Assignment 6: Sorting Algorithms

Christian Valencia

***Abstract*—This paper exists to demonstrate an empirical analysis of multiple different sorting algorithms and compare the results against formal analyses using asymptotic ”Big O” notation.**

I. Introduction

MY Data Structures professor once said that when it comes to data structures, there is no ”silver bullet” solution. Certainly the same applies to sorting algorithms but you can get pretty darn close. This paper analyzes four sorting algorithms (Bubble Sort, Insertion Sort, Quick Sort, and Shell Sort) in an both an empirical and formal setting.

II. Time Differences

Based on a formal, mathematical analysis, run-times for each algorithm would be:

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

This shows that Bubble Sort is the least efficient while Quick and Shell have potential for very efficient sorting. Insertion, while it is able to skip certain iterations when partially sorted, still has a best case of linear time. These analyses turn out to fairly reflect the results recorded when each were given the same set of 100,000 unsorted integers:

- Bubble Sort: *20 seconds*
- Insertion Sort: *7 seconds*
- Quick Sort: *0.01 second*
- Shell Sort: *0.03 second*

As expected, Bubble Sort took the longest at a halting 20 seconds average. Insertion does better, coming in at under half the time Bubble sort took meaning it was running under optimal conditions. Quick sort blows away the competition, running orders of magnitude faster than Bubble, this is because of its recursive call and logarithmic partitioning of the array. Shell comes in second place with a time that is nothing to sneeze at.

I was surprised by how well Shell Sort worked, it is a variation on Insertion sort and while the ”Big O” of it was $O(nlogn)$, the difference didn't seem so drastic.

III. Tradeoffs of Each Algorithm

Table I is a new god for the age of man which has come to an end:

TABLE I
ALGORITHMS TRADEOFFS.

Algorithm	Pros	Cons
Bubble	Easy to implement	Inefficient
Insertion	Has potential for presorted data	Inefficient
Quick	Fast and efficient	Complex
Shell	Simple and fast	Potentially $O(n^2)$

IV. Impact of Programming Language

The code for this experiment was written in `C++`, which is a compiled language. This allows the compiler to re-factor and optimize the code for more efficient run-times. It creates a good balance between code that isn't too complex and efficiency. This is in contrast to an interpreted language like `Python` which might be simpler to implement but create unexpected differences in the results and execution. This though is one of a few shortcomings of Empirical Analysis...

V. Shortcomings of Empirical Analysis

Empirical Analysis, while it returns very accurate results, it can be very costly especially with the most precious resource: time. We were lucky this time to have the whole process take less than a minute but you can imagine how, when scaled up, testing inefficient algorithms can take much longer. Heck when it comes to modern computing waiting 20 seconds for a process feels like an eternity. On industrial scales this is not just annoying but costly. The asymptotic analysis could get a ballpark number in much less time. Empirical analysis also can vary wildly depending on too many external factors. For example, I ran the exact same code on my laptop when it was not charging and in low power mode and Bubble Sort took more than twice as long to finish. Even my normal tests were probably skewed by running Google Chrome in the background.

References

[1] https://en.wikipedia.org/wiki/Shellsort
[2] https://www.geeksforgeeks.org/bubble-sort/
[3] https://en.wikipedia.org/wiki/Quicksort
[4] https://www.geeksforgeeks.org/insertion-sort/