# Experiment 3 A

**Student Name:** K.SATWIK **UID: 22BCS16246**
**Branch:** **CSE** **Section/Group: Ntpp 602-A**
**Semester:** **6TH** **Date of Performance:03/02/25**
**Subject Name: AP Lab-2** **Subject Code: 22CSH-352**

## 1. TITLE:

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.

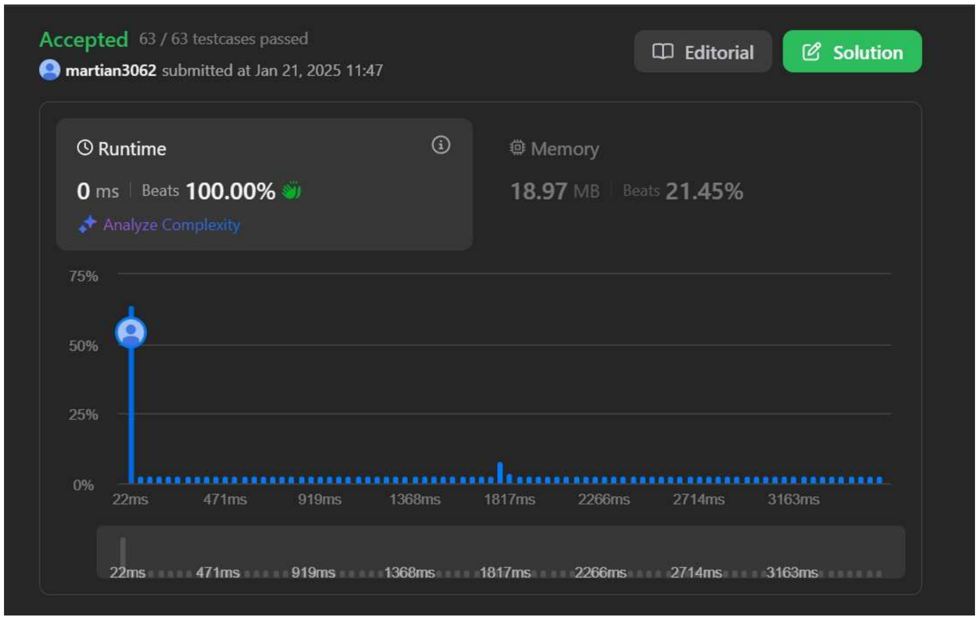
## 3. Algorithm

- Start DFS with the root node at depth 0.
- If the node is null, return the current depth.
- Recursively explore left and right children, increasing depth by 1.
- Return the maximum depth from left or right subtree.

### Implemetation/Code

```
class Solution:
    def maxDepth(self, root: Optional[TreeNode]) -> int:
        def dfs(root, depth):
            if not root: return depth
            return max(dfs(root.left, depth + 1), dfs(root.right, depth + 1))
              return dfs(root, 0)
```

## Output


Accepted 63 / 63 testcases passed
martian3062 submitted at Jan 21, 2025 11:47

Runtime
0 ms | Beats 100.00%

Memory
18.97 MB | Beats 21.45%

**Time Complexity** : O( n)

**Space Complexity :** O(h )

**Learning Outcomes:-**

- Understand how to use depth-first search for tree traversal.
- Gain skills in calculating the depth or height of binary trees.

# Experiment 3 B

**Student Name:** K.SATWIK **UID: 22BCS16246**
**Branch: BE-CSE** **Section/Group: Ntpp 602-A**
**Semester: 6TH** **Date of Performance:03/02/25**
**Subject Name: AP Lab-2** **Subject Code: 22CSH-352**

## 1. TITLE:

KTH Smallest Element in a BST

## 2. AIM:

Given the root of a binary search tree, and an integer k, return *the* k$^{th}$ *smallest value* ***(1-indexed)*** *of all the values of the nodes in the tree*.

## 3. Algorithm

- Perform an in-order traversal of the binary tree starting from the root.
- Use a generator to yield nodes' values one by one in their in-order sequence.
- Iterate up to the kth element of the generator.
- Return the kth smallest element from the traversal.

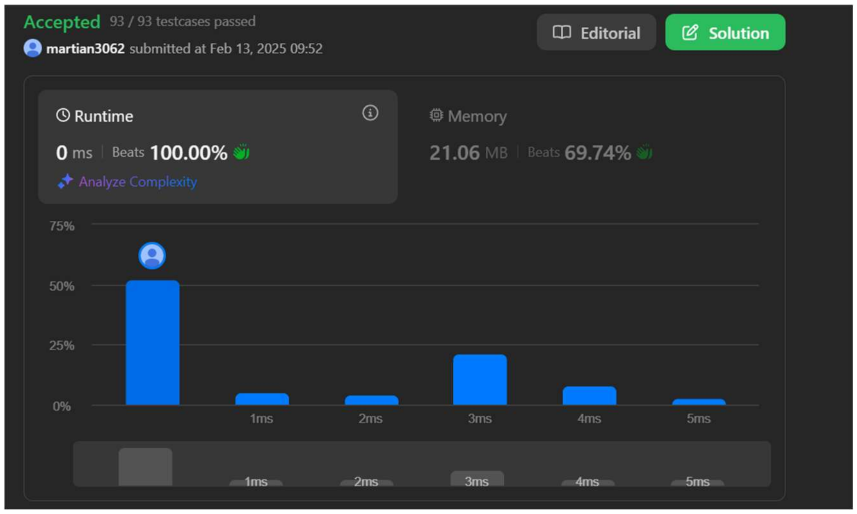
**Implemetation/Code:**

```
class Solution:
    def kthSmallest(self, root: TreeNode, k: int) -> int:
        def inorder(node):
            if not node:
                return
            yield from inorder(node.left)
            yield node.val
            yield from inorder(node.right)
        gen = inorder(root)
        for _ in range(k):
            result = next(gen)
        return result
```

## Implemetation/Code:

```
class Solution:
    def kthSmallest(self, root: TreeNode, k: int) -> int:
        def inorder(node):
            if not node:
                return
            yield from inorder(node.left)
            yield node.val
            yield from inorder(node.right)
        gen = inorder(root)
        for _ in range(k):
            result = next(gen)
        return result
```

## Output


Accepted 93 / 93 testcases passed
martian3062 submitted at Feb 13, 2025 09:52

Runtime
0 ms | Beats 100.00%

Memory
21.06 MB | Beats 69.74%

**Time Complexity** : O( k)

**Space Complexity :** O(h )

## Learning Outcomes:-

- Learn how to perform and apply in-order traversal in binary trees to solve problems.
- Python generators to manage state and produce results on demand during tree traversal