Daily Coding Problem #200

Problem

This problem was asked by Microsoft.

Let X be a set of n intervals on the real line. We say that a set of points P "stabs" X if every interval in X contains at least one point in P. Compute the smallest set of points that stabs X.

For example, given the intervals `[(1, 4), (4, 5), (7, 9), (9, 12)]`, you should return `[4, 9]`.

Solution

One naive solution would be to evaluate every possible subset of start points and endpoints. In other words, we would first generate all subsets of points from the interval boundaries. Then, beginning with the smaller subsets and proceeding up to the larger ones, check each one to see if it intersects every interval at some point. Once a satisfactory subset is found, we return it.

Here is how this might look:

```
from itertools import combinations

def get_points(intervals):
    points = set([element for pair in intervals for element in pair])

    subsets = []
    for size in range(1, len(points)):
```

```
            subsets.extend(list(combinations(points, size)))

    for start, end in intervals:
        for subset in subsets:
            # Is there any point that is between the start and end of all
intervals?
            if all(any(start <= point <= end for point in subset)
                    for start, end in intervals):
                return subset
```

However, this is horribly inefficient. We have $O(2^N)$ subsets, and for each one we are matching its points against every interval start point and endpoint, so this will be $O(2^N * N^2)$.

To gain some insight, let's take a look at the following intervals: [(1, 4), (4, 4), (3, 9)]. It is clear that if we choose 4, we can satisfy every interval. So we might guess that choosing the earliest endpoint is an optimal strategy. Can we prove this?

In fact, yes! First, we know that we must choose a point less than or equal to the earliest endpoint. Otherwise, the interval that contains that endpoint will never get stabbed. On the other hand, we know that that endpoint is at least as good as any lesser point in that interval, because it intersects every interval that the lesser point does, and potentially others. For instance, in the example above, if we chose 3 instead of 4, we would not have been able to catch (4, 4).

This shows that an optimal greedy strategy exists. First, we sort the intervals by ascending endpoint. Then, as we traverse the list, whenever we reach an interval that is not stabbed by any points in our solution, we take its endpoint and add it to our solution. Since the intervals are sorted, we need only consider the most recently added endpoint to determine if there is an intersection.

```
def get_points(intervals):
    intervals.sort(key=lambda x: (x[1], x[0]))

    points = []
    latest_endpoint = None

    for start, end in intervals:
        if start <= latest_endpoint:
            continue
        else:
```

```
            points.append(end)
            latest_endpoint = end

    return points
```

Sorting the intervals is O(n * log n), and we only pass through the list once, so the time complexity is O(n * log n).
