1 Goals

- Additional practice using dynamic memory;
- Practice with abstract classes and template (generic) types;
- Practice implementing copy and move constructors, copy and move assignment operators;
- More practice with unit testing;
- 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 on \texttt{ada} (which is running Ubuntu) using \texttt{g++}, \texttt{cmake}, \texttt{make}, \texttt{valgrind}, and \texttt{gdb}. \textit{It is highly recommended, however, that you use the Department-supplied virtual machine (VM) on your computer for this class (which is also based on Ubuntu).}

2 Instructions

1. Accept the GitHub classroom repo for HW-3, and clone it to your local environment. See piazza for the classroom link.

2. Implement the \texttt{ArraySeq} functions. See below for additional details.

3. Ensure your \texttt{ArraySeq} function implementations pass all provided unit tests. The unit tests are mostly identical to HW-2. As in HW-2, you will not be required to write additional unit tests for this assignment. You are free to add additional tests as needed, but you will not be graded on them. Note that the unit tests provided are not comprehensive, i.e., even if all unit tests for your implementation pass, there still could be issues (which may reveal themselves in the performance tests).

4. Ensure your implementation does not have memory issues as reported by valgrind. To run valgrind, use the command: \texttt{valgrind ./hw3_test}. This will run the tool over the unit tests, and will report any memory leaks or other memory issues in your implementation.

5. Run the performance tests via the \texttt{hw3_perf} executable. As in HW-3, you will need to redirect the output of the tests to a file \texttt{output.dat}. Once generated, use the provided gnuplot script to create graphs from your results. For this assignment, three graphs will be created. One graph is for testing the erase function, one for the insert function, and one for updating values in the linked sequence. Each operation (erase, insert, update) includes a comparison between your \texttt{ArraySeq} implementation and the standard C++ resizable array implementation \texttt{std::vector} over three general cases (beginning, middle, and end of sequence) and increasingly larger sequence sizes.
6. Create an assignment write up that includes the three performance graphs, and an explanation of the results reported in the graphs (similar to HW-1 and HW-2). Finally, your write up should also contain a brief paragraph on any implementation issues and/or challenges you ran into and how you addressed them (if applicable).

3 Additional Details and Requirements

**Stream Insertion Operator.** You must implement the string insertion operator for ArraySeq. See the corresponding unit test for how to properly format the output of ArraySeq objects.

**Constructors.** Since we are using C++ member variable initialization, the default constructor is empty. However, you will need to implement both a copy and move constructor. Your copy constructor should call the overloaded copy assignment operator. Similarly, your move constructor should call your overloaded move assignment operator (similar to the copy case but by also calling std::move()).

**Assignment Operators.** You must implement both a copy and move assignment operator. Your copy assignment operator must make a complete copy of the incoming resizable array. This should be done without calling any ArraySeq functions except for make_empty(). The general algorithm for doing this was discussed in class.

**Destructor.** You must implement a destructor. Your destructor can reuse your make_empty() function.

**Bounds checking.** Your index operator, insert, and remove functions must check that valid indexes are provided. If a valid index is not provided, your functions must throw a std::out_of_range exception. The corresponding message for the out of range exception should be the qualified name of the function, e.g., "ArraySeq<T>::operator[]", "ArraySeq<T>::insert()", etc.

**Resizing an array.** You must maintain both an element count (via the count member variable) and the array capacity (via the capacity member variable). When inserting into an ArraySeq that is “full” (i.e., has already reached the capacity of the array), you must “resize” the array. The algorithm for this was discussed in class. All of your code for handling a resize event must be placed in the resize() function. Detecting if a resize is necessary should be done in your insert function, after checking for an invalid/valid index.

**The sort function.** Like HW-2, we are including a non-working sort() function in HW-3, which we will implement in a later assignment. For now, you only need to leave the sort() function empty.