Instructions:

1. The test is closed book, closed notes.

2. For most of the problems, I am interested in testing whether you understand the techniques and concepts more than I am interested in the solution to the particular problem. For example, if I ask you to prove that a problem is NP-hard, I am more interested in learning if you know how to prove that a problem is NP-hard, than I am in the specifics of the problem. If I ask you to prove that a greedy algorithm is correct using an exchange argument, I am more interested in learning if you know how an exchange argument works, than I am in the specifics of the problem. I ask these questions in the context of specific problems to allow you to demonstrate your understanding in a concrete setting. Of course you have to take into account the specifics of the problem, but make sure to explain the general method/technique/concept that you are using as well.

3. 25% partial credit is given for the answer “I don’t know.” A blank answer will be interpreted as “I don’t know.” An answer that displays a major conceptual error will likely receive a grade of zero. It is perfectly fine to give an incomplete answer, e.g. ”Here is how one proves a problem NP-hard, but I don’t know how to prove this problem NP-hard.” I will make a judgement call on how much credit such answers should receive.

4. I will assume that if you write something, that you are asserting that you have good confidence in the correctness of what you write. It is a bad strategy to give an answer that you do not have good confidence in.

5. Answer up to 4 of the problems. Clearly write the number of the problem that you are answering. If you answer more than 4, an arbitrary 4 answers will be graded.

6. If you are uncertain about anything, ask a question!
1. Define what a matroid is. Give a greedy algorithm to find the maximum weight independent set in a matroid, and prove that this algorithm is correct using an exchange argument.

2. Assume a router sees a stream of IP packets from two different sources. So the router sees a packet, and then can do some minimal computation, then forwards the packet, sees the next packet, etc. The router is trying to determine the Jaccard similarity of the destination IP addresses for the two different sources while using very little space. Let $A$ be the collection of destination IP addresses for the first source, and $B$ be the collection of destination IP addresses for the second source. The Jaccard similarity of $A$ and $B$ is the ratio of the number of elements of their intersection and the number of elements of their union, namely

$$\frac{|A \cap B|}{|A \cup B|}$$

Assume that we have a uniform hash function $h$ that maps IP addresses to integers, where the range is sufficiently large that the probability of a collision is negligible. Explain how to maintain a random variable whose expectation is the Jaccard similarity using only a constant amount of storage space. For full credit you must prove that the expectation is the Jaccard similarity.

3. Consider the problem of constructing maximum cardinality bipartite matching.

   (a) Construct an integer linear program for this problem
   
   (b) Show how to convert any solution to the relaxed linear program to a solution of the integer linear program with the same objective value in polynomial time. Note that “relaxed” here means that the constraint that the variables take on integer values does not apply.
   
   (c) Construct the dual linear program.
   
   (d) Give a natural English interpretation of the dual problem modeled by this linear program.
   
   (e) Does the dual problem always have an integer optimal solution? Justify your answer.
   
   (f) Explain how to give a simple proof that a graph doesn’t have a matching of a particular size. You should be able to come up with a method that would convince someone who knows nothing about linear programming.

4. Consider the following problem. The input is a graph $G = (V, E)$. Feasible solutions are subsets $S$ of the vertices $V$. The objective is to maximize the number of edges with one endpoint in $S$ and one endpoint in $V - S$. 

2
(a) Give a simple polynomial-time randomized algorithm for this problem and show that it is 2 approximate.

(b) Develop a deterministic polynomial-time 2-approximation algorithm for this problem using the method of conditional expectations, which considers the vertices one by one, but instead of flipping a coin for each vertex \( v \), puts \( v \) in the bipartition that would maximize the expected number of edges in the cut if coin flips were used for the remaining vertices.

(c) Give a simple greedy algorithm that ends up implementing this policy.

(d) Prove that this greedy algorithm has approximation ratio at most 2.

5. Consider the Vertex Cover problem. Let the parameter \( k \) be the number of vertices in the minimum cardinality vertex cover.

   (a) Define “fixed parameter tractable algorithm”.
   
   (b) Define “kernelization”.
   
   (c) Prove that a problem has a fixed parameter tractable algorithm if and only if it is kernelizable.
   
   (d) Give a kernelization for vertex cover.

6. Prove that the Move-To-Front (MTF) algorithm is 2-competitive for the List Update problem.

7. Prove that the competitive ratio of every deterministic algorithm for the List Update problem is at least 1.99.

8. Here we are considering distributed algorithms in the CONGEST model. Let \( d \) be the diameter of \( G \). Let \( n \) be the number of vertices in \( G \). Let \( \Delta \) be the maximum degree of any vertex in \( G \). Note that the algorithm does not know \( d \), \( \Delta \), or \( n \) a priori. You can assume that each node in the network starts with a unique ID integer consisting of \( O(\log n) \) bits. Note that these ID’s are not necessarily 1, ..., \( n \).

   Assume that \( \Delta = 3 \). A \( k \)-bit coloring is an assignment of \( k \) bits to each node. A \( k \)-bit coloring is proper if every pair of adjacent nodes are assigned different bit strings. Given a deterministic algorithm to give a proper \( O(1) \)-bit coloring in time \( O(\log^* n) \).

9. Recall that the input for the Knapsack problem consists of \( n \) items with positive values \( v_1, \ldots, v_n \), positive integer weights \( w_1, \ldots, w_n \) and weight limit \( L \). The problem is to find a maximum value subset of the items subject to the constraint that the total weight is at most \( L \).
(a) Give a dynamic programming algorithm that runs in time $O(nL)$.

(b) Define polynomial time approximation scheme.

(c) Explain how to convert your dynamic programming algorithm into a polynomial time approximation scheme. You can get partial credit for an informal explanation, but full credit requires a formal explanation.

10. Prove using Yao’s technique that every Monte Carlo comparison-based sorting algorithm requires $\omega(n)$ time. Start by defining Monte Carlo algorithm, and stating the version of Yao’s technique that is appropriate for this use.