Solve the Sorting Mystery*

Due: Saturday October 9 @ 6am to Github Classroom Assignment Repo

Introduction

What a day... some digital trouble makers hacked into my Project 3 repo and wreaked havoc. I had a collection of sorting algorithms implemented in the repo. It was great stuff. But then these trouble-makers came in, changed the names of the functions from bubble sort, quicksort, etc to mystery01, myster02, etc. To create even more trouble, they only left a compiled version of the sort functions in a static-link library. So, I can't even look at the code.

Well, I'm really busy, but I still need to figure this out. I need you to "demystify" the sorting algorithms. I had implemented 5 sorting algorithms so far (in alphabetic order):

- optimized bubble sort
- insertion sort
- merge sort
- quicksort (with last element as pivot)
- selection sort

Right now, the names of the functions are mystery01, mystery02, etc. There's no relationship between the 01, 02, etc and the order of the sorting functions above.... or is there?

Each of the mystery-sort functions sorts an array of integers in ascending order. Each takes two arguments: first - an array of integers, and second - the number of elements in the array.

Disentangling this mess will largely be up to you to figure out. However, as we've discussed in class, the algorithms above fall into two complexity classes: $O(n^2)$ and $O(n \lg n)$. So, an obvious investigation strategy is to use timing. If you use timing, you should use the std::chrono library over the older, less precise <ctime>. However, you know something about how these functions rearrange elements as they are processing the data or how they respond to different initial orderings of the data. Can you think of a way to use this in your analysis? (Btw - I don't mean trying to interpret the disassembly of the library...)

Your Task

- Develop a strategy for determining the actual name of the sorts executed by mystery01, mystery02, etc.
- Implement your strategy in your main driver .cpp file, but don't put everything in main(), please!
- Use std::chrono for gathering any timing data. See the template repo's main.cpp for an example of how to do this.

The functions are contained in a static linked library. Static linked libraries are platform dependent. So one that works on Linux (WSL as well) will not work on Mac and vice versa. Of course, the library is compiled with no debugging symbols and with compiler optimization at the O3 level. All this to say... I don't think time invested in reverse engineering is time well spent... AND I'm not asking you to prove me wrong. Don't cheat!

Your Submission

You will submit a short write-up/lab report for this project **in the root folder of the github repo**. Your write up should include the following:

**This project is inspired by a similar one by Julie Zelenski and colleagues at Stanford University.*

- Who you are
- What you're doing (briefly describe the problem)
- How you planned to do it (strategy)
- What you figured out (mapping of mystery functions to which sorting algo it is)
- How you figured it out (briefly describe how you determined which algorithm was implemented in each function). Plots might be nice...

Your write-up can be a Word Doc or PDF.

Things you don't have to do:

- No CATCH tests for mystery sorter
- No new data structure implementations

Grading

	Points Possible
Properly identifying each mystery sort	12 pts each
Write up	40 points

**This project is inspired by a similar one by Julie Zelenski and colleagues at Stanford University.*