Description

56. Merge Intervals

Given a collection of intervals, merge all overlapping intervals.

```
For example,
Given [1,3],[2,6],[8,10],[15,18],
return [1,6],[8,10],[15,18].
```

Idea

The primitive idea is to select any two intervals and merge then together, we need a nested interation to do that, so the time complexity would be O(n^2). How do we speed up? We can make one side of the intervals to be ordered, and compare the other side. We can sort the intervals by start. Then we check the intervals next to each other, say i1, i2. Because it's already sorted, we know i1.start <= i2.start. So if i1.end < i2.start, we know i1 and i2 have no overlap, we can put i1 into result list. Otherwise, it means there is an overlap between i1 and i2, we can do the merge. The merged i.start is i1.start because we know i1.start <= i2.start, and merged i.end is the maximum between i1.end and i2.end. The sort will take O(nlogn), and merge will take O(n), so

Java

```
class Solution {
    public List<Interval> merge(List<Interval> intervals) {
        if (intervals == null || intervals.size() == 0) {
            return intervals;
        }

        Collections.sort(intervals, (a, b) -> a.start - b.start);
        List<Interval> res = new ArrayList<>();
        Interval prev = intervals.get(0);

        for (int i = 1; i < intervals.size(); i++) {
            Interval cur = intervals.get(i);
```

```
            if (cur.start > prev.end) {
                res.add(prev);
                prev = cur;
            } else {
                prev.end = Math.max(prev.end, cur.end);
            }
        }
        res.add(prev);
        return res;
    }
}
```

C++

```
class Solution {
public:
    vector<Interval> merge(vector<Interval>& intervals) {
        if (intervals.empty()) {
            return intervals;
        }
        vector<Interval> res;
        sort(intervals.begin(), intervals.end(),
             [](Interval a, Interval b){return a.start < b.start;});
        Interval pre = intervals[0];
        for (int i = 1; i < intervals.size(); i++) {
            Interval cur = intervals[i];
            if (cur.start > pre.end) {
                res.push_back(pre);
                pre = cur;
            } else {
                pre.end = max(pre.end, cur.end);
            }
        }
        res.push_back(pre);

        return res;
    }
};
```

Summary

- To speed up a nested loop, we can fix outter loop. Inorder to fix outter loop, we may want to sort.