CSE 332 Data Abstractions, Winter 2011

Homework 1

Due Friday, Jan 14, 2011 at the **beginning** of lecture. Please be sure your work is readable (either written clearly or typed). This homework has six problems.

Problem 1: Recurrence Relations

Consider the following recurrence relation, similar to one seen in lecture 3:

T(1)=5, and for n greater than 1, $T(n) = 1 + 2T(\lfloor n/2 \rfloor)$

Note: $\lfloor n/2 \rfloor$ is the 'floor' of 'n/2': it rounds down to the next largest integer.

a. Give T(n) for n=integers 1 through 8

b. Expand the recurrence relation to get the closed form, as seen in lecture #3. Show your work; do not just show the final equation.

Problem 2: Induction

For the following inductive proofs, clearly state the base case, induction step and inductive hypothesis, as described in lecture.

a. Prove Weiss 1.12.a by induction

b. Prove the following by induction: $\sum_{i=1}^{n} i = \frac{n(n+1)}{2}$

Problem 3: Run-times

Say we have 4 versions of a program we'd like to run on some input n. Each takes a certain amount of time to run, as a function of n:

- P1 takes n days to run
- P2 takes n^2 days to run
- P3 takes 2^n days to run
- P4 takes $\log_2 n$ days to run

So, to run P2 on an n of 4 would take 16 days.

a. For each version of the program, calculate the value of n (rounded down) we could compute if we let the program run for 12 billion years , which is (very) roughly how long until the Earth's sun dies. You can leave the answer in scientific notation, or as 2 raised to some power in scientific notation.

b. Let's say we have access to a computer which runs one million times faster than the one above; so we could compute P1(1) in one millionth of a day. Write out the n we could compute for each of the above algorithms given this new processing power.

(See back of this page for remaining problems)

Problem 4: Horner's Rule

The classic way to evaluate a polynomial is called Horner's rule, which can be stated recursively as follows:

Let $p(x) = a_0 + a_1x + a_2x^2 + \cdot\cdot\cdot + a_nx^n$

To compute $p(c)$ for some constant c, first evaluate $q(c)$ where :

$$q(x) = a_1 + a_2x + \cdot\cdot\cdot + a_nx^{n-1}$$

recursively, then:

$$p(c) = a_0 + cq(c).$$

1. Provide a base case for this method. That is, explain how to do the "last step" without recursion.
2. Prove, by induction, that Horner's method, including your base case, works for any n. That is, prove it works for a polynomial of any degree.
3. For a polynomial of degree n, as a function of n, how many additions and how many multiplications are used to evaluate the polynomial in Horner's rule.
4. Provide an elegant, **non-recursive** pseudocode function for Horner's rule where the coefficients are stored in an array A[0…n], with A[i] containing a_i. Hint: this can be done in about 5 lines of code

Problem 5. Big Oh Notation

For each of the following, either prove true (using our definitions for Big Oh, Big Omega and Big Theta, as appropriate) or explain why it is false:

a. If we have an algorithm that runs in O(n) time, and make some changes that cause it to run 10 times slower, it will still run in O(n) time.

b. If $f(n) = O(g(n))$ and $h(n) = O(k(n))$, then $f(n) - h(n) = O(g(n) - k(n))$.

c. If $f(n) = O(g(n))$ and $h(n) = O(k(n))$, then $f(n) + h(n) = O(g(n) + k(n))$.

d. $(2^{n+1}) = \Theta(2^n)$

e. $(2^n)^{1/3} = \Theta(2^n)$

Problem 6. Algorithm analysis

Weiss 2.7a (give the best big-O bound you can for each of the 6 program fragments; you do not need to explain why).