HW3 - Efficient Sorting

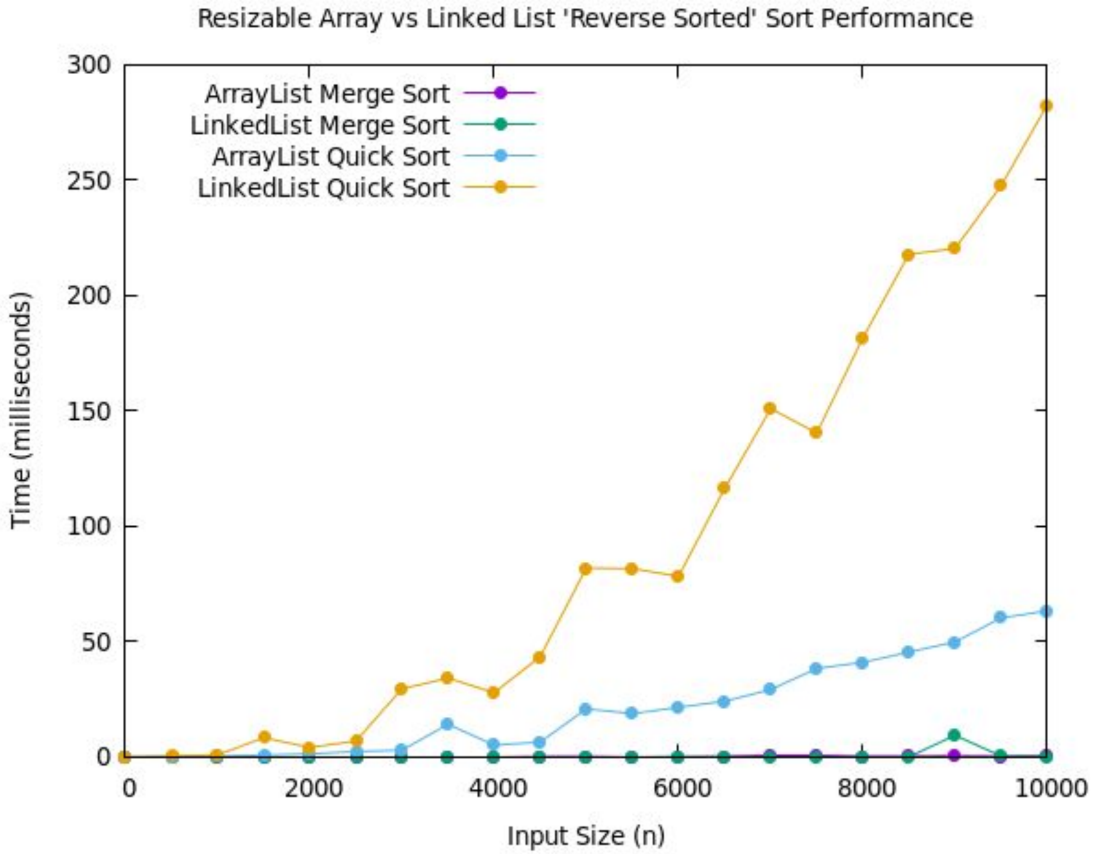
CPSC 223 - Wesley Muehlhausen

Issues and Challenges

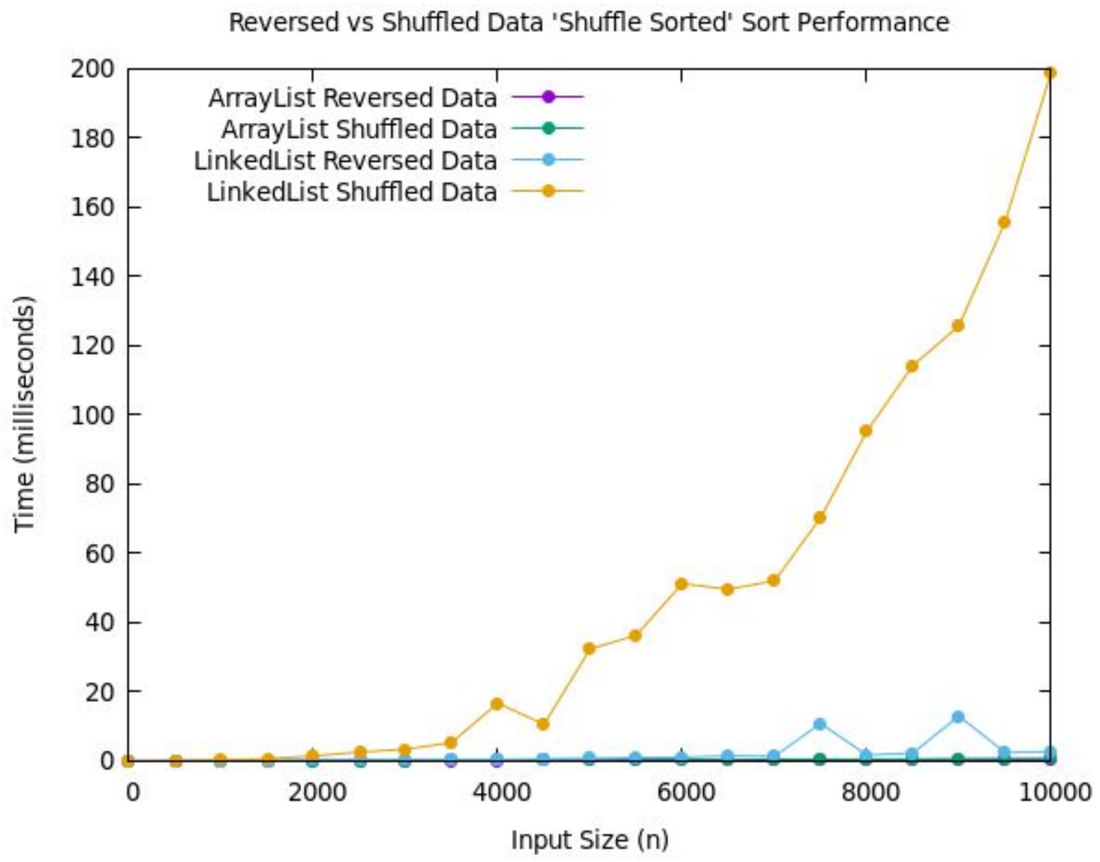
- This project was probably the hardest so far. I was able to get all 4 of the functions to work for the unit tests as well as the tests that I thought could break it. But, the unit tests seg faulted for all of my functions except for performance test 1. Linked List implementations took much longer since swapping nodes is much more complex than with arrays.

- Turning the pseudo code into code for the array list was pretty easy. But the fact that they didn’t pass the performance test was really confusing. Even though linked list implementations are more complicated, it was almost easier for me to code this part of the project. I would also like to add that I think it was easier to write the quick sort algorithm rather than the merge sort.

- One of the reasons why all of my performance tests did not not work was because of specific conditions in my sorting algorithms. Most specifically, I needed cases for it the incoming lists were size 1, and if during the recursive part of the sorting algorithms, the splits were size one. Once I added those conditions, everything worked.



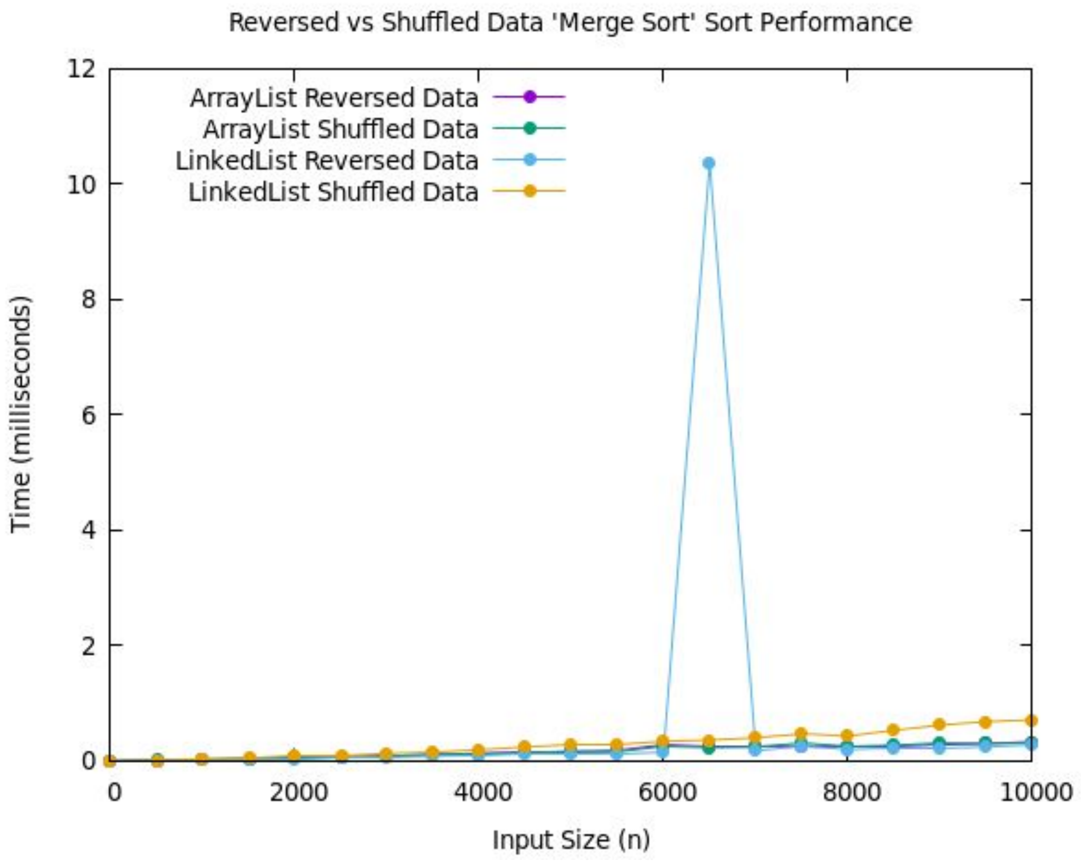
-This graph makes a lot of sense. As we know, Quick Sort has a worst case time complexity of O(n^2) which is worse than Merge Sort worst case which is O(nlogn). This is why Quicksort performs much worse in the graph. Also, I would guess that LinkedList performs worse than ArrayList because of the general nature of splicing taking more individual operations than swapping used in ArrayList. It could also be that my LinkedList implementations are more complicated in general.



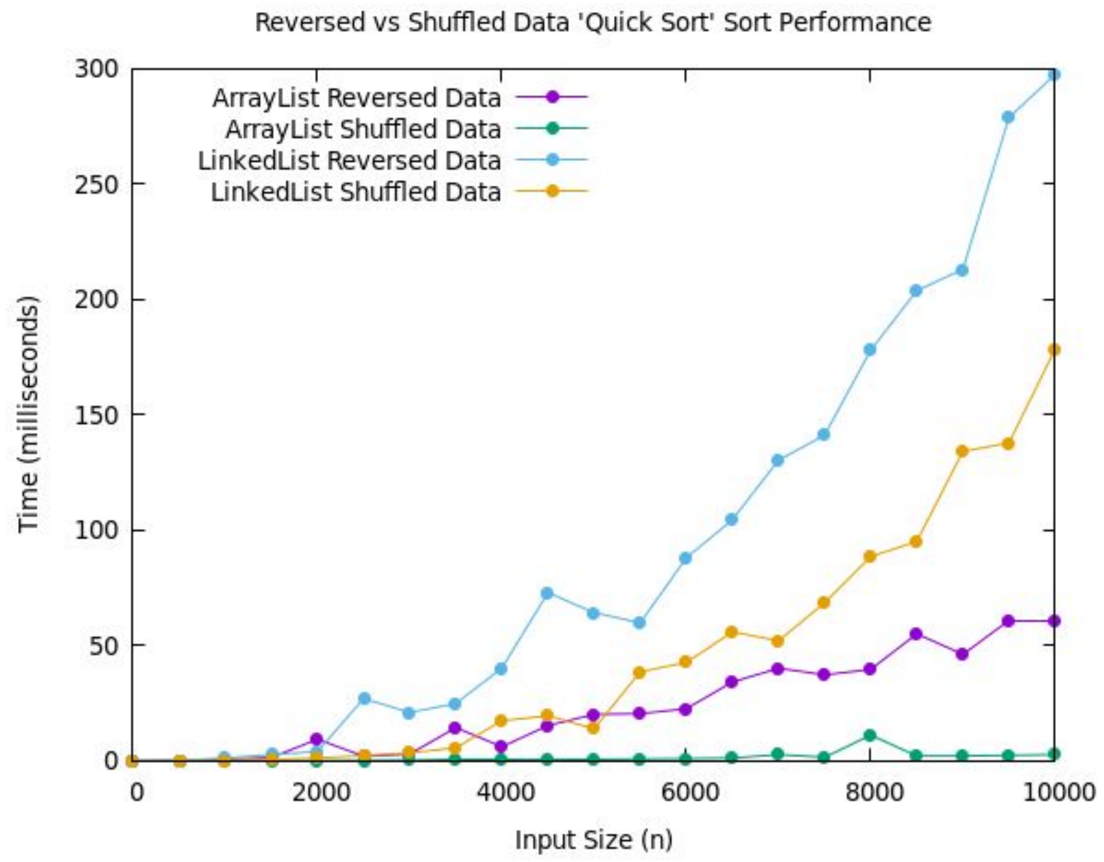
-This graph is not as I would expect it to be. On average case, quicksort and mergesort are both nlogn which would make me think that they would be fairly similar given that the list was shuffled and not in reverse order. The only reason I could see the linked list quicksort being so slow is that my implementation had too many cases or was more complex than it needed to be.

-Looking at the perf file, the labels on this graph should be:

- Purple: A.L. merge sort
- Green: L.L merge sort
- Blue: A.L. quick sort
- Orange: L.L. quick sort



-I am astonished that my implementations for merge sort perform so well and are so uniform. I would guess that my linked list implementations would not perform as well as my array list implementations because I might have made it too complicated but I guess not. The reason this graph is fairly flat overall is because regardless of the data coming in, for merge sort the best and worst case time complexities are both O(nlogn).



-So for both LinkedList and ArrayList, Reverse data takes more time than the shuffled data which makes sense given quicksort worst case scenario vs its average time. It is interesting that ArrayList reversed data is still faster than linked list shuffled data because of the fact that it theoretically should be better time complexity. I think this offset is because of my implementation.