**PROGRAMMING AND WRITING ASSIGNMENT**

Write a program that performs the following tasks:

* Display a friendly greeting to the user
* Prompt the user for a file name
* Accept that file name
* Attempt to open the file for input (see note below)
* On failure, display an appropriate error message and exit
* Read the data in the file and populate an array
* Display the array
* Copy the original array
* Sort the copy using exchange sort
* Display the array
* Display the metrics (number of comparisons and number of exchanges)
* Sort the now-sorted array again using exchange sort
* Display the array
* Display the metrics (number of comparisons and number of exchanges)
* Reverse the array
* Display the now-reversed array
* Sort the reversed array using exchange sort
* Display the array
* Display the metrics (number of comparisons and number of exchanges)
* Repeat the above block for insertion sort and for selection sort

Write a summary paper in which you give results for an array of 10,000 random integers and state whether or not the metrics produced by your program are consistent with predictions made in class.

As always, provide source and executable on a USB drive and a printout of the source code on paper. Provide the required report on paper.

We’ll need a number of helper functions and a couple of global variables to make this happen…

void displayArray(int a [], int s)

// if there are 200 elements or fewer, display the entire array

// (ten columns, values right-justified in the columns)

// if there are more than 200 elements,

// display the first 100 elements and the last 100 elements

void swapElement(int [] a, int i, int j){

// exchange elements i and j in array a

// and increment a global “swapCount” variable

bool compareElement(int [] a, int i, int j){

// return true if a[i] <= a[j], false otherwise

// and increment a global “compareCount” variable

void copyArray(int [] source, int [] dest, int s){

// copy the first s elements from source into dest

void reverseArray(int [] a, int s){

// reverse the s elements in array a

The prototypes for the three simple sort algorithms are

void exchangeSort(int a [], int s);

void insertionSort(int a [], int s);

void insertionSort(int a [], int i, int j);

// sort from index i to index j inclusive

void selectionSort(int a [], int s);

Note that we’re thinking ahead with insertion sort, as we’re going to use it as a “clean-up” sort next week. The complete code associated with the first prototype is

void insertionSort(int a [], int s){

insertionSort(a, 0, s-1);

}

The second insertion-sort function does all of the work.

The format of the file is: a single integer stating the number of entries to be read, followed by *at least* that many integers. For example, the file

10 32 65 11 2 5 34 66 22 98 9

contains ten elements: 32, 65, 11, 2, 5, 34, 66, 22, 98, and 9.

The leading 10 is *not* one of the data elements.

That way you can open the file, read one integer, create your arrays, and then read the data into the arrays.

What happens if there aren’t enough integers in the file?