Experiment 6

Student Name: Shivam Yadav **UID: 22BCS15259**
Branch: CSE **Section: 638/A**
Semester: 6th **DOP: 18-02-2014**
Subject: AP **Subject Code:22CSP-351**

Aim:

Problem-1: Convert Sorted Array to Binary Search Tree

Algorithm:

Algorithm for Convert Sorted Array to Binary Search Tree (BST):

1. Define a TreeNode class with 'val', 'left', and 'right' attributes.

2. Implement a method sortedArrayToBST(int[] nums) that:

a. Calls a helper method buildBST(nums, left, right) with initial bounds (0, nums.length - 1).

3. In buildBST:

a. If left > right, return null (base case).

b. Calculate mid as left + (right - left) / 2 to avoid overflow.

c. Create a new TreeNode with nums[mid].

d. Recursively build the left subtree with range (left, mid - 1).

e. Recursively build the right subtree with range (mid + 1, right).

f. Return the root node.

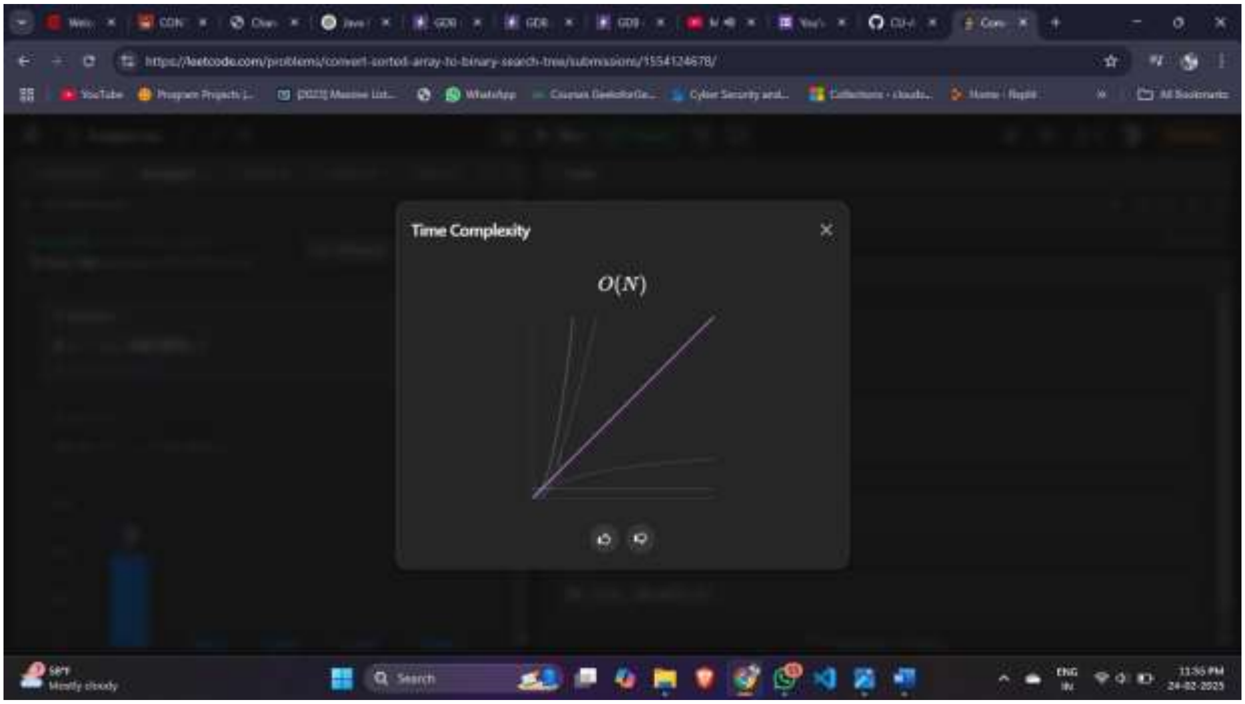
4. The recursion will ensure the tree is height-balanced.

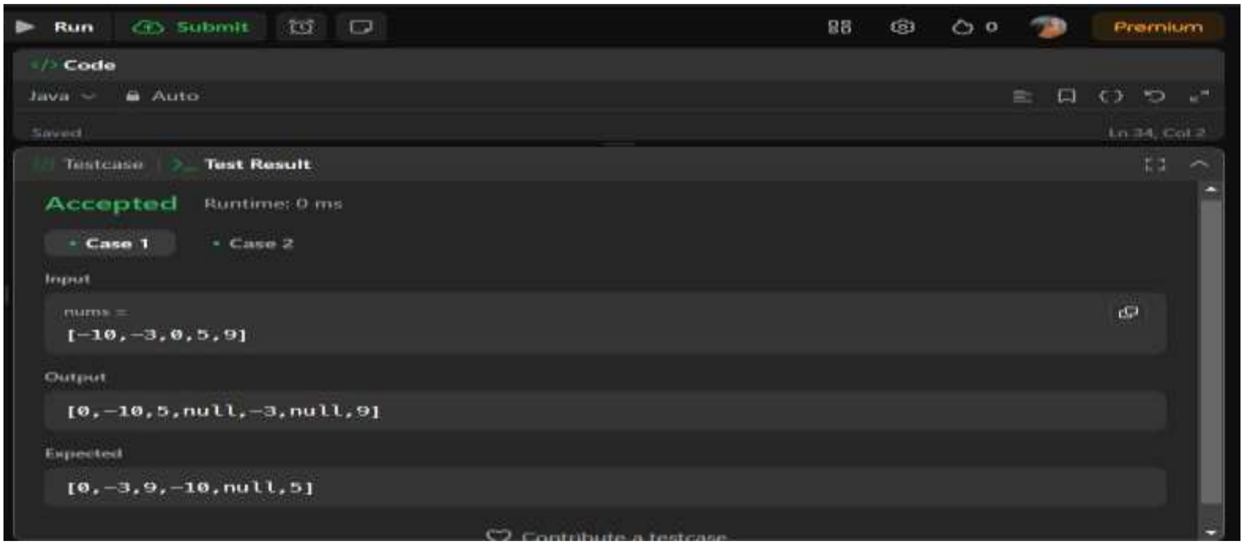
Code:

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}
public 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 - left) / 2;
        TreeNode root = new TreeNode(nums[mid]);
        root.left = helper(nums, left, mid - 1);
        root.right = helper(nums, mid + 1, right);
        return root;
    }
}
```

Link:- https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/

Output:


https://leetcode.com/problems/convert-sorted-array-to-binary-search-tree/submissions/1554124678/
Time Complexity
O(N)
Search
11:56 PM
24-02-2025



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]
```

Aim:

Problem-2: Maximum Depth of Binary Tree

Algorithm :

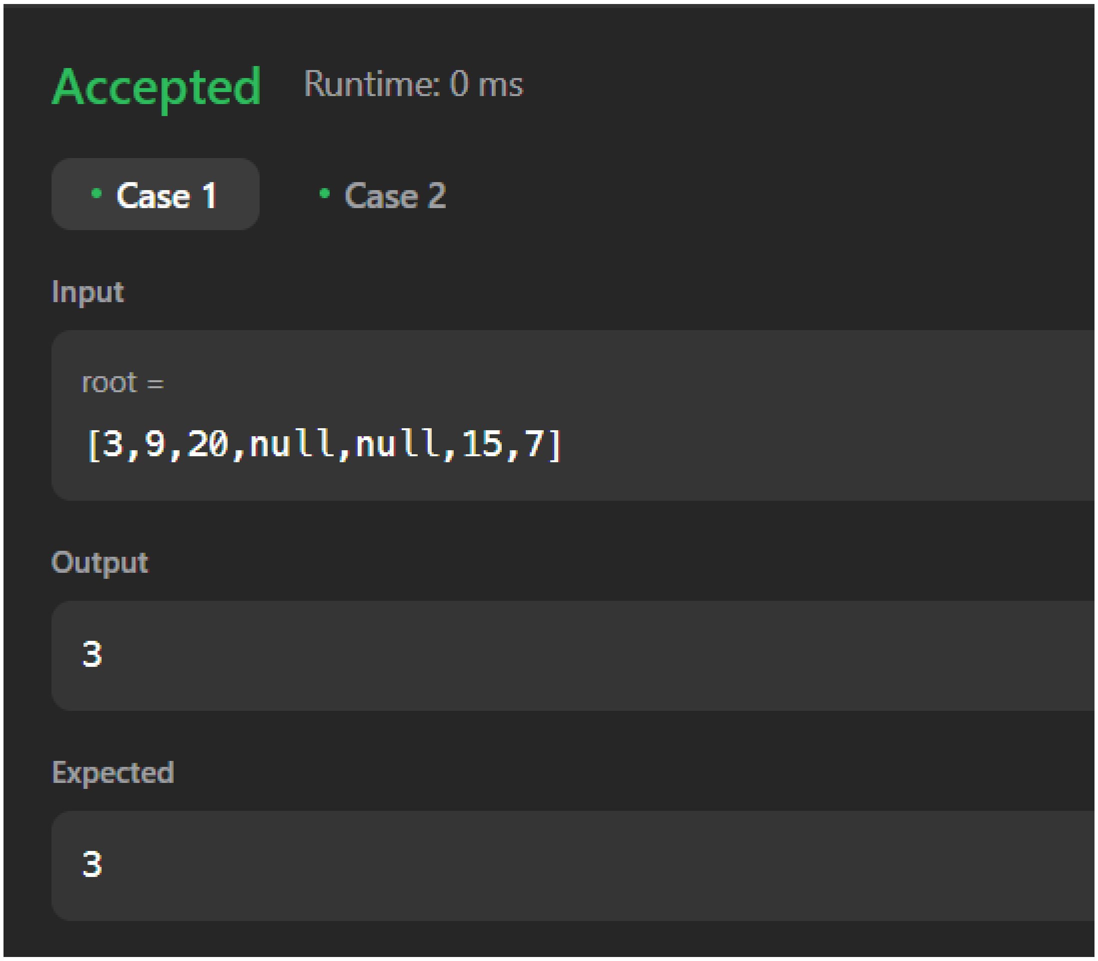
```
  1. Define a TreeNode class with 'val', 'left', and 'right' attributes.
// 2. Implement a method maxDepth(TreeNode root) that:
//    a. If root is null, return 0 (base case).
//    b. Recursively find the depth of the left subtree using maxDepth(root.left).
//    c. Recursively find the depth of the right subtree using maxDepth(root.right).
//    d. Return 1 + maximum of left and right subtree depths.
// 3. The recursion will traverse all nodes, ensuring the longest path is found.
```

Code :

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}
public class Solution {
    public int maxDepth(TreeNode root) {
      if (root == null) {
        return 0;
      }
      int leftDepth = maxDepth(root.left);
      int rightDepth = maxDepth(root.right);
      return Math.max(leftDepth, rightDepth) + 1;
    }
  }
```

Link:- https://leetcode.com/problems/maximum-depth-of-binary-tree/

OUTPUT:-



Accepted Runtime: 0 ms

Input

root =

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

Output

3

Expected

3

- **Learning Outcome: -**
  You learn how a binary tree is represented using nodes, where each node has.
- The code demonstrates recursion as an effective technique for tree traversal, where:
- The code helps you understand the concept of maximum depth (height) of a binary tree, which is the longest path from the root to a leaf node.
- The solution ensures that the resulting tree is balanced, meaning that the depth of the two subtrees of every node never differs by more than one.