In preparing your solutions to the exam, you are **allowed to use (almost) any sources** including textbooks, other professors, previous homeworks and solutions, any static sources on the Internet, or (especially!) each other.

The only resources you may *not* use are (1) generative AI/large language models (LLM use is strictly prohibited in this class; see `https://hendrix-cs.github.io/csci382/#ai` for my reasoning), and (2) me. Of course, I am happy to answer general questions, go over homework problems or examples from class, 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 23**, 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. 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) In big-Θ terms, how many wobbly pairs could there be in an array of size n in the worst case?

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

(c) 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. Also, note that if your algorithm only ever adds 1 to a counter, it is doomed to take as long as your answer to part (a).

Question 2. 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.

Question 3. Suppose you are running a web site that is visited by the same set of people every day. Each visitor claims membership in one or more *demographic groups*; for example, a visitor might describe himself as male, 40–49 years old, a resident of Arkansas, an academic, a blogger, and a fan of extreme downhill snowmobile freestyle jumping. Your site is supported by advertisers. Each advertiser has told you which demographic groups may see its ads and how many of its ads you must show each day. Altogether, there are n visitors, k demographic groups, and m advertisers.

Question 3 is courtesy of Jeff Erickson, `http://jeffe.cs.illinois.edu/teaching/algorithms/hwex/f06/final.pdf`.

Design and analyze an efficient algorithm to determine, given all the data described in the previous paragraph, whether you can show each visitor exactly *one* ad per day, so that every advertiser has (at least) its desired number of ads displayed in total, and every ad is shown to someone in an appropriate demographic group.