Question 1. (20 points) Recall that, in the activity-selection problem that we covered in class, we had a single copy of a resource and the problem was to use it optimally (i.e., to satisfy as many of the n requests as possible). In this question you are asked to design an algorithm that, given the n requests as input, computes in $O(n \log n)$ time the smallest number of copies of the resource that sastisfy *all of the n requests*. Note that buying n copies of the resource would satisfy all requests, but would be wasteful if they can be satisfied by buying fewer than n copies.

The ith request is described by a pair of numbers (s_i, f_i) where s_i is the desired start time (the time at which a copy of the resource is needed) and f_i is the finish time, $s_i \leq f_i$. Your solution should not depend on assuming $f_i - s_i$ to be integer, or to be small (i.e., the start and finish times can be of type `float`, or they can be arbitrarily large integers).

Question 2. (20 points) Given a sequence $A = a_1 \dots a_n$ of n distinct symbols, an *increasing subsequence* of A is a subsequence whose elements are increasing in left-to-right order. Suppose that you are given software that computes a longest common subsequence (LCS) of any two input sequences X and Y. Explain how that software can be used to solve the problem of finding a longest increasing subsequence (LIS) of an input sequence X. Your solution is supposed to use the LCS software without any modification of its internals: You can only control the inputs to it, and make use of what it returns.

Question 3. (15 points) The matrix-chain multiplication problem was explained in class, and a dynamic programming algorithm for it was presented (here we refer to the problem of computing only the optimal cost, rather than an actual solution that achieves it). Recall that the algorithm presented in class works by filling the entries of a table in a certain order. Show what the table would look like after you fill all its entries when the input to the problem is $8, 3, 2, 19, 18, 7$, i.e., there are five matrices, the first of which is an 8×3 matrix, the second a 3×2, etc.

Question 4. (25 points) Let $S = \{I_1, \dots, I_n\}$ be a set of intervals where $I_i = [l_i, r_i]$, $l_i < r_i$. The intervals are given sorted according to their r_is (hence $r_1 < r_2 < \dots < r_n$). For simplicity, we assume that no two of the l_is and r_is coincide (i.e., there are $2n$ distinct interval endpoints). Each interval has a positive weight w_i (that may be quite different from the length $r_i - l_i$ of that interval). The weight of a subset of S is the sum of the weights of the intervals in that subset. A subset of S is *acceptable* if none of its intervals overlap. Give an $O(n^2)$ time algorithm for computing the maximum weight of an acceptable subset of S. You need only compute the maximum possible weight, not the subset that achieves it.

Hint. Use dynamic programming.

Comment. An $O(n \log n)$ time solution exists but you are not responsible for it.

Question 5. (20 points) Suppose you have m workers $W_1, \dots, W_m$ and n tasks $T_1, \dots, T_n$. You also have a sequence S of pairs of the form (W_i, T_j) whose meaning is "worker W_i is

qualified for task T_j"; the pairs in S are not given in any particular order. A worker can be qualified for many tasks, but can be assigned to only one task. A task can have many workers who are qualified to do it, but no more than one worker can be assigned to it. Consider the following algorithm for assigning workers to tasks.

1. Mark all workers and tasks as being "available".
2. Go through the pairs in S in the order of their appearance in S: For each such pair (W_i, T_j), if both W_i and T_j are marked "available" then assign worker W_i to task T_j and mark both W_i and T_j as being "unavailable" (otherwise ignore that pair and move on to the next pair in S).

Let M be the number of worker-to-task assignemnts produced by the above algorithm, and let Opt be the number of worker-to-task assignemnts in an optimal solution (which is a solution that maximizes the number of worker-to-task assignments for input S, i.e., it minimizes the number of unassigned workers and tasks). Answer the following questions.

1. Give an example for which $M < Opt$. That is, you have to exhibit an input S for which the above algorithm fails to provide an optimal solution. Make sure you state what M and Opt are for the S you used.
2. Prove that $M \geq Opt/2$.

Date due: Tuesday September 24, 2013