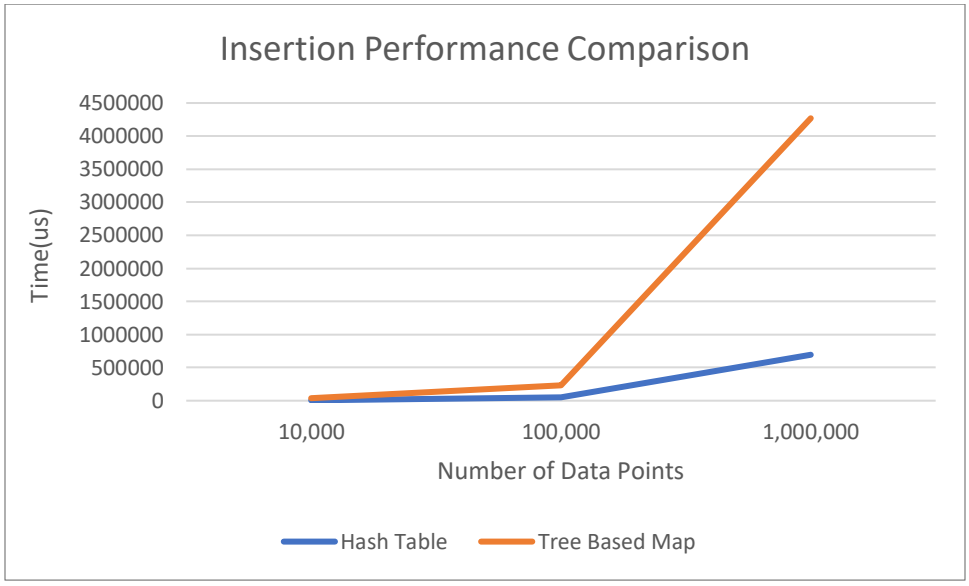
Part 3: Comparing the Performance of Ordered and Unordered Map

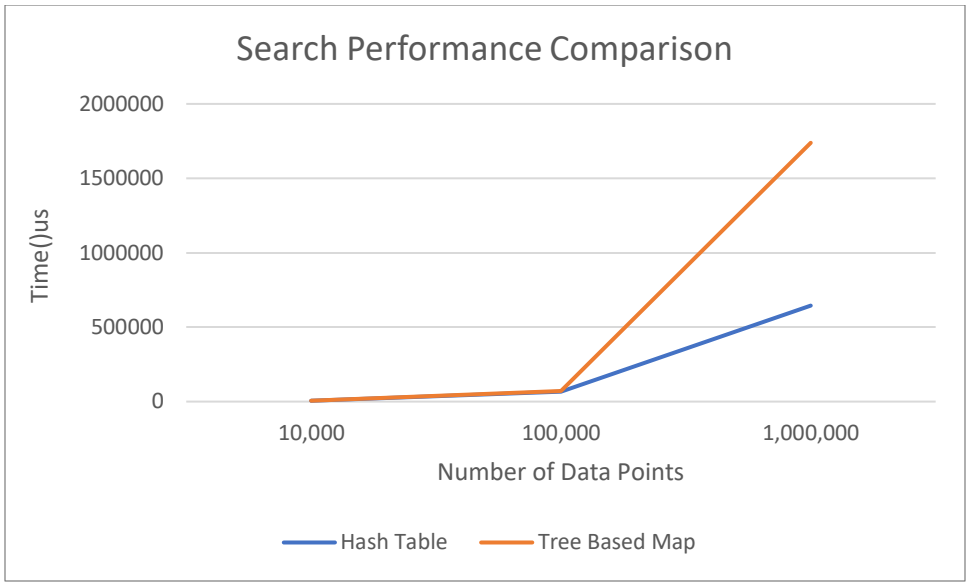
Part A. 2 Table Records of Execution Time

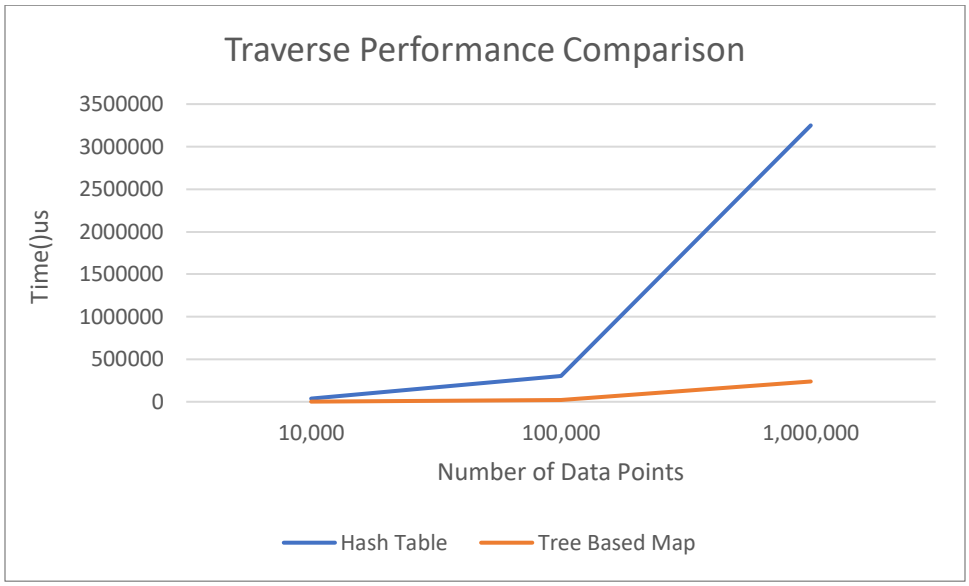
Hash Table Based Unordered Map Performance (microsecond us)			
Number of Insertions	10,000	100,000	1,000,000
Insert	5174	49285	694048
Search	5760	66632	644864
Traverse	37523	304205	3249548
Size of Map	17687	176929	1770294

Tree Based Ordered Map Performance (microsecond us)			
Number of Insertions	10,000	100,000	1,000,000
Insert	37647	236393	4268959
Search	5968	70251	1738596
Traverse	1912	20280	238643
Size of Map	9391	93981	940536

Part B. 3 Graphs that compare operation's performance







Part C. Worst Time Complexity of Ordered Map and Unordered Map

Tree Based Ordered Map:

Insert ~ O(log n), where n is the number of nodes in the tree. The findHeight function is constant, and the tree is a balanced tree.

Search ~ O(log n), where n is the number of nodes in the tree. Because the tree is automatically balanced and recursively calls the search method.

Traverse ~ O(n), where n is the number of nodes in the tree. Each node in the tree is visited once when traversing the tree-based map.

Hash Table Based Unordered Map

Insert ~ O(n), where n is the size of the unordered map. When current load factor is larger than max load factor, we need to rehash and move all node to another hash table.
Search ~ O(n), where n is the size of the unordered map. When the key does not exist, we construct a default key value pair and insert that to the table. If current load factor is larger than max load factor, we need to rehash and move all node to another hash table.
Traverse ~ O(n), where n is the size of the unordered map. We need to iterate through every node in unordered map.

Part D. Commentary on Performance of These Two Data Structures

When I was observing execution time of two data structures implemented by me, I figured out that insertion and search works more efficiently on hash table based unordered map, while traverse works more efficiently on tree based ordered map. For insert, if the current load factor does not exceed max load factor, the insertion of my ordered map is O(1), given that I insert new node to the start of singly linked list. If rehash happens, it will be O(n), where n is size of the unordered map. The AVL tree based ordered map has an average O(log n) for insertion. When n gets large, log n is also a bigger number. So, I think that's why unordered map is more efficient when dealing with insertions. For search, it takes O(1) for unordered map to figure out which bucket the key value exists. Assuming key exists and no new key value pair will be constructed, it takes O(k), where k is number of nodes on the specific singly linked list to locate the node. The AVL tree based ordered map has an average of O(log n) for search. And this is the reason my unordered map behaves better than my ordered map. For traverse, even though unordered map has approximately two times of elements than ordered map, unordered map still behave significantly slower than ordered map. I think that's because I passed a reference of the unordered map object to the iterator class, which slows the process. My ordered map works better than STL ordered map. My unordered map inserts and searches faster than STL unordered map but traverses slower than STL unordered map. I guess STL unordered map have more advanced ways of manipulating iterator class.

Part E. Self-Reflection

I learned how to read other people's code and build upon that. When I run my peer's code on stepik it gives me segfault and I figured out the root of AVL tree is not initialized to null pointer. I also learned how to apply some knowledge I learned in COP3503, like friend class,

which turned to be very helpful for this project. When I was starting the project, I thought about using the linked list I wrote in COP3503. But as I scratch the thing on my paper, I figured out there is no need for using doubly linked list in this case and the functions are not very helpful. So, I wrote myself a simple node class that handles everything and largely increases the performance. Another thing I learned in this project is that we need to think from another direction sometimes to solve a problem. In the rehash function, I considering several approaches to assigning the new buckets to the unordered map. But they are very time consuming. Finally I just swap the two vector of buckets in one line, which is simple and efficient.