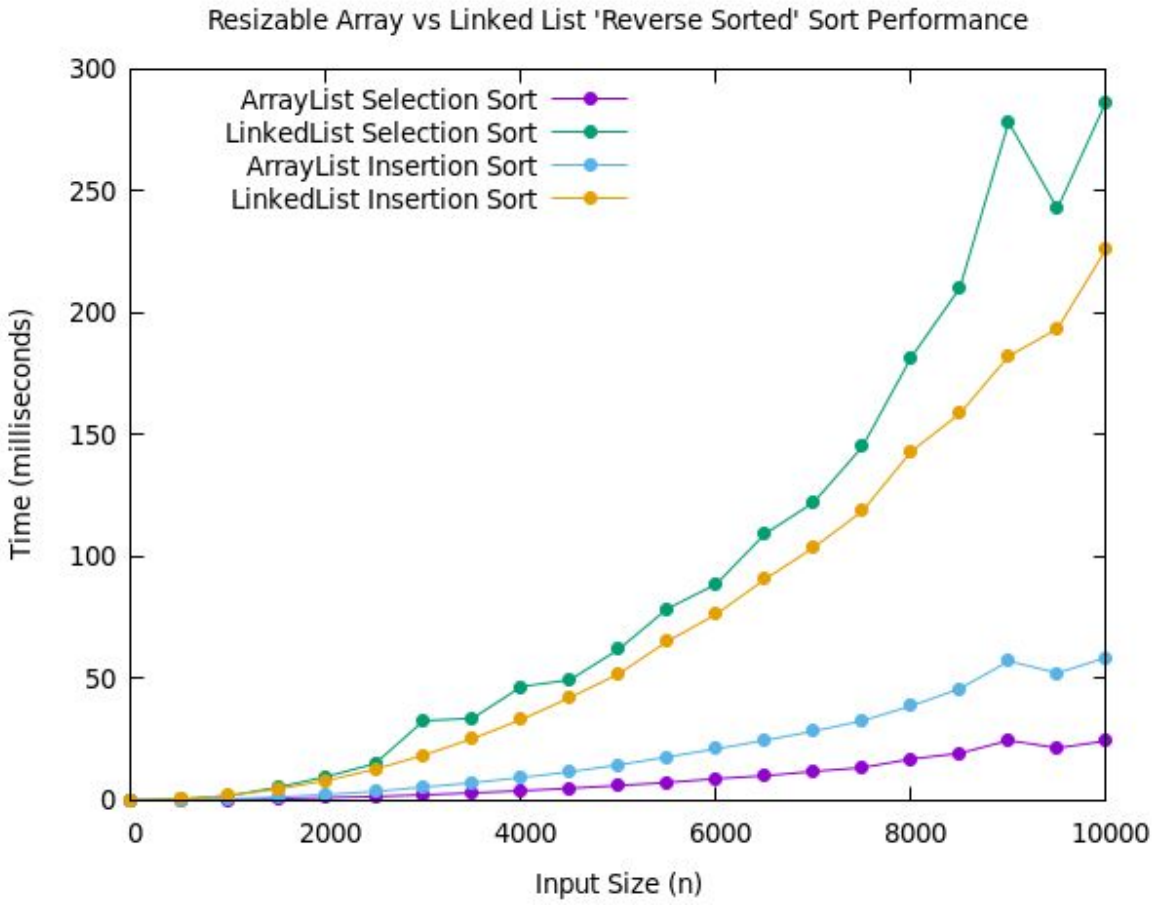
CPSC 223 - HW2
Wesley Muehlhausen
Due: 9 / 23 / 2020

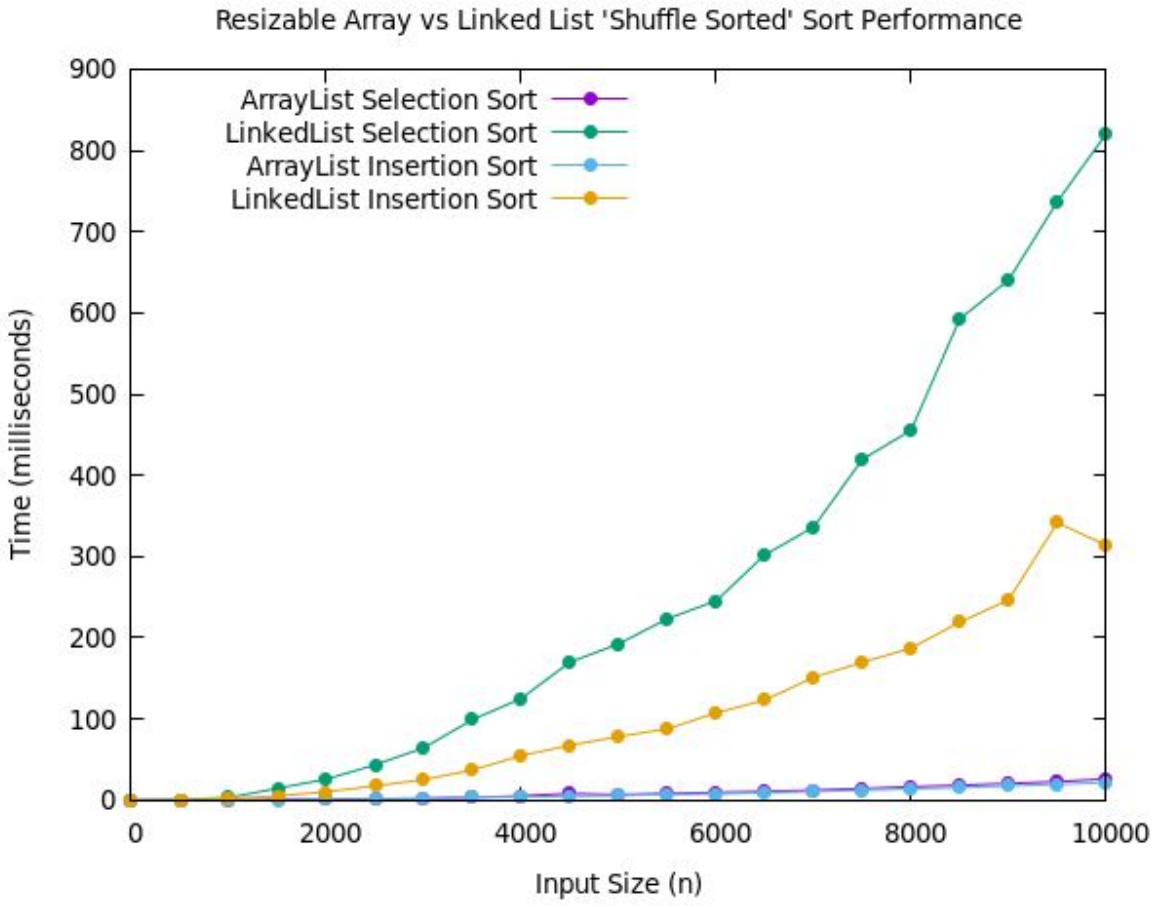
HW2 Selection Sort & Insertion Sort

Reflection:

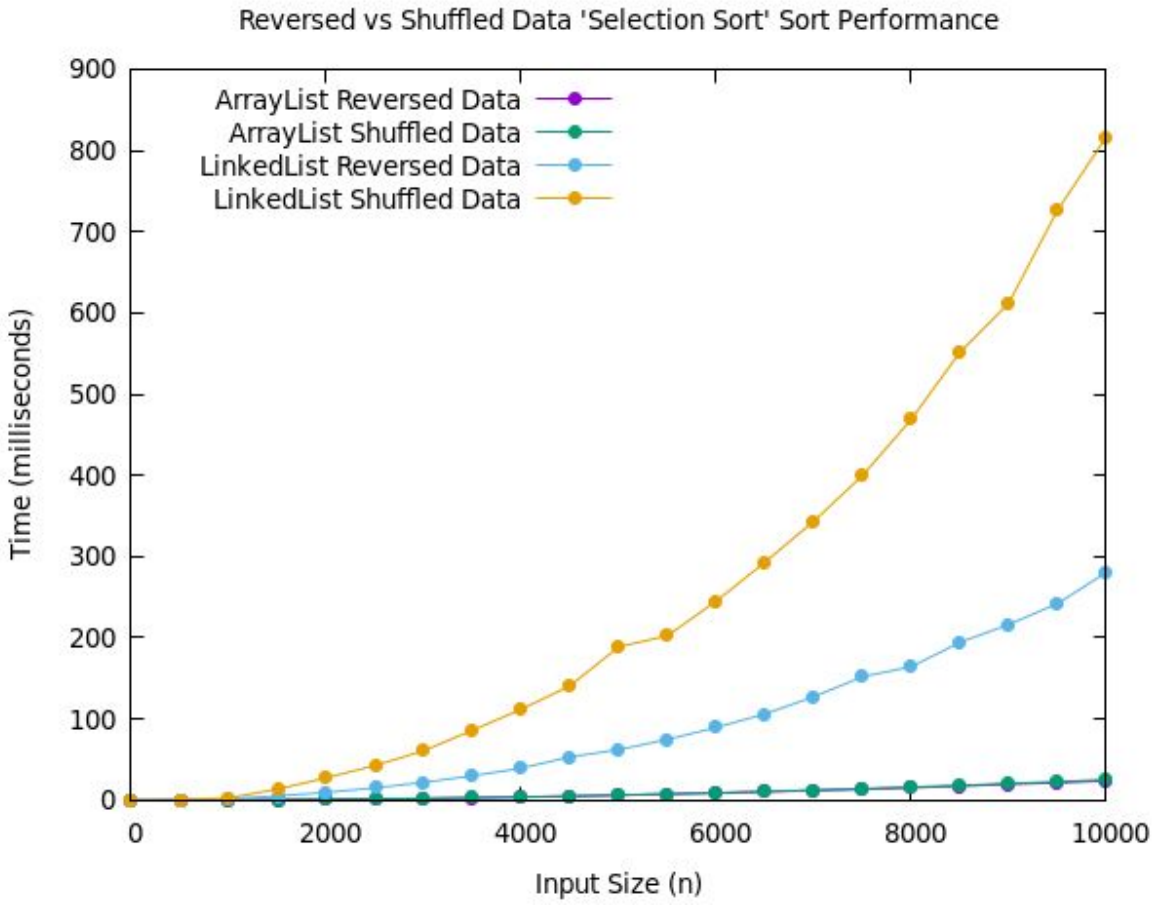
- This homework was very nitty gritty. The first challenge of this was setting up the main loops for the sorting functions. The first problem I had was getting the bounds right so there were no seg faults.
- One huge problem and tiny fix was remembering to put an if statement at the beginning of the sorts to see if the lists were too small or already sorted.
- The arraylist functions were pretty easy. I breezed through those. The linked list was much more difficult because of the slicing.
- One thing that helped me was putting pointers before the nodes that I wanted to manipulate instead of on them. This made it so they could be accessed better.
- Overall though, splicing was very hard, and it made it so much easier to code the splicing while I was drawing it on paper.
- Also what I didn't expect going into this homework was how many potential cases there were for this assignment. That was what made this assignment so strenuous.
- There were also the stupid errors that I didn't notice, like using an "=" instead of an "==" inside an if statement and not noticing.



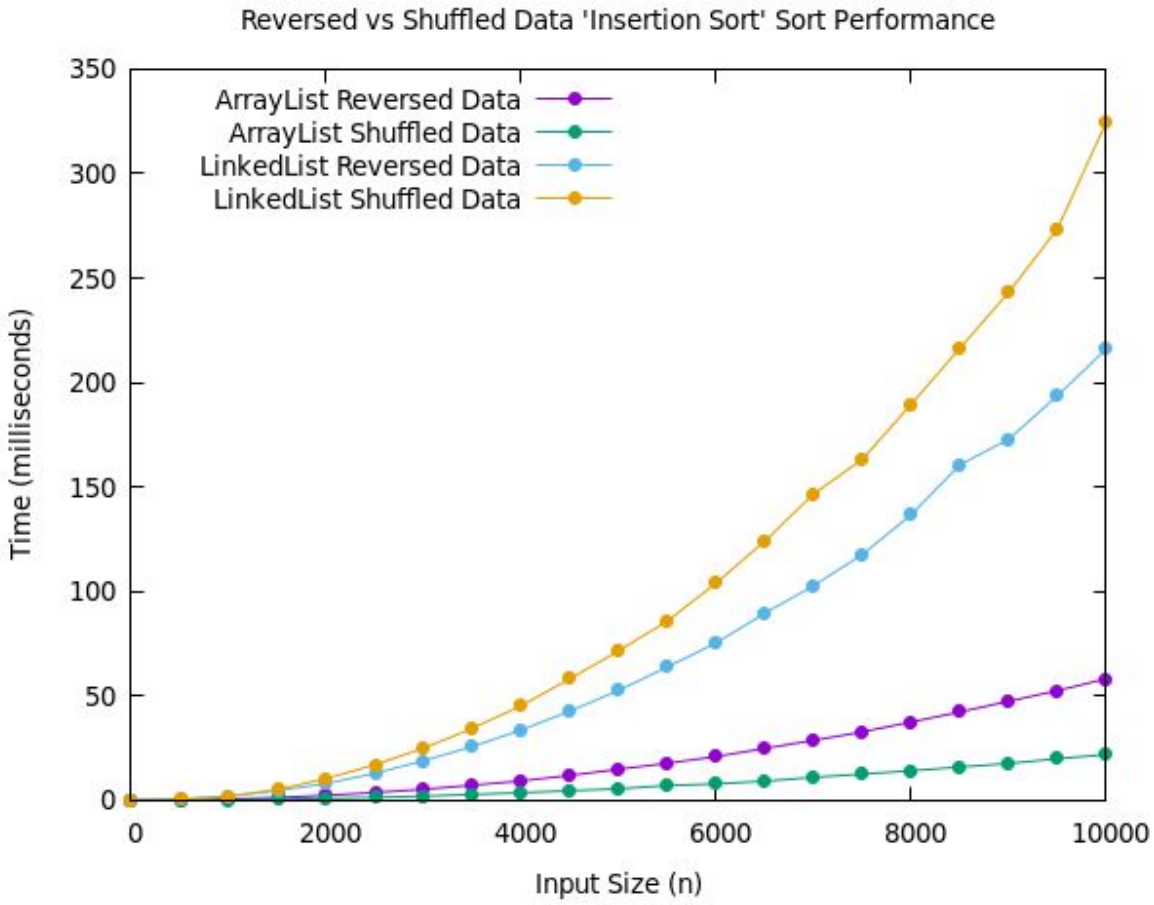
As we can see in this graph, Array List beats Linked List easily in performance. This is the very worst list to be sorted since it is in reverse sorted order which means the maximum number of swaps. This graph makes sense since Array List performs data swaps very efficiently because only two pieces of data have to move (unlike an arraylist remove or add). Linked Lists on the other hand do not do very well with swaps because the nodes have taken many moves to do so, on top of that the pointers have to traverse the list which also slows the program down. It is also notable to point out that for the Array List, selection sort is faster, but with the Linked List, insertion sort is faster.



As we can see in this graph, Array List beats Linked List easily in performance. This is an average list to be sorted since it is in random sorted order which means a random number of swaps. This graph makes sense, being the same case as the previous test, since the Array List performs data swaps very efficiently. The Array Lists had minor performance boosts on this probably because the randomness of the list made for less swaps. Linked Lists on the other hand did not do very well. It is interesting that the linked list performance tanked for this order compared to reverse sorted order going from (200-300ms to 300-800ms). It is also notable that the Linked List selection sort was much slower than the insertion sort for this test compared to the last test.



As we see in this graph, using selection sort, Array List reversed data barely has the best performance, followed closely by Array List shuffled data. These two performances are virtually the same, both very fast as expected. Linked List reversed data easily has the best performance for a Linked List, followed not very closely by Linked List shuffled data. Based of the graph, shuffled data poses more of a performance issue than reverse sorted data.



The performances of the Linked List are better than the selection sort performances, but the reversed and shuffled data performances are similarly related to each other like last time. Alternatively, we see that for Array List, the reversed data took much longer than the random data. This is what I would expect because it is the worst case which means it does maximum swaps.