57. Insert Interval

<https://leetcode.com/problems/insert-interval/>

1. **Listen**

**Problem Statement:**

You are given an array of non-overlapping intervals intervals where intervals[i] = [starti, endi] represent the start and the end of the ith interval and intervals is sorted in ascending order by starti. You are also given an interval newInterval = [start, end] that represents the start and end of another interval.

Insert newInterval into intervals such that intervals is still sorted in ascending order by startiand intervals still does not have any overlapping intervals (merge overlapping intervals if necessary).

Return intervalsafter the insertion.

**Input:**

1. You are given an array of non-overlapping intervals intervals where intervals[i] = [starti, endi] represent the start and the end of the ith interval and intervals is sorted in ascending order by starti.
2. You are also given an interval newInterval = [start, end] that represents the start and end of another interval.

**Goal:**

Insert newInterval into intervals such that intervals is still sorted in ascending order by starti and intervals still does not have any overlapping intervals (merge overlapping intervals if necessary).

**Return:**

Return intervalsafter the insertion.

1. **Examples**

Example 1:

**Input:** intervals = [[1,3],[6,9]], newInterval = [2,5]

**Output:** [[1,5],[6,9]]

Example 2:

**Input:** intervals = [[1,2],[3,5],[6,7],[8,10],[12,16]], newInterval = [4,8]

**Output:** [[1,2],[3,10],[12,16]]

**Explanation:** Because the new interval [4,8] overlaps with [3,5],[6,7],[8,10].

**Constraints:**

* 0 <= intervals.length <= 104
* intervals[i].length == 2
* 0 <= starti <= endi <= 105
* intervals is sorted by starti in ascending order.
* newInterval.length == 2
* 0 <= start <= end <= 105

**Test Cases:**

* newInterval is first interval (don’t have to merge)
* newInterval is last interval (don’t have to merge)
* newInterval is middle interval (must merge intervals)
* newInterval[starti] == interval[start] or == interval[end]
* newInterval[endi] == interval[start] or == interval[end]

1. **Brute Force**

**Solution 1:**

**Intuition:**

Let’s establish our main goal.

We must insert **newInterval** into **intervals** such that

1. **intervals** is still sorted in ascending order by **starti**

and

1. **intervals** still does not have any overlapping intervals

Often with interval problems, it’s easier to visual this by **drawing a number line**.

At first glance, it seems like there can only be three different cases.

1. Insert **newInterval** at the start of **intervals** (and we don’t merge).

**Input**: intervals = [[3,5],[6,9]], newInterval = [1,2]



1. Insert **newInterval** at the end of **intervals** (and we don’t merge).

**Input**: intervals = [[3,5],[6,9]], newInterval = [10,14]



1. Insert **newInterval** in the middle of **intervals** (and we don’t merge).

**Input**: intervals = [[1,3],[6,9]], newInterval = [4,5]



1. **newInterval** overlaps with single interval in **intervals** (we must merge).

**Input**: intervals = [[1,3],[5,7],[8,10]], newInterval = [4,6]



1. **newInterval** overlaps with multiple intervals in **intervals** (we must merge).

**Input**: intervals = [[1,3],[5,7],[8,10]], newInterval = [4,9]



newInterval could overlap with an indefinite number of intervals.

**Algorithm:**

We must iterate over the sorted intervals in the **intervals** array to find the insertion point.

We place a condition down saying “does the current intervals from the intervals array overlap with newInterval?”

If not, then we add the current interval to a solution list.

If so, we merge the two intervals and add the merged interval to the solution list.

But what would be the condition to merge two intervals?

Let’s look at a simple case:



What conditions can we pull from this?

We would want to merge ethe two intervals if:

1. newInterval[end] > interval[start]



and

1. newInterval[start] < interval[end]

But how do we merge?

1. We take the minimum between the start values
2. We take the maximum between the end values
3. And we create a new interval with the minimum and maximum values as start and end respectively.

Once we have this merged interval, we don’t actually want to add it to the solution list just yet, because we know that this interval could still be overlapping with an indefinite number of additional intervals.

For Example:

Original:



9 > 3

and

1 < 6

After first merge:



After second merge:



Once we finish merging all necessary intervals, we can add the merged interval to the solution list, as well as the rest of intervals in the intervals array since we just solved the constraint for nonoverlapping intervals.

**Time Complexity: O(N)**

Since the intervals array is already sorted in ascending order, we can iterate over each interval in the intervals array once.

Space Complexity: O(1)

Not including the solution set, no additional space is necessary.

1. **Optimize**

No need to optimize.

1. **Walkthrough**
2. **Implement**
3. **Test**