Assignment 3 - Sorting -- *WRITEUP.pdf*

## Introduction

This program implements different sorting algorithms in order to understand their differences and determine the best execution times based on the circumstance.

## Main Learnings

*What you learned from the different sorting algorithms?*

What I mainly learned from the different sorting algorithms was the different ways to have efficient and inefficient ways to sort the same array in terms of how many moves/compares they take. What else I learned can be explained by the following questions.

*Under what conditions do sorts perform well?*

Insertion sort works best when the array is already mostly sorted (complexity n). Shell sort works best when the array is mostly sorted with $O(nlog(n))$. Quick sort works best when the pivot divides the array into 2 equal halves with best time complexity of $O(nlog(n))$ Heapsort works best when all elements are equal with $O(n)$.

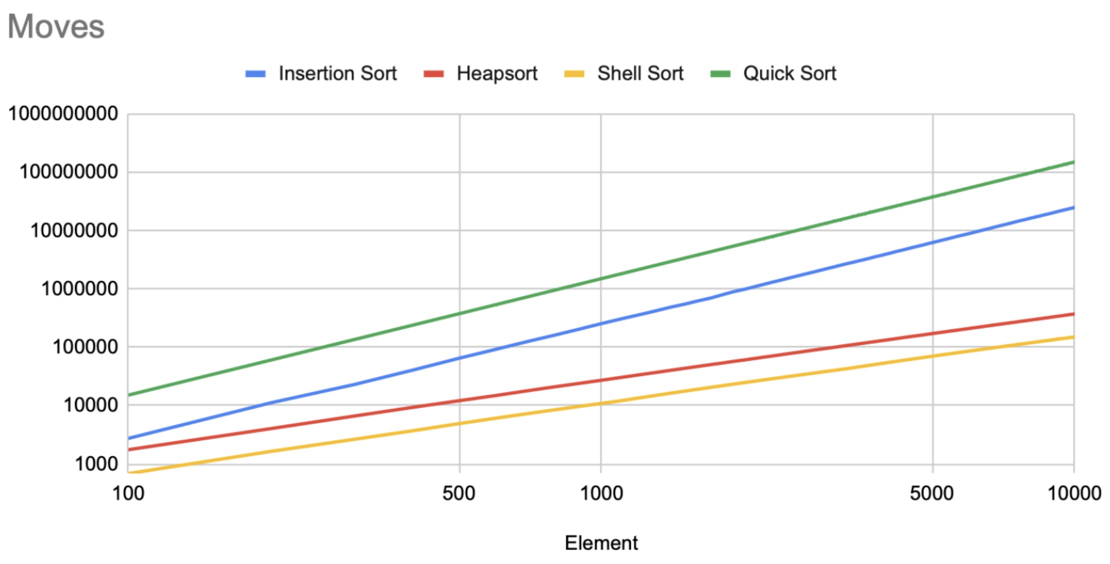
*Under what conditions do sorts perform poorly?*

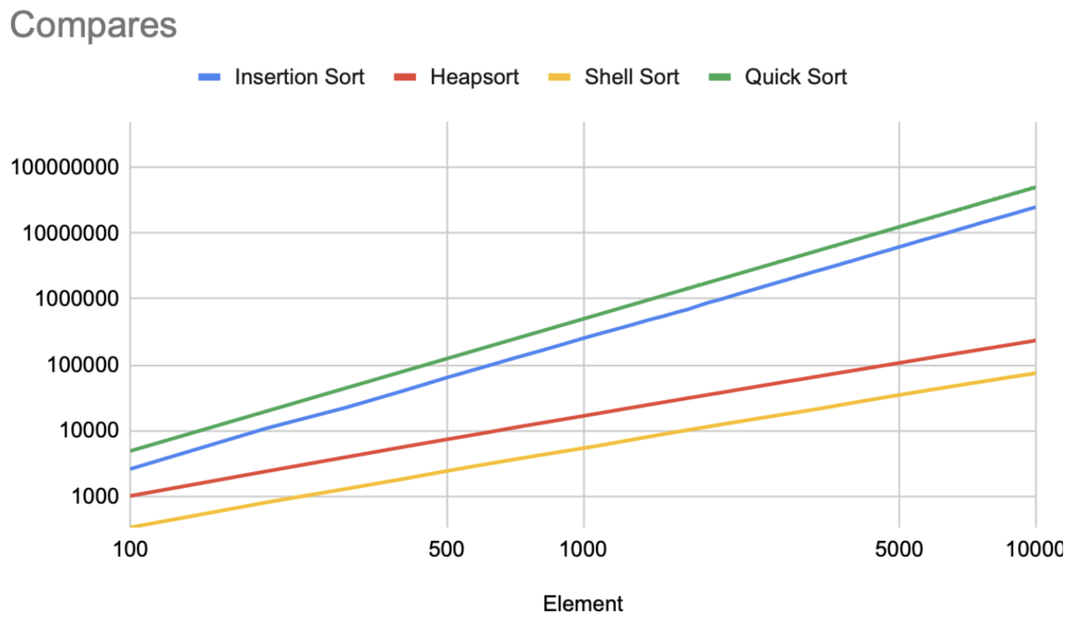
Quicksort and Insertion sorts' worst case is when the elements are reversely sorted with the time complexity of $O(n^2)$. The worst and average case and for heapsort is time complexity of $O(nlog(n))$. The worst case for shell sort depends on the gap sizes and can have the time complexity of $O(n^2)$.

What conclusions can you make from your findings?

I found that in general, Heapsort would be less preferred in use as its best and worst case time complexities are the same at $O(nlog(n))$. This means it is slower than Quicksort, whereas Insertion sort is faster than Quicksort with a smaller n. I mainly found how each sort behaves differently in terms of the size of the array → how at one point Heapsort would be better to use because it's time complexities at all levels are the same.

## Graphs and Analysis





After graphing the moves it takes for all sorting algorithms to sort an array, I found that quicksort tends to increase faster and has a greater slope, Insertion sort has a similar slope to it and shell sort takes the fewest number of moves in general. Quick sort also "quivers" a bit in its graph, the values going back and forth a bit which is likely due to the fact that it is a recursive function and uses a "partition" to split the array in half and swap elements from the left and right accordingly. Shell sort had also had a similar pattern but much less noticeable and it is likely due to its use of gaps in order to compare and swap values in the array.

Compares also had a similar graph to moves in terms of which sorts had more compares/moves at each size of the array. What was interesting was noticing the variation in insertion sort's graph, possibly due to its method of comparing values to see if they are sorted or unsorted and then moving accordingly.