ANALYSIS OF SORTS

A Brief Overlook On Various Sort Algorithms

Demis Mota

Regis University
August 14, 2020

Analysis

A runtime analysis on various sorting methods was conducted. Chart 1 shows the average runtimes vs number of data elements to be sorted in logarithmic scale. These averages were taken from ten tests ran on each type of sorting algorithm. As can be seen as the amount of data to be sorted increases so does the time it takes to sort the data. I didn't find any major surprises in the code except for initial runtime seemed to be skewed due to initial compilation and the Java Virtual Machine's initial load time. One can see as the tests continued the times tapered off and gave a more consistent trend. The runtimes for each of the sorts did not really surprise me as the Big-O values for each tended to coincide with the results that were obtained.

Bubble Sort Analysis

This type of sort was the slowest amongst all the sorts. This sort tends to $O(n^2)$ with larger amounts of data taking way longer. This type of sort appears to be ok to run for small amounts of data. The implementation of this sort is rather simple and only requires two for loops.

Selection Sort Analysis

The second slowest algorithm in the bunch runs surprisingly fast for small data sets. A large data set slowed it's time at a rate of $O(n^2)$. This sort requires a couple of loops to implement. Like Bubble Sort it is rather simple to implement. It runs faster than bubble sort but slower than insertion sort.

Insertion Sort Analysis

The third slowest sort and ran extremely fast for small amounts of data. This sort also tends to $O(n^2)$ and when the data is mostly sorted can run at O(n) time depending on implementation. This too is a simple algorithm and takes an element and places it in the right slot. This algorithm is faster than bubble and selection sorts, but not by much.

Shell Sort Analysis

This algorithm is a bit more challenging to implement but has a varied run time dependent on the size of the data and how sorted the data is. This algorithm at it's worse runs at $O(n^2)$ but at it's best it runs at O(nlog(n)).

Merge, Quick, and Java Sort Analysis

This family of algorithms are considered divide-and-conquer algorithms as you recursively shrink your array, and once small enough you begin to build your array from the pieces. In doing so the sort runs at O(nlog(n)) and only speeds up with implementation variations. The java.util.arrays sort() method uses a combination of merge sort and tim sort, it also promises O(nlogn) performance in all cases (Arrays Java Platform, 2020).

Conclusion

I found no discrepancies in any of the sort algorithms or any weird tendencies from various test data, except for the initial sort which seemed to always be an anomaly due to Java's Virtual Machine getting loaded into memory. Java's in house algorithm was the fastest with quick sort coming in at a close second. The simplest to use is Java's sort method provided in the java.util.arrays library. Frankly, most programming languages come with it's own sort method

and can reduce the difficulty in implementing a sort method and also usually provides the most efficient time complexity.

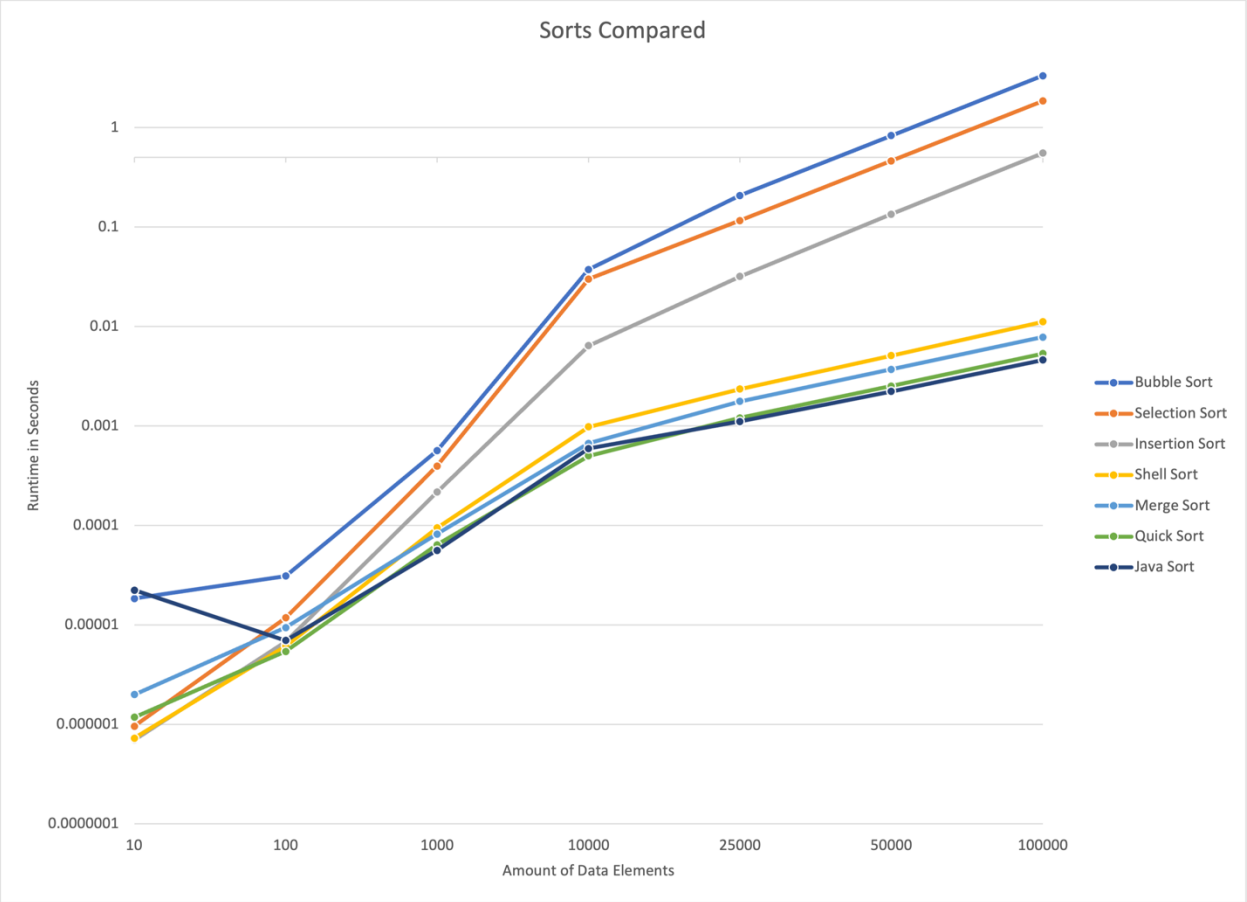

Chart 1

References

Arrays (java platform SE 7). (2020, June 24). Retrieved August 13, 2022, from https://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html#sort(java.lang.Object[])