1 Goals

- Implement different versions of mergesort and quicksort in `C++`;
- More practice with pointers, arrays, etc.;
- More practice with unit tests and running and analyzing performance tests.

Note that you may use whatever environment you like for this class, but your programs must be able to compile and run using `g++` (or `clang++`), `cmake`, and `make`. For this assignment, you will also need `valgrind` to check for memory errors and leaks as well as `gnuplot` for generating performance graphs. You may also find it helpful to have `gdb` for helping to debug segmentation faults. The department provides a remote development server (`ada.gonzaga.edu`) running Ubuntu that can be accessed using an ssh remote connection from within VS Code on your own computer. The remote server contains all of the tools you need for assignments in this class. Depending on your computer's configuration, you may be able to install the tools you need locally to complete the homework assignments. It is important that you start assignments early in this class so that if you have questions you can ask them and get them answered with enough time before the homework deadline (to avoid late penalties).

2 Instructions

1. Use `git clone` to clone the classroom repository created for you and to obtain the starter code (either onto the remote development server if using `ada` or locally if you are using your own machine). Be sure to frequently add, commit, and push your updated files back to your GitHub repository (via `git add`, `git commit`, and `git push`).
2. Copy your LinkedSeq and ArraySeq member function implementations from HW-2 and HW-3, respectively, into the provided `linkseq.h` and `arrayseq.h` files. Be sure your code compiles.
3. Implement the sorting functions declared in the `arrayseq.h` and `linkseq.h` provided with the starter code. See the comments and lecture notes for details.
4. Make sure your implementations pass the basic tests provided in `hw4_test.cpp`. The tests cover sequences of size 0, 1, 2, and 3. You must add a set of tests to check your sort implementations over 4-element sequences. Note that you do not need to try all combinations of four elements. Instead provide cases that are unique, but that fully test your algorithms.
5. You must run `valgrind` on the `hw4_test` program to ensure it does not detect any memory issues in your implementations. You should run `valgrind` once you get the unit tests to pass. It may also help if your program has memory issues that are causing segmentation faults or other errors. Similarly, `gdb` can also be helpful in these cases. Note that you should be able to correctly insert, erase, etc., from the sequences after they are sorted (thus, you should include these cases in your tests as well).
6. Once your code is completed, all of the unit tests pass (including your new tests), and you do not have memory issues as reported by `valgrind`, run the provided performance tests in `hw4_perf`. The

performance tests will generate a data file with testing results, which you will then graph using `gnuplot`. You will need to add the corresponding graphs to your assignment write up (next step).

7. Create your assignment write up and add it as a PDF file to your assignment (GitHub) repository. You **must** save your writeup as a PDF file and name it `hw4-writeup.pdf`. (Note no capital letters, no spaces, etc.) Be sure to push all of your source code and your write up by the due date so it can be graded. (Note that you can check that everything is in your repo from the GitHub website and/or using the `git status` command.) See below for expectations concerning your assignment writeup. Once you have submitted your files to your GitHub repository and are ready to submit your assignment for grading, you must fill out the grading submission form. A link to the form will be provided in piazza (in the same post as the GitHub classroom link).

3 Additional Requirements

Use Only the Available Helper Functions. You are only allowed to use the functions declared in the class files in your implementation, and you may not modify the function signatures (parameter and return types) in any way.

Quick Sort with Randomized Pivot. See the comments and code provided in `linkedseq.h` and `arrayseq.h` for information on obtaining a random index value. Note that you'll be using a defined seed (provided for you), and must not change the seed value in your submission. Note you can change it if you'd like to experiment with different seed values prior to your submission, but your submitted code must have the original seed defined, etc.

Sorting with Linked Lists. See the lecture notes for details regarding applying the sorting algorithms over linked lists. You must follow this approach. In particular, you are not allowed to move values between nodes, and instead, must adjust pointers, etc. Again, details and high-level pseudocode is provided in the lecture notes.

Homework Writeup. Your homework write up must contain each of the graphs produced by the performance test together with an explanation as to why you think the performance tests came out the way they did based on what you know about the implementations. In addition, you must compare your results to those of HW-1. As in HW-3, you do not need to describe the unit tests, but should briefly describe the tests you added. Finally, briefly describe any challenges or issues you faced in completing the assignment.