Exercises for 3 Aug 2020

1. Write a hash table that stores `std::string` objects using the *chaining* approach to collision resolution. Start with a very simple hash function.

   see the code at HashTable.h, HashTable.cpp

2. Make up some strings to store in your hash table and plot the number of elements in each chained list against the bucket number. Try out a couple of different hash functions and note the effect they have on the distribution of list lengths.

   see the code at main.cpp

3. [optional] Generalize your hash table, making it a class template that can store any copyable type as long as you pass in an appropriate hash function.

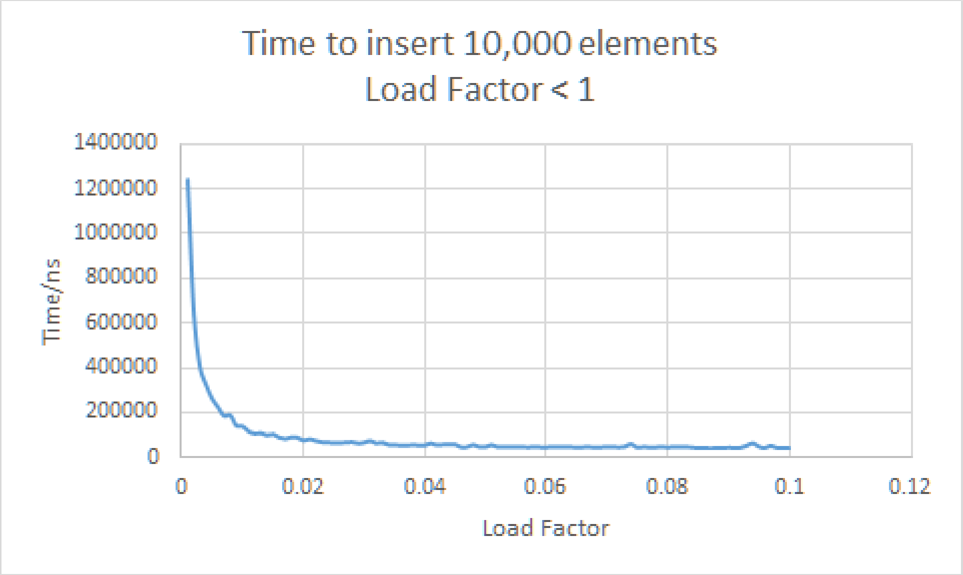
Exercises for 5 Aug 2020

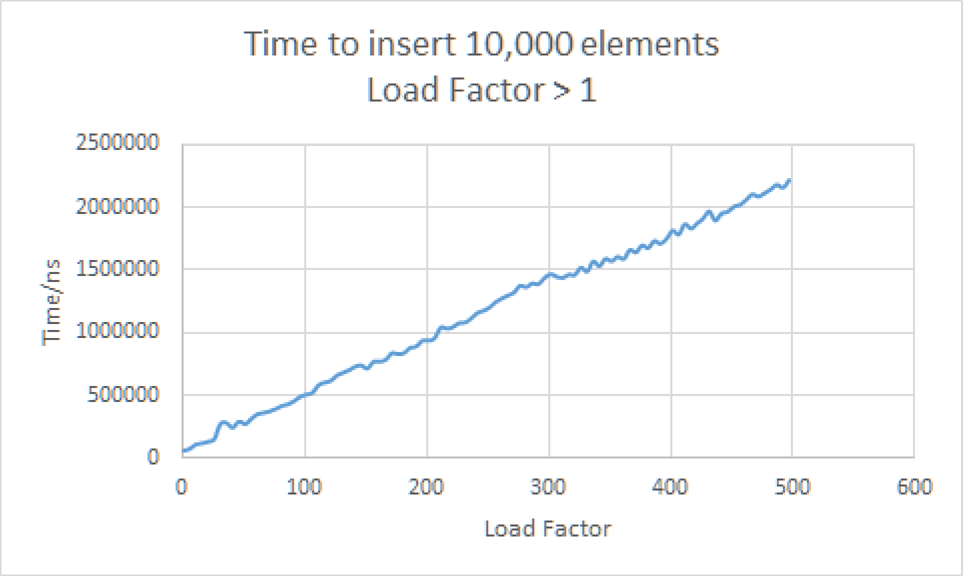
1. Write a program that inserts 10,000 integers (0 through 9,999) into a `std::unordered_set<int>`. Let this program set the maximum load factor for the set (using `unordered_set::max_load_factor()`) according to a value passed at the command line.

   see the code at main.cpp;

2. Plot the average time that your program takes to insert 10,000 elements against load factor. If you're running on Windows and the Windows timing trick doesn't work on your platform, you may time the complete execution time for your program rather than isolating just the insertion with `std::chrono`. Why did you choose the load factor values that you did?

   As we can see from the following figures: when load factor is smaller than 1, the time decreases logarithmically with load factor's increasing; when load factor is bigger than 1, the time increases linearly with load factor's increasing. Thus, 1 is the best choice of the load factor.





3. [optional] Use gperf to generate a hash table that can hash any of your classmates' names in constant time.
4. [optional] Implement a template for hash table class that stores keys-value pairs like `std::unordered_map<T>`.