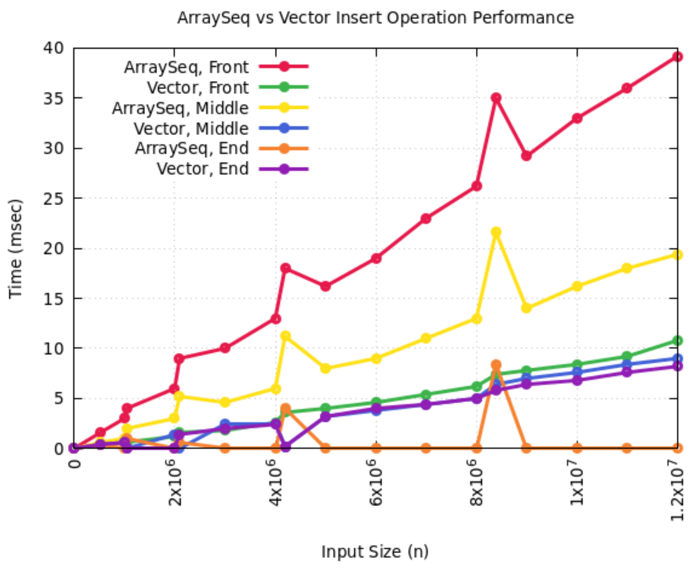
NAME: Ben Puryear

FILE: HW-3_WriteUp.pdf

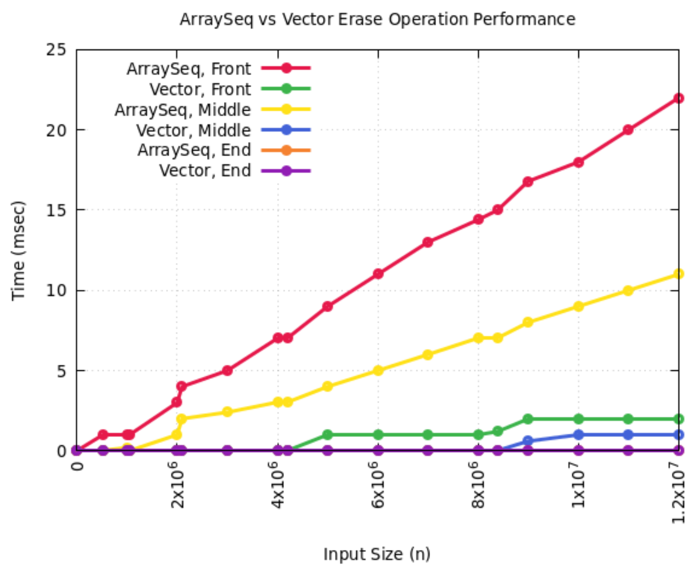
DATE: Fall 2021

DESC: This pdf goes over the basics accomplished throughout all the HW-3 related files.

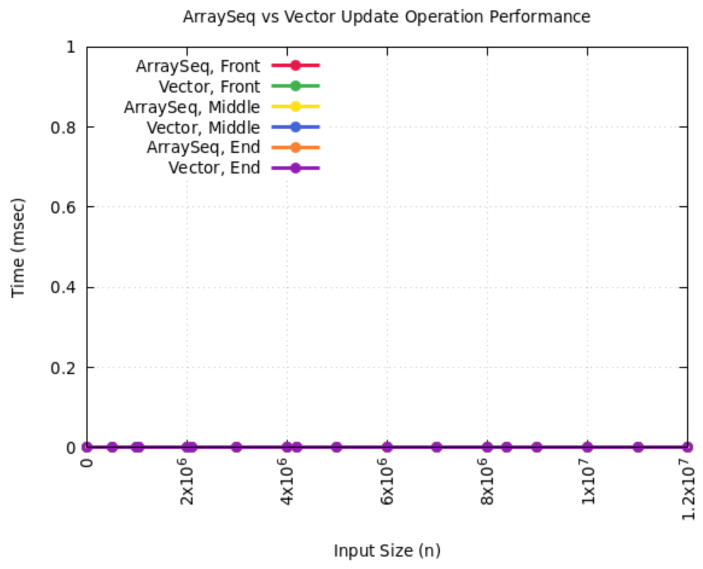
Graphs / Explanations



The first graph is the insert() graph. The first thing that sticks out are the periodic spikes in the ArraySeq cases. This is due to the need to use resize() inside insert() because the new count after insert will be larger than the current capacity, so it has to copy the current array into a new array that is twice as large. Another thing that sticks out is how well the performance is for inserting at the end of an ArraySeq. Unless it is having to call resize(), it just uses an assignment operator. This means it doesn't have to move any contents or create and copy a new array (unless resize() is called). It also makes sense how ArraySeq front insertion is the worst performing case. This is because ArraySeq insertion works by starting off moving all the contents after the insert index (inclusive) of array up an index, then array[index] = elem. When the index is the front, that means it needs to move every single index up one. That is why the graph is moving up linearly (Ignoring the resize() calls).



This next graph visualizes the performance of erase(). One observation is that because erase() uses on the final index just changes the count of the array, the resulting performance is a constant. Similarly to insertion, ArraySeq’s implementation of erase() on the first index has the worst performance. This is due to having to move al of the indexes down one. Another observation is how all of the ArraySeq cases are linear, and therefore predictable.



This final graph visualizes the update operation performance. This graph shows how all instances of ArraySeq and Vector are constant. This is because ArraySeq’s update functions just use assignment operators, with no insertion or node changes.

Issues I faced:

The only issue I encountered was figuring out how to implement the move assignment operator.