1 Goals

- Additional practice with key-value pair collections and the Map API;
- Practice implementing balanced binary search trees;
- Additional practice with dynamic memory;
- Additional practice with recursion;
- 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-8, and clone it to your local environment. See piazza for the classroom link.
2. Copy your `map.h`, `sequence.h`, `arrayseq.h`, `binsearchmap.h`, `hashmap.h`, and `bstmap.h` files into your repository from HW-7. Be sure these files are included in your final submission.
3. Implement `AVLMap` (in `avlmap.h`). See below and lecture notes 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 ./hw8_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 `hw8_perf` executable. As in previous homework, 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. Focus on explaining the differences between the implementations and give reasonable rationale explaining why the differences exist. In addition, fill out the following table regarding the *worst-case* asymptotic complexity, using *Big-O* notation, for each of the functions listed below for each `Map` implementation.

Operation	*Map Implementation*					
	ArrayMap	LinkedMap	BinSearchMap	HashMap	BSTMap	AVLMap
insert						
erase						
contains						
find_keys						
sorted_keys						

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

Be sure to review the lecture notes regarding the AVL Map implementation for HW-8. For this homework, you must implement the functions as specified in the lecture notes. Helper function declarations are provided for recurisive solutions. You may **not** modify the signatures of helper functions. You also must **not** add any additional helper functions.

When defining helper functions that return `Node` pointers, you must use the following syntax (in this case, for the `erase()` helper).

```
template<typename K, typename V>
typename BSTMap<K,V>::Node* AVLMap<K,V>::erase(const K& key, Node* st_root)
{
  ...
}
```

The implementation of the AVL Map functions should be identical to those of BST Map except for insert, erase, and height. In addition, you will need to implement the new helper functions (for rotations and rebalancing).

A print() helper function is provided to help with debugging. In particular, print outputs each node in an AVL tree including the node's key value and height. The left (lft) and right (rgt) children are also shown (much like a directory structure).

Finally, for this assignment, you should expect the performance tests to take a bit longer (upwards of 2 minutes). For me, running the performance tests took approximately 1 minute and 45 seconds.