CS G713: Advanced Algorithms	**Problem Set 1**
Assigned: September 26, 2008	Due: October 10, 2008 before class

Guidelines (different from those on problem set 0) If you write "I do not know how to solve this problem" then you get 1/4 of the score for the problem. If you write nonsense then you get 0.

As we are going to learn in this class, *time* and *space* are very valuable resources. Strive to give effective, compact solutions. Your solutions should touch on all the main points, but long calculations or discussions are not required nor sought.

Do not worry if you sometimes "do not get it." The problems are meant to stimulate you, not to overwork you. Unless specified otherwise, you can collaborate, but you must acknowledge all your collaborators in your solutions. To hand in your solutions: Give it to me, slide it under my door West Village H (246), or email it to csg713-instructor@ccs.neu.edu.

Problem 1. Puzzle: Find the missing elements You are given as input a permutation of $(1, 2, \ldots, n)$, except that exactly one element is replaced with 0 (say the permutation is stored in an array $A[1], A[2], A[3], ..., A[n]$ where $A[j] = 0$ for exactly one j).

Give an algorithm to find the missing element that runs in linear time and uses constant space. For this problem you should think of any operation involving $a, b \leq n^{1000}$ as taking constant time and space.

Now solve the variant in which exactly *two* elements are set to 0, and you want an algorithm that finds them both and runs in linear time and constant space.

If you are unable to solve this problem, you can obtain partial credit by dropping the space requirement and simply developing algorithms that find the missing elements in linear time.

Problem 2. Bit-complexity of iterated operations In this problem we are going to count the *bit-complexity* of operations. You can assume throughout that performing arithmetic on t-bit integers requires time $t \cdot \operatorname{poly}\log t$.

You are given as input a sequence of n numbers $a_1, a_2, \ldots, a_n$ where $|a_i| \leq c$ for every i, where c is an absolute constant (e.g. $c = 17$). Show how to compute the product of the numbers in time $n \cdot \operatorname{poly}\log n$.

Now instead of integers you are given as input n $c \times c$ matrices $A_1, A_2, \ldots, A_n$ whose entries are again bounded in absolute value by the absolute constant c. (Note the input length has again bit-length that is $O(n)$.) Show how to compute the product of the matrices in time $n \cdot \operatorname{poly}\log n$.

Now you are given as input n 2×2 matrices $A_1, \ldots, A_n$ as well as n vectors $V_1, V_2, \ldots, V_n$ of 2 coordinates, where all coordinates (both in the matrices and in the vectors) have absolute value at most c again. You are interested in computing x_n where $x_0 := (1, 1) \in \mathbb{R}^2$ and $x_i := A_i x_{i-1} + V_i$. Show how to compute x_n in time $n \cdot \operatorname{poly}\log n$.

Problem 3. CLRS Problem 15-2. Consider the problem of neatly printing a paragraph on a printer. The input text is a sequence of n words of lengths $l_1, l_2, \ldots, l_n$, measured in characters. We want to print this paragraph neatly on a number of lines that hold a maximum of M characters each. Our criterion of "neatness" is as follows. If a given line contains words i through j, where $i \leq j$, and we leave exactly one space between words, the number of extra space characters at the end of the line is

$$M - j + i - \sum_{k=i}^{j} l_k,$$

which must be nonnegative so that the words fit on the line. We wish to minimize the sum, over all lines except the last, of the cubes of the numbers of extra space characters at the ends of the lines. Give a dynamic-programming algorithm to print a paragraph of n words neatly on a printer. Analyze the running time of your algorithm.

Problem 4. CLRS Problem 16-1.b. Consider the problem of making change for n cents using the fewest number of coins. We have seen in class a dynamic programming solution to this problem.

Prove that the greedy algorithm works correctly when you have (an infinite supply of) k types of coins with values $c^0, c^1, \ldots, c^k$ respectively, where c, k are integers bigger than 1.