CIS 320 Final

Please write your answers succinctly and rigorously. Good luck, and have fun!

NAME: _____

Problem 1. Mark each of the statements below as **True** or **False** (1 point each)

1. The worst-case running time for any algorithm that can sort n integers in the range 1 to 1000 is $\Omega(n \log n)$.

2. If we start with an integer linear program and remove the integrality constraints to obtain a linear program, the optimal value of the objective function remains the same.

3. Naively, multiplication seems harder than addition, but we know a divide and conquer algorithm that can multiply two n bit integers in $O(n)$ time — just as fast as addition.

4. In class we gave a dynamic program that solves the Knapsack problem in polynomial time, so Knapsack cannot be NP-complete.

5. The problem of computing minimax equilibrium strategies in a two player zero sum game can be expressed as a linear program with a number of variables and constraints that are both linear in the number of actions in the game.

Problem 2. Answer each of the questions below and give a couple of sentences of justification (2 points each):

1. Suppose you have a divide and conquer algorithm that at each level of recursion, on an input of length n runs in time $O(n)$, and then makes a recursive call on two sub-problems each of size $n/2$. What is the running time of the algorithm?

2. Is it true that there are learning problems for which there are polynomial time weak learning algorithms, but for which strong learning requires exponential time?

3. To prove an $\Omega(f(n))$ lower bound for the expected running time of a randomized algorithm, does it suffice to give a distribution of problem instances that causes every deterministic algorithm to take expected time $\Omega(f(n))$?

4. Is it true that Kruskal's algorithm, which solves the minimum spanning tree problem, can after a simple modification also be used to solve the *maximum* spanning tree problem?

Problem 3. (TA Trouble)

Your head TAs have to write your final exam this week, but they also have to finish all of their own coursework and have paper deadlines. They have an integer number of hours H to complete n tasks, and for each task t they have designed a function $u_t : \mathbb{Z}_+ \to \mathbb{R}_0$ such that $u_t(h)$ is the utility of spending h hours on that task. (E.g. when t is the task of writing the final, u_t represents how happy the students will be with the final and how well it reflects their knowledge.) Assume that u_t is a non-decreasing function, so more hours on task t means a better outcome. Given these functions u_t for each of the n tasks, give an algorithm that maximizes $\sum_{i\in[n]} u_i(h)$, assuming each task can only be allocated an integer number of hours, and the total amount of allocated time cannot be greater than H. Provide a proof of correctness and prove that the runtime of your algorithm is polynomial in n and H (10 points).

Problem 4. (Cheating Chaps)

Suppose you are teaching an algorithms class, and have built a social network amongst the students in the class based on past collaboration data: the vertices in the graph are students, an there is an edge between two students if they have previously collaborated on a problem set.

You are preparing for an online final exam, but you don't want the students to cheat (they are not to be trusted). To foil them, you have decided to make multiple versions of the exam, and to assign versions of the exam to students so that no two neighbors in the graph are given the same copy of the exam. An assignment like this is called "safe." What is the fewest number of exams you need to prepare to produce a safe assignment? Unfortunately this problem is NP-hard, but we might be able to come up with a decent approximation algorithm.

(a) Describe an algorithm to produce a safe assignment of exam versions to students which uses at most $d+1$ exam versions, where d is the maximum number of collaborators any single student has had (i.e. the maximum degree of the graph). Your algorithm must run in $O(d \cdot n)$ time. Prove the correctness and run-time of this algorithm. (5 points)

(b) Describe or draw a graph instance with $n = 4$ vertices where your algorithm will always produce the exactly optimal number of exam copies. Briefly justify your answer. (2 points)

(c) Describe or draw a graph instance with $n = 4$ vertices where an algorithm solving part a) does not produce the exact minimum number of exam copies. Briefly justify your answer. (3 points)

Problem 5. You are the executive chef at a 3-Michelin-star restaurant and need to prepare N smoothies from K kinds of ingredients before tonight's fancy dinner at your restaurant. For each smoothie $1 \leq i \leq N$, you have perfected its recipe — a set of ingredients $l(i) \subseteq \{1, \ldots, K\}$.

You only have a single blender to prepare the smoothies. After preparing a smoothie i, you need to wash the blender cup before making another smoothie j, *unless* this next smoothie j includes all the ingredients of the current smoothie i. Formally, if you are making smoothie i followed by smoothie j, you wash the cup after making smoothie i iff $l(i) \not\subseteq l(j)$.

You are very lazy and want to prepare the smoothies $1, \ldots, N$ in the order that makes you wash the cup the *least* number of times. Design a bipartite-matching based algorithm that outputs such an ordering of the smoothies that solves your problem. (10 points)

Problem 6. A SAT instance ϕ is called *positive* if it has no negative literals in any clause. For example: $\phi = (x_1 \vee x_3 \vee x_5) \wedge (x_1 \vee x_2) \wedge (x_3 \vee x_4)$ is a positive SAT instance. Clearly all positive SAT instances are satisfiable — just set all of the variables equal to "TRUE". However, we can ask whether it is possible to satisfy ϕ with a satisfying assignment that sets only k literals to TRUE. (In the example above, we can satisfy ϕ by setting only 2 literals to TRUE). If ϕ has a satisfying assignment that has at most k literals set to TRUE, call ϕ k-satisfiable. We can define the following computational problem, called POSITIVE-SAT: Given a positive SAT instance ϕ and an integer k, determine whether or not ϕ is k-satisfiable.

Prove that POSITIVE-SAT is NP-complete. (10 points).