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]`.

Solution 1

Hi guys!

Here's a pretty straight-forward and concise solution below.

```
public List<Interval> insert(List<Interval> intervals, Interval newInterval) {
    List<Interval> result = new LinkedList<>();
    int i = 0;
    // add all the intervals ending before newInterval starts
    while (i < intervals.size() && intervals.get(i).end < newInterval.start)
        result.add(intervals.get(i++));
    // merge all overlapping intervals to one considering newInterval
    while (i < intervals.size() && intervals.get(i).start <= newInterval.end) {
        newInterval = new Interval( // we could mutate newInterval here also
                Math.min(newInterval.start, intervals.get(i).start),
                Math.max(newInterval.end, intervals.get(i).end));
        i++;
    }
    result.add(newInterval); // add the union of intervals we got
    // add all the rest
    while (i < intervals.size()) result.add(intervals.get(i++));
    return result;
}
```

Hope it helps.

written by shpolsky original link here

Solution 2

Solution 1: (7 lines, 88 ms)

Collect the intervals strictly left or right of the new interval, then merge the new one with the middle ones (if any) before inserting it between left and right ones.

```
def insert(self, intervals, newInterval):
    s, e = newInterval.start, newInterval.end
    left = [i for i in intervals if i.end < s]
    right = [i for i in intervals if i.start > e]
    if left + right != intervals:
        s = min(s, intervals[len(left)].start)
        e = max(e, intervals[~len(right)].end)
    return left + [Interval(s, e)] + right
```

Solution 2: (8 lines, 84 ms)

Same algorithm as solution 1, but different implementation with only one pass and explicitly collecting the to-be-merged intervals.

```
def insert(self, intervals, newInterval):
    s, e = newInterval.start, newInterval.end
    parts = merge, left, right = [], [], []
    for i in intervals:
        parts[(i.end < s) - (i.start > e)].append(i)
    if merge:
        s = min(s, merge[0].start)
        e = max(e, merge[-1].end)
    return left + [Interval(s, e)] + right
```

Solution 3: (11 lines, 80 ms)

Same again, but collect and merge while going over the intervals once.

```
def insert(self, intervals, newInterval):
    s, e = newInterval.start, newInterval.end
    left, right = [], []
    for i in intervals:
        if i.end < s:
            left += i,
        elif i.start > e:
            right += i,
        else:
            s = min(s, i.start)
            e = max(e, i.end)
    return left + [Interval(s, e)] + right
```

written by StefanPochmann original link here

Solution 3

```
vector<Interval> insert(vector<Interval>& intervals, Interval newInterval) {
    vector<Interval> ret;
    auto it = intervals.begin();
    for(; it!=intervals.end(); ++it){
        if(newInterval.end < (*it).start) //all intervals after will not overlap
with the newInterval
            break;
        else if(newInterval.start > (*it).end) //*it will not overlap with the ne
wInterval
            ret.push_back(*it);
        else{ //update newInterval bacause *it overlap with the newInterval
            newInterval.start = min(newInterval.start, (*it).start);
            newInterval.end = max(newInterval.end, (*it).end);
        }
    }
    // don't forget the rest of the intervals and the newInterval
    ret.push_back(newInterval);
    for(; it!=intervals.end(); ++it)
        ret.push_back(*it);
    return ret;
}
```

My question is why this code need 500ms !?

written by Erudy original link here