1 Reading:

Read and complete all (participation and challenge) activities for chapter 10 (Week #7 reading).

2 Goals

- More practice using vectors in C++;
- Practice using binary search in different situations.

Like with previous homework assignments, you will need to use both CMake and Google Test. Both are installed on ada and the department’s linux virtual machine. Note that CMake also requires the make command, which is also installed on ada that the virtual machines. You will need to install each of these on your own if you are using a different environment.

3 Instructions

1. Your primary task is to implement another vector based version of the Collection abstract class, but this time using binary search for insert, remove, and both find operations. (Note that you must use binary search for the operations as described in class and the lecture notes.) You will have almost identical files as for HW4, except instead of linked_list_collection.h you will have binsearch_collection.h, instead of hw4_tests.cpp you will have hw5_tests.cpp, and instead of hw4_perf.cpp you will have hw5_perf.cpp. You will also need to make small changes to your CMakeLists.txt file for HW5.

2. As with prior assignments, carefully consider the additional test cases you will need to write for hw5_test.cpp.

3. Like for HW4, you must run your implementation through the performance test code. Similar to HW4, you must:

   (a). Run your program at least three times for each of the five test files and record the results. (Note that you must run each of the test files the same number of times.)

   (b). Using the run results, create an overall average for each of the three runs, for each operation and test file.

   (b). Create a table of the results. Your table should be formatted similarly to the following (yet to be filled in) table.
4. Similar to HW4, create graphs showing the performance of your implementation compared with your HW3 vector-based implementation and your HW4 linked list implementation. You must create separate graphs for each operation, i.e., one graph comparing each add implementation, one graph comparing each remove implementation, one for find, one for range, and one for sort).

5. Hand in a hard-copy printout of your source code, with a cover sheet. Be sure to carefully read over and follow all guidelines outlined in the cover sheet.

6. Submit your source code via GitHub classroom. Your source code must be submitted on the due date. See piazza for additional details and instructions.

Additional Information and Hints for HW5

- In the starter code, we define the following helper function.

  ```cpp
  bool binsearch(const K& key, int& index) const;
  ```

  You must implement and use this helper for performing binary search in your collection functions. The function returns true if the given key value is in the vector and false if it isn’t. The function also returns the index of the location of the matching key-value pair if the key is found, and the index of where the key-value pair should be located if the key was not found.

- To insert a value into a vector, use the insert function. For instance, the following statement inserts the key-value pair p into kv_list before the i-th index in the list (shifting all values from i to the right in the vector).

  ```cpp
  kv_list.insert(kv_list.begin() + i, p);
  ```

- Again, your collection functions add, remove, find value, and find range must all appropriately use the binsearch helper function to perform their tasks.

- Unlike in HW3 and HW4, you don’t need to call the sort helper function provided by C++ since you are maintaining a sorted key-value vector in this assignment. This means that your sort function should be blazingly fast!