

13
.5

Maximum Depth of Binary Tree

EASY

■ What You Will Learn

- ✓ Recursive definition of depth: $1 + \max(\text{left depth}, \text{right depth})$
- ✓ DFS recursive solution in 3 lines
- ✓ BFS iterative solution using level count
- ✓ Difference between height and depth

■ Intuition First

The maximum depth (or height) of a binary tree is the length of the longest path from the root down to any leaf node. Think of it like finding the tallest branch of a tree.

The beautiful recursive insight: the depth of any node = 1 (for itself) + the maximum depth of its left or right subtree. Base case: a null node has depth 0.

■ The Recursive Formula

- $\text{depth}(\text{null}) = 0$ (base case)
- $\text{depth}(\text{node}) = 1 + \max(\text{depth}(\text{node.left}), \text{depth}(\text{node.right}))$
- That's it — two lines of logic covers every case
- The recursion naturally picks the deeper of the two subtrees

■ Visual

3 depth = ?

/ \

9 20 depth = ?

/ \

15 7 depth = 1 (leaves)

$\text{depth}(15) = 1, \text{depth}(7) = 1$

$\text{depth}(20) = 1 + \max(1, 1) = 2$

$\text{depth}(9) = 1 + \max(0, 0) = 1$

$\text{depth}(3) = 1 + \max(1, 2) = 3 \leftarrow \text{answer}$

■ Dry Run — Recursion Tree

```
maxDepth(3)

■■■ maxDepth(9) → 1 + max(0,0) = 1
■■■ maxDepth(20) → 1 + max(maxDepth(15), maxDepth(7))
= 1 + max(1, 1) = 2
= 1 + max(1, 2) = 3 ✓
```

■ Java Code — Recursive (3 lines!)

```
public static int maxDepth(TreeNode root) {
    if (root == null) return 0; // Base case: empty tree
    int leftDepth = maxDepth(root.left);
    int rightDepth = maxDepth(root.right);
    return 1 + Math.max(leftDepth, rightDepth);
}
```

■ Java Code — BFS Iterative (count levels)

```
public static int maxDepthBFS(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++; // Each BFS level = one more depth unit
        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;
}
```

■■ Complexity

Case	Time	Space	What it means
Recursive	O(n)	O(h)	Visit every node. Stack depth = height h (O(log n) balanced, O(n) skewed)
BFS	O(n)	O(w)	Queue width w = max nodes at any level (O(n/2) for bottom level)

■■ Common Mistakes

- Returning $\max(\text{left}, \text{right})$ without adding 1 — forgets current node
- Confusing depth (root=1) vs height (root=0) — LeetCode uses depth (root counts as 1)
- Not handling null root — always check first

■ Height vs Depth

- Depth of a node = distance from ROOT to that node
- Height of a node = distance from that node to the deepest LEAF below it
- maxDepth of tree = height of root = depth of deepest leaf
- LeetCode's maxDepth = number of nodes on longest path from root to leaf

■ Visual Thinking

Imagine drilling down into the tree, taking the deepest possible path.

At each node, you ask: 'Which side is deeper — left or right?' and take that path.

The answer bubbles up from the leaves: leaves return 1, parents add 1 to max child.

Animation: depth values appear at leaves first, then propagate upward like bubbles.

■ Practice Problems

[Easy] Minimum depth of binary tree (nearest leaf from root)

[Easy] Count number of nodes in complete binary tree

[Med] Balanced binary tree check using max depth

[Med] Diameter of binary tree (uses same idea)

■ Quick Revision Box

- $\text{maxDepth} = 1 + \max(\text{leftDepth}, \text{rightDepth})$
- Base case: null $\rightarrow$ return 0
- Recursive is cleanest — $O(n)$ time, $O(h)$ space
- BFS: count levels — $O(n)$ time, $O(w)$ space

■ Depth of root = 1, depth of null = 0