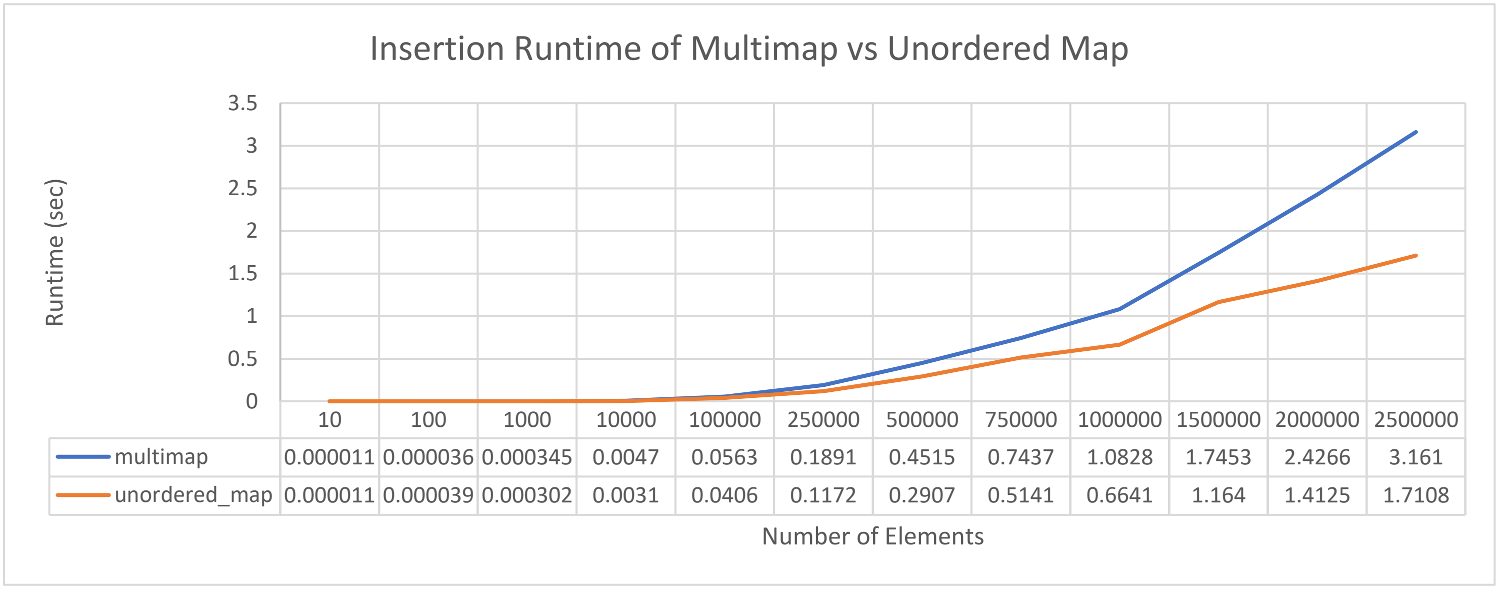
Alex Holt – A57959647

Hypothesis: For insertions in a balanced binary tree and hash table, I predict that the hash table's time complexity will be constant while the binary tree's will be logarithmic.

Methods: To solve this problem, I wrote the code at the following link: https://github.com/Holtster2000/cse431_hw4/blob/master/q3.cpp

I created a program in c++ to measure the runtime complexity of inserting into a multimap, which is based on a balanced binary search tree, and an unordered_map, which is based on a hash table. I am using the method described on Mimir with ctime to measure the runtime duration.

Results: From the output of the program, I created a graph showing the runtime for inserting a given number of random elements into the data structure.



	10	100	1000	10000	100000	250000	500000	750000	1000000	1500000	2000000	2500000
multimap	0.000011	0.000036	0.000345	0.0047	0.0563	0.1891	0.4515	0.7437	1.0828	1.7453	2.4266	3.161
unordered_map	0.000011	0.000039	0.000302	0.0031	0.0406	0.1172	0.2907	0.5141	0.6641	1.164	1.4125	1.7108

As shown, the time complexity of the multimap is growing noticeably faster than the unordered map. Since we were anticipating the multimap to be logarithmic and the unordered map to be constant, our hypothesis was correct. There is most likely some error in the way things were timed that is leading to the increasing complexity, but the results were overall what we expected.

Discussion: I thought that the multimap might grow way faster than it did but was surprised to see it stay quick all the way into the millions of insertions. Also, the hash table was a bit slower than I initially thought but again this might be due to some sort of timing error in my code.

Conclusions: Under the conditions tested, a binary tree backed dictionary will grow slightly less than twice as fast as a hash table back dictionary in terms of time complexity.