EXPERIMENT-6

Student Name: Shubham Sharma	**UID:**22BET10358
Branch: BE -IT	**Section/Group:**22BET_IOT-703(A)
Semester: 6th	**Subject Code:** 22ITP-351

PROBLEM-1

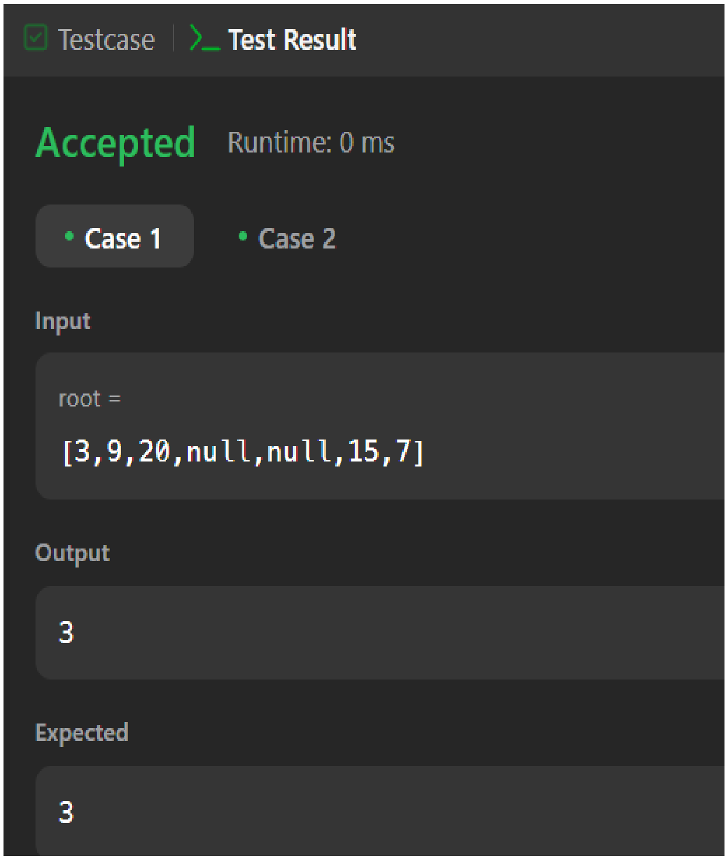
AIM:-

Maximum Depth of Binary Tree

CODE:-

```
class Solution {
    public int maxDepth(TreeNode root) {

        if (root == null) return 0;
        int leftDepth = maxDepth(root.left);
        int rightDepth = maxDepth(root.right);

        return 1 + Math.max(leftDepth, rightDepth);
    }
}
```

OUTPUT:-



Accepted Runtime: 0 ms

Input

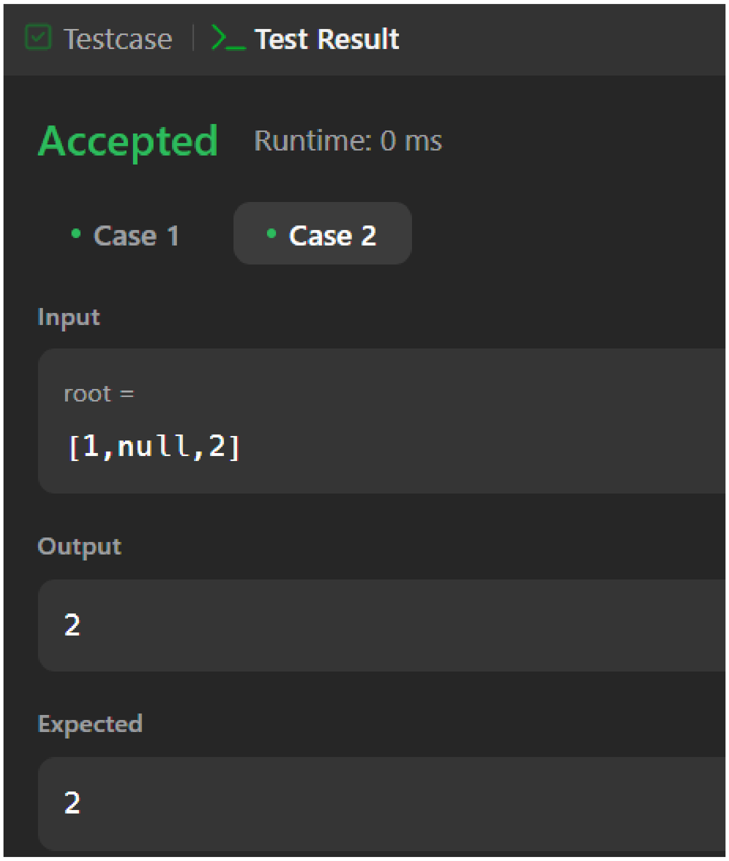
```
root =
[3,9,20,null,null,15,7]
```

Output

```
3
```

Expected

```
3
```



Accepted Runtime: 0 ms

Input

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

Output

```
2
```

Expected

```
2
```

PROBLEM-2

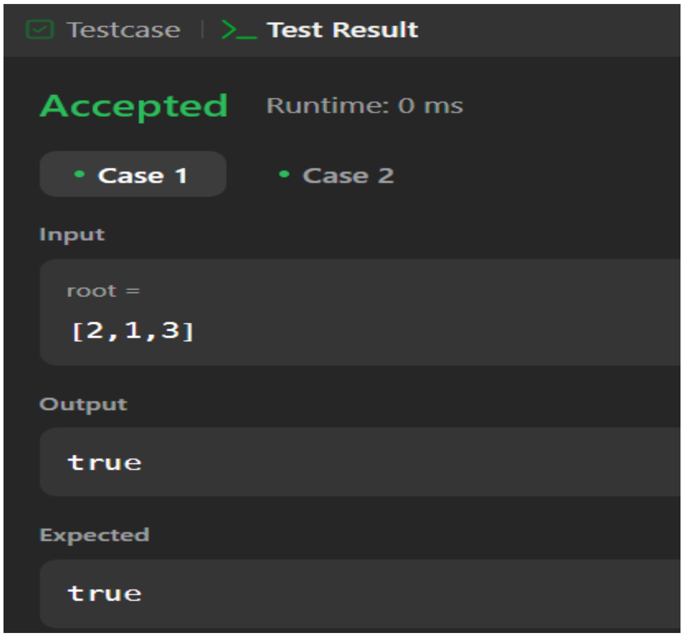
AIM:-

Validate Binary Search Tree

CODE:-

```
class Solution {
    public void inorder(List<Integer> res,TreeNode root)
    {
        if(root==null)
        {
            return;
        }
        inorder(res,root.left);
        res.add(root.val);
        inorder(res,root.right);
    }
    public boolean isValidBST(TreeNode root) {
        ArrayList<Integer> res=new ArrayList<Integer>();
        inorder(res,root);
        for(int i=0;i<res.size()-1;i++)
        {
            if(res.get(i)>=res.get(i+1))
            {
                return false;
            }
        }
        return true;
    }
}
```

OUTPUT:-



Accepted Runtime: 0 ms

Input

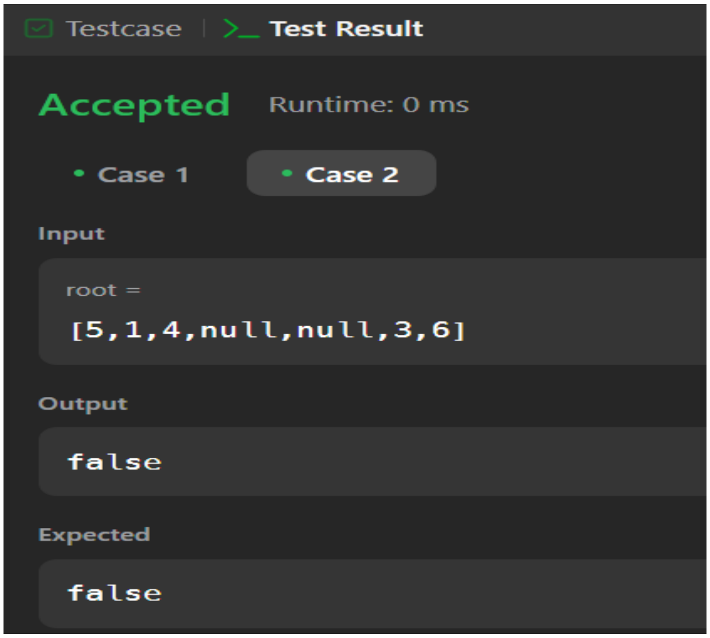
root =
[2,1,3]

Output

true

Expected

true



Accepted Runtime: 0 ms

Input

root =
[5,1,4,null,null,3,6]

Output

false

Expected

false

PROBLEM-3

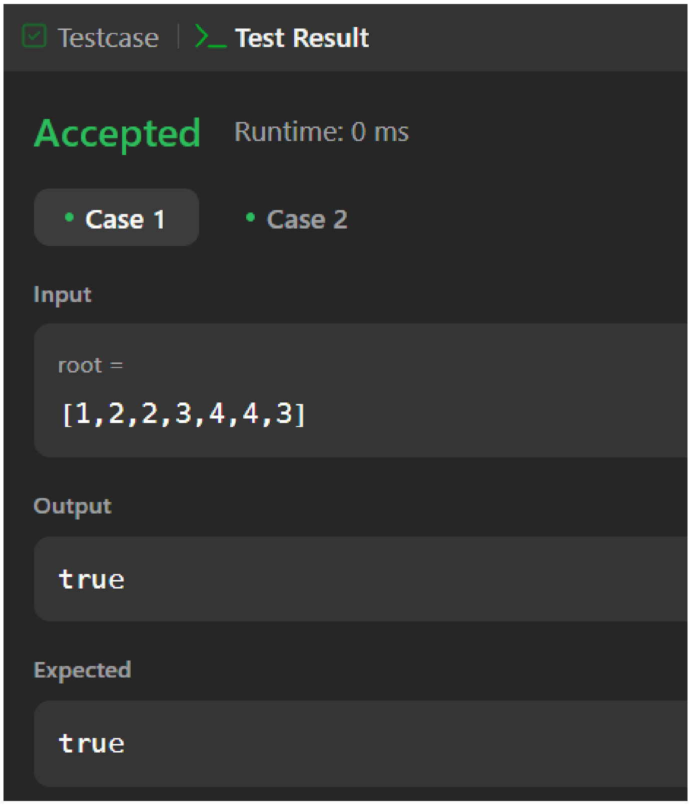
AIM:-

Symmetric Tree

CODE:-

```
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);
    }
}
```

OUTPUT:-



Accepted Runtime: 0 ms

Input

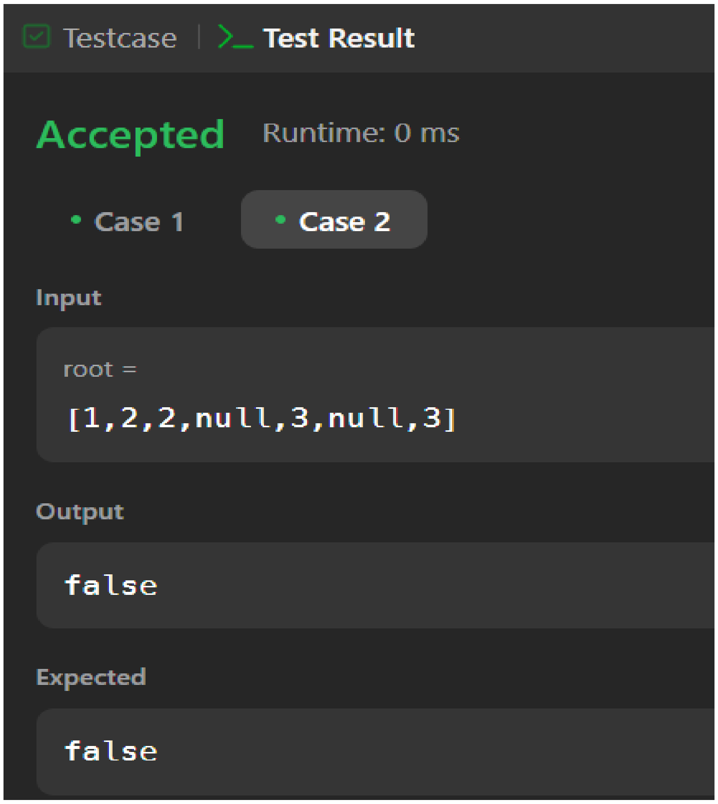
```
root =
[1,2,2,3,4,4,3]
```

Output

```
true
```

Expected

```
true
```



Accepted Runtime: 0 ms

Input

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

Output

```
false
```

Expected

```
false
```

PROBLEM-4

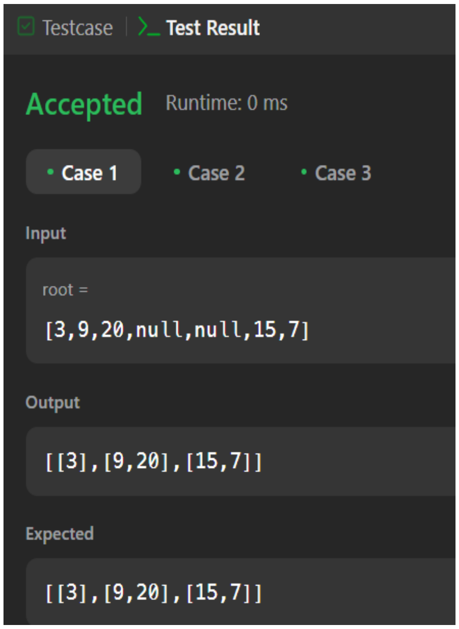
AIM:-

Binary Tree Level Order Traversal

CODE:-

```
class Solution {
    public List<List<Integer>> levelOrder(TreeNode root) {
        Queue<TreeNode> q = new LinkedList<>();
        List<List<Integer>> finalAns = new ArrayList<List<Integer>>();
        if(root==null){
            return finalAns;
        }
        q.add(root);
        while(!q.isEmpty()){
            int levels = q.size();
            List<Integer> subLevels = new ArrayList<>();
            for(int i=0;i<levels;i++){
                if(q.peek().left!=null){
                    q.add(q.peek().left);
                }
                if(q.peek().right!=null){
                    q.add(q.peek().right);
                }
                subLevels.add(q.remove().val);
            }
            finalAns.add(subLevels);
        }
        return finalAns;
    }
}
```

OUTPUT:-



Accepted Runtime: 0 ms

Input

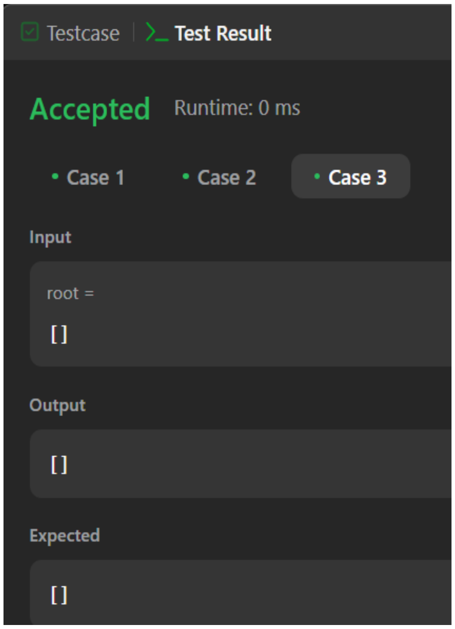
```
root =
[3,9,20,null,null,15,7]
```

Output

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

Expected

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



Accepted Runtime: 0 ms

Input

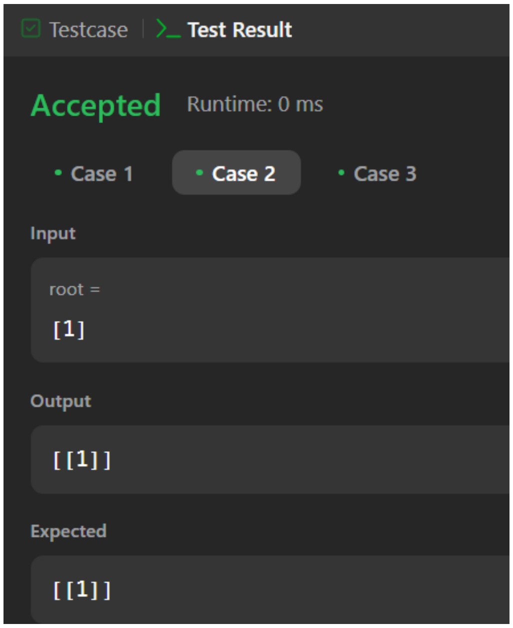
```
root =
[]
```

Output

```
[]
```

Expected

```
[]
```



Accepted Runtime: 0 ms

Input

```
root =
[1]
```

Output

```
[[1]]
```

Expected

```
[[1]]
```

PROBLEM-5

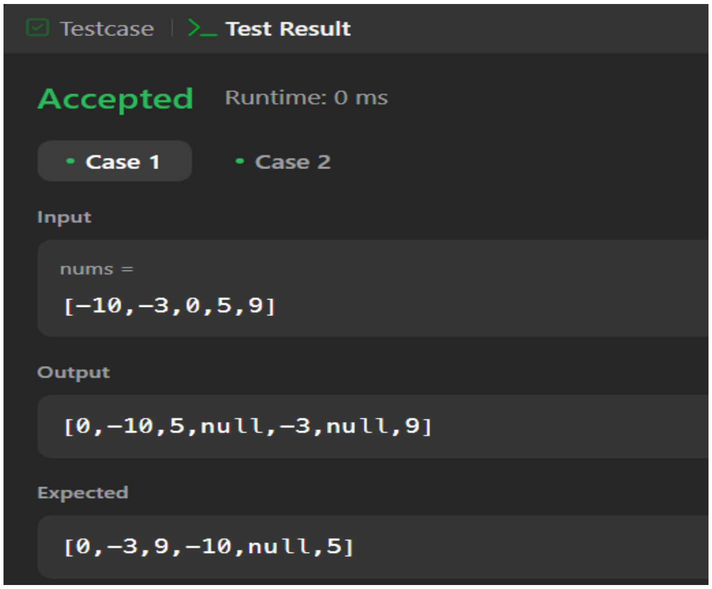
AIM:-

Convert Sorted Array to Binary Search Tree

CODE:-

```
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 TreeNode sortedArrayToBST(int[] nums) {
        return helper(nums, 0, nums.length - 1);
    }

    private TreeNode helper(int[] nums, int left, int right) {
        if (left > right) return null;
        int mid = (left + right) / 2;
        TreeNode root = new TreeNode(nums[mid]);
        root.left = helper(nums, left, mid - 1);
        root.right = helper(nums, mid + 1, right);
        return root;
    }
}
```

OUTPUT:-



Accepted Runtime: 0 ms

Input

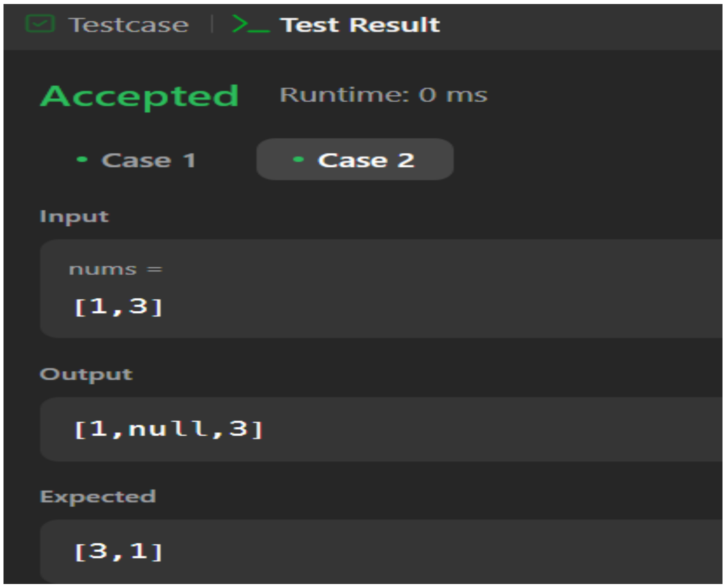
```
nums =
[-10,-3,0,5,9]
```

Output

```
[0,-10,5,null,-3,null,9]
```

Expected

```
[0,-3,9,-10,null,5]
```



Accepted Runtime: 0 ms

Input

```
nums =
[1,3]
```

Output

```
[1,null,3]
```

Expected

```
[3,1]
```

PROBLEM-6

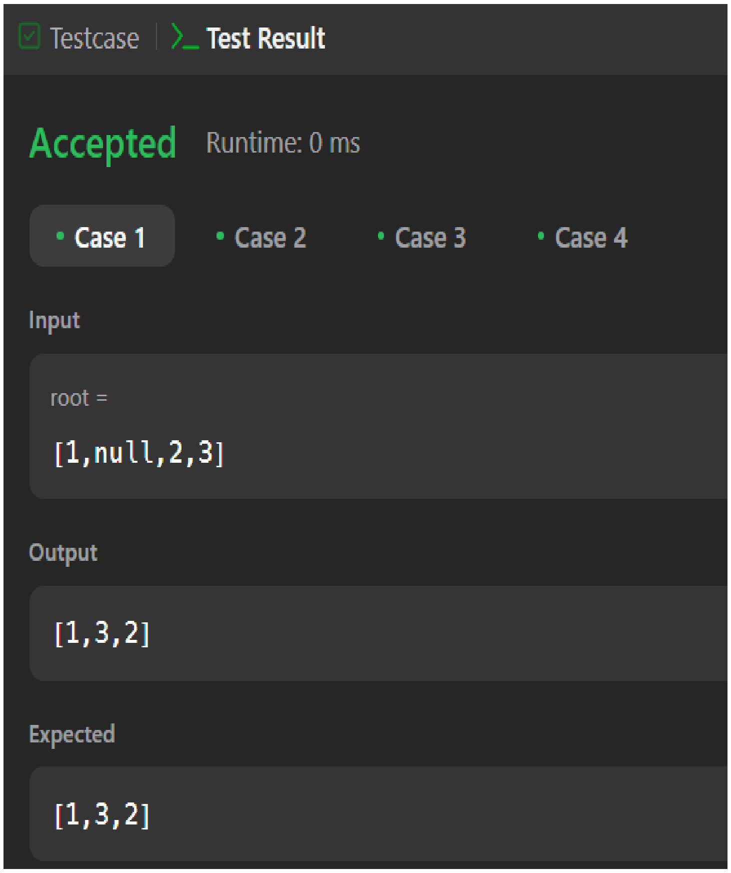
AIM:-

Binary Tree Inorder Traversal

CODE:-

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

        inorder(root, res);
        return res;
    }

    private void inorder(TreeNode node, List<Integer> res) {
        if (node == null) {
            return;
        }
        inorder(node.left, res);
        res.add(node.val);
        inorder(node.right, res);
    }
}
```

OUTPUT:-



Accepted Runtime: 0 ms

Input

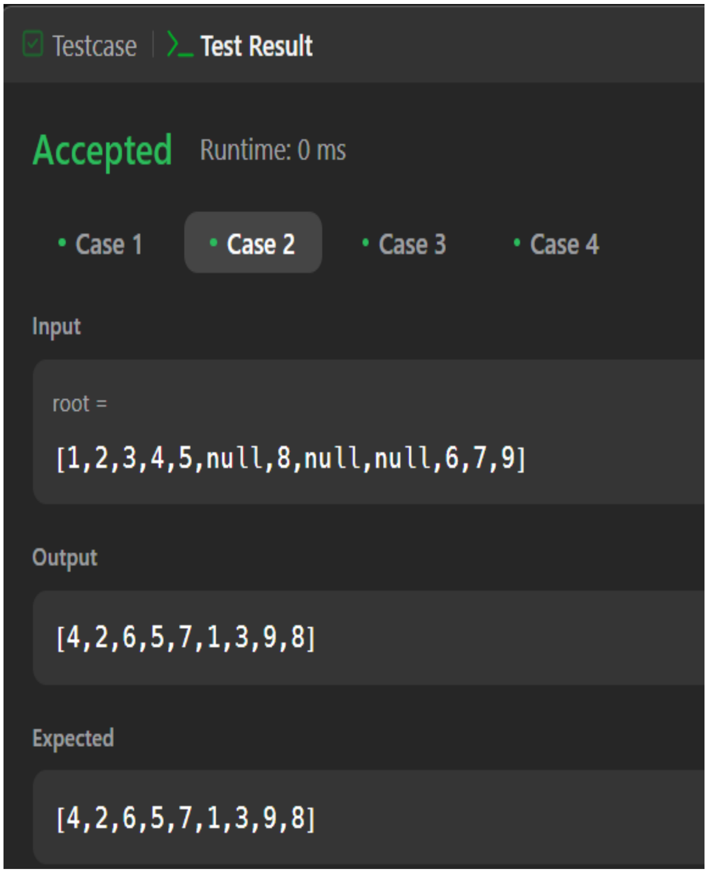
```
root =
[1,null,2,3]
```

Output

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

Expected

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



Accepted Runtime: 0 ms

Input

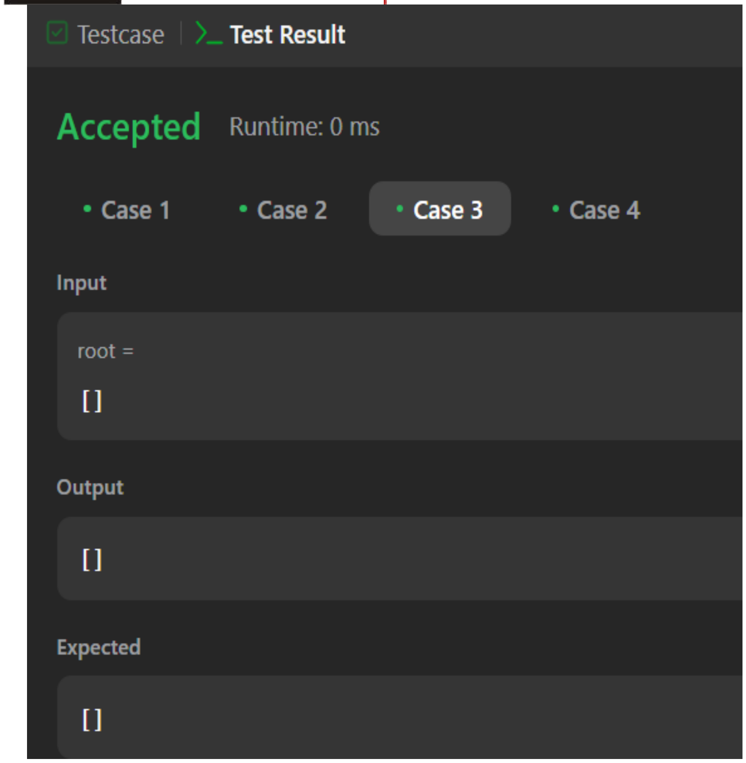
```
root =
[1,2,3,4,5,null,8,null,null,6,7,9]
```

Output

```
[4,2,6,5,7,1,3,9,8]
```

Expected

```
[4,2,6,5,7,1,3,9,8]
```



Accepted Runtime: 0 ms

Input

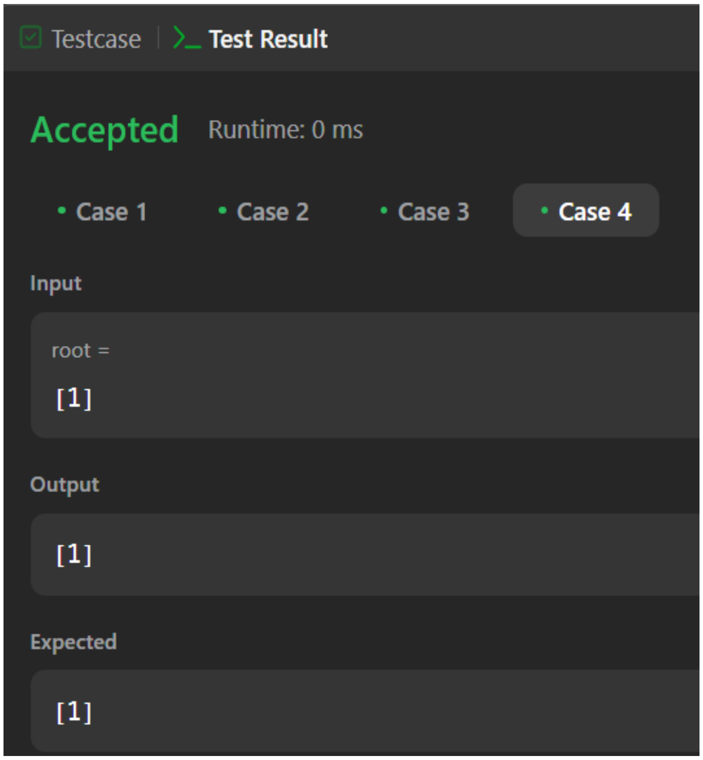
```
root =
[]
```

Output

```
[]
```

Expected

```
[]
```



Accepted Runtime: 0 ms

Input

```
root =
[1]
```

Output

```
[1]
```

Expected

```
[1]
```

PROBLEM-7

AIM:-

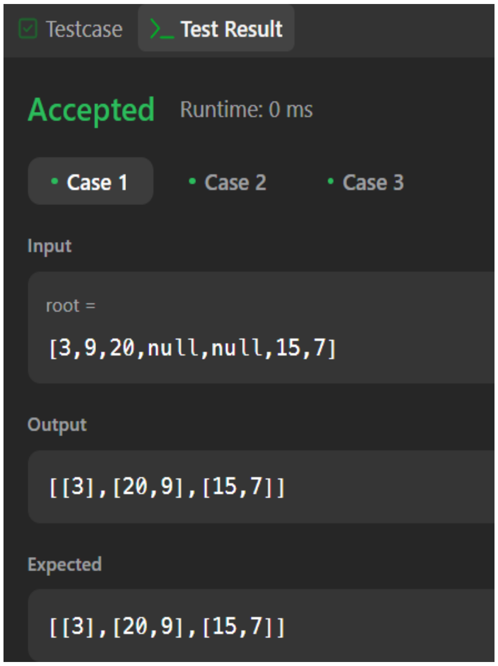
Binary Zigzag Level Order Traversal

CODE:-

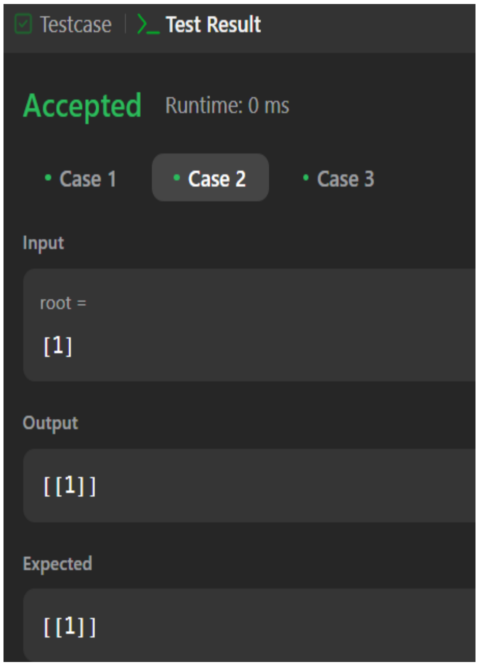
```
public class Solution {
   public List<List<Integer>> zigzagLevelOrder(TreeNode root)
   {
      List<List<Integer>> sol = new ArrayList<>();
      travel(root, sol, 0);
      return sol;
   }

   private void travel(TreeNode curr, List<List<Integer>> sol, int level)
   {
      if(curr == null) return;

      if(sol.size() <= level)
      {
         List<Integer> newLevel = new LinkedList<>();
         sol.add(newLevel);
      }

      List<Integer> collection  = sol.get(level);
      if(level % 2 == 0) collection.add(curr.val);
      else collection.add(0, curr.val);

      travel(curr.left, sol, level + 1);
```

```
            travel(curr.right, sol, level + 1);
        }
    }
```

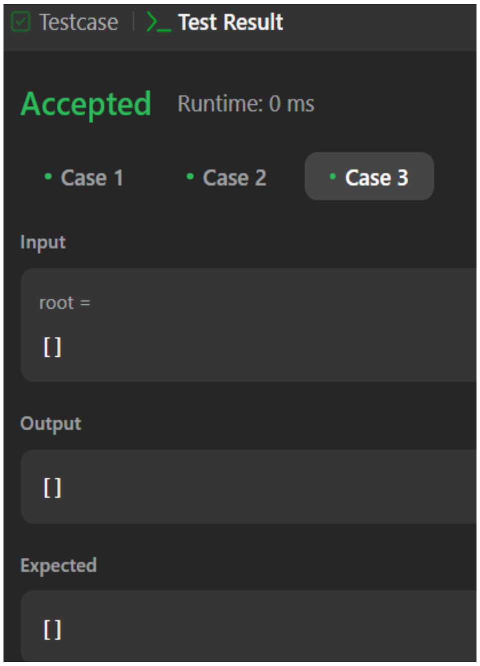
OUTPUT:-



```
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]]
```



```
Testcase    >_ Test Result

Accepted   Runtime: 0 ms

• Case 1   • Case 2   • Case 3

Input
root =
[1]

Output
[[1]]

Expected
[[1]]
```



```
Testcase    >_ Test Result

Accepted   Runtime: 0 ms

• Case 1   • Case 2   • Case 3

Input
root =
[]

Output
[]

Expected
[]
```

PROBLEM-8

AIM:-

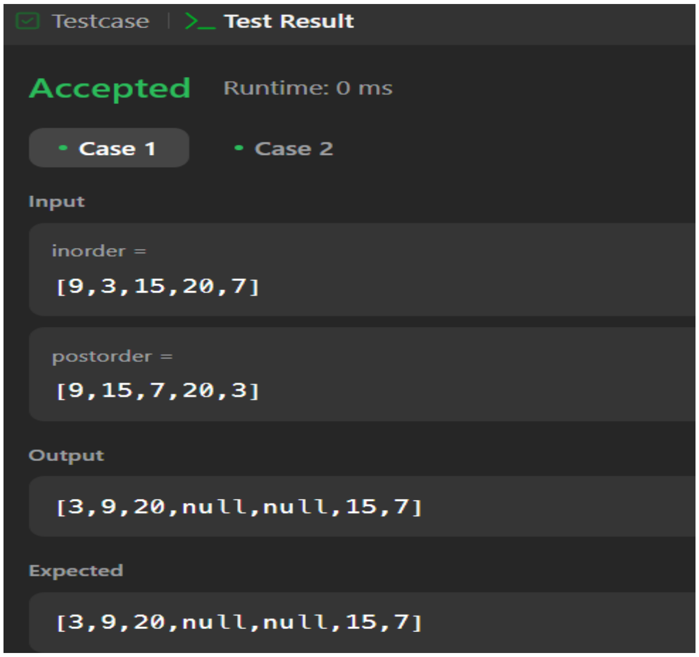
Construct Binary Tree from Inorder and Postorder Traversal

CODE:-

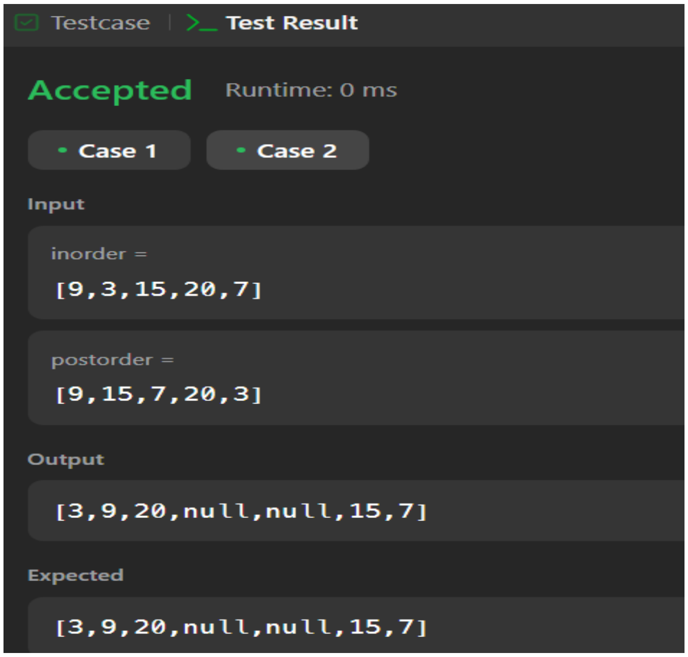
```
class Solution {
    public TreeNode buildTree(int[] in, int[] post) {
        HashMap<Integer,Integer> map=new HashMap<>();
        for(int i=0;i<in.length;i++){
            map.put(in[i],i);
        }
        return helper(in,post,map,0,post.length-1);
    }
    int ind=0;
    private TreeNode helper(int[] in,int[] post,HashMap<Integer,Integer> map,int s,int e){
        if(s>e){
            return null;
        }
        int val=post[post.length-1-ind];
        ind++;
        TreeNode root=new TreeNode(val);
        if(s==e){
            return root;
```

```
                }
                int i=map.get(val);
                root.right=helper(in,post,map,i+1,e);
                root.left=helper(in,post,map,s,i-1);
                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]
```



```
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]
```

PROBLEM-9

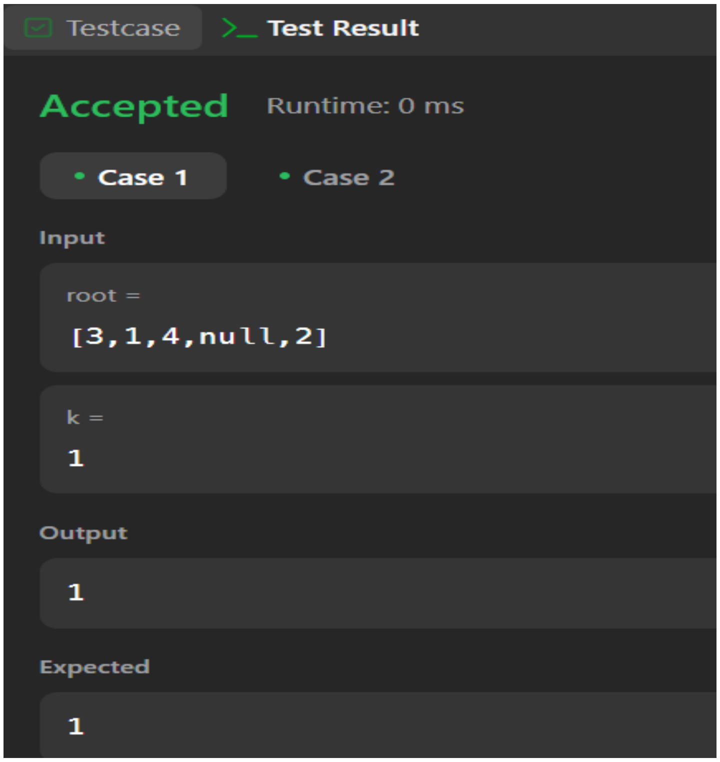
AIM:-

Kth Smallest element in a BST

CODE:-

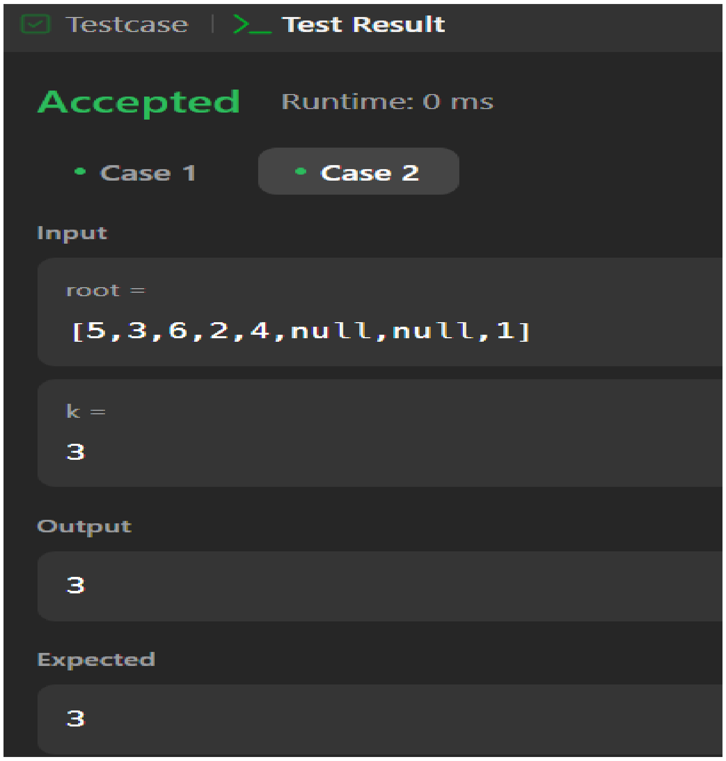
```
class Solution {
    private int count = 0;
    public int kthSmallest(TreeNode root, int k) {
        TreeNode result = helper(root, k);
        return result != null ? result.val : 0;
    }

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

        TreeNode left = helper(root.left, k);
        if (left != null) return left;

        count++;
        if (count == k) return root;
        return helper(root.right, k);
    }
}
```

OUTPUT:-



Accepted Runtime: 0 ms

```
Input
root =
[3,1,4,null,2]
k =
1
Output
1
Expected
1
```



Accepted Runtime: 0 ms

```
Input
root =
[5,3,6,2,4,null,null,1]
k =
3
Output
3
Expected
3
```

PROBLEM-10

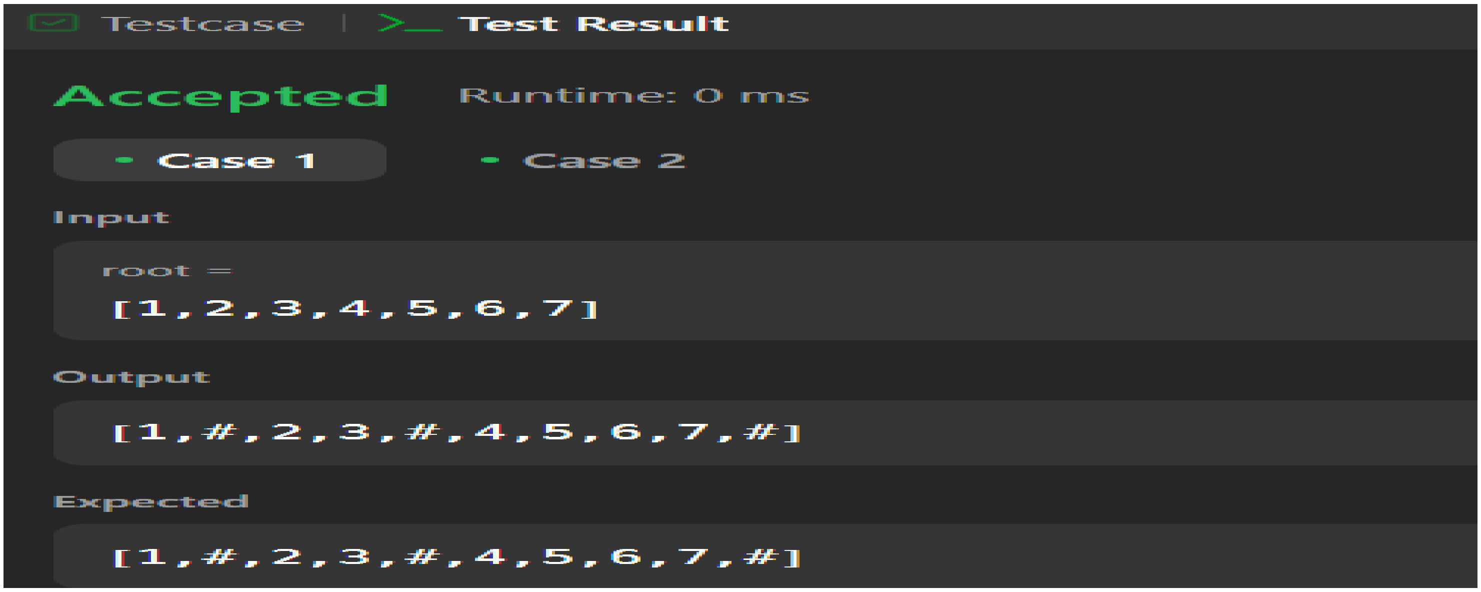
AIM:-

Populating Next Right Pointers in Each Node

CODE:-

```
class Solution {
    public Node connect(Node root) {
        Queue<Node> q = new LinkedList<>();
        if (root == null ) return root;
        q.offer(root);
        while(!q.isEmpty()){
            int level = q.size();
            for(int i =0; i< level; i++){
                Node cur = q.poll();
                if (cur.left != null && cur.right !=null) {
                    q.offer(cur.left);
                    q.offer(cur.right);
                }

                if (q.isEmpty() || i == level -1)
                    cur.next = null;
                else
                    cur.next = q.peek();
            }

        }
        return root;
    }
    }
```

OUTPUT:-



Testcase | >_ Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

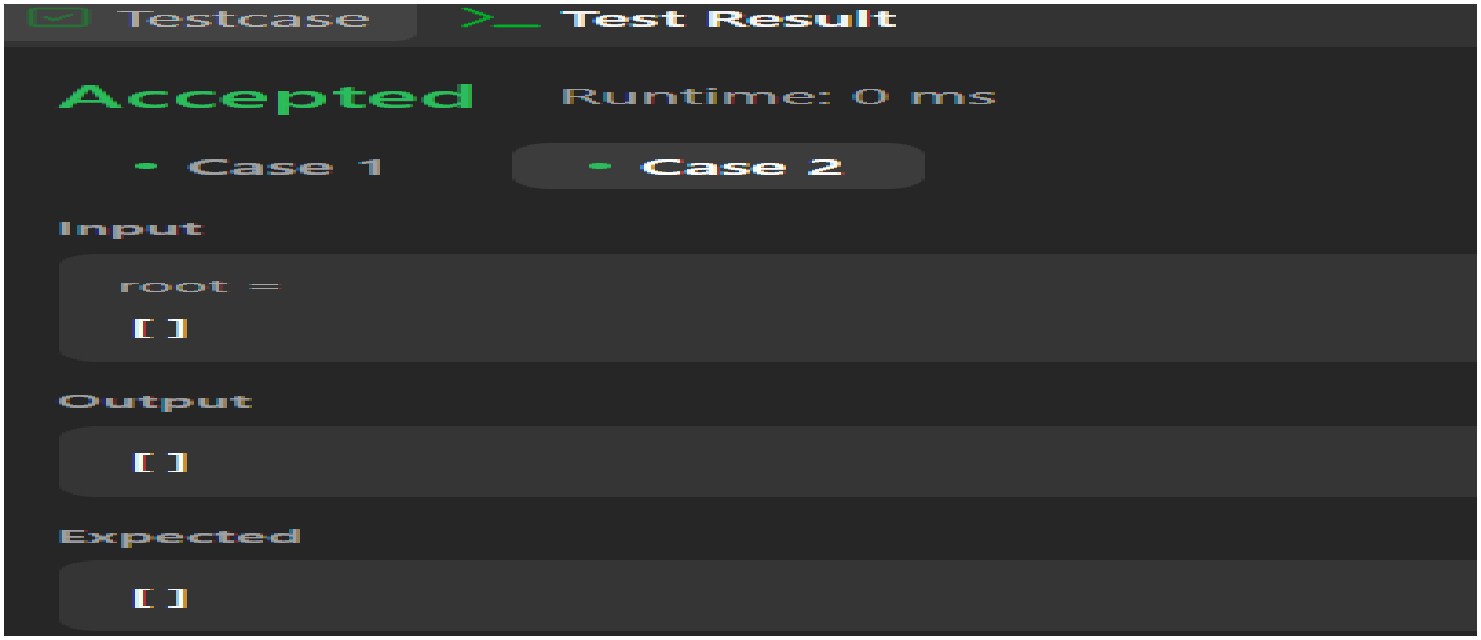
```
root =
[1,2,3,4,5,6,7]
```

Output

```
[1,#,2,3,#,4,5,6,7,#]
```

Expected

```
[1,#,2,3,#,4,5,6,7,#]
```



Testcase >_ Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

```
root =
[]
```

Output

```
[]
```

Expected

```
[]
```