Sorting Algorithms

Jerrick Torres - 2301036

I. Introduction

THE four sorting algorithms (and their respective theoretical run-times)that I've chosen are:

	Bubble	**Selection**	**Insertion**	**Quick**
Average	$\Theta(n^2)$	$\Theta(n^2)$	$\Theta(n^2)$	Θ(n log n)
Worst	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(n^2)$

A. Why not Mathematical Analysis?

It is evident that if one attempted to assess the run-times of these algorithms using only Mathematical Analysis, they would be unable to differentiate measured performance due to all four algorithms having a theoretical worst-case run time of $O(n^2)$ due to the hypothetical nature of this method. Thankfully, we are able to measure and compare 'ACTUAL' performance of these sorting algorithms using Empirical Analysis.

One thing to keep in mind is that this experimental method's results will most likely vary between different systems (even if the same exact code is run), as there are a multitude of non-implementation factors that will affect real run-time. Some aspects that may provide time alterations during Experimental Analysis include: quality and type of hardware, compiler selected, and operating system. Hence, while collecting results, to minimize time-variation per trial, I made sure that only Docker and the Terminal were the only applications open.

II. Predictions

Due to prior knowledge from class, it was obvious that Quick Sort would be the fastest sorting algorithm of the four, as it is the only non-simple sorting algorithm. Additionally, knowing that bubble sort is the simplest sorting algorithm easily let me guess that Bubble Sort would be the slowest. Also, as the list of doubles are not presorted, I assumed that Insertion Sort would be faster than Selection Sort, but significantly longer than Quick Sort. However, I really had no frame of reference to infer the specific magnitudes of the time differences between the sorting algorithms.

III. Observations

While coding the implementation for all four algorithms, I started off with Insertion Sort as Rene provided the code for that. When it came to Selection Sort and Bubble Sort, I had no problems producing their code. However, Quick Sort was one of the most complicated algorithms that I've ever produced. Only after a few hours of step-by-step analysis of Quick Sort, only then did I start to understand the partition and recursion aspects of the algorithm.

When testing the each sorting algorithm's respective run-times, I used a list of 100,000 doubles, due to the `<ctime>` standard class only counting whole seconds. Unfortunately, when Quick Sort was first tested, it outputted a run-time of 0 seconds. This is due to the exponential speed that Quick Sort does its job and the limitations of the standard library time class. So, I replaced all C `<ctime>` duration timers with C++ `<chrono>` duration timers as they measure in milliseconds.

IV. Findings and Conclusion

Luckily, my predictions were correct. From slowest to fastest: Bubble Sort, Insertion Sort, Selection Sort, then Quick Sort. ... is that Bubble Sort took significantly longer to sort than the other two simple sorting algorithms. What surprised me however, is that Quick Sort was not only significantly faster, but exponentially faster than all three other methods.

	Bubble	**Selection**	**Insertion**	**Quick**
ctime	67	20	17	0
chrono	64.6398	19.907	16.58	0.0208605

A. Assignment Limitations

While this assignment provided me first-hand knowledge regarding these sorting algorithms' runtimes, sadly there was no 'space complexity' aspect that needed to be monitored according to the requirements. However, it is important to keep in mind that despite Quick Sort being the fastest sorting algorithm of the four, it also has the worst space complexity of all. This means that Quick Sort would probably not be recommended in systems with limited memory (ie. embedded systems).

Space Complexity	**Bubble**	**Selection**	**Insertion**	**Quick**
Worst Case	$O(1)$	$O(1)$	$O(1)$	$O(n)$

B. Why choose each Algorithm?

Bubble Sort is the simplest to implement, sufficient enough if the list is small, sorts in-place, and is the most efficient with nearly sorted arrays. However, it has the worst time-complexity and will take the longest time with large lists.

Insertion Sort is the second simplest to implement and efficient with small lists. On the other hand, its worst-case run-time increases at a magnitude of n^2, requires shifting of elements, and incredibly inefficient for reverse-sorted data.

Selection Sort also performs well on a small list and is faster than Bubble Sort and slightly slower than Insertion Sort. It sorts in-place (similar to Bubble Sort), and its run-time is heavily influenced by how the provided array is sorted. It also has a n^2 time-complexity hence should not be used for large numbers of elements.

Quick Sort is one of the best sorting algorithms and the only complex sorting algorithm of the four. It sorts large lists exponentially quicker than the other sorting algorithms and sorts in-place. In addition to not being recommended for systems with limited memory, it also can't be implemented in systems that don't support recursion.

ACKNOWLEDGMENT

Thanks Rene, for a great semester!