CSE 331: Introduction to Algorithm Analysis and Design Fall 2011

HOMEWORK 2

Due Friday, September 23, 2011 by 1:15pm in class

IMPORTANT: Please submit each problem separately, i.e. each problem should begin on a new page and only the pages for one problem should be stapled together. Failure to do so might result in some problem(s) not being graded.

For general homework policies and our suggestions, please see the policy document.

Sections 2.2 and 2.4 from the textbook might be useful for this homework.

For this homework, you can assume that addition, subtraction, multiplication and division of two numbers can be done in $O(1)$ time. (Side question: Is this assumption justified?)

Do not turn the first problem in.

---

1. (**Do NOT turn this problem in**) This problem is just to get you thinking about asymptotic analysis and input sizes.

   An integer $n \geq 2$ is a prime, if the only divisors it has is 1 and n. Consider the following algorithm to check if the given number n is prime or not:

   > For every integer $2 \leq i \leq \sqrt{n}$, check if i divides n. If so declare n to be *not* a prime. If no such i exists, declare n to be a prime.

   What is the function $f(n)$ such that the algorithm above has running time $\Theta(f(n))$? Is this a polynomial running time– justify your answer. (A tangential question: Why is the algorithm correct?)

2. (40 points) Order the following running times in ascending order so that if one (let's call it $f(n)$) comes before another (let's call it $g(n)$), then $f(n)$ is $O(g(n))$:

   $$2^{n^{\sqrt{2}}},\ n^3,\ 2^{(\log_{96} n)^{1.0000001}},\ 2^{\log_4 n},\ n^n,\ n! + 10^{1000}.$$

   Briefly argue why your order is correct (formal proof is not needed).

   (For this problem, it might help to know Stirling's approximation:

   $$n! = \sqrt{2\pi n}\left(\frac{n}{e}\right)^n e^{\lambda_n},$$

   where

   $$\frac{1}{12n+1} < \lambda_n < \frac{1}{12n}.$$

   Also recall that $x = \log_a b$ implies that $a^x = b$.)

   (*Hint*: It *might* be useful to first rank all the functions other than $n!+10^{1000}$ and $2^{(\log_{96} n)^{1.0000001}}$. Once you are done with rest of the four functions, put $2^{(\log_{96} n)^{1.0000001}}$ in its correct place and then put the $n! + 10^{1000}$ function in its correct position once you are done with the rest.)

3. ($30 + 15 = 45$ points) We will consider sorting in this problem. Recall that given n integers $a_1, \ldots, a_n$, its (ascending) sorted order are the numbers from the smallest to the largest. So for example, given

$$3, 17, 1, -9, 100, 51, 1$$

the sorted order is

$$-9, 1, 1, 3, 17, 51, 100.$$

(Note that the numbers need not be distinct.) Consider the following algorithm for sorting n numbers that are assumed be given as an array $A[1..n]$:

(a) $\texttt{max} \leftarrow n$.

(b) While $\texttt{max} > 1$ repeat:

i. Let $1 \leq j \leq \texttt{max}$ be the index such that $A[j]$ is the largest number among $A[1], \ldots, A[\texttt{max}]$.

ii. Swap the values in $A[\texttt{max}]$ and $A[j]$.

iii. Decrement max by 1.

Convince yourself that the algorithm above does compute the sorted order of the numbers $A[1], \ldots, A[n]$. (You do *not* have to include the proof of correctness in your submission.) Consider the following questions related to the run time of the algorithm:

- Figure out a $f(n)$ such that the running time of the algorithm above is $O(f(n))$. Prove your bound on the running time.
- Figure out a $g(n)$ such that the running time of the algorithm above is $\Omega(g(n))$. Prove your bound on the running time.

(*Hint:* You can pick $g(n)$ and $f(n)$ such that $g(n)$ is $\Theta(f(n))$.)

4. (15 points) Exercise 6 in Chapter 1.

(*Note/Hint:* In real life, you will almost never come across a problem whose description will match exactly with one you will see in this course. More often, you will come across problems that you have seen earlier *but* are stated in a way that don't look like the version you have seen earlier. *One way* to solve this problem would be to to simulate that situation. In algorithms-speak, you can to *reduce* the problem here to one that you have seen already.)