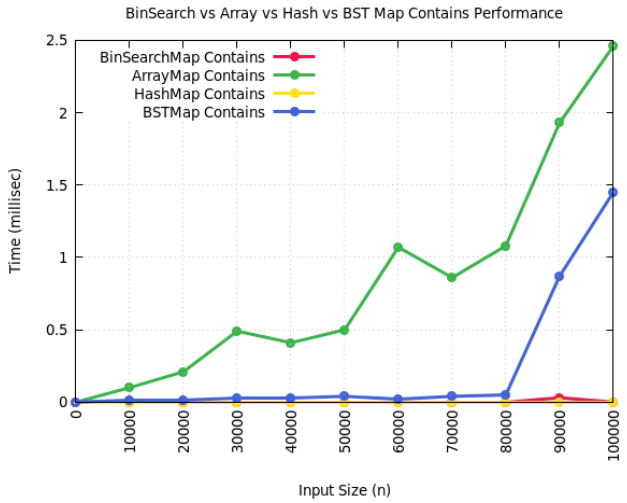
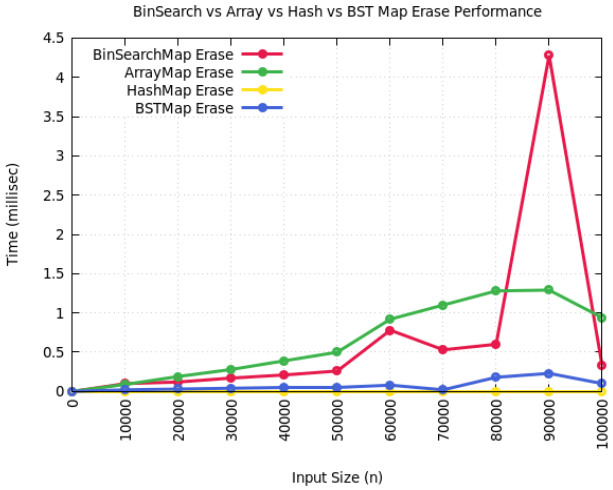
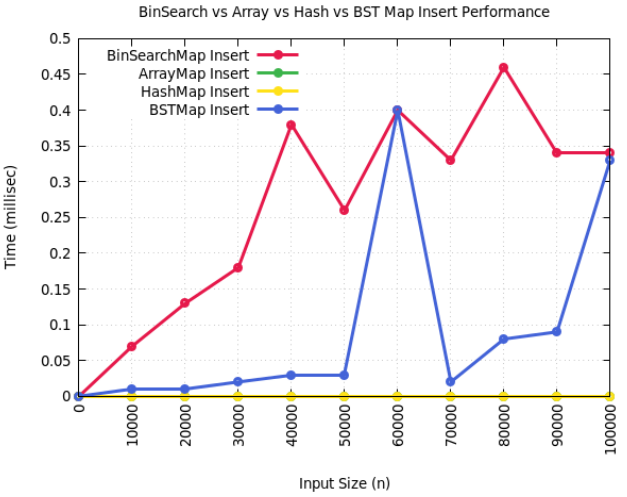
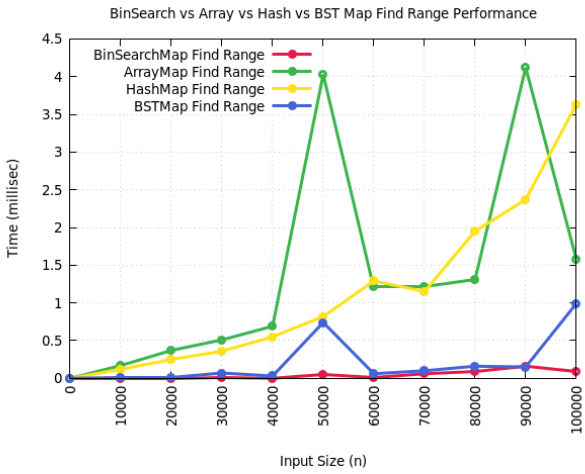
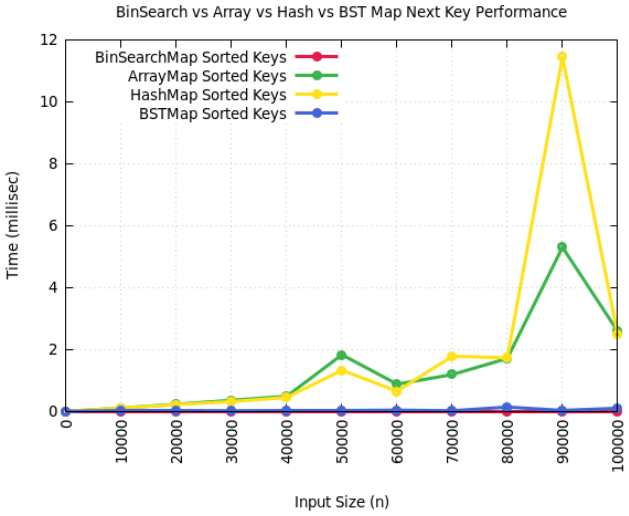
Samuel Sovi
CPSC 223
HW-8

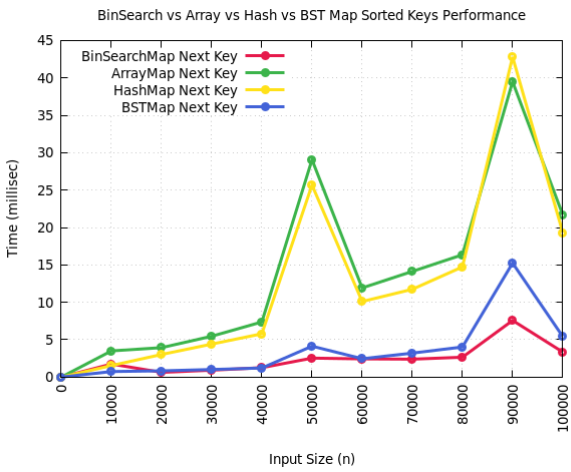


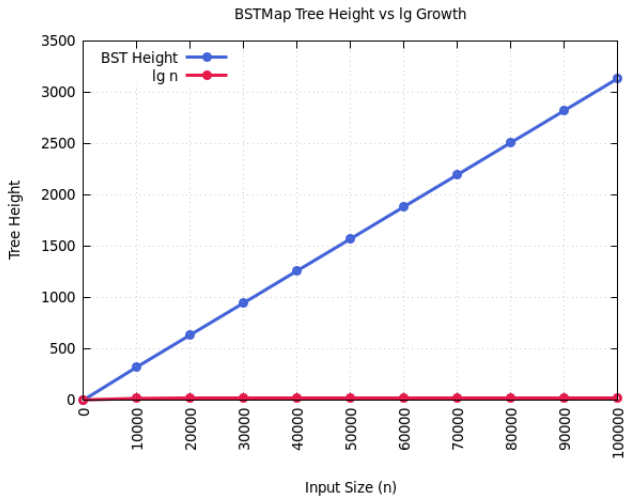












Shown above are my graphs comparing my BinarySearchMap, my Arraymap, my HashMap and my BSTMap as well as one graph that compares my BST Tree Height vs the lg Growth. For the Contains() function, we can see that ArrayMap has a bigger time while all the rest are faster because ArrayMap searching is O(n) due to its unsorted nature while all the others have their elements either sorted or hashed, which increases the efficiency of finding whether or not a given element exists within the Map. Similar logic can be applied to my erase graph, because erase involves finding the element and erasing it. In this graph, however, BinarySearchMap is slightly slower than BSTMap and HashMap because it must shift more elements after erasing. BSTMap also must shift the inorder successor. For insert, ArrayMap is O(1) because it just needs to insert at the end. HashMap's insert is O(n) but it will be closer to O(1) on average. BSTMap is O(n) but will ideally be closer to O(logn). BinarySearchMap is O(n) because it must shift elements after inserting in sorted order. HashMap and ArrayMap must search the whole Map to perform find_keys() whereas both BinarySearchMaps can perform BinarySearch to get to the desired location. The same logic can be applied to next_key() where HashMap and ArrayMap must search the whole Map to perform next_key() whereas both BinarySearchMaps can perform BinarySearch to get to the desired location. For sorted_keys, both BinarySearchMap and BSTMap are already sorted so they take less time to generate a sorted arrayseq of keys whereas HashMap and ArrayMap must get all keys and then call sort() on the generated arrayseq. Our final graph shows a relation between our BST height and the ideal height of a binary search tree to show the amount of impact that a Balanced Tree has on operation time due to its height. Since this tree is not balanced, the tree height can grow to extremely large numbers (up to n, the number of elements).

During this project I struggled with a number of things. While I thought I wouldn't make this mistake anymore after the last assignment, I still had a number of Nodes that I forgot to break connections with when creating my erase() function. I also realized that my HashMap was not performing sorted_keys() properly because I was only taking elements with hash codes between the two keys' hash codes. I also didn't realize that I was already getting all the elements in sorted order by getting all elements in inorder fashion. As a result, I could remove my call to sort().