

## Grokking the Coding Interview: Patterns for Coding Questions

20% 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)

## Insert Interval (medium)

### We'll cover the following

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

### Problem Statement

Given a list of non-overlapping intervals sorted by their start time, **insert a given interval at the correct position** and merge all necessary intervals to produce a list that has only mutually exclusive intervals.

#### Example 1:

```
Input: Intervals=[[1,3], [5,7], [8,12]], New Interval=[4,6]
Output: [[1,3], [4,7], [8,12]]
Explanation: After insertion, since [4,6] overlaps with [5,7], we merged them into one [4,7].
```

#### Example 2:

```
Input: Intervals=[[1,3], [5,7], [8,12]], New Interval=[4,10]
Output: [[1,3], [4,12]]
Explanation: After insertion, since [4,10] overlaps with [5,7] & [8,12], we merged them into [4,12].
```

#### Example 3:

```
Input: Intervals=[[2,3],[5,7]], New Interval=[1,4]
Output: [[1,4], [5,7]]
Explanation: After insertion, since [1,4] overlaps with [2,3], we merged them into one [1,4].
```

### Try it yourself

Try solving this question here:

Java

Python3

JS

C++

```
19 }
20
21 public static void main(String[] args) {
22     List<Interval> input = new ArrayList<Interval>();
23     input.add(new Interval(1, 3));
24     input.add(new Interval(5, 7));
25     input.add(new Interval(8, 12));
26     System.out.print("Intervals after inserting the new interval: ");
27     for (Interval interval : InsertInterval.insert(input, new Interval(4, 6)))
28         System.out.print("[ " + interval.start + ", " + interval.end + " ] ");
29     System.out.println();
30
31     input = new ArrayList<Interval>();
32     input.add(new Interval(1, 3));
33     input.add(new Interval(5, 7));
34     input.add(new Interval(8, 12));
35     System.out.print("Intervals after inserting the new interval: ");
36     for (Interval interval : InsertInterval.insert(input, new Interval(4, 10)))
37         System.out.print("[ " + interval.start + ", " + interval.end + " ] ");
38     System.out.println();
39
40     input = new ArrayList<Interval>();
41     input.add(new Interval(2, 3));
42     input.add(new Interval(5, 7));
43     System.out.print("Intervals after inserting the new interval: ");
44     for (Interval interval : InsertInterval.insert(input, new Interval(1, 4)))
45         System.out.print("[ " + interval.start + ", " + interval.end + " ] ");
46     System.out.println();
47 }
```

RUN

SAVE

RESET

Close

Output

4.073s

Intervals after inserting the new interval:  
Intervals after inserting the new interval:  
Intervals after inserting the new interval:

### Solution

- 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

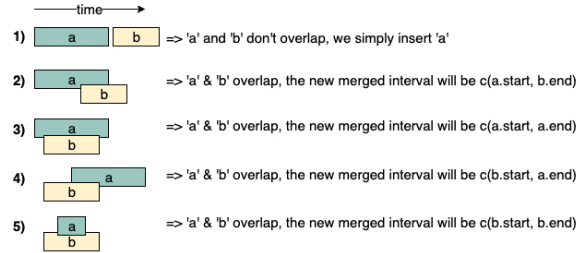
- Introduction
- Order-agnostic Binary Search (easy)
- Ceiling of a Number (medium)
- Next Letter (medium)
- Number Range (medium)
- Search in a Sorted Infinite Array (medium)
- Minimum Difference Element (medium)
- Bitonic Array Maximum (easy)
- Problem Challenge 1
- Solution Review: Problem Challenge 1
- Problem Challenge 2

If the given list was not sorted, we could have simply appended the new interval to it and used the `merge()` function from [Merge Intervals](#). But since the given list is sorted, we should try to come up with a solution better than  $O(N * \log N)$

When inserting a new interval in a sorted list, we need to first find the correct index where the new interval can be placed. In other words, we need to skip all the intervals which end before the start of the new interval. So we can iterate through the given sorted list of intervals and skip all the intervals with the following condition:

```
intervals[i].end < newInterval.start
```

Once we have found the correct place, we can follow an approach similar to [Merge Intervals](#) to insert and/or merge the new interval. Let's call the new interval 'a' and the first interval with the above condition 'b'. There are five possibilities:



The diagram above clearly shows the merging approach. To handle all four merging scenarios, we need to do something like this:

```
c.start = min(a.start, b.start)
c.end = max(a.end, b.end)
```

Our overall algorithm will look like this:

1. Skip all intervals which end before the start of the new interval, i.e., skip all `intervals` with the following condition:

```
intervals[i].end < newInterval.start
```

2. Let's call the last interval 'b' that does not satisfy the above condition. If 'b' overlaps with the new interval (a) (i.e. `b.start <= a.end`), we need to merge them into a new interval 'c':

```
c.start = min(a.start, b.start)
c.end = max(a.end, b.end)
```

3. We will repeat the above two steps to merge 'c' with the next overlapping interval.

## Code

Here is what our algorithm will look like:

Java

Python3

C++

JS

```

1  import java.util.*;
2
3  class Interval {
4      int start;
5      int end;
6
7      public Interval(int start, int end) {
8          this.start = start;
9          this.end = end;
10     }
11 };
12
13 class InsertInterval {
14
15     public static List<Interval> insert(List<Interval> intervals, Interval newInterval) {
16         if (intervals == null || intervals.isEmpty())
17             return Arrays.asList(newInterval);
18
19         List<Interval> mergedIntervals = new ArrayList<>();
20
21         int i = 0;
22         // skip (and add to output) all intervals that come before the 'newInterval'
23         while (i < intervals.size() && intervals.get(i).end < newInterval.start)
24             mergedIntervals.add(intervals.get(i++));
25
26         // merge all intervals that overlap with 'newInterval'
27         while (i < intervals.size() && intervals.get(i).start <= newInterval.end) {
28             newInterval.start = Math.min(intervals.get(i).start, newInterval.start);

```

RUN

SAVE

RESET

Close

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)

Output

Intervals after inserting the new interval: [1, 3][4, 7][8, 12]  
Intervals after inserting the new interval: [1, 3][4, 12]  
Intervals after inserting the new interval: [1, 4][5, 7]

Time complexity

As we are iterating through all the intervals only once, the time complexity of the above algorithm is  $O(N)$ , where 'N' is the total number of intervals.

Space complexity

The space complexity of the above algorithm will be  $O(N)$  as we need to return a list containing all the merged intervals.

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

← Back

Merge Intervals (medium)

✓ MARK AS COMPLETED

Next →

Intervals Intersection (medium)

Report an Issue