Exam 2—D&C, DP, flow networks

In preparing your solutions to the exam, you are allowed to use any sources including your textbook, other students and professors, previous homeworks and solutions, or any sources on the Internet. You **may ask me for feedback on potential solutions**, but I will not give you any hints. Of course, I am also happy to answer general questions, go over homework problems, or answer clarifying questions about exam problems.

The exam will take place in class on Friday, November 15 (MC Reynolds 317, 1:10-2:00pm). You are not allowed to bring any notes, textbooks, calculators, or any other resources with you to the exam. Bring only something to write with; I will provide a fresh copy of the exam, paper for writing your solutions, and scratch paper.

As usual, to "design and analyze" an algorithm means to (a) describe the algorithm, (b) prove/justify its correctness, and (c) analyze its asymptotic running time. Full credit will only be given for the most efficient possible algorithms. Algorithms must be clearly explained (using pseudocode if appropriate) in sufficient detail that another student could take your description and turn it into working code. You may **freely cite any theorems proved in class** (without proof), or **use algorithms covered in class as subroutines**.

Question 1. Given an array $A[0 \ldots n-1]$ of integers, a *wobbly pair* is a pair of integers in the array that are "out of order": that is, where $i < j$ but $A[i] > A[j]$. For example, the array

$$A = [2, -1, 17, 10, 3, 8]$$

has 6 wobbly pairs, namely, $(2,-1)$, $(17,10)$, $(17,3)$, $(17,8)$, $(10,3)$, and $(10,8)$. Put another way, if you imagine each number "looking" down the array to its right, there is a wobbly pair each time a number can "see" another number which is smaller than it. The number of wobbly pairs is in some sense a measure of how far away A is from being sorted; in fact, the number of wobbly pairs is exactly the minimum number of *adjacent swaps* needed to sort the array, and a sorted array has zero wobbly pairs.

You keep on using that word. I do not think it means what you think it means.

(a) Describe, and analyze the running time of, a simple brute-force algorithm to compute the number of wobbly pairs in a given array.

(b) Now design and analyze a more efficient algorithm to compute the number of wobbly pairs in a given array.

Question 2. You are given a set of n widgets and a positive integer G. Each widget w_i also has a *froob* value f_i (a positive real number) and a *grump* value g_i (a positive integer). As you can tell from their names, froob is good and grump is bad. Your goal is to pick a subset of the widgets such that their total froob is as big as possible, subject to the constraint that their total grump must be $\leq G$ (you can only deal with so much grump).

(a) Design and analyze an $O(nG)$-time algorithm to find an optimal subset, given as input the number of widgets n, the maximum allowable grump G, and two size-n arrays containing the froob and grump values for the widgets. (You may assume the arrays are 1-indexed if it is helpful.) Be sure your algorithm finds not just the maximum possible froob but an actual subset of widgets which has that total froob.

(b) Explain (one sentence should be sufficient) why the problem would be much harder if the grump values were allowed to be positive *real numbers* rather than just positive integers.

Question 3. Define the *Escape Problem* as follows. We are given a directed graph $G = (V, E)$ (imagine a network of one-way roads; two-way roads can be modelled by a pair of one-way roads in opposite directions). A certain collection of nodes $X \subseteq V$ are designated as *populated nodes*, and a certain other collection $S \subseteq V$ are designated as *safe nodes*. Assume that X and S are disjoint, that is, no node is both populated and safe. Note, however, there may be other nodes which are neither populated nor safe.

In case of an emergency, we want evacuation routes from the populated nodes to the safe nodes. An evacuation route is a path that starts at some populated node and ends at a safe node. There must be an evacuation route for each populated node, and no two routes may share an edge. However, it is OK if routes share vertices (including the possibility of multiple routes all leading to the same safe node). That is, more formally, a set of evacuation routes is a set of paths in G so that (i) each node in X is the tail of one path, (ii) the last node on each path lies in S, and (iii) the paths do not share any edges. Such a set of paths gives a way for the occupants of the populated nodes to "escape" to S, without overly congesting any edge in G.

Given G, X, and S, describe and analyze a polynomial-time algorithm to either find a set of evacuation routes, or decide that no such set of evacuation routes exists.

Question 3 is adapted from Kleinberg & Tardos, exercise 7.14.