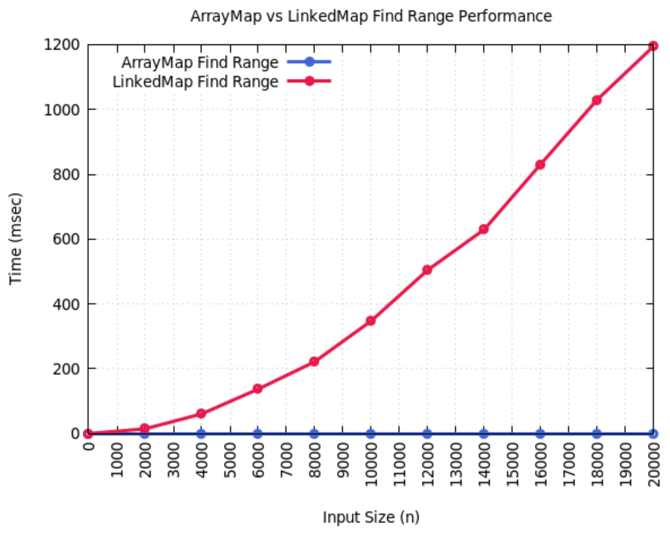
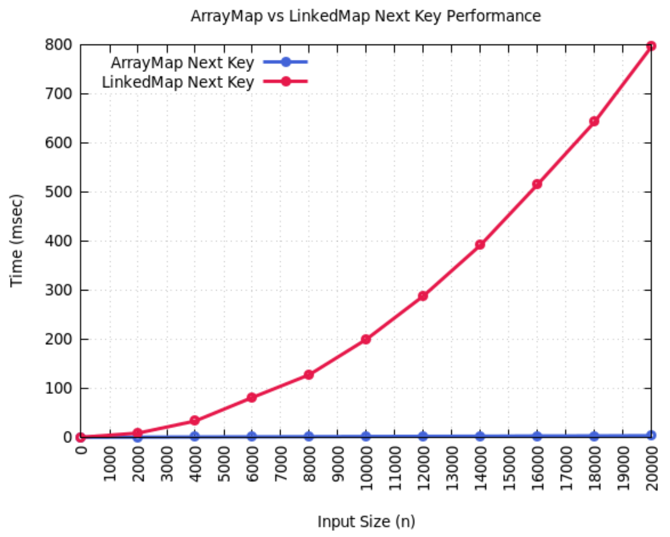
HW5 Writeup (Spring 2022)

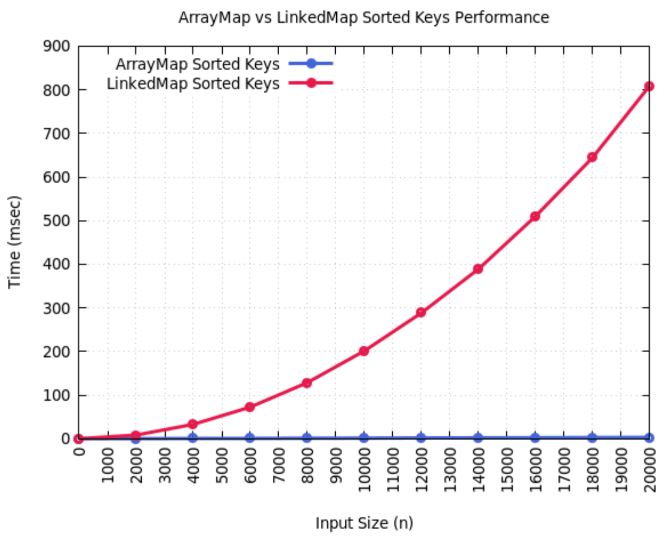
Plots:



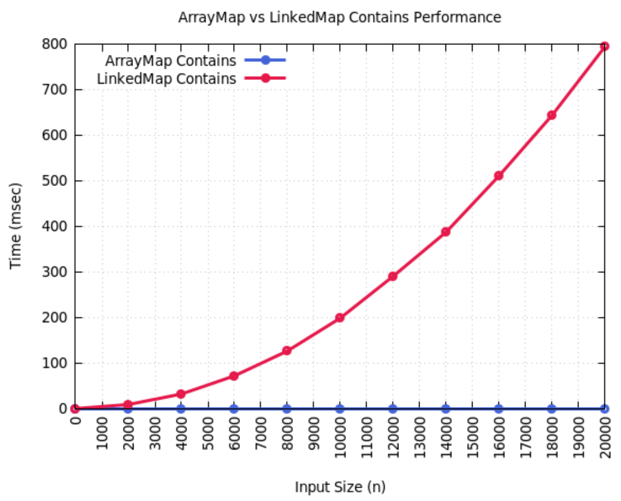
The plot above visualizes the performance of my implementation of the find_keys() function for a map based upon both a resizable array and a linked sequence. Due to the necessary call to the insert() method of a sequence, the linked sequence call is more complex as iterating over a linked list takes more time than indexing in a resizable array.



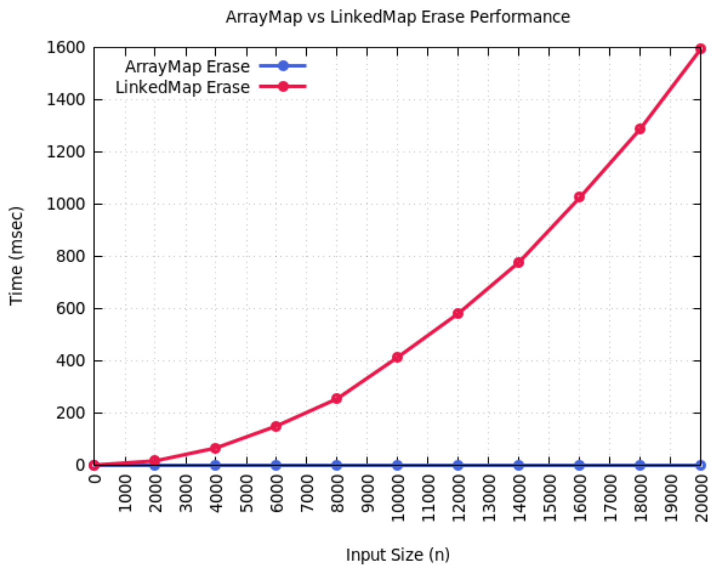
This next plot is associated with the next_key() function for both implementations of a Map in this assignment. Not unlike what occurs in the find_keys() method, searching for the next key in a sequence also requires indexing. This function uses the access and update operators to index to the desired space in a sequence, but my implementation of it also sorts the sequence beforehand. Thus, the linked sequence, again, has a greater time complexity due to iterating, or indexing, over nodes that are linked together.



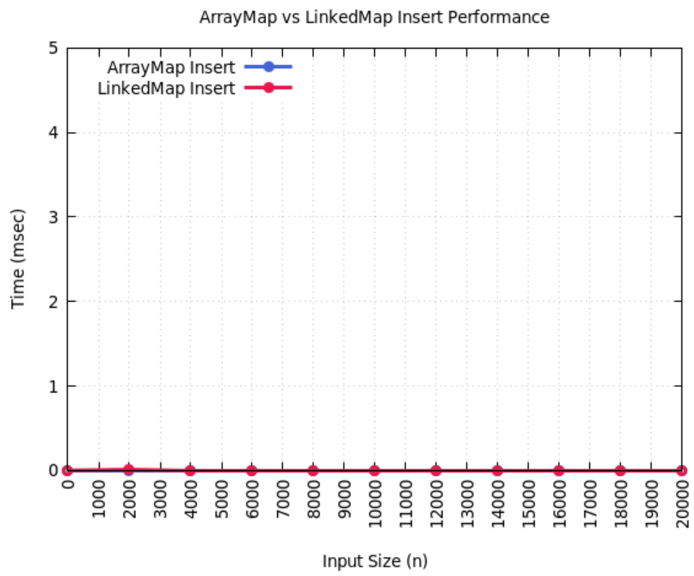
My implementation of the sorted_keys() function holds time complexities as depicted above. This function utilizes an array sequence to store the keys taken from a linkedmap of key-value pairs. Thus, the insert function calls decrease time complexity overall. However, this improvement is still outweighed by the need to access the values of said linked map by iterating over a linked list. As this is more complex than indexing an array, it is no surprise that the LinkedMap function call has a greater time complexity.



As redundant as it may be, the contains function, though identical for arraymap and linkedmap, require different types of indexing using the access and update operators defined within respective arrayseq and linkedseq classes. Simply put, a linked sequence indexes the linked list it holds by iterating through each node until the proper index is reached, set as the condition to check for in the loop. Conversely, an arrayseq call for most of these functions only requires an array-based indexing to return the desired value.



The erase performance graph above is the last of the graphs that show a distinct difference in time complexity between an ArrayMap and a LinkedMap. It is as much an issue of indexing in the erase() methods found in LinkedSeq and ArraySeq as it is for the other functions presently accounted for. This method takes even longer to perform on a LinkedMap due to the need to link together the nodes left in the list after one has been removed. Also, there is a call to contains() which, as already stated, adds on another iteration through said sequence.



In contrast, the insert() performance graphs appear to be relatively uneventful. This is due to the amortized O(1), or constant, time complexity that signifies the insert method of an ArraySeq type. However, the restrictions placed on this function included a forced addition to the end as well as the assumption that no repetitive keys would be placed within the sequence. As such goals were accomplished, one can only assume that the impressive performance of the insert method can be tied to restrictions that ensured a competitive implementation of the LinkedSeq.

Consensus:

Though the insert method still requires indexing using the access and update operators, and iteration in the case of the linked list, the need to resize the array balances out the time complexity between ArrayMap and LinkedMap in the case of their identical insert methods.

Challenges:

I faced a multitude of challenges during this assignment, despite its advertised ease of completion. However, most of these issues turned out to be remnants that were missed in previous assignments. I spent more time fixing bugs in the LinkedSeq and ArraySeq implementations I used in HWs 2, 3, & 4 than I did debugging my newer Map implementations. For example, previously only integers were used in testing my implementations, so the sorting methods utilized in HW4 had been set up to expect integer values rather than any values (abstract ‘T' types of indexed returns). Also among these challenges was a slight error in understanding when to delete items in a sequence versus an entire sequence of items. This brought on a particularly challenging debugging process.