Nathan Darrett

Professor Amarasinghe

CSCI 41

04 November 2023

Sorting Algorithm Analysis

INTRODUTION

The project that I conducted consisted of running several sorting algorithms on several types of arrays and observe how their times of the arrays grew in size. In this report I will provide graphs of the different times from the sorting algorithms and I will discuss and also explain about which one is the best to use when needing to sort for large arrays of infinitely big sizes.

SORTING ALGORITHM PROCESS

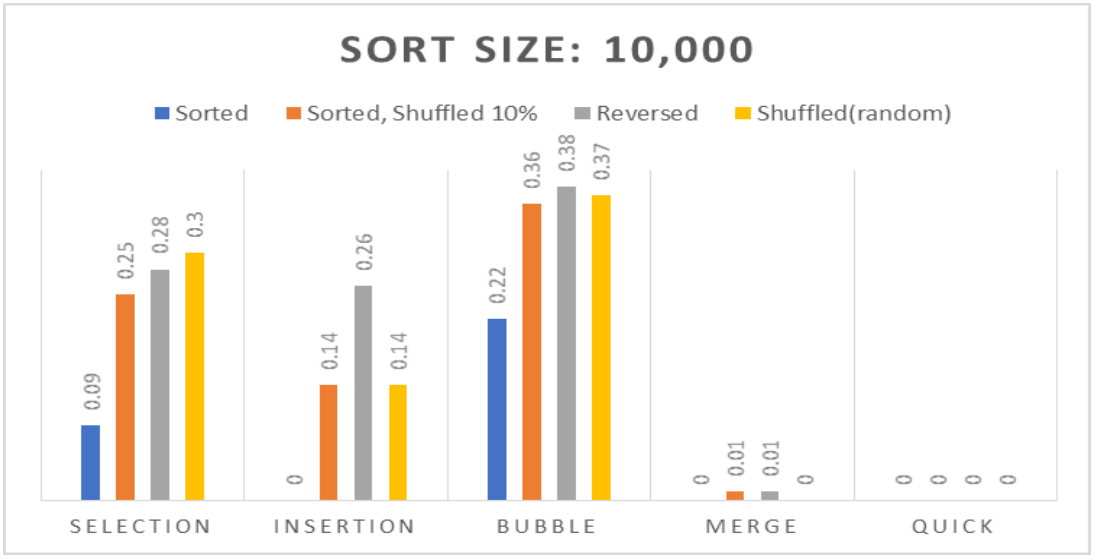
The sorting algorithms I used for this project consists of time complexities ranging from O(nlogn) to O(n^2). This includes Selection Sort, Insertion Sort, Bubble Sort, Merge Sort, and Quick Sort. I used three different array sizes(sizes of 10,000, 100,000 and 1,000,000) for all five sorting algorithms, as well as 4 array types for each size (random, shuffled 10%, reversed and sorted), which makes up a total of 60 sorts.I utilized four different array types to observe how these sorting algorithms perform in various scenarios in best cases of the array already being sorted and also from the array being completely unsorted beginning to end. Most importantly we also increase the size to see if the time taken to run will increase, decrease or stay consistent.

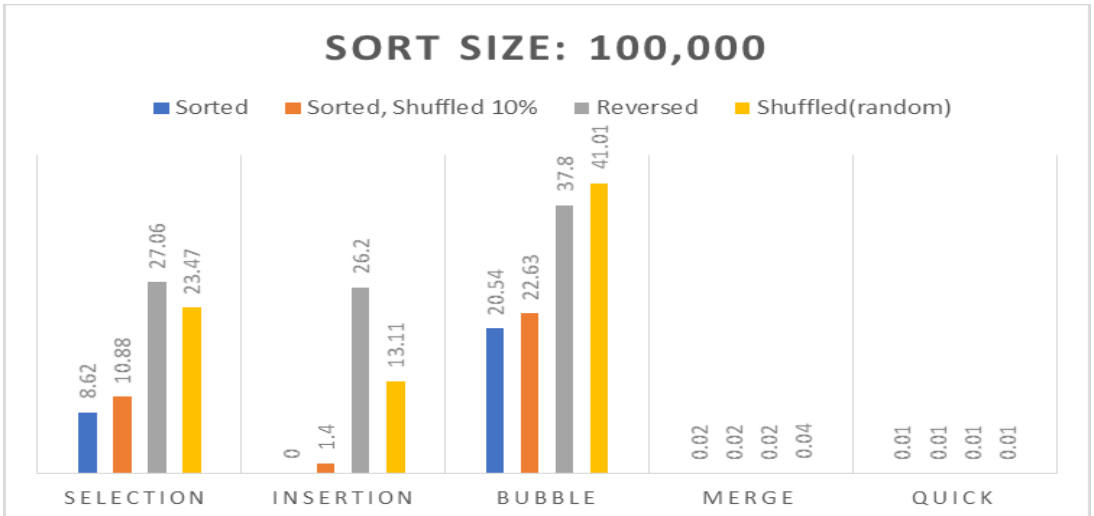
RESULT

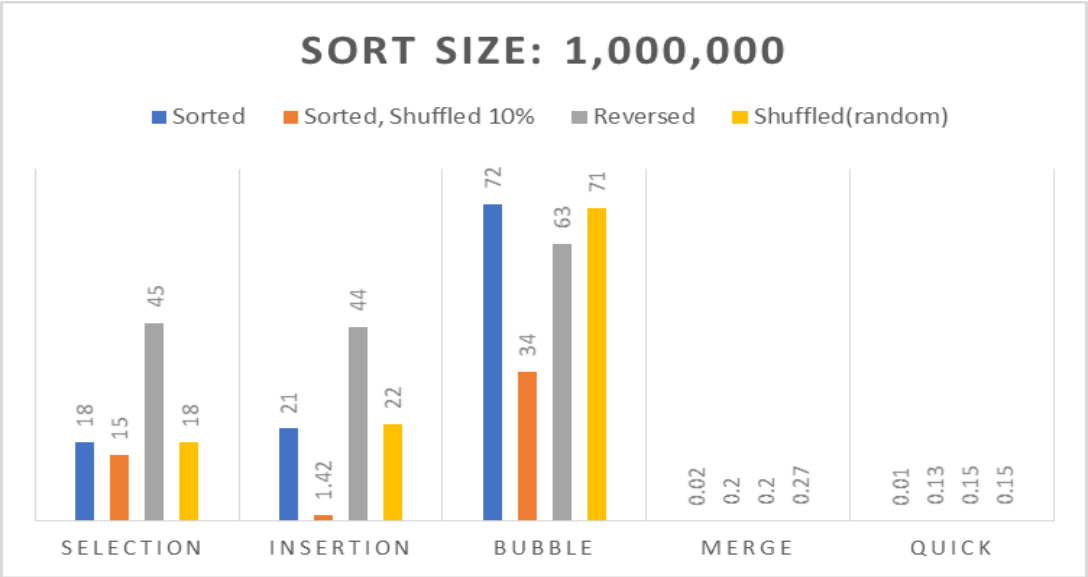
The program used to conduct all the possible sorts were done on a ASUS Aspire 5, 64 bit machine with a 11th Gen Intel(R) Core(™) i5-1135G7 at 2.40GHz, containing 8 GB of ram and running on Windows 10 Home (22H2). All the sorts which added up to 60 tests in total took roughly around 6.1 hours to run. Below you see the time taken to run for all the sorts.

BAR GRAPHS OF SORTS FROM 10K, 100K, AND 1M

The first two graphs with sort sizes 10,000 and 100,000 are measured in seconds, except for sort size 1,000,000. Reason being because of how long it took, so it will be measured in minutes on its graph.







DISCUSSION

Selection sort has a time complexity of Ω(n) - O(n^2). For reasons, I did not add a case where it can be Ω(n) to get a more accurate time of what selection sort(as well as other sorts in this analysis) can do with an already sorted array. The sorted array did just fine with time for a size of 10,000, but as the size grew bigger the time increased. From the looks of it you can really see how selection sort is O(n^2) from time it took with a reversed array. With the reversed array and bigger size, the time grew exponentially from 0.23 milliseconds on a 10,000 sized array to 45 minutes with a 1,000,000 sized array.

Insertion Sort has a time complexity also just like selection sort of Ω(n) - O(n^2). Also for the same reason with selection sort, I did not check if it is already sorted to see how Insertion Sort handles a sorted array. The time taken with a 10,000 sized array was almost instant being 0 seconds, but the time grew significantly with a bigger size of 1,000,000 with a time of 44

minutes. We can see that Insertion Sort and Selection Sort have close times when sizes grew larger.

Bubble Sort has a time complexity of Ω(n) - O(n^2). By looking at the charts we can see that bubble sort was by far the worst sorting algorithm out of all of the sorts when it came to bigger values sets. It took about an hour and twelve minutes to finish its worst run, which surprisingly wasn't the reversed array, but the sorted array.

Merge Sort has a time complexity of Ω(nlogn)-O(nlogn). With knowing this we can say that it will have better times complexities already from the 3 previous sorts that were tested. Merge Sort had a worst time of 0.27 seconds which is pretty fast for a size of 1,000,000 values. Unlike the Selection Sort, Insertion Sort, and Bubble Sort, the times with Merge sort were very consistent throughout all its test.

Quick Sort has a time complexity of Ω(nlogn)-O(n^2). The reason being O(n^2) is because the pivot point for Quick Sort is crucial for how the sorting algorithms acts. In the program I chose to get a random pivot point each time in hopes of a good run time. Quick sort really owns up to its name as the times for all of its tests were instant at 0 seconds. Although Quicksort can be unstable leading to O(n^2), with the right pivot point it can dominate all the sorting algorithms.

CONCLUSION

After running all the sorts with all types of possible sorts and sizes we can conclude that Quicksort was the best. Although Quicksort had faster execution times, I'd argue that Merge Sort is superior just because it's time complexity Ω(nlogn)-O(nlogn). On the other hand the worst sorting algorithms for sorts in terms of time would be Selection Sort and Insertion, but most

definitely Bubble Sort. The way Bubble Sort works is just inefficient compared to the other sorting algorithms making it the worst. So in conclusion not matter what specs you have on your computer or laptop we can safely say out of the 5 sorting algorithms that Merge Sort is the most beneficial one.