CS 374 **Spring 2015**

Homework 7

Nikhil Unni (nunni2)

3. Now we can use a greedy algorithm to calculate the minimum number of intervals.

```
Sort p by increasing x_i;
Sort I by increasing b_j for [a_j, b_j] = I_j;
out ← ∅;
i ← 0;
j ← 0;
while i < ||p|| do
    k ← j;
    while k < ||I|| and I_{k+1} contains p_i do
        k ← k + 1
    end
    out ← I_k ∪ out;
    while I_j does not contain p_{i+1} do
        j ← j + 1;
    end
    while I_k contains p_i do
        i ← i + 1;
    end
end
return out;
```

Algorithm 1: Greedy minimum interval cover

The algorithm basically just sorts the points and the intervals by their endpoints, and, starting from the left-most point, greedily finds the interval that contains the point with the farthest endpoint. After it's found this "longest interval" it updates the point index accordingly, taking out the left-most points that are also included in this long interval.

The algorithm is worst-case $O(\|I\|\|p\|)$.

We can prove that this greedy solution will always yield the least number of intervals. Suppose, at each vertex, starting from the left-most, we picked an interval containing this point that was **not** the farthest reaching (largest b_j). This means that there might be an interval that contains more points than the interval that we chose. This local bad choice is necessarily a global bad choice because, moving forward to the point that was included by the best local choice, we have the option to remove our original choice and use its local best choice, and now we have coverage of the old point, the current point, and any point inbetween for the same price as our original bad local decision.
Inductively, the local good choices add up to the best possible global solution.