In preparing your solutions to the exam, you are **allowed to use any sources** including textbooks, other students and professors, previous homeworks and solutions, or any sources on the Internet. The only source you may *not* use is me. Of course, I am happy to answer general questions, go over homework problems, or answer clarifying questions about exam problems.

As usual, to "design and analyze" an algorithm means to (a) **describe** the algorithm, (b) **prove/justify** its correctness, and (c) **analyze** its asymptotic running time. Full credit will only be given for the most efficient possible algorithms. Algorithms must be clearly explained (using pseudocode if appropriate) in sufficient detail that another student could take your description and turn it into working code. You may **freely cite any theorems proved in class** (without proof), or **use algorithms covered in class as subroutines**.

The exam will take place on Monday, November 18, in class. You should bring something to write with, but no other external resources are allowed. I will provide fresh copies of the exam as well as blank paper. If you have accommodations that you wish to make use of, let me know soon so we can make appropriate arrangements.

Question 1. You are working IT support for the 3948th Intergalactic Linguistics Convention (ILC). Since so many different languages are spoken by the attendees, translators are used to allow people to communicate; each translator knows two languages and can translate back and forth between them. For example, if an English speaker and a Klingon speaker want to have a conversation, they could find a translator who knows both languages. Sometimes multiple translators might even be needed. For example, the same two people could also communicate if they could find an English-Japanese translator, a Japanese-Xzcvrbl translator, and a Xzcvrbl-Klingon translator.

For the purposes of this problem we will not worry about the fact that messages tend to be garbled by repeated translation; the ILC translators are so good that this does not happen.

Due to the high number of attendees and different languages they speak (last year's convention had 6.8 trillion attendees speaking 3 trillion different languages) finding translators can be difficult. This is where you come in! You are developing an app that attendees can use to help them find appropriate translators.

Before the convention, you will be given a list of all the languages that might be spoken by attendees. During the convention, the app needs to support two operations: first, as translators arrive, they can use the app to check in and list the two languages they speak. Second, attendees can query the app by selecting two languages, and the app should tell them whether they can communicate (based on what translators have checked in so far), and if so, what translators they could use. You may assume that the app will be able to communicate with a central server cluster[1] where you can store any data structures you wish.

[1] 4096 networked Cray-Z 5e Pro XT servers, each with ten 80-core GPUs (Galactic Processing Units) running at 5.2 bogomips, with an attached 512 zettabyte storage array

(a) Let L be the number of languages spoken, and T the number of translators who have checked in so far. Given two languages as input, explain how, in $\Theta(L + T)$ time, you can either find a sequence of intermediate languages

that can be translated to allow communication between the two input languages, or report that communication is impossible.

(b) Even $\Theta(L+T)$ can be a long time when L and T are very large; it turns out that your app wastes a lot of time trying to find a way to communicate between languages where the ultimate answer is that it is impossible. So as a first step, you want to have it simply check whether there is any way for two people to communicate at all, before doing a more expensive search to find an actual sequence of intermediate languages. Explain how to implement translator check-ins and translation queries so that the app can quickly determine whether there is any way at all to translate between two given languages or not. Make sure that both translator check-ins and preliminary translation queries take $O(\lg L)$ time or better—in particular, we don't want the app to get slower as the convention goes on!

Note that the app will still take $\Theta(L+T)$ time to search for an appropriate sequence of translators in the case that there *is* a way to communicate, but that's OK—people don't mind waiting if they already know the search is going to be successful. We just want to be able to tell them quickly if there is no way to communicate.

Question 2. Your close friend Polly Thyme skips up to you and says "Yo! I've got a fresh new algorithm for matrix multiplication. It totally beats Strassen." In talking about the procedure with your friend Polly, you realize that for a problem of size n, it divides the problem into 21 subproblems, each of which is one-third the size of the original problem, recursively solves each subproblem, and then combines the solutions in time $O(n^2)$. Do you believe your friend's claim that her algorithm is faster than Strassen's algorithm? Justify your answer.

Feel free to look up Strassen's algorithm and its running time; we did not talk about it in class.

Question 3. Given an array $A[0 \ldots n-1]$ of integers, a *wobbly pair* is a pair of integers in the array that are "out of order": that is, where $i < j$ but $A[i] > A[j]$. For example, the array

$$A = [2, -1, 17, 10, 3, 8]$$

has 6 wobbly pairs, namely, $(2,-1)$, $(17,10)$, $(17,3)$, $(17,8)$, $(10,3)$, and $(10,8)$. Put another way, if you imagine each number "looking" down the array to its right, there is a wobbly pair each time a number can "see" another number which is smaller than it. The number of wobbly pairs is in some sense a measure of how far away A is from being sorted (in fact, the number of wobbly pairs is exactly the minimum number of *adjacent swaps* needed to sort the array, and a sorted array has zero wobbly pairs).

(a) Describe, and analyze the running time of, a simple brute-force algorithm to compute the number of wobbly pairs in a given array.

(b) Now design and analyze a more efficient algorithm to compute the number of wobbly pairs in a given array. (*Hint*: think about modifying merge sort.)

Question 4. You are given a set of n potential players for your fantasy sportsball team and a salary cap S. Each player p_i also has an *awesomeness* score a_i (a positive real number) and a (non-negotiable) salary requirement s_i

(a positive integer). Your goal is to pick a set of players such that their total awesomeness is as large as possible, subject to the constraint that their total salary must not exceed S.

Unlike some other games, sportsball teams can be any size, so you are free to choose any number of players from 1 up to n.

Design and analyze a $\Theta(nS)$-time algorithm to find an optimal set of sportsball players, given as input the number of players n, the salary cap S, and two size-n arrays containing the awesomeness and salary values for the players. (You may assume the arrays are 1-indexed if it is helpful.) Be sure your algorithm finds not just the maximum possible awesomeness but an actual set of players which has that total awesomeness.

Optionally, for 1 token of extra credit: k-sportsball is a variant of sportsball in which all teams must have exactly k players. Explain how to modify your algorithm to take this into account. That is, given as input the set of n players, each with awesomeness score and salary requirement, the salary cap S, and a team size $k \leq n$, explain how to find the most awesome possible team of exactly k players which does not exceed the salary cap, in $\Theta(knS)$ time or better.