

Grokking the Coding Interview: Patterns for Coding Questions

66% completed



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

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

Number Range (medium)

We'll cover the following

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

Problem Statement

Given an array of numbers sorted in ascending order, find the range of a given number 'key'. The range of the 'key' will be the first and last position of the 'key' in the array.

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

Example 1:

```
Input: [4, 6, 6, 6, 9], key = 6
Output: [1, 3]
```

Example 2:

```
Input: [1, 3, 8, 10, 15], key = 10
Output: [3, 3]
```

Example 3:

```
Input: [1, 3, 8, 10, 15], key = 12
Output: [-1, -1]
```

Try it yourself

Try solving this question here:

JavaPython3JS C++

```
1 const find_range = function(arr, key) {
2   result = [-1, -1];
3   // TODO: Write your code here
4   return result;
5 };
6
7
8 console.log(find_range([4, 6, 6, 6, 9], 6))
9 console.log(find_range([1, 3, 8, 10, 15], 10))
10 console.log(find_range([1, 3, 8, 10, 15], 12))
11
```

RUNSAVERESET

Solution

The problem follows the **Binary Search** pattern. Since Binary Search helps us find a number in a sorted array efficiently, we can use a modified version of the Binary Search to find the first and the last position of a number.

We can use a similar approach as discussed in [Order-agnostic Binary Search](#). We will try to search for the 'key' in the given array; if the 'key' is found (i.e. `key == arr[middle]`) we have two options:

- When trying to find the first position of the 'key', we can update `end = middle - 1` to see if the key is present before `middle`.
- When trying to find the last position of the 'key', we can update `start = middle + 1` to see if the key is present after `middle`.

In both cases, we will keep track of the last position where we found the 'key'. These positions will be the required range.

Code

Here is what our algorithm will look like:

JavaPython3C++JS

```
1 function find_range(arr, key) {
2   result = [-1, -1];
3   result[0] = binary_search(arr, key, false);
4   if (result[0] !== -1) { // no need to search - if 'key' is not present in the input array
```

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

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

```
5     result[1] = binary_search(arr, key, true);
6   }
7
8   return result;
9 }
10
11 // modified Binary Search
12 function binary_search(arr, key, findMaxIndex) {
13   let keyIndex = -1;
14   let start = 0;
15   let end = arr.length - 1;
16   while (start <= end) {
17     mid = Math.floor(start + (end - start) / 2);
18     if (key < arr[mid]) {
19       end = mid - 1;
20     } else if (key > arr[mid]) {
21       start = mid + 1;
22     } else { // key === arr[mid]
23       keyIndex = mid;
24       if (findMaxIndex) {
25         start = mid + 1; // search ahead to find the last index of 'key'
26       } else {
27         end = mid - 1; // search behind to find the first index of 'key'
28       }
29     }
30   }
31   return keyIndex;
32 }
```

RUN

SAVE

RESET

Close

Output

1.801s

```
[ 1, 3 ]
[ 3, 3 ]
[ -1, -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 applying to them. [See how](#)

← Back

Next →

Next Letter (medium)

Search in a Sorted Infinite Array (medi...

Mark as Completed

Report an Issue

Ask a Question