Maximum Depth of Binary Tree

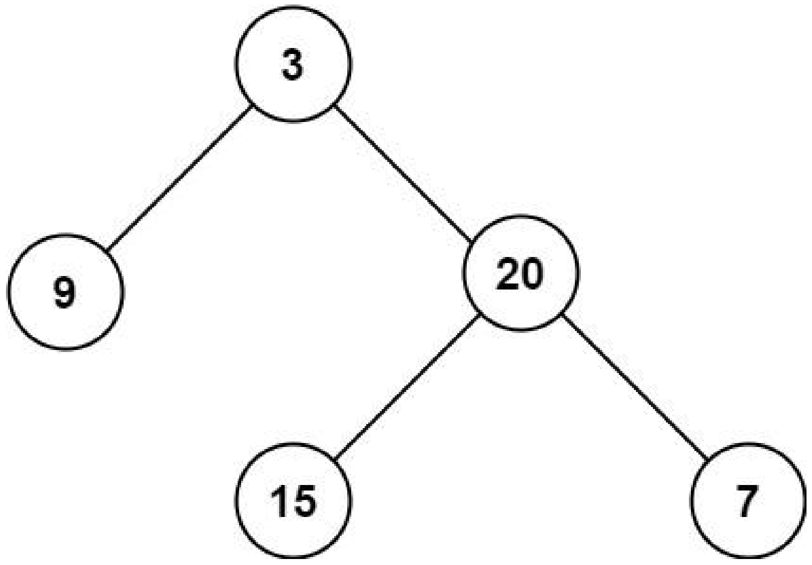
Question:

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

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:



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*

Approach 1:

Using Recursive Depth-First Search to find the maximum height.

Solution 1:

```
def maxDepth(self,root):

        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)
```

Time Complexity: O(T)

Space Complexity: O(1)

Approach 2:

Using Iterative Breadth-First Search using Queue to find the maximum height.

Solution 2:

```
def maxDepth(self, root: Optional[TreeNode]) -> int:
        depth = 0
        level = [root] if root else []
        while level:
                depth += 1
                queue = []
                for i in level:
                        if i.left:
                                queue.append(i.left)
                        if i.right:
                                queue.append(i.right)
                level = queue
        return depth
```

Time Complexity: O(T)

Space Complexity: O(T)

Approach 3:

Using Iterative Depth-First Search using Stack to find the maximum height.

Solution 3:

```
def maxDepth(self,root):
        ans = 0
        if not root:
                return 0
        stack = [] # stack
        stack.append((root,1))
        while DFS:
                root, depth = stack.pop()
                if depth > ans:
                        ans = depth
                if root.left:
                        stack.append((root.left, depth + 1))
                if root.right:
                        stack.append((root.right, depth + 1))
        return ans
```

Time Complexity: O(T)

Space Complexity: O(T)