Daily Coding Problem #21

Problem

This problem was asked by Snapchat.

Given an array of time intervals (start, end) for classroom lectures (possibly overlapping), find the minimum number of rooms required.

For example, given [(30, 75), (0, 50), (60, 150)], you should return 2.

Solution

First, notice that the minimum number of classroom halls is the maximum number of overlapping intervals.

Now let's consider the naive approach. We could go through each interval and check every other interval and see if it overlaps, keeping track of the largest number of overlapping intervals.

```
def overlaps(a, b):
    start_a, end_a = a
    start_b, end_b = b
    # It doesn't overlap if it's like this:
    #   |start_a .... end_a|  <---> |start_b ... end_b|
    # or like this:
    #   |start_b .... end_b|  <---> |start_a ... end_a|
    # so return not or either of these
    return not (end_a < start_b or start_a > end_b)

def max_overlapping(intervals):
    current_max = 0
    for interval in intervals:
        num_overlapping = sum(overlaps(interval, other_interval)
            for other_interval in intervals
            if interval is not other_interval)
        current_max = max(current_max, num_overlapping)
    return current_max
```

This would take O(n^2) time, since we're checking each interval pairwise. Can we do any better?

One solution is to extract the start times and end times of all the intervals and sort them. Then we can start two pointers on each list, and consider the following:

- If the current start is before the current end, then we have a new overlap. Increment the start pointer.
- If the current start is after the current end, then our overlap closes. Increment the end pointer.

All that's left to do is keep a couple variables to keep track of the maximum number of overlaps we've seen so far and the current number of overlaps.

```
def max_overlapping(intervals):
    starts = sorted(start for start, end in intervals)
    ends = sorted(end for start, end in intervals)

    current_max = 0
    current_overlap = 0
    i, j = 0, 0
    while i < len(intervals) and j < len(intervals):
        if starts[i] < ends[j]:
            current_overlap += 1
            current_max = max(current_max, current_overlap)
            i += 1
        else:
            current_overlap -= 1
            j += 1
    return current_max
```

This runs in O(n log n) time, since we have to sort the intervals.