**Sorting Algorithms Time Complexity**

Luis Morales

Stockton University

CSCI-3103-001: Data Structures and Algorithms I

Instructor Byron Hoy

December 12, 2023

The sorting algorithms I will be talking about are bubble sort, selection sort, insertion sort, merge sort, shell sort, quick sort, and heap sort. While testing the speed of these sorting algorithms, I started from 10,000 elements and went up to 200,000 elements in an array. I will be ranking the sorting algorithms from slowest to fastest. Based on the data set that I got, bubble sort is by far the slowest sorting algorithm of them all. In my laptop, I was only able to test with 150,000 elements in an array. It took my computer 59,580 ms or 59.58 seconds to sort the 150,000 elements in the array. All the other sorting algorithms took less than 30 seconds for an array with 200,000 elements. A very clear weakness of bubble sort is that it is very slow with large data sets. The only positive that I see from this sorting algorithm was that it was easy to implement.

The second slowest sorting algorithm based on my data was the selection sort algorithm. The selection sort algorithm is also slow with large data sets; however, it is roughly four times faster than bubble sort which is a lot better. For a large data set with 200,000 elements, it took 21.763 seconds to sort the elements. Coding the algorithm was also relatively easy, so overall, selection sort is better than bubble sorts but falls behind the other sorting algorithms.

The third slowest sorting algorithm from data is the insertion sort algorithm. Based on my data, insertion sort is four or five times faster than the selection algorithm. One of the reasons insertion sort is faster than selection sort is because, according to the time complexity chart, the worst-case scenario for both insertion sort and selection sort is O(n2) but the best-case scenario for selection sort is the same as the worst case, but for insertion sort, it is O(n). On my data, selection sort and insertion sort never took a similar time while running the same number of elements, so it is safe to assume that insertion sort never had a worst-case scenario, especially when there were more than 100,000 elements.

The fourth slowest sorting algorithm is shell sort. It is interesting, because according to my data, shell sort and heap sort took the same amount of time to sort 10,000 and 20,000 elements but shell sort started to take longer as the number of elements increased. For 10,000 elements, it took both sorting algorithms 11 seconds and for 20,000 it was 17 seconds for shell sort and 18 seconds for heap sort. However, as mentioned before, after 30,000 elements, it took shell sort two times longer to sort the elements than it took heap sort. I do not know why shell sort and heap sort perform similarly at the start when both have different Big-O notations for the worst- and best-case scenario. Other than the smaller sets, merge sort, quick sort, and heap sort are faster than shell sort with larger data sets.

As for the last three, it is not easy to tell which one is faster because all three have a O(nlogn) for the best-case scenario and O(nlogn) for the worst-case scenario except for quick sort which is O(n2). Looking at my data, for smaller sets, quick sort is faster than the other two but as the data got bigger, the run time for quick sort would sometimes be two times longer than merge sort and heap sort. After sorting 200,000 elements, quick sort was almost two times slower and at some point, quick sort was three times slower than merge sort and heap sort which likely means it had the worst-case scenario or at least close to the worst-case scenario. It was weird with quick sort because sometimes it was faster than other times for example, with 120,000 elements, it took 0.091 seconds and with 130,000 elements, it took 0.06 seconds to sort. My only guess would be that the difference was if it was a best-case scenario or a worst-case scenario. While it is slower than merge sort and heap sort, I would not say that it is a big disadvantage because the difference was less than one second. I personally like quick sort a lot more than heap sort.

With data sets less than 100,000 for both merge sort and heap sort, it took both a similar time to sort the elements but as the data set got bigger, merge sort took a little more time to sort and while it is also true for heap sort, the time it increased by for merge sort was more than that for heap sort. Overall, for larger data sets, heap sort is faster than merge sort but like I mentioned before with quick sort, the difference between the three is less than 1 second so there is not much of a difference. However, the biggest disadvantage of merge sort is that it requires more memory storage because it must create two arrays temporary arrays that stores halves of the original array to sort the elements. Other than storage problems, I think merge sort is much easier to code than heap sort.

Tables for Sorting Algorithms:

As the number of elements increases, the time it takes to sort also increases exponentially.

As the number of element increases, the time it takes to sort increases exponentially but takes less time than bubble sort.

As the number of element increases, the time it takes to sort increases exponentially but takes less time than selection sort.

As the number of element increases, the time it takes to sort increases however, it sometimes takes less time than the previous number of elements. It is the second fastest sorting algorithm.

As the number of element increases, the time it takes to sort increases however, after 70,000 elements, it sometimes takes less time than the previous number of elements.

As the number of element increases, the time it takes to sort increases however, after 70,000, it sometimes takes less time than the previous number of elements and sometimes it is a lot less.

As the number of element increases, the time it takes to sort increases however, after 50,000, it sometimes takes less time than the previous number of elements and sometimes it is a lot less. It is the fastest sorting algorithm.

A table with numbers and letters

Description automatically generated

This is a photo of the time complexity chart taken from Professor Byron Hoy’s PowerPoint slides.

Works Cited

Koffman. Data Structures: Abstraction and Design Using Java. Wiley. 4th Edition