Instructions:

1. The test is closed book, closed notes.

2. If you are taking the final for preliminary exam credit, then write your preliminary number given to you by Keena on the exam answer sheet, but do not write your name. If you are not taking the final for preliminary exam credit, write your name on the exam answer sheet.

3. If you are taking the course for some grading option other than the standard letter grading option, please specify that on your exam answer sheet.

4. 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 a problem 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.

5. 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.

6. Solve up to 8 of the problems. Clearly write the number of the problem that you are solving. If you answer more than 8, an arbitrary 8 answers will be graded.

7. **If you are uncertain about anything, ask a question!**
1. Two of the three following definitions of big Omega are logically equivalent. Identify the two that are logically equivalent, and prove that they are equivalent. Take care in your exposition. Prove that the third definition is not logically equivalent to the other two. You may assume that we are only considering functions that are defined on the positive integers and are positive and non-decreasing.

(a) \( f(n) = \Omega(g(n)) \) iff \( \exists c > 0 \exists N \forall n > N f(n) \geq cg(n) \)

(b) \( f(n) = \tilde{\Omega}(g(n)) \) iff \( \exists c > 0 \forall n f(n) \geq cg(n) \)

(c) \( f(n) = \tilde{O}(g(n)) \) iff \( \exists N \forall n > N \exists c > 0 f(n) \geq cg(n) \)

2. Give as simple estimates as you can of the following sums that are accurate to within a multiplicative constant. You do not need to fully prove the correctness of each of your answers, but give some convincing justification for your answers.

(a) \( \sum_{i=1}^{\sqrt{n}} i \log \log i \)

(b) \( \sum_{i=1}^{\log n} \frac{(\log i)^2}{(\log \log i)^3} \)

(c) \( \sum_{i=1}^{\log \log n} i^{3i} \)

(d) \( \sum_{i=1}^{\sqrt{n}} i^3 \frac{3^i}{3^i} \)

3. Show that the expected case running time of Randomized Quicksort is \( O(n \log n) \) using linearity of expectation. Assume that the splitter is picked uniformly at random from the subarray in question.

4. Consider the offline paging problem where there are \( k \) pages of fast memory and \( k + 1 \) pages of slow memory. The input is a sequence of requests to pages in slow memory. If the page requested is to the unique page that is not in fast memory, then a page fault occurs. In response to a page fault, the offline algorithm has to evict a page of its choice from fast memory, and copy the requested page to fast memory. Give the offline algorithm that minimizes the number of page faults. Prove that this algorithm is correct/optimal using an exchange argument.
5. Consider the following energy management problem faced by a city. The city owns a hydro-electric dam that can generate up to $m$ megawatt-hours of electricity per day at a cost of $L$ dollars. Even if less than $m$ megawatt-hours is generated, the cost is still $L$. Additionally, on each day the city can buy additional electricity for $c$ dollars per megawatt-hour. The city can store unused energy in batteries that have a capacity of $B$ megawatt-hours, but this costs $h$ dollars per megawatt-hour per day. The city knows its energy needs for $n$ consecutive days are $d_1, \ldots, d_n$ megawatt-hours. Give an algorithm that computes the optimal least-cost energy generation schedule that meets the city’s energy needs on each day. This schedule needs to specify on each day, how much energy to generate from the dam, how much energy to buy, how much energy to take out of the batteries, and how much energy to store in the battery. The algorithm’s running time should be polynomial in $n$ and $B$.

Assume all parameters are positive integers. Note that for simplicity that each day is being represented as a point in time; Obviously in real-life batteries could be filled/emptied continuously instead of discretely.

6. Consider the situation where you have a web server running over $n$ days. On some days the server is up and running, and on other days the server is down. There events are not Bernoulli trials. There may be dependence between the server states on various days. Let $X_i$ be the random variable denoting the number of consecutive days that the server is up starting from day $i$, $1 \leq i \leq n$. All you know is that for all $i$, the probability that $X_i$ is more than $k$ is at most $1/k^4$. Calculate as tight of an upper bound as you can on the expected value of $\max(X_1, \ldots, X_n)$.

7. Explain how to modify Dijkstra’s algorithm to compute “longest” path in graphs with positive edges weights, where now the “length” of a path is the minimum edge weight on the path. So you want to find path where the minimum edge weight is maximized.

8. Show that the problem of solving an integer linear program is NP-hard using the fact that 3-CNF-SAT is NP-complete. Recall that the input for 3-CNF-SAT is a Boolean formula in conjunctive normal form, with exactly three distinct literals per clause, and the problem is to determine if the formula is satisfiable.

9. Show that if there is a polynomial-time 100-approximation algorithm for the clique problem, then there is a polynomial-time 10-approximation algorithm for the clique problem. The input for the clique problem is an undirected graph $G$, and the output is the largest clique (a collection of mutually adjacent vertices).

10. Consider the online paging problem where there are $k$ pages of fast memory and $k+1$ pages of slow memory. So an online algorithm sees over time a sequence of requests to pages in
slow memory. If the page requested is to the unique page that is not in fast memory, then
a page fault occurs. In response to a page fault, the online algorithm has to evict a page
of its choice from fast memory, and copy the requested page to fast memory. Assume that
each of the pages of slow memory is equally likely to be requested, independent of other
requests.

(a) Show that for every deterministic online algorithm, the expected number of requests
between page faults is $\Omega(k)$. What sort of distribution does the number of requests
between page faults have.

(b) Show that the expected number of requests between page faults for the optimal offline
algorithms is $O(k \log k)$.

(c) Conclude that the expected number of page faults for every deterministic online algo-
rithm is $\Omega(\log k)$ times the optimal number of page faults.

11. Consider the paging problem where there are $k$ pages of fast memory and $k + 1$ pages
of slow memory. So an online algorithm sees over time a sequence of requests to pages
in slow memory. If the page requested is to the unique page that is not in fast memory,
then a page fault occurs. In response to a page fault, the online algorithm has to evict a
page of its choice from fast memory, and copy the requested page to fast memory. Assume
that you are given that there is an input distribution such that the expected number of
page faults for every deterministic online algorithm is $\Omega(\log k)$ times the optimal number of
page faults. Show that for every randomized online algorithm, there is an input sequence
such that the expected number of page faults for the online algorithm is $\Omega(\log k)$ times the
optimal number of page faults. So basically, you are asked to reprove that Yao’s technique
works in the case of approximation/competitive ratio for a minimization problem.

12. Consider the problem where the input is $n$ numbers, and the output is the largest and
smallest number. Prove using an adversarial argument that every deterministic comparison-
based algorithm requires $3n/2 - O(1)$ comparisons. So the only way the algorithm can access
the input is by comparing two numbers in the input.

13. Consider the problem computing the shortest tour connected $n$ points in the Euclidean
plane. Give a polynomial time 2-approximation algorithm for this problem. You must
prove that your algorithm is a 2-approximation algorithm.

14. Prove the fact that max-flow equals min-cut is a consequence of strong linear programming
duality (that the objectives for primal and dual optimal solutions are the same).

15. Consider the following problem. The input is a directed graph with positive weights on the
edges, and a designated vertex $s$. The goal is to find the least cost spanning tree $T$ rooted
at \( s \) with all edges in \( T \) directed away from \( s \). So in \( T \) there is a unique path in \( T \) from \( s \) to each vertex in the graph. The cost of a tree \( T \) is just the sum of the weights of the edges. Explain how this problem can be solved in polynomial time.

16. Assume that we want to prove that an online scheduling algorithm \( A \) guarantees that the average waiting time for for a job is at most 3 times the average waiting time for Shortest Remaining Processing Time scheduling algorithm (which is optimal for the objective of average waiting time) using a potential function \( \Phi \) in the manner that we used to analyze Round Robin. State what statements need to be proved in order to complete this proof. The purpose of this question is to determine if you know how a potential function argument works.

A few reminders about scheduling: Each job \( i \) has a release time \( r_i \), which is the earliest time that it can be run, and a size \( p_i \). If job \( i \) runs on a speed \( s \) processor, it will complete after running \( p_i/s \) time units. The waiting time for a job \( i \) is the difference in time \( C_i - r_i \) between \( r_i \) and when it is completed at time \( C_i \). The average waiting time for \( n \) jobs is then \( \sum_{i=1}^{n} (C_i - r_i)/n. \)