Assignmment 3: Quick Sort and performance measurement

Date Assigned:  Monday, February 20th, 2017
Date Due:  Monday, March 6th, 2017: 9:15am
Total Points:  50 pts

Goals for the project:
1. Implement Quicksort within a provided OO design framework
2. Analyze the time complexity of Insertion Sort, MergeSort, Quicksort (last element is pivot) and Quicksort Random (pivot is random element in range) on large file.

For this project, you will implement QuickSort in the same manner as MergeSort and InsertionSort in Assignment 2. You may add this code to your Sorting Project, and submit either to the provided GitHub repo or on Grace (rename folder cs380s17hw3-<PUNETID>).

In the same way that you did with Assignment #2, you will need subclass SortAlgorithm and implement Quicksort in QuickSort.h /QuickSort.cpp. You will also need to write a new driver named PerformanceDriver.cpp that will run each of these four (MergeSort, InsertionSort, QuickSort, QuickSort Random) algorithms and time the execution of each sort routine and count how many times needSwap is called by each sort.

Three new files, largeMountains.txt, largeMountainsASC.txt, and largeMountainsDESC.txt are provided, and each of these contain 1,000,000 randomly generated mountains.

Your driver will for each of the three files (with some exceptions noted below):

Read the first 100 mountains, and sort using each of the four algorithms in DESC order, keeping track of:

1) Time the call to sort() in each case (discussed more later) and
2) The number of needSwap()s called.

For the sake of simplicity, for each sort algorithm, reload the data from the file before sorting.

We will then read in 1,000, then 10,000, then 100,000, then 1,000,000 mountains and do a similar analysis. However, because of time constraints, we will not run all four algorithms on all three files for all these different sizes. For example, the time estimate for running Quicksort (using the first partition algorithm in your book) on 1 million items from largeMountainsASC.txt is several days on my system. So, don't run Quicksort (using the first partition algorithm in your book) on either largeMountainsASC.txt or largeMountainsDESC.txt with 1 million items. Quicksort on largeMountains.txt with 1 million items has a reasonable runtime, as does Quicksort Random (about a minute). Also, from what you know about the run-time of Insertion Sort, you should also be concerned about running this sorting algorithm for 1 million items, so do not run Insertion Sort on any of 1,000,000 mountain cases.
All told, you should have 55 runs of various sizes, which are delineated in the table below (MS-Merge-Sort, QS=QuickSort, QS-R=QuickSort with random pivot):

<table>
<thead>
<tr>
<th>File \ Size</th>
<th>100</th>
<th>1,000</th>
<th>10,000</th>
<th>100,000</th>
<th>1,000,000</th>
</tr>
</thead>
<tbody>
<tr>
<td>largeMountains.txt</td>
<td>ALL</td>
<td>ALL</td>
<td>ALL</td>
<td>ALL</td>
<td>MS, QS, QS-R</td>
</tr>
<tr>
<td>largeMountainsASC.txt</td>
<td>ALL</td>
<td>ALL</td>
<td>ALL</td>
<td>ALL</td>
<td>MS, QS-R</td>
</tr>
<tr>
<td>largeMountainsDESC.txt</td>
<td>ALL</td>
<td>ALL</td>
<td>ALL</td>
<td>ALL</td>
<td>MS, QS-R</td>
</tr>
<tr>
<td>Total Runs</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>7</td>
</tr>
</tbody>
</table>

and the output should look as follows:

**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: (random)
- Swaps: XXXX
- Time: XXXX

<repeat for each size and for each file as appropriate>

**What to submit:**

1. **An electronic copy of your Visual Studio solution on Grace (or commit to your repo on GitHub)**
   a. The program must be in the drop folder (or on GitHub) by 9:15am on the day that it is due. Anything submitted after that will be considered late. Please rename this folder `cs380s17hw3-<PUNETID>` if submitting to Grace, otherwise commit to the provided GitHub HW3 repo.

2. **A hard copy of your program**
   a. The hard copy of your code (`QuickSort.h /QuickSort.cpp/PerformanceDriver.cpp` in addition to any other files you may have changed) must be printed in color, double-sided, and stapled if necessary. Make sure line numbers are turned on, and that you have followed all the coding standards and there are no memory leaks!
3. A Word (or Google Doc) that has two tables (graphs would be nice too) summarizing the results for both the runtimes and number of swaps that you found. In addition to answers for the questions given below:

   a) Does the growth in runtimes match what we could expect from our analysis of each algorithm?
   b) Do the number of class to needSwap () grow in the way we would expect based on our runtime analysis or each algorithm?
   c) Given your results, how long would you expect 1 million mountains from largeMountains.txt to take to sort using Insertion Sort?
   d) Given your results, how long would you expect 1 million mountains from largeMountainsDESC.txt to take to sort using regular QuickSort?
   e) Given your results, how long would you expect 1 million mountains from largeMountainsASC.txt to take to sort using regular QuickSort?

Important Notes:

1) For deeply recursive algorithms, you may need to increase the available stack space for your project.

Properties | Configuration Properties | Linker | System |
Stack Reserve Size 50000000 (number of bytes)
Stack Commit Size 50000000
2) **For the larger files and for particular algorithms**, you will have many, many calls to needSwap which may necessitate changes in some of your data types. Visit the following website on native datatypes in C++ and choose the appropriate one:


3) There are at least two ways to implement counters in C++:

```cpp
#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

Be sure to do all of you timing via “Run without debugging” and without memory debugging.

4) Be careful with your Partition function. It relies on needSwap, but does it have access to this functionality?

**Bonus:**

Implement Quicksort additionally using the Median-of-3 method for choosing your pivot.