

## Grokking the Coding Interview: Patterns for Coding Questions

36% completed

- Count Paths for a Sum (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1**
- Problem Challenge 2
- Solution Review: Problem Challenge 2

### Pattern: Two Heaps

- Introduction
- Find the Median of a Number Stream (medium)
- Sliding Window Median (hard)
- Maximize Capital (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

### Pattern: Subsets

- Introduction
- Subsets (easy)
- Subsets With Duplicates (easy)
- Permutations (medium)
- String Permutations by changing case (medium)
- Balanced Parentheses (hard)
- Unique Generalized Abbreviations (hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

### Pattern: Modified

#### Binary Search

- Introduction
- Order-agnostic Binary Search (easy)
- Ceiling of a Number (medium)
- Next Letter (medium)
- Number Range (medium)
- Search in a Sorted Infinite Array (medium)
- Minimum Difference Element (medium)
- Bitonic Array Maximum (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2
- Problem Challenge 3
- Solution Review: Problem Challenge 3

## Solution Review: Problem Challenge 1

### We'll cover the following

- Tree Diameter (medium)
- Solution
- Code
  - Time complexity
  - Space complexity

### Tree Diameter (medium)

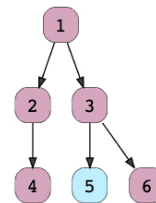
Given a binary tree, find the length of its diameter. The diameter of a tree is the number of nodes on the **longest path between any two leaf nodes**. The diameter of a tree may or may not pass through the root.

Note: You can always assume that there are at least two leaf nodes in the given tree.

#### Example 1:

Output: 5

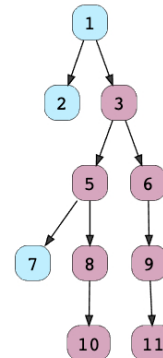
Explanation: The diameter of the tree is: [4, 2, 1, 3, 6]



#### Example 2:

Output: 7

Explanation: The diameter of the tree is: [10, 8, 5, 3, 6, 9, 11]



### Solution

This problem follows the [Binary Tree Path Sum](#) pattern. We can follow the same **DFS** approach. There will be a few differences:

- At every step, we need to find the height of both children of the current node. For this, we will make two recursive calls similar to **DFS**.
- The height of the current node will be equal to the maximum of the heights of its left or right children, plus '1' for the current node.
- The tree diameter at the current node will be equal to the height of the left child plus the height of the right child plus '1' for the current node:  $\text{diameter} = \text{leftTreeHeight} + \text{rightTreeHeight} + 1$ . To find the overall tree diameter, we will use a class level variable. This variable will store the maximum diameter of all the nodes visited so far, hence, eventually, it will have the final tree diameter.

### Code

Here is what our algorithm will look like:

```
Java Python3 C++ JS
29 // the height of right sub-trees + '1' for the current node
30 const diameter = leftTreeHeight + rightTreeHeight + 1;
31
32 // update the global tree diameter
33 this.treeDiameter = Math.max(this.treeDiameter, diameter);
34
35 // height of the current node will be equal to the maximum of the heights of
36 // left or right subtrees plus '1' for the current node
37 return Math.max(leftTreeHeight, rightTreeHeight) + 1;
38 }
39 }
```

Pattern: Bitwise XOR

Introduction

Single Number (easy)

Two Single Numbers (medium)

Complement of Base 10 Number (medium)

Problem Challenge 1

Solution Review: Problem Challenge 1

Pattern: Top 'K' Elements

Introduction

Top 'K' Numbers (easy)

Kth Smallest Number (easy)

'K' Closest Points to the Origin (easy)

Connect Ropes (easy)

Top 'K' Frequent Numbers (medium)

Frequency Sort (medium)

Kth Largest Number in a Stream (medium)

'K' Closest Numbers (medium)

Maximum Distinct Elements (medium)

Sum of Elements (medium)

Rearrange String (hard)

Problem Challenge 1

Solution Review: Problem Challenge 1

Problem Challenge 2

Solution Review: Problem Challenge 2

Problem Challenge 3

Solution Review: Problem Challenge 3

Pattern: K-way merge

Introduction

MW

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

```
40
41
42 const treeDiameter = new TreeDiameter();
43 const root = new TreeNode(1);
44 root.left = new TreeNode(2);
45 root.right = new TreeNode(3);
46 root.left.left = new TreeNode(4);
47 root.right.left = new TreeNode(5);
48 root.right.right = new TreeNode(6);
49 console.log(`Tree Diameter: ${treeDiameter.find_diameter(root)}`);
50 root.left.left = null;
51 root.right.left.left = new TreeNode(7);
52 root.right.left.right = new TreeNode(8);
53 root.right.right.left = new TreeNode(9);
54 root.right.left.right.left = new TreeNode(10);
55 root.right.right.left.left = new TreeNode(11);
56 console.log(`Tree Diameter: ${treeDiameter.find_diameter(root)}`);
```

RUN

SAVE

RESET

**Time complexity** ⓘ

The time complexity of the above algorithm is  $O(N)$ , where 'N' is the total number of nodes in the tree. This is due to the fact that we traverse each node once.

**Space complexity** ⓘ

The space complexity of the above algorithm will be  $O(N)$  in the worst case. This space will be used to store the recursion stack. The worst case will happen when the given tree is a linked list (i.e., every node has only one child).

Interviewing soon? We've partnered with [Hired](#) so that companies apply to you, instead of the other way around. [See how](#) ⓘ

← Back

Problem Challenge 1

MARK AS COMPLETED

Next →

Problem Challenge 2