CS 271 (Spring 2025) Project 2 – Sort of?

Instructor: Prof. Michael Chavrimootoo
Due: Wednesday, February 12 2025, 11:59PM

1 Project Overview

In this project, you will implement four common sorting algorithms, namely insertion sort, merge sort, quicksort, and randomized quicksort, and you will compare them. By the end of this project, you will have a better understanding of these sorting algorithms and their runtimes.

1.1 Learning Goals

The goal of this project is to get you to be more familiar with the standard sorting algorithms, and explore their differences.

1.2 Use of Other Work

While you may consult any code you have personally written for this class, and any code in the course textbook, you are not allowed to use any other sources, which includes and is not limited to, your peers outside your group, other textbooks, the internet, and AI tools (such as Gemini and ChatGPT).

1.3 Group Work

You can complete this project individually or in pairs. I will leave the pairing if you wish to do it that way. However, if you work in pairs, you must detail your individual contributions in your report (see Section 2.4).

I expect everyone to contribute meaningfully to **both** the coding and non-coding parts of the project.

2 Submissions

2.1 Gradescope

Projects will be submitted via Gradescope, and you will be graded on passing certain tests. Each test contains multiple assertions, and you will need to carefully design your tests, which you will also need to provide. Every time you make a new submission, Gradescope will run the autograder to check that you pass all the tests, and will give you your results.

2.2 Compatibility

The tests are compiled using C++17. Different compilers/platforms may exhibit different behaviors, so I recommend testing your code on the Olin machines before submitting as those are closest to what Gradescope will test your code on. If your project fails to compile/run, you will receive a zero.

2.3 Code Submission

You must submit two files:

1. `sorting.tpp` - which will contain your implementations, compatible with the header file I'm providing you with.

2. `studentTests.cpp` - which will contain your own tests. *You will be graded on the quality of your own tests.* Here is how I will be compiling your tests, all your files are in the same directory
   ```
   g++ -std=c++17 studentTests.cpp -o studentTests
   ./studentTests
   ```

If you're missing either of these two files, the autograder will halt, and give you a zero.

Gradescope allows you the ability to upload the contents of a GitHub repository, so feel free to work on GitHub and upload from there. The autograder will ignore your other files.

Tip 1 *Learning to use* `git` *correctly for both collaborative and personal projects is a tremendous skill to have; you will certainly use it throughout your careers. Remember to make meaningful commit messages throughout the development of the project.*

Moreover, you must document your code properly. This includes:

- A full comment header at the top of each source file containing the file name, the developer names, the creation date, and also a brief description of the file's contents.
- Each method should have a full comment block including a description of what the method does, a "Parameters" section that lists explicit pre-conditions on the inputs to the function, and a "Return" section that explicitly describes the return value and/or any side effects.
- Make careful use of inline comments to explain various parts of the code that may not be obvious from the code syntax.

You will be graded on the quality of your code.

2.4 Report Submission

Along with your code, you must submit a report on Gradescope as a single PDF file. The report is an essential component of this project, and it is yet another way in which you communicate your technical contributions as a group. An individual who has never seen your project's description should be able to

understand what the project is about and how your solution works solely by reading the report and your comments.

Your report should be organized in the following (brief) sections:

- Project Contributors, and their individual contributions (if working in pairs).
- Group Challenges (if applicable) - Discuss any major problem your group encountered while implementing the project. This can also include challenges you encountered with your group.
- Individual Reflection - Describe your individual contributions to the project and briefly describe any challenges you (not your group) faced in this project.
- Known Issues - If there are any known issues with your code, mention those here. For each issue, try to best explain what may be causing the issue and what would be a potential solution; by thinking through the cause of the issue, you may find a way to fix it :)
- Additional Information - Any additional information you find relevant; keep it brief.
- Analysis - See the next section.

You may use your notes from class or the textbook to guide you through the implementations, and you cannot use any other resources.

While you cannot collaborate to *write* your reports, you are certainly free to discuss elements of the reports with your group, the TA, and me. However, you cannot produce any written material or recordings during those discussions.

Be careful: When writing your reports, you may not use text from the project description or other sources verbatim. You can only use your own words. If there is evidence that you received unauthorized help in your reports, you will receive a zero for that part of the assignment and a formal academic integrity violation report will be filed.

2.5 Analysis Requirements

While merge sort runs in $\Theta(n \log(n))$, it also consumes a lot of memory. Quicksort on the other hand works in-place, but does not provide a $\Theta(n \log(n))$ guarantee. However, as we discussed in class, there are strategies to improve quicksort to get better runtime guarantees.

In your analysis, you will compare the runtime of your implementations for `insertionSort`, `mergeSort`, `quickSort`, and `randomizedQuicksort`, and you will compare the performance of all those implementations with the performance of C++'s `sort` method which is part of the `algorithm` library (so you'll need to add `#include <algorithm>` to use it).

You are provided with four categories of `.in` files, each in a directory: `sortedData`, `reversedData`, `randomizedData`, and `patternedData`. For each category, provide a plot that is constructed as follows. For each algorithm, draw a curve, with the x values taken from `.in` file names; if a file is called `X.in`, that means it contains `X` elements. On the y-axis, plot the time taken for the given algorithm to sort the elements of that file.

If you're working in a group, you should use the same data and generate the same plot. However, your comments on the plots must be made individually. You may (individually) include additional data points if you wish. You **must include** your plots in your report.

Comment on the different performances and explain major differences. For example, how does randomized quicksort compare to regular quicksort, and why? Also comment on anything that jumps out to you, and try to explain why that happens.

Feel free to run additional analyses if you (individually or as a group) find them interesting!

Warning: if your plots don't seem to match the asymptotic runtimes of the algorithms, then you may be doing something wrong, and should adjust your code accordingly.