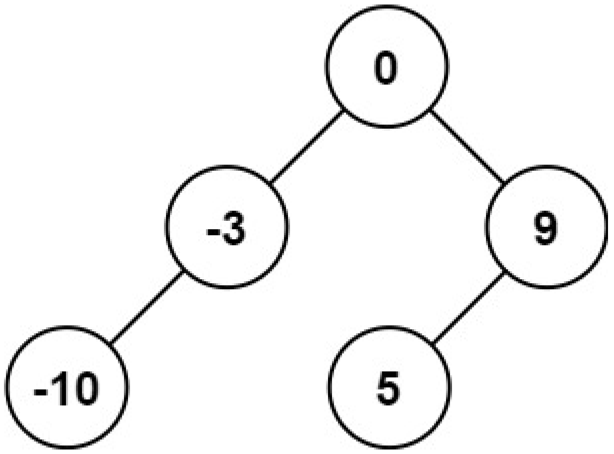
Experiment 6

Student Name: Ankit Kumar Yadav **UID: 22BCS11046**
Branch: BE-CSE **Section/Group: 638_A**
Semester: 6th **Date of Performance:25/02/25**
Subject Name: Advance Programming Lab -2 **Subject Code: 22CSP-351**

1. **problem:-Convert Sorted Array to Binary Search Tree**
2. **Aim:-**

Given an integer array nums where the elements are sorted in ascending order, convert it to a height-balanced binary search tree.

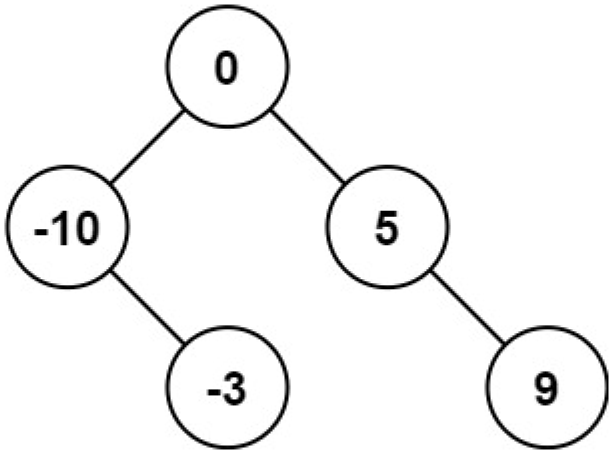


Example 1:

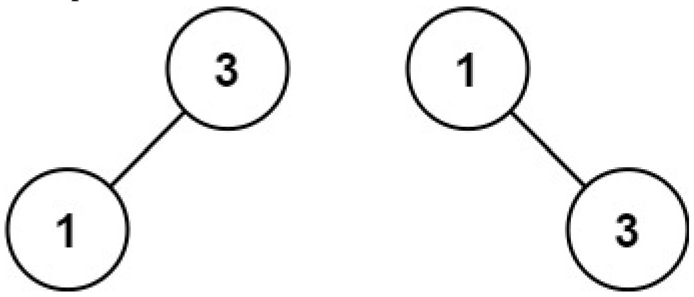
Input: nums = [-10,-3,0,5,9]

Output: [0,-3,9,-10,null,5]

Explanation: [0,-10,5,null,-3,null,9] is also accepted:



Example -2



Input: nums = [1,3]

Output: [3,1]

Explanation: [1,null,3] and [3,1] are both height-balanced BSTs.

Constraints:

1 <= nums.length <= 104

-104 <= nums[i] <= 104

nums is sorted in a strictly increasing order.

Link:- [Convert Sorted Array to Binary Search Tree - LeetCode]()

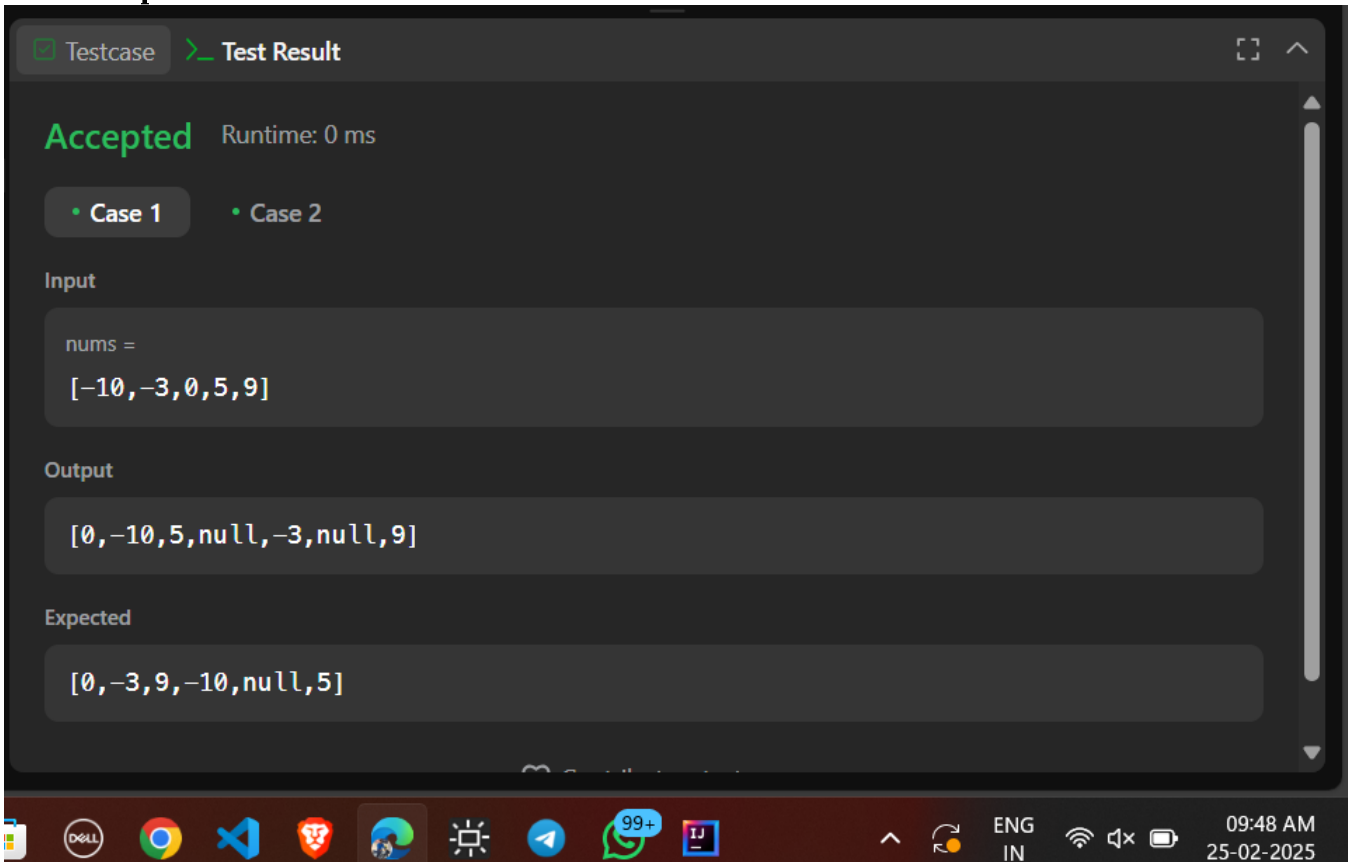
3. Objective:

The objective is to construct a height-balanced binary search tree from a sorted array while maintaining optimal depth. By selecting the middle element as the root and recursively constructing the left and right subtrees, the tree remains balanced. This ensures efficient searching and traversal, leveraging the properties of binary search trees.

4. Implementation/Code:

```
class Solution {
    public TreeNode sortedArrayToBST(int[] nums) {
        return buildBST(nums, 0, nums.length - 1);
    }

    private TreeNode buildBST(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 = buildBST(nums, left, mid - 1);
        root.right = buildBST(nums, mid + 1, right);

        return root;
    }
}
```

Output:-



Accepted Runtime: 0 ms

Case 1 Case 2

Input

```
nums =
[-10,-3,0,5,9]
```

Output

```
[0,-10,5,null,-3,null,9]
```

Expected

```
[0,-3,9,-10,null,5]
```

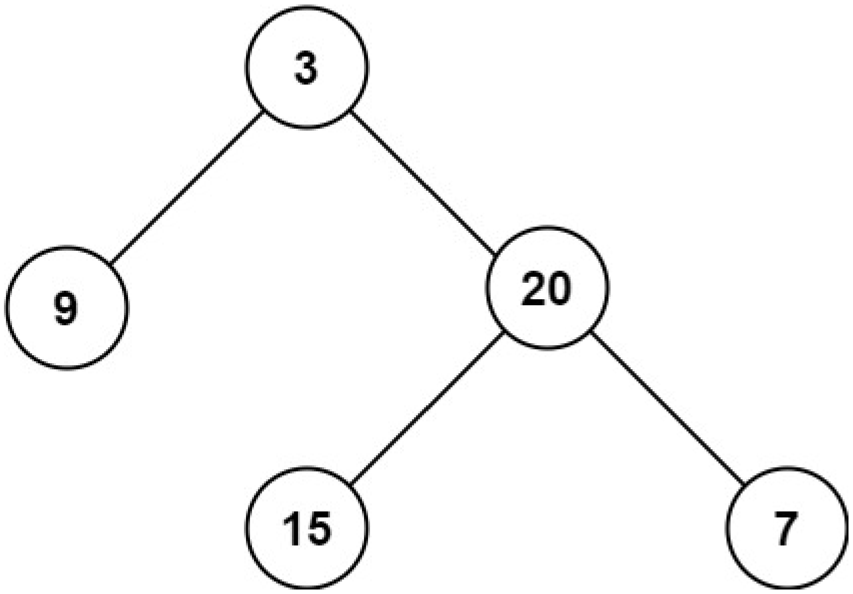
5. Learning outcomes:

1. Understand how to construct a balanced BST from a sorted array using recursion.
2. Learn to divide and conquer problems using recursive approaches effectively.
3. Gain knowledge of binary search tree properties and their advantages in searching operations.
4. Develop skills in handling edge cases, such as small arrays or arrays with an even number of elements.

..

1. **Problem-2:** Maximum Depth of Binary Tree
2. **Aim:**

Given the root of a binary tree, return its maximum depth.
A binary tree's maximum depth is the number of nodes along the longest path from the root node down to the farthest leaf node.
Example 1:



image

Input: root = [3,9,20,null,null,15,7]
Output: 3

Example 2:

Input: root = [1,null,2]
Output: 2
Constraints:
The number of nodes in the tree is in the range [0, 104].
-100 <= Node.val <= 100

Link:- Maximum Depth of Binary Tree - LeetCode

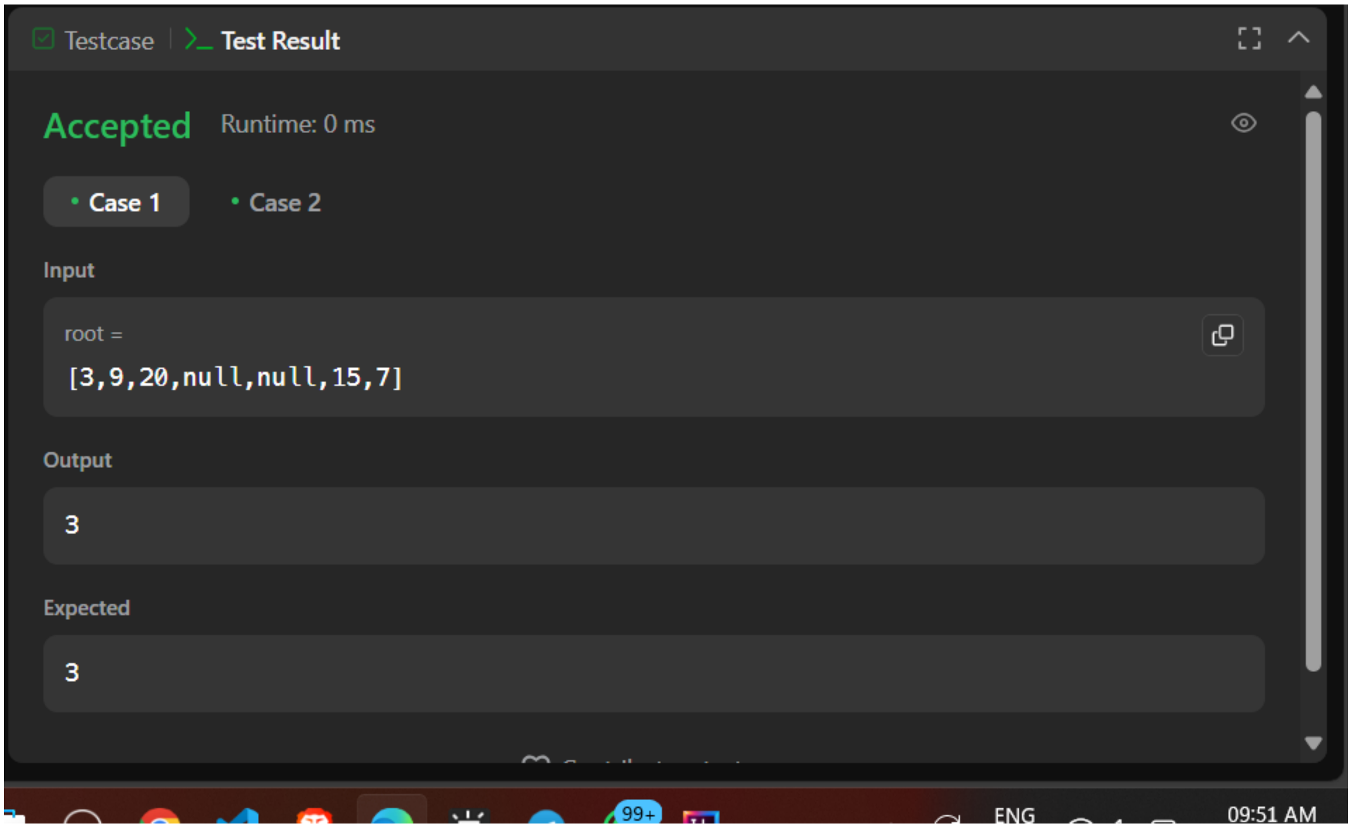
3. **Objective:**

The objective is to determine the maximum depth of a binary tree by computing the longest path from the root node to a leaf node. This involves implementing depth-first search (DFS) recursively and breadth-first search (BFS) iteratively to explore all levels of the tree efficiently.

4. Implementation/Code:

```
class Solution {
    public int maxDepth(TreeNode root) {
        if (root == null) return 0;

        Queue<TreeNode> queue = new LinkedList<>();
        queue.offer(root);
        int depth = 0;

        while (!queue.isEmpty()) {
            int size = queue.size();
            depth++;

            for (int i = 0; i < size; i++) {
                TreeNode node = queue.poll();
                if (node.left != null) queue.offer(node.left);
                if (node.right != null) queue.offer(node.right);
            }
        }
        return depth;
    }
}
```

Output:-



Accepted Runtime: 0 ms

Input

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

Output

```
3
```

Expected

```
3
```

6. Learning outcomes:

1. Learn how to traverse a binary tree using both DFS (recursion) and BFS (iteration).
2. Develop an understanding of recursive function calls and their space complexity.
3. Gain insights into tree traversal techniques and their applications in solving real-world problems.
4. Improve problem-solving skills by handling various tree structures, including skewed and balanced trees.