CMPSC 250
Analysis of Algorithms
Fall 2012
Bob Roos

Lab 3
Tues., 18 September 2012
Hand In by 2:45pm Tues., 25 Sept.

I strongly suggest that you create a new subdirectory called "lab3" or something similar and place all your files from this lab in that directory. At the end of the assignment I'll ask you to create a "zip" file to upload to the Sakai drop box.

Inversions

We often measure the "unsortedness" of an array by determining how many pairs of entries are out of order with respect to each other. In other words, we count pairs `a[i]`, `a[j]` for which $i < j$ but `a[i]` $>$ `a[j]`. Such a pair is called an *inversion*—see page 252 in your text. For instance, in the array:

a:	8	10	3	0	12	9	11	4
	0	1	2	3	4	5	6	7

there are 13 inversions:

```
a[0] = 8, a[2] = 3
a[0] = 8, a[3] = 0
a[0] = 8, a[7] = 4

a[1] = 10, a[2] = 3
a[1] = 10, a[3] = 0
a[1] = 10, a[5] = 9
a[1] = 10, a[7] = 4

a[2] = 3, a[3] = 0

a[4] = 12, a[5] = 9
a[4] = 12, a[6] = 11
a[4] = 12, a[7] = 4

a[5] = 9, a[7] = 4

a[6] = 11, a[7] = 4
```

1. Create a document for answers to some of the questions below. BE SURE TO PUT YOUR NAME, THE DATE, AND THE LAB NUMBER at the top of the file so I'll be able to identify it as yours when I print it out!

   In your document, list all the inversions for the following array of strings. Just give the array indices of the inversions, e.g., if `a[3]` $>$ `a[6]` just write "`3, 6`".

a:	"adze"	"pyx"	"woofer"	"zither"	"kerf"	"zloty"	"iamb"
	0	1	2	3	4	5	6

2. (a) Download file `Lab3Part1.java` and complete it by writing a `static` Java method named "`inversions`" that takes one argument, a `Comparable` array, and returns an `int` equal to the number of inversions in the array. (You will need to use the `compareTo` method to compare elements `a[i]` and `a[j]` for the different values of `i` and `j`.) Make sure the results for the three given sample arrays are correct (you should be able to check these by hand.)

   (b) Now add some code at the bottom of the `main` method that creates an `int` array of 30 random values from 0 through 99 (use `StdRandom.uniform` for this) and then counts the inversions in it using your `inversions` method.

   Run your program at least ten times (more is better) and compute the average number of inversions that you see for the 30-element array. (Optional—Can you get your program to do this?)

   In your document, show all the outputs (you can comment out the first three calls to "`inversions`" and only show the output for the 30-element random array) and show your calculated average.

   (c) What is the maximum number of inversions possible in a 30-element array? How does your calculated average compare to this maximum? (For instance, is it close to the maximum, or is it some recognizable percentage of the maximum?)

   (d) What is a general formula for the maximum possible number of inversions in an array of size n? How did you obtain it?

3. Download the programs "`Insertion.java`" and "`Selection.java`" from the Sakai site. These are modified versions of the ones on the textbook web site. The modifications permit arrays of different sizes to be filled with random values and sorted, printing out execution times of each sort.

   For each sorting method, determine a constant of proportionality between the running time and the function n^2. You don't need to plot a graph. Feel free to increase the maximum value of n (I set it at 20000 but it can be higher).

   In particular, try to find values c_i and c_s such that $T_{insertion}(n) \approx c_i n^2$ and $T_{selection}(n) \approx c_s n^2$. Would you expect there to be a noticeable difference in the constants? Why or why not? Do you have any other observations? (For instance, was there a clear-cut convergence in both cases, or was there a lot of "noise"?)

4. In both programs the initialization of arrays was preceded by a call to `StdRandom.setSeed`. What does this method do? Why is it critical for purposes of comparison that both programs execute `setSeed` with the same value?

5. Download "`Bubble.java`" from the Sakai site. Complete the "`sort`" method to implement "bubble sort." Try to determine a constant c_b such that $T_{bubble}(n) = c_b n^2$. If you locate a bubble sort algorithm on line, you may adapt it, but please give credit to the source. Note that the `sort` method must be compatible with the `Bubble.java` program I've provided—it should be a `void` method, it must take a single argument of type `Comparable[]`, it should not read or print anything.

6. Move up one level from your `lab3` directory and type "`zip -r lastnamelab3.zip lab3`" (if you named it something else, use the something else). Now upload this to your drop box.