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 question 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.” False or completely unsubstantiated assertions will receive lesser credit.

4. Solve up to 6 of the problems. Clearly write the number of the problem that you are solving. If you answer more than 6, an arbitrary 6 answers will be graded.

5. If you are uncertain about anything, ask a question!
1. In the Knapsack problem the input consists of the weights $w_1, \ldots, w_n$, and values $v_1, \ldots, v_n$ of $n$ coins and a weight limit $W$. A feasible solution is a subset of the coins of aggregate weight at most $W$. The objective is to maximize the aggregate value. Give an $O(nW)$ time, and $O(n + W)$ space algorithm to compute the optimal solution to this optimization problem (note that the algorithm needs to output the actual subject of coins, not just the subset’s aggregate weight and value).

2. In the Knapsack problem the input consists of the weights $w_1, \ldots, w_n$, and values $v_1, \ldots, v_n$ of $n$ coins and a weight limit $W$. A feasible solution is a subset of the coins of aggregate weight at most $W$. The objective is to maximize the aggregate value. Assume you have an $O(nW)$ time dynamic programming algorithm that solves this problem. Show how to derive a polynomial time approximation scheme for the knapsack problem. Start by defining polynomial time approximation scheme. Note that you need not have answered the previous problem in order to answer this problem. You must have some reasonable justification that you indeed have given a polynomial time approximation scheme.

3. Consider a setting where you have two computer networking routers $A$ and $B$. Each router has collected a list $L_A$ and $L_B$ of IP source addresses for the packets that have passed through the router that day. An IP address is $n$ bits, and thus there are $2^n$ possible IP addresses. Now the two routers want to communicate via a two-way channel (assume there is no delay of messages) to determine whether there was some source that sent a packet through one of the routers, but not the other. Prove using an adversarial argument that every deterministic protocol for this problem must sent $2^n - O(1)$ bits for some possible instance.

4. Assume that you hash $n$ items into a closed-addressed hash table of size $n$, with the probability that each item hashes to a particular table energy is $1/n$, independent of where the other items hash. Let $x_i$ be the number of items that hash to table entry $i$. Show that $E[\max_i x_i] = O(\log n / \log \log n)$. You may use without proof the fact that $\binom{n}{k} \leq \left(\frac{ne}{k}\right)^k$.
5. Consider the following online problem. You are given a sequence of bits $b_1, \ldots, b_n$ over time. Each bit is in an envelope. You first see the envelope for $b_1$, then the envelope for $b_2$, etc. When you get the envelope $i$, you can either look inside to see the bit, or destroy the envelope (in which case you will never know what the bit is). You know a priori that at least $n/2 + 1$ of the bits are 1. Your goal is to find an envelope containing a 1 bit. You want to open as few envelopes as possible. Show using Yao’s technique that every randomized Monte Carlo algorithm for this problem that fails with probability at most $1/n$ must open $\Omega(\log n)$ envelopes in expectation. Start by stating clearly the version of Yao’s technique that is applicable here. You do not need to prove the correctness of this version of Yao’s technique.

6. Consider a two person game specified by an $m$ by $n$ payoff matrix $P$ (note that some entries could be negative). The two players can be thought of as a row player and a column player. The number of possible moves for the row player is $m$ and the number of possible moves for the column player is $n$. Each player picks a probability distribution over its moves, and then money is exchanged. In particular, the row player pays the column player $E_{r,c}[P_{r,c}]$ dollars, where the expectation is over the joint probability distribution of the row and column players move. We assume that the game is played sequentially, so that one player specifies his probability distribution, the other players see that probability distribution, and then specifies a response probability distribution (we will assume that this player makes the best possible response). Each player wants to be paid as much money as possible, and if this is not possible, to pay as little as possible, in expectation.

(a) Show that the problem of finding the best probability distribution for the player that goes first (say the row player for concreteness) can be expressed as a linear program.

(b) Explain what linear programming duality says about how the amount of money that the row player makes if she goes first compares to the amount of money the row player makes if she goes second. You must have some justification here.

7. The input for the maximum cut problem is an undirected graph $G = (V, E)$. A feasible solution is a subset $S$ of the vertices. The objective is the number of edges between $S$ and $V - S$. Give a deterministic algorithm for this optimization problem, and show that it is $2$-approximate. Include a definition of “2-approximate” that is appropriate for this setting.
8. Consider the paging problem. Consider the following deterministic online algorithm.

**Algorithm Description:** Each page P has an associated bit: FRESH or STALE. If requested page P in fast memory, then P’s associated bit is set to FRESH. If the requested page P is not in fast memory, then an arbitrary STALE page is selected from the STALE pages in fast memory and ejected, and P’s associated bit is set to FRESH. If the request page P is not in fast memory, and all pages in fast memory are FRESH, then make all pages in fast memory STALE, select an arbitrary STALE page from the STALE pages in fast memory to evict, and P associated bit is set to FRESH.

We want to consider the homework problem that showed that this algorithm with k pages of fast memory is 2-competitive/approximate against the optimal algorithm for k/2 pages of fast memory, using the following potential function

\[ \Phi = \sum_{p \in ON} \text{credit}(p) + 2 \cdot \sum_{p \in OPT} (1 - \text{credit}(p)) \]

where \( ON \) is the algorithm’s cache and \( OPT \) is the adversary/optimal cached, and \( \text{credit}(p) \) is equal to 1 if page p is in the algorithm’s fast memory and is FRESH, and \( \text{credit}(p) = 0 \) otherwise.

(a) Write the key equation that you need to show holds for every access.

(b) Identify the case where you need to use the assumption that the algorithm’s cache is twice as large as the adversarial/optimal cache, and show that the key equation holds in this case.

9. Show that the 3-coloring problem is NP-hard using a reduction from 3-CNFSAT. Recall the input for the 3-coloring problem is an undirected graph, and the question is whether each vertex can be colored with 1 of 3 possible colors so that no pair of adjacent vertices are colored the same color. Recall that input to the 3-CNFSAT problem is a Boolean formula in conjunctive normal form (AND of OR’s of variables or negation of variables) with exactly 3 literals per clause.

You may assume the existence of a graph \( S \) with the following special property: \( S \) has 4 designated vertices \( x, y, z \) and \( t \), and \( S \) is 3 colorable if and only if at last one of \( x, y \) or \( z \) is colored the same color as \( t \) is colored.

10. Show using linearity of expectations that the expected running time of randomized quick-sort (where the splitter/pivot is picked uniformly at random from the subarray being sorted) is \( O(n \log n) \). Make sure to clearly define your variables.
11. Consider the Ford-Fulkerson algorithm for network flow on a graph network with $V$ vertices, $E$ edges, and maximum flow $F$. We gave an algorithm $A$ with worst-case running time $\Theta(FL)$. We gave an algorithm $B$ with worst-case running time $\Theta(VE^2)$. We gave an algorithm $C$ with worst-case running time $\Theta(VE\log F)$. State, with justification, which of these algorithms are polynomial time algorithms, which are strongly polynomial time algorithms, and which are pseudo-polynomial time algorithms. You should start with a definition of polynomial time algorithm, strongly-polynomial time algorithm, and pseudo-polynomial time algorithm.

12. The input to the vertex cover problem is an undirected graph $G$. The feasible solutions are subsets $S$ of vertices such that every edge is incident on at least one vertex in $S$. The objective is to minimize the number of vertices in $S$.

(a) Explain how to express this problem as a integer linear program.

(b) Explain how to use the relaxed linear program (where variables are allowed to be non-integer rational numbers) to develop a polynomial time approximation algorithm for this problem that is 2-approximate. You must justify that the algorithm is 2-approximate. Give a definition of 2-approximate that is appropriate for this setting.