1 Goals

- More practice with the map (key-value pairs) interface/ADT;
- More practice with binary search;
- More practice with 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 `sequence.h`, `arrayseq.h`, and `arraymap.h` files (implementations) from HW-5 into your HW-6 repository. These must be present in your repository for me to grade your submission.
3. Implement the functions declared in `binsearch.h`. See the comments and lecture notes for details.
4. Make sure your implementations pass the basic tests provided in `hw6_test.cpp`. You do not need to write any additional unit tests for this assignment (however, you are encouraged to write additional tests as you see fit). Note the tests provided are in no way comprehensive.
5. You must run `valgrind` on the `hw6_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.
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 `hw6_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). Note that running `hw6_perf` to completion should take around 15 seconds to complete. If it takes longer than this for you, there is likely something wrong with your implementation.

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 `hw6-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

A number of details for HW-6 were discussed in class and are provided in the lecture notes. Additional information is provided below regarding suggestions and requirements.

Implementing the `bin_search()` helper function. 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()`, `find_keys()`, `next_key()`, and `prev_key()`. 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).

Homework writeup. Your homework write up must contain each of the graphs produced by the performance tests 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. Finally, briefly describe any challenges or issues you faced in completing the assignment.