FALL 2008: **COT 5407** INTRO. TO ALGORITHMS

[HOMEWORK 1; DUE SEP 23 AT START OF CLASS]

General submission guidelines and policies: ADD THE FOLLOWING SIGNED STATEMENT. Without this statement, your homework will not be graded.

> I HAVE ADHERED TO THE COLLABORATION POLICY FOR THIS CLASS. IN OTHER WORDS, EVERYTHING WRITTEN DOWN IN THIS SUBMISSION IS MY OWN WORK. FOR PROBLEMS WHERE I RECEIVED ANY HELP, I HAVE CITED THE SOURCE, AND/OR NAMED THE COLLABORATOR.

Read the handout on **Homework guidelines and collaboration policy** from your course website before you start on this homework. This is very important.

Problems

0. (**Regular**) Did you follow the instructions above?

1. (**Exercise**) Write down the time complexities of performing LINEARSEARCH and BINARYSEARCH in a sorted array of n elements.

2. (**Exercise**) Write down the precise **invariants** for each of the following algorithms: SELECTIONSORT, INSERTIONSORT, BUBBLESORT, SHAKERSORT, and MERGE (not MERGESORT).

3. (**Regular**) Given two monotonically increasing functions, $f(n)$ and $g(n)$, prove or disprove:
$$\min(f(n), g(n)) = \Theta(f(n) + g(n)).$$
Redo the above problem with min replaced with max.

4. (**Regular**) (Exercise 4-1(c,d), p85) Use the *Master method* to solve the recurrences in (a), (b), and (c). You may assume that $T(n)$ is constant for $n < 2$.
   (a) $T(n) = 15T(n/4) + 4n^2$.
   (b) $T(n) = 27T(n/3) + 3n^3$.
   (c) $T(n) = 18T(n/2) + 2n^4$.

5. (**Regular**) Solve one of the problems in Problem 4 above using the *Substitution method*.

6. (**Regular**) Show that for any real constants a and b, where $b > 0$,
$$(n - a)^b = \Theta(n^b).$$
Note that $f(n) = \Theta(g(n)$ if and only if $f(n) = O(g(n))$ and $g(n) = O(f(n))$.

7. (**Exercise**) (Exercise 3-2(e), p58) If $f(n) = n^{\log_2 c}$ and $g(n) = c^{\log_2 n}$, indicate which of these relationships are true and prove your answers: $f(n) = O(g(n))$, $f(n) = \Omega(g(n))$, and $f(n) = \Theta(g(n))$.

8. (**Extra Credit**) In our first class (Sep 2), we discussed and analyzed a simple algorithm for the SEARCH problem. We discussed two variants – one where X, the number to be searched, was bounded on both sides, and another where x was bounded below, but unbounded above. Binary search was the best strategy for the first version. The best strategy for the second version involved doing a doubling search followed by a binary search. This could be thought of as doing a LINEARSEARCH for m, the smallest exponent of 2 greater than x. What if we consider doing doubling search for m? Can we push this even further? Analyze the best algorithm for this problem.