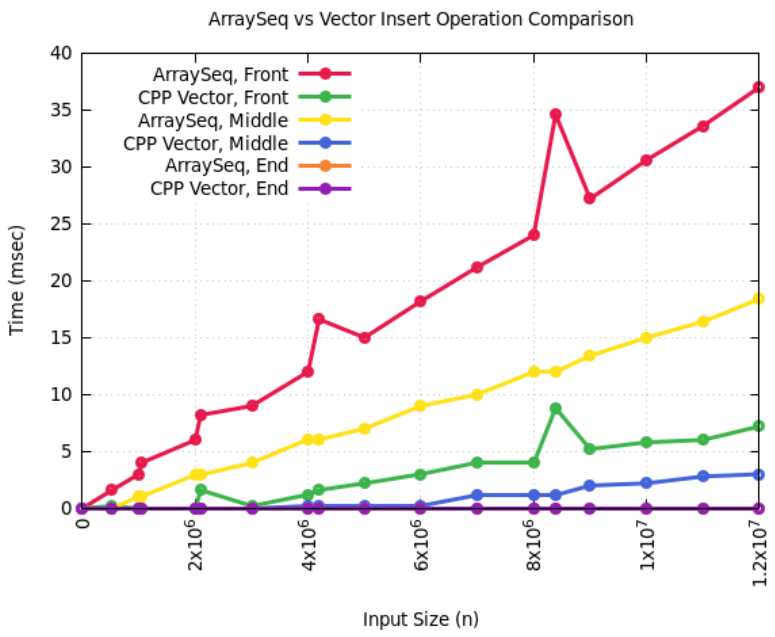
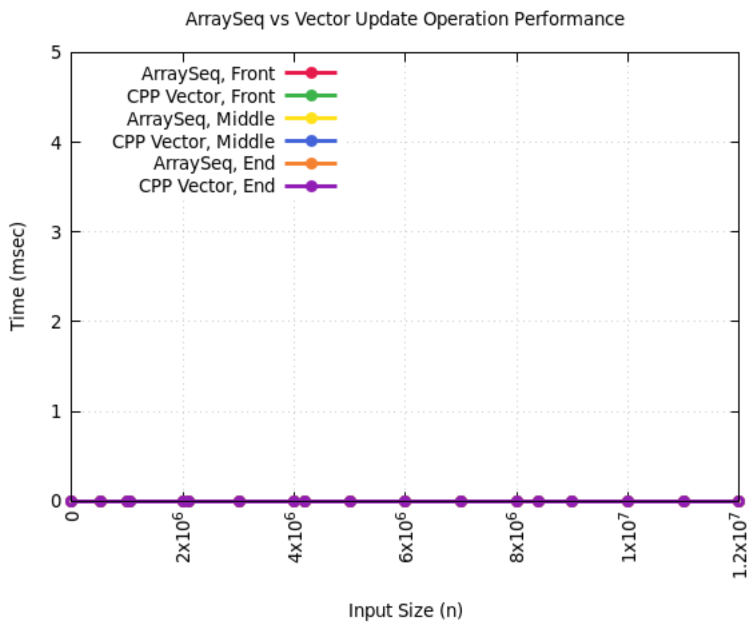
Homework 3 Writeup (Spring 2022)

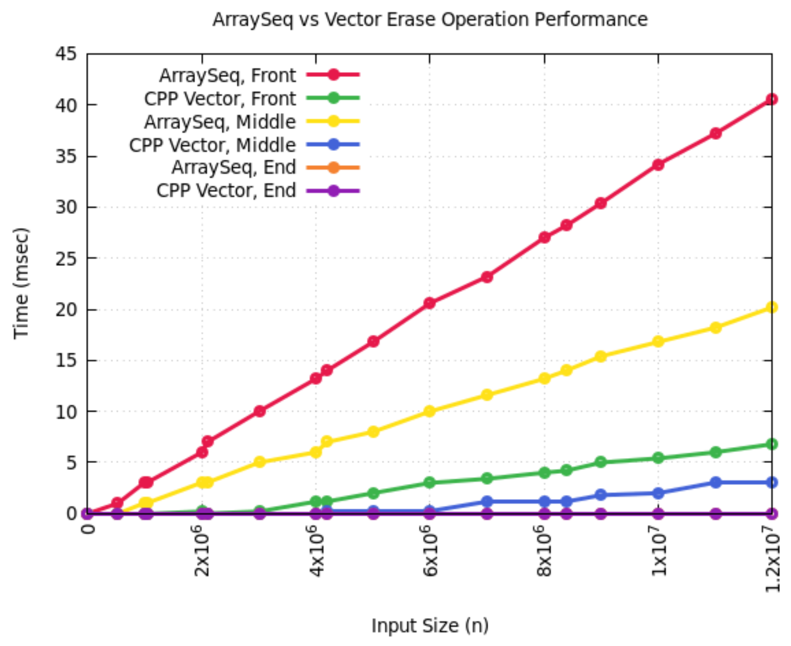
Plots



From the above graph the insert operation in my *ArraySeq* data structure has periodic bumps in what is seemingly linear performance. Due to the need to resize when the size of the sequence meets the capacity of the array, it is understandable that said bumps will occur. The resize function, of course, adds more steps for the insert function to perform. Thus, time complexity is increased when the array needs to be resized. Inserting at the front takes the longest because every existing element in the array must shift right in the list before the new element can be inserted.



The performance graph for the update operation in both a resizable array and a vector is, in short, uneventful. The advantage of using an array is the ability to access the value at an index by directly calling an access operation with brackets. So, one can comprehend that the time complexity would be linear and extremely close to, if not actually, zero. The same goes for how a vector is implemented and used in C++ so the graph of a vector's performance will also look similar.



Erase does not have a need to resize the array to change its total capacity. Therefore, the plots in this graph look more linear than those present in the graph of the insert operation. The extended amount of time taken for erasing at the front and middle of my ArraySeq data structure can be explained by the implementation. More specifically, the shifting of every element still in the array after one is removed requires iteration and with it, more time.

Compared To LinkedSeq

Compared to the resulting performance of LinkedSeq and considering the increased sequence sizes applied to the ArraySeq, the LinkedSeq appears to be quicker than my ArraySeq. The inserting into the LinkedSeq is generally quicker except for inserting into the end of an ArraySeq, an instance that is equally instantaneous. For updates, ArraySeq is exclusively better as, again, this is an advantage using arrays. Finally, the erase operation in the LinkedList also performs better. Therefore, the overall consensus is that a LinkedSeq performs better when operating on the ends of a list, while the ArraySeq shines when operating on the middle section of a list.

Challenges

The primary difficulty I had with this assignment was determining how to implement the overloaded copy assignment operator without using the insert function. Eventually, it came to me that, like what is done in my resize function, I could create a temporary array to act as a middleman. That is, transfer the subject array to the temporary array and then transfer the temporary array into the destination array. With this problem came errors that would escape the test harness and allow valgrind to return no memory leaks and no failed tests even if errors were present.