1 Goals

- Practice with our key-value pair collections and our version of a Map API;
- Practice with implementing and using binary search;
- More practice with asymptotic analysis and 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-5, 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 from HW-4. Be sure these files are included in your final submission.

3. Implement `ArrayMap`, `LinkedMap`, and `BinSearchMap`. See below for additional details.

4. 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).

5. Ensure your implementation does not have memory issues as reported by valgrind. To run valgrind, use the command: `valgrind ./hw5.test`. This will run the tool over the unit tests, and will report any memory leaks or other memory issues in your implementation.

6. Run the performance tests via the `hw5.perf` executable. As in HW-5, you will need to redirect the output of the tests to a file `output.dat`. Once generated, use the provided gnuplot script to create performance graphs from your results.

7. Create an assignment write up that includes the performance graphs and an explanation of the results reported in the graphs. In addition, fill out the following table regarding the worst-case asymptotic complexity, using $\mathcal{O}$ notation, for each of the functions listed below for each Map implementation.
### 3 Additional Details and Requirements

Note that a number of details for HW-5 were discussed in class and are provided in the lecture notes. Additional information is provided below regarding suggestions and requirements.

**Start with ArrayMap and LinkedMap.** As described in class, both the ArrayMap and LinkedMap implementations are straightforward. It is recommended to stub out all function implementations in each of the three implementations (so you can compile the project), and then start on ArrayMap. Once you have ArrayMap completed, LinkedMap should be straightforward as well. Each function implementation for LinkedMap should be identical to those in ArrayMap, including the `sorted_keys()` function.

**Implementing the bin_search() function in BinSearchMap.** You must implement the `bin_search()` helper function in BinSearchMap iteratively (i.e., without recursion), and you may not use any other helper functions for performing binary search (or in general in your BinSearchMap implementation). Specifications for the function are provided in `binsearchmap.h` and were discussed in class. Note that to call `bin_search()`, you must provide both a key and an index value. This index value will be set by `bin_search()` in most cases (thus, the index is an “output function parameter” since index is passed by non-const reference). Note that `bin_search()` should never return an invalid index into the underlying ArraySeq, unless the collection is empty (in which case, `bin_search()` should return false and not modify the given index). If the key is not present in the collection, `bin_search()` should also return false, but will set the index (roughly to where the key-value pair should be inserted).

**Using bin_search() in your BinSearchMap functions.** You must use `bin_search()` in your implementations for `operator[]`, `insert()`, `erase()`, `contains()`, and `find_keys()`. Additional details for how these should work with `bin_search()` were provided in class. Note that for `find_keys()`, you must use `bin_search()` to find the starting index to search from (and search through keys in order until you pass `k2`, after which you should stop searching). Finally, for `sorted_keys()`, you should not call any sort functions (since the underlying sequence is already sorted by key).

---

#### MAP IMPLEMENTATION

<table>
<thead>
<tr>
<th>Operation</th>
<th>ArrayMap</th>
<th>LinkedMap</th>
<th>BinSearchMap</th>
</tr>
</thead>
<tbody>
<tr>
<td>insert</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>erase</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>contains</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>find_keys</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>sorted_keys</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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).
Start and ask questions early. Please start early and ask questions early. It is important that you carefully *think* through your `bin_search()` function implementation and how it relates to the various functions that must call it (as stated above). Starting early and asking questions will help you to finish the assignment on time. One example of this is understanding what guarantees are made by `bin_search()` regarding the index returned when the key is not present in the underlying sequence. Understanding how `bin_search()` functions in this case will help you to create correct implementations of your `insert()` and `find_keys()` function.