**EXPERIMENT6**

**Student Name:** Piyush Prashar **UID:**22BET10096

**Branch:** BE -IT **Section/Group:**22BET\_IOT-703(A)

**Semester:** 6th **Subject Code:** 22ITP-351

**PROBLEM-1**

**AIM:-**

[Maximum Depth of Binary Tree](https://leetcode.com/problems/longest-nice-substring/)

**CODE:-**

class Solution {

public int maxDepth(TreeNode root) {

if (root == null) return 0;

int leftDepth = maxDepth(root.left);

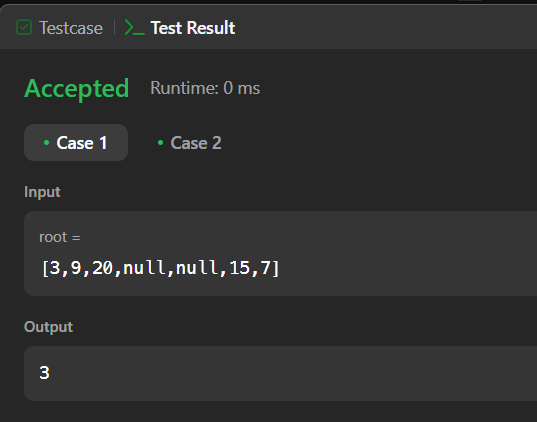
int rightDepth = maxDepth(root.right);

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

}

}

**OUTPUT:-**

****

**PROBLEM-2**

**AIM:-**

Validate Binary Search Tree

**CODE:**-

class Solution {

public boolean isValidBST(TreeNode root) {

return valid(root, Long.MIN\_VALUE, Long.MAX\_VALUE);

}

private boolean valid(TreeNode node, long minimum, long maximum) {

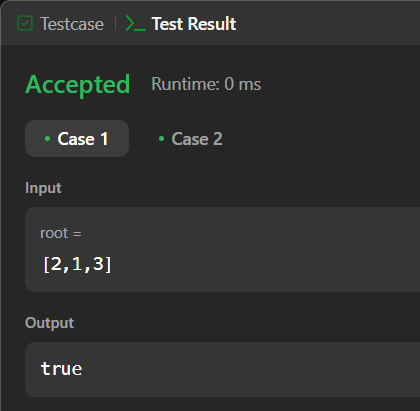
if (node == null) return true;

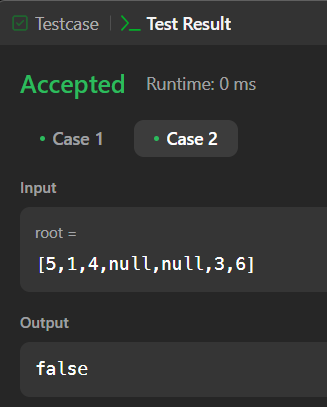
if (!(node.val > minimum && node.val < maximum)) return false;

return valid(node.left, minimum, node.val) && valid(node.right, node.val, maximum);

}

}**OUTPUT:-**

****

****

**PROBLEM-3**

**AIM:-**

Symmetric Tree

**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) {

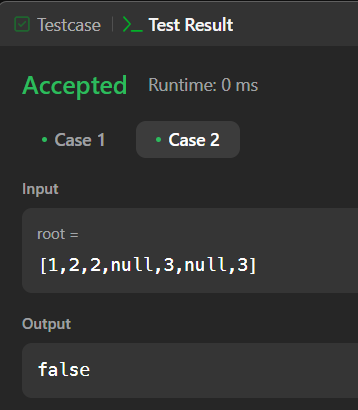
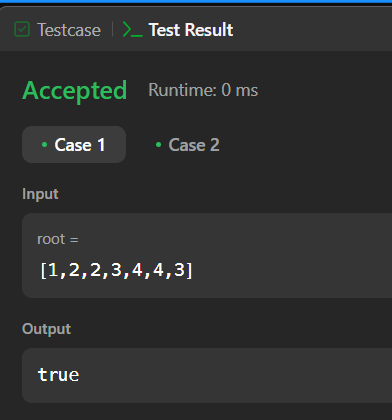
return false;

}

return node1.val == node2.val && isMirror(node1.left, node2.right) && isMirror(node1.right, node2.left);

}

} **OUTPUT:**

**-**

**PROBLEM-4**

**AIM:-**

Binary Tree Level Order Traversal

**CODE:-**

class Solution {

    public List<List<Integer>> levelOrder(TreeNode root)

    {

        List<List<Integer>>al=new ArrayList<>();

        pre(root,0,al);

        return al;

    }

    public static void pre(TreeNode root,int l,List<List<Integer>>al)

    {

        if(root==null)

            return;

        if(al.size()==l)

        {

            List<Integer>li=new ArrayList<>();

            li.add(root.val);

            al.add(li);

        }

        else

            al.get(l).add(root.val);

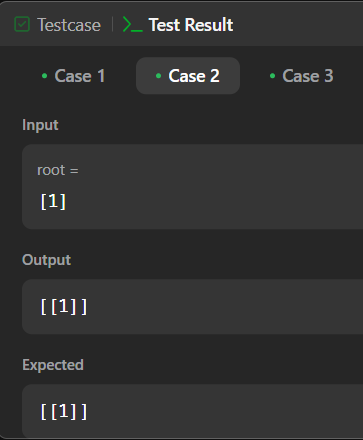
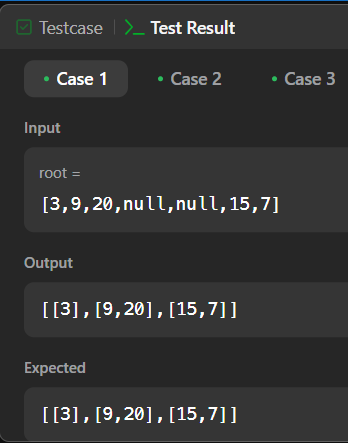
        pre(root.left,l+1,al);

        pre(root.right,l+1,al);

    }

}

**OUTPUT:-**

****

**PROBLEM-5**

**AIM:-**

Convert Sorted Array to Binary Search Tree

**CODE:-**

class Solution {

public TreeNode sortedArrayToBST(int[] nums) {

return constructBST(nums,0,nums.length-1);

}

public TreeNode constructBST(int[] nums,int left,int right){

if(left>right) return null;

int mid=left+(right-left)/2;

TreeNode root=new TreeNode(nums[mid]);

root.left=constructBST(nums,left,mid-1);

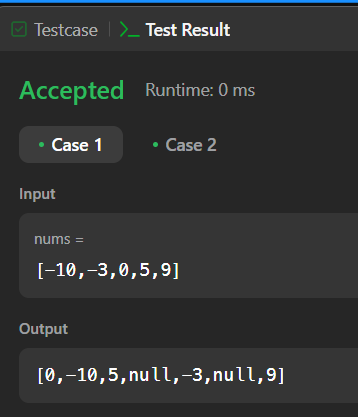
root.right=constructBST(nums,mid+1,right);

return root;

}

}

**OUTPUT:-**

****

**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);

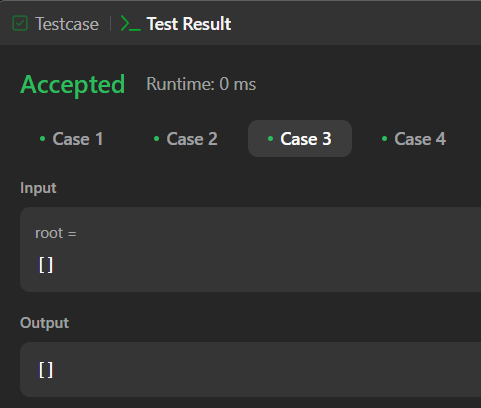
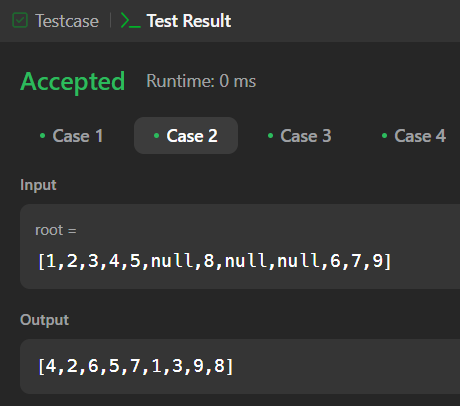
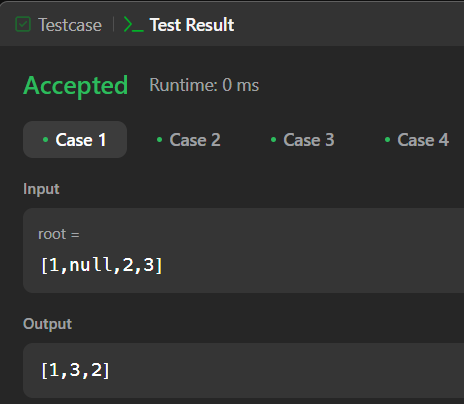
res.add(node.val);

inorder(node.right, res);

}

}

**OUTPUT:-**

****

**PROBLEM-7**

**AIM:-**

Binary Zigzag Level Order Traversal.

**CODE:-**

class Solution {

public List<List<Integer>> zigzagLevelOrder(TreeNode root) {

Queue<TreeNode> st = new LinkedList<>();

List<List<Integer>> ans = new ArrayList<>();

if(root!=null)st.add(root);

int k = 0;

while(!st.isEmpty()){

ArrayList<Integer> arr = new ArrayList<>();

int n = st.size();

// PUSHING TILL QUEUE.SIZE()

for(int i = 0 ; i<n ; i++){

TreeNode temp = st.poll();

if(temp==null)break;

arr.add(temp.val);

if(temp.left!=null)st.add(temp.left);

if(temp.right!=null)st.add(temp.right);

}

// REVERSING IF K IS ODD

if(k%2!=0){

Collections.reverse(arr);

ans.add(new ArrayList<>(arr));

}else ans.add(new ArrayList<>(arr));

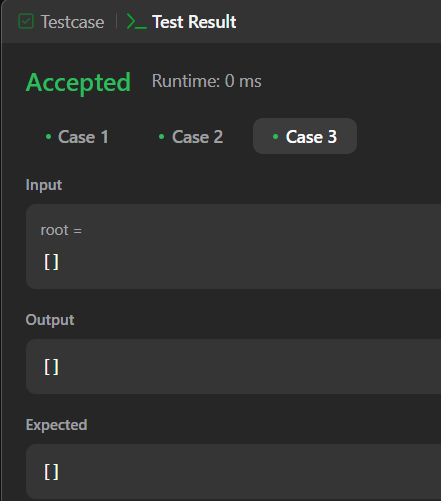
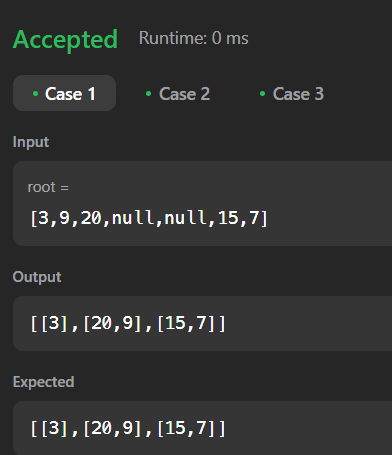
// INCREASING COUNTER

k++;

}return ans;

}

}

****

**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);

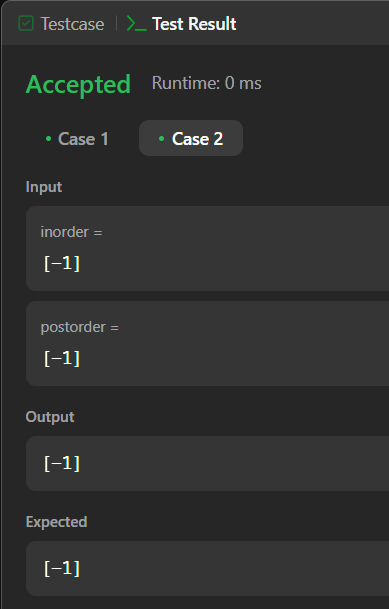
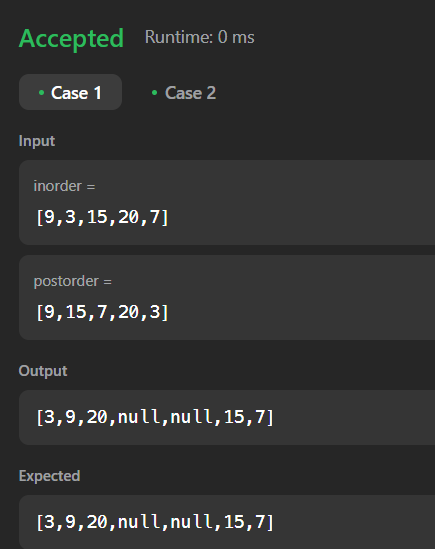
return root;

}

}

}

**OUTPUT:-**

****

**PROBLEM-9**

**AIM:-**

Kth Smallest element in a BST

**CODE:-**

class Solution {

int count = 0;

public int helper(TreeNode root, int k){

if(root == null)

return -1;

int left = helper(root.left, k);

if(left != -1)

return left;

count++;

if(count == k)

return root.val;

return helper(root.right, k);

}

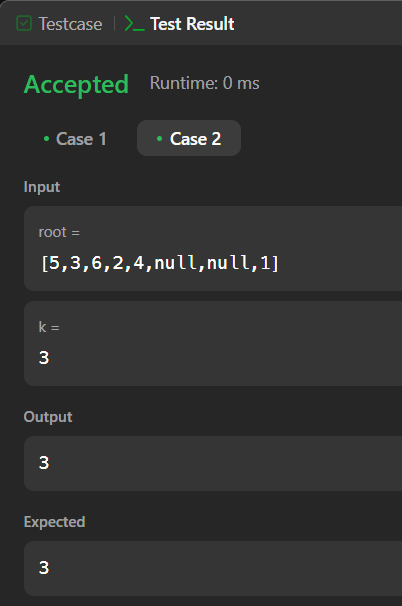
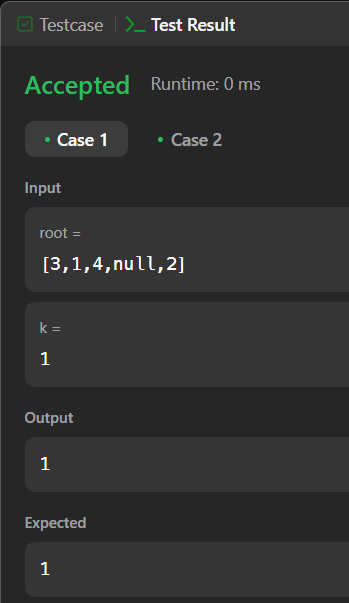
public int kthSmallest(TreeNode root, int k) {

return helper(root, k);

}

}

} **OUTPUT:-**

****

**PROBLEM-10**

**AIM:-**

Populating Next Right Pointers in Each Node

**CODE:-**

class Solution {

public Node connect(Node root) {

if (root == null) return null;

Queue<Node> q = new LinkedList<>();

q.offer(root);

while (!q.isEmpty()) {

int n = q.size();

for (int i = 0; i < n; i++) {

Node node = q.poll();

if (i != n - 1) {

node.next = q.peek();

}

if (node.left != null) q.offer(node.left);

if (node.right != null) q.offer(node.right);

}

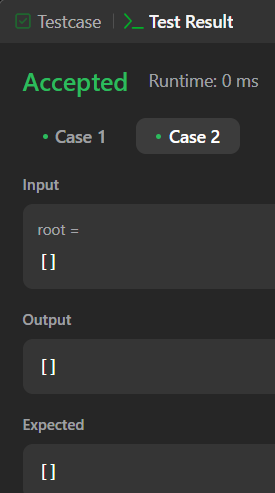
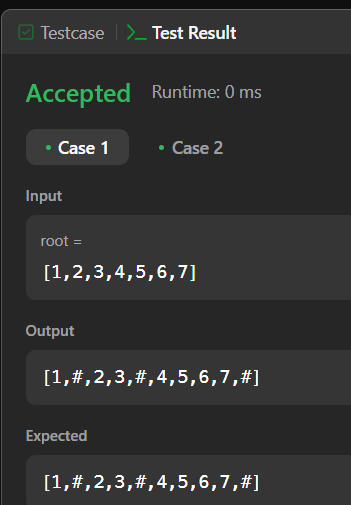
}

return root;

}

}

**OUTPUT:-**

****