22C:44 Homework 1

Due in class on Tuesday, Sept 5, 2000

Each problem is worth 10 points.

1. One problem with the BubbleSort function presented in class is that it's running time $\Theta(n^2)$ even when the input is a sorted sequence of n numbers. Here is an attempt to fix that problem. An *inversion* of a sequence $A = a_1, a_2, \ldots, a_n$ is a pair of integers (i, j), $1 \leq i < j \leq n$ for which $a_i > a_j$. Thus an inversion is a pair of numbers in the sequence that are out of order and the total number of inversions of a sequence represents the amount of "disorder" in that sequence.

   (a) [2 points] Show that, with each swap BubbleSort decreases the total number of inversions of the input sequence by 1.

   (b) [1 point] Even if there are very few swaps to be made, in order to discover the swaps, BubbleSort makes lots of comparisons. For the sequence $1, 2, 4, 3, 5$ how many swaps does BubbleSort make and how may comparisons does it make? By "comparisons" I mean comparisons between pairs of elements in the sequence.

   (c) [2 points] Now consider the following BetterBubbleSort

```
BetterBubbleSort( A[1..n] ) {
        done ← False; i ← 1;
        while (!done) && (i < n) do {
                done ← True;
                for j ← n downto i+1 do {
                        if (A[j-1] > A[j]) then {
                                done ← False;
                                swap(A, j-1, j);
                        }
                }
                i ← i + 1;
        }
}
```

   For the sequence $1, 2, 4, 3, 5$ how many swaps does BetterBubbleSort make? How may comparisons does it make?

   (d) [2 points] Show that the running time of BetterBubbleSort is $O(nI)$ where n is the number of elements in the input sequence and I is the total number of inversions of the input sequence.

   (e) [1 point] True or False: The running time of BetterBubbleSort is $\Theta(nI)$, where n and I are as in part (d).

   (f) [2 points] Your friend claims that the running time of BetterBubbleSort is $\Theta(n + I)$. Show that this claim is false. Again n and I are as in part (d).

2. Partition the following set of 15 functions into equivalence classes such that $f(n)$ and $g(n)$ are in the same equivalence class iff $f(n) = \Theta(g(n))$.

$$\begin{array}{ccccc} 4^{\lg n} & n^n & 8 + 1/e^n & (n+1)! & 2^n \\ \lg(n!) & 1 & n^2 & 3^n + n^2 & \sqrt{\lg(n)} \\ n! & n^n + n! & 3^n & n \log_{10} n & \lg \sqrt{(n)} \end{array}$$

   Then arrange the equivalence classes into a sequence

$$C_1, C_2, C_3, \ldots$$

   such that for any $f(n) \in C_i$ and $g(n) \in C_{i+1}$ $f(n) = O(g(n))$.

3. Prove or disprove the following:

   (a) $f(n) + g(n) = \Theta(\min(f(n), g(n)))$

   (b) $f(n) = \Theta(f(n/2))$

   (c) $\sqrt{n} \lg n = \Omega(n)$

   (d) $n^2 / \lg^2(n) = O(n/\lg(n))$

4. The *Fibonacci* sequence is defined as: $F_1 = 1$, $F_2 = 1$ and $F_n = F_{n-1} + F_{n-2}$ for all integers $n > 2$. So the first few elements of this sequence are

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, \ldots$$

   Here is a recursive function that computes F_n given n.

```
Fibonacci(integer n) {
        if (n == 1) || (n == 2)
                return 1
        else
                return Fibonacci(n-1) + Fibonacci(n-2)
}
```

   (a) [2 points] Let $A(n)$ denote the total number of additions performed by the above function in calculating `Fibonacci(n)`. Clearly, $A(1) = A(2) = 0$ and $A(3) = 1$. Setup a recurrence relation for $A(n)$ for all integers $n > 2$.

   (b) [2 points] Show that

$$A(n) = \frac{\phi^n - \hat{\phi}^n}{\sqrt{5}} - 1.$$

   Here $\phi = (1 + \sqrt{5})/2$ is the *golden ratio* and $\hat{\phi} = (1 - \sqrt{5})/2$ is called the conjugate of the golden ratio.
   (**Hint:** First read up the material on Fibonacci numbers on pages 36-37 in your textbook. Then write down the first few elements in the sequence $A(1), A(2), A(3), \ldots$ and derive a connection between $A(n)$ and F_n.)

   (c) [2 points] Express $A(n)$ as $\Theta(f(n))$ for some $f(n)$. Make $f(n)$ as simple as possible.

   (d) [4 points] Rewrite `Fibonacci(n)` so that it is as efficient as possible. Analyse the running time of this function.