Daily Coding Problem #119

Problem

This problem was asked by Google.

Given a set of closed intervals, find the smallest set of numbers that covers all the intervals. If there are multiple smallest sets, return any of them.

For example, given the intervals `[0, 3], [2, 6], [3, 4], [6, 9]`, one set of numbers that covers all these intervals is `{3, 6}`.

Solution

This problem bcomes clearer if we sort the intervals by the starting points. For example, intervals `[[10, 20], [1, 6], [3, 8], [7, 12]]` should become `[[1, 6], [3, 8], [7, 12], [10, 20]]`.

Now, to cover the first interval, `[1, 6]`, we must pick a number in between the interval. However, if the next interval intersects, then we can solve an easier interval problem of picking a point between their intersection. This would let us use 1 less point to cover the intervals. Then, we can look at the third intersection and so forth to find the first k intervals which all intersect. Once we find an interval that doesn't intersect, we can pick a point in the intersection of all the previous intervals. Then we repeat the process starting from the current interval.

In the above example, the intersection of `[[1, 6], [3, 8]]` is `[3, 6]` while the intersection of `[7, 12], [10, 20]` is `[10, 12]`.

```
def covering(intervals):
    intervals.sort(key=lambda x: x[0])

    result = []
    i = 0

    while i < len(intervals):
        interval = intervals[i]

        while i < len(intervals) and intersecting(intervals[i], interval):
            interval = (max(intervals[i][0], interval[0]), min(intervals[i][1],
interval[1]))
            i += 1

        result.append(interval[1])
    return result

def intersecting(x, y):
    return not (x[0] > y[1] or y[0] > x[1])
```

The main while loop takes O(n) since we iterate through the intervals, and sorting the interval takes O(n log n), so this takes O(n log n) time.