**Programming Project 1**

1. [60%] Implement insertion sort, counting sort, merge sort, and *randomized* quick sort algorithms. (In a randomized quick sort algorithm, the pivot is selected randomly.) More specifically, your program should do the following:
2. Let a user select the *sorting algorithm* to use and specify the number of input data *n* where 1 ≤ n ≤ 1,000. Return an error message, if the selected sorting method is not supported by your program or n < 1 or n > 1,000.

Note 1: A textual command line interface is enough to do this. (Do not waste your precious time for a fancy GUI (Graphical User Interface). This is not a graphics course!)

Note 2: To generate random numbers, use srand(time(Null)) and rand() functions. You can find a lot of information about these functions online. For example, see <http://www.cplusplus.com/reference/clibrary/cstdlib/rand/> and <http://www.cplusplus.com/reference/clibrary/cstdlib/srand/>

1. Randomly generate an array of n positive integers and print them.

Note 3: For counting sort, ensure that the value of each randomly generated data item is an integer that ranges between 0 and 99. For the other sorting algorithms, data values are arbitrary integers returned by the pseudo random number generator.

1. Sort the randomly generated data in non-descending order using the selected sorting algorithm. Print the final result.
2. If n ≤ 20, create random numbers that range between 0 and 15 and do simple *textual visualization* of the sorting process as follows. (Do not do this if n > 20.) For a number *k* that israndomly generated in Step 2, print “\*” *k* times and move to a new line. For example, if the data to sort are (7, 3, 5), your program needs to display the following before starting to sort them:

\*\*\*\*\*\*\*

\*\*\*

\*\*\*\*\*

After executing one step of the selected sorting algorithm, print the array to show the current state. For instance, if the array becomes (3, 7, 5) after executing one step of the selected sorting algorithm, your program needs to print the following:

\*\*\*

\*\*\*\*\*\*\*

\*\*\*\*\*

1. [30%] Using the Method 1 or Method 2 discussed in class, compute the instruction counts for your insertion sort, counting sort, and merge sort implementations. Clearly state which method you use for instruction counts. (Note: You do not have to do this for quick sort.) Ensure that your counts are Ѳ(n2) for insertion sort, Ѳ(n) for counting sort, and Ѳ(nlgn) for merge sort. If not, either your implementation is suboptimal or your instruction counts are incorrect. Correct any problems before submitting your assignment.
2. [10%] 10% of the grade will be based on *good coding style* and *meaningful* comments.

All programming must be done using **C or C++ in Linux** where your code will be tested. Create a tar file that includes (1) source code files, (2) a Makefile to produce an executable, (3) a readme file that clearly describes how to run your code, and (4) a pdf file for instruction counts.

Submit only the tar file through the Blackboard. The name of the tar file should be yourfirstname\_yourlastname\_proj1.tar (Do not use special characters like #, @, or &, because they have caused Blackboard problems in the past.) Suppose that your assignment files are under the directory of /your\_userid/yourfirstname\_yourlastname\_proj1/ and you are under that directory right now. To create a tar file under /your\_userid directory, do the following in Linux command line:

>cd ..

>tar cvf yourfirstname\_yourlastname\_proj1.tar yourfirstname\_yourlastname\_proj1

To view the content of the created tar file, do the following in Linux command line:

>tar tvf yourfirstname\_yourlastname\_proj1.tar