Description

57. Insert Interval

Given a set of non-overlapping intervals, insert a new interval into the intervals (merge if necessary).

You may assume that the intervals were initially sorted according to their start times.

Example 1:
Given intervals [1,3],[6,9], insert and merge [2,5] in as [1,5],[6,9].

Example 2:
Given [1,2],[3,5],[6,7],[8,10],[12,16], insert and merge [4,9] in as [1,2],[3,10],[12,16].

This is because the new interval [4,9] overlaps with [3,5],[6,7],[8,10].

Idea

Because intervals are already sorted, we only need to find out where the newInterval' start and end should be, and merge if necessary. Because it's sorted according to thier start times, we can firstly check newInterval.start. We iterate intervals and find some intervals.get(i), if i.end < new.start, that means those are intervals before newInterval without any overlap, so we put those i directly into our result. Now we have the first i.end >= new.start, they are overlapped, so we choose Math.min(i.start, new.start) as the new.start to merge. And we also need to choose Math.max(i.end, new.end) as the new.end to merge, until we find some i which i.start > new.end. Starting from this i, they are no longer overlapped, so we add the updated/merged newInterval to the result, and add the rest of the non-overlapped intervals into the result.

Java

```java
class Solution {
  public List<Interval> insert(List<Interval> intervals, Interval newInterval) {
    List<Interval> res = new ArrayList<>();
    int i = 0;
    while (i < intervals.size() && intervals.get(i).end < newInterval.start) {
        res.add(intervals.get(i++));
    }

    while (i < intervals.size() && newInterval.end >= intervals.get(i).start) {
        newInterval.start = Math.min(newInterval.start, intervals.get(i).start);
        newInterval.end = Math.max(newInterval.end, intervals.get(i).end);
        i++;
    }

    res.add(newInterval);

    while (i < intervals.size()) {
        res.add(intervals.get(i++));
    }
    return res;
  }
}
```

C++

```
class Solution {
public:
    vector<Interval> insert(vector<Interval>& intervals, Interval newInterval) {
        vector<Interval> res;
        int i = 0;
        while (i < intervals.size() && intervals[i].end < newInterval.start) {
            res.push_back(intervals[i++]);
        }

        while (i < intervals.size() && intervals[i].start <= newInterval.end) {
            newInterval.start = min(newInterval.start, intervals[i].start);
            newInterval.end = max(newInterval.end, intervals[i].end);
            i++;
        }

        res.push_back(newInterval);

        while (i < intervals.size()) {
            res.push_back(intervals[i++]);
        }

        return res;
    }
};
```

Summary

- Discuss different situations and code accordingly.
- Reduce the problem into a sequence of sub problems.
- Pure implementation problem.