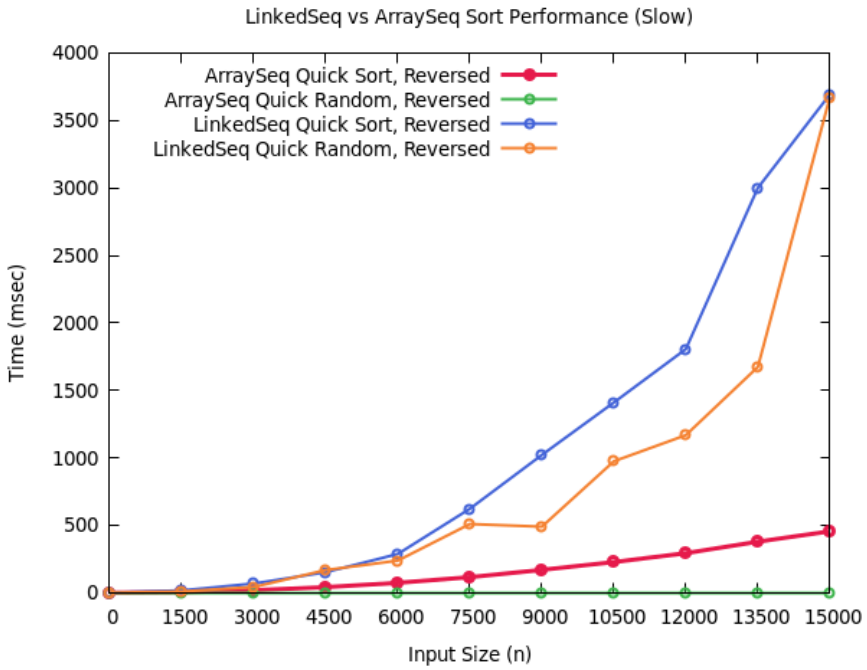
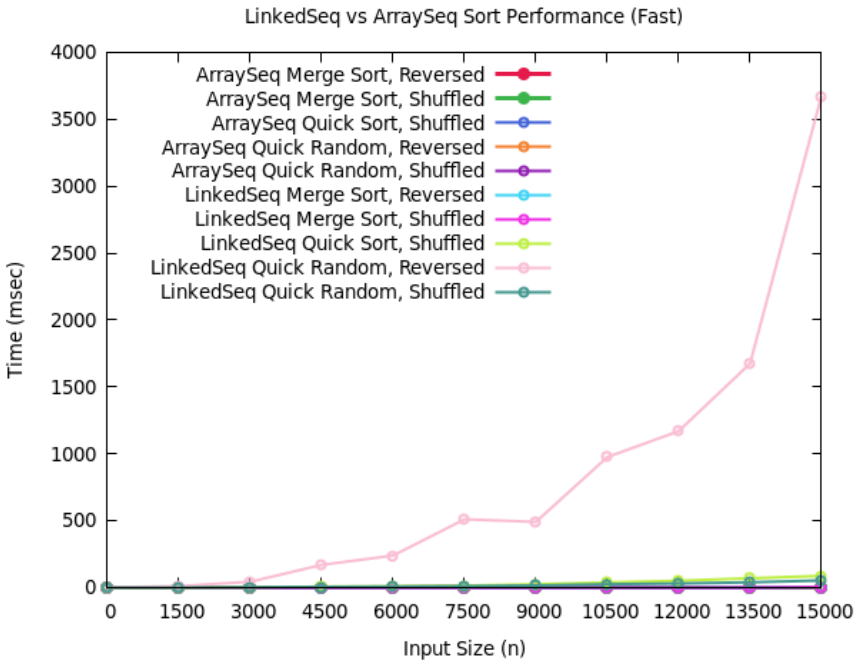
HW4 Writeup (Spring 2022)

Plots



The graph above depicts the worst cases of both quick sort implementations over the resizable array and the linked sequence. It is understandable that these cases take longer as the worst case of quick sort is when the values to be sorted are in a reverse order. Also, the linked sequence quick sort cases appear to be taking an excess amount of time to sort the nodes. I suspect this is due to an incorrect or lack of consideration in utilizing the associated tail pointer or this is due to weird and more complex implementation of quick sort for a linked sequence. Because my implementation is likely wrong for quick sorting a linked sequence, a comparison between the sorting algorithms in HW1 will not be accurate. As merge and quick sort are named efficient sorting methods, I can presume that the idea performance graph above would have a max time of less than 500 milliseconds.



This second graph shows the merge sort cases, the best quick sort cases, and repeats a couple of worst quick sort cases for reference. As I previously mentioned the reversed case for a random quick sort on the linked list appears to be incorrect in reference to the HW1 sorting methods and others within this homework. That said, I cannot attest to anything beyond the apparent successful performance of my merge sort algorithm as well as the best cases of my quick sort algorithms.

Tests

To better test my implementations of each sorting algorithm involved in this assignment, I wrote six new tests focused on the behavior of my algorithms when sorting four item lists. In using unique values across each test type rather than across each individual test, I could track if each respective pair of algorithms produced identical results. *FourElemMergeSort* tested merge sort's ability to recognize a relatively sorted or unsorted list and properly shift only the necessary values. *FourElemQuickSort* tested the normal quick sort's ability to choose a pivot from an even list and still sort values correctly despite their completely shuffled order of insertion. Finally, *FourElemRandQuickSort* really did the same thing as the prior test, but it was necessary to add due to the extra step of swapping the pivot value with the first value in the list.

Challenges

Much of my impediments were associated with lists failing to be sorted by the random pivot quick sort algorithms. Initially, the random quick sort algorithms for both the resizable array and the linked sequence did not produce a sorted list. After careful consideration pertaining to indexing the pivot point and organizing the remerging or reattaching phase, I was successful in producing sorted lists. However, my attempts at optimizing these algorithms for better performance came up short.