Lab 4

1. Improve the BubbleSort implementation so that in the best case (which means here that the input is already sorted), the algorithm runs in O(n) time. Explain why your new version works --- in other words, prove that the best case running time of your code is O(n). Call your new Java file BubbleSort1.java.

2. Recall that in BubbleSort, at the end of the first pass through the outer loop, the largest element of the array is in its final sorted position. After the next pass, the next largest element is in its final sorted position. After the *i*th pass (i=0,1,2,…), the largest, second largest,…, i+1st largest elements are in their final sorted position. Use this observation to cut the running time of BubbleSort in half. Implement your solution in code, and prove that you have improved the running time in this way. Call your new Java file, which contains the improvements from this problem and the previous problem, BubbleSort2.java.

3. In this lab folder, I have given you an environment for testing sorting routines. Insert into this environment the original BubbleSort file along with your new BubbleSort1 and BubbleSort2 classes, and run the SortTester class. What are the results? Are the results what you expected? Explain why the running times turned out in the way that they did.

4. The following algorithm is intended to compute the value of m^n for any positive integer m and any nonnegative integer n.

```
long exp(long m, int n) {
   if(n == 0) return 1;
   if(n == 1) return m;
   if(n % 2 == 0) return exp(m*m, n/2);
   else return exp(m*m, n/2) * m;
}
```

   (a) Prove the algorithm is correct by arguing by induction on n. That is, for a fixed $m \geq 1$, show that for each n, the output of `exp` on input m, n is m^n.
   (b) Compute the asymptotic running time of this algorithm. Prove your answer using either the Guessing Method or the Master Formula.

5. Suppose S consists of n operations $p_1, p_2, \ldots, p_n$. Suppose that the running time of p_i is $\Theta(i)$ if i is a *power* of 3 (such as 3, 9, 27, 81, etc), but is O(1) in all other cases. What is the amortized running time of each operation? Prove your answer by defining an amortized cost function, showing that it bounds S, and performing the necessary computation of the average amortized cost of the operations in S.

   Warning. Do not attempt to compute the actual running time of the operations, collectively or individually. Doing so results in a complicated formula and defeats the purpose of the exercise. One of the purposes of amortized analysis is to give an easy-to-calclate bound for the actual running time.

6. An array A holds n integers, and all integers in A belong to the set {0, 1, 2}. Describe an O(n) sorting algorithm for putting A in sorted order. Your algorithm may not make use of auxiliary storage such as arrays or hashtables (more precisely, the only additional space used, beyond the given array, is O(1)). Give an argument to explain why your algorithm runs in O(n) time.