22C:44 Homework 1

Due by 5 pm on Thursday, 2/1

Problem 1 is worth 20 points, 10 for each part. The other three problems are worth 10 points each.

1. Yet another simple sorting algorithm is called *Bubble Sort*. Given below is an implementation of this.

```
BubbleSort( A[1..n] ) {
      for i ← 1 to n-1 do
            for j ← n downto i+1 do
                  if (A[j-1] > A[j]) then {
                        t ← A[j]; A[j] ← A[j-1]; A[j-1] ← t;
                  }
}
```

   (a) Analyze the above implementation of `BubbleSort` *exactly* in the model of computation defined in class and determine its worst case running time as a function of n. In other words, mimic the analysis of `InsertionSort` we did in class, paying close attention to the exact number of time units each statement takes. For full credit show all your calculations.

   (b) In earlier versions of *Mathematica*, changing the value of a single element in a large data structure meant rewriting the entire data structure into memory. Programming languages that have this underlying model of computation are called *applicative languages*. The following problem is motivated by the existence of applicative languages.

   Suppose that in the above implementation of Bubble Sort, each write-operation on A takes n times units, where n is the size of the array A. Accessing `A[j]` for any given j would still take 1 unit of time, but assigning a new value to `A[j]` would take n units of time. For example, the statement `A[j-1] ← t` would take

   1 time unit to get `t`
   2 time units to get `j` and calculate `j-1`
   n time units to assign the value `t` to `A[j-1]`

   for a total of $n + 3$ time units.

   Given this, somewhat strange, model of computation do an *exact* worst case analysis of the running time of `BubbleSort`.

2. The *upper triangle* of an $n \times n$ matrix M is defined as the entries $M[i, j]$ for which $i \leq j$. The following is an implementation of a matrix multiplication algorithm that takes $n \times n$ matrices A and B and multiplies them assuming that A and B have non-zero entries only in their respective upper triangles.

```
MatrixMultiply( A[1..n, 1..n], B[1..n, 1..n] ) {
      Define C[1..n, 1..n];
      for i ← 1 to n do
            for j ← 1 to n do {
                  C[i, j] ← 0;
                  for k ← 1 to min{i, j} do
                        C[i, j] ← C[i, j] + A[i, k] * A[k, j];
            }
}
```

In the above code, `min{i, j}` computes and returns the minimum value among `i` and `j`.

Analyze this algorithm and determine its running time as a function of n. Express your answer as $\Theta(f(n))$. So you are being asked not for an exact analysis, but only an asymptotic analysis. Show all your calculations for full credit.

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

$\frac{n^2}{\ln^2 n}$	$\lg\lg n$	$(n+10)^5$	$((n+16)(8n^{0.5}+\lg n))$	$\lg^2 n$
$\lg(n!)$	$\frac{n}{\lg n}$	$\frac{1}{n}$	$7n^5 - 30n + 2$	$\frac{8n}{n!} + 20$
20	$\frac{8n^2}{\lg^2 n} + n\lg n$	$2^{\lg^2 n}$	$n^{\lg n} + 80n^5$	$n^{3/2}$

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 $g(n) \in C_{i+1}$ $f(n) = o(g(n))$.

4. Prove or disprove the following:
   (a) $f(n) + g(n) = \Theta(\max(f(n), g(n)))$
   (b) $f(n) + o(f(n)) = \Theta(f(n))$
   (c) $f(n) = O(f(n)^2)$
   (d) $(n+1)^2 = O(n^2)$
