Advanced Algorithms, Fall 2009	**Problem Set 1**
Assigned: September 22	Due: October 6 before class

Guidelines 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. I need your solutions on paper (not file). To hand them in: give it to me or slide them under my door West Village H (246).

Problem 1. CLRS 8.2-2 + 8.3-3 + 9.3-3. More on sorting:

(1)Give a deterministic algorithm to find the pivot in the quicksort algorithm so that the resulting sorting algorithm is a deterministic algorithm performing $O(n \cdot \log n)$ comparisons in the worst case.

Consider sorting n elements with respect to their keys stored in array $A[1..n]$. Recall a sorting algorithm is *stable* if for any two elements with the same key, their relative order in the sorted output sequence B is the same as their relative order in the input sequence.

Now recall Counting Sort to sort keys that are at most k:

```
for i := 1 to k do
    c[i] := 0
for j := 1 to n do
    c[A[j]] := c[A[j]] + 1
//c[i] now contains the number of elements equal to i
for i :=  2 to k do
    c[i] := c[i] + c[i-1]
// c[i] now contains the number of elements <= i
for j := n downto 1 do
    B[c[A[j]]] := A[j]
    c[A[j]] := c[A[j]] - 1
```

(2) Prove that counting sort is stable.

(3) Now consider using Counting Sort inside Radix Sort to sort n elements between 0 and $n^2 - 1$. Prove that Radix Sort works.

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 $t \cdot \text{poly} \log t$ (i.e., if a, b are two integers such that $MAX(|a|, |b|) \leq t$ then $a + b, a \cdot t, \ldots$ can be computed in time $t \cdot \text{poly} \log t$.

(1) 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 $n \cdot \text{poly} \log n$.

(2) 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 $n \cdot \text{poly} \log n$.

(3) (Challenge!) 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 $n \cdot \text{poly} \log n$.

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 (which you should define) 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.