

Grokking the Coding Interview: Patterns for Coding Questions

72% completed



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)
- '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
- Merge K Sorted Lists (medium)
- Kth Smallest Number in M Sorted Lists (Medium)
- Kth Smallest Number in a Sorted Matrix (Hard)
- Smallest Number Range (Hard)
- Problem Challenge 1
- Solution Review: Problem Challenge 1

Pattern : 0/1 Knapsack



/Dynamic

Single Number (easy)

We'll cover the following



- Problem Statement
- Try it yourself
- Solution
 - Solution with XOR
- Code

Problem Statement

In a non-empty array of integers, every number appears twice except for one, find that single number.

Example 1:

```
Input: 1, 4, 2, 1, 3, 2, 3
Output: 4
```

Example 2:

```
Input: 7, 9, 7
Output: 9
```

Try it yourself

Try solving this question here:

JavaPython3JS C++

```
1 using namespace std;
2
3 #include <iostream>
4 #include <vector>
5
6 class SingleNumber {
7 public:
8     static int findSingleNumber(const vector<int>& arr) {
9         // TODO: Write your code here
10        return -1;
11    }
12 };
13
14 int main(int argc, char* argv[]) {
15     cout << SingleNumber::findSingleNumber(vector<int>{1, 4, 2, 1, 3, 2, 3}) << endl;
16 }
```

RUNSAVERESET

Solution

One straight forward solution can be to use a **HashMap** kind of data structure and iterate through the input:

- If number is already present in **HashMap**, remove it.
- If number is not present in **HashMap**, add it.
- In the end, only number left in the **HashMap** is our required single number.

Time and space complexity Time Complexity of the above solution will be $O(n)$ and space complexity will also be $O(n)$.

Can we do better than this using the **XOR Pattern**?

Solution with XOR

Recall the following two properties of XOR:

- It returns zero if we take XOR of two same numbers.
- It returns the same number if we XOR with zero.

So we can XOR all the numbers in the input; duplicate numbers will zero out each other and we will be left with the single number.

Code

Here is what our algorithm will look like:

JavaPython3C++JS

```
1 using namespace std;
2
3 #include <iostream>
4 #include <vector>
```

🌐

Explore

📁

Tracks

📖

My Courses

📄

Edpresso

👤

Refer a Friend

+

Create

🔗

Dynamic Programming

○

 Introduction

○

 0/1 Knapsack (medium)

○

 Equal Subset Sum Partition (medium)

○

 Subset Sum (medium)

○

 Minimum Subset Sum Difference (hard)

○

 Problem Challenge 1

○

 Solution Review: Problem Challenge 1

○

 Problem Challenge 2

○

 Solution Review: Problem Challenge 2

🔗

Pattern: Topological Sort (Graph)

○

 Introduction

○

 Topological Sort (medium)

○

 Tasks Scheduling (medium)

○

 Tasks Scheduling Order (medium)

○

 All Tasks Scheduling Orders (hard)

○

 Alien Dictionary (hard)

○

 Problem Challenge 1

○

 Solution Review: Problem Challenge 1

○

 Problem Challenge 2

○

 Solution Review: Problem Challenge 2

🔗

Miscellaneous

○

 Kth Smallest Number (hard)

🔗

Conclusions

○

 Where to Go from Here

Mark Course as Completed

```
5
6 class SingleNumber {
7     public:
8         static int findSingleNumber(const vector<int>& arr) {
9             int num = 0;
10            for (int i=0; i < arr.size(); i++) {
11                num = num ^ arr[i];
12            }
13            return num;
14        }
15    };
16
17    int main(int argc, char* argv[]) {
18        cout << SingleNumber::findSingleNumber(vector<int>{1, 4, 2, 1, 3, 2, 3}) << endl;
19    }
```

RUN

SAVE

RESET

🔄

Output

0.758s

4

Close

Time Complexity: Time complexity of this solution is $O(n)$ as we iterate through all numbers of the input once.

Space Complexity: The algorithm runs in constant space $O(1)$.

Interviewing soon? We've partnered with [Hired](#) so that companies apply to you instead of you applying to them. [See how](#) 🕒

← Back

Next →

Introduction

Two Single Numbers (medium)

✅

Mark as Completed

🗨

 Report an Issue

🔍

 Ask a Question