Project 1: Sorting Algorithm Implementations

Within project 1, using our doubly linked lists, we implement selection sort, insertion sort, merge sort, and quicksort.

Algorithm Correctness

The correctness of each algorithm is verified with the `sort_correctness_test`. There are 5 test cases: no elements, one element, two elements, three elements, and seven elements.

Graphs

Every sorting algorithm was run against each given input data file 20 times, and the median time taken to sort in nanoseconds was plotted. Selection sort was the exception because it took exceptionally long on some input files, so it was only run once.

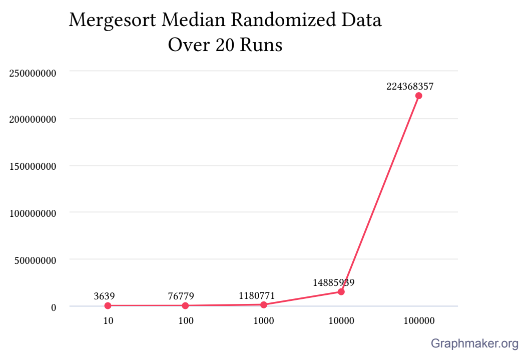
Mergesort

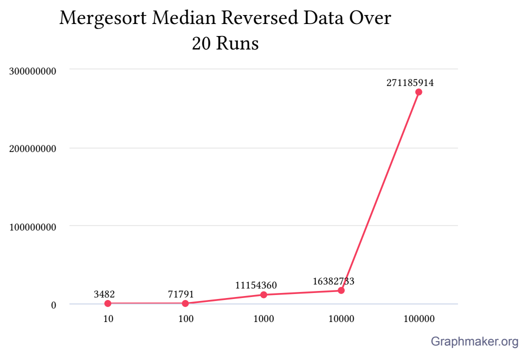
Mergesort was implemented using the pseudo code from the CS101 textbook (page 436). The main difference from my implementation to the textbooks is manipulating pointers instead of indexing and slices. This is because indexing is not $O(1)$ for doubly linked lists and slicing is not implemented for my doubly linked lists.

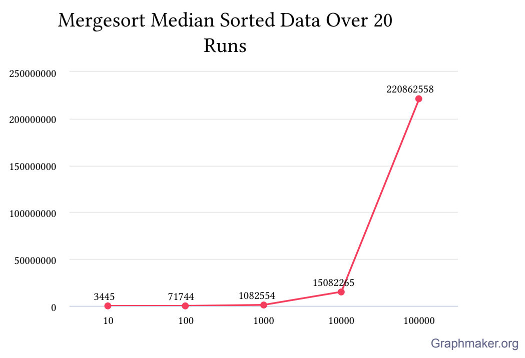
Issues

Unnecessary copies of arrays are made in my implementation of Mergesort. Specifically, a copy is made of the left and right of every subarray because I could not find an eloquent way to "borrow" a slice of my doubly linked list.

Runtime changes and efficiency





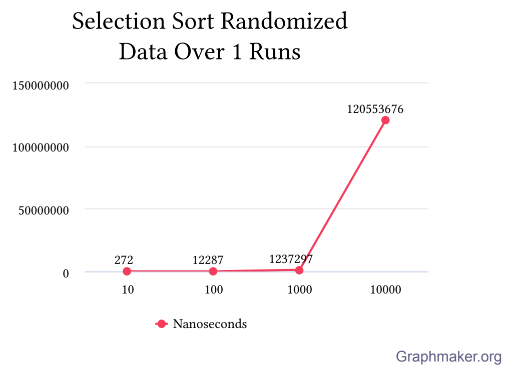


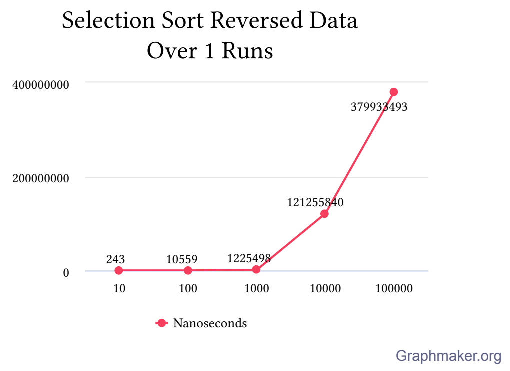
As expected for mergesort, every graph is roughly the same since the order of the data does not matter in its time. Furthermore, the graphs show roughly $O(\log(n))$.

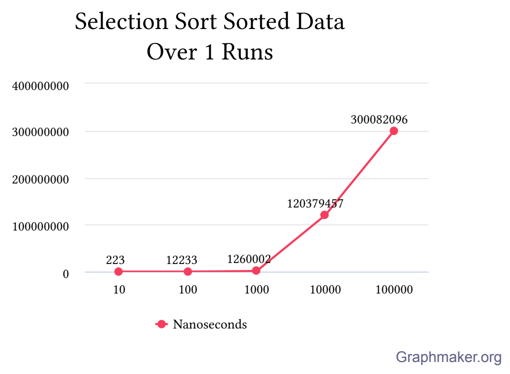
Selection Sort

Selection sort was exceedingly easy to implement. I ran into no issues and it very surprisingly worked the first try I ran it.

Runtime changes and efficiency







As expected for selection sort, every graph shows roughly $O(n^2)$.