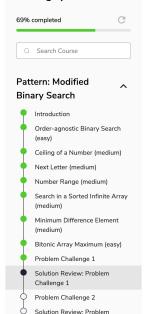


## Grokking the Coding Interview: Patterns for Coding Questions



## Pattern: Bitwise XOR

Introduction

Solution Review: Problem Challenge 3

Challenge 2 Problem Challenge 3

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

# Solution Review: Problem Challenge 1



#### Search Bitonic Array (medium)

Given a Bitonic array, find if a given 'key' is present in it. An array is considered bitonic if it is monotonically increasing and then monotonically decreasing. Monotonically increasing or decreasing means that for any index i in the array arr[i] != arr[i+1].

Write a function to return the index of the 'key'. If the 'key' is not present, return -1.

#### Example 1:

```
Input: [1, 3, 8, 4, 3], key=4
Output: 3
```

#### Example 2:

```
Input: [3, 8, 3, 1], key=8
Output: 1
```

#### Example 3:

```
Input: [1, 3, 8, 12], key=12
Output: 3
```

### Example 4:

```
Input: [10, 9, 8], key=10
Output: 0
```

## Solution

The problem follows the **Binary Search** pattern. Since Binary Search helps us efficiently find a number in a sorted array we can use a modified version of the Binary Search to find the 'key' in the bitonic array.

Here is how we can search in a bitonic array:

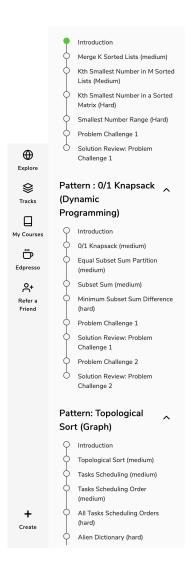
- 1. First, we can find the index of the maximum value of the bitonic array, similar to Bitonic Array Maximum. Let's call the index of the maximum number maxIndex.
- 2. Now, we can break the array into two sub-arrays:
  - Array from index '0' to maxIndex, sorted in ascending order.
  - Array from index maxIndex+1 to array\_length-1, sorted in descending order.
- 3. We can then call **Binary Search** separately in these two arrays to search the 'key'. We can use the same Order-agnostic Binary Search for searching.

## Code

Here is what our algorithm will look like:

```
function search_bitonic_array(arr, key) {
    const maxIndex = find_max(arr);
    const keyIndex = binary_search(arr, key, 0, maxIndex);
    if (keyIndex != -1) {
        return keyIndex;
    }
    return binary_search(arr, key, maxIndex + 1, arr.length - 1);
    }
}

// find index of the maximum value in a bitonic array
function find_max(arr) {
    let start = 0,
    lend = arr.length - 1;
    while (start < end) {
        const mid = Math.floor(start + (end - start) / 2);
    if (arr[mid] > arr[mid] + 1]) {
        end = mid;
    } else {
        start = mid + 1;
    }
}
```





## Time complexity

Since we are reducing the search range by half at every step, this means that the time complexity of our algorithm will be  $O(\log N)$  where 'N' is the total elements in the given 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  $\quad\times$  applying to them. See how  $\odot$ 

