Assignment 6: Sorting

Matt Raymond

***Abstract*—The point of this paper is to implement algorithms for multiple sorting algorithms and compare asymptotic analysis of these algorithms against the empirical analysis.**

I. Introduction

THERE are many different algorithms for sorting inputs. All of them have their own strengths and weaknesses, as well as special use cases where they would be more or less efficient. In this paper, we will explore four sorting algorithms (Bubble Sort, Quick Sort, Insertion Sort, and Cycle Sort) in both asymptotic notation and empirical analysis and compare the results.

II. Time Differences

Based on asymptotic analysis, we assumed that our runtimes would be as follows:

- Bubble Sort: $O(n^2)$
- Cycle Sort: $O(n^2)$
- Quick Sort: *Worst case is* $O(n^2)$*, best case is* $O(nlogn)$
- Insertion Sort: *Worst case is* $O(n^2)$*, best case is* $O(n)$

We expected the performance of Bubble and Cycle Sort, and Quick and Insertion Sort to be the same because they were the same based on asymptotic analysis. However, despite the fact that some of our asymptotic growth rates are equal, the performance of our algorithms still differ in real-world:

- Bubble Sort: *31 seconds*
- Cycle Sort: *46 seconds*
- Quick Sort: *<1 second*
- Insertion Sort: *6 seconds*

This is probably due to the relative complexities of each algorithm. For example, Cycle Sort has many more instructions per iteration than Bubble Sort, so even though the time complexity is $O(n^2)$ for each, every cycle took more time for Cycle Sort, resulting in a longer runtime. I believe that this was the same thing that caused the difference between Insertion Sort and Insertion Sort.

Overall, I was surprised by the difference in runtime between $O(n^2)$ and $O(nlogn)$, and it was much more drastic than I would have thought.

III. Tradeoffs of Each Algorithm

Each algorithm has its own pros and cons, which are illustrated in *Table I*:

TABLE I
Algorithms Tradeoffs.

Algorithm	Pros	Cons
Bubble	Simple	Inefficient
Cycle	Fewest possible array writes	Inefficient
Quick	Fast for random data	Complex
Insertion	Fast for pre-sorted data	Complex

IV. Impact of Programming Language

These programs were implemented using `C++`, which allowed us to write more specific and technical code than other programming languages such as `Python`. For example, `Python` makes it a lot more difficult to use pointers, which you can't use directly. This means that we were better able to optimize our code and decrease the runtime. Additionally, since `C++` is a compiled language instead of an interpreted language, our program ran much faster than it would in `Python`. However, if we had used `Assembly`, our code might have been even faster (since we could perform only the necessary commands), but it also would have been much more difficult to write. Overall, writing our code in `C++` allowed us to write our code quickly while still being able to optimize it.

V. Shortcomings of Empirical Analysis

Empirical Analysis has many shortcomings: it's not cost effective, it needs to be implemented, it's dependant on many variables (including the platform/hardware and compilers/linkers). However, there are also some plus sides. For example, it shows us that two algorithms with identical Big O's can have very different run times in the real world due to the spread of data that we're working with. Quick Sort proved to be much faster than Insertion Sort because our data was completely randomized. This was the optimal setup for Quick Sort, and we were able to see this reflected in the runtime (*<1 second*).

References

[1] https://www.geeksforgeeks.org/cycle-sort/
[2] https://www.geeksforgeeks.org/bubble-sort/
[3] https://www.geeksforgeeks.org/quick-sort/
[4] https://www.geeksforgeeks.org/insertion-sort/