1. I will demonstrate the effective run-time of selection sort, insertion sort, shell sort, merge sort and quick sort by collecting the number of comparisons and swaps that result from the input of integer arrays of varying sizes. The array sizes will be 10k, 100k, 1m, 10m, and 100m. The integers in the arrays will be ordered randomly to avoid best and worst cases for each sort method. The same randomly assorted array of each size will be passed to all sort methods for consistency.

4. Link to the Comparison Graph Link to the Swap Graph
The results of comparisons and swaps were so different from sorting method to the next that the line graph is difficult to read. Therefore, I will link the graphs in this analysis so that you may interact with the graph to hide portions (selection and insertion sort) and get a better look at the data. I apologize for the inconvenience and will work on developing my graphing skills.

The amount of comparisons in the average cases for selection and insertion sorts were vastly larger than the merge, shell, and quick sorts. Out of the remaining three sorts merge has the least amount of comparisons followed by quick sort.

The amount of swaps in the average cases for insertion sort were much higher than that of the other sorts. The lowest amount of swaps were generated in the quick and selection sorts.

These results lead me to conclude that quick sorts has the best runtime because the amount of comparisons are just above average in relation to it's most efficient counterparts but quick sorts swaps are significantly lower.