Ground Rules

1. This exam is due at 4:00 pm on Tuesday, November 24. Turn it in on Canvas.

2. You may not discuss this exam with anyone except me until after you have turned it in.

3. The only written materials that you may consult are your copy of the course textbook *Introduction to Algorithms*, your lecture notes, your midterm, your homeworks, and the solutions distributed this term. In particular you may *not* look on the internet for information about these problems.

4. In all questions that ask you to give an algorithm, you must argue that your algorithm is correct and analyze its running time. In any question that asks you to compute or analyze something you have to justify your work.

5. Any changes or clarifications will be posted as announcements on Canvas, so check regularly.
Problem 1: (20 points)

Battleship is a game where each player has a set of ships that he or she secretly places on a grid, where ships cover from 2 to 4 consecutive squares. The opponents then take turns “shooting” at squares on their opponent’s grid. When there is part of a ship at the chosen grid square the ship is hit. When all of the squares in a ship are hit the ship is sunk. When all of a player’s ships are sunk that player loses the game. In the actual game a player is told whether a shot results in a hit or a miss and when each ship is sunk. This information allows players to use complex strategies.

This question asks you to analyze algorithms for a simplified solitaire version of Battleship. Each of \( n \) locations in an array are set to 1 with probability \( p \) and 0 with probability \( 1 - p \). (Assume that \( p > 0 \).) The player then picks locations until all of the 1 values have been found. The goal is to minimize the number of positions picked.

a) (4 points) If the player has a deterministic strategy known to an adversary and the adversary is allowed to permute the array before the player begins, how should the adversary pick a permutation to make the player use a large number of choices? What will be the expected number of moves in a game in this case?

b) (2 points) Suppose that the player picks a random permutation of the locations as the order to examine locations. Argue that the expected number of moves will be the same no matter what the adversary does (as long as the adversary must pick her permutation before the player randomly chooses his).

c) (14 points) What is the expected number of moves in a game using the randomized strategy described above?

Problem 2: (20 points)

I mentioned in class that Java’s ArrayList code does not double the size of the underlying array each time you try to append to a full array. It instead increases the array size of the underlying array by \( 3/2 \). This results in less memory being used, but the table expands more often.

a) (10 points) Use the accounting method to show that the amortized cost of an append to the end of a Java ArrayList takes constant amortized time, and give the amortized cost.

b) (10 points) Use the potential method to show that the amortized cost of an append to the end of a Java ArrayList takes constant amortized time, and give the amortized cost.

Problem 3: (20 points)

Alice and Bob are international agents, and a certain agency with a 3-letter acronym believes that they met at a party and that Alice gave Bob secret information. It is known that they both attended the party, but it is not known if they attended at the same time. A number of people have been interviewed by the agency, but nobody noticed whether Alice and Bob were at the party at the same time or if one left before the other arrived. You have a list of the attendees and have a number of statements in one of two forms: “C and D were at the party at the same time” and “E left before F arrived.” Many of these statements involve Alice and Bob.

They want you to write a program to determine if it is possible that Alice and Bob were present at the same time. Your program should either return a possible schedule of arrival and departure times for all of the people attending the party that is consistent with all of the witness statements and has Alice
and Bob present at the same time, or else present evidence that shows that no such schedule exists. You need not have specific times in your schedule. Just give a possible order of arrivals and departures for all of the attendees.

**Problem 4:** (20 points)

Bellman-Ford takes $\Theta(VE)$ time on a general directed graph. This problem asks about computing shortest paths on a directed graph after calculating its strongly connected components. This will not always lead to a faster run time than Bellman-Ford, because the entire graph can be a single SCC. However, if there are a number of small SSCs it is possible to do substantially better than Bellman-Ford.

Suppose a graph $G = (V, E)$ has $k$ strongly connected components. Let $SCC_1, SCC_2, \ldots, SCC_k$ be the strongly connected components, and let $(V_i, E_i)$ be the vertices and edges in $SCC_i$ for $1 \leq i \leq k$. Let $E'$ be the set of edges that connect vertices from different strongly connected components. Describe an algorithm to compute the shortest paths from a vertex $s$ to all of the other vertices that takes advantage of the structure of $G$. Give its run time in terms of $|V_i|$ and $|E_i|$ for $1 \leq i \leq k$ and $|E'|$. Explain why your algorithm is correct and how you computed the run time. Explain how you detect negative cycles reachable from $s$.

As a specific example, what is your algorithm’s run time on a graph with $\sqrt{|V|}$ strongly connected components, each with $\sqrt{|V|}$ vertices and $\Theta(V)$ edges, and $|E'| = \Theta(V^2)$?

**Problem 5:** (20 points)

You have a toy store, and your specialty is Russian nesting dolls. A Russian nesting doll consists of a set of painted wooden dolls of decreasing size that are placed one inside another. (See the pictures posted on the course home page.) You have dozens of sets of dolls of various sizes. The number of with nesting dolls in each set varies widely.

Some children played with the collection and un-nested all of the dolls and mixed them up. Fortunately they put the two halves of each individual doll together. Unfortunately there is no obvious way to determine which dolls belong in which sets. Because they are hand made even dolls in similar-looking sets may not be interchangeable, because they may not fit properly.

You have labeled each of the $n$ individual dolls. You also have tested each pair of dolls to determine if one can fit inside of the other. You want to write a computer program to determine which dolls should be placed into which sets. Describe an algorithm that will decide for each doll which other doll should be nested directly inside of it in a way that will minimize the number of sets of dolls. (Note that the number of sets is the number of outermost dolls, where an outermost doll is a doll that is not nested in any other doll.) Explain why your algorithm is correct and give its run time.
Problem 6: Extra Credit (20 points)

Your company has bought a new faster computer. There are \( n \) software applications \( \{A_1, \ldots, A_n\} \) that run on the old computer, and you would like to port some of them so that they will run on the new computer instead. For technical reasons, the first \( k \) applications (\( \{A_1, \ldots, A_k\} \)) must remain on the old system. For each of the other applications there is a positive monetary benefit \( b_i \) that you get if you port application \( A_i \) to the new system. The obvious solution is to move jobs \( \{A_{k+1}, \ldots, A_n\} \) to the new system to get all of the benefits, but there is a complication. The applications interact with one another, and there are communication costs when interacting applications are running on different systems. For each pair of jobs \( (A_i, A_j) \) there is a monetary cost \( c_{ij} \) if \( A_i \) runs on the old system and \( A_j \) runs on the new system.

Find a polynomial time algorithm that will determine which set of applications \( S \) should be moved to maximize the profit, which is defined to be:

\[
\sum_{i \mid A_i \in S} b_i - \sum_{i,j \mid A_i \in S \land A_j \notin S} c_{ij}
\]