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]`.

Solution 1

The idea is to sort the intervals by their starting points. Then, we take the first interval and compare its end with the next intervals starts. As long as they overlap, we update the end to be the max end of the overlapping intervals. Once we find a non overlapping interval, we can add the previous "extended" interval and start over.

Sorting takes O(n log(n)) and merging the intervals takes O(n). So, the resulting algorithm takes O(n log(n)).

I used an anonymous comparator and a for-each loop to try to keep the code clean and simple.

```
public List<Interval> merge(List<Interval> intervals) {
    if (intervals.size() <= 1)
        return intervals;

    // Sort by ascending starting point using an anonymous Comparator
    Collections.sort(intervals, new Comparator<Interval>() {
        @Override
        public int compare(Interval i1, Interval i2) {
            return Integer.compare(i1.start, i2.start);
        }
    });

    List<Interval> result = new LinkedList<Interval>();
    int start = intervals.get(0).start;
    int end = intervals.get(0).end;

    for (Interval interval : intervals) {
        if (interval.start <= end) // Overlapping intervals, move the end if need
ed
            end = Math.max(end, interval.end);
        else {                     // Disjoint intervals, add the previous one an
d reset bounds
            result.add(new Interval(start, end));
            start = interval.start;
            end = interval.end;
        }
    }

    // Add the last interval
    result.add(new Interval(start, end));
    return result;
}
```

written by brubru777 original link here

Solution 2

```
public class Solution {
    public List<Interval> merge(List<Interval> intervals) {
        Collections.sort(intervals, new Comparator<Interval>(){
            @Override
            public int compare(Interval obj0, Interval obj1) {
                return obj0.start - obj1.start;
            }
        });

        List<Interval> ret = new ArrayList<>();
        Interval prev = null;
        for (Interval inter : intervals) {
            if (  prev==null || inter.start>prev.end ) {
                ret.add(inter);
                prev = inter;
            } else if (inter.end>prev.end) {
                // Modify the element already in list
                prev.end = inter.end;
            }
        }
        return ret;
    }
}
```

written by danchou original link here

Solution 3

Just go through the intervals sorted by start coordinate and either combine the current interval with the previous one if they overlap, or add it to the output by itself if they don't.

```
def merge(self, intervals):
    out = []
    for i in sorted(intervals, key=lambda i: i.start):
        if out and i.start <= out[-1].end:
            out[-1].end = max(out[-1].end, i.end)
        else:
            out += i,
    return out
```

written by StefanPochmann original link here