Nicolas Corfmat
CSE13S, Winter 2023
ncorfmat@ucsc.edu

Putting Your Affairs in Order

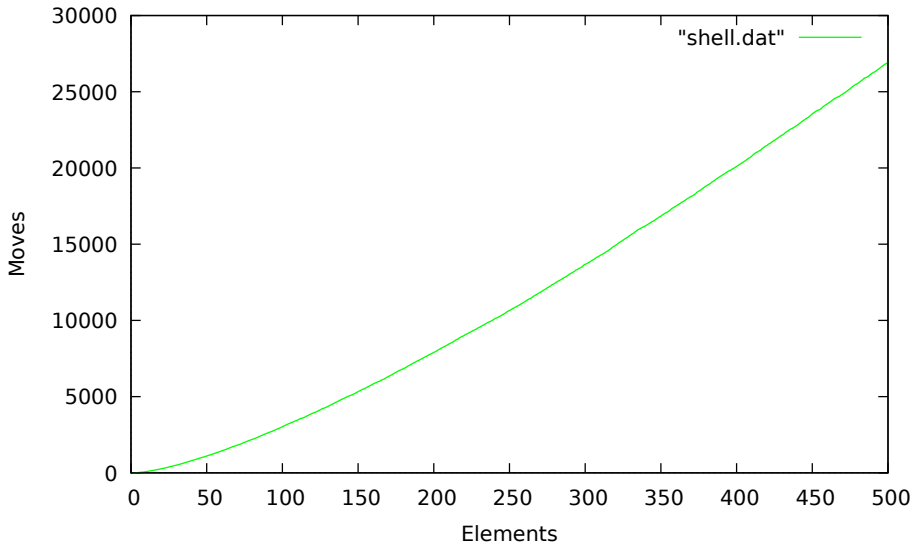
Assignment 3: Sorting

1 Description of Program

This program sorts arrays of pseudo-random numbers using the following four algorithms: Shell Sort, Heap Sort, Batcher Sort, and Quicksort. The program simultaneously tracks the number of moves, compares, and swaps performed by each sorting algorithm, and conveniently plots this information on a graph. The user can then independently compare differences between each sorting algorithm.

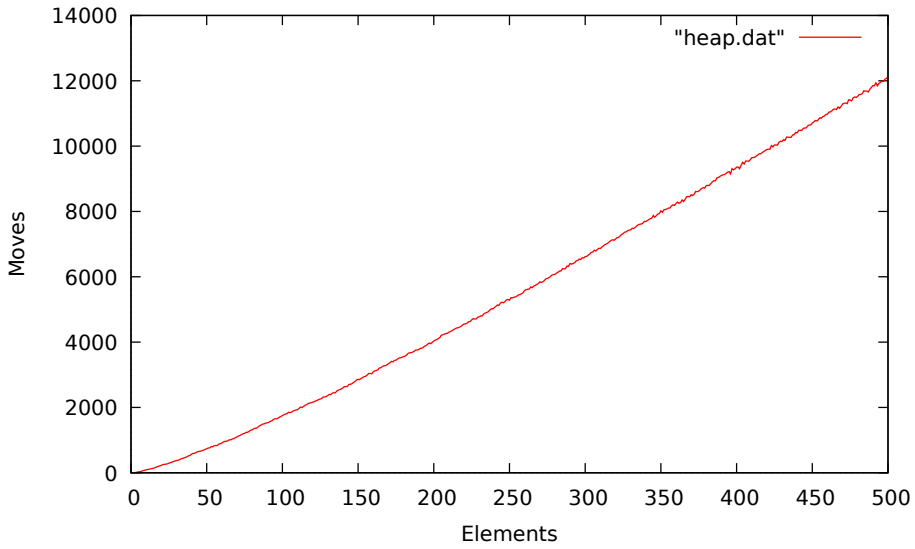
2 Shell Sort

Observing the Shell Sort plot, it is apparent that the relationship between elements in the array and moves performed is very slightly curved, almost like a positive quadratic function. This leads us to believe that in the long-run, the rate of number of moves-per-element would increase as the array size gets larger, however, for moderately-sized arrays, we can assume an almost linear relationship. In regards to the volume of moves performed compared to elements in the array, the current rate can be said to be on the higher end, deeming this approach not so efficient at sorting.



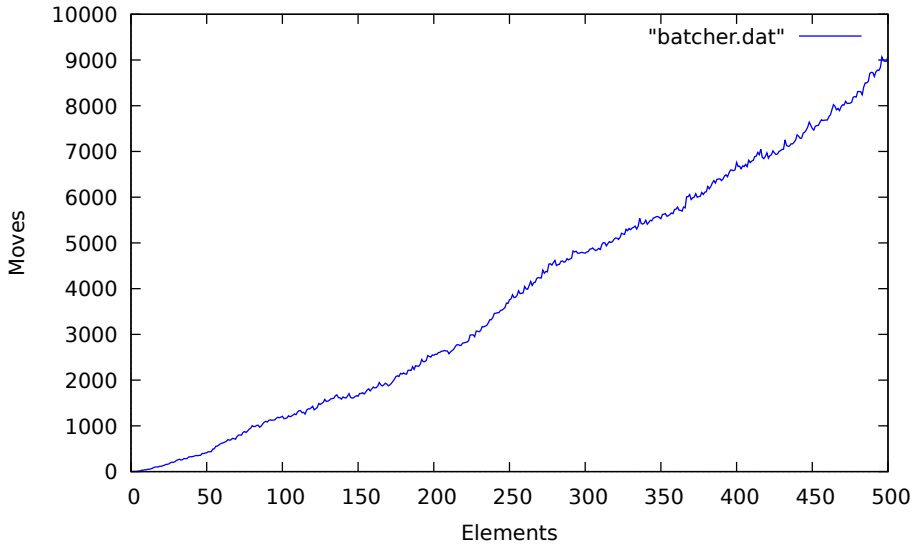
3 Heap Sort

Upon first glance, Heap Sort resembles more-so a linear relationship compared to Shell Sort. Additionally, we can denote that the number of moves performed at each element is significantly less than the previous sorting algorithm (roughly twice as fast). One way to tell whether a sorting algorithm is efficient, is by determining the rate of moves per element. The lower the rate, the less moves the program has to run and therefore, the faster it can sort. In the case of Heap Sort, it has proven itself to be a more effective algorithm compared to Shell Sort.



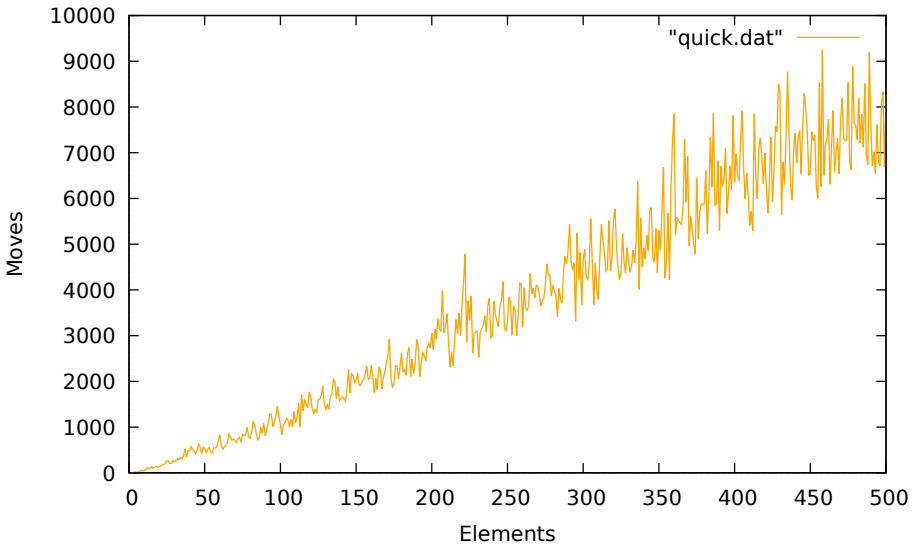
4 Batcher Sort

One of the more fluctuous algorithms, Batcher Sort orders elements in an array at a more inconsistent rate. The relationship between moves performed and array size can be described as rather linear, although it is hard to say for sure. Perhaps as the number of elements increases, a more concrete relationship will surface. Comparing it to Shell Sort and Heap Sort, however, it is evident that the moves-per-element rate is smaller than the previous two algorithms, making this one faster and more effective at sorting.



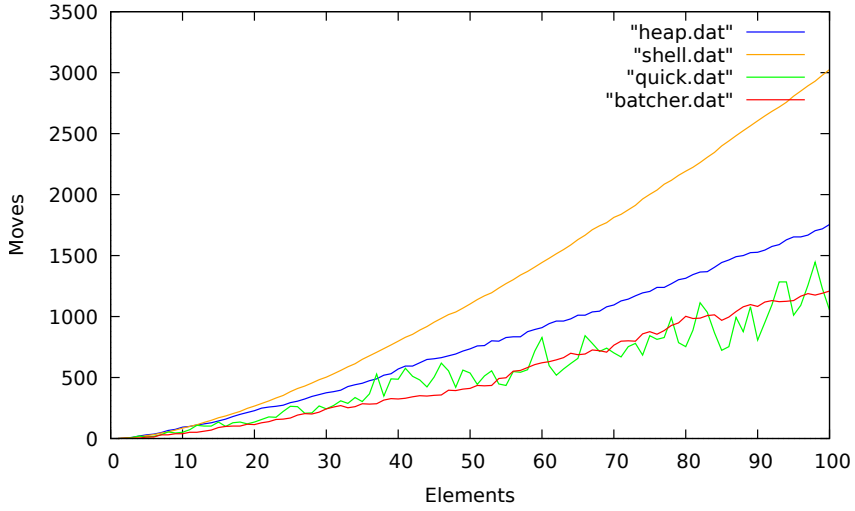
5 Quicksort

Unlike any of the previous three sorting algorithms, Quicksort is by far the most unique. To put it simply, it appears that as the number of elements increases, the fluctuation rate of moves performed also increases—a rather inconsistent pattern when it comes to sorting. Despite this chaotic trend, on average, it does not perform much moves compared to Shell Sort, Heap Sort, and Batcher Sort. In fact, the rate of moves per element is roughly identical to that seen for Batcher Sort, making this algorithm relatively effective at sorting arrays of the sizes shown on the plot.



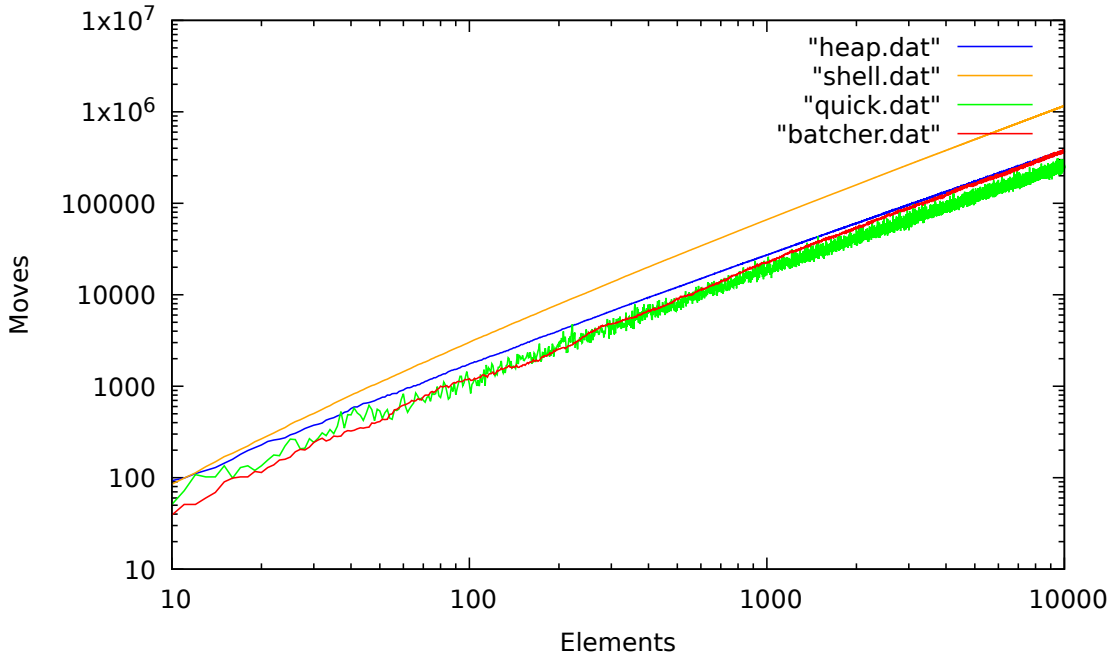
6 Sorting With Minimal Elements

Before making conclusions about each sorting algorithm's ability to efficiently sort an array of pseudo-random elements, we must first recognize that certain algorithms perform better for arrays of different sizes. As a result, we must analyze the performances of each sorting approach under varying conditions—in this case, arrays with no more than 100 elements. According to the plot below, we can identify which algorithms require more moves by searching for the ones that lie highest on the graph and have the largest slope. Unsurprisingly, Shell and Heap Sort tend to be less effective at sorting compared to Batcher Sort and Quicksort, whose graphs are increasing at a much slower rate.



7 Sorting Large Number of Elements

Let us now get a broader perspective on the performances of each sorting algorithm. Although each graph may appear linear, this is in fact caused by the domain and range being log-scaled. In reality, as the number of elements increases, the rate of moves-per-element also grows. Previously deemed “similar” in sorting performance, Batcher Sort and Quicksort eventually begin drifting away from each other, with Quicksort requiring less moves in the long-run.



8 Conclusion

By studying the ability of each algorithm to sort arrays of pseudo-random numbers, it became clear that each algorithm performs well the less elements they have to sort through, as proven by the exponential tendencies of each graph. Consequently, sorting algorithms perform poorly the more elements they have to sort, as this calls for nested loops to loop more, yielding exponentially larger run-times. In the case of the four sorting algorithms analyzed in this experiment, Batcher Sort and Quicksort are most efficient at sorting, with the latter being more effective for larger arrays.

9 Credits

- All four sorting algorithms were derived from python code provided by Darrell Long. Although deviations from his code exist in the provided pseudo code, most of it (especially the structure) remains the same.
- Code for calculating log was provided by Darrell Long through Discord. The only changes made to this were swapping certain data types in order to fit with the structure of the program.