inorderIndexMap = new HashMap<>();
        postIndex = postorder.length - 1;

        for (int i = 0; i < inorder.length; i++) {
            inorderIndexMap.put(inorder[i], i);
        }

        return buildTreeHelper(postorder, 0, inorder.length - 1);
    }

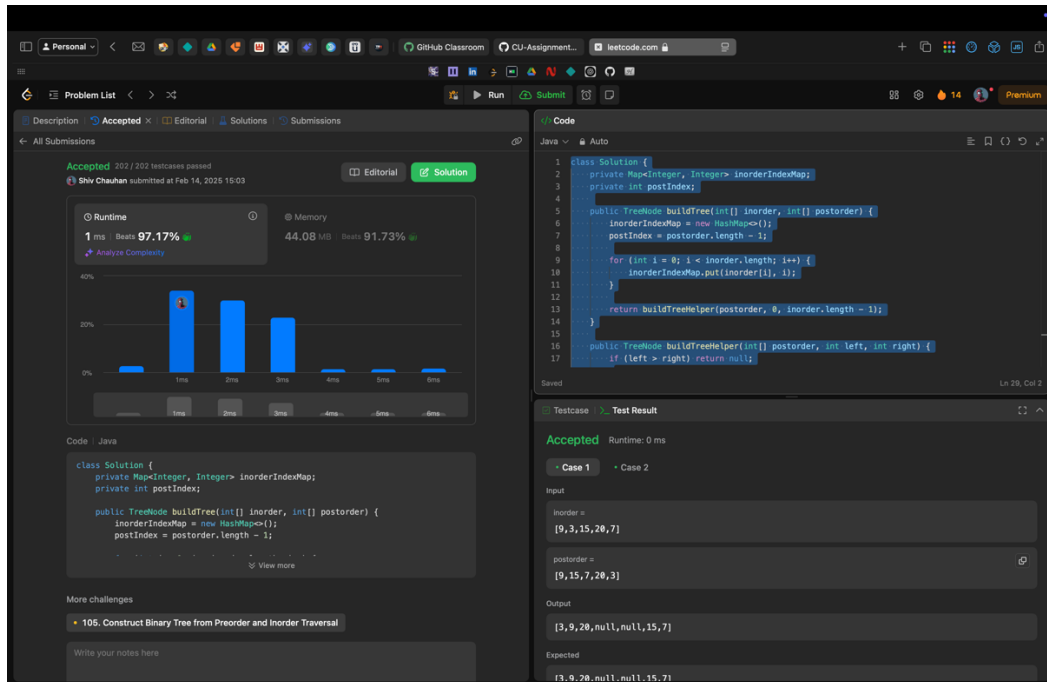
    public TreeNode buildTreeHelper(int[] postorder, int left, int right) {
        if (left > right) return null;

        int rootValue = postorder[postIndex--];
        TreeNode root = new TreeNode(rootValue);

        int inorderIndex = inorderIndexMap.get(rootValue);

        root.right = buildTreeHelper(postorder, inorderIndex + 1, right);
        root.left = buildTreeHelper(postorder, left, inorderIndex - 1);

        return root;
    }
}
```



Find Bottom Left Tree Value

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

        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            leftmostValue = queue.peek().val;

            for (int i = 0; i < levelSize; i++) {
                TreeNode current = queue.poll();
                if (current.left != null) queue.offer(current.left);
                if (current.right != null) queue.offer(current.right);
            }
        }

        return leftmostValue;
    }
}
```

Accepted 79 / 79 testcases passed  
Shiv Chauhan submitted at Feb 14, 2025 15:06

Runtime: 3 ms | Beats 57.22%  
Memory: 44.80 MB | Beats 47.80%

Code | Java

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

        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            leftmostValue = queue.peek().val;

            for (int i = 0; i < levelSize; i++) {
                TreeNode current = queue.poll();
                if (current.left != null) queue.offer(current.left);
                if (current.right != null) queue.offer(current.right);
            }
        }

        return leftmostValue;
    }
}
```

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

root =  
[2,1,3]

Output

1

Expected

1

## Binary Tree Maximum Path Sum

```
class Solution {
    private int maxSum;

    public int maxPathSum(TreeNode root) {
        maxSum = Integer.MIN_VALUE;
        maxGain(root);
        return maxSum;
    }

    private int maxGain(TreeNode node) {
        if (node == null) return 0;

        int leftGain = Math.max(maxGain(node.left), 0);
        int rightGain = Math.max(maxGain(node.right), 0);

        int currentPathSum = node.val + leftGain + rightGain;
        maxSum = Math.max(maxSum, currentPathSum);

        return node.val + Math.max(leftGain, rightGain);
    }
}
```

Personal | Problem List | Accepted | Editorial | Solutions | Submissions

Accepted 96 / 96 testcases passed  
Shiv Chauhan submitted at Feb 14, 2025 15:08

Runtime: 0 ms | Beats 100.00% | Memory: 44.85 MB | Beats 13.06%

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

Code | Java

```
class Solution {
    private int maxSum;

    public int maxPathSum(TreeNode root) {
        maxSum = Integer.MIN_VALUE;
        maxGain(root);
        return maxSum;
    }
}
```

More challenges

- 129. Sum Root to Leaf Numbers
- 666. Path Sum IV
- 687. Longest Univalue Path

Write your notes here

Code

```
1 class Solution {
2     private int maxSum;
3
4     public int maxPathSum(TreeNode root) {
5         maxSum = Integer.MIN_VALUE;
6         maxGain(root);
7         return maxSum;
8     }
9
10    private int maxGain(TreeNode node) {
11        if (node == null) return 0;
12
13        int leftGain = Math.max(maxGain(node.left), 0);
14        int rightGain = Math.max(maxGain(node.right), 0);
15
16        int currentPathSum = node.val + leftGain + rightGain;
17        maxSum = Math.max(maxSum, currentPathSum);
18    }
19 }
```

Testcase | Test Result

Accepted Runtime: 0 ms

Case 1 Case 2

Input

root =  
[1,2,3]

Output

6

Expected

6

Contribute a testcase

## Vertical Order Traversal of Binary Tree

```
class Pair<K, V> {
    private K key;
    private V value;

    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey() { return key; }
    public V getValue() { return value; }
}

class Solution {
    public List<List<Integer>> verticalTraversal(TreeNode root) {
        TreeMap<Integer, TreeMap<Integer, PriorityQueue<Integer>>> map = new
        TreeMap<>();
        Queue<Pair<TreeNode, int[]>> queue = new LinkedList<>();
        queue.offer(new Pair<>(root, new int[]{0, 0}));

        while (!queue.isEmpty()) {
            Pair<TreeNode, int[]> pair = queue.poll();
            TreeNode node = pair.getKey();
            int col = pair.getValue()[0], row = pair.getValue()[1];

            map.putIfAbsent(col, new TreeMap<>());
            map.get(col).putIfAbsent(row, new PriorityQueue<>());
            map.get(col).get(row).offer(node.val);

            if (node.left != null) queue.offer(new Pair<>(node.left, new
            int[]{col - 1, row + 1}));
            if (node.right != null) queue.offer(new Pair<>(node.right, new
            int[]{col + 1, row + 1}));
        }

        List<List<Integer>> result = new ArrayList<>();
        for (TreeMap<Integer, PriorityQueue<Integer>> colMap : map.values())
        {
            List<Integer> colList = new ArrayList<>();
            for (PriorityQueue<Integer> nodes : colMap.values()) {
                while (!nodes.isEmpty()) {
```

```

        collList.add(nodes.poll());
    }
}
result.add(collList);
}

return result;
}
}

```

The screenshot displays a LeetCode submission interface. The top section shows the problem status as 'Accepted' with 34/34 test cases passed. The user 'Shiv Chauhan' submitted the solution on Feb 14, 2025, at 15:26. The runtime statistics indicate a runtime of 3 ms, which beats 82.70% of other submissions, and a memory usage of 42.92 MB, which beats 13.11%. A bar chart visualizes the runtime performance across different time intervals. The code editor shows a Java solution for the 'Maximum Number of Events That Can Be Attended' problem. The solution uses a TreeMap to store events and a PriorityQueue to process them. The test case results show that the solution passed all three test cases, with a runtime of 0 ms for the first case.

**Runtime Statistics:**

- Runtime: 3 ms (Beats 82.70%)
- Memory: 42.92 MB (Beats 13.11%)

**Code (Java):**

```

class Pair<K, V> {
    private K key;
    private V value;

    public Pair(K key, V value) {
        this.key = key;
        this.value = value;
    }
}

class Solution {
    public List<List<Integer>> verticalTraversal(TreeNode root) {
        TreeMap<Integer, TreeMap<Integer, PriorityQueue<Integer>>> map = new TreeMap<>();
        Queue<Pair<TreeNode, Integer>> queue = new LinkedList<>();
        queue.offer(new Pair<>(root, new Integer[]{0, 0}));

        while (!queue.isEmpty()) {
            Pair<TreeNode, Integer> pair = queue.poll();
            TreeNode node = pair.getKey();
            int col = pair.getValue()[0], row = pair.getValue()[1];

            map.putIfAbsent(col, new TreeMap<>());
            map.get(col).putIfAbsent(row, new PriorityQueue<>());
            map.get(col).get(row).offer(node.val);

            if (node.left != null) queue.offer(new Pair<>(node.left, new Integer[]{col - 1, row + 1}));
            if (node.right != null) queue.offer(new Pair<>(node.right, new Integer[]{col + 1, row + 1}));
        }

        List<List<Integer>> result = new ArrayList<>();
        for (int col : map.keySet()) {
            List<Integer> rowValues = new ArrayList<>();
            for (int row : map.get(col).keySet()) {
                rowValues.add(map.get(col).get(row).poll());
            }
            result.add(rowValues);
        }

        return result;
    }
}

```

**Testcase Results:**

- Case 1:** Accepted. Runtime: 0 ms. Input: `root = [3,9,20,null,null,15,7]`. Output: `[[9], [3,15], [20], [7]]`. Expected: `[[9], [3,15], [20], [7]]`.