Name: Devesh

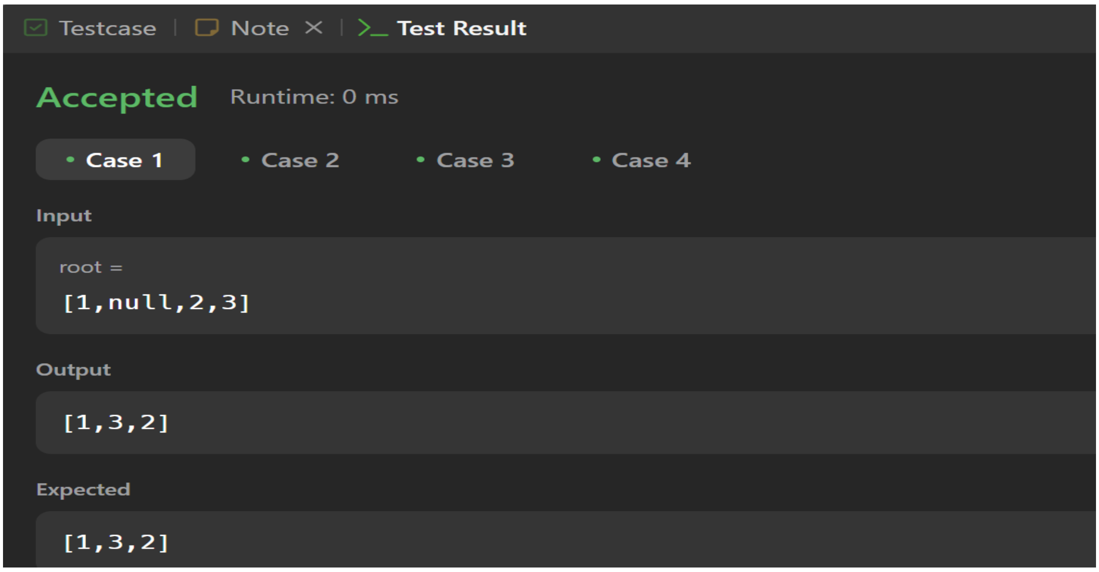
UID: 22BCS16690

Subject – Advance Programming Lab - II

Q 1 Binary Tree In order Traversal

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

OUTPUT:



Accepted Runtime: 0 ms

Input

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

Output

```
[1,3,2]
```

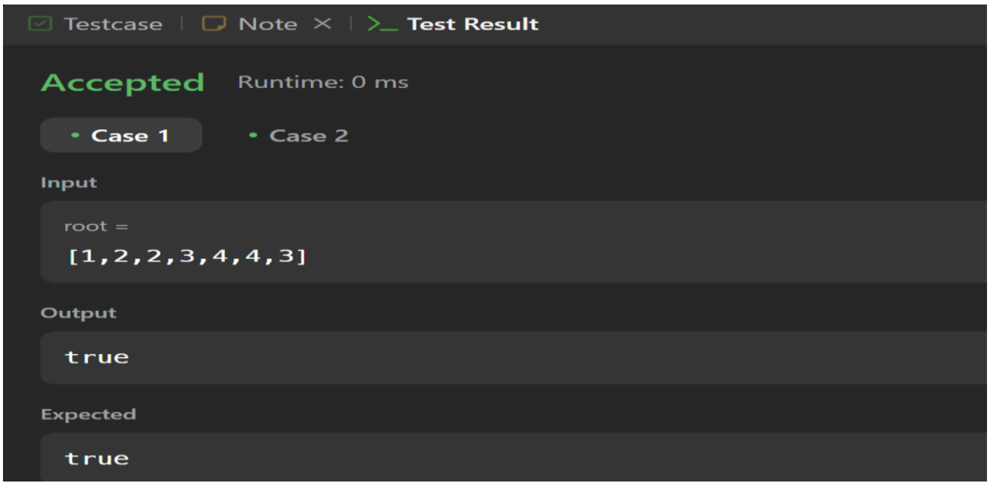
Expected

```
[1,3,2]
```

Q 2 Symmetric Tree

```
class Solution {
    public boolean isSymmetric(TreeNode root) {
       Queue<TreeNode> q = new LinkedList<>();
       q.add(root.left);
       q.add(root.right);
       while(!q.isEmpty()){
        TreeNode left = q.poll();
        TreeNode right = q.poll();
        if(left == null && right == null){
            continue; }
        if(left == null || right == null){
            return false; }
        if(left.val != right.val){
            return false;
        }
        q.add(left.left);
        q.add(right.right);
        q.add(left.right);
        q.add(right.left);
       }
       return true;   }}
```

OUTPUT:



Accepted Runtime: 0 ms

Input

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

Output

```
true
```

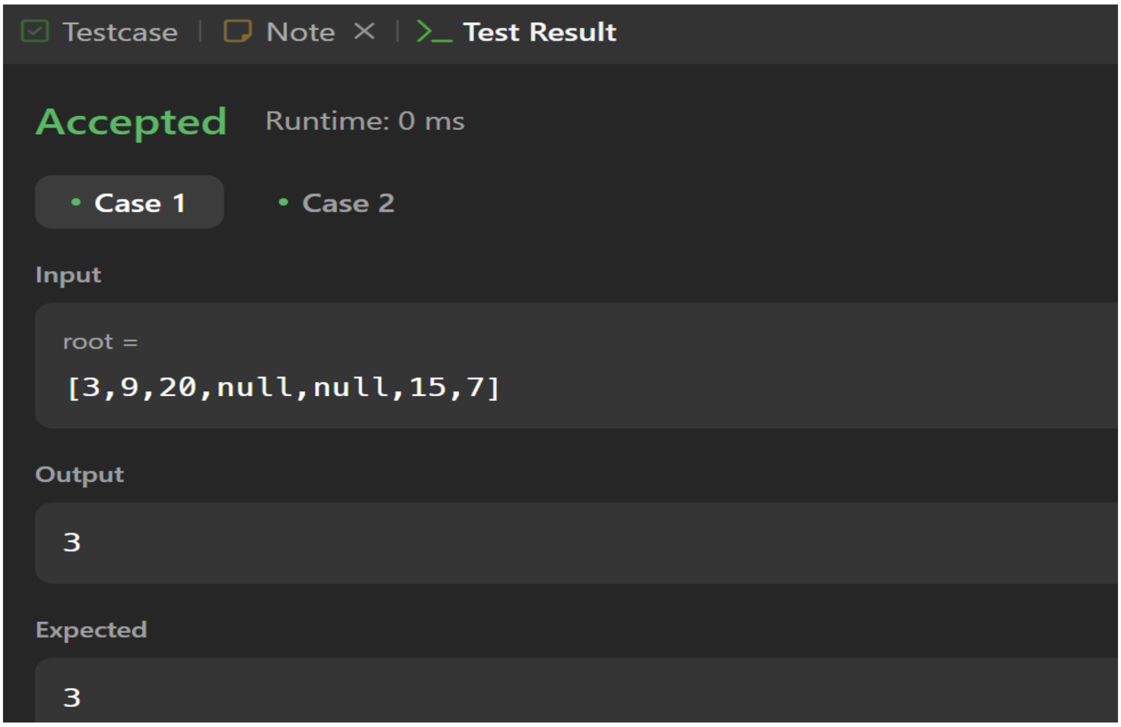
Expected

```
true
```

Q3 Maximum Depth of Binary Tree

```
class Solution {
    public int maxDepth(TreeNode root) {
      return   helper(root);
    }
    int helper(TreeNode root){
        if(root == null){
            return 0;
        }
        int length = 0;
      int left =  helper(root.left);
      int right =  helper(root.right);

       length = Math.max(left, right) + 1;
      return length;

    }
}
```

OUTPUT:



Accepted Runtime: 0 ms

Input

root =

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

Output

3

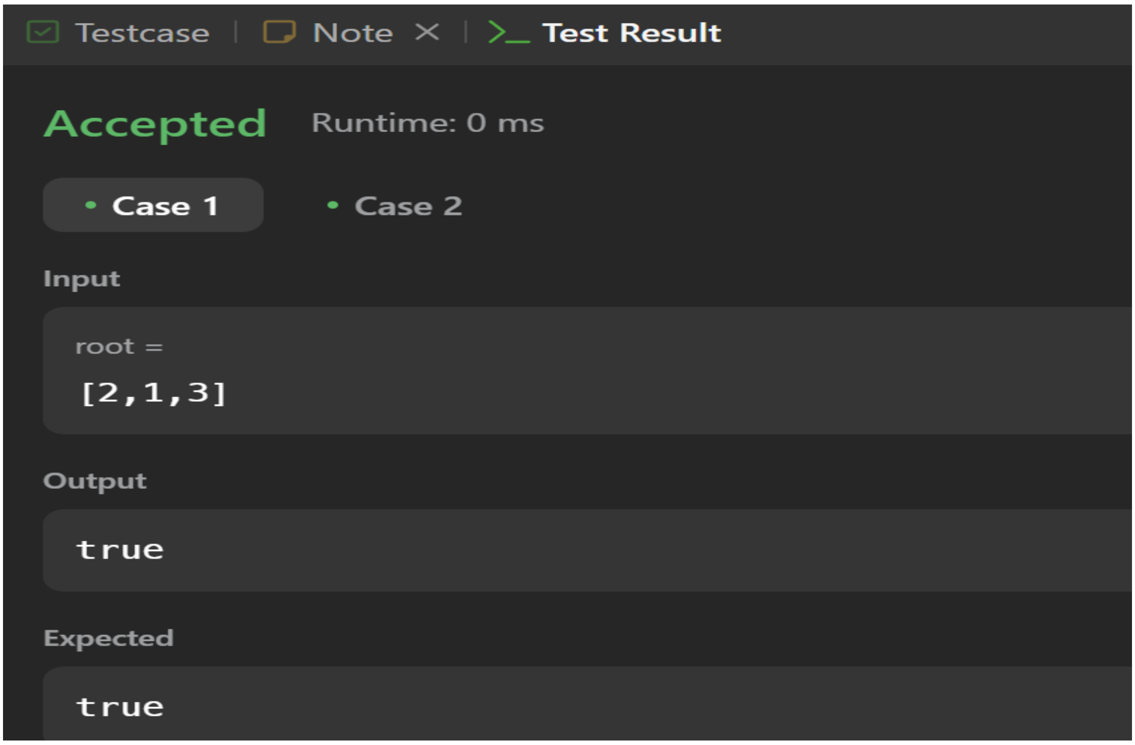
Expected

3

Q 4 Validate Binary Search Tree

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}
public class Solution {
    public boolean isValidBST(TreeNode root) {
        return helper(root, Long.MIN_VALUE, Long.MAX_VALUE);
    }
    private boolean helper(TreeNode node, long min, long max) {
        if (node == null) return true;
        if (node.val <= min || node.val >= max) return false;
        return helper(node.left, min, node.val) && helper(node.right, node.val, max);
    }
}
```

OUTPUT:



Accepted Runtime: 0 ms

Input

root =
[2,1,3]

Output

true

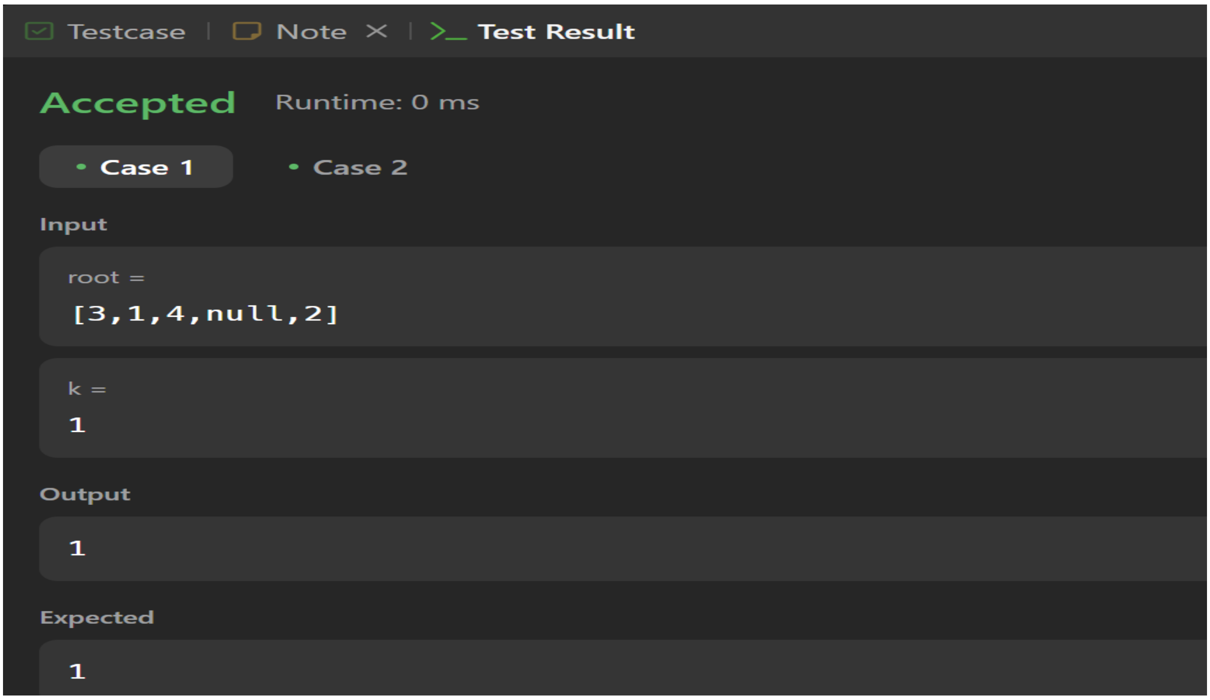
Expected

true

Q 5 Kth Smallest Element in a BST

```
class Solution {
   private int count = 0;
   public int result = 0;
   public int kthSmallest(TreeNode root, int k) {
      inorder(root, k);
      return result; }
   public void inorder(TreeNode root, int k) {
      if(root == null) {
         return;
      }
      inorder(root.left, k);
      count++;
      if(k == count) {
         result = root.val;
         return;
      }
      inorder(root.right, k);
   }}
```

OUTPUT:



Accepted   Runtime: 0 ms

Input

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

k =
1

Output

1

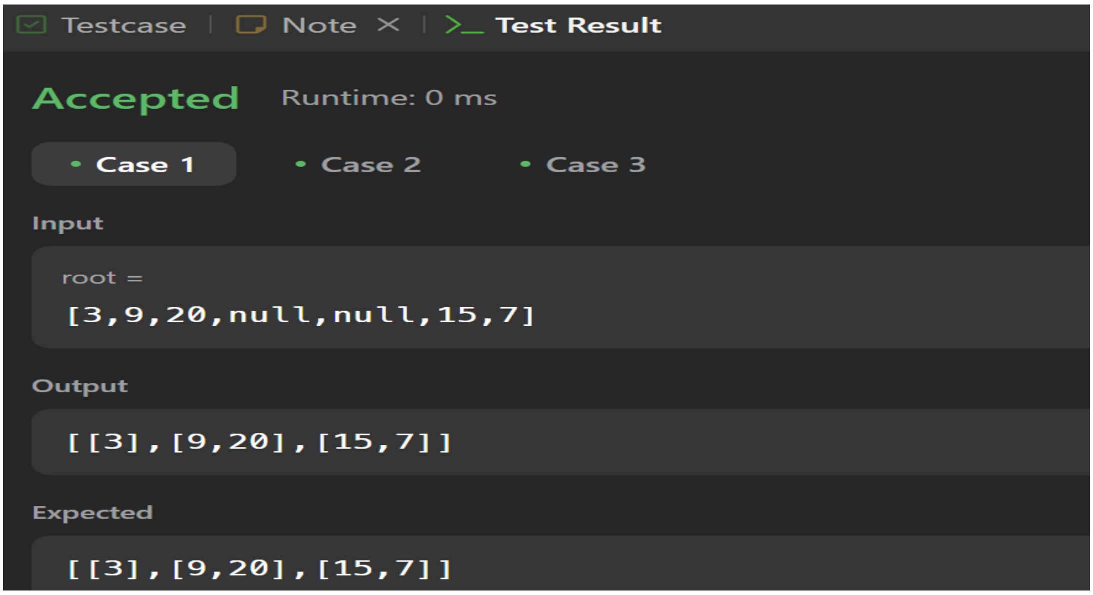
Expected

1

Q 6 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<>(levelSize);
      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;  } }
```

OUTPUT:



Accepted Runtime: 0 ms

Input

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

Output

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

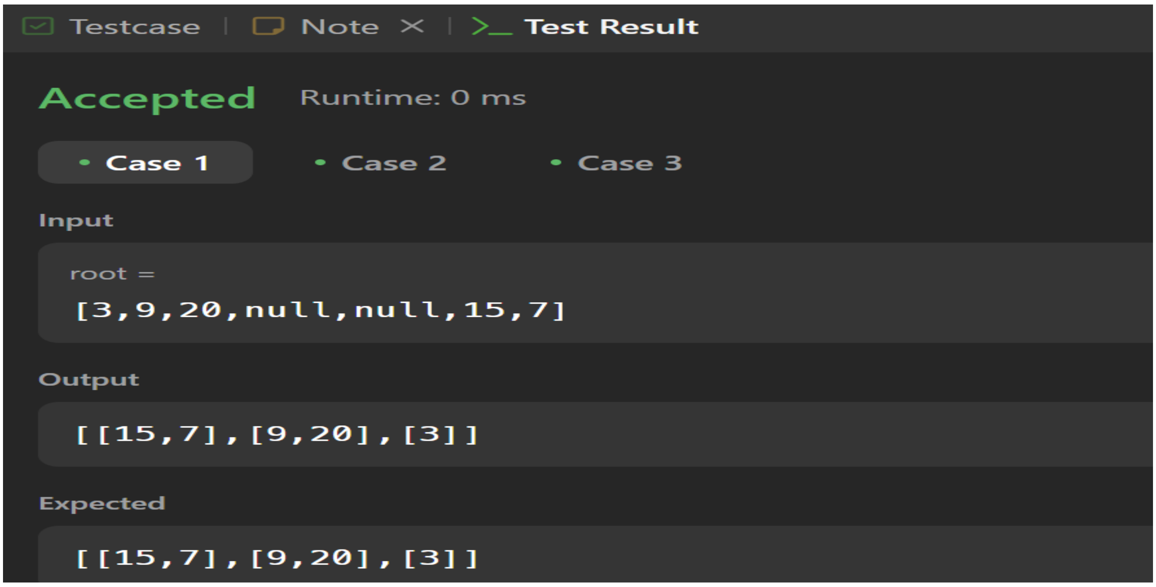
Expected

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

Q 7 Binary Tree Level Order Traversal II

```
class Solution {
    public List<List<Integer>> levelOrderBottom(TreeNode root) {
        List<List<Integer>> levels = new ArrayList<>();
        if (root == null) return levels;
        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        while (!queue.isEmpty()) {
            int n = queue.size();
            List<Integer> level = new ArrayList<>();
            for (int i = 0; i < n; 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);
            }
            levels.add(0, level);
        }
        return levels;
    }
}
```

OUTPUT:



Accepted Runtime: 0 ms

Input

root =

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

Output

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

Expected

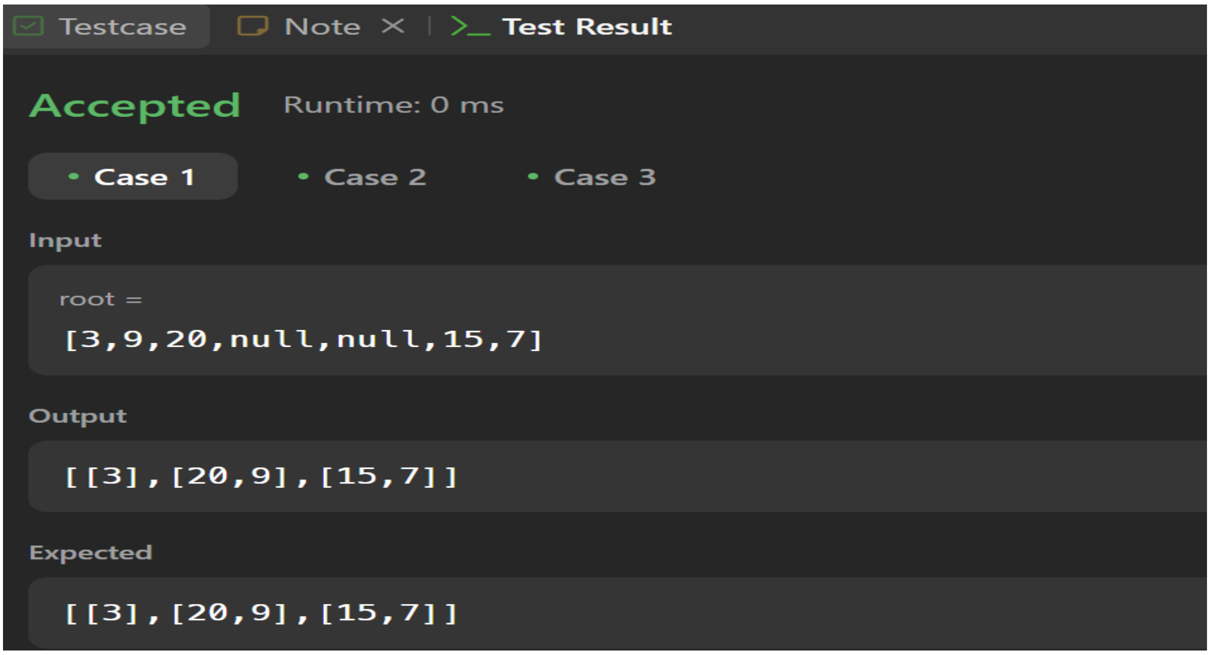
```
[[15,7],[9,20],[3]]
```

Q 8 Binary Tree Zigzag Level Order Traversal

```
class Solution {
public List<List<Integer>> zigzagLevelOrder(TreeNode root) {
   List<List<Integer>> result = new ArrayList<>();
if (root == null) {
  return result;
}
Deque<TreeNode> queue = new LinkedList<>();
queue.add(root);
boolean reverse = false;
while (!queue.isEmpty()) {
  int levelSize = queue.size();
  List<Integer> currentLevel = new ArrayList<>(levelSize);
  for (int i=0; i < levelSize; i++) {
    if (!reverse) {
      TreeNode currentNode = queue.pollFirst();
      currentLevel.add(currentNode.val);
      if (currentNode.left != null) {
        queue.addLast(currentNode.left);
      }
      if (currentNode.right != null) {
        queue.addLast(currentNode.right);
      }
    } else {
      TreeNode currentNode = queue.pollLast();
      currentLevel.add(currentNode.val);
      if (currentNode.right != null) {
        queue.addFirst(currentNode.right);
      }
```

```
          if (currentNode.left != null) {
            queue.addFirst(currentNode.left);
          }
        }
      }
      reverse = !reverse;
      result.add(currentLevel);
    }
    return result;
  }
    }
```

OUTPUT:

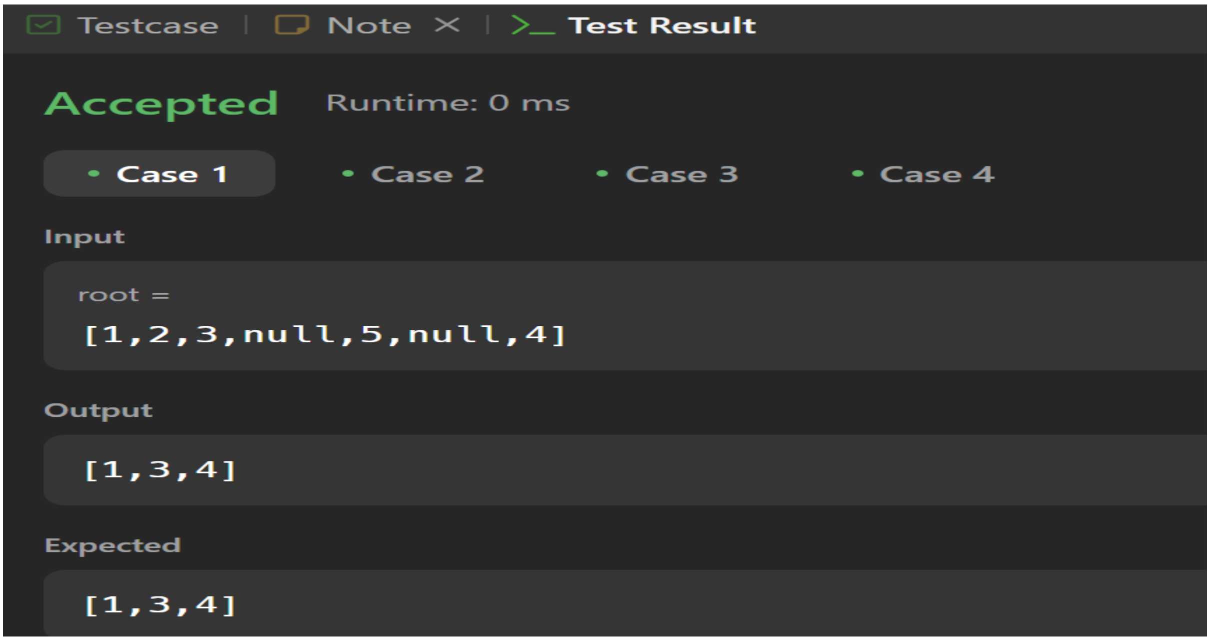


```
Accepted    Runtime: 0 ms

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

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

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

Q 9 Binary Tree Right Side View

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

OUTPUT:

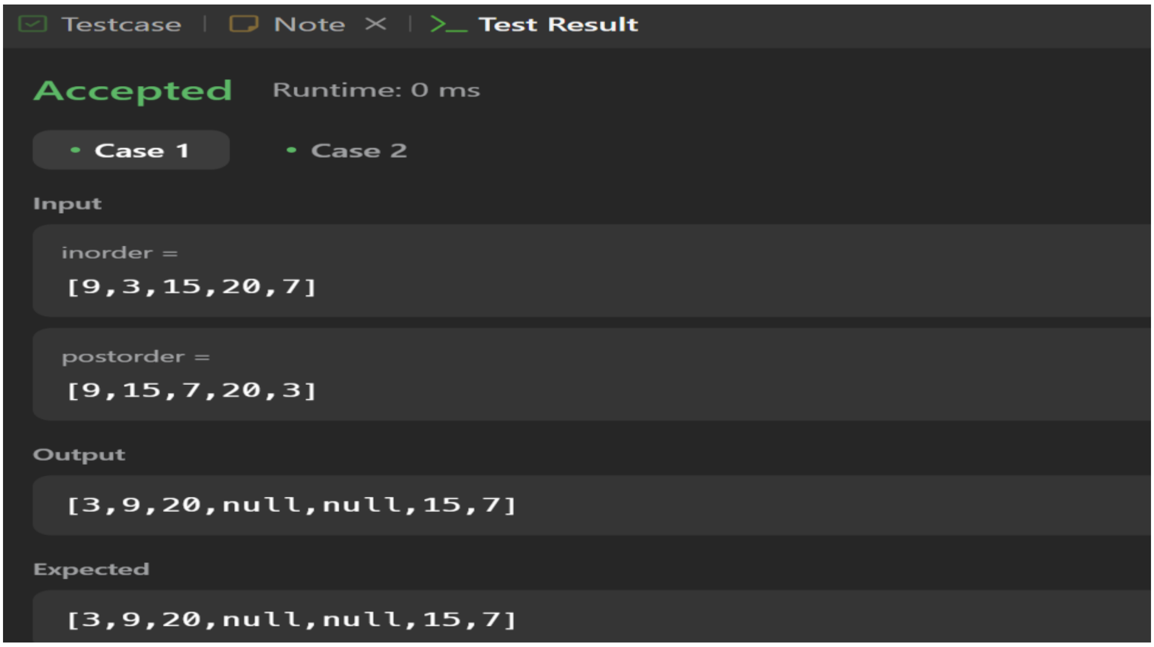


```
Accepted    Runtime: 0 ms

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

Output
[1,3,4]

Expected
[1,3,4]
```

Q 10 Construct Binary Tree from Inorder and Postorder Traversal

```
class Solution {
    public TreeNode buildTree(int[] inorder, int[] postorder) {
        return postOrIn(postorder, 0, postorder.length - 1, inorder, 0, inorder.length - 1);
    }
     public TreeNode postOrIn(int[] post, int psi, int pei, int[] in, int isi, int iei) {
        if (isi > iei)
            return null;
        int idx = isi;
        while (in[idx] != post[pei])
            idx++;
        int tel = idx - isi;
        TreeNode root = new TreeNode(post[pei]);
        root.left = postOrIn(post, psi, psi + tel - 1, in, isi, idx - 1);
        root.right = postOrIn(post, psi + tel, pei - 1, in, idx + 1, iei);
        return root;
    }
}
```

OUTPUT:



Accepted Runtime: 0 ms

Input

inorder =

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

postorder =

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

Output

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

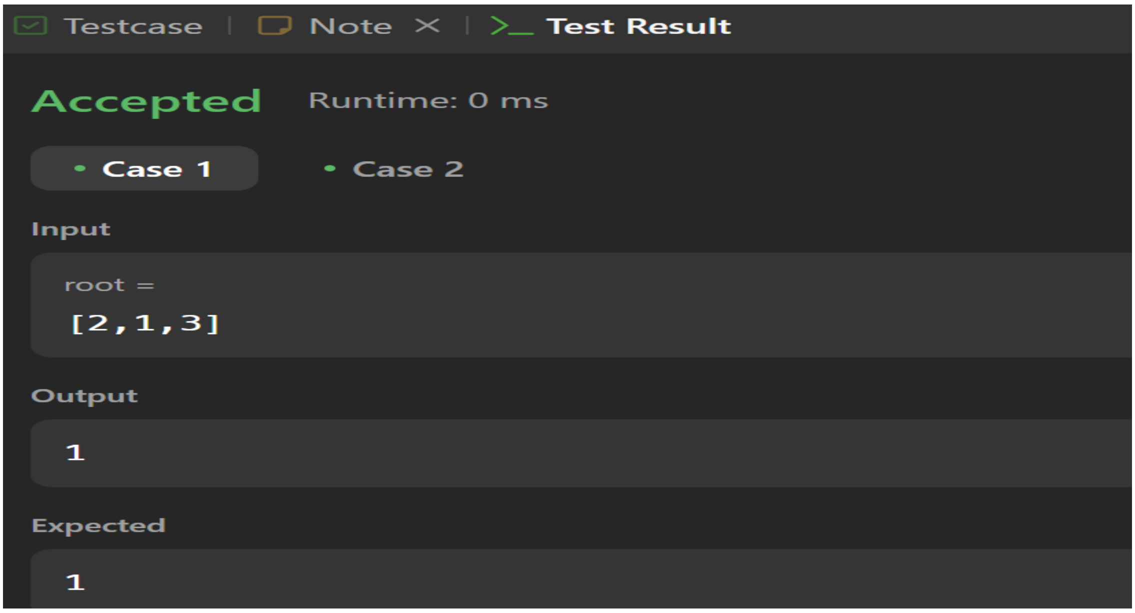
Expected

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

Q 11 Find Bottom Left Tree Value

```
public class Solution {
   public int findBottomLeftValue(TreeNode root) {
      if (root == null)
         return 0;
      Queue<TreeNode> q = new LinkedList<>();
      q.add(root);
      int leftNode = 0;
      while (!q.isEmpty()) {
         int size = q.size();
         leftNode = q.peek().val;
         for (int i = 0; i < size; i++) {
            TreeNode tmp = q.poll();
            if (tmp.left != null)
               q.add(tmp.left);
            if (tmp.right != null)
               q.add(tmp.right); } }
      return leftNode; } }
```

OUTPUT:



Accepted Runtime: 0 ms

Input

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

Output

```
1
```

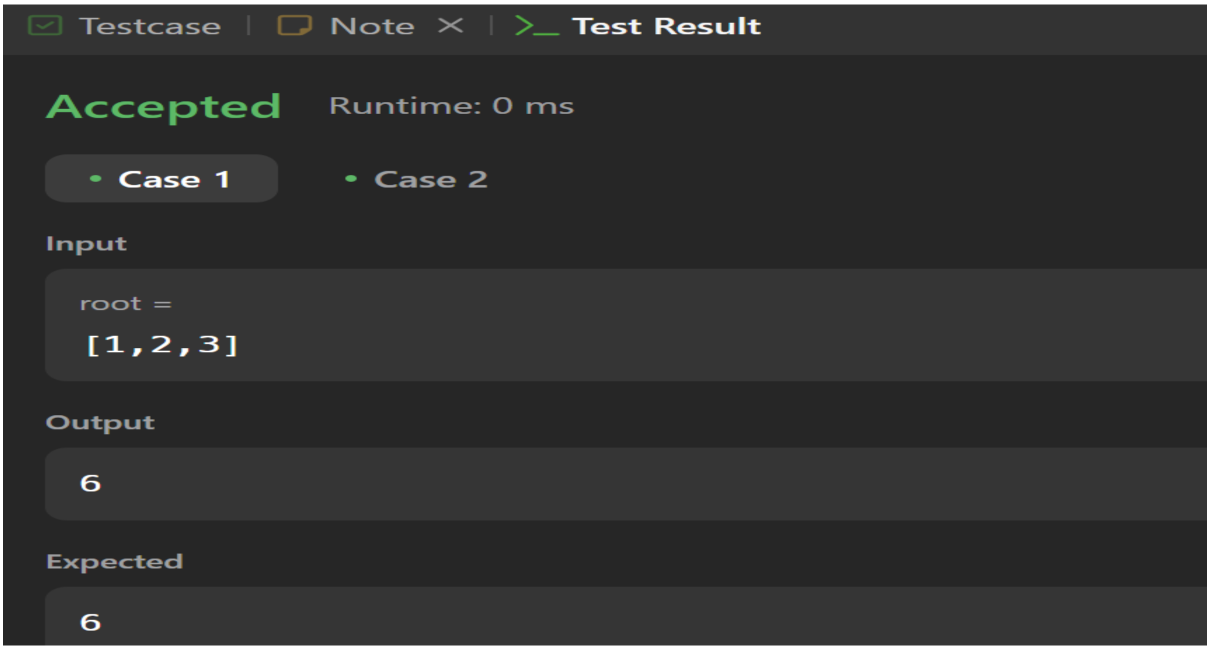
Expected

```
1
```

Q 12 Binary Tree Maximum Path Sum

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

OUTPUT:



Accepted Runtime: 0 ms

Input

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

Output

```
6
```

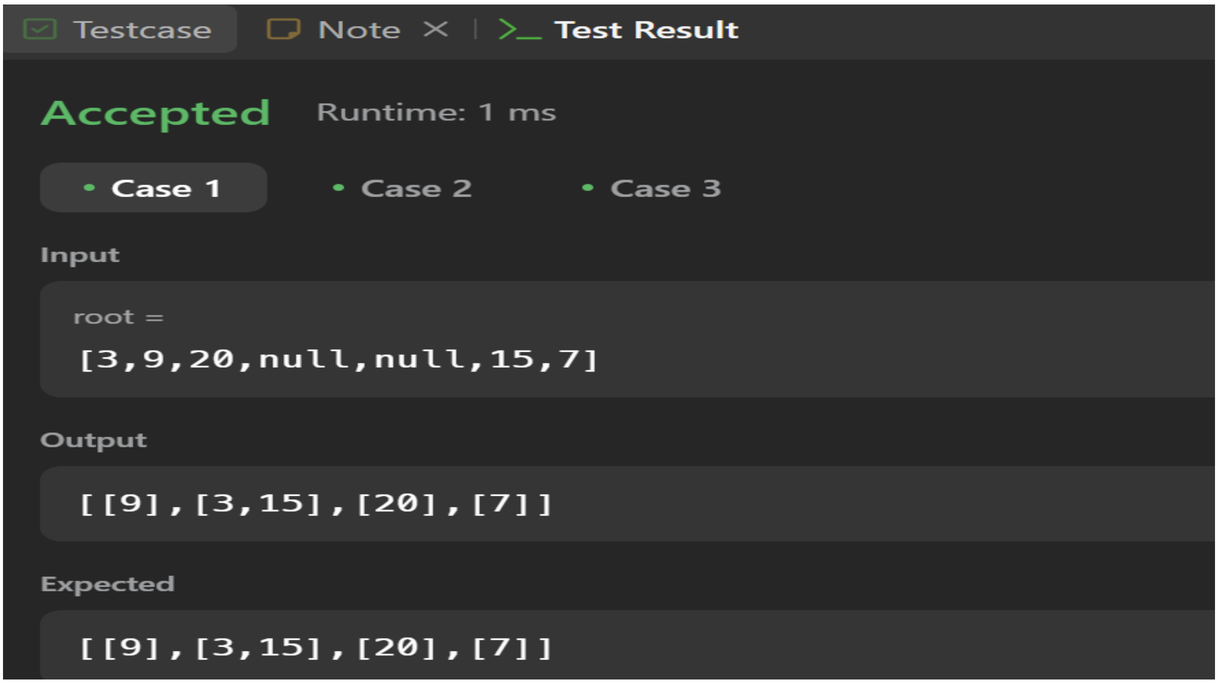
Expected

```
6
```

Q 13 Vertical Order Traversal of a Binary Tree

```
class Solution {
    public List<List<Integer>> verticalTraversal(TreeNode root) {
        List<List<Integer>> collection = new ArrayList<>();


        TreeMap<Integer, Map<Integer, List<Integer>>> treeMap = new TreeMap<>();
        inorderTraversal(root, treeMap, 0, 0);
        for (var colEntry : treeMap.entrySet()) {
            List<Integer> list = new ArrayList<>();


            for (var rowValues : colEntry.getValue().values()) {
                Collections.sort(rowValues);
                list.addAll(rowValues);
            }
            collection.add(list);
        }
        return collection;
    }
 private static void inorderTraversal(TreeNode node, TreeMap<Integer, Map<Integer,
List<Integer>>> treeMap, int column,  int row) {
        if (node == null) {
            return;
        }
        treeMap.computeIfAbsent(column, k -> new TreeMap<>()).computeIfAbsent(row, k ->
new ArrayList<>()).add(node.val);
        inorderTraversal(node.left, treeMap, column - 1, row + 1);
        inorderTraversal(node.right, treeMap, column + 1, row + 1);
    }
}
```

OUTPUT:



Accepted Runtime: 1 ms

Input

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

Output

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

Expected

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