Programming Project #1
CS3410

Compare the execution times for Insertion Sort and Quicksort. You will implement each of these algorithms in either C++ or Java, and then do empirical testing to determine for which input sizes Insertion Sort is faster and for which Quicksort is faster.

Specific guidelines:

a. The data to be sorted can just be type int. You don't need to put any extra work into the algorithms such that they can handle multiple data types; i.e., you don't have to worry about Comparators, etc. like we did in Data Structures.
b. You will work in teams of 2.
c. Both algorithms should be coded in the same language, and using the same style (i.e., try to make the quality of the code approximately the same).
d. You can just use a loop and the randomizer to create the array to be sorted.
e. You will need to instrument the code to accurately measure the amount of time required for the sort. For accuracy, don't just use the Unix-level time command to time the entire program execution. Instead, time only the sort portion of the program; i.e., don't include the time required to initialize the array of numbers in the time totals. Recommend you look at *nanoTime*() from the System class.
f. I would like you to test the two algorithms performance on pre-sorted data (both in forward- and reverse-sorted order).
g. I would like you to code up both of the Partition algorithms looked at in class (the 1 and the 2-pointer methods), and analyze the differences. I would also like you to vary the way the pivot is chosen (e.g., last, median, median of 3) and see how that impacts performance on both random and pre-sorted data.
h. You can use any development environment you choose.

Required for turn-in:

a. A CS Cover Sheet
b. Listings of your code, including the two algorithms, variants, and test code (properly commented, and in compliance with the appropriate CS style guide).
c. Test results. This should include graphs comparing the overall algorithm performance and graphs comparing various options (like comparing partition algorithms. Produce results for large input sizes up to 1M points (this high for Quicksort only), but you will also need "close-ups" of interesting sections (like where the performance of the two algorithms crosses).
d. A discussion of how your results corresponded to what you expected (given the complexity of the algorithms) and how sensitive the results were to changes in the input. If results were not what was expected, offer an explanation for the results.
e. A signed statement regarding how well your final code worked. For example, "My code compiled and ran properly, and produced the correct output" or "My code compiled correctly, but core dumps when executed."
f. (optional) Comments on problems you ran into/lessons learned.