

## Grokking the Coding Interview: Patterns for Coding Questions

73% completed



### 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)
- '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

## Two Single Numbers (medium)

### We'll cover the following

- Problem Statement
- Try it yourself
- Solution
  - Code
  - Time Complexity
  - Space Complexity

### Problem Statement

In a non-empty array of numbers, every number appears exactly twice except two numbers that appear only once. Find the two numbers that appear only once.

#### Example 1:

Input: [1, 4, 2, 1, 3, 5, 6, 2, 3, 5]  
Output: [4, 6]

#### Example 2:

Input: [2, 1, 3, 2]  
Output: [1, 3]

### Try it yourself

Try solving this question here:

JavaPython3JS C++

```
1 class TwoSingleNumbers {
2
3     public static int[] findSingleNumbers(int[] nums) {
4         // TODO: Write your code here
5         return new int[] { -1, -1 };
6     }
7
8     public static void main(String[] args) {
9         int[] arr = new int[] { 1, 4, 2, 1, 3, 5, 6, 2, 3, 5 };
10        int[] result = TwoSingleNumbers.findSingleNumbers(arr);
11        System.out.println("Single numbers are: " + result[0] + ", " + result[1]);
12
13        arr = new int[] { 2, 1, 3, 2 };
14        result = TwoSingleNumbers.findSingleNumbers(arr);
15        System.out.println("Single numbers are: " + result[0] + ", " + result[1]);
16    }
17 }
18
```

RUNSAVERESET↺

### Solution

This problem is quite similar to [Single Number](#), the only difference is that, in this problem, we have two single numbers instead of one. Can we still use XOR to solve this problem?

Let's assume `num1` and `num2` are the two single numbers. If we do XOR of all elements of the given array, we will be left with XOR of `num1` and `num2` as all other numbers will cancel each other because all of them appeared twice. Let's call this XOR `n1xn2`. Now that we have XOR of `num1` and `num2`, how can we find these two single numbers?

As we know that `num1` and `num2` are two different numbers, therefore, they should have at least one bit different between them. If a bit in `n1xn2` is '1', this means that `num1` and `num2` have different bits in that place, as we know that we can get '1' only when we do XOR of two different bits, i.e.,

```
1 XOR 0 = 0 XOR 1 = 1
```

We can take any bit which is '1' in `n1xn2` and partition all numbers in the given array into two groups based on that bit. One group will have all those numbers with that bit set to '0' and the other with the bit set to '1'. This will ensure that `num1` will be in one group and `num2` will be in the other. We can take XOR of all numbers in each group separately to get `num1` and `num2`, as all other numbers in each group will cancel each other. Here are the steps of our algorithm:

- Taking XOR of all numbers in the given array will give us XOR of `num1` and `num2`, calling this XOR as `n1xn2`.
- Find any bit which is set in `n1xn2`. We can take the rightmost bit which is '1'. Let's call this

Challenge 4

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 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

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

rightmostSetBit.

3. Iterate through all numbers of the input array to partition them into two groups based on rightmostSetBit. Take XOR of all numbers in both the groups separately. Both these XORs are our required numbers.

Code

Here is what our algorithm will look like:

JavaPython3C++JS

```
1 class TwoSingleNumbers {
2
3     public static int[] findSingleNumbers(int[] nums) {
4         // get the XOR of the all the numbers
5         int n1xn2 = 0;
6         for (int num : nums) {
7             n1xn2 ^= num;
8         }
9
10        // get the rightmost bit that is '1'
11        int rightmostSetBit = 1;
12        while ((rightmostSetBit & n1xn2) == 0) {
13            rightmostSetBit = rightmostSetBit << 1;
14        }
15        int num1 = 0, num2 = 0;
16        for (int num : nums) {
17            if ((num & rightmostSetBit) != 0) // the bit is set
18                num1 ^= num;
19            else // the bit is not set
20                num2 ^= num;
21        }
22        return new int[] { num1, num2 };
23    }
24
25    public static void main(String[] args) {
26        int[] arr = new int[] { 1, 4, 2, 1, 3, 5, 6, 2, 3, 5 };
27        int[] result = TwoSingleNumbers.findSingleNumbers(arr);
28        System.out.println("Single numbers are: " + result[0] + ", " + result[1]);
29    }
30 }
```

RUN

SAVE

RESET

Close

Output

Single numbers are: 6, 4  
Single numbers are: 3, 1

Time Complexity

The time complexity of this solution is  $O(n)$  where 'n' is the number of elements in the input array.

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

Single Number (easy)

Complement of Base 10 Number (me...

Mark as Completed

Report an Issue

Ask a Question