Algorithms Winter 2007
Take Home Exam Instructor: Anna Karlin
Due on **March 7**.

Instructions: This is a take home exam with the following rules and instructions:

- The work you turn in should be entirely your own. You are not allowed to collaborate or discuss the problems, solutions, or any aspect relating to this exam with your classmates, or anyone else, except the course staff. If you are having any difficulty understanding any of the questions or think something is ambiguous, come talk to or send an email to the course staff.
- Please refer to the grading guidelines sheet and make sure you follow the guidelines laid out there. You do not need to prove anything that is proven in the book or in class. When you refer to something in the book, a page number is useful.
- Devote sufficient time for writing your solutions in a clear manner. We are looking for correctness, precision and clarity of exposition.

Questions:

1. Your friend is working as a camp counselor, and he is in charge of organizing activities for a set of campers. One of his plans is the following mini-triathlon exercise: each contestant must swim 20 laps in 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 swims 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.) Biking and running can be done in parallel.

   Each contestant, say the ith, has a projected *swimming time* s_i (the expected time it will take him or her to complete the 20 laps), a projected *biking time* b_i (the expected time it will take him or her to complete the 10 miles of bicycling), and a projected *running time* r_i (the expected time it will take him or her to complete the 3 miles of running). Your friend wants to decide on a schedule for the triathlon: 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 in which all contestants will be finished wiht all three legs of the triathlon, 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 soon as possible? More precisely, give a greedy algorithm that produces a schedule whose completion time is as small as possible. Prove the correctness of your algorithm and analyze its running time.

   Example: 2 contestants, with $s_1 = 10$, $b_1 = 5$, $r_1 = 10$, and $s_2 = 1$, $b_1 = 20$, $r_1 = 30$. The best schedule is to send the second contestant first, for a completion time of 51.

2. Recall the problem of finding the number of inversions in an array (Section 5.3). Recall that we are given a sequence of n numbers $a_1, \ldots, a_n$, which we assume are all distinct, and we define an inversion to be a pair $i < j$ such that $a_i > a_j$.

We motivated the problem of counting inversions as a good measure of how different two orderings are. However, one might feel that this measure is too sensitive. Let's call a pair a *significant inversion* if $i < j$ and $a_i > 2a_j$. Give an $O(n \log n)$ algorithm to count the number of significant inversions. Prove the correctness of your algorithm and analyze its running time.

3. You're helping to run a high-performance computing system capable of processing several terabytes of data per day. For each of the n days, you're presented with a quantity of data: on day i, you're presented with x_i terabytes. For each terabyte you process, you receive a fixed revenue, but any unprocessed data becomes unavailable at the end of the day (i.e., you can't work on it on any future day).

   You can't always process everything each day because you're constrained by the capabilities of your computing system, which can only process a fixed number of terabytes in a given day. In fact, it's running some one-of-a-kind software that, while very sophisticated, is not totally reliable, and so the amount of data you can process goes down with each day that passes since the most recent reboot of the system. On the first day after a reboot, you can process s_1 terabytes, on the second day s_2 and so on, up to s_n. We assume that $s_1 > s_2 > s_3 > \ldots > s_n > 0$. (Of course, on day i, you can only process up to x_i terabytes, regardless of how fast your system is.) To get the system back to peak performance, you can choose to reboot it; but on any day you choose to reboot the system, you can't process any data at all.

   The problem: Given the amounts of available data $x_1, x_2, \ldots, x_n$ for the next n days, and given the profile of your system as expressed by $s_1, s_2, \ldots, s_n$ (and starting from a freshly rebooted system on day 1), choose the days on which you're going to reboot so as to maximize the total amount of data you process.

   Example: Suppose $n = 4$, that $(x_1, x_2, x_3, x_4) = (10, 1, 7, 7)$ and $(s_1, s_2, s_3, s_4) = (8, 4, 2, 1)$.

   Then the best solution is to reboot on day 2 only. In this way, you process 8 terabytes on day 1, then 0 on day 2, then 7 on day 3, then 4 on day 4, for a total of 19. (Note that if you didn't reboot at all, you'd process $8 + 1 + 2 + 1 = 12$, and other rebooting strategies give you less than 19 as well.

   (a) Give an example of an instance with the following properties:
   - There is a "surplus" of data in the sense that $x_i > s_1$ for every i.
   - The optimal solution reboots the system at least twice.

   In addition to the example, you should say what the optimal solution is. You do not need to provide a proof that it is optimal.

   (b) Give a polynomial time dynamic programming algorithm that takes values for $x_1, \ldots, x_n$ and $s_1, s_2, \ldots, s_n$ and returns the total *number* of terabytes processed by the optimal solution, and the days on which reboots should occur. Prove the correctness of your algorithm and analyze its running time.

4. Suppose you're acting as a consultant for the port authority of a small Pacific Rim nation. They're currently doing a multi-billion-dollar business per year, and their revenue is constrained almost entirely by the rate at which they can unload ships that arrive in the port.

Handling hazardous materials adds additional complexity to what is, for them, an already complicated task. Suppose a convoy of ships arrives in the morning and delivers a total of n cannisters, each containing a different kind of hazardous material. Standing on the dock is a set of m trucks, each of which can hold up to k containers.

Here are two related problems, which arise from different types of constraints that might be placed on the handling of the hazardous materials. For each of the two problems, either give a polynomial time algorithm to solve it (complete with proof of complexity and analysis of running time) or give a proof that it is NP-complete (Suggestion: 3-Coloring).

(a) For each cannister, there is a specified subset of the trucks in which it may be safely carried. Is there a way to load all n cannisters into the m trucks so that no truck is overloaded, and each container goes in a truck that is allowed to carry it?

(b) In this different version of the problem, any cannister can be placed in any truck; however, there are certain pairs of cannisters that cannot be placed together in the same truck. (The chemicals they contain may react explosively if brought into contact.) Is there a way to load all n cannisters into the m trucks so that no truck is overloaded, and no two cannisters are placed in the same truck when they are not supposed to be?