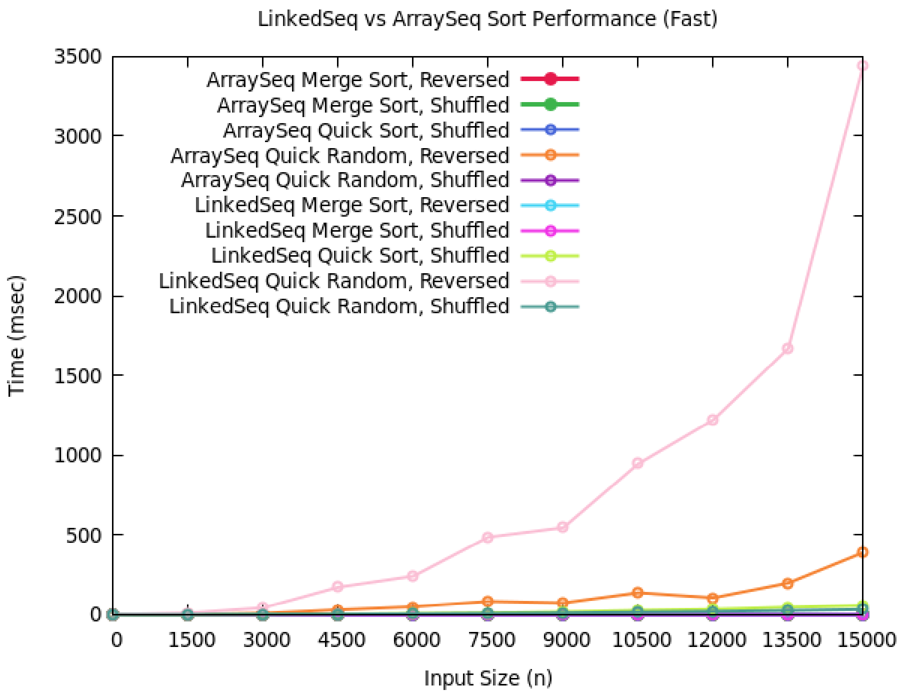
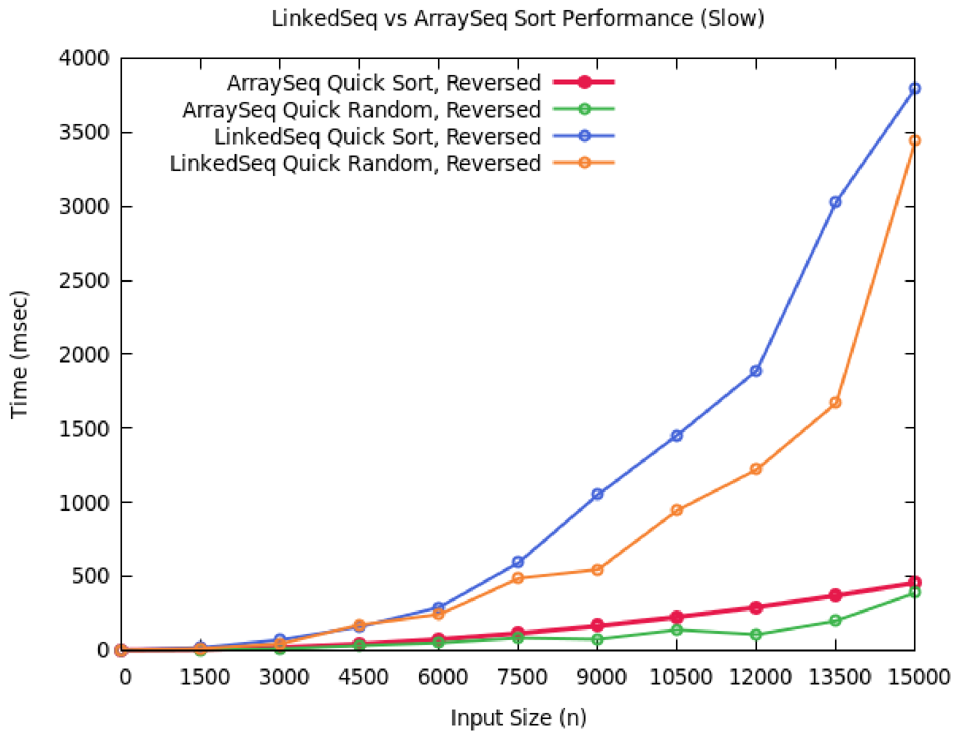
Samuel Sovi
CPSC 223
HW-4





In the first graph, we see the performance running of our faster, more ideal sorts for each of the types of data structures. While majority of the tests are under 500 msecs, we can see that the LinkedSeq's reversed is slower compared to the rest. This may be because my implementation favors Linked Sequences that are already sorted because the order that the nodes are added into the "smaller" and "larger" nodes are the same that they were in the original Linked Sequence. If I chose to implement the Linked Sequence's Quick Sort by adding to the front of the "smaller" and "larger" Linked Sequences, then this case would have been more favorable (at the cost of an "already sorted" linked sequence's case). In our second graph, we can see that the

Samuel Sovi
CPSC 223
HW-4
operations performed on the ArraySeq are slower than most in the first graph. This is because quicksort is not as favorable for reversed arrays. Judging by the similar times in the two line graphs, it also may be the case that the pivot was chosen as an element somewhere near the start (the pivot of the regular quick sort). Compared to our sorts from homework 1, the faster sorts used in this assignment are faster with times less than 100 msec whereas the homework 1 sorts had times ranging from 150 msec to 450 msec. For my test cases, I used one case that had 4 elements ordered, one case with 4 elements in reverse order, one case with 4 elements that were not in order and a last case that had 4 elements but there were duplicates. The last case was to confirm that my sorts could handle duplicates since there can be repeating values in an ADT. I did these 4 cases for each of the 3 types of sorts for both of my ADTS.

One of the main struggles in this project for me was making sure that I had all my nodes in my linked sequence pointing to the correct things. Some were pointing to the wrong nodes while others were pointing to nodes when they should have been having their next value be nullptr. Originally I tried to use the loophole of for loops using the size so that I did not have to worry if there were some excess nodes that were being pointed to by my tail pointer. Eventually, I finished that version, but decided that I would see the problem through to the best of my ability to use it as an assessment of myself and my abilities. I ended up cutting all nodes off at the correct location and using while loops that looped til the next equaled nullptr. This saved a lot of time and work for the computer in sorting my arrays while keeping the Linked Sequence true to its intended size.