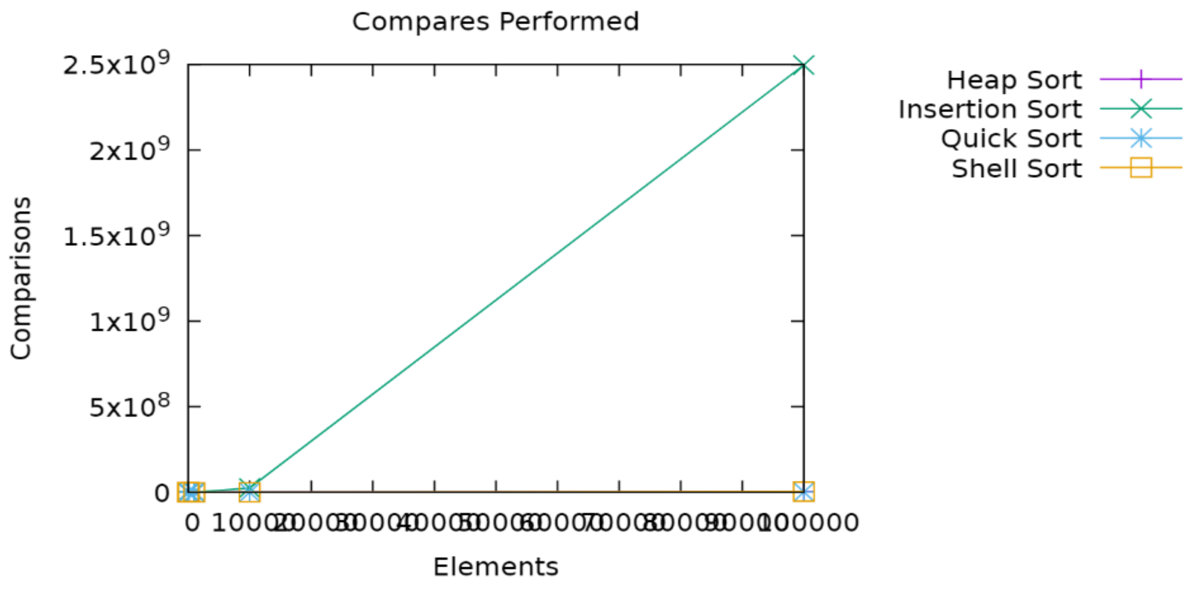
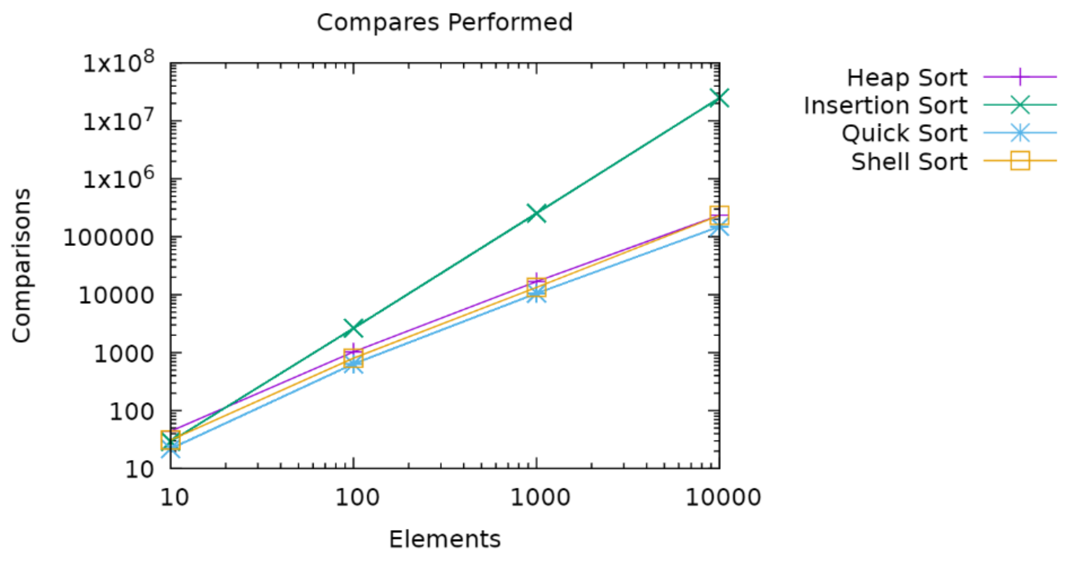
Kelly Liu
Professor Long
October 17, 2021
CSE 13S

Write Up Assignment 3
Sorting: Putting your affairs in order

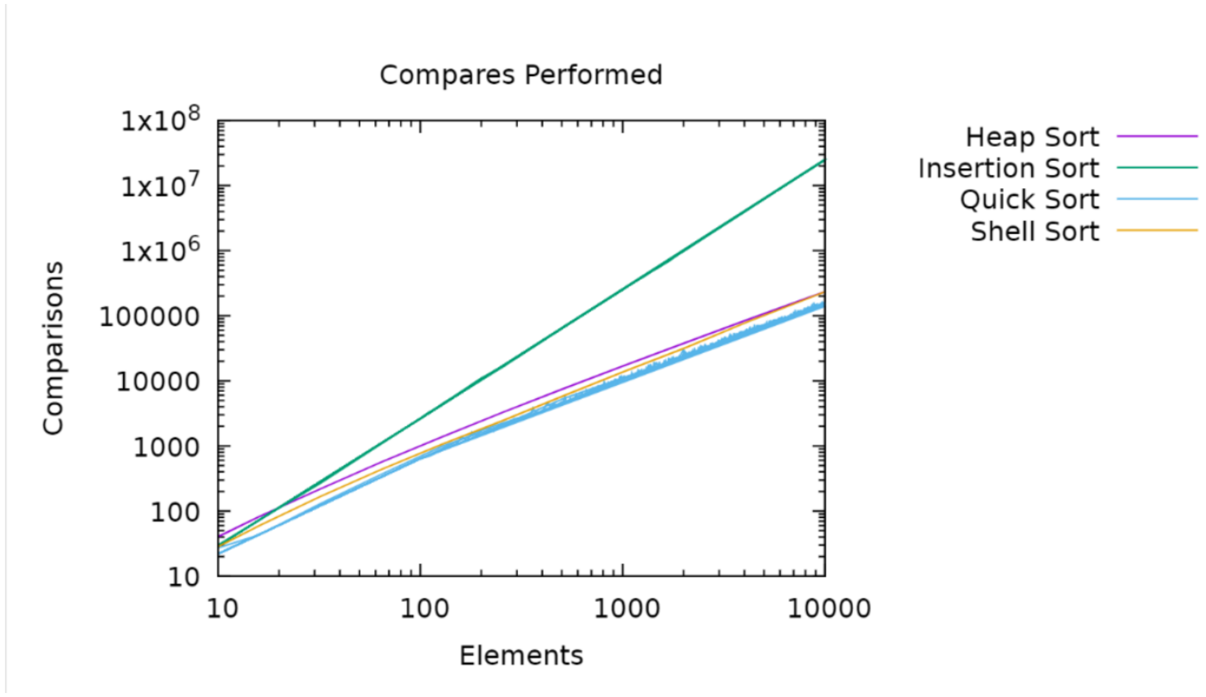
Graphing Analysis:



As we can clearly see from the graph, given a very large element size- insertion sort becomes super inefficient. We are comparing too many elements when using quick sort, meaning our speed will be drastically bigger than other sorting methods.The main point of this particular graph was to show the huge difference between insertion sort and the other three methods.

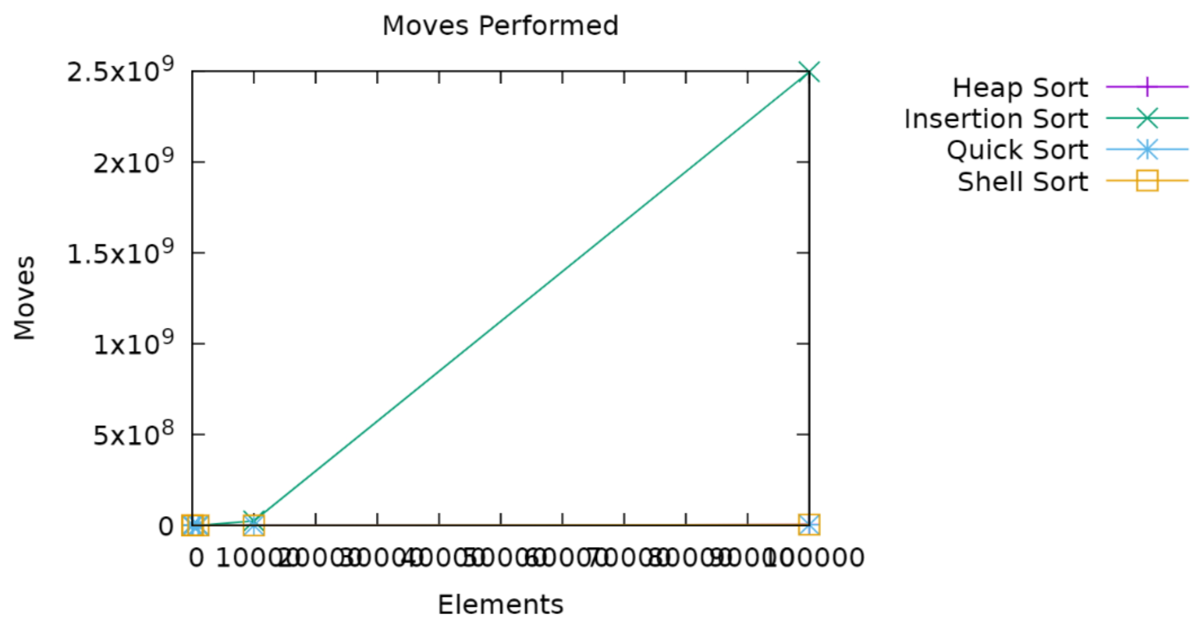


(4 points plotted above)

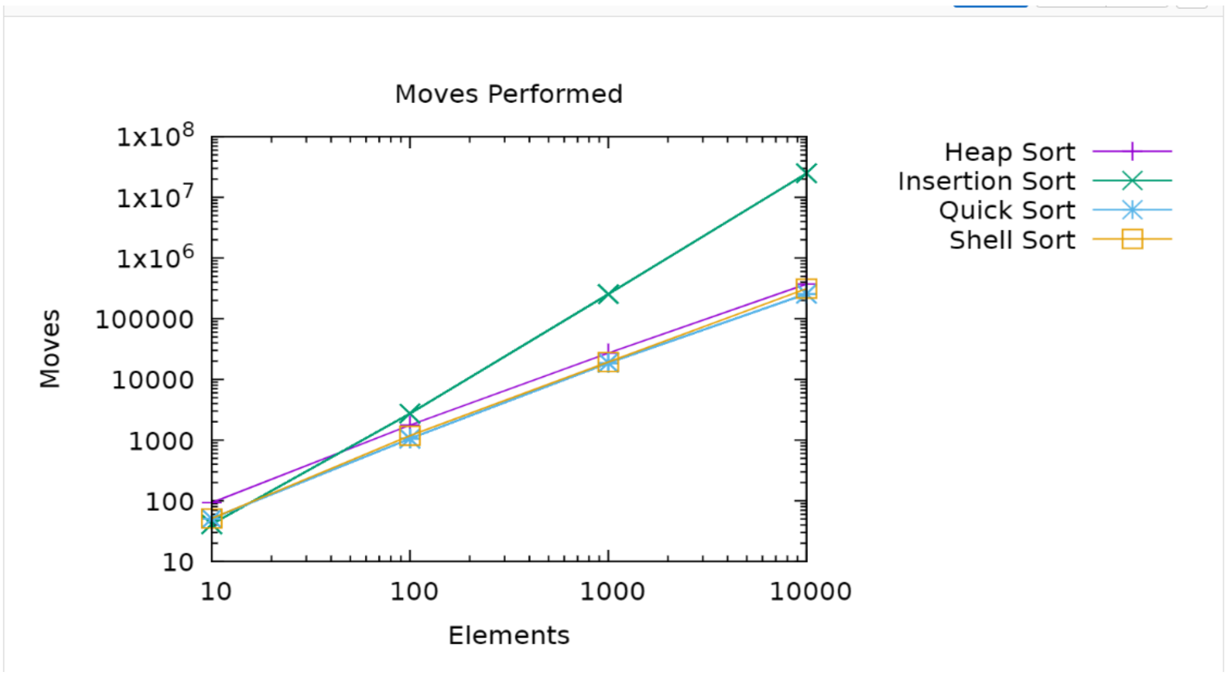


(17 points plotted above)

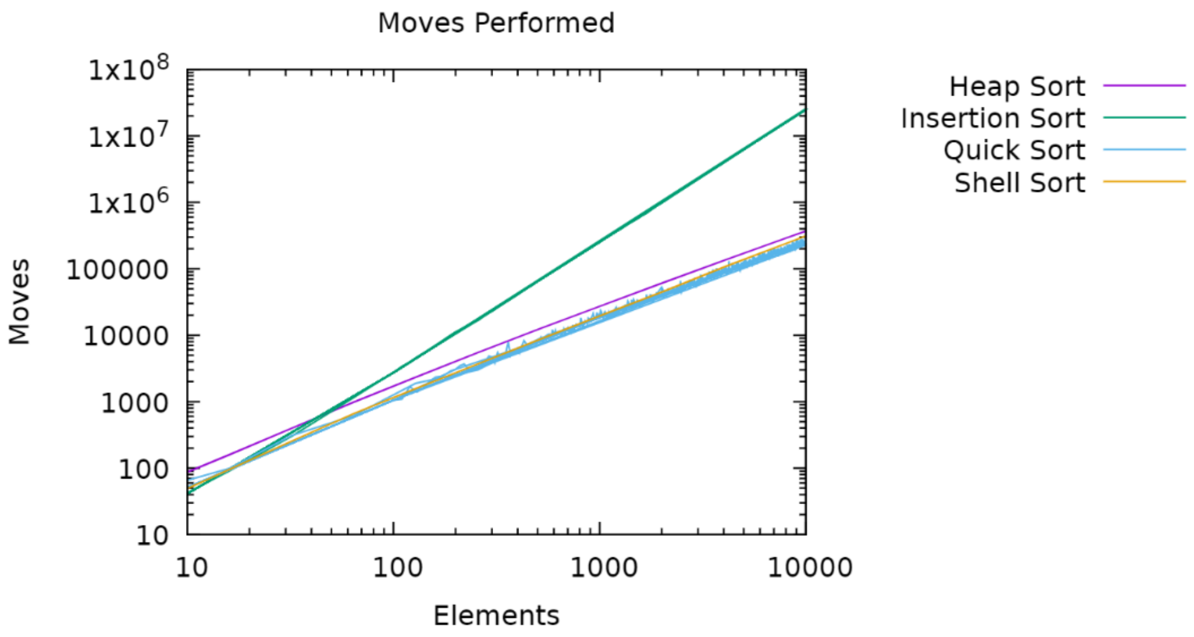
Now we look at a smaller element size so we can compare all four sorting functions. We have already determined that insertion sort is widely inefficient when there are alot of elements. With this graph we can roughly estimate that heap sort is the most inefficient compared to shell and quick sort even though it’s not by much. We can also tell that heap sort isn’t necessarily the best method to pick ever because it is even more inefficient than insertion sort until around maybe 200 elements.The most efficient sort is shown as quick sort as from 10 to 10000 elements, quick sort has the least amount of comparisons needed to sort an unordered array. However, the efficiency of quick sort is comparable to shell sort.



Similarly to comparing the amount of comparisons, we can still see that insertion sort needs the most moves to sort an unordered array when there are alot of elements in the array. Consequently, we cannot tell anything much about the other three functions as there is a drastic difference between efficiency of insertion sort and the rest.



(4 points plotted above)



(17 points plotted above)

Again, we now look at a graph with a lowered element size to gauge the differences between all sorting functions.The previous conclusions that we made in our compares graph holds up with this graph. We can still see that insertion sort is the most inefficient after around 50 elements and that heap sort is more inefficient than quick and shell. The main difference between this graph and the compares graph, is the fact that this shows a closer efficiency between quick and shell as we can see that the two lines are neck and neck.

Statistical Analysis
(100 moves):

Based on statistical analysis, we can compare the efficiency of our different sorting functions. We can see that insertion has 2741 moves which give or take double the amount of moves than the other three methods of 1755(heap), 1183(shell), and 1053(quick). Heap sort, as we can predict from the graphs, has the second highest number of moves at 1755- 572 more moves than shell sort and 702 more moves than quick sort. Quick sort is more efficient than shell sort, having 130 less moves than shell sort. This is also backed up by the amount of comparisons that our function has to do. Insertion sort being drastically bigger than the other three, and then heap having 228 more than shell and 389 more than quick sort. Quick sort is also quicker than shell sort again by having 161 less compares.

(10,000 moves):

We can use a higher element size to check if our results hold true (which they do). Based on the statistical evidence provided here, we can see that the insertion sort has 24,901,706 moves compared to 256,734(quick), 372,558(heap), and 313,890 (shell). I'm not even going to bother subtracting the amount of moves between insertion sort and the rest because intuition can be used to see based on the statistical evidence that insertion sort is simply inefficient. Again, we can tell that heap has more moves than both quick and shell by 115,824(heap compared to quick) and 58,668 (heap compared to shell). Quick and shell being the closest but quick still being more efficient by 57,156 moves. The amount of compares between the four sorting methods goes similarly along with the amount of moves. Insertion having 24,891,699 compares, quick having 149,913 compares, heap having 235,318 compares, and shell having 234,418 compares. Even though heap is relatively close in compares as shell, we can still conclude that heap is slower based on the amount of moves calculated above. The same results are given here in that insertion has way more compares than other sorting functions, and heap having a bit more than both quick and shell then quick being a bit faster than shell.

Conclusion:

In both the graph and statistical analysis, they both give the same conclusions over and over again regardless of the amount of elements in the array. The graph tells us that based on preliminary findings, most efficient to least efficient goes from quick sort to shell sort to heap sort to insertion sort. This was then accurately backed up by numerical values on how many compares and moves each particular sorting method has given a small and large element size. My findings include the fact that insertion should never be used because it is just way too inefficient having to do way more moves and compares in order to sort an unordered array. Heap, while not as slow as insertion (we do note that heap is more inefficient than insertion for a small amount of element size), is still not comparable to quick and shell. While shell and quick are indeed comparable, we can see that in higher element size, the difference of 57,156 moves and 84,505 compares is still a big difference.