CS380 Algorithm Design & Analysis
Assignment 4: Quick Sort and performance measurement

Date Assigned: Monday, March 9, 2015
Date Due: Friday, March 20 @ 9:15am
Total Points: 50 pts

For this project you will implement QuickSort in the same manner as MergeSort and InsertionSort in project 2. You may add this code to your Sorting Project. Your implementation of QuickSort must allow for both the original partition algorithm (first one in the book) and randomization.

You will also need to write a new driver that will run each of these four (mergesort, insertion sort, quicksort using regular partition, quicksort using randomization) algorithms and time the execution of each sort routine and track how many times needSwap is called by each sort. Three new files, largeMountains.txt, largeMountainsASC.txt, and largeMountainsDESC.txt are provided. These each contains 1,000,000 randomly generated mountains.

Your driver must, for each of the three files:

Read the first 100 mountains, and sort in the ASC direction using each of the four algorithms. Time the call to sort() in each case (see below for how to tack times) and track the number of needSwap()s called. Sort in the DESC direction using each algorithm. For each sort algorithm, reload the data from the file before sorting.

Run Insertion sort and Quicksort with regular partition again for each of the sizes 1,000, 10,000, and 100,000. Run Merge sort and randomized Quicksort for each of the sizes 1,000, 10,000, 100,000, and 1,000,000.

Run regular Quicksort on only largeMountains.txt for 1,000,000 items. Don't do this for the other two files. It takes hours, and hours, and hours…., like 150 hours ☺

OUTPUT

```
Filename: largeMountains.txt
Size: 100
Insertion Sort:
      Swaps: XXXX
      Time:  XXXX
Merge Sort:
      Swaps: XXXX
      Time:  XXXX
Quick Sort (regular):
      Swaps: XXXX
      Time:  XXXX
Quick Sort (randomized):
      Swaps: XXXX
      Time:  XXXX
```

<repeat for each size and file>
<Total (54 runs): insertion sort (4 sizes, 3 files), merge sort (5 sizes, 3 files), regular quicksort (4 sizes, 3 files), randomized quicksort (5 sizes, 3 files), regular quicksort (1,000,000 items, largeMountains.txt)>

What to Submit
I will pull your project out of Subversion. You must provide me with a color, double sided hard copy of:

- QuickSort.h / QuickSort.cpp
- PerformanceDriver.cpp
- Any other NEW or CHANGED source files
- A printout of the text file containing your answers to the questions below.
- A print out of your output

Your code must build without any warnings. You must follow the C++ coding standards. Check for memory leaks!

Start early! **THIS MAY TAKE HOURS TO RUN, PLAN ACCORDINGLY.**

Get the files at:
http://zeus.cs.pacificu.edu/shereen/cs380sp15/Assignments/04largeMountains.zip

You may have many, many calls to needSwap. See link below for sizes of native data types in C++. Use the appropriate one. http://msdn.microsoft.com/en-us/library/s3f49ktz(v=vs.90).aspx

Hints on using timers in C++:

```
#include <ctime>
clock_t start, finish;
start = clock();
sort(); // Call your sorting algorithm
finish = clock();
cout << "Time for sort (seconds): " << ((double)(finish -
 start))/CLOCKS_PER_SEC;
```

OR

#include <windows.h> //and follow this link
http://stackoverflow.com/questions/1739259/how-to-use-queryperformancecounter/1739265#1739265
Be sure to do all of you timing via "Run without debugging" and without memory debugging.

For deeply recursive algorithms, you may need to increase the available stack space for your project.
Properties | Configuration Properties | Linker | System |

Stack Reserve Size 41943949 **(number of bytes)**

Stack Commit Size 41943949

Questions to answer in the text file containing the table:
Answer all of the following questions with complete, English sentences. Provide data where appropriate to backup your claims.

1. Does the growth in runtimes match what we expect from our previous analysis of each algorithm?
2. Do the number of calls to needSwap() grow in the way we would expect based on our runime analysis?
3. What type of impact does the temporary container have on the runtime of Merge Sort?
4. Given your results, how long do you expect 1million mountains from the largeMountains.txt file to take to sort using Insertion Sort?
5. Given your results, how long do you expect 1million mountains from the largeMountainsDESC.txt file to take to sort using regular QuickSort?
6. Given your results, how long do you expect 1million mountains from the largeMountainsASC.txt file to take to sort using regular QuickSort
7. Does it seem reasonable for the C-library function qsort() to actually use quicksort or should qsort() use some other algorithm to sort?