


Tyler Mong
CSCI-3101-001 - Data Structures & Algorithms I
Professor Hoy
12 December 2023

Findings:  [CSCI-3101-001] Final Project: Sorting Algorithm Run Times
(Graphs for all algorithms at a given number of elements appear on the right. Graphs for each algorithm at all numbers of elements appear on the bottom.)

From the seven different algorithms tested (bubble, selection, insertion, merge, shell, quick, and heap), the efficiency of each algorithm becomes apparent, especially with larger datasets.

Starting off, Bubble Sort performed the worst by far. With a relatively small dataset of 10,000 elements, its performance seemed fine at first glance, however, when the time was recorded by the computer, it took nearly 200ms, which is far too long for this small of an array. This figure continued to grow as the number of elements increased, taking ~20s for 100,000 elements and ~80s for 200,000 elements. The time Bubble Sort takes is by far its biggest weakness. While there is not much to say about its strengths, the simplicity to understand and implement could be seen as a benefit.

Moving on, Selection Sort also performed quite poorly, however still significantly better than Bubble Sort. Rather than taking 20s and 80s for 100,000 and 200,000 elements respectively, it instead took 4s and 18s. While this is not good by any means, it still triumphs Bubble Sort's performance. While comprehension is subjective, in my opinion, understanding and implementing Bubble Sort was a bit more intuitive compared to Selection Sort, but even with that being said, it is still on the easier side compared to some of the upcoming algorithms.

Next is Insertion Sort. Again while it is not fast by any means, it still beats out the previous two. It took 700ms for the array of 100,000 elements and 3s for the array of 200,000 elements. Understanding Insertion Sort is relatively easy, especially when you make the connection that it is essentially the same way people sort playing cards that they are holding. Despite it beating out the first two algorithms by a pretty significant margin, compared to the upcoming four, using this algorithm is impractical.

Moving forward we have Merge Sort, which took only 13ms for 100,000 elements and 23ms for 200,000. For the array of 200,000 elements that is 130x faster than Insertion Sort, ~800x faster than Selection Sort, and ~3400x faster than Bubble Sort. Merge Sort works recursively and with two methods, where it breaks down each array into smaller sub-arrays, and then merges these sub-arrays into a sorted array. While understanding this algorithm is not too difficult, the code itself is quite lengthy, but regardless of this, its efficiency makes it a worthwhile algorithm.

The fifth algorithm is Shell Sort. The run times for this algorithm were nearly identical to the previous, taking 11ms and 25ms for the 100,000 and 200,000 element arrays. The amount of code for Shell Sort was much less compared to Merge Sort, however understanding it, for me at

least, was more difficult. Personally, I would rather type more, easier-to-understand code, rather than less, harder-to-understand code.

Following this is Quick Sort, the quickest algorithm out of the seven tested. It took only 6ms for 100,000 elements and 14ms for 200,000 elements, beating all of the other algorithms by a relatively significant amount. Similar to Merge Sort, this algorithm required two methods, three with my implementation, one for partitioning and swapping elements, and another for recursively calling the method on each sub-array. While understanding this algorithm took some time, I think that because of its superior efficiency compared to the others, implementing it makes sense, even if it requires multiple methods.

Finally is Heap Sort. While it did not perform as well as Quick Sort, the times were still respectable with 10ms and 21ms for 100,000 and 200,000 elements. This algorithm acts differently compared to the others as this one uses binary heaps to sort the data. This also took two methods and was by far the hardest to understand and implement. For this reason, I do not think it is a sorting algorithm that I would personally use, however, it still does perform well, so I could see the usefulness of it.

Works Cited

(All references for algorithms were also included in the code itself.)

[Merge Sort in 3 Minutes](#)

[Shell Sort | GeeksforGeeks](#)

[Learn Quick Sort in 13 minutes](#) ⚡

[Heap Sort | Sorting Algorithms | GeeksforGeeks](#)