CSE 60111: Complexity and Algorithms

Sample midterm exam

Name:	Email Address:

1. This is a closed-book and closed-notes exam. You are allowed three "cheat sheets" on standard 8.5 by 11 inch paper, handwritten on the front and back.
2. Print your full name and your email address in the boxes above.
3. Print your name or initials at the top of every page.
4. Please write clearly and legibly. If I can't read your answer, I can't give you credit.
5. When asked to design an algorithm, you must also provide a runtime analysis for that algorithm. However, proofs of correctness are NOT required for algorithms on this exam. However, problems that explicitly ask you to prove something still require you to do proofs!
6. There are 5 problems on the exam. Your grade will be calculated based only on 4 out of the 5 problems. Please feel free to try all of them, though; I'll take the maximum of the 4 for your final grade.
7. Remember, these are NOT necessarily in order of difficulty. Please read all the problems first, and don't allow yourself to get stuck on a single problem.

#	1	2	3	4	5	Total
Max	20	20	20	20	20	80
Score						

1. Suppose you are given an array $A[1..n]$ of n *distinct* integers (no fractions or reals allowed), sorted in increasing order. Describe and analyze an algorithm to determine whether there is an index i such that $A[i] = i$.

   Hint: You should do better than $O(n)$.

2. Inspired by the fact that you have almost completed your midterms, you decide to make plans to show off your extreme dancing skills! You sign up for a dance competition that meets next week, and luckily are able to obtain a copy of the n songs that judges will play during the competition (in chronological order). Even more luckily, you've practiced and have an extremely good grasp of your ability and the judges: so for each value k, you know that if you dance to the k^{th} song, you will be awarded exactly $score[k]$ points, but then you'll be exhausted and will miss the next $wait[k]$ songs (meaning you'll get 0 points for songs $k+1$ through $k + wait[k]$ if you do dance).

   Armed with this information, you decide to strategize! Your goal is make your score as high as possible, so that you can win this thing. Describe and analyze an efficient algorithm to compute your maximum possible score; the input to your algorithm will be the pair of arrays, $score[1..n]$ and $wait[1..n]$.

3. A company is planning a party for its employees. The employees in the company are organized into a strict hierarchy, that is, a tree with the company president at the root. The organizers of the party have assigned a real number to each employee measuring how fun the employee is. In order to keep things social, there is one restriction on the guest list: an employee cannot attend the party if their immediate supervisor is present. On the other hand, the president of the company must attend the party, even though she has a negative fun rating; it is her company, after all. Give an algorithm that makes a guest list for the party that maximizes the sum of the fun ratings of the guests.

4. A graph is called *3-regular* if every vertex in the graph has 3 edges adjacent to it.

   Suppose you are given a 3-regular graph, with a nonnegative weight $w(v)$ associated with each vertex v in the graph. Your goal is to choose an independent set S in G, so that the sum of the weights on the vertices in S is as large as possible.

   Consider the heaviest first greedy algorithm, which iteratively adds the heaviest vertex in G to S, then deletes that vertex and all of its neighbors from G (and then repeats as long as there is a vertex left in the graph). Prove that this greedy algorithm returns an independent set whose total weight is at least 1/3 times the maximum possible weight of an independent set in G.

5. Suppose we are choosing between the following 3 algorithms:

   - Algorithm A solves a problem with input size by n by dividing it into four subproblems, each of size $n/2$, recursively solving each subproblem, and then combining the solutions in quadratic time.
   - Algorithm B solves problems of size n by recursively solving a subproblem of size $n - 1$ and one of size $n - 2$, and then combining the solutions in constant time.
   - Algorithm C solves problems of size n by dividing them into three subproblems, each of size $n/6$, and then combining the solutions in constant time.

   What are the running times of each of these problem (in big-O notation), and which would you choose as the best?

(scratch paper)

(scratch paper)