1 Goals

- Additional practice using pointers and linked lists;
- Practice implementing merge and quick sort;
- 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 ada (which is running Ubuntu) using g++, cmake, make, valgrind, and gdb. 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-4, and clone it to your local environment. See piazza for the classroom link.

2. Copy your linkedseq.h and arrayseq.h files into your repository. You will be implementing new functions in each of these files for merge sort and quick sort (see below for details).

3. Ensure your function implementations pass all provided unit tests. 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. Implement four new unit tests for your sorting functions. These tests will specifically be for 4-element sequences. Each test should be for a different combination, i.e., merge sort for array sequence, merge sort for linked list, quick sort for array sequence, quick sort for linked sequence. See hw4_test.cpp for details.

5. Implement four new unit tests to ensure that your array sequence and linked sequence work correctly after they have been sorted. Each test should be for a different combination, i.e., merge sort for array sequence, merge sort for linked list, quick sort for array sequence, quick sort for linked sequence. Thus, you will need to implement 8 new unit tests in total. See hw4_test.cpp for details.

6. Ensure your implementation does not have memory issues as reported by valgrind. To run valgrind, use the command: valgrind ./hw4_test. This will run the tool over the unit tests, and will report any memory leaks or other memory issues in your implementation.
7. Run the performance tests via the `hw4_perf` executable. As in HW-4, you will need to redirect the output of the tests to a file `output.dat`. Once generated, use the provided gnuplot script to create a performance graph from your results. Note that not all possible cases are tested. In particular, the worst cases for quick sort are not performed since these take considerably longer than the shuffled case.

8. Create an assignment write up that includes the performance graph, and an explanation of the results reported in the graphs, a comparison of the graph results to those from HW-1 (for the “inefficient” sorting algorithms), and a brief description of your new test cases (namely, why you chose the tests you did). 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

**New ArraySeq functions.** You must add the following two **public** functions to `ArraySeq`:

```cpp
// implements merge sort over current sequence
void merge_sort();

// implements quick sort over current sequence
void quick_sort();
```

The implementation of these functions will simply call the following **private** helper functions with the appropriate arguments.

```cpp
// helper functions for merge and quick sort
void merge_sort(int start, int end);
void quick_sort(int start, int end);
```

You must write these functions to implement the merge sort and quick sort algorithms as described in class. You are **not** allowed to define any other helper functions for implementing merge sort and quick sort. Finally, implement the `Sequence sort()` function such that it calls the public `quick_sort()` function.

**New LinkedSeq functions.** You must add the following two **public** functions to `LinkedSeq`:

```cpp
// implements merge sort over current sequence
void merge_sort();

// implements quick sort over current sequence
void quick_sort();
```

The implementation of these functions will call the following **private** helper functions (in addition to additional steps, such as setting the head and tail pointers, as described in class).
// helper functions for merge and quick sort
Node* merge_sort(Node* left, int len);
Node* quick_sort(Node* start, int len);

Note that when implementing these helper functions, you must state the signatures as follows:

```cpp
template<typename T>
typename LinkedSeq<T>::Node* LinkedSeq<T>::merge_sort(Node* left, int len) {
    ...
}

template<typename T>
typename LinkedSeq<T>::Node* LinkedSeq<T>::quick_sort(Node* start, int len) {
    ...
}
```

You must write these functions to implement the merge sort and quick sort algorithms as described in class. In particular, you are not allowed to move values between linked list nodes, and instead, you must use the node “splicing” methods. This effectively means you are adjusting pointers to sort the lists (as opposed to moving values). Again, see the lecture notes for details. As above, you are not allowed to define any other helper functions for implementing merge sort and quick sort. Finally, implement the `Sequence sort()` function such that it calls the public `merge_sort()` function.