WRITEUP: Assignment 3

Nelly Sin

October 2021

1 What I Have Learned:

The different sorts I have worked with this week were all unique and have different number of compares, moves, and the time it takes to sort the elements of an array.
Although, my shell sort is very off compared to the referenced sorting result that was posted. To me, it's very interesting to see how a little change of insertion sort can be still a lot quicker than the original, like the shell sort.

1.1 Insertion sort

I noticed in insertion sort, the sort takes one element by another and compares – much like how we usually sort things. Such that the best time complexity will be *O(n)* – we will only be sorting in ascending order. And comparing will not be in large increments, such that n + 1 to n + 3 and so on *O(*n^2*)*.

1.2 Heap sort

In heap sort, the sort compares from the bottom of the list, and find the largest element. If it finds the largest element, it will move it upwards till it reaches to the top of the tree. After, heap sort will take the element out and swap it to the end of the array. Heap sort has a complexity time of O(nlog(n)), since heap also compares sorts from in each element with no gaps in between the sorts. Based on the description on the assignment 3, provided by Professor Long, it doesn't have a worst time complexity besides O(nlog(n), based on the sorting the lower bound of the array.

1.3 Shell sort

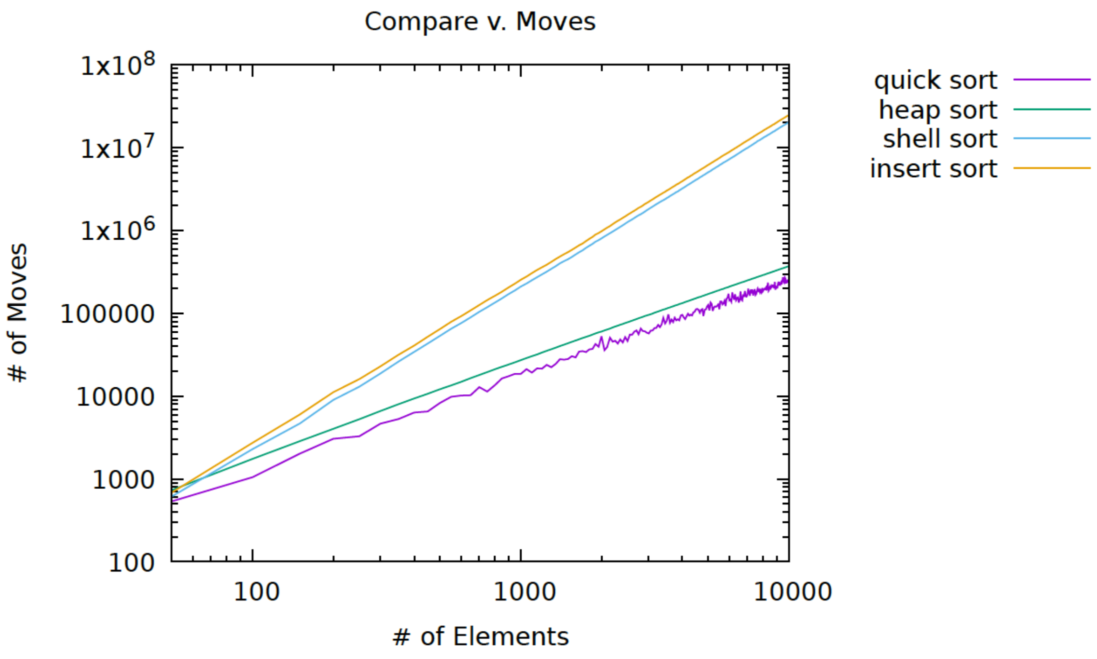
Since my shell sort isn't working very well for moves and comparisons. But, based on my thought on the process of building shell sort. It compares each element based on a gap, so I believe the best time complexity for shell sort is

O(nlog(n)) because, it is based on the generator and gap – a fixed number of which element to compare and sort.

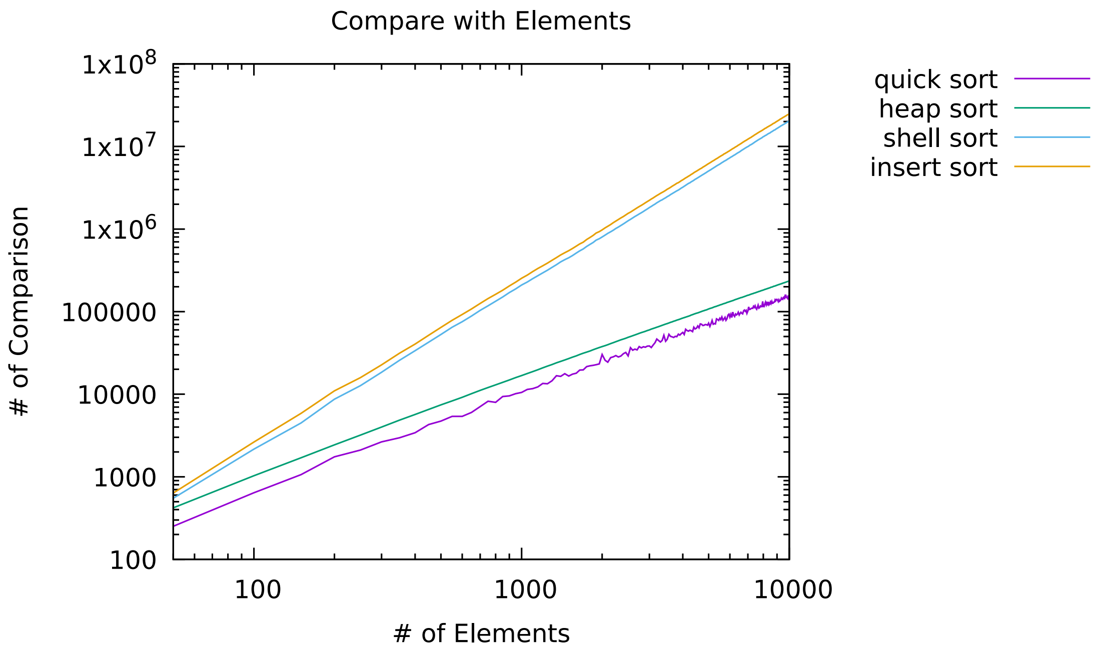
1.4 Quick sort

Quick sort is the fastest sort compared to the rest. It partitions the section of the array and focus on the section and assigning the elements in ascending order. The time complexity of quick sort is O(nlog(n)), alike heap sort it sorts from the pivot and partitioning strategy of quick sort. But, the worst scenario for quick sort is $On^2, because it's an in\text{-}place algorithm it may be because the pivot choice that is made.$

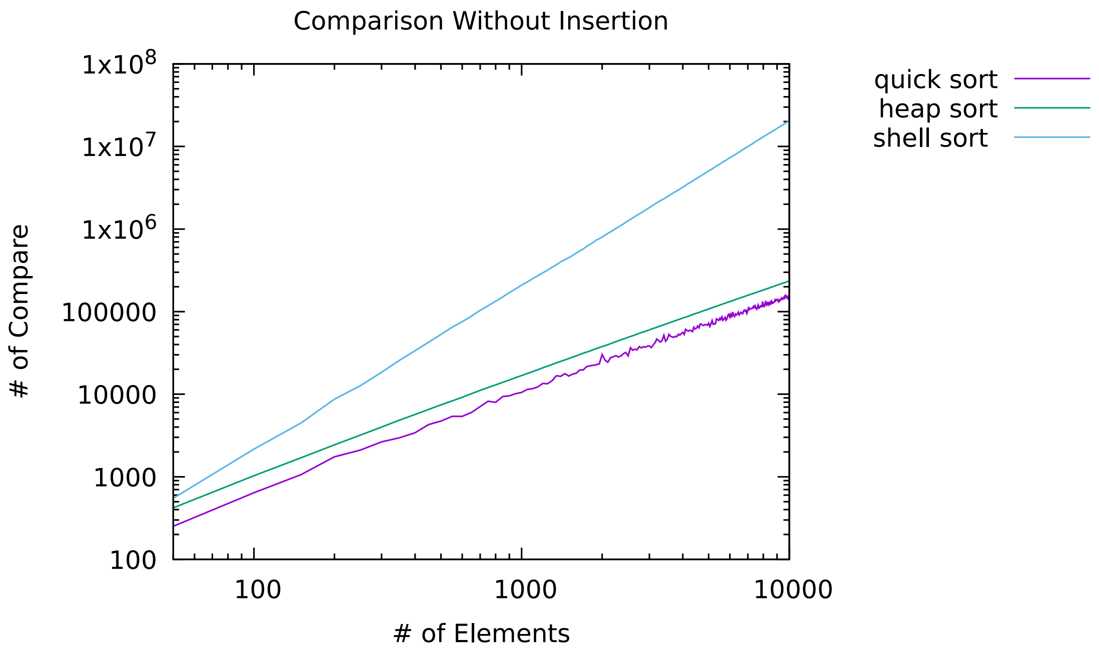
1.5 Graphics from GNU plot

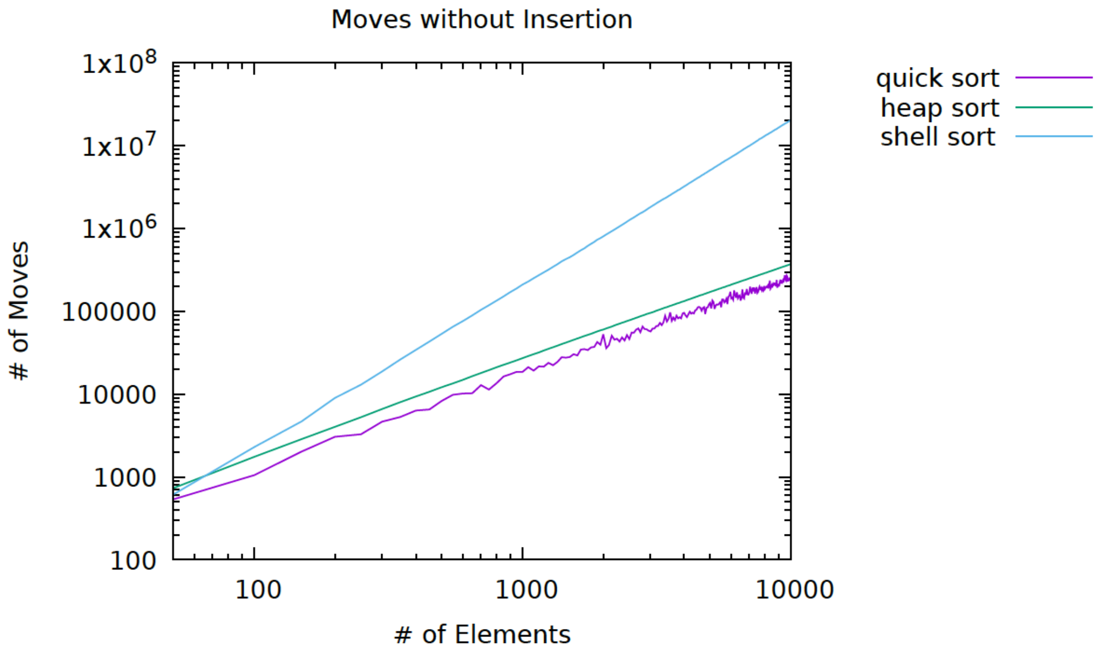


Just by analysis, (not very looking much into shell sort since it is very off) the shell, insert, and heap sort are pretty linear and there is not much pivoting happening. But, quick sort is very strange, it's altering a lot, I believe it's because of the partitioning strategy quick sort has.Quick sort also have the lowest moves.



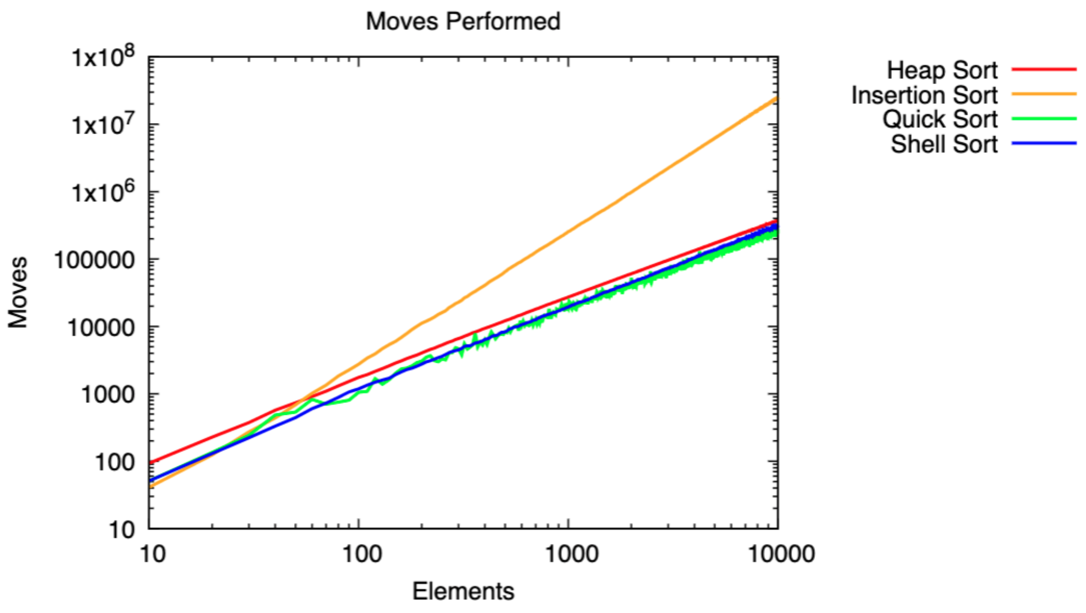
Comparisons with the elements are similar as well. The quick sort's altering begin to be evident when it's almost at the end. Quick sort seems to have the lowest comparisons.





These are graphs without insertion. These three – shell, heap, and quick sorts, has the least moves and comparisons, unlike insertion sort. Unfortunately for my shell sort, this is suppose to draw attention to the compares and moves can

go up to 1 x 10^6
(I'm referencing off of the graph provided by the assignment by Professor Long.)



This is part of the assignment WRITEUP example provided by Professor Long.

1.6 Conclusion

In conclusion, this lab taught me the difference of the four sorting algorithms and the importance of time complexity. In the process of creating my program for all four, I have learned more about pointers, arrays, and functions. This was also a small introduction to memory and allocation of memory.