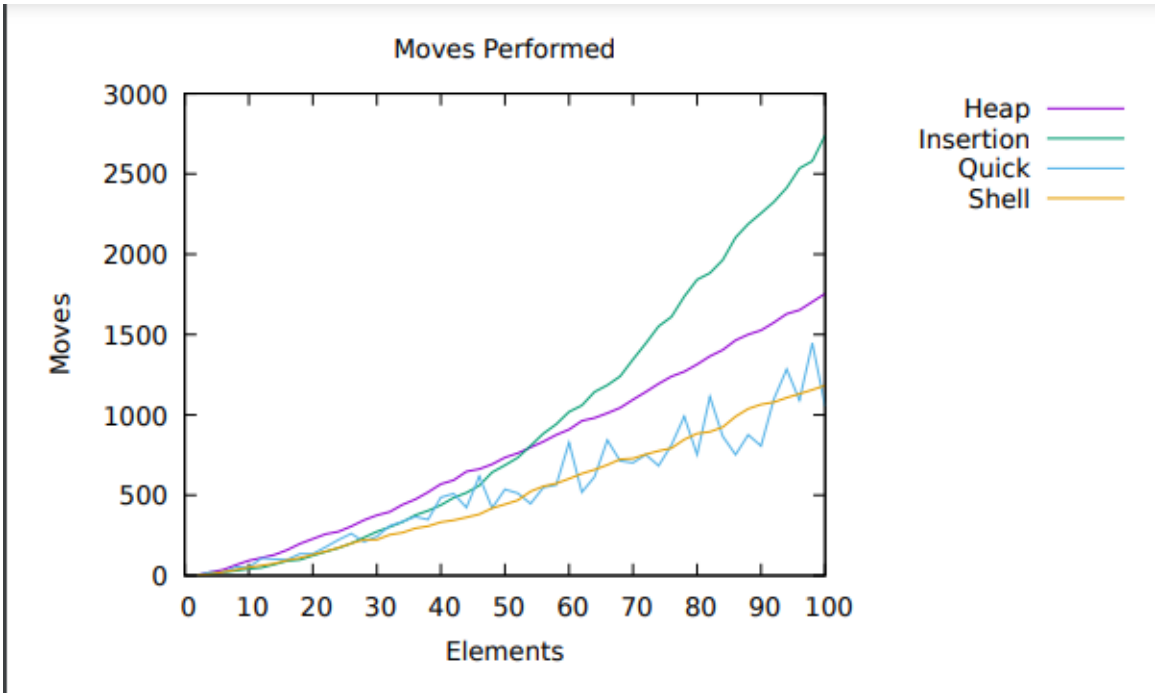
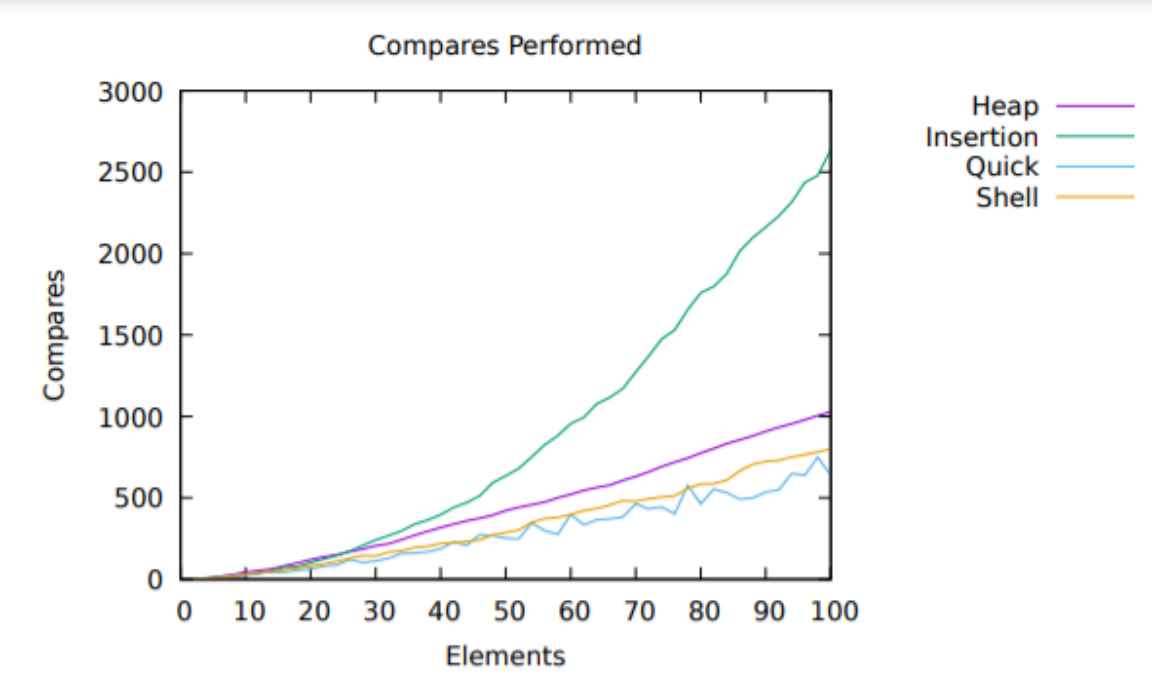
Assignment 3 WRITEUP.pdf



This graph represents each sort and its amount of moves that it takes the more elements that it is given. In analyzing the graphs we can come to the conclusion that for Insertion sort takes the longest the elements it's given and takes progressively longer. We can also observe that although initially Heap sort takes the longest but then progressively gets more efficient with less moves. This is due to the way that Heap sort uses a tree like architecture of finding the max value then excluding that value then finding the next max value and sorting in such an order. Quick sort's line fluctuates greatly due to the way the function is written. We know that it is a recursively built function that sorts from both ends of the array and sorts them. Due to this if the array is built in a way such that the max is in the middle element then quick can take much longer and the moves can fluctuate. Also because the quick sort function utilizes swaps mostly that shows between swaps the moves count will grow much quicker. Lastly we see that Shell is on average the most consistent and its efficiency is on par with quick sort. This can be attributed to to the fact that Shell generates gaps and can compare elements over gaps in the array such that it is much faster than checking with the neighboring elements.



We can see that since insertion sort compares the elements with the one in front it may go through all of the elements of the array and will have comparisons with them all. For Heap sort it is significantly less comparisons than insertion sort however this is due to the tree like structure of heap that will ultimately have less comparisons because the comparison will go on betweens the nodes of the tree rather than all of the elements. Shell sort goes through the array of elements using a gap function so that there will be overall less comparisons due to the sort structure able to compare itself more efficiently and effectively by making longer comparisons across the array. This way of sorting is more dynamic and versatile and since as the gaps grow smaller there will be less comparisons. Lastly quicksort uses the least amount of comparisons because it is making comparisons between partitions making the comparisons the most efficient.

I was able to learn from these sorting algorithms how similar to memory allocations these arrays have certain runtimes and efficiency that will take up lots of memory when dealing with bigger numbers. It shows the importance of efficiency and speed and how that type of coding is most applicable in the real world. It allows us to realize that although the most simple sorting algorithms insertions is simple to grasp in terms of logic we can see the value that helper functions, generators and recursive code can bring.