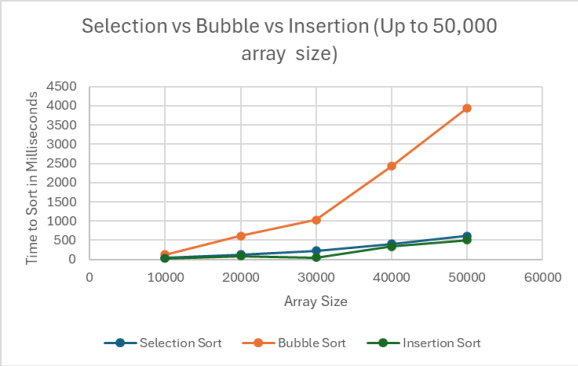
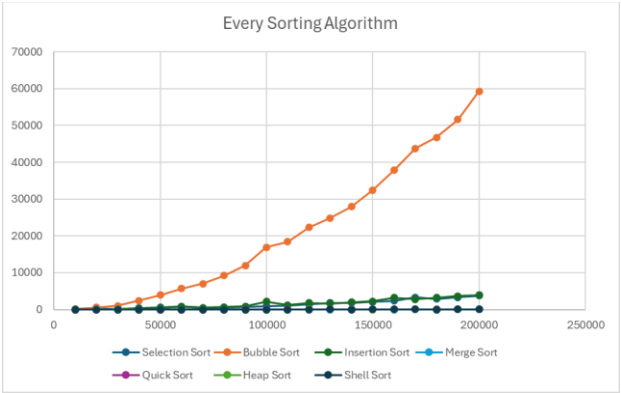
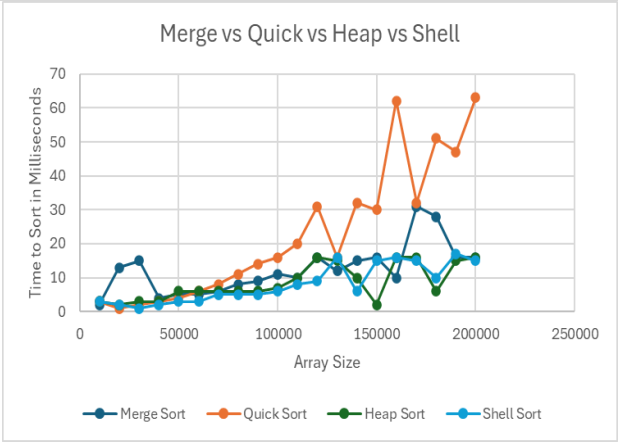
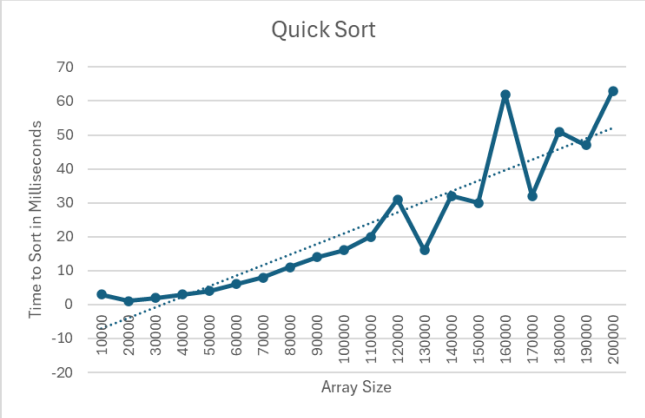
In programming, there will be plenty of time in which you will need to sort several different data types. This can range from arranging integers from least to greatest, to sorting different objects depending on their properties. The limits are endless, and sorting can fill a wide range of situations. With that being said, there are also a wide range of algorithms used in order to sort data, all of which differ in complexity and efficiency. In our Project 3, we used a few of the more common sorting algorithms and tested how quickly they could sort through different array sizes with randomly generated elements from 0 to 100. We started at a lower array size, such as 10,000, and incrementing by 10,000 stopped the test at 200,000. The results we gathered were both informative and interesting. First, we'll start by dissecting the first three and arguably least complicated of the sorting algorithms, Selection, Bubble and Insertion Sort. I would categorize these algorithms as "Tier 2" in comparison to the rest we tested. These algorithms get the job done and do so reliably, but when using large data pools come out to be wildly inefficient, especially Bubble Sort. If you look at the figure on the left, you can see that we measured the time it took to sort each data set in milliseconds. At around the 50,000-array size marker, insertion sort and selection sort take around half a second, while bubble sort takes about 4 seconds. Of the three sorting algorithms, its clear bubble sort comes out last in terms of efficiency. So why would someone ever use these sorting algorithms in the first place? Well Bubble Sort and Insertion Sort are both stable, which means key value pairs cannot be swapped and will always remain in the same order they were in the original data set (GeeksforGeeks). However, an algorithm like Selection Sort is not stable, which means it can run into this issue. Our simple data set does not implement hashing, so this would not be a problem for our test, but it is something to take into consideration otherwise. These algorithms only shine when sorting through smaller datasets. When we compare our Tier 2 algorithms to our other algorithms, which would include Merge, Quick, Heap and Shell Sort, the time in which it takes to sort is not even close. If you look on the graph on the right, you can see Bubble, Insertion and Selection sort all somewhat above the array size axis, every other algorithm however has completely flat lines on it. That's because the sorting rate in milliseconds is so low it can't even get off the ground. These algorithms are what I'd call "Tier 1", it's essentially a different ballpark in comparison to the three previously mentioned algorithms. If we consult the graph below the previously mentioned one, we can see somewhat consistent growth from every algorithm, with Quick Sort obviously taking the most time, but by negligible numbers. If you







observe closely, you can see a lot of spikes in time, some noticeable ones are Merge Sort in the 0 to 50,000-array size range, and Quick Sort in the 150,000-array size range. The interesting part about having a randomly generated array is that sometimes you can be dealt a data set that's opportune for one algorithm and bad for another. For instance, Quick Sort's worst case time complexity is O(n^2), which can happen when smallest or largest element is always chosen as a pivot point (GeeksforGeeks). In our implementation of Quick Sort, we always have it always select the last element, which may or may not be the smallest or largest. In terms of stability, none of the Tier 1 algorithms are stable except for Merge Sort, and with its efficient time complexity make it seem like a very good choice for any data set. However, all these Tier 1 algorithms only shine with large data sets. For something that is much smaller than an array size of 10,000, using these algorithms will become less efficient than using Tier 2 algorithms. Efficiency like mentioned before is also affected by how the data set is laid out, Shell Sort will work really well if you have a near sorted data set, possibly achieving Omega(n log(n)), but can possibly operate as poorly as O(n^2), which would be on par with our Tier 2 algorithms.(GeeksforGeeks). A lot of these algorithms are most effective on a case-by-case basis, once you really understand the type of data set you will be sorting is when you can choose the best tool for the job.



Work Cited:

Manwani, Chirag. "Stable and Unstable Sorting Algorithms." *GeeksforGeeks*, 21 Nov. 2024, www.geeksforgeeks.org/stable-and-unstable-sorting-algorithms/.

"Quick Sort." *GeeksforGeeks*, 16 Nov. 2024, www.geeksforgeeks.org/quick-sort-algorithm/.

"Merge Sort - Data Structure and Algorithms Tutorials." *GeeksforGeeks*, 16 Nov. 2024, www.geeksforgeeks.org/merge-sort/.

"Shell Sort." *GeeksforGeeks*, 11 July 2024, www.geeksforgeeks.org/shell-sort/.