Introduction to Algorithms	**Problem Set 2**
CS 482 Spring 2000	**Due February 11, 2000**

(1) Here's a problem that's closely related to the *k-center problem* from class — however, the goal here is to cover points using a variable number of centers of fixed radius; and everything takes place in one dimension. It turns out that the one-dimensional nature of the problem makes it possible to solve the problem exactly (rather than just approximately) in polynomial time.

To be concrete: Let's consider a long, quiet country road with houses scattered very sparsely along it. (We can picture the road as a long line segment, with an eastern endpoint and a western endpoint.) Further, let's suppose that despite the bucolic setting, the residents of all these houses are avid cell phone users. You want to place cell phone base stations at certain points along the road, so that every house is within 4 miles of one of the base stations.

Give an efficient algorithm that achieves this goal, using as few base stations as possible.

(2) Your friend Raj is a camp counselor, and he is in charge of organizing activities for a set of junior-high-school-age campers. One of his plans is the following mini-triathalon exercise: each contestant must swin 20 laps of a pool, then bike 10 miles, then run 3 miles. The plan is to send the contestants out in a staggered fashion, via the following rule: the contestants must use the pool one at a time. In other words, first one contestant swins the 20 laps, gets out, and starts biking. As soon as this first person is out of the pool, a second contestant begins swimming the 20 laps; as soon as he/she's out and starts biking, a third contestant begins swimming ... and so on.)

Each contestant has a projected *swimming time* (the expected time it will take him or her to complete the 20 laps), a projected *biking time* (the expected time it will take him or her to complete the 10 miles of bicycling), and a projected *running time* (the time it will take him or her to complete the 5 miles of running. Raj wants to decide on a *schedule* for the triathalon: an order in which to sequence the starts of the contestants. Let's say that the *completion time* of a schedule is the earliest time at which all contestants will be finished with all three legs of the triathalon, assuming they each spend exactly their projected swimming, biking, and running times on the three parts.

What's the best order for sending people out, if one wants the whole competition to be over as early as possible? More precisely, give an efficient algorithm that produces a schedule whose completion time is as small as possible.

(3) *Hidden surface removal* is a problem in computer graphics that scarcely needs an introduction — when Woody is standing in front of Buzz you should be able to see Woody but not Buzz; when Buzz is standing in front of Woody, ... well, you get the idea.

The magic of hidden surface removal is that you can often compute things faster than your intuition suggests. Here's a clean geometric example to illustrate a basic speed-up that can be achieved. You are given n non-vertical lines in the plane, labeled $L_1, \ldots, L_n$, with the i^{th} line specified by the equation $y = a_i x + b_i$. We will make the assumption that no three of the lines all meet at a single point. We say line L_i is *uppermost* at a given x-coordinate x_0 if its y-coordinate at x_0 is greater than the y-coordinates of all the other lines at x_0: $a_i x_0 + b_i > a_j x_0 + b_j$ for all $j \neq i$. We say line L_i is *visible* if there is some x-coordinate at which it is uppermost — intuitively, some portion of it can be seen if you look down from "$y = \infty$."

Give an algorithm that takes n lines as input, and in $O(n \log n)$ time returns all of the ones that are visible. The figure on the back gives an example.

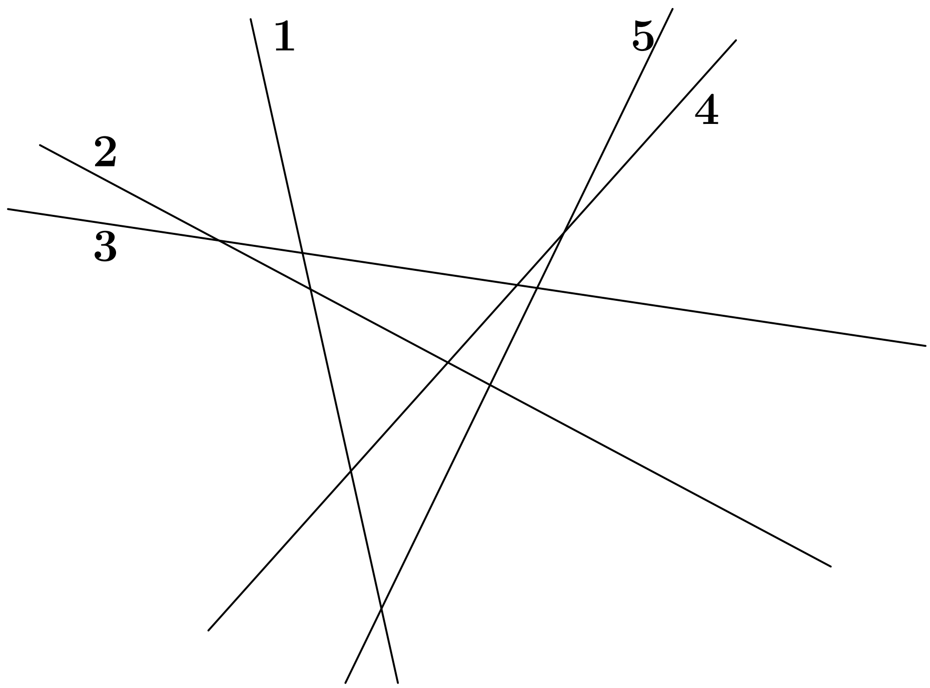

Figure 1: An instance with five lines (labeled “1”–“5” in the figure). All the lines except for “2” are visible.