

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 runtime is 0 ms, which is 100.00% faster than other submissions, and the memory usage is 41.84 MB, which is 27.14% less than other submissions. A bar chart shows the runtime distribution, with the user's submission being the fastest. The code editor on the right shows the Java solution, which is the same as the one provided in the first block. The bottom section shows the test results, with a single test case 'Case 1' passing. The test case input is '[1,null,2,3]' and the output is a binary tree diagram with root 1, left child 2, and right child 3.

Accepted 71 / 71 testcases passed
Shiv Chauhan submitted at Feb 14, 2025 14:49

Runtime: 0 ms, Beats 100.00%
Memory: 41.84 MB, Beats 27.14%

Code: Java

```
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);
    }
}
```

Testcase: Case 1, Case 2, Case 3, Case 4

root = [1,null,2,3]

Test Result: 1, 2, 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 'Accepted' tabs. The 'Accepted' tab is active, showing a submission by Shiv Chauhan on Feb 14, 2025, at 14:50. The performance metrics indicate a runtime of 0 ms (beats 100.00%) and memory usage of 42.05 MB (beats 29.37%). A bar chart shows the runtime distribution across 100% of submissions. The code editor on the right contains the Java solution. The test case section shows the input array [1, 2, 2, 3, 4, 4, 3] and a corresponding tree diagram where the root node 1 has two children, both labeled 2. Each of these nodes has two children, labeled 3 and 4 respectively, forming a symmetric structure.

```
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);
    }
}
```

Testcase 1: Case 1

root = [1, 2, 2, 3, 4, 4, 3]

```
graph TD
    1((1)) --- 2L((2))
    1 --- 2R((2))
    2L --- 3L((3))
    2L --- 4L((4))
    2R --- 4R((4))
    2R --- 3R((3))
```

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 displays the LeetCode submission interface for the problem "Maximum Depth of Binary Tree". The top section shows the problem description and the user's submission status: "Accepted" with 39/39 test cases passed. The user's name "Shiv Chauhan" and submission time "Feb 14, 2025 14:51" are also visible.

The "Runtime" section shows a performance of 0 ms, beating 100.00% of other submissions, and a memory usage of 42.67 MB, beating 60.81% of other submissions. A bar chart below this section shows the distribution of runtime performance across different time intervals.

The "Code" section displays the Java solution in a code editor. The code is as follows:

```
1 class Solution {
2     public int maxDepth(TreeNode root) {
3         if (root == null){
4             return 0;
5         }
6         int left = maxDepth(root.left);
7         int right = maxDepth(root.right);
8         return Math.max(left, right) + 1;
9     }
10 }
```

The "Testcase" section shows the input for Case 1: "[3,9,20,null,null,15,7]". Below the input, a binary tree diagram is shown, representing the structure of the input array. The root node is 3, with a left child 9 and a right child 20. Node 20 has a left child 15 and a right child 7.

At the bottom, there are links to "More challenges" and a "Write your notes here" section.

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);  
    }  
}
```

The screenshot displays a web browser window with the LeetCode problem "98. Validate Binary Search Tree" open. The problem description states: "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." Two examples are provided: Example 1 shows a tree with root 2, left child 1, and right child 3, with input [2,1,3] and output true. Example 2 shows a tree with root 5, left child 1, and right child 4, with input [5,1,4] and output false.

On the right side of the browser, a code editor shows the following Java solution:

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

Below the code editor, the "Testcase" tab is selected, showing "Accepted" results with a runtime of 0 ms. The "Case 1" tab is active, displaying the input [2,1,3] and the output true, which matches the expected result true.

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);

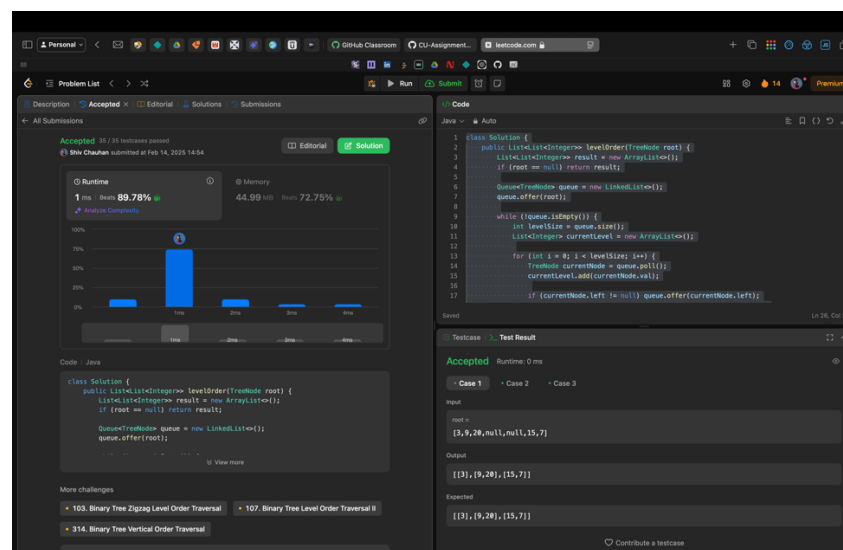
        while (!queue.isEmpty()) {
            int levelSize = queue.size();
            List<Integer> currentLevel = new ArrayList<>();

            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 shows a LeetCode submission interface. On the left, the 'Problem List' tab is active, displaying the problem 'Binary Tree Level Order Traversal II' (Problem 107). The submission is marked as 'Accepted' and shows performance metrics: Runtime 1 ms (94.47%), Memory 43.19 MB (32.77%). A bar chart shows the submission is faster than 94.47% of other submissions. The 'Code' tab on the right shows the Java code for the solution, which is identical to the code block above. The 'Testcase' tab at the bottom shows the input and output for a specific test case, which is 'Accepted'.

Accepted 34 / 34 testcases passed
Siv Chauhan submitted at Feb 14, 2025 14:57

Runtime 1 ms from 94.47%
Memory 43.19 MB from 32.77%

Code - Java

```
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;
    }
}
```

Testcase 1 - Case 1 - Case 2 - Case 3

Accepted Runtime: 0 ms

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

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

Expected: [[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 displays the LeetCode submission interface for the "Binary Tree ZigZag Order Traversal" problem. The submission is marked as "Accepted" with 33/33 test cases passed. The runtime is 0ms, beating 100.00% of other submissions, and the memory usage is 42.26 MB, beating 57.42%. The code editor shows the Java solution, which uses a queue and a boolean flag to alternate between left-to-right and right-to-left traversal for each level. The test case section shows an input tree structure and the expected zigzag output.

Runtime Performance:

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

Test Case:

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

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

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

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 top section shows the problem status as "Accepted" with 217/217 test cases passed. The user "Shiv Chauhan" submitted the solution on Feb 14, 2025, at 15:27. The performance metrics are shown in a bar chart: Runtime is 1 ms (Beats 72.53%), Memory is 41.65 MB (Beats 99.04%). The code is written in Java and is visible in the "Code" tab. The "Testcase" tab shows the input [1,2,3,null,5,null,4] and a diagram of the binary tree with root 1, left child 2, right child 3, and children 5 and 4 for node 2. The bottom section lists more challenges, including "116. Populating Next Right Pointers in Each Node" and "545. Boundary of Binary Tree".

Accepted 217 / 217 testcases passed
Shiv Chauhan submitted at Feb 14, 2025 15:27

Runtime 1 ms Beats 72.53%
Memory 41.65 MB Beats 99.04%

Code Java

```
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;
    }
}
```

Testcase Test Result

Case 1 Case 2 Case 3 Case 4

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

Diagram of the binary tree structure:

```
graph TD
    1((1)) --> 2((2))
    1 --> 3((3))
    2 --> 5((5))
    2 --> 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);
    }

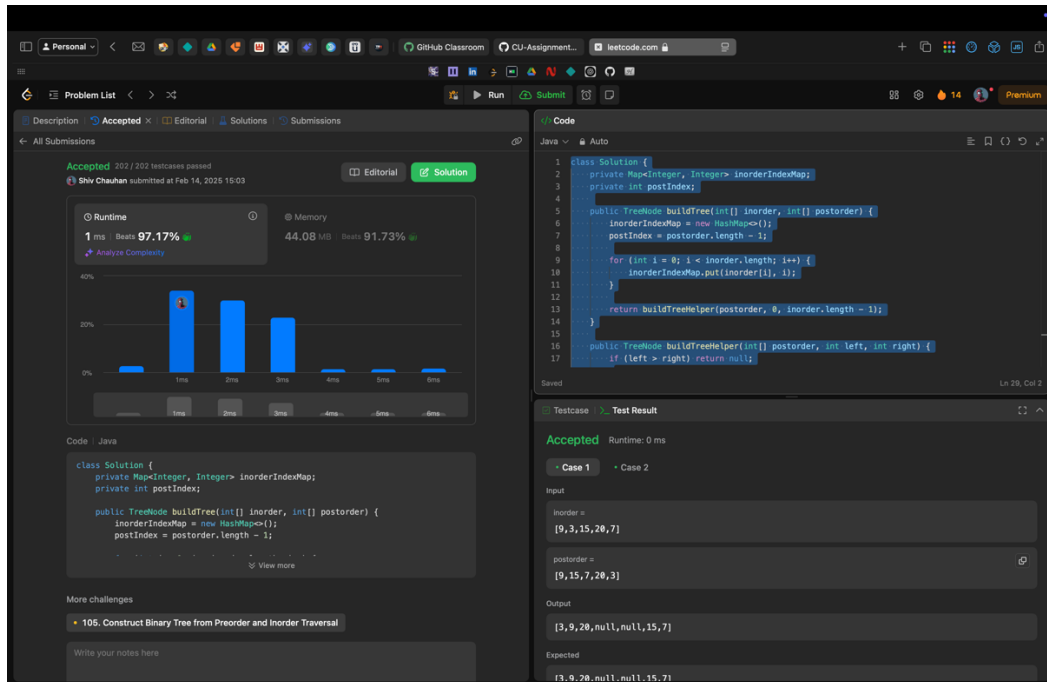
    public TreeNode buildTreeHelper(int[] postorder, int left, int right) {
        if (left > right) return null;

        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

```

1 class Solution {
2     public int findBottomLeftValue(TreeNode root) {
3         Queue<TreeNode> queue = new LinkedList<>();
4         queue.offer(root);
5         int leftmostValue = root.val;
6
7         while (!queue.isEmpty()) {
8             int levelSize = queue.size();
9             leftmostValue = queue.peek().val;
10
11             for (int i = 0; i < levelSize; i++) {
12                 TreeNode current = queue.poll();
13                 if (current.left != null) queue.offer(current.left);
14                 if (current.right != null) queue.offer(current.right);
15             }
16         }
17     }
18 }

```

Testcase 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%
0% 50% 100%
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. On the left, the 'Submissions' tab is active, showing a bar chart for runtime performance. The chart indicates a runtime of 3ms, which beats 82.70% of other submissions. Below the chart, the Java code for a 'Pair' class is visible. On the right, the 'Code' tab shows the full Java solution for a binary tree traversal problem. The solution uses a recursive approach with a queue to store nodes and their coordinates. The 'Testcase' tab shows the input '[3,9,20,null,null,15,7]' and the expected output '[[9],[3,15],[20],[7]]'. The submission is marked as 'Accepted' with a runtime of 0ms.

Runtime Performance:

Runtime	Beats
3 ms	82.70%

Java Code:

```

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

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

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

```

Testcase Results:

Accepted Runtime: 0 ms

Case 1: Input: [3,9,20,null,null,15,7] Output: [[9],[3,15],[20],[7]] Expected: [[9],[3,15],[20],[7]]