Sorting Algorithm Analysis

Jin Jung
CPSC 353 (Section 2)
Chapman University
jijung@chapman.edu

I. Introduction

This document is a brief report and analysis of sorting algorithms: Bubble sort, Quick sort, Insertion sort, and Gnome sort.

II. Sorting Methods and Expected Results

The data used to test were randomly generate values of type double that range from -10000.00 to 10000.00. We should note that certain sorting algorithms are affected by the variance range of the input data.

A. Bubble Sort

The expected runtime of bubble sort is $O(n^2)$. We expect Bubble Sort to be the slowest out of all considered algorithms since it has to make n^2 regardless of state of data.

B. Quick Sort

The expected runtime of quick sort is O(n*log(n)). We expect quick sort to be the best performing on average based on asymptotic analysis. This should not deviate too much since we are using randomized data and therefore the worst case of $O(n^2)$ occurring is low.

C. Insertion Sort

The expected runtime of Insertion Sort is $O(n^2)$. We expect this algorithm to run faster than bubble sort, but slower than quick sort. For small input data, we expect this algorithm to perform very efficiently since the number of inversions will be small and performance may approach O(n).

D. Gnome Sort

The expected runtime of gnome Sort is $O(n^2)$. Similar to insertion sort, we expect gnome sort to perform close to O(n) if the data is presorted. This is not the case for our test, but we expect gnome sort to perform better than bubble sort.

III. Testing Process and Actual Results

Tests were performed with input data size of 50, 50000, and 500000. For small input data size of 50, the measured time for Insertion, Quick, and Gnome sort were almost identical. Bubble sort took twice as long to complete. For input data size of 50000, Bubble sort: 7.72s, Insertion sort: 1.38s, Quick sort: .0075s, and Gnome sort: 5.23. As expected Quick sort was extremely fast, Insertion sort both performed somewhat efficiently, while Gnome sort performed only slightly better than Bubble sort. For input data of size 500000, Bubble sort: 948s, Insertion sort: 166s, Quick sort: 0.1091s , and Gnome sort: 597. Once again, bubble sort was incredibly inefficient compared to Quick and Insertion sort. Gnome sort performed significantly worse than Quick and Insertion sort, and only slightly better than Bubble sort. Quick sort is once again the best performing algorithm by a wide margin.

A. Trade Offs

There is no such thing as a perfect sorting algorithms. From the results of our tests, we can see that increasing complexity of implementation generally improves the performance time. As expected, insertion sort performed comparably well with quick sort when the input data size remained small, but Quick sort performed exponentially better than Insertion sort as the data size grew. It was surprising to see O(n*log(n)) perform so much faster than $O(n^2)$.

B. Choice of Programming Language

C++ was the Programming Language used to performing the Sorting tests. C++ is a versatile language capable of low level programming, and gives users complete control over memory management. In comparison, a programming language like Python would have made it more difficult to optimize implementation of our sorting algorithms. Using the built-in libraries saved implementation time, but introduced a step that will differ for other programming languages based on their respective built-in functions that achieve similar behavior.

C. Shortcomings of Empirical Analysis

Although empirical analysis allows us to see how efficient an algorithm is performing in a specific use case as well as allowing us to see concrete values for time/memory usage, it comes with certain disadvantages. First, empirical analysis is unable to account for all possible data inputs since we have time and resource constraints. Second, the performance depends on hardware and software environments and results will differ whenever empirical analysis is done on other systems. Lastly, empirical analysis is time consuming and expensive since we need to fully implement and execute the algorithms in order to test them.

References

[1] https://www.geeksforgeeks.org/quick-sort/
[2] https://www.geeksforgeeks.org/gnome-sort-a-stupid-one/
[3] https://www.geeksforgeeks.org/insertion-sort/
[4] Data Structures Algorithms in C++ 2ed Goodrich