Assignment 6 (Report): A Comparative Look At Sorting Algorithms

Brandon C. Briseno
Schmid College of Science and Technology
Chapman University
Orange, California 92866
CPSC 350-01: Data Structures (German)
Website: http://www1.chapman.edu/ brise105/
Email: brise105@mail.chapman.edu — SID: 1932124

***Abstract*—This report briefly discusses a comparative analysis of three sorting algorithms: Insertion Sort, Selection Sort, Bubble Sort, and Quick Sort.**

I. Introduction

Asymptotic analysis offers a fast and cost efficient approximation of performance. However, it is also important to acknowledge just how drastically algorithmic and environmental conditions can affect the performance of a sorting program. May 18, 2018

II. Analysis

A. Insertion Sort: O(n^2)

Insertion sort is comparable to sorting a deck of cards. Furthermore, it is most effective when sorting data sets around the same size of a deck of cards. Meanwhile, it is quite simple and effective at sorting "pre-sorted" values. This is due to the fact that it doesn't require alot of overhead memory allocation. However, a scenario involving a large and fully randomized set of data would show that insertion sort has extremely poor worst case performance.

B. Selection Sort: O(n^2)

Selection sort operates in place and finds the smallest value in the unsorted list to swap it with the leftmost unsorted value and shifting the array one index to the right.Furthermore, selection sort is reliably quadratic in runtime as the number of potential comparison checks is minimized.

C. Bubble Sort: O(n^2)

Bubble sort is quick and dirty. Use this algorithm when you're on an island sorting coconuts.

D. Quick Sort: O(n^2)

Quick sort is quite efficient with data sets that can be completely stored in main memory. Furthermore, Quicksort can operate in-place on an array, therefore it requires small additional amounts of memory to perform the sorting. Quick sort is quite similar to selection sort, except for the fact it uses a simple partition and it doesn't always choose worst-case partition.

III. Conclusion

For my data set, I used a 100,000 randomized integer values ranging from 1-100000. However, the program is set to handle a large data set of arbitrary size thanks to the use of C++ and dynamic memory allocation. Selection sort was by far the worst in terms of performance, which was surprising compared to insertion sort and given how the data set was. Furthermore, while selection sort was by far the worst in terms of performance, Bubble sort was not trailing to far behind. Meanwhile, Quick sort completely crushed the data set in fractions of a second. The time differences between bubble/selection and quick sort astounded me (Quick sort is so fast!). While empirical analysis is more accurate, it takes forever for some of the algorithms to finish, I could imagine how this could waste huge amounts of resources when dealing with low capacity CPUs and massive data sets.

IV. References

Data Structures and Algorithms in C++ (2nd Edition). Goodrich, Tamassia, and Mount. Wiley Sons. 2011. Print.