**CSE 331: Introduction to Algorithm Analysis and Design** Fall 2012

HOMEWORK 2

**Due Friday, September 21, 2012 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) Exercise 4 in Chapter 1.

3. (45 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 $O(g(n))$:

   $$2^{n^{1.00001}},\ n^{3^{10}},\ 2^{(\log_{96} n)^{\sqrt{2}-0.4}},\ 9^{\log_{10.1} n},\ n^n,\ 10^{1000} \cdot n!.$$

   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 $10^{1000} \cdot n!$ and $2^{(\log_{96} n)^{\sqrt{2}-0.4}}$. Once you are done with rest of the four functions, put $2^{(\log_{96} n)^{\sqrt{2}-0.4}}$ in its correct place and then put the $10^{1000} \cdot n!$ function in its correct position once you are done with the rest.)

4. (15 points) The Big G company in the bay area decides it has not been doing enough to hire CSE grads from UB so it decides to do an exclusive recruitment drive for UB students. The Big G decides to fly over $n$ CSE majors from UB to the bay area during December for on-site interview on a single day. The company sets up $m$ slots in the day and arranges for $n$ Big G engineers to interview the $n$ UB CSE majors. (You can and should assume that $m > n$.) The fabulous scheduling algorithms at Big G's offices draws up a schedule for each of the $n$ majors so that the following conditions are satisfied:

   - Each CSE major talks with every Big G engineer exactly once;
   - No two CSE majors meet the same Big G engineer in the same time slot; and
   - No two Big G engineers meet the same CSE major in the same time slot.

   In between the schedule being fixed and the CSE majors being flown over, the Big G engineers were very impressed with the CVs of the CSE majors (including, ahem, their performance in CSE 331) and decide that Big G should hire all of the $n$ UB CSE majors. They decide as a group that it would make sense to assign each CSE major $C$ to a Big G engineer $E$ in such a way that after $C$ meets $E$ during her/his scheduled slot, all of $C$'s and $E$'s subsequent meetings are canceled. Given that this is December, the Big G engineers figure that taking the CSE majors out to the nice farmer market at the ferry building in San Francisco during a sunny December day would be a good way to entice the CSE majors to the bay area.

   In other words, the goal for each engineer $E$ and the major $C$ who gets assigned to her/him, is to *truncate* both of their schedules after their meeting and cancel all subsequent meeting, so that no major gets *stood-up*. A major $C$ is stood-up if when $C$ arrives to meet with $E$ on her/his truncated schedule and $E$ has already left for the day with some other major $C'$.

   Design an *efficient* algorithm that always finds a valid truncation of the original schedules so that no CSE major gets stood-up.

   To help you get a grasp of the problem, consider the following example for $n = 2$ and $m = 4$. Let the majors be $C_1$ and $C_2$ and the Big G engineers be $E_1$ and $E_2$. Suppose $C_1$'s original schedule is

   $E_1$; lunch; $E_2$; dinner

   and $C_2$'s original schedule is

   Breakfast; $E_1$; coffee break; $E_2$.

   In this case the (only) valid truncation is for $C_1$ to get assigned to $E_2$ in the third slot and for $C_2$ to get assigned to $E_1$ in the second slot. (And as a bonus all four get to have dinner!)

   (*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.)