







Grokking the Coding Interview: Patterns for Coding Questions

71% completed


Introduction Pattern: Sliding Window Pattern: Two Pointers Pattern: Fast & Slow pointers Pattern: Merge Intervals Pattern: Cyclic Sort Pattern: In-place Reversal of a LinkedList Pattern: Tree Breadth First Search Pattern: Tree Depth First Search Pattern: Two Heaps Pattern: Subsets Pattern: Modified Binary Search 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)

Introduction

We'll cover the following 

- What could go wrong with the above algorithm?
- Important properties of XOR to remember

XOR is a logical bitwise operator that returns 0 (false) if both bits are the same and returns 1 (true) otherwise. In other words, it only returns 1 if exactly one bit is set to 1 out of the two bits in comparison.

A	B	A xor B
0	0	0
0	1	1
1	0	1
1	1	0

It is surprising to know the approaches that the XOR operator enables us to solve certain problems. For example, let's take a look at the following problem:

Given an array of $n - 1$ integers in the range from 1 to n , find the one number that is missing from the array.

Example:

Input: 1, 5, 2, 6, 4
Answer: 3

A straight forward approach to solve this problem can be:

1. Find the sum of all integers from 1 to n ; let's call it `s1`.
2. Subtract all the numbers in the input array from `s1`; this will give us the missing number.

This is what the algorithm will look like:

JavaPython3JS C++

```
1 function find_missing_number(arr) {
2   const n = arr.length + 1;
3   // find sum of all numbers from 1 to n.
4   let s1 = 0;
5   for (let i = 1; i <= n; i++)
6     s1 += i;
7
8   // subtract all numbers in input from sum.
9   arr.forEach((num) => {
10     s1 -= num;
11   });
12
13   // s1, now, is the missing number
14   return s1;
15 }
16
17
18 console.log(`Missing number is: ${find_missing_number([1, 5, 2, 6, 4])}`);
```

RUNSAVERESET

Close

Output2.924s

Missing number is: 3

Time & Space complexity: The time complexity of the above algorithm is $O(n)$ and the space complexity is $O(1)$.

What could go wrong with the above algorithm? 

While finding the sum of numbers from 1 to n , we can get integer overflow when n is large.

Create