Program #1 (25 points)

OK, so you’ve heard all about four different sorting algorithms (selection, insertion, merge, and quick) and you know that some are supposed to be way more efficient than others. But hey, why take my word for it? You’re a programmer! Prove it! (The efficiency thing, I mean. I already know you’re a programmer.)

Write a program that:

• applies each of the sorting algorithms listed above to the same array of random numbers, and counts the operations each algorithm performs to achieve the sort, then reports the result as a kind of scorecard – example:

FOR A 2000-MEMBER ARRAY, RESULTS WERE: *

SELECTIONSORT: 40000532 OPERATIONS
INSERTIONSORT: 40001007 OPERATIONS
MERGESORT: 3078 OPERATIONS
QUICKSORT: 2986 OPERATIONS

* I totally made up these results. Yours, of course, will be real.

• performs the above tasks as many times as the user wants to, allowing the user to specify the size of the array each time
• performs the above tasks several times on different arrays of the same size, reporting results as they’re found, then calculating and displaying average results; for trial runs, use the largest arrays you can (sized with longs, for example)
• you will need to document how you are counting operations – i.e. what counts, and what doesn’t; try to be fair to all of your algorithms
• Determine through experimentation (and document) how large the data set needs to be before relative run-time efficiencies make a significant difference. You will need to document what “significant” means in this context.

Extra credit options (worth up to 5 points each):

• Present your results in some kind of graphical form. I will leave up to your creative mind exactly how to do this.
• Implement another sorting algorithm (several are available; some well-known sorts include bubble sort, radix sort, and bucket sort) and indicate where you found it, or, if you devised your own, explain your algorithm. Perform the same tests on this algorithm as those described for the algorithms presented in class.