Experiment 6

Student Name: Sikander Singh Nanglu **UID: 22BET10031**
Branch: BE-IT **Section/Group: 22BET-IOT-701/A**
Semester: 6 **Date of Performance: 28/02/2025**
Subject Name:AP2 **Subject Code: 22ITP-351**

1.Aim: Implement the following problem:- Maximum Depth of Binary Tree, Validate Binary Search Tree, Symmetric Tree, Binary Tree Level Order Traversal, Convert Sorted Array to Binary Search Tree, Binary Tree Inorder Traversal, Binary Zigzag Level Order Traversal, Construct Binary Tree from Inorder and Postorder Traversal, Kth Smallest element in a BST, Populating Next Right Pointers in Each Node

2.Objective: The objective is to build and manipulate binary trees using traversal methods, validation, and transformations. You will calculate tree depth, validate BST properties, perform level-wise/in-order traversals, convert a sorted array to a BST, populate next right pointers, and construct trees from traversal sequences. This enhances understanding of binary trees, BSTs, and their real-world applications.

3.Implementation/Code:

(A) Maximum Depth of Binary Tree

```
class Solution {
public:
int maxDepth(TreeNode* root) {
if (root == nullptr) return 0;
return 1 + max(maxDepth(root->left), maxDepth(root->right));
}};
```

(B)Validate binary Search tree

```
class Solution {
public:
TreeNode* prev = nullptr;
bool isValidBST(TreeNode* root) {
if (!root) return true;
if (!isValidBST(root->left)) return false;
if (prev && root->val <= prev->val) return false;
prev = root;
return isValidBST(root->right);
}
};
```

(C)Symmetric Tree

```
class Solution {
public:
bool isMirror(TreeNode* t1, TreeNode* t2) {
if (!t1 && !t2) return true;  // Both are NULL
if (!t1 || !t2) return false; // One is NULL, the other is not
return (t1->val == t2->val) &&
isMirror(t1->left, t2->right) &&
isMirror(t1->right, t2->left);
}
bool isSymmetric(TreeNode* root) {
if (!root) return true;
return isMirror(root->left, root->right);
}
};
```

(D) Binary Tree Level Order Traversal

```
 class Solution {
 public:
 vector<vector<int>> levelOrder(TreeNode* root) {
 vector<vector<int>> result;
 if (!root) return result;
queue<TreeNode*> q;
q.push(root);
while (!q.empty()) {
int size = q.size();
vector<int> level;
for (int i = 0; i < size; i++) {
TreeNode* node = q.front();
q.pop();
level.push_back(node->val);
if (node->left) q.push(node->left);
if (node->right) q.push(node->right);
}
result.push_back(level);
}
return result;
}
};
```

(E)Convert Sorted Array to Binary Search Tree

```
class Solution {
public:
TreeNode* sortedArrayToBST(vector<int>& nums) {
return helper(nums, 0, nums.size() - 1);
}
TreeNode* helper(vector<int>& nums, int left, int right) {
if (left > right) return nullptr;
int mid = left + (right - left) / 2;
TreeNode* root = new TreeNode(nums[mid]);
root->left = helper(nums, left, mid - 1);
root->right = helper(nums, mid + 1, right);
return root;
}
};
```

(F) Binary Tree Inorder Traversal

```
class Solution {
public:
void inorder(TreeNode* root, vector<int>& result) {
if (!root) return;
inorder(root->left, result);   // Visit left subtree
result.push_back(root->val);   // Visit root
inorder(root->right, result);  // Visit right subtree
}
vector<int> inorderTraversal(TreeNode* root) {
vector<int> result;
inorder(root, result);
return result;
}
};
```

(G) Binary Zigzag Level Order Traversal

```
class Solution {
public:
vector<vector<int>> zigzagLevelOrder(TreeNode* root) {
vector<vector<int>> result;
if (!root) return result;
queue<TreeNode*> q;
q.push(root);
bool leftToRight = true;
while (!q.empty()) {
```

```
int size = q.size();
vector<int> level(size);
for (int i = 0; i < size; i++) {
TreeNode* node = q.front();
q.pop();
int index = leftToRight ? i : (size - 1 - i);
level[index] = node->val;
if (node->left) q.push(node->left);
if (node->right) q.push(node->right);
}
result.push_back(level);
leftToRight = !leftToRight;
}
return result;
}
};
```

(H) Construct Binary Tree from Inorder and Postorder Traversal

```
class Solution {
public:
unordered_map<int, int> inorderMap;
int postIndex;
TreeNode* buildTreeHelper(vector<int>& inorder, vector<int>& postorder, int left, int right) {
if (left > right) return nullptr;
int rootVal = postorder[postIndex--];
TreeNode* root = new TreeNode(rootVal);
int inorderIndex = inorderMap[rootVal];
root->right = buildTreeHelper(inorder, postorder, inorderIndex + 1, right);
root->left = buildTreeHelper(inorder, postorder, left, inorderIndex - 1);
return root;
}
TreeNode* buildTree(vector<int>& inorder, vector<int>& postorder) {
postIndex = postorder.size() - 1;
for (int i = 0; i < inorder.size(); i++) {
inorderMap[inorder[i]] = i;
}
return buildTreeHelper(inorder, postorder, 0, inorder.size() - 1);
}
};
```

(I) Kth Smallest element in a BST

```
class Solution {
public:
```

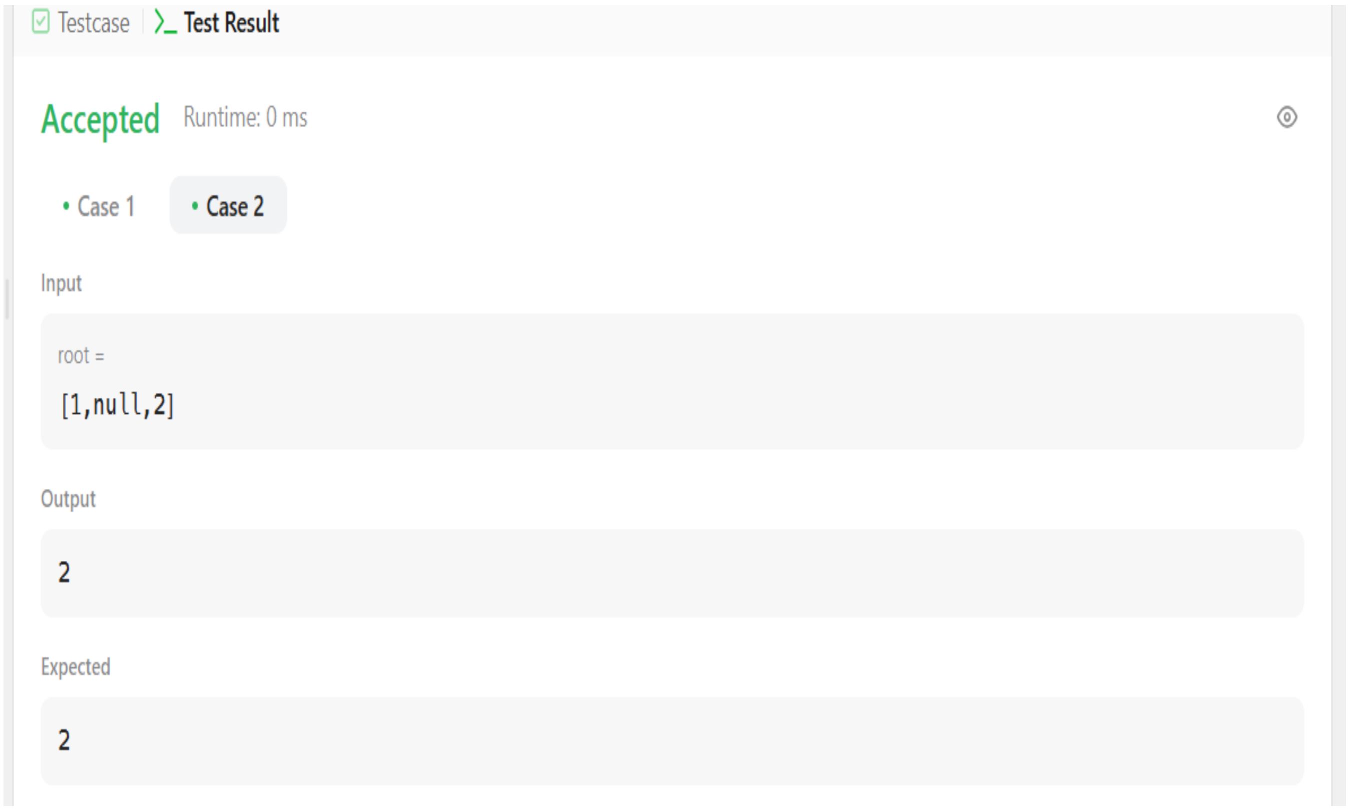
```
int kthSmallest(TreeNode* root, int k) {
int count = 0, result = 0;
inorder(root, k, count, result);
return result;
}
void inorder(TreeNode* root, int k, int& count, int& result) {
if (!root) return;
inorder(root->left, k, count, result);
count++;
if (count == k) {
result = root->val;
return;
}
inorder(root->right, k, count, result);
}
};
```

(J) Populating Next Right Pointers in Each Node

```
class Solution {
public:
Node* connect(Node* root) {
if (!root) return nullptr;
Node* leftmost = root;
while (leftmost->left) {
Node* current = leftmost;
while (current) {
current->left->next = current->right;
if (current->next) {
current->right->next = current->next->left;
}
current = current->next;
}
leftmost = leftmost->left;
}
return root;
}
};
```

4.Output:

(A) Maximum Depth of Binary Tree



Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

root =

```
[1,null,2]
```

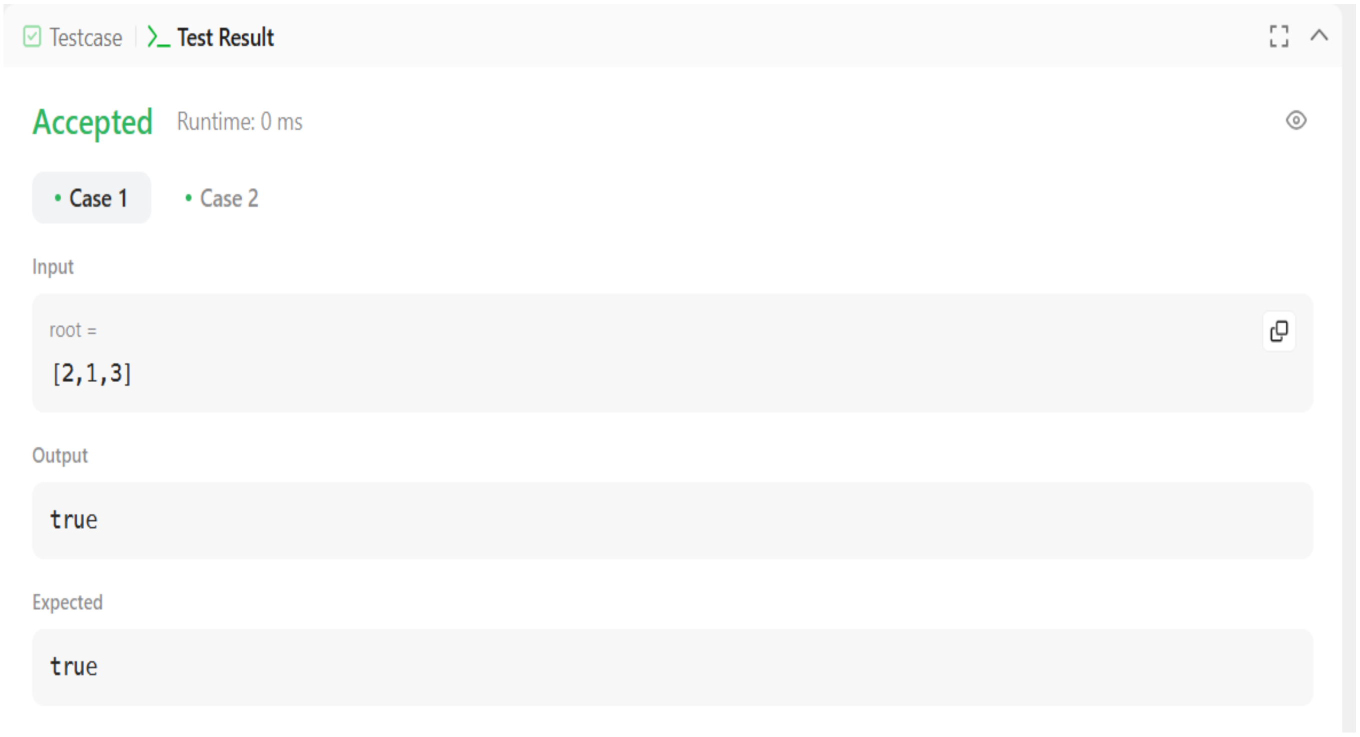
Output

```
2
```

Expected

```
2
```

(B)Validate binary Search tree



Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

root =

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

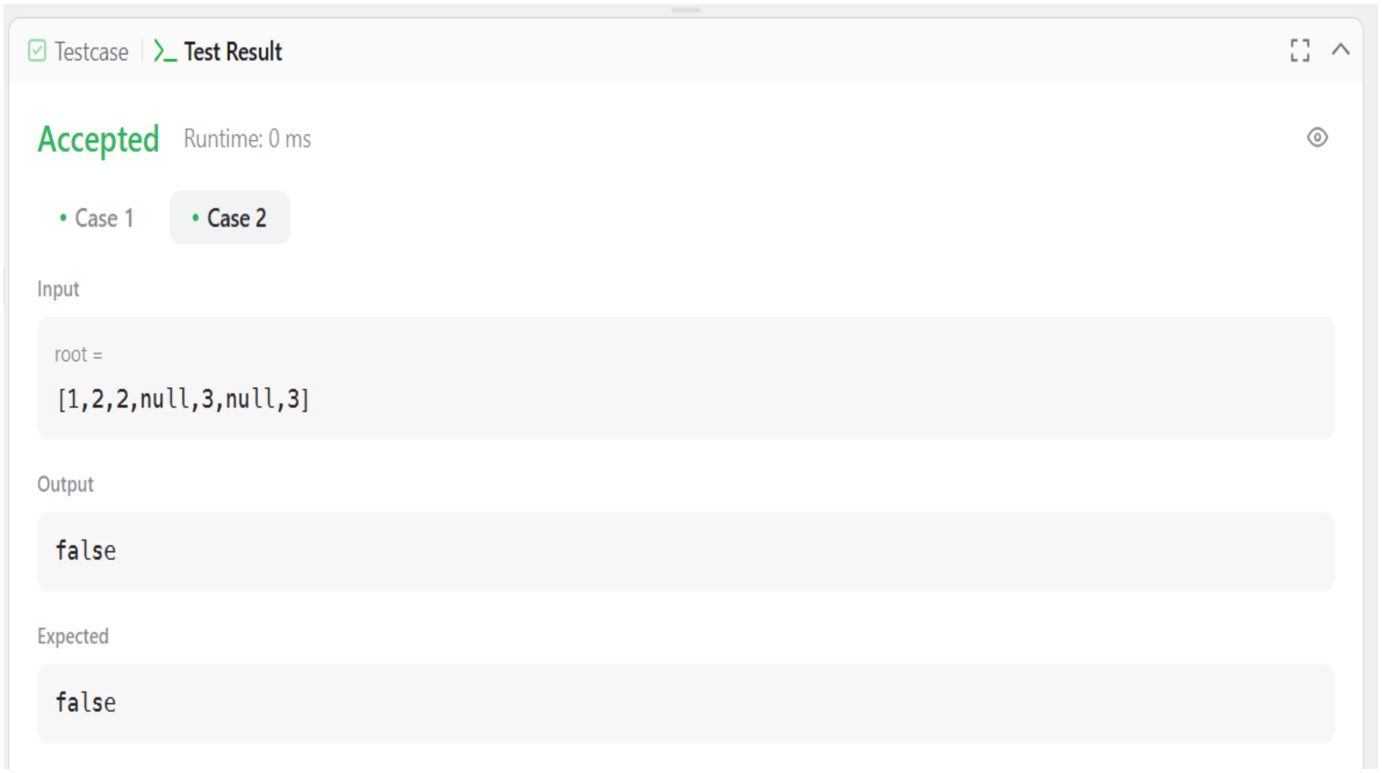
Output

```
true
```

Expected

```
true
```

(C)Symmetric Tree



Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

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

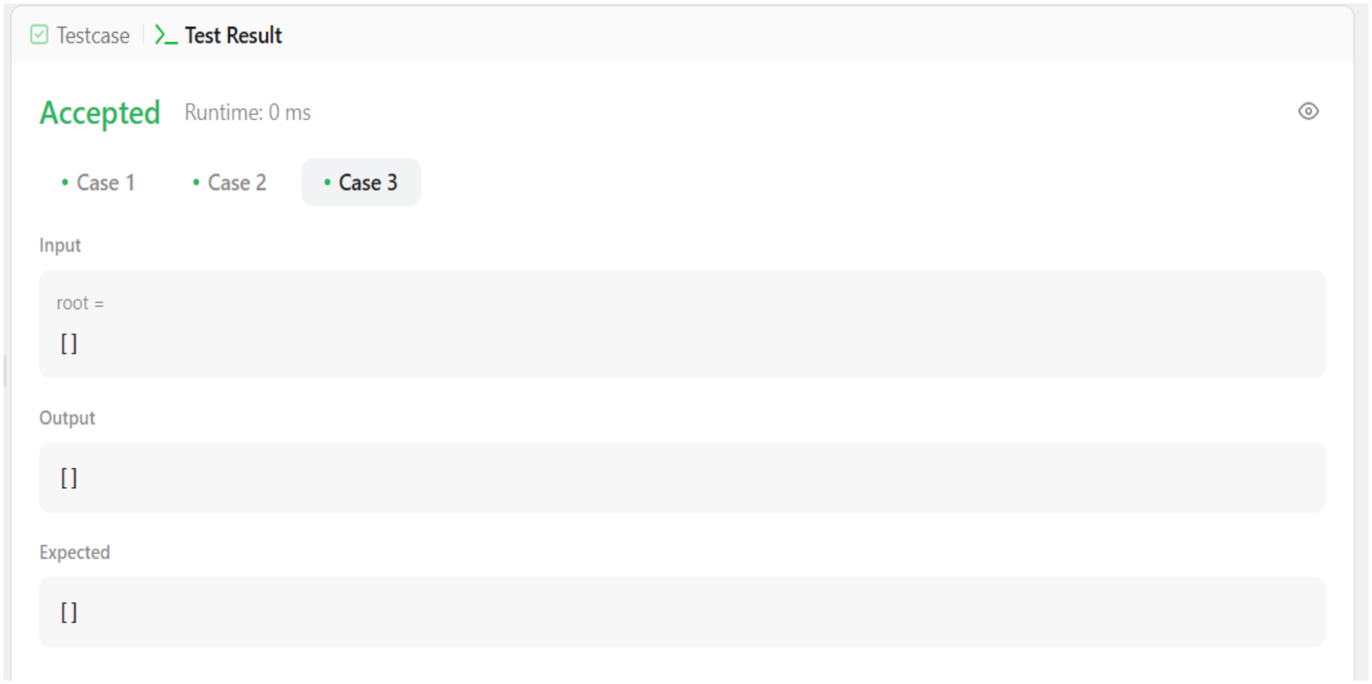
Output

```
false
```

Expected

```
false
```

(D) Binary Tree Level Order Traversal



Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3

Input

```
root =
[]
```

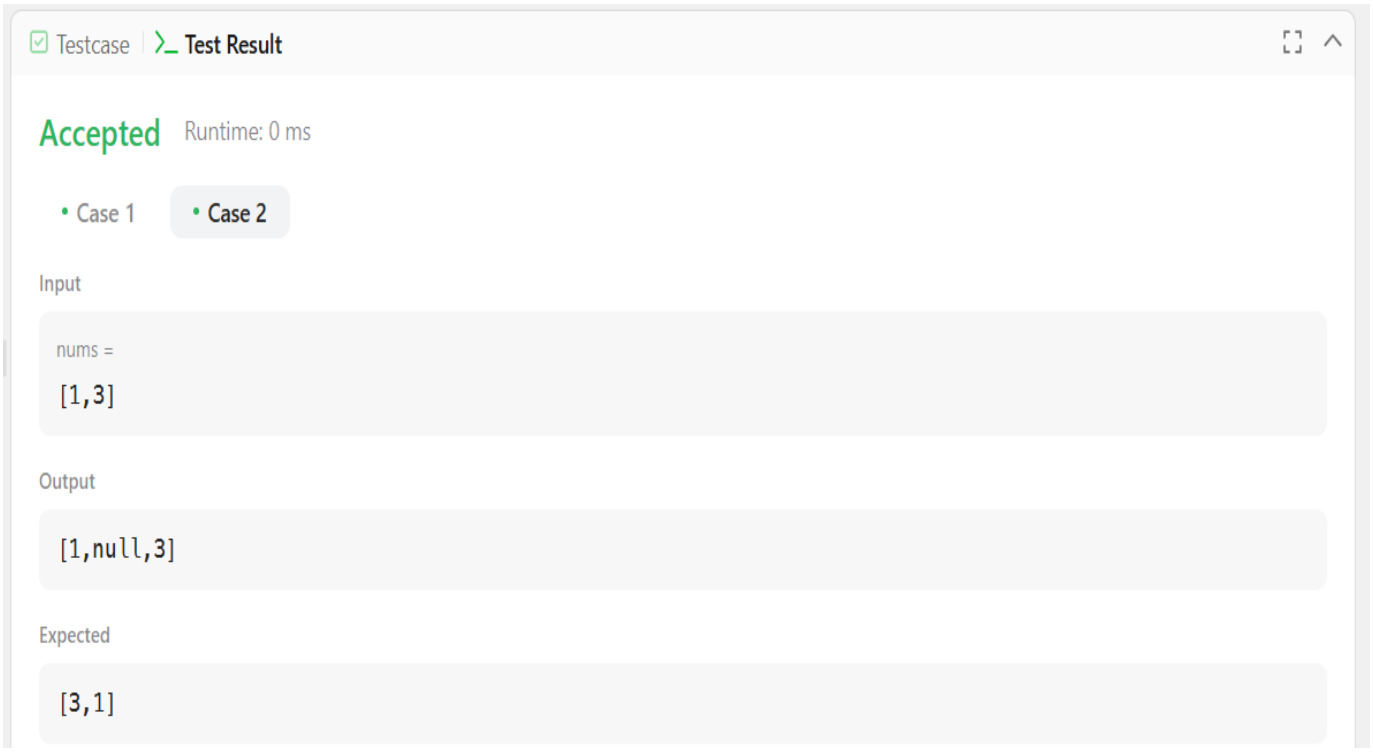
Output

```
[]
```

Expected

```
[]
```

(E)Convert Sorted Array to Binary Search Tree



Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

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

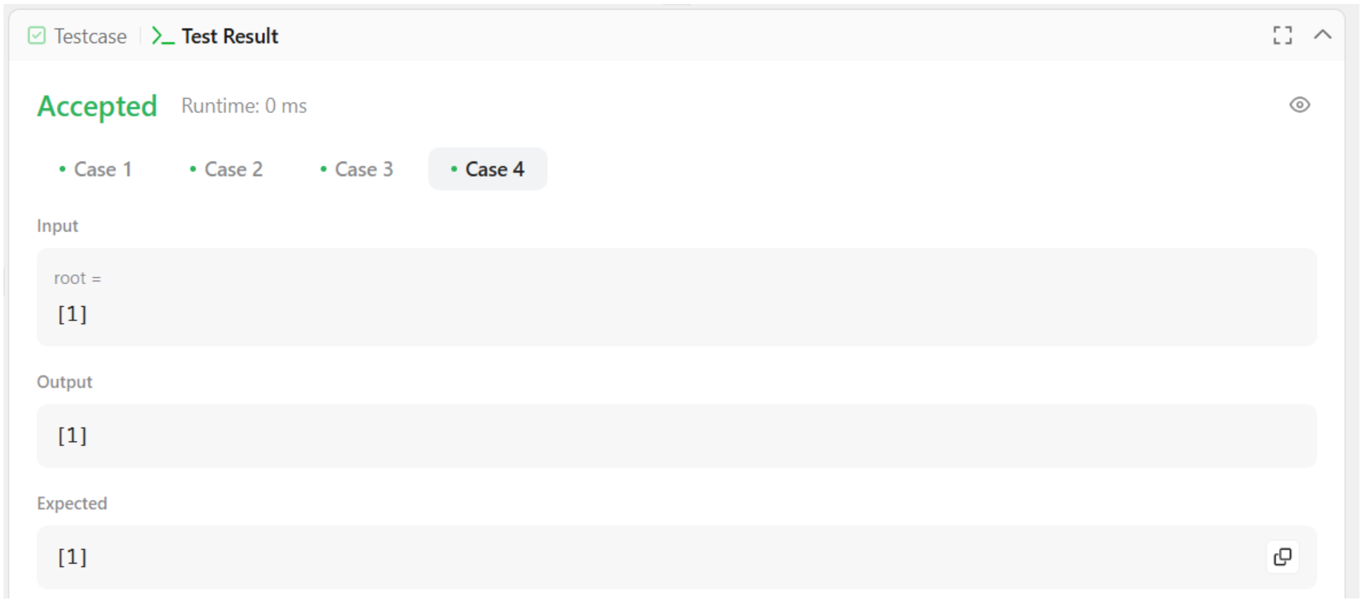
Output

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

Expected

```
[3,1]
```

(F) Binary Tree Inorder Traversal



Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2 • Case 3 • Case 4

Input

```
root =
[1]
```

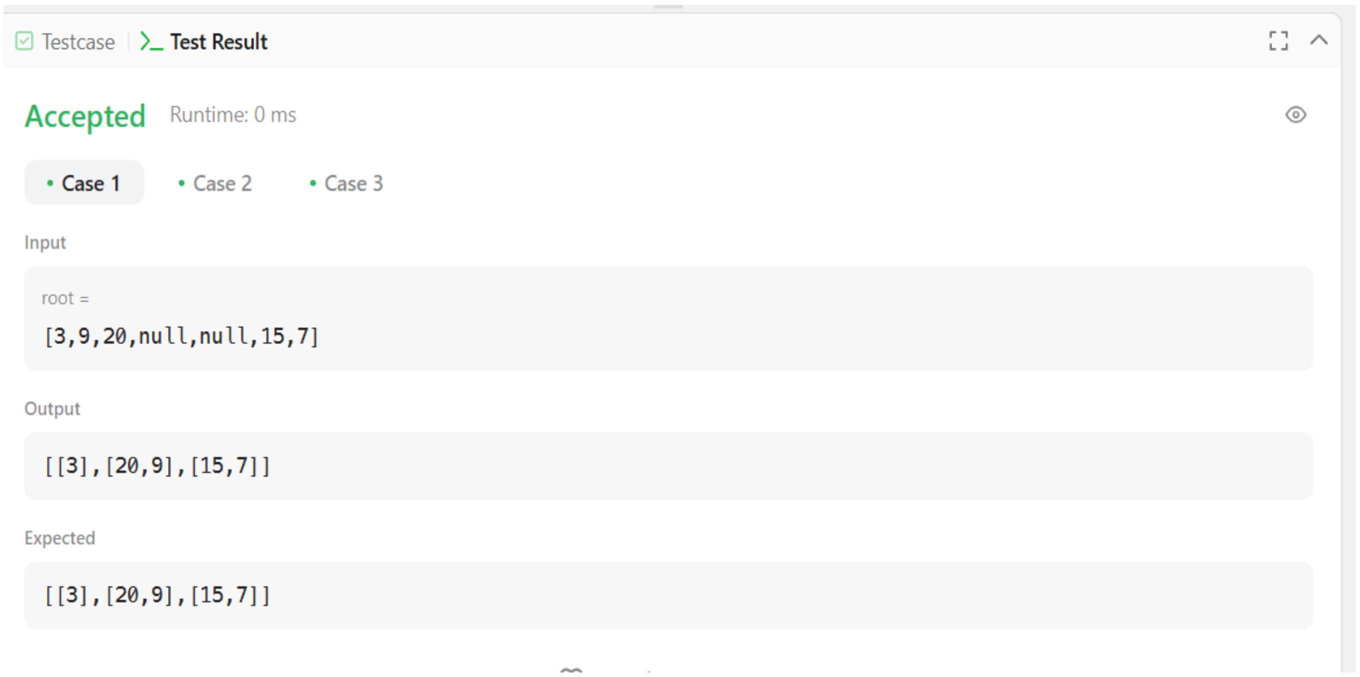
Output

```
[1]
```

Expected

```
[1]
```

(G) Binary Zigzag Level Order Traversal



Accepted Runtime: 0 ms

Input

root =

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

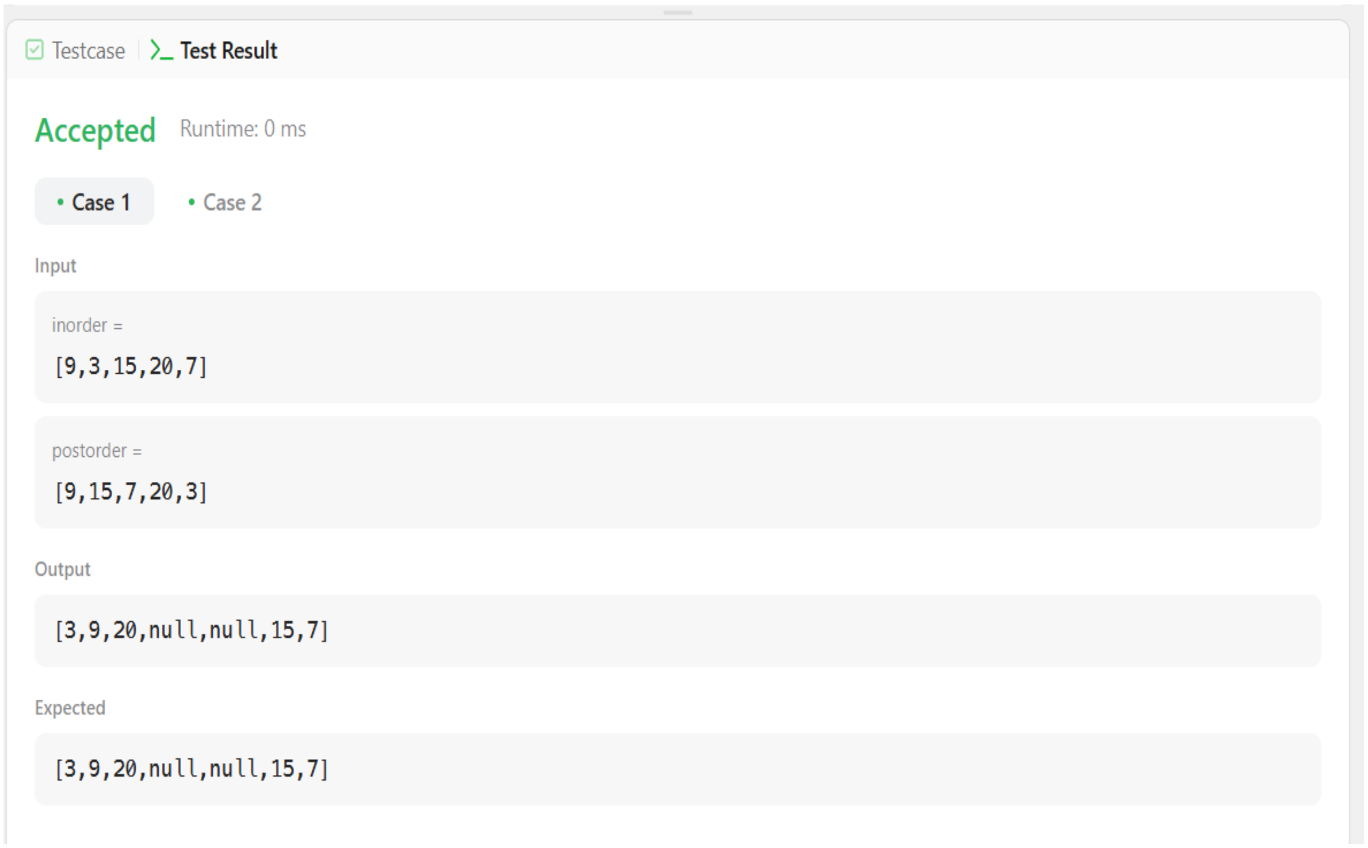
Output

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

Expected

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

(H) Construct Binary Tree from Inorder and Postorder Traversal



Accepted Runtime: 0 ms

Input

inorder =

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

postorder =

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

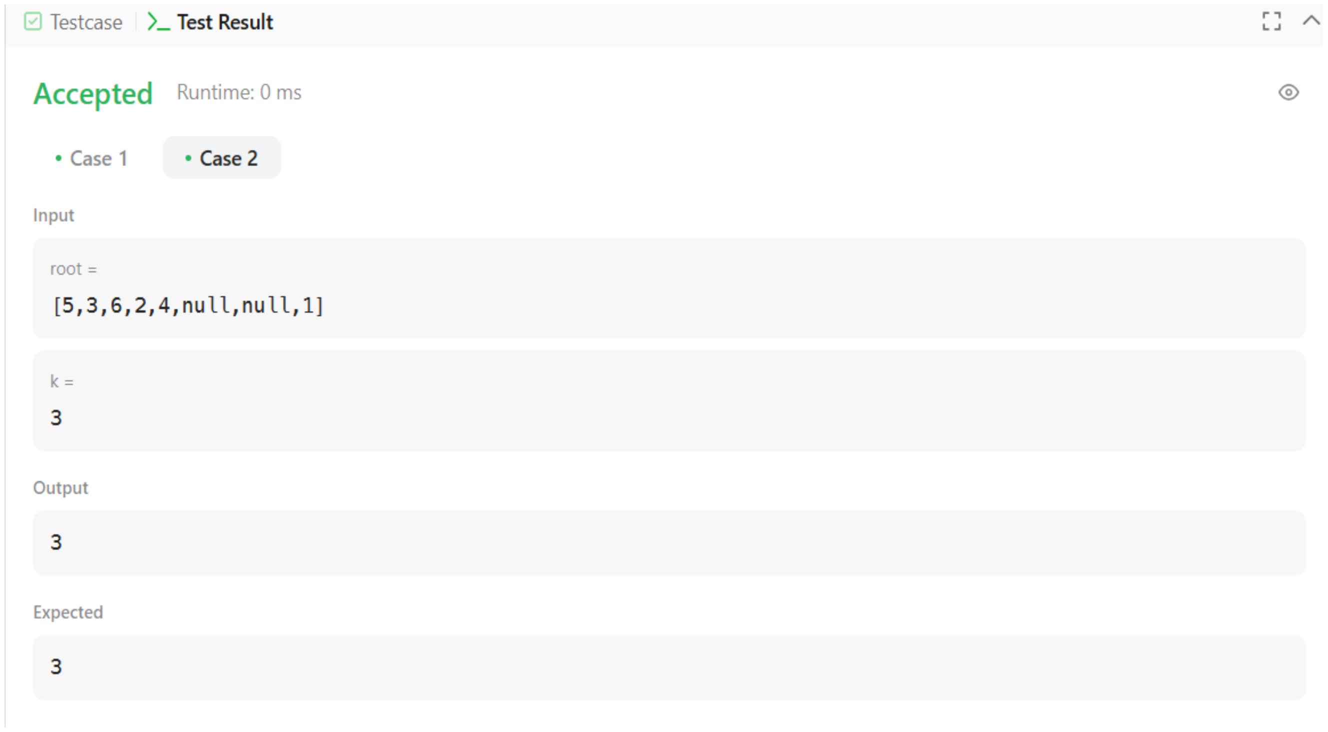
Output

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

Expected

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

(I) Kth Smallest element in a BST



Accepted Runtime: 0 ms

Input

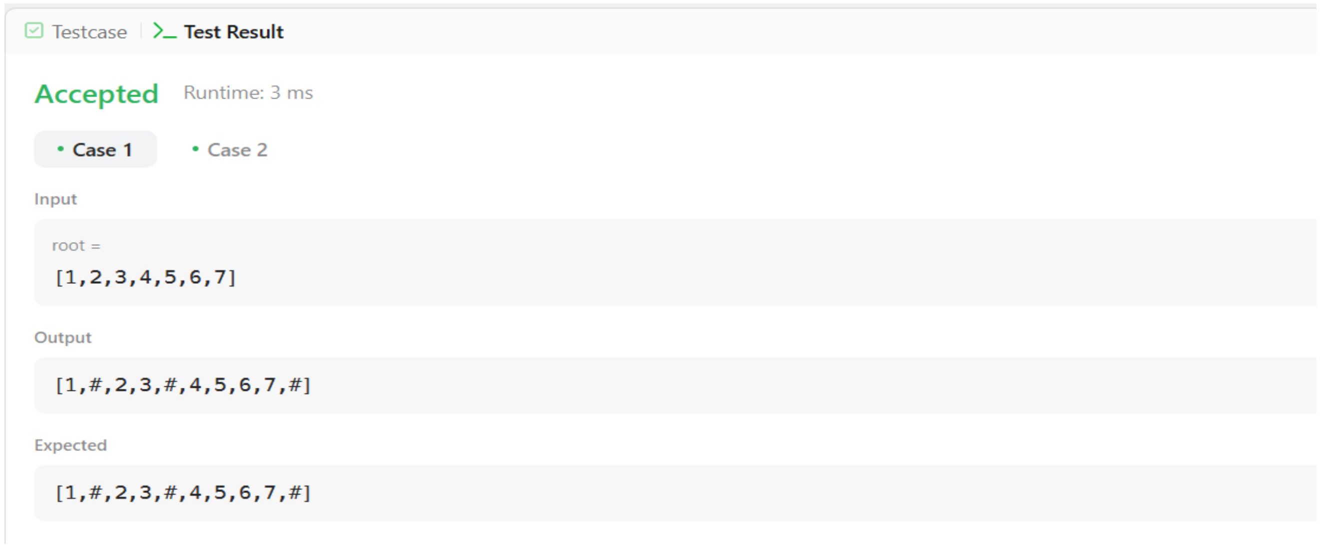
root =
```
[5,3,6,2,4,null,null,1]
```

k =
```
3
```

Output
```
3
```

Expected
```
3
```

(J) Populating Next Right Pointers in Each Node



Accepted Runtime: 3 ms

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,#]
```

5.Learning Outcomes:-

- Ability to analyze problems, evaluate information, and make logical decisions.
- Capability to identify, understand, and develop solutions to complex issues.
- Proficiency in expressing ideas clearly, both verbally and in writing
- Willingness to learn new skills and adjust to changing environments.
- Ability to work effectively with others in diverse environments.