I haven't ever timed my programs like this before, so I honestly wasn't too sure what to expect. As such, after timing the sorting algorithms I implemented, the times weren't any more or less drastic than I thought they would be. Some algorithms were obviously quicker than others, but that was to be expected from their worst case runtimes.

When picking one algorithm over another, there are various tradeoffs to consider. Firstly, each algorithm has its own worst case runtime. Additionally, certain algorithms are more efficient or effective depending on the dataset that they are run on. In the case of insertion sort, for example, it is two times faster than bubble sort since it is doing copies and not swaps. However, if the input is sorted in reverse, its runtime is the exact same as bubble sort. Similarly, quicksort has a Big-O runtime of O(nlogn) if the data input is random, but quadratic if the input is not random.

The assignment was done using C++, a compiled language. As such, once the code for the assignment was compiled it will run faster than if it was written in an interpreted language like Java or Python.

Some shortcomings of this empirical analysis are that I did not thoroughly describe tradeoffs between each and every sorting algorithm we learned about and discussed in class. Another shortcoming of this empirical analysis is that I did not expand very much on other reasons as to how compiled languages can be better or worse than interpreted languages in certain cases.