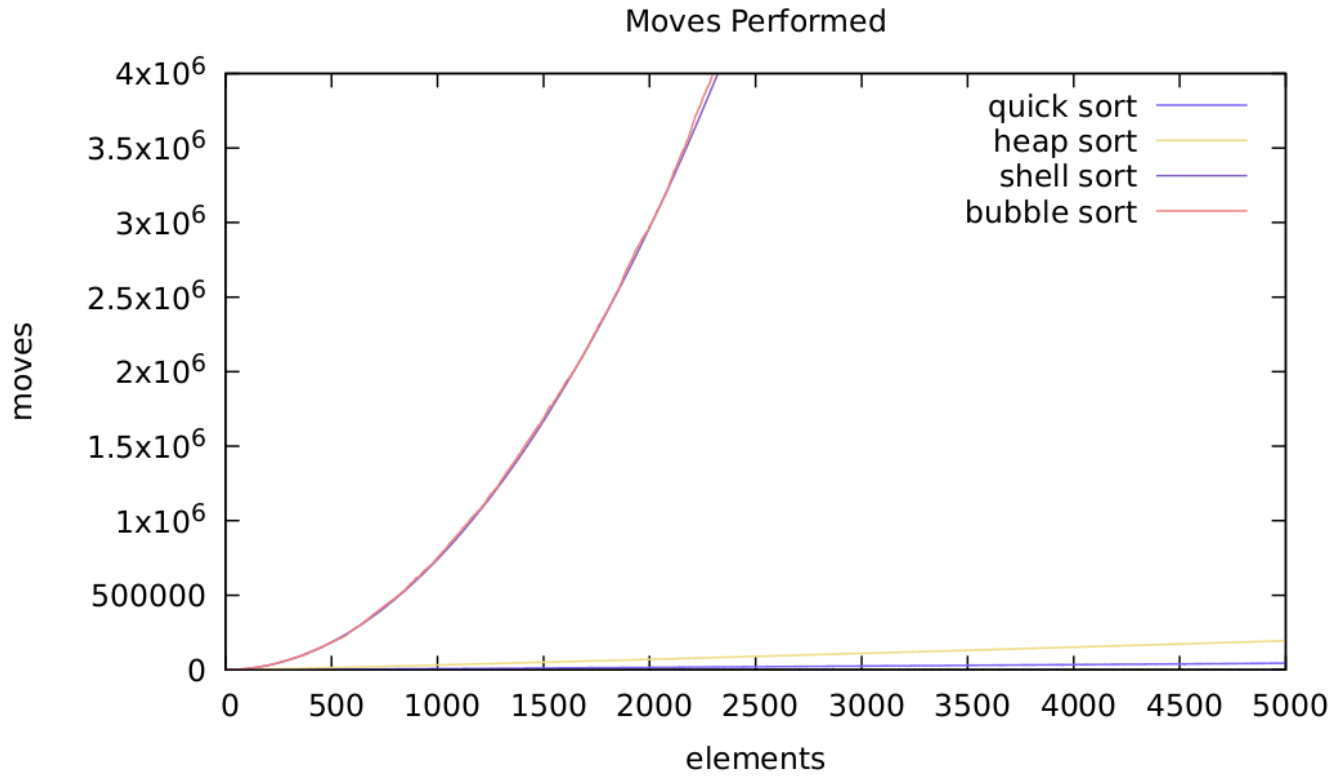
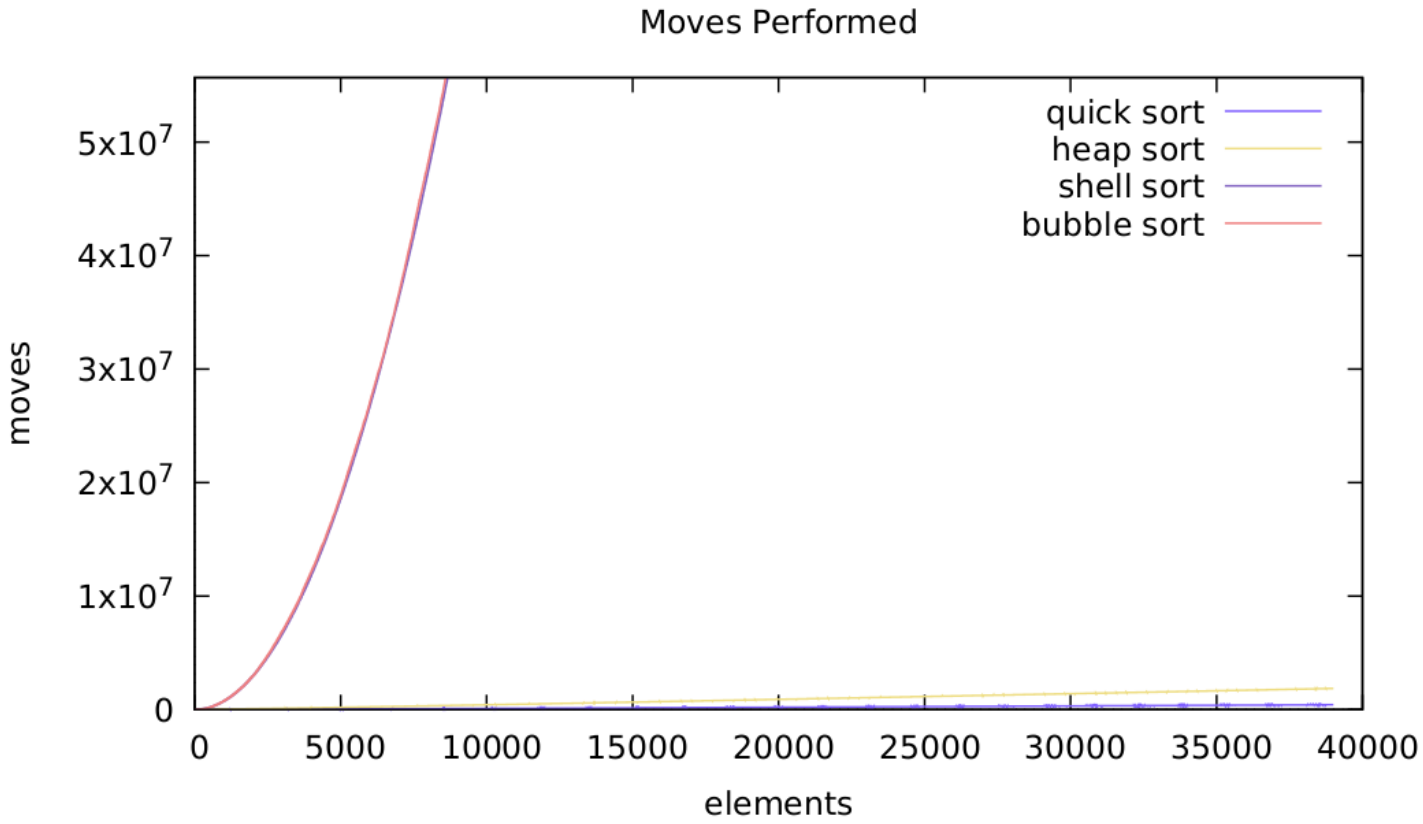
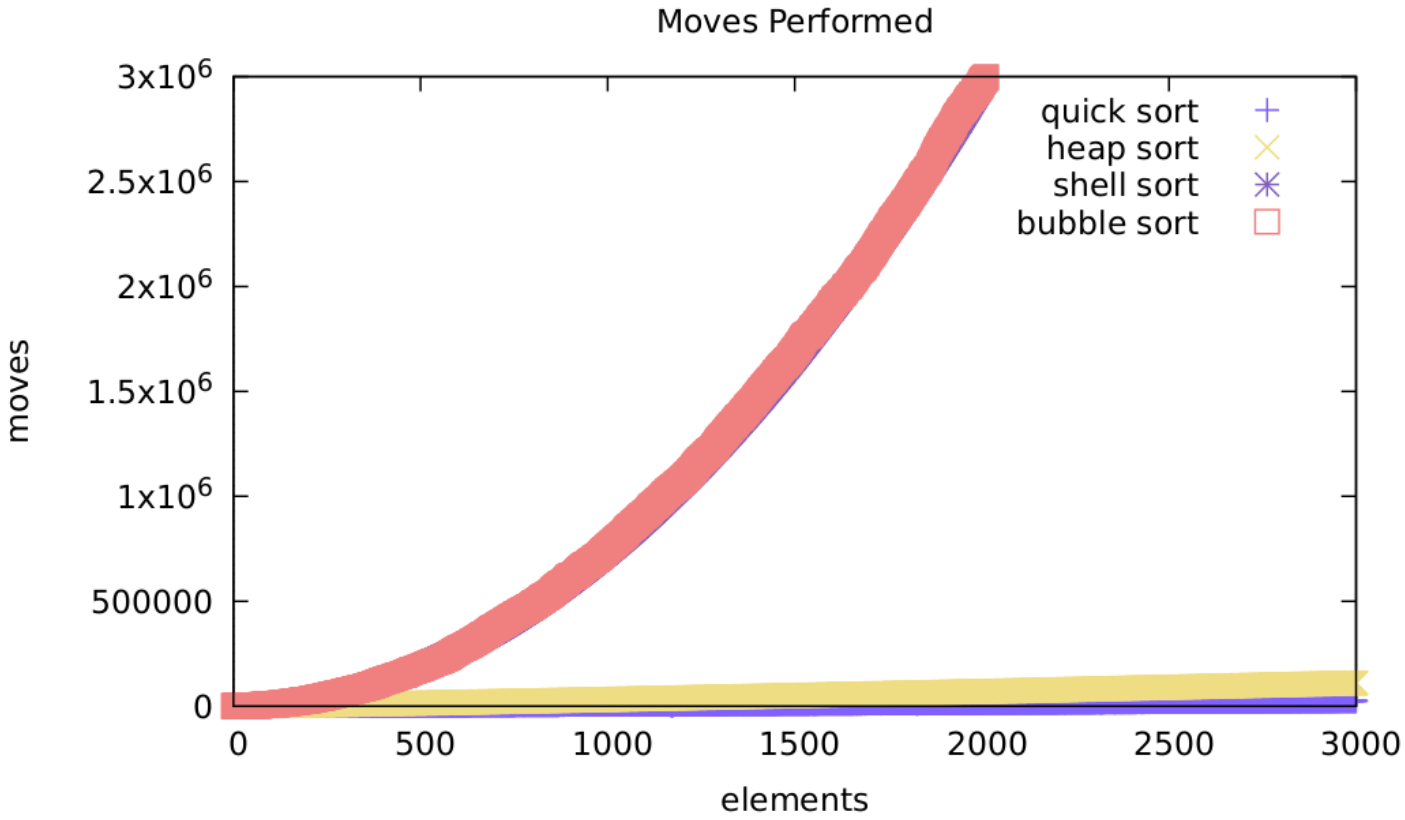


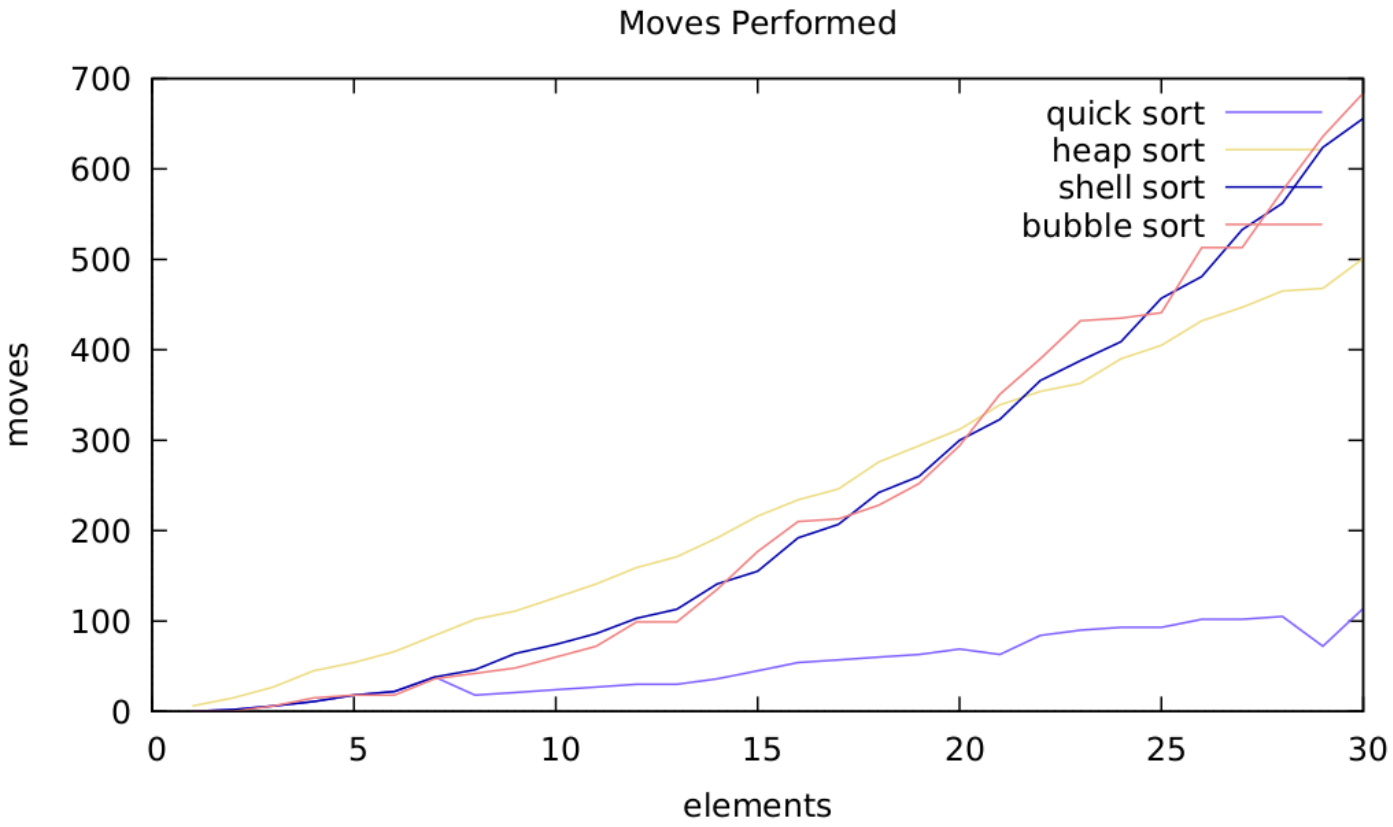
It seemed that quick sort was by far the fastest sort compared to the others as the amount of elements grew. In second place was heap sort. And though it was close, bubble sort had the worst time complexity than the others. The amount of moves as the amount of elements grew increased significantly how many moves bubble sort would have to do to sort things out. Even when doing the bash script to output the data, bubble sort took the longest to finish outputting.



When there's more elements, we can see this pattern of time complexity continue on. Quick sort can be seen at the very bottom below heap sort's time complexity. We can see that the lines are not smooth, which means it's not exactly linear for the increasing amount of elements. This is the same with heap sort, but it's less noticeable. I should note that I had to end the program early for both bubble sort and shell sort because it took so long to produce the results. As the amount of elements grows, the time it takes for bubble sort and shell sort to output the results takes extremely long. This coincides with their time complexity. Quick sort and heap sort did not take that long to produce their results.



When I reverse the array, I get this graph. It's still very similar in nature to the graphs above. This concludes that no matter in what way the arrays are, they will have the same time complexity as they are supposed to for their sorting algorithms. The reasoning for this is because the sorting algorithms will still go through the same processes that it did previously, no matter if they're reversed or the numbers are mixed around.



When there's less elements to sort, bubble, quick, and shell are pretty to each other in terms of best time complexity. Only heap sort seems to be worse when there's only a couple values in the array. However, as the amount grows out, its line will start going below shell sort and bubble sort. Quick sort seems to be, as the name implies, quick throughout and consistently the different number of elements. Heap sort and quick sort are best for a long amount of numbers because their algorithms account for that and work around it. Bubble sort and shell sort remain the same no matter the amount, which causes problems when there's a lot of elements.

I used this link to help me understand structures in C more.
https://www.tutorialspoint.com/cprogramming/c_structures.htm

I used this link to help me understand more about masking and how to do it.
https://learn.parallax.com/tutorials/language/propeller-c/propeller-c-simple-protocols/spi-example/bit-masks-better-code

I used this link to teach me about `puts`, which is a very helpful tool for debugging (so I don't always have to do `printf`).
https://www.tutorialspoint.com/c_standard_library/c_function_puts.htm

I would like to mention that my other class that I'm currently taking, CSE 30, also goes through different sorting algorithms in Python. I used that knowledge from that class to help me understand the quick sort algorithm in C that I have created. I have linked the Google Colab that my professor uses to teach Python lessons.
https://colab.research.google.com/drive/1jfk6wrSwB9m7fOYWbtv4MUIqEoDvRd0f?usp=sharing#scrollTo=5GLWg7vlSREC

I used this to help me learn about `opterr` and how I use it for this assignment. I ended up not using it, but I shall still cite it here.
https://www.gnu.org/software/gawk/manual/html_node/Getopt-Function.html