Nayeel Imtiaz
Professor Long
CSE 13s
10 October 2021

Write Up for Project 3 - “Sorting”

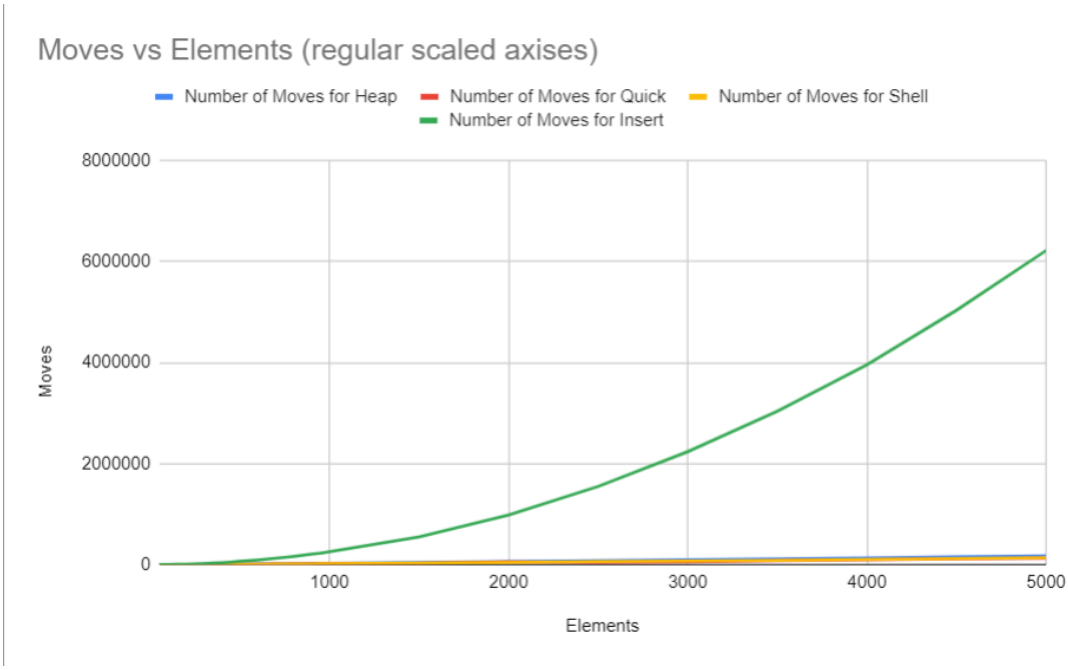
Introduction: The purpose of this lab is to implement different sorting algorithms from the command line. There are four different sorting algorithms to be implemented for this lab which are shell sort, insert sort, quick sort, and heap sort. The final executable will be only sorting a dynamically sized array of unsigned integers using the algorithms specified by the user from the command line. This program will also be able to keep track of statistics such as the number of moves, comparisons, and elements in the array.

Discussion: For any type of sorting algorithm, there are times when it is more efficient at sorting the array. For example, insertion sort has O(n) time complexity when the algorithm is given and already sorted array, but it has an O(n^2) time complexity when the algorithm is given an array in reverse order. Insertion sort is really easy to implement, but it is one of the slowest algorithms on average because its average time complexity is O(n^2). Heap sort, on the other hand, is a much better algorithm to use on average as its average time complexity is O(nlog(n)). Quick sort is another really good algorithm to use regularly as it has a time complexity of O(nlog(n)).

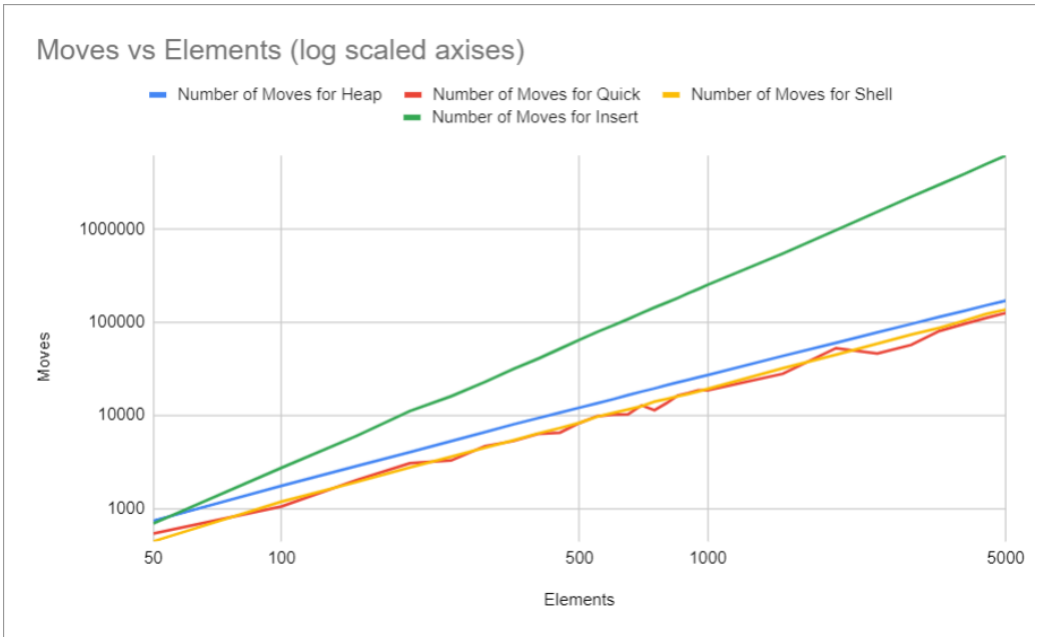
I experimented with the different algorithms quite a bit. I found out I was able to sort an array of 10 million elements using quick sort with no problem. It was able to sort the algorithm almost instantly. On the other hand, insert sort had problems, and it took a while for it to finish sorting. Once again, this enforces how the insert sorting algorithm is not the best algorithm to use when sorting most arrays. It is only quicker in a handful of scenarios (for example when the array is already sorted). Insertion sort was the algorithm I had the easiest time to understand. But after doing this lab, I realized how inefficient of an algorithm it is and one should use other sorting algorithms like heap sort and quick sort over insertion sort.

Results:

<u>Moves vs. Elements Analysis</u>



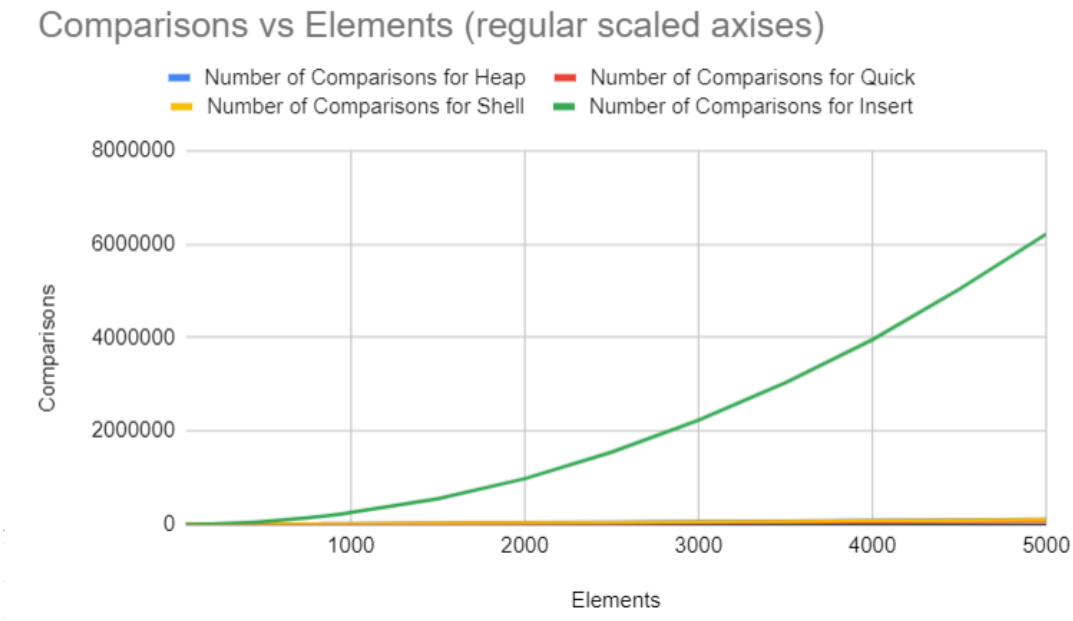
As it can be seen from the first linear scale graph, insertion sort makes a parabolic shape while all the other sorting algorithms make almost a horizontal line. This is why insert sort has a time complexity of O(n^2). Heap sort, shell sort, and quick sort all appear to be close to a horizontal line complexity, suggesting that it is near linear time complexity. To be more precise, these three algorithms average at O(nlogn) time complexity.



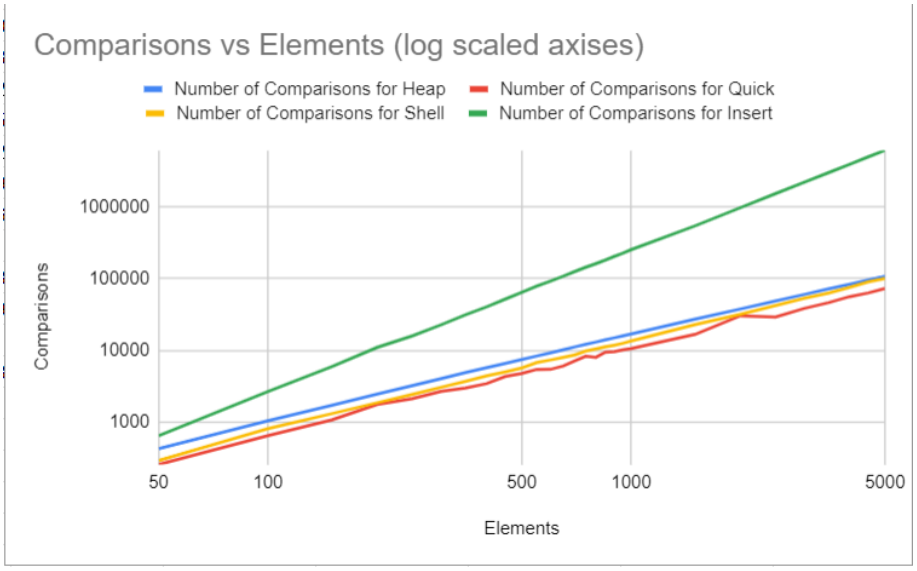
The log scaled graph for moves vs elements also shows that shell sort, quick sort, and heap sort lines change at a slower rate than the insert sort line. The swirly lines for the quick sort and

shell sort lines suggest experimental errors. The lines on the log scaled graph should be 100% linear.

Comparisons vs. Elements Analysis



The linear scaled graph for comparisons vs elements also shows that insert sort gets slower exponentially by the number of elements. Not much can be observed about the other 3 algorithms as they all look almost the same in this graph (much more efficient than insert sort). It is imperative to observe the other 3 sorting algorithms on a log scaled graph.



In the log scaled graph, it can be seen that the shell sort is slightly faster than the heap sort algorithm and the quick sort is a little faster than the shell sort algorithm. Once again the swirly lines on the graph suggest some degree of experimental deviation. Time complex functions such

as O(n^2) and O(nlogn) are only theoretical speeds. They do not necessarily model real algorithm speeds perfectly, but they are pretty good estimations still.

Conclusion:

In conclusion, different sorting algorithms are quicker than others. Quick sort appears to be the fastest sorting algorithm to use while insert sort is the slowest one by far. Heap sort and shell sort fall close behind quick sort, but insert sort is nowhere near the other three when it comes to speed.