G22.1170: Fundamental Algorithms
Problem Set 1
(Due Tuesday, October 24, 2000)

The problems in this problem set involve notations for orders of growth (*big-oh*, *big-omega*, *big-theta*) as well as different techniques used in the analysis of algorithms (such as *recursion-tree*, *telescoping*, *summing factors*, *range-* and *domain-transformations*). The first three problems are rather easy, but the last problem may require some thought. You may consult the Hand Out 1, which discusses these materials in depth. Try to give the exact solutions to the recurrence equations wherever possible.

Problem 0.1 a. Order the following functions by their growth rate (the most slowly-growing function appearing first); if two functions are same, group them together.

(1)	1	(2)	7	(3)	$7^{\lg n}$
(4)	$(\lg n)^{\lg n}$	(5)	$\sqrt{n}\lg^2 n$	(6)	n
(7)	$n \lg n$	(8)	$n^{\frac{1}{\lg n}}$	(9)	$n^{\lg 7}$
(10)	$n^{1+\frac{\lg\lg n}{\lg n}}$	(11)	$n^{\lg\lg n}$	(12)	$\left(1-\frac{1}{n}\right)^n$
(13)	$\left(1-\frac{1}{7}\right)^n$	(14)	$\left(1+\frac{1}{n}\right)^n$	(15)	$\left(1+\frac{1}{7}\right)^n$

b. Suppose $T_1(n)$ is $\Omega(f(n))$ and $T_2(n)$ is $\Omega(g(n))$. Which of the following statements are true? Justify your answer.

1. $T_1(n) + T_2(n) = \Omega\left(\max(f(n), g(n))\right).$
2. $T_1(n)\, T_2(n) = \Omega\left(f(n)\, g(n)\right).$

c. Let us change the definition of Ω, as follows: $T(n) = \Omega(f(n))$, if there is a positive constant C such that

$$T(n) \geq C \cdot f(n), \quad \text{infinitely often, } i.e.\text{, for infinitely many values of } n.$$

Now answer the previous question for this definition of Ω.

Problem 0.2 a. Solve the following recurrence equation

$$T(n) = T(n-1) + 4\, n^3.$$

b. One can write a self-recursive algorithm to tile a mutilated checkerboard using L-trominoes. (See the Solutions to Problem Set 0, for the sketch of an algorithm.) Write down the recurrence equation for this algorithm, assuming that it takes a unit time step to place an L-tromino on the checkerboard. Show that the time, $T(N)$, it takes to tile a chekerboard of size $2^N \times 2^N$ is given by

$$T(N) = \frac{4^N - 1}{3} = \frac{2^N \cdot 2^N - 1}{3},$$

i.e., the number of tiles required to tile the checkerboard.

Problem 0.3 What are the time-complexities of the two Fibonacci Programs presented in the Problem Set 0? Present the complexities using the O notations. You may count only the additions and assume that each addition takes $O(1)$ time irrespective of the size of the summands.

Hint: Finding the exact complexity of Program 1 may be hard, and you need tools not discussed in the class. You may use a guess based on the experiments, you ran on the programs and then verify your guess. But if you are going to do an exact analysis then the following fact will come handy:

$$\text{Fibonacci}(n) = \frac{\phi^n - \psi^n}{\sqrt{5}}, \quad \text{where } \phi = \frac{1+\sqrt{5}}{2} \text{ and } \psi = \frac{1-\sqrt{5}}{2}.$$

Problem 0.4 Use Recursion-Tree to solve the following Recurrence Equations:

1.

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(0.8\,n) + n, \quad \text{if } n > 1. \end{aligned}$$

2.

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(0.8\,n) + T(0.1\,n) + n, \quad \text{if } n > 1. \end{aligned}$$

3.

$$\begin{aligned} T(1) &= 1 \\ T(n) &= T(0.8\,n) + T(0.2\,n) + n, \quad \text{if } n > 1. \end{aligned}$$

Problem 0.5 a. Dr. Supe R.Hacker of Happy Hackers, Inc. has discovered a family of algorithms, A_1, A_2, A_3, ... The complexity of these algorithms are given by the following set of recurrence equations.

$$T_1(1) = T_2(1) = \cdots = T_m(1) = \cdots = 1,$$

and for $n > 1$,

$$\begin{aligned} T_1(n) &= 2\,T_1(\tfrac{n}{2}) + n \\ T_2(n) &= 2\,T_2(\tfrac{n}{2}) + T_1(n) \\ &\vdots \\ T_m(n) &= 2\,T_m(\tfrac{n}{2}) + T_{m-1}(n) \\ &\vdots \end{aligned}$$

What is the complexity of the m^{th} algorithm A_m?

b. Dr. Dupe R.Hacker, also, of Happy Hackers, Inc. has more recently discovered another set of algorithms, A'_1, A'_2, A'_3, ... The complexity of these algorithms are given by the following set of recurrence equations.

$$T_0(n) = T_1(1) = T_2(1) = \cdots = T_m(1) = \cdots = 1,$$

and for $n > 1$,

$$\begin{aligned} T_1(n) &= 4\,T_0(\tfrac{n}{2}) + n \\ T_2(n) &= 8\,T_1(\tfrac{n}{4}) + n \\ &\vdots \\ T_m(n) &= 2^{m+1} T_{m-1}(\tfrac{n}{2^m}) + n \\ &\vdots \end{aligned}$$

What is the complexity of the m^{th} algorithm A'_m?

c. For what values of m are the Supe R's algorithms more efficient than Dupe R's? How about the converse? Is there any m for which both Supe R's and Dupe R's algorithms have comparable efficiency? Justify your answer(s).