**CSC430 PROJECT 1 DUE DATE: OCTOBER 22, 2014**

The classic question “Which sorting algorithm is faster?” does not have an easy answer. The speed of a sorting algorithm can depend on many factors, such as the type of elements that are sorted, the environment where the sorting is done and how the elements are distributed.

For example, sorting an array of 50 integers can be very different from sorting a large database; sorting a linked list can be different from sorting an array. Often, a particular algorithm is fast in some cases and slow in other cases.

This project will provide an opportunity to use and analyze various sorting algorithms.

**Specifications**

Write a program to compare the relative performance of different sorting algorithms on six datasets containing integers. Ultimately, the data should be sorted in ascending order.

The six input datasets are:

•Inverse100.txt (100 integers sorted in opposite order – descending)

•Random100.txt (100 integers randomly distributed)

•Almost100.txt (100 integers almost in order – ascending)

•Inverse1000.txt ( 1000 integers sorted in opposite order – descending)

•Random1000.txt (1000 integers randomly distributed)

•Almost1000.txt (1000 integers almost in order – ascending)

You should include the following sorting algorithms in your analysis. Feel free to place all of these in the same file. This is an analysis problem, not a structured software solution.

•Selection Sort

•Insertion Sort

•Heap Sort

•Merge Sort

•QuickSort

•An additional sorting algorithm of your choice

To measure the performance of the various sorting routines, count the number of comparisons and swaps required to achieve the desired ascending sorted order. You will be required to perform this analysis for all sorting routines for all six datasets. Be sure to measure compares as occurrences in each algorithm where array elements themselves are compared.

You should utilize global variables as counters. This is required for the recursive algorithms. Place all functions in one file, if necessary. For counting the comparisons, consider each place in each algorithm where an array element is compared to another. Swaps in the algorithms should utilize an included function swap(). Be sure to utilize two separate counters; one for total swaps and one for total comparisons.

**Deliverables**

•All final source code files

•Printed output using the six provided input files.

•A summary table that describes the results of the experiment along with a summary statement (paragraph or so) highlighting your conclusions from the results.