For all homework problems where you are asked to give an algorithm, you must prove the correctness of your algorithm and establish the best upper bound that you can give for the running time. You should always write a clear informal description of your algorithm in English. You may also write pseudocode if you feel your informal explanation requires more precision and detail. As always, try to make your answers as clear and concise as possible.

1. The *risk-free currency exchange problem* offers a risk-free way to make money. Suppose we have currencies $c_1, \ldots, c_n$. (For example, c_1 might be dollars, c_2 rubles, c_3 yen, etc.) Some pairs of currencies (not necessarily all of them) can be exchanged. If two currencies c_i and c_j have an exchange rate $r_{i,j}$, then you can exchange one unit of c_i for $r_{i,j}$ units of c_j. Note that if $r_{i,j} \cdot r_{j,i} > 1$, then you can make money simply by trading units of currency i into units of currency j and back again. This almost never happens, but occasionally (because the updates for exchange rates do not happen quickly enough) for very short periods of time exchange traders can find a sequence of trades that can make risk-free money. That is, if there is a sequence of currencies $c_{i_1}, c_{i_2}, \ldots, c_{i_k}$ such that $r_{i_1,i_2} \cdot r_{i_2,i_3} \cdots r_{i_{k-1},i_k} \cdot r_{i_k,i_1} > 1$, then trading one unit of c_{i_1} into c_{i_2} and trading that into c_{i_3} and so on will yield a profit.

   Design an efficient algorithm to detect if a risk-free currency exchange exists. (You need not actually find it.)

2. You are given a graph $G = (V, E)$ and a minimum spanning tree T. The weight of one of the edges in T is increased. Give a linear time algorithm that determines the new MST.

3. Give a family of set cover problems where the set to be covered has n elements, the minimum set cover is size $k = 3$, and the greedy algorithm returns a cover of size $\Omega(\log n)$. That is, you should give a description of a set cover problem that works for a set of values of n that grows to infinity – you might begin, for example, by saying, "Consider the set $X = \{1, 2, \ldots, 2^b\}$ for any $b \geq 10$, and consider subsets of X of the form...", and finish by saying "We have shown that for the example above, the set cover returned by the greedy algorithm is of size $b = \Omega(\log n)$." (Your actual wording may differ substantially, of course, but this is the sort of thing we're looking for.) Explain briefly how to generalize your construction for other (constant) values of k. (You need not give a complete proof of your generalization, but explain the types of changes needed from the case of $k = 3$.)

4. Consider the following scheduling problem: we have two machines, and a set of jobs $j_1, j_2, j_3, \ldots, j_n$ that we have to process one at a time. To process a job, we place it on a machine. Each job j_i has an associated running time r_i. The load on the machine is the sum of the running times of the jobs placed on it. The goal is to minimize the completion time, which is the maximum load over all machines.

   Suppose we adopt a greedy algorithm: each job j_i is put on the machine with the minimum load after the first $i - 1$ jobs. (Ties can be broken arbitrarily.) Show that this strategy yields a completion time within a factor of 3/2 of the best possible placement of jobs. (Hint: Think of the best possible placement of jobs. Even for the best placement, the completion time is at least as big as the biggest job, and at least as big as half the sum of the jobs. You may want to use both of these facts.) Give an example where a factor of 3/2 is achieved.

   Suppose now instead of 2 machines we have m machines. What is the performance of the greedy solution, compared to the optimal, as a function of m? Give a family of examples (that is, one for each m – if they are very similar, it will be easier to write down!) where the factor separating the optimal and the greedy solutions is as large as you can make it.

5. Consider an algorithm for integer multiplication of two n-digit numbers where each number is split into three parts, each with $n/3$ digits.

   (a) Design and explain such an algorithm, similar to the integer multiplication algorithm presented in class. Your algorithm should describe how to multiply the two integers using only six multiplications on the smaller parts (instead of the straightforward nine).

   (b) Determine the asymptotic running time of your algorithm. Would you rather split it into two parts or three parts?

   (c) Suppose you could use only five multiplications instead of six. Then determine the asymptotic running time of such an algorithm. In this case, would you rather split it into two parts or three parts?

   (d) Challenge problem– this is optional, and not worth any points. Solving it will simply impress the instructors. Find a way to use only five multiplications on the smaller parts. Can you generalize to when the two initial

n-digit numbers are split into k parts, each with n/k digits? Hint: also consider multiplication by a constant, such as 2; note that multiplying by 2 does not count as one of the five multiplications. You may need to use some linear algebra.