Assignment 6: Write Up

After completing Assignment 6, there were many conclusions I was able to draw up regarding sorting algorithms. In particular, regarding their complexities and their runtimes.

In terms of complexities, the order of how complex the algorithms were is Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, and Quick Sort. The complexity of Quick Sort and Merge Sort (due to the recursive helper functions required to make them work) are far greater than the other three, whose complexities are somewhat comparable. Furthermore, Merge Sort also requires more memory, used to create extra merge arrays, than the rest of the algorithms. The other four algorithms' space requirements are similar since all four work by partitioning the original array only and not making more arrays.

In terms of runtime, the order is mostly the reverse of the complexity order. However, the runtimes usually were dependent on the data set. For example, all five algorithms took less than a second when sorting 100, 1000, and 10,000 numbers. However, after sorting 100,000 numbers, the different runtimes finally began to appear. Bubble Sort had the highest runtime at 29 seconds, Insertion and Selection were second with 10 seconds, and Quick Sort/Merge Sort took 0 seconds. The first three sorting algorithms' results got drastically worse after that 100,000 numbers test, culminating with Bubble Sort taking nearly an hour to sort 1 million numbers and Selection/Insertion sort taking about half that. Quick and Merge Sort did not take any more than a second until the 10 million numbers test, where Merge Sort took 5 seconds while Quick Sort took 3 seconds. The difference between the two is most likely because of the sheer number of repeats present in the data set, as the data only went from 0 – 20. For the other three algorithms, Bubble Sort always took the longest while Insertion and Selection Sort were exponentially shorter than Bubble Sort but still took far longer than Quick/Merge Sort to complete. Insertion was usually faster than Selection except for the 750,000 numbers test, where Insertion Sort took 80 more seconds than Selection Sort. However, Selection Sort took nearly 3 minutes longer than Insertion Sort with 1 million numbers, so most likely, the differences are due to testing limits. After looking at the overall data, the order of runtimes from best to worst was Quick/Merge Sort, Insertion Sort, Selection Sort, and finally Bubble Sort. I did not pick between Quick Sort and Merge Sort for the fastest algorithm due to my testing limitations, causing me not to get enough data to make an educated selection.

Looking back at the simulation's overall results, I see a considerable tradeoff between the algorithms; the more complex they are, the faster they become, and the fewer machines that can use them. As a result, programmers must strike a balance between runtime and complexity with sorting algorithms. Most of the time, in today's era, machines can handle the complexity of a Quick Sort or Merge Sort. However, some devices cannot, meaning Selection Sort or Insertion Sort will have to do despite the longer runtimes. Looking at the actual testing I did, my usage of C++ for this assignment made recursion possible, allowing for Quick Sort and Merge Sort. However, if I were stuck using a different language, like assembly, I would be forced to stick with the other three sorts due to the complexity problem. Finally, the biggest flaw in the testing was the lack of comprehensive test data. For my tests, I created ten different test files of varying sizes and varying data points. However, these tests only looked at these ten files instead of the hundreds of even bigger files needed to get a good result regarding the runtimes of these algorithms. Therefore, if I had a way to create hundreds of more test files that are more adequately suited for the tests, I would get better results. Then, I could make better judgments about these algorithms' runtimes, particularly the Quick and Merge Sorts.

Overall, the assignment gave me a better understanding of how different these algorithms are with both their runtimes after 10,000 numbers and their complexities for programming them.