CCPS616 Lab 1 - Sorting and Problem Size

Preamble

In this lab you will write a C++ program that, among other things, implements and tests three sorting algorithms. You must implement these from the ground up without linking any external libraries. Your lab should be completed in a single cpp file called **lab1.cpp**, and should compile at the command line very simply as follows:

```
g++ -o lab1 lab1.cpp
```

Please do not submit labs that don't even compile. We're past that. I'm not going to fix your syntax errors. If you can't make any progress on the lab whatsoever, submit a simple program that prints a message saying as much. For this, I will award you a single mark.

Lab Description

We've done a lot of hand-waving about $O(n^2)$ sorting algorithms being faster than O(nlogn) sorting algorithms when the problem gets below a certain size. In this lab, we will put such claims to the test. You will implement **three** sorting algorithms:

First, implement standard, out-of-the-box Selection sort and Merge sort. Each should be in their own function. An unsorted array goes in, a sorted array comes out. Implementation details beyond this are up to you.

Next, implement a modified Merge sort (henceforth referred to as "*MergeSel*") that cuts to Selection sort when the sub-arrays get small enough. I.e., instead of recursively calling Merge sort all the way down to a subarray of a single element, call Selection sort instead when the subarrays are < **x** elements. The ideal value for **x** will be determined by you.

The astute among you might wonder why we would use Merge sort and Selection sort, instead of Quicksort and Insertion sort. Merge and Selection sorts have the same best/average/worst case complexity (nlogn, n^2 respectively). We use them so our results will be as predictable as possible and not influenced by the initial "sortedness" of our input array.

Testing

In your main function you will generate integer arrays containing **n** elements whose values are randomly distributed between **1** and **4n**. Make **n** suitably large so that we can see past the overhead involved and get a clear picture of the performance of each of our three sorting algorithms.

First, perform a sanity check on your Merge and Selection sort implementations. Test each using five different values of **n**. You might start with n = {5e2, 1e3, 5e3, 1e4, 5e4}. Beyond this, you might be waiting *hours* for selection sort to complete. Perform each of these five trials 10 times and compute the average (of course generating new random arrays each time).

Once you're satisfied that the results obtained from your Merge sort and Selection sort implementations make sense, perform the same tests on your MergeSel sort implementation. You'll do the same 10x5 trials, but this time we add another dimension for **x** – the size at which MergeSel kicks to Selection sort. The values for **x** are completely up to you. Your goal is to try and improve on Merge sort as much as possible. Have fun experimenting and follow the data!

Results

Your program should write out all test results to a plain text file. That is, running your program should produce a plain text file containing all experimental results described previously. Thus, your source code file must contain your sort implementations as well as all the test code that will produce the output file. Name your output file CCPS616_Lab1_AlexUfkes.txt, where my name is replaced with yours. Format your output file in a way that's easy for me to read. An example is below. All quantities in the table will of course be filled with real data produced by your program:

```
CCPS616 - Lab 1 – Alex Ufkes
Each result is the average of 10 trials

n =     5e2       1e3       5e3       1e4       5e4
Sel     XXXms     XXXms     XXXms     XXXms     XXXms
Merge   XXXms     XXXms     XXXms     XXXms     XXXms

MergeSel
x=a1    XXXms     XXXms     XXXms     XXXms     XXXms
x=a2    XXXms     XXXms     XXXms     XXXms     XXXms
x=a3    XXXms     XXXms     XXXms     XXXms     XXXms
x=a4    XXXms     XXXms     XXXms     XXXms     XXXms
x=a5    XXXms     XXXms     XXXms     XXXms     XXXms

Based on this, MergeSel performed best (we hope) when x = aX
```

Submission

Labs are to be submitted individually!

Submit your source file (**lab1.cpp**) and a sample output file (**CCPS616_Lab1_YourName.txt**) produced by your program to D2L. The sample output file, produced on your own personal computer, will give me an idea of how your computer's power compares to mine and may inform certain decisions related to problem tractability in future labs.