Assignment 8 Results

In this project I implemented selection sort, merge sort, and shell sort in c++. With a very small text file (~10 lines), the runtimes for each sorting algorithm were nearly identical. However, when sorting a file with 1,000 double values, it becomes obvious that some of the algorithms are faster than the others. I was not exactly surprised by the time differences between the algorithms. As the input size grows, the average case runtimes will diverge quite quickly since (all in the average case) selection sort is in $O(N^2)$, merge sort is in O(NlogN), and shell sort is in $O(N^{1.5})$.

When the input had 1,000 double values, selection sort took 5 times longer than merge sort; 10,000 values took 37 times longer, and 100,000 values took 324 times longer. The duration of shell sort is a little bit more complicated. When using a gap value of 3 or 4, the runtime is approximately halfway between the other two algorithms, but when using small or large gap values like 1 or 5, the runtime is approximately the same as selection sort. I am guessing this is because more sorting must be done when the gap is farther apart after running the insertionsortinterleaved() method. In the average case, selection sort is the slowest algorithm, followed by shell sort, with merge sort being the quickest. It is worth noting that selection sort only stores a temporary variable for sorting, whereas merge and shell sort both store extra arrays. This means that the auxilary space complexity of selection sort is smaller than the others.

Analyzing these algorithms in c++ will be quicker than interpreted languages, but not as quick as assembly or any other lower-level languages. My empirical analysis is limited because it compares the average cases of the runtimes because I did not sort anywhere close to an infinite number of doubles. That being said, my experimental results agree with the upper bounds given in Zybooks.