1 Goals

- Implement a resizable array in `C++`;
- More practice using interfaces (abstract classes) and templates;
- 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. Finish the implementation of `arrayseq.h`. This will be similar to HW-2, but for a resizable array data structure as opposed to a linked list.
3. There are 19 total unit tests provided in `hw3_test.cpp`. For this assignment you do not need to write new unit tests. The tests are largely the same as those for HW-2, except for a resize test.
4. You must run `valgrind` on the `hw3_test` program to ensure it does not detect any memory leaks in your `arrayseq.h` implementation. 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.
5. Once your code is completed, the unit tests pass, and you do not have memory issues as reported by `valgrind`, run the provided performance tests in `hw3_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).
6. 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 `hw3-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

Resizable array growth. Your resizable array must start with a capacity of 0. Subsequent resizes should result in a capacity 1 array, followed by a capacity 2 array, followed by a capacity 4 array, and so on. Thus, the array growth pattern should be size 0, 1, 2, 4, 8, 16, 32, and so on. The array should never shrink.

Resize events. If an array is full with respect to the capacity, when the next valid insert is requested (via a call to `resize`), the `resize()` helper function must be called to increase the capacity of the array. Note that the array should not be resized unless an element is to be added to a valid index. The `resize()` helper must increase the size of the array (see above), copy the elements into the new array, and delete the old array. See the lecture notes for additional details.

Check for valid sequence indexes. As in HW-2, your functions must check for valid indexes and throw exceptions as appropriate.

Implement all of the "essential operations". You must implement all six of the C++ essential operations: default (empty) constructor, copy constructor, move constructor, copy assignment, copy move, and a destructor. Note that your destructor simply needs to call the `clear()` function and the body of your default constructor should be empty. Also note that for this (and future assignments), you are **not** allowed to call `insert` to perform a "deep copy", e.g., within your copy assignment operator. Calling `insert` will result in additional overhead (and potentially repeatedly call resize) leading to an inefficient solution.

Homework Writeup. For this homework assignment, like in HW-2, you will be creating three separate performance graphs, which will be generated for you via the provided `plot_script.gp` file. As in HW-2, your homework write up must contain each of the graphs 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-2. Note that HW-3 results support much larger sequence sizes. For this assignment, the graphs also compare your resizable array implementation to the C++ `vector` class (like in HW-2, which compared your linked list implementation to the C++ `list` class). For this assignment, you do not need to describe the unit tests, since these are the same as those in HW-2 (but over the array sequence). Finally, briefly describe any challenges or issues you faced in completing the assignment.