

Grokking the Coding Interview: Patterns for Coding Questions

22% completed

Intervals

- Introduction
- Merge Intervals (medium)
- Insert Interval (medium)
- Intervals Intersection (medium)
- Conflicting Appointments (medium)
- 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: Cyclic Sort

- Introduction
- Cyclic Sort (easy)
- Find the Missing Number (easy)
- Find all Missing Numbers (easy)
- Find the Duplicate Number (easy)
- Find all Duplicate Numbers (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: In-place Reversal of a LinkedList

- Introduction
- Reverse a LinkedList (easy)
- Reverse a Sub-list (medium)
- Reverse every K-element Sub-list (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Tree Breadth First Search

- Introduction
- Binary Tree Level Order Traversal (easy)
- Reverse Level Order Traversal (easy)
- Zigzag Traversal (medium)
- Level Averages in a Binary Tree (easy)
- Minimum Depth of a Binary Tree (easy)

Solution Review: Problem Challenge 1

We'll cover the following

- Minimum Meeting Rooms (hard)
- Solution
- Code
 - Time complexity
 - Space complexity
- Similar Problems

Minimum Meeting Rooms (hard)

Given a list of intervals representing the start and end time of 'N' meetings, find the **minimum number of rooms** required to **hold all the meetings**.

Example 1:

```
Meetings: [[1,4], [2,5], [7,9]]
Output: 2
Explanation: Since [1,4] and [2,5] overlap, we need two rooms to hold these two meetings. [7,9] can occur in any of the two rooms later.
```

Example 2:

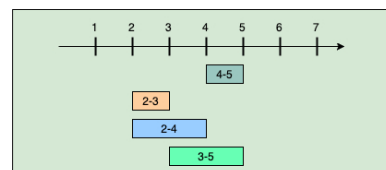
```
Meetings: [[6,7], [2,4], [8,12]]
Output: 1
Explanation: None of the meetings overlap, therefore we only need one room to hold all meetings.
```

Example 3:

```
Meetings: [[1,4], [2,3], [3,6]]
Output: 2
Explanation: Since [1,4] overlaps with the other two meetings [2,3] and [3,6], we need two rooms to hold all the meetings.
```

Example 4:

```
Meetings: [[4,5], [2,3], [2,4], [3,5]]
Output: 2
Explanation: We will need one room for [2,3] and [3,5], and another room for [2,4] and [4,5].
Here is a visual representation of Example 4:
```



Solution

Let's take the above-mentioned example (4) and try to follow our [Merge Intervals](#) approach:

Meetings: [[4,5], [2,3], [2,4], [3,5]]

Step 1: Sorting these meetings on their start time will give us: [[2,3], [2,4], [3,5], [4,5]]

Step 2: Merging overlapping meetings:

- [2,3] overlaps with [2,4], so after merging we'll have => [[2,4], [3,5], [4,5]]
- [2,4] overlaps with [3,5], so after merging we'll have => [[2,5], [4,5]]
- [2,5] overlaps [4,5], so after merging we'll have => [2,5]

Since all the given meetings have merged into one big meeting ([2,5]), does this mean that they all are overlapping and we need a minimum of four rooms to hold these meetings? You might have already guessed that the answer is NO! As we can clearly see, some meetings are mutually exclusive. For example, [2,3] and [3,5] do not overlap and can happen in one room. So, to correctly solve our problem, we need to keep track of the mutual exclusiveness of the overlapping meetings.

- Level Order Successor (easy)
- Connect Level Order Siblings (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- Solution Review: Problem Challenge 2

Pattern: Tree Depth First Search

- Introduction
- Binary Tree Path Sum (easy)
- All Paths for a Sum (medium)
- Sum of Path Numbers (medium)
- Path With Given Sequence (medium)
- Count Paths for a Sum (medium)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2
- 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

Here is what our strategy will look like:

1. We will sort the meetings based on start time.
2. We will schedule the first meeting (let's call it **m1**) in one room (let's call it **r1**).
3. If the next meeting **m2** is not overlapping with **m1**, we can safely schedule it in the same room **r1**.
4. If the next meeting **m3** is overlapping with **m2** we can't use **r1**, so we will schedule it in another room (let's call it **r2**).
5. Now if the next meeting **m4** is overlapping with **m3**, we need to see if the room **r1** has become free. For this, we need to keep track of the end time of the meeting happening in it. If the end time of **m2** is before the start time of **m4**, we can use that room **r1**, otherwise, we need to schedule **m4** in another room **r3**.

We can conclude that we need to **keep track of the ending time of all the meetings currently happening** so that when we try to schedule a new meeting, we can see what meetings have already ended. We need to put this information in a data structure that can easily give us the smallest ending time. A **Min Heap** would fit our requirements best.

So our algorithm will look like this:

1. Sort all the meetings on their start time.
2. Create a min-heap to store all the active meetings. This min-heap will also be used to find the active meeting with the smallest end time.
3. Iterate through all the meetings one by one to add them in the min-heap. Let's say we are trying to schedule the meeting **m1**.
4. Since the min-heap contains all the active meetings, so before scheduling **m1** we can remove all meetings from the heap that have ended before **m1**, i.e., remove all meetings from the heap that have an end time smaller than or equal to the start time of **m1**.
5. Now add **m1** to the heap.
6. The heap will always have all the overlapping meetings, so we will need rooms for all of them. Keep a counter to remember the maximum size of the heap at any time which will be the minimum number of rooms needed.

Code

Here is what our algorithm will look like:

Java

Python3

C++

JS

```

1  const Heap = require('./collections/heap'); //http://www.collectionsjs.com
2
3  class Meeting {
4    constructor(start, end) {
5      this.start = start;
6      this.end = end;
7    }
8  }
9
10
11 function min_meeting_rooms(meetings) {
12   // sort the meetings by start time
13   meetings.sort((a, b) => a.start - b.start);
14
15   let minRooms = 0,
16       minHeap = new Heap([], null, ((a, b) => b.end - a.end));
17   for (i = 0; i < meetings.length; i++) {
18     // remove all the meetings that have ended
19     while (minHeap.length > 0 && meetings[i].start >= minHeap.peek().end) {
20       minHeap.pop();
21     }
22     // add the current meeting into min_heap
23     minHeap.push(meetings[i]);
24     // all active meetings are in the min_heap, so we need rooms for all of them.
25     minRooms = Math.max(minRooms, minHeap.length);
26   }
27   return minRooms;
28 }

```

RUN

SAVE

RESET

Close

Output

3.738s

Minimum meeting rooms required: 2

Minimum meeting rooms required: 2

Minimum meeting rooms required: 1

Minimum meeting rooms required: 2

Minimum meeting rooms required: 2

Time complexity

The time complexity of the above algorithm is $O(N * \log N)$, where 'N' is the total number of meetings. This is due to the sorting that we did in the beginning. Also, while iterating the meetings we might need to poll/offer meeting to the priority queue. Each of these operations can take $O(\log N)$. Overall our algorithm will take $O(N \log N)$.

Space complexity

MW

Explore

Tracks

My Courses

Edpresso

Refer a Friend

Create

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)

Space complexity

The space complexity of the above algorithm will be $O(N)$ which is required for sorting. Also, in the worst case scenario, we'll have to insert all the meetings into the **Min Heap** (when all meetings overlap) which will also take $O(N)$ space. The overall space complexity of our algorithm is $O(N)$.

Similar Problems #

Problem 1: Given a list of intervals, find the point where the maximum number of intervals overlap.

Problem 2: Given a list of intervals representing the arrival and departure times of trains to a train station, our goal is to find the minimum number of platforms required for the train station so that no train has to wait.

Both of these problems can be solved using the approach discussed above.

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

← Back

Problem Challenge 1

✓ MARK AS COMPLETED

Next →

Problem Challenge 2

Report an Issue