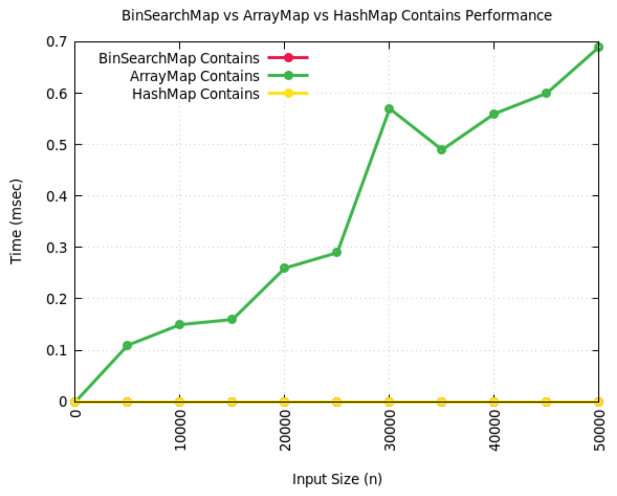
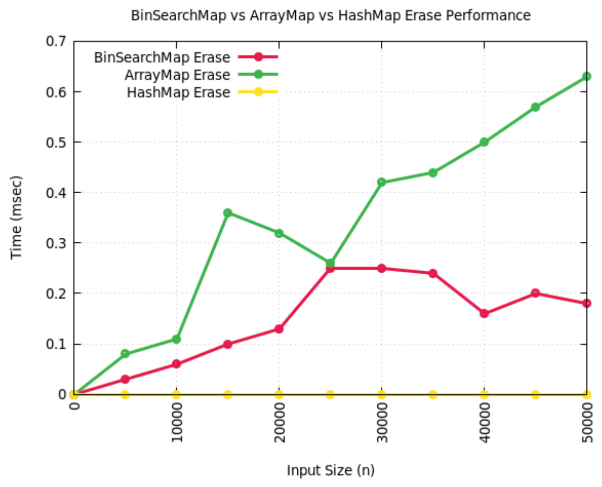
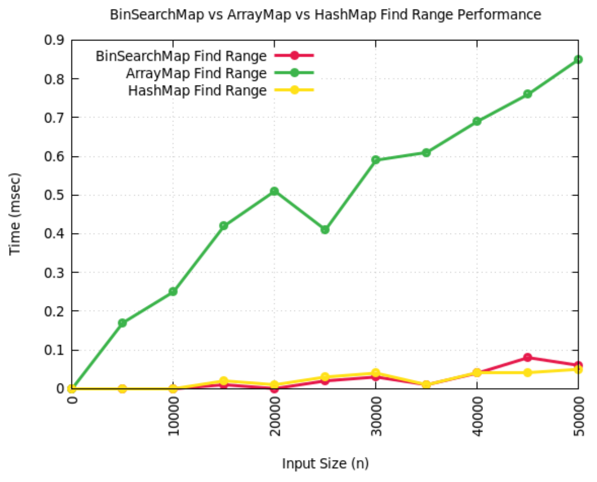
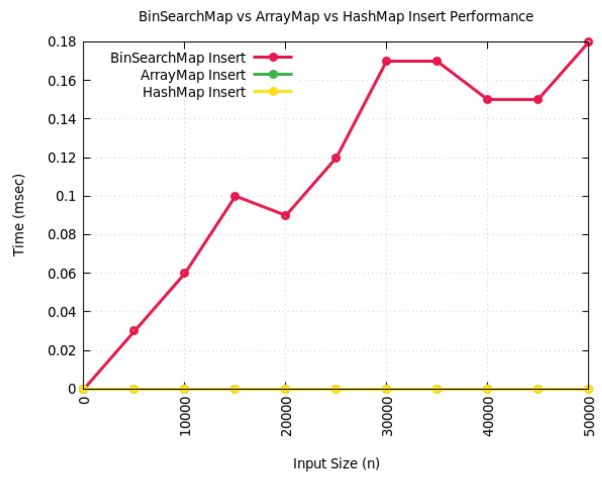
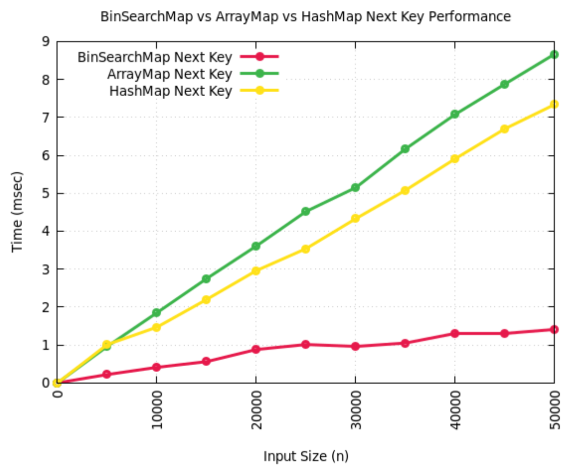
Samuel Sovi
CPSC 223
HW-7

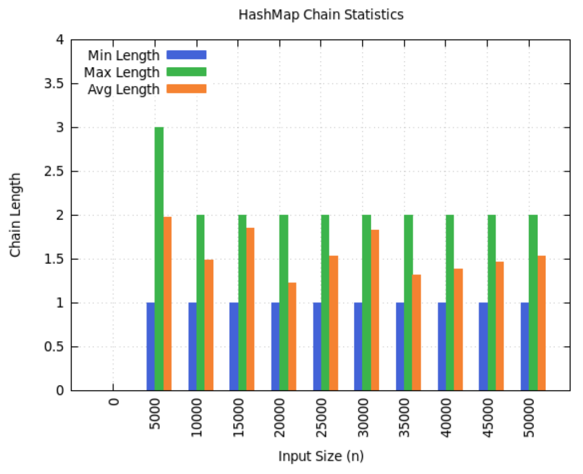












Shown above are my six graphs that compare my HashMap with my BinarySearchMap and ArrayMap. One of the main purposes of a Hash Table, or in this case, a HashMap is to improve the performance of updating elements, the contains function, insert and erase. This is because with a good hashing function, a hash table or hash map can become almost O(n). In my first graph, we can see that both BinarySearchMaps and HashMaps perform much better than my ArrayMap. This is because my HashMap can access very quickly due to its almost O(1) access/searching times while my BinarySearchMap accesses/searches for elements quickly with its O(logn) access time. Meanwhile my array map has to iterate through to find if an element is contained within. A similar result can be seen with my Erase function in all 3. However, my erase function shows a slower time there due to the fact that it must shift elements when erasing. my HashMap does not need to shift elements, however, due to its nature. Insert has a very fast time as well for all but my BinarySearchMap which trades off the insert for other functions. The insert for BinarySearchMap must insert into the sorted order and takes time to do so. My BinarySearchMap and HashMap outperformed the ArrayMap in the find range performance due to their natures of having sorted elements. However, my BinarySearchMap outperformed my HashMap in the find next performance graph due to it having all elements in sorted order whereas my HashMap has elements hashed into buckets but multiple unsorted elements can be in the same bucket. For the last graph, we can see that the hashing function proved to be slightly effective for this specific group of elements since my minimum chain size was 1 and my average chain size ranged from 1.5 to 2 as well. In a perfect world with a perfect hashing function for a perfect set of elements, we would have a max chain size of 1 as well, but my result of 2 to 3 was definitely a decent showing.

In this assignment I had many difficulties with memory allocation that I didn't seem to have as much of on previous assignments. This could be due to the time constraints I had to deal with due to conflicting class tests and commitments or even newer material. Personally, I struggled the most with managing memory leaks and passing two of the tests in the hw7_test.cpp file. I believe that my issue there was with my resize and rehash function which I was only able to complete after many attempts. I always ended up running into the same thought process even when commenting out my whole function and re-writing it from scratch. I am still struggling to find where I went wrong with the rehashing and resizing, but I will continue to try to find out how I can improve. I believe this assignment really showed me that I have lots of room to improve and probably ego-checked me (in a good way). I hope to further improve and learn more on my journey as a computer science student and beyond.

Having been given the opportunity to submit this assignment late now, I can say that I successfully figured out my mistakes and fixed all my memory leaks. I only needed help with figuring out errors that valgrind caught that had to do with my misunderstanding of whether or not copy constructors and move constructors called the default constructor "by default". Now I understand this better and value giving more time to myself to finish these assignments.