

## Assignment 6

<b>Name: Abhigyan</b>	<b>Uid: 22BCS10097</b>
<b>Branch: BE_CSE</b>	<b>Semester: 6<sup>th</sup></b>
<b>Section: IOT_637-B</b>	<b>Subject: AP Lab II</b>

### 108. Convert Sorted Array to Binary Search Tree

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

**Code:**

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int val) {
        this.val = val;
        this.left = null;
        this.right = null;
    }
    TreeNode(int val, TreeNode left, TreeNode right) {
        this.val = val;    this.left = left;    this.right =
        right;
    }
} 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; // Base case: when no elements left
int mid = left + (right - left) / 2; // Find the middle index

        TreeNode root = new TreeNode(nums[mid]); // Middle element becomes root
root.left = buildBST(nums, left, mid - 1); // Construct left subtree    root.right
= buildBST(nums, mid + 1, right); // Construct right subtree    return root;

    }

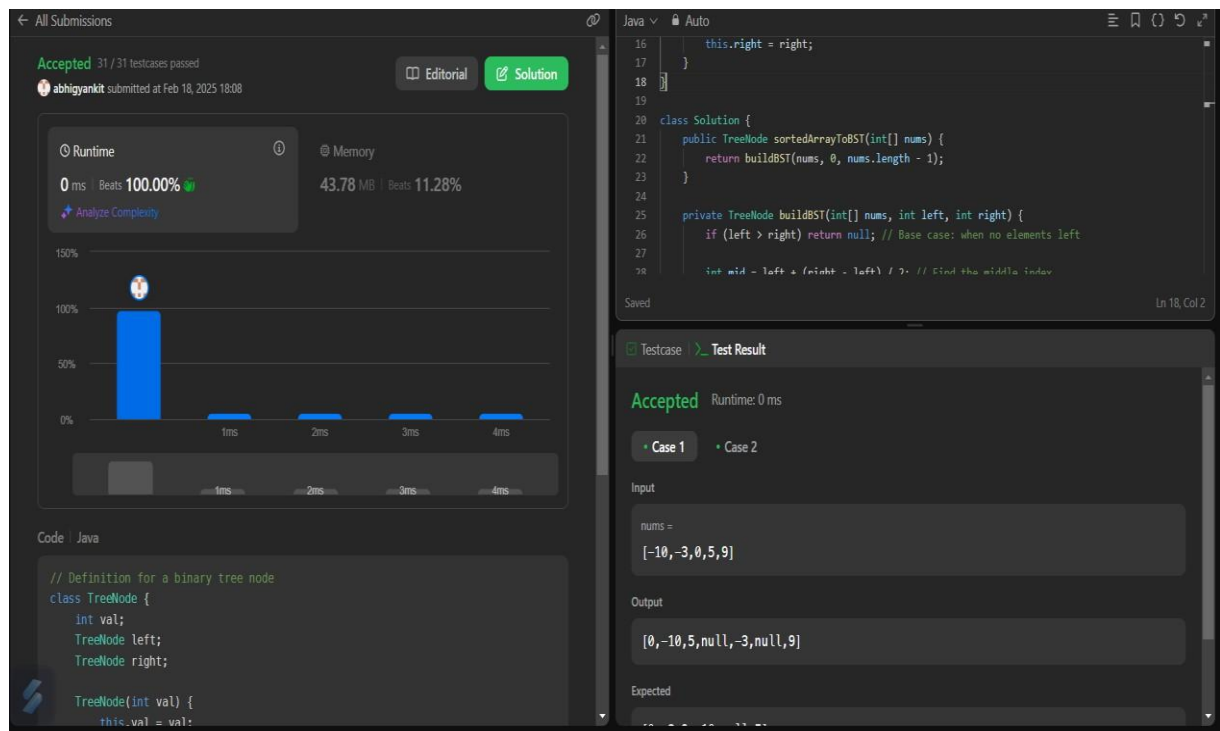
    // Utility function to print inorder traversal (for testing)
public void inorderTraversal(TreeNode root) {    if
(root != null) {
        inorderTraversal(root.left);
System.out.print(root.val + " ");
inorderTraversal(root.right);
    }
}

    // Main method for testing    public
static void main(String[] args) {
Solution solution = new Solution();
int[] nums = {-10, -3, 0, 5, 9};

        TreeNode root = solution.sortedArrayToBST(nums);
System.out.println("Inorder Traversal of BST:");
solution.inorderTraversal(root);
    }
}

```

## Output:



### 104. Maximum Depth of Binary Tree

**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.

#### Code:

```
class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}

class Solution {
    public int maxDepth(TreeNode root) {
        if (root == null) {
            return 0; // Base case: if the node is null, the depth is 0
        }
        int left = maxDepth(root.left);
        int right = maxDepth(root.right);
        return Math.max(left, right) + 1;
    }
}
```

```

    }

    int leftDepth = maxDepth(root.left);    int
    rightDepth = maxDepth(root.right);    return
    Math.max(leftDepth, rightDepth) + 1;

    }

}

```

## Output:

The screenshot displays a LeetCode submission for the problem "Maximum Depth of Binary Tree". The submission is accepted, with 39 / 39 testcases passed. The runtime is 0 ms (Beats 100.00%) and the memory usage is 43.02 MB (Beats 17.36%).

The code is written in Java and defines a `TreeNode` class and a `Solution` class. The `maxDepth` method in the `Solution` class calculates the maximum depth of the binary tree.

The test result shows the input `root = [3, 9, 20, null, null, 15, 7]` and the output `3`. The expected output is also `3`.

```

class TreeNode {
    int val;
    TreeNode left;
    TreeNode right;
    TreeNode(int x) { val = x; }
}

class Solution {
    public int maxDepth(TreeNode root) {
        if (root == null) {
            return 0; // Base case: if the node is null, the depth is 0
        }
        int leftDepth = maxDepth(root.left);
    }
}

```