


## Grokking the Coding Interview: Patterns for Coding Questions


59% completed





 Solution Review: Problem Challenge 2


### Pattern: Two Heaps





 Introduction

 Find the Median of a Number Stream (medium)

 Sliding Window Median (hard)


 Maximize Capital (hard)


 Problem Challenge 1


 Solution Review: Problem Challenge 1


### Pattern: Subsets





 Introduction


 Subsets (easy)


 Subsets With Duplicates (easy)


 Permutations (medium)


 String Permutations by changing case (medium)


 Balanced Parentheses (hard)


 Unique Generalized Abbreviations (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: 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

## Solution Review: Problem Challenge 1

### We'll cover the following



- Next Interval (hard)
- Solution
  - Code
  - Time complexity
  - Space complexity

### Next Interval (hard)

Given an array of intervals, find the next interval of each interval. In a list of intervals, for an interval 'i' its next interval 'j' will have the smallest 'start' greater than or equal to the 'end' of 'i'.

Write a function to return an array containing indices of the next interval of each input interval. If there is no next interval of a given interval, return -1. It is given that none of the intervals have the same start point.

#### Example 1:

**Input:** Intervals [[2,3], [3,4], [5,6]]

**Output:** [1, 2, -1]

**Explanation:** The next interval of [2,3] is [3,4] having index '1'. Similarly, the next interval of [3,4] is [5,6] having index '2'. There is no next interval for [5,6] hence we have '-1'.

#### Example 2:

**Input:** Intervals [[3,4], [1,5], [4,6]]

**Output:** [2, -1, -1]

**Explanation:** The next interval of [3,4] is [4,6] which has index '2'. There is no next interval for [1,5] and [4,6].

### Solution

A brute force solution could be to take one interval at a time and go through all the other intervals to find the next interval. This algorithm will take  $O(N^2)$  where 'N' is the total number of intervals. Can we do better than that?

We can utilize the **Two Heaps** approach. We can push all intervals into two heaps: one heap to sort the intervals on maximum start time (let's call it `maxStartHeap`) and the other on maximum end time (let's call it `maxEndHeap`). We can then iterate through all intervals of the `maxEndHeap` to find their next interval. Our algorithm will have the following steps:

1. Take out the top (having highest end) interval from the `maxEndHeap` to find its next interval. Let's call this interval `topEnd`.
2. Find an interval in the `maxStartHeap` with the closest start greater than or equal to the start of `topEnd`. Since `maxStartHeap` is sorted by 'start' of intervals, it is easy to find the interval with the highest 'start'. Let's call this interval `topStart`.
3. Add the index of `topStart` in the result array as the next interval of `topEnd`. If we can't find the next interval, add '-1' in the result array.
4. Put the `topStart` back in the `maxStartHeap`, as it could be the next interval of other intervals.
5. Repeat the steps 1-4 until we have no intervals left in `maxEndHeap`.

#### Code

Here is what our algorithm will look like:

```
1  const Heap = require('./collections/heap'); //http://www.collectionsjs.com
2
3
4  class Interval {
5    constructor(start, end) {
6      this.start = start;
7      this.end = end;
8    }
9  }
10
11
12  function find_next_interval(intervals) {
13    const n = intervals.length;
14
15    // heaps for finding the maximum start and end
16    const maxStartHeap = new Heap([], null, ((a, b) => a[0] - b[0]));
17    const maxEndHeap = new Heap([], null, ((a, b) => a[0] - b[0]));
18
```

⌕

Explore

📁

Tracks

📖

My Courses

☕

Edpresso

👤+

Refer a Friend

+

Create

(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

```
19  const result = Array(n).fill(0);
20  for (endIndex = 0; endIndex < n; endIndex++) {
21      maxStartHeap.push([intervals[endIndex].start, endIndex]);
22      maxEndHeap.push([intervals[endIndex].end, endIndex]);
23  }
24
25  // go through all the intervals to find each interval's next interval
26  for (i = 0; i < n; i++) {
27      // let's find the next interval of the interval which has the highest 'end'
28      const [topEnd, endIndex] = maxEndHeap.pop();
```

RUN

SAVE

RESET

↺

Close

Output

5.540s

Next interval indices are: 1,2,-1

Next interval indices are: 2,-1,-1

Time complexity

The time complexity of our algorithm will be  $O(N\log N)$ , where 'N' is the total number of intervals.

Space complexity

The space complexity will be  $O(N)$  because we will be storing all the intervals in the heaps.

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

×

← Back

Next →

Problem Challenge 1

Introduction

✔ Mark as Completed

🕒 Report an Issue

🔍 Ask a Question