EXPERIMENT-6

Student Name: Sanya Saxena **UID:**22BET10166

Branch: BE -IT **Section/Group:**22BET_IOT-703(A)

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

PROBLEM-1

AIM:-

Maximum Depth of Binary Tree

CODE:-

```
/**
 * Definition for a binary tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() : val(0), left(nullptr),
right(nullptr) {}
 *     TreeNode(int x) : val(x), left(nullptr),
right(nullptr) {}
 *     TreeNode(int x, TreeNode *left, TreeNode
*right) : val(x), left(left), right(right) {}
 * };
 */
class Solution {
public:
    int maxDepth(TreeNode* root) {
        if(root==nullptr){
            return 0;
        }
        int leftdepth=maxDepth(root->left);
        int rightdepth=maxDepth(root->right);

        return 1+max(leftdepth,rightdepth);
    }
};
```


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
root =
[3,9,20,null,null,15,7]
Output
3
Expected
3
Contribute a testcase

Testcase | Test Result

Accepted Runtime: 0 ms

• Case 1 • Case 2

Input

root =

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

Output

```
2
```

Expected

```
2
```

Contribute a testcase

PROBLEM-2

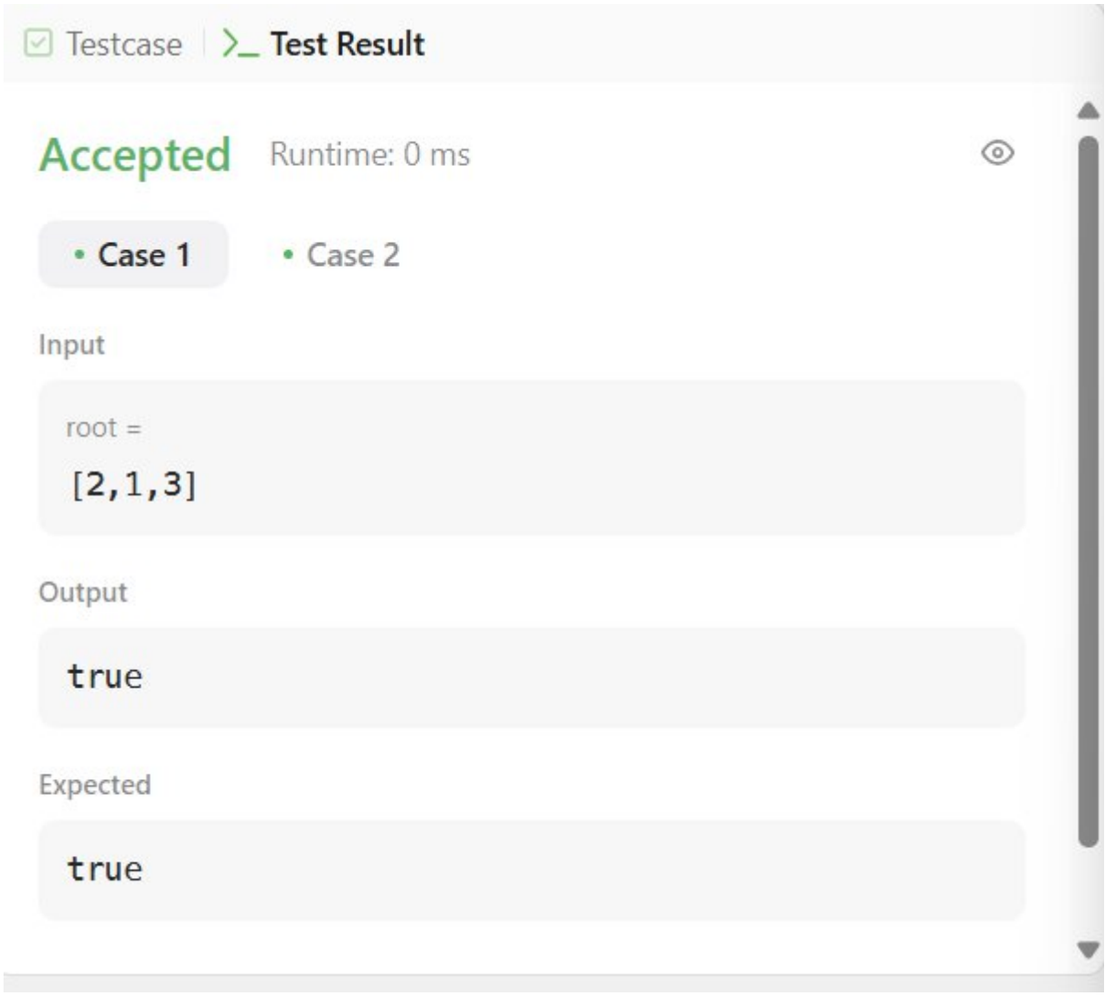
AIM:-

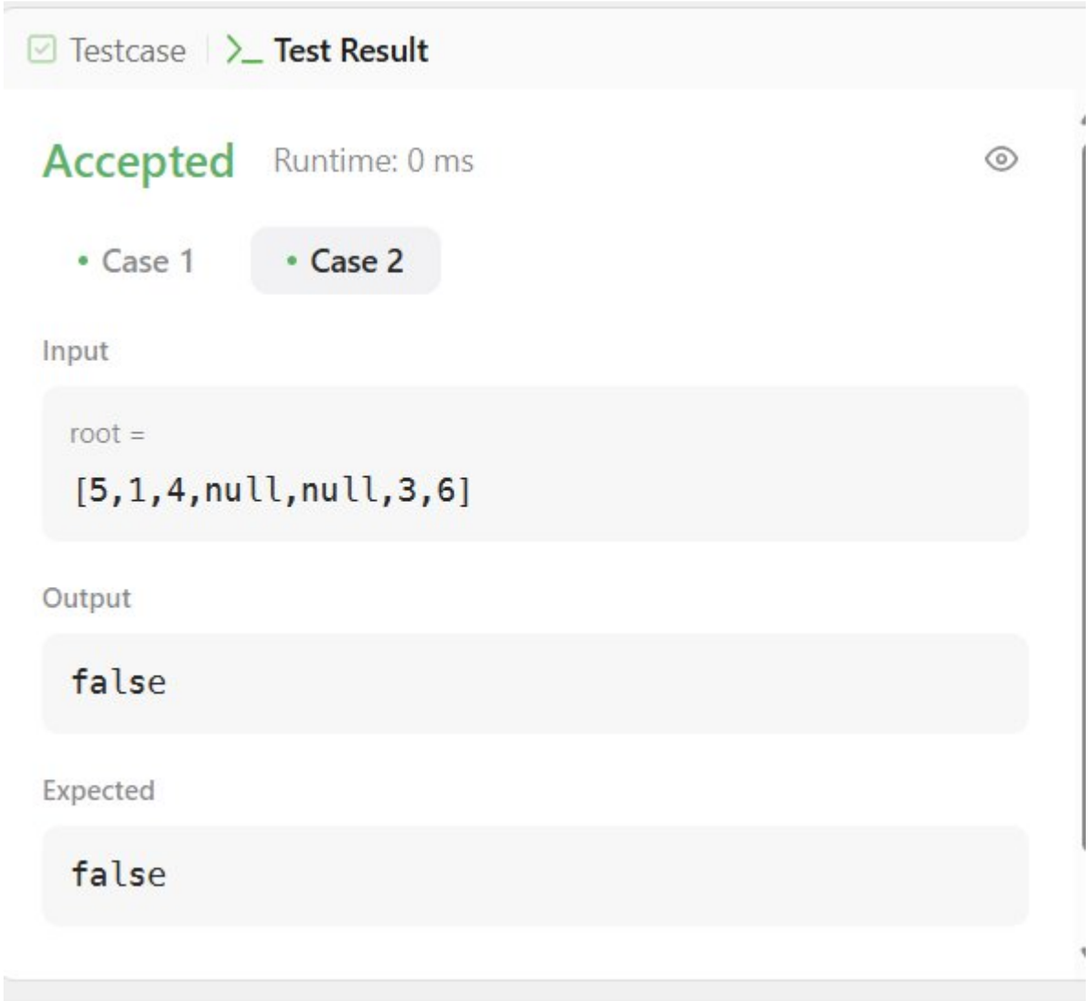
Validate Binary Search Tree

CODE:-

```
class Solution {


bool isPossible(TreeNode* root, long long l, long long r){
    if(root == nullptr)  return true;
    if(root->val < r and root->val > l)
        return isPossible(root->left, l, root->val) and
                            isPossible(root->right, root->val, r);
    else return false;
}


public:
    bool isValidBST(TreeNode* root) {
        long long int min = -1000000000000, max = 1000000000000;
        return isPossible(root, min, max);
    }
};
```

OUTPUT:-


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
root =
[2,1,3]
Output
true
Expected
true


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
root =
[5,1,4,null,null,3,6]
Output
false
Expected
false

PROBLEM-3

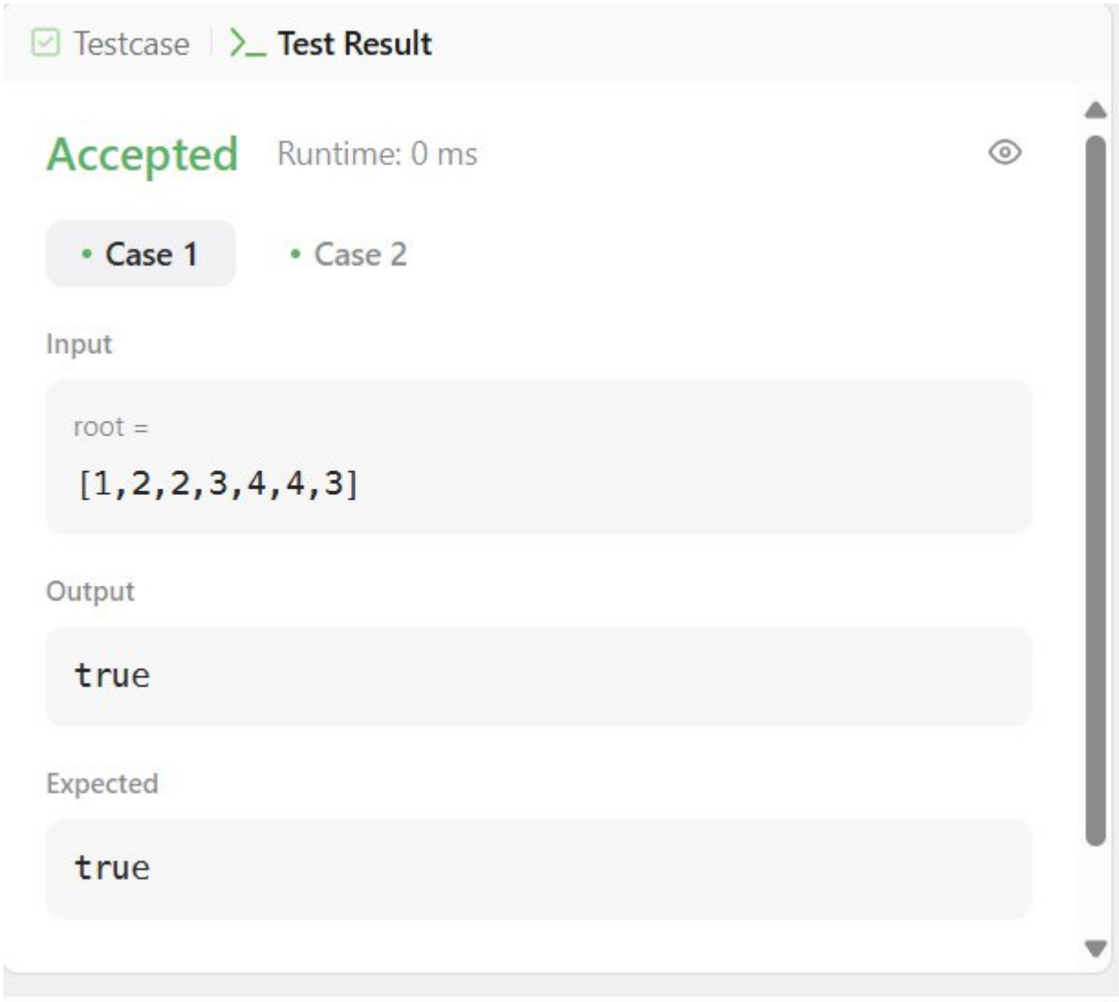
AIM:-

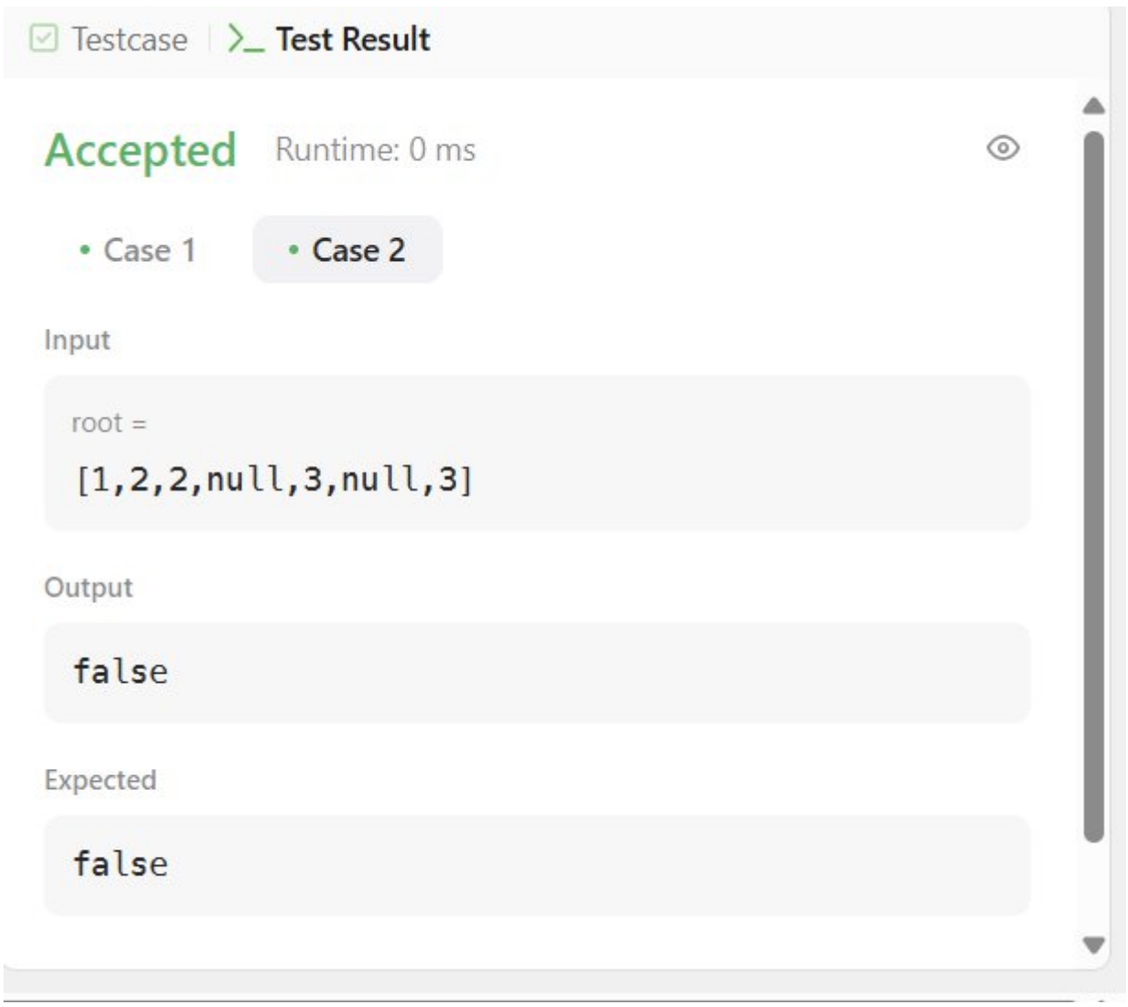
Symmetric Tree

CODE:-

```
class Solution {     public int
numDecodings(String s) {         if
(s.charAt(0) == '0') {             return 0;
        }

        int n = s.length();
int[] dp = new int[n + 1];
dp[0] = dp[1] = 1;


        for (int i = 2; i <= n; i++) {             int one =
Character.getNumericValue(s.charAt(i - 1));             int two
= Integer.parseInt(s.substring(i - 2, i));

            if (1 <= one && one <= 9) {
                dp[i] += dp[i - 1];
            }
            if (10 <= two && two <= 26) {
                dp[i] += dp[i - 2];
            }
        }

        return dp[n];
    }
}
```

OUTPUT:-


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
root =
[1,2,2,3,4,4,3]
Output
true
Expected
true


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Input
root =
[1,2,2,null,3,null,3]
Output
false
Expected
false

PROBLEM-4

AIM:-

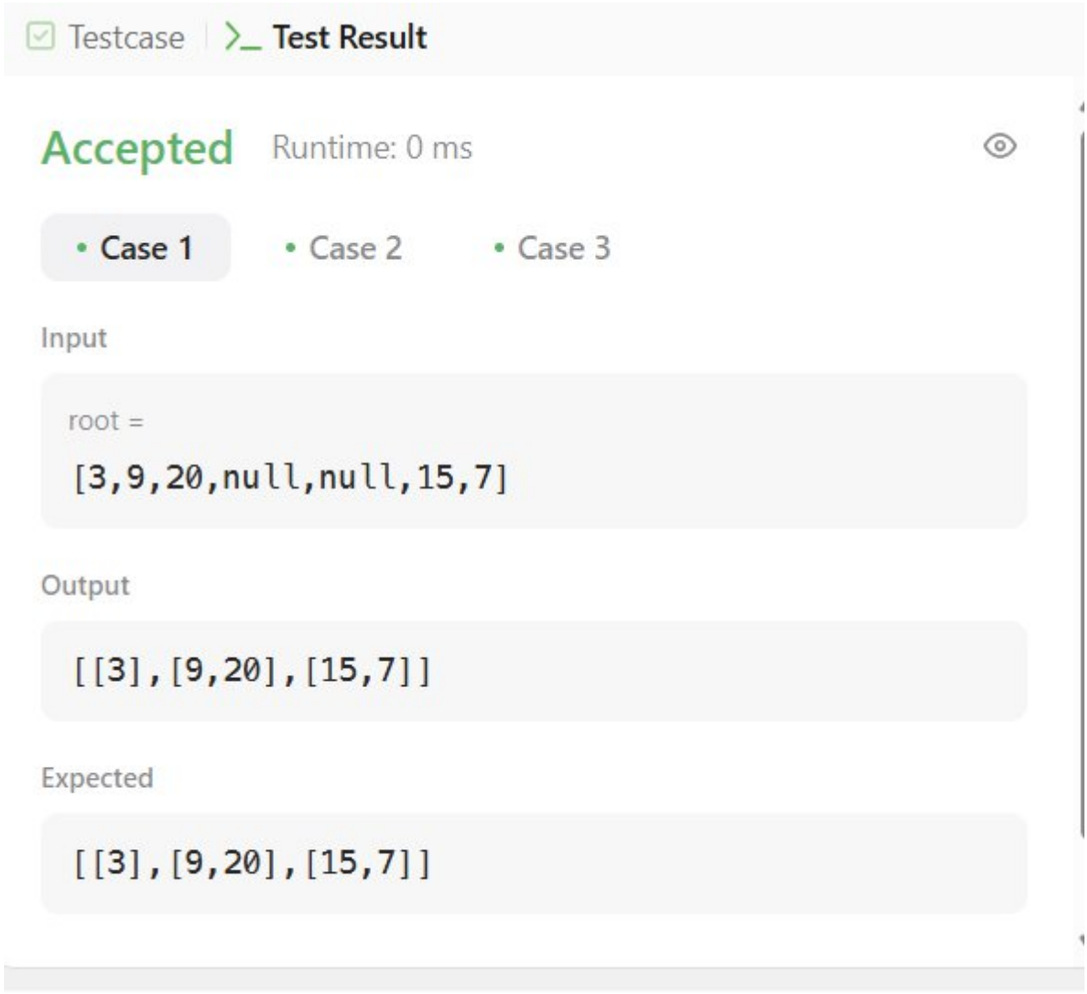
Binary Tree Level

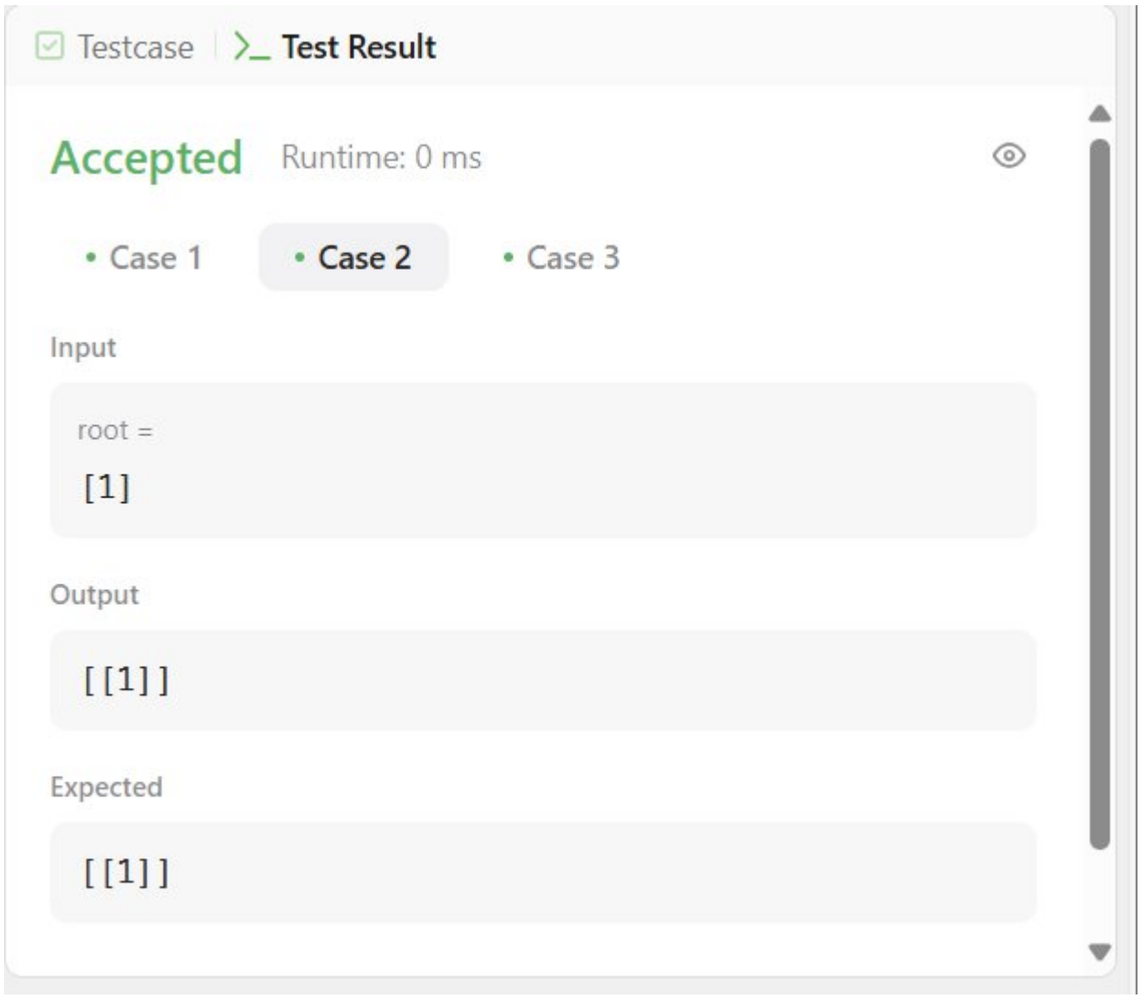
Order Traversal

CODE:-

```
class Solution {     public int coinChange(int[]
coins, int amount) {         int[] minCoins = new
int[amount + 1];         Arrays.fill(minCoins,
amount + 1);         minCoins[0] = 0;


       for (int i = 1; i <= amount; i++) {             for (int j = 0; j < coins.length;
j++) {                 if (i - coins[j] >= 0) {                     minCoins[i] =
Math.min(minCoins[i], 1 + minCoins[i - coins[j]]);
               }
           }
       }


       return minCoins[amount] != amount + 1 ? minCoins[amount] : -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],[9,20],[15,7]]
Expected
[[3],[9,20],[15,7]]


Testcase
Test Result
Accepted
Runtime: 0 ms
Case 1
Case 2
Case 3
Input
root =
[1]
Output
[[1]]
Expected
[[1]]

PROBLEM-5

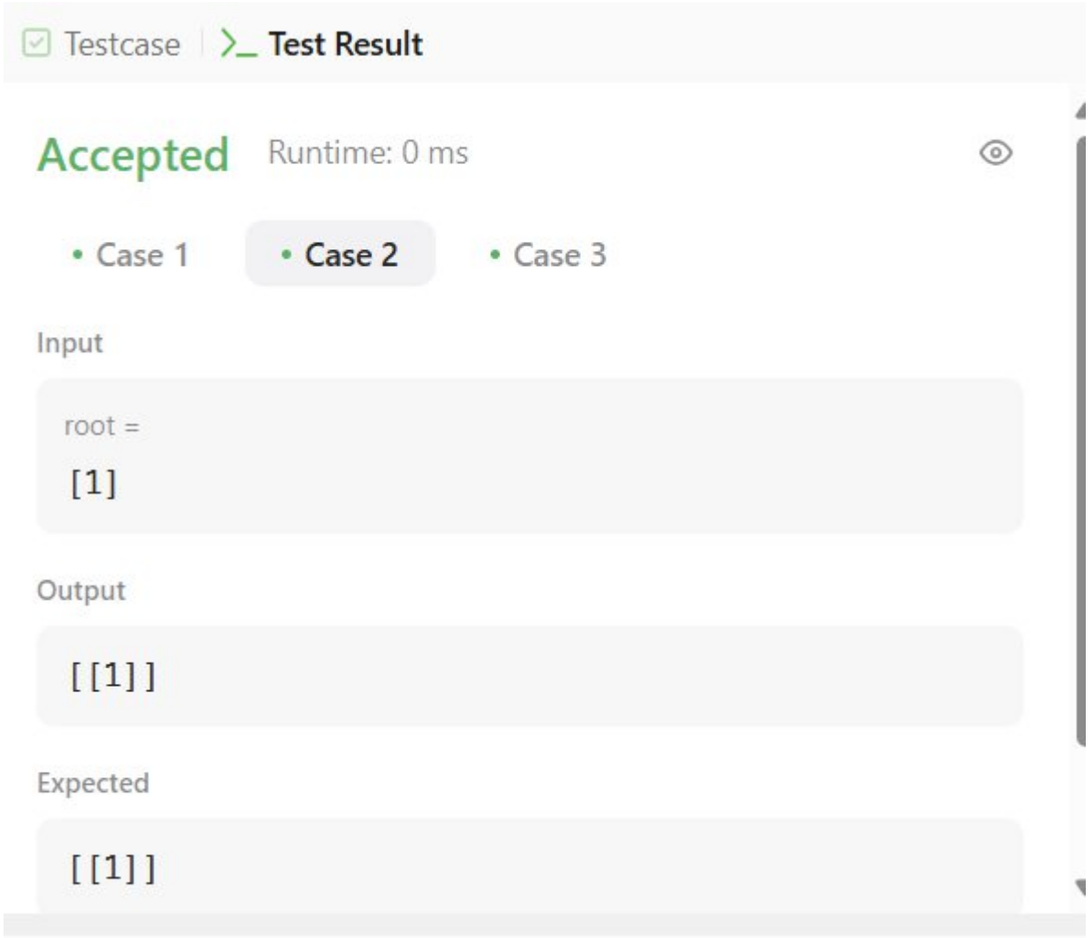
AIM:-

Convert Sorted Array to Binary Search

CODE:-

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

OUTPUT:



PROBLEM-6

AIM:-

Binary Tree

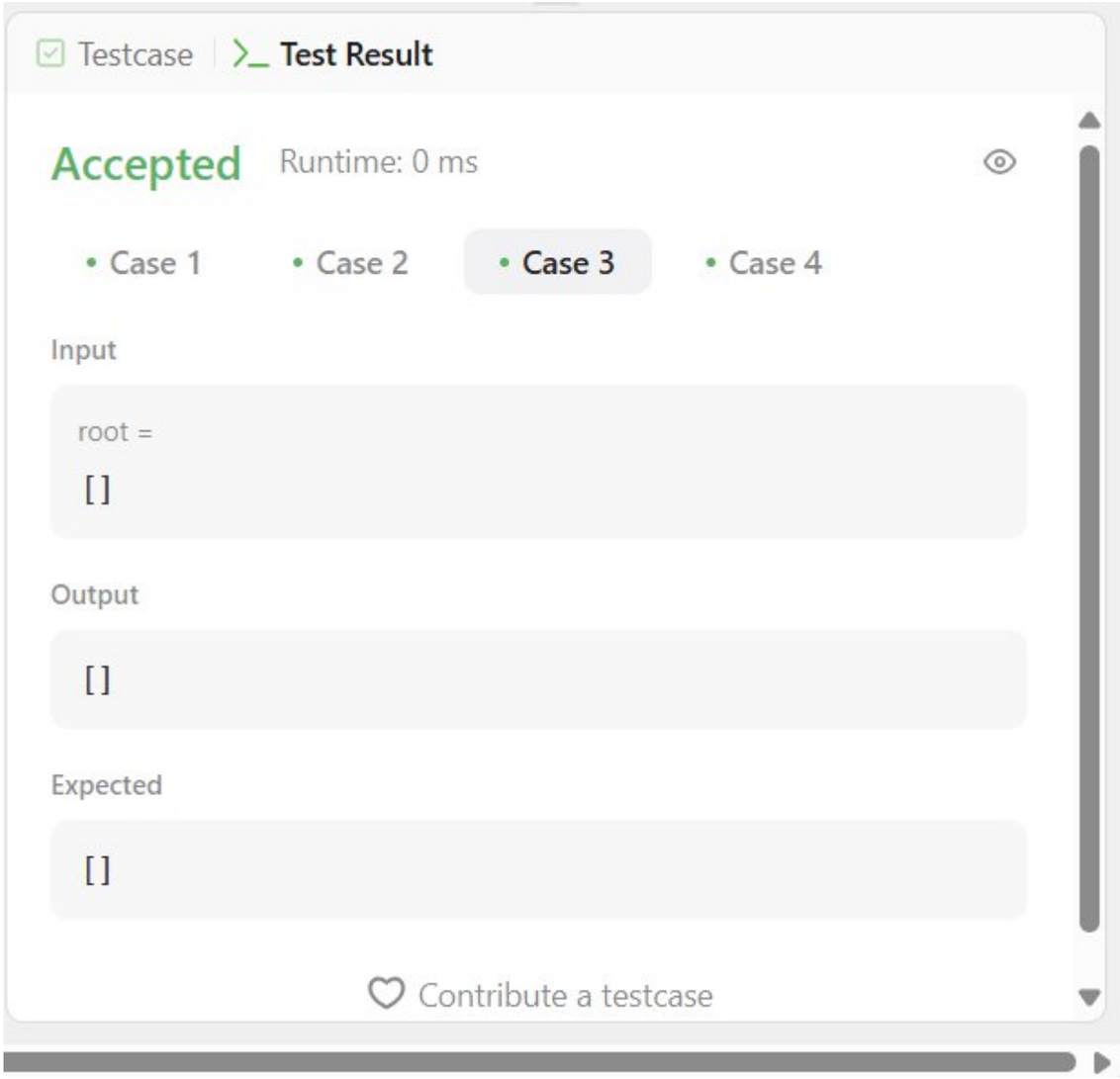
Inorder Traversal

CODE:-

```
/**
 * Definition for a binary
tree node.
 * struct TreeNode {
 *     int val;
 *     TreeNode *left;
 *     TreeNode *right;
 *     TreeNode() :
val(0), left(nullptr),
right(nullptr) {}
 *     TreeNode(int x) :
val(x), left(nullptr),
right(nullptr) {}
 *     TreeNode(int x,
TreeNode *left,
TreeNode *right) :
val(x), left(left),
right(right) {}
 * };
 */
class Solution {
public:
    void
inorder(TreeNode*
root,vector<int>&nums)
{
        if(root==NULL){
            return ;
        }

inorder(root
->left,nums);
```

```
nu
ms.p
ush_back
(root->val);


inorder(root
->right,nums);


    }
  vector<int>
inorderTraversal(TreeN
ode* root) {
      vector<int>nums;
      inorder(root,nums);
      return nums;
  }
};
```

OUTPUT:-



PROBLEM-7

AIM:-

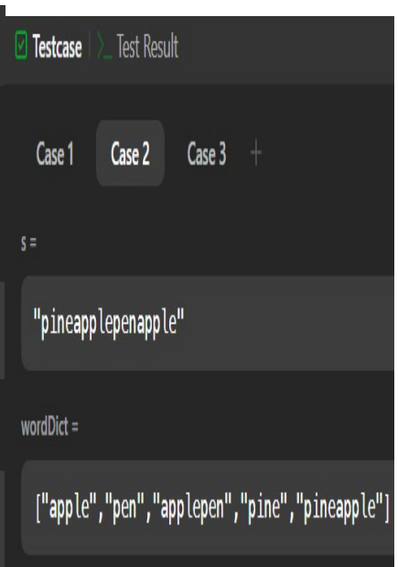
Binary Zigzag Level Order Traversal

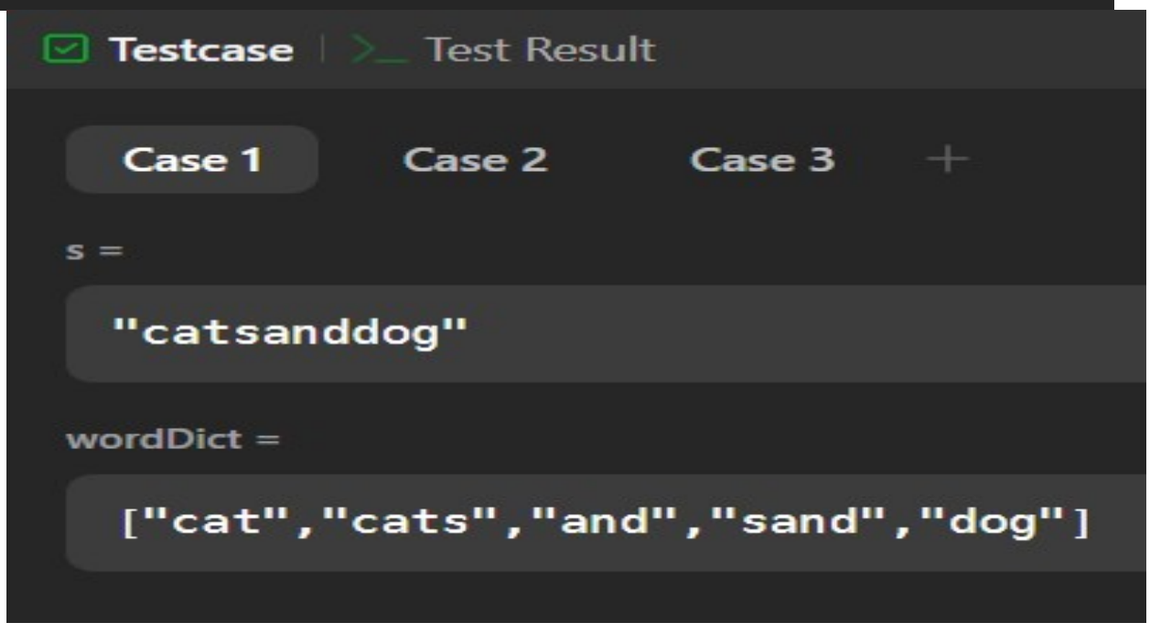
CODE:- import java.util.*;

```
class Solution {     public List<String> wordBreak(String s,
List<String> wordDict) {
        Set<String> wordSet = new HashSet<>(wordDict);
        Map<Integer, List<String>> memo = new HashMap<>();

        return backtrack(s, 0, wordSet, memo);
    }
```

```
    private List<String> backtrack(String s, int start, Set<String> wordSet, Map<Integer,
List<String>> memo) {         if (memo.containsKey(start)) {              return
memo.get(start);
        }

        List<String> result = new ArrayList<>();

        if (start == s.length())
{             result.add("");
return result;
        }

        for (int end = start + 1; end <= s.length(); end++) {
            String word = s.substring(start, end);

            if (wordSet.contains(word)) {
                List<String> sublist = backtrack(s, end, wordSet, memo);
for (String sub : sublist) {                    if (sub.isEmpty())
{                         result.add(word);
                    } else {
                        result.add(word + " " + sub);
                    }
                }
            }
        }
                memo.put(start, result);
return result;
    }
}
```


Testcase
Test Result
Case 1
Case 2
Case 3
s =
"catsandog"
wordDict =
["cats","dog","sand","and","cat"]


Testcase
Test Result
Case 1
Case 2
Case 3
s =
"pineapplepenapple"
wordDict =
["apple","pen","applepen","pine","pineapple"]


Testcase
Test Result
Case 1
Case 2
Case 3
s =
"catsanddog"
wordDict =
["cat","cats","and","sand","dog"]