FALL 2008: **COT 5407** Intro. to Algorithms

[Homework 2; Due Oct 2 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**.

Problems

9. (**Exercise**) Solve as many problems as possible from Exercise 3-3(a) p58.

10. (**Regular**) Solve Exercise 4.1-5 p67. Solve using the substitution method.

11. (**Regular**) Solve Exercise 4.2-5 p72. Solve using the recursion-tree method.

12. (**Exercise**) Solve as many problems as possible from 4-1 and 4-4 p85-86.

13. (**Exercise**) Solve Exercise 7.1-1, p148; Exercise 7.2-2, p153; Exercise 7.2-4, p153;

14. (**Regular**) Study Randomized-Partition and Randomized-QuickSort from p154.
    (a) Implement the **median-of-3** method for choosing the pivot (described on p162);
    (b) Implement R. Sedgewick's idea (1978) to avoid recursive calls when the size of the array is at most k. Set the value of k to be 8, which is the cutoff value used in the 1997 Microsoft C library implementation of QuickSort. Use InsertionSort for arrays of size at most k. There is no need to show code for InsertionSort.

15. (**Regular**) Sorting algorithms are not constrained in the way they treat "equal" items in the input list. Therefore, if all the items in a list are equal, any permutation of the inputs is a correct output. A sorting algorithm is said to be *stable* if the relative order of any equal items in the input list is not changed in the output list. For example, if your list contained scores of students in an exam. Say that your input list has Adam with a score of 78 in location 7, and Alice with a score of 78 in location 11, Then after a *stable* sorting based on exam scores, the output list is guaranteed to have Adam appearing before Alice. An *unstable* sort provides no such guarantee. The output list (although it is still sorted) may or may not have Adam appearing before Alice.

    In order to show that a sorting algorithm is stable, one would need a mathematical proof. However, to prove that it is not stable, all we need is a simple "counterexample".

It should be obvious to you that InsertionSort, BubbleSort, and MergeSort are stable sorting algorithms (first convince yourself of this). For each of the following sorting algorithms, state which ones are **not** stable with the help of simple examples: Selection Sort, and , QuickSort. Devise the smallest example you can build, if you claim the algorithm is not stable. Write down a brief argument if you think it is stable. You should consider a sorting algorithm to be stable if the algorithm given to you in the book or the one given to you in class (or a minor variant thereof) is stable.