Experiment 6

StudentName: Bhuvanendra Sai **UID:**22BCS10786

Branch:BE-CSE **Section/Group:**22BCS-IOT-640(B)

Semester:6th **DateofPerformance:**10/03/25

SubjectName:ADVANCED PROGRAMMING LAB - 2 **SubjectCode:**22CSP-351

PROGRAM-1

1) **Aim:**MaximumDepth ofBinary Tree.

2) **Objective:** The objective of this code is to determine the maximum depth (or height) of a binary tree. Thedepthisdefinedasthenumberofnodesalongthelongestpathfromtherootnodetoaleafnode.The code uses a recursive approach to traverse the tree and compute the depth.

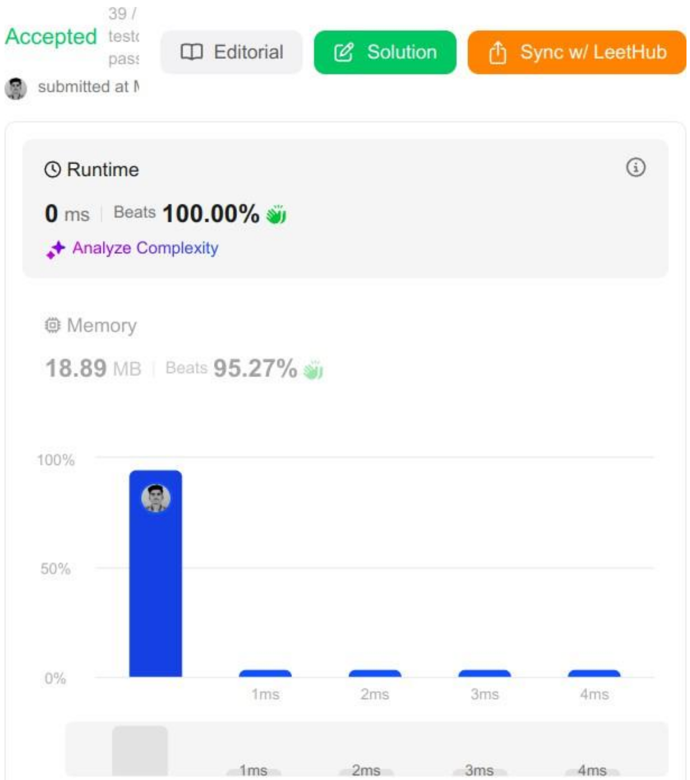
3) **Implementation/Code:**

```
#include <iostream>
using namespace std;

//structTreeNode{
//    int val;
//    TreeNode*left;
//    TreeNode*right;
//    TreeNode(intx):val(x),left(NULL),right(NULL){}
// };

classSolution{ public:
   intmaxDepth(TreeNode*root){ if
      (!root) return 0;
      int left_depth = maxDepth(root->left);
      intright_depth=maxDepth(root->right);
      return max(left_depth, right_depth) + 1;
   }
};
```

4) Output:



5) LearningOutcomes:

- UnderstandingRecursioninTree Traversals.
- BinaryTreeDepthCalculation.
- Use of Base Case in Recursion.
- Comparison of Left and Right Subtrees.
- Implementation of a Recursive Function in C++.

PROGRAM-2

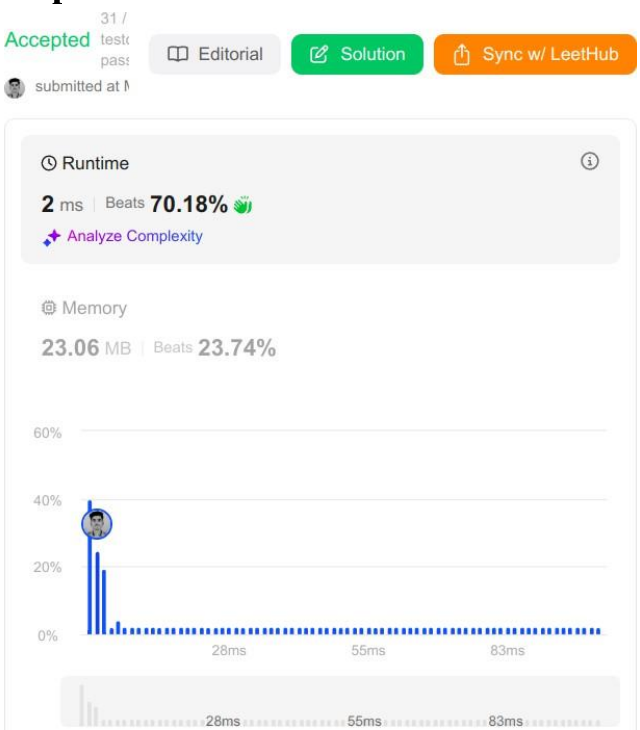
1) Aim:Convert SortedArray toBinary Search Tree.

2) Objective: The objective of this code is to convert a sorted array into a height-balanced Binary Search Tree (BST). A height-balanced BST is a binary tree where the depth of the left and right subtrees of every nodedifferbyatmostone.ThealgorithmusesarecursiveapproachtoconstructtheBSTbyselectingthe middle element of the array as the root, ensuring balanced tree formation.

3) Implementation/Code:

```
/**
 * Definition for a binary tree node.
 * structTreeNode{
 *     int val;
 *     TreeNode*left;
 *     TreeNode*right;
 *     TreeNode():val(0),left(nullptr),right(nullptr){}
 *     TreeNode(intx):val(x),left(nullptr),right(nullptr){}
 *     TreeNode(intx,TreeNode*left,TreeNode*right):val(x),left(left),right(right){}
 * };
 */
classSolution{ public:
    TreeNode*sortedArrayToBST(vector<int>&nums){ return
        buildBST(nums, 0, nums.size() - 1);
    }
private:
    TreeNode*buildBST(vector<int>&nums,intleft,intright){ if
        (left > right) return nullptr;
        int mid = left + (right - left) / 2;
        TreeNode*root=newTreeNode(nums[mid]);
        root->left = buildBST(nums, left, mid - 1);
        root->right=buildBST(nums,mid+1,right);
        return root;
    }
};
```

4) Output:



5) LearningOutcomes:

- UnderstandingBinarySearchTree(BST)Construction.
- Recursive Divide-and-Conquer Approach.
- Optimal Root Selection for Balanced Trees.
- EfficientTreeConstructionUsingO(n)TimeComplexity.
- PracticalImplementationofTreeNodeClassinC++.

PROGRAM-3

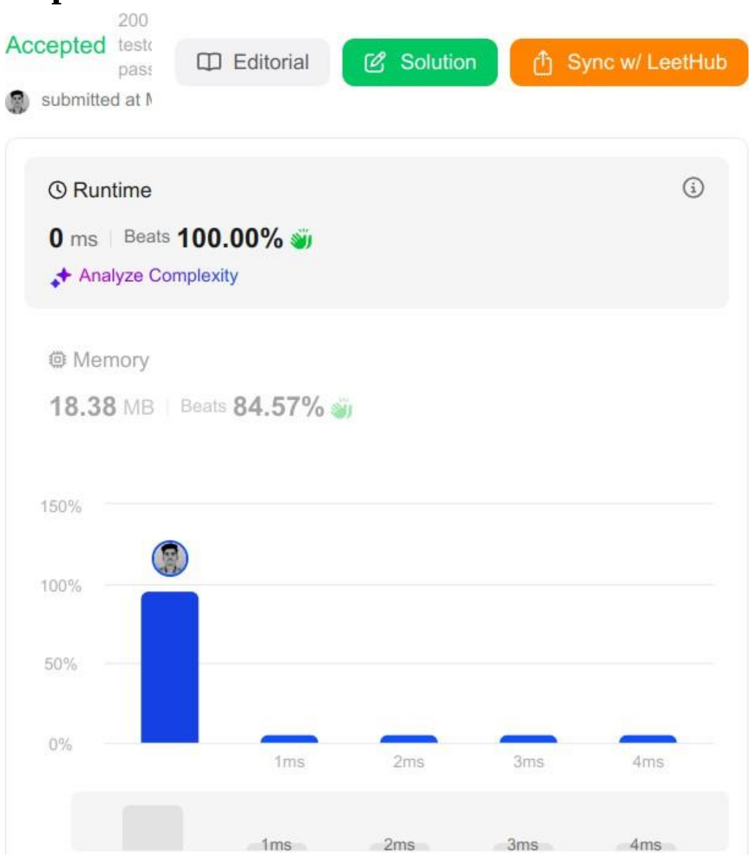
1) Aim: SymmetricTree.

2) Objective:
Theobjectiveofthiscodeistodeterminewhetheragivenbinarytreeissymmetric.Abinary tree is symmetric if it is a mirror image of itself around its center. The solution uses a recursive approach to check if the left and right subtrees of the root node are mirror images of each other.

3) Implementation/Code:

```
/**
 * Definition for a binary tree node.
 * structTreeNode{
 *     int val;
 *     TreeNode*left;
 *     TreeNode*right;
 *     TreeNode():val(0),left(nullptr),right(nullptr){}
 *     TreeNode(intx):val(x),left(nullptr),right(nullptr){}
 *     TreeNode(intx,TreeNode*left,TreeNode*right):val(x),left(left),right(right){}
 * };
 */
classSolution{ public:
    boolisMirror(TreeNode*t1,TreeNode*t2){ if
        (!t1 && !t2) return true;
        if (!t1 || !t2) return false;
        return (t1->val == t2->val) &&
               isMirror(t1->left,t2->right)&&
               isMirror(t1->right, t2->left);
    }
    boolisSymmetric(TreeNode*root){
        return isMirror(root, root);
    }
};
```

4) Output:



5) LearningOutcomes:

- UnderstandingTreeSymmetry.
- RecursiveTreeTraversal.
- Concept of Mirror Trees.
- Handling Base Cases in Recursion.
- EfficientCheckingUsingO(n)Time Complexity.