Sorting Algorithms and Assignment 6

Data Structures, CPSC 350, Professor German, R.

Matsui, Donovan. Author. 12/4/2015

***Abstract*—Often times when writing programs, large quantities of data must be sorted. Developers are forced to choose between a variety of sorting algorithms with various benefits and downfalls. Assignment 6 provided hands-on experience with three such algorithms, which we implemented ourselves and timed while performing sorts on three duplicate arrays of doubles. Doing so provided concrete evidence of the expected theoretical run-times for each algorithm based on big Oh analysis. There were additional conclusions which are to be discussed as well.**

I. INTRODUCTION

THERE comes a time in every developer's career when he, she, it, or they must sort a large quantity of data. Data sets may be sorted by a variety algorithms, some of which are easy to write, some of which are better with nearly sorted data, and others yet which are plainly and simply difficult to implement. Assignment 6 asked us to write and time three algorithms: quick sort, insertion sort, and an additional algorithm of our choosing. Unfortunately for me, I wanted to be an over-achiever and decided to design and implement my own sorting algorithm.

II. WHY YOU SHOULD NEVER TRY TO DESIGN YOUR OWN SORTING ALGORITHM

My first instinct was to go to my trusty whiteboard and play around with some math. Easy enough. I had been toying around with the concept of an averaging-based sorting algorithm and within half an hour had designed one that for all intents and purposes, appeared functional. The next step was to open up notepad++ and start coding away. An hour into it, I ran my new algorithm, coined Dino-sort, on 10,000 random doubles. It didn't work as intended. A Couple hours and several whiteboard/coding circuits later, I was consistently getting better results. Maybe 60-70% sorted with some anomolies here and there. This gave me hope, so I plugged away another few hours, tweaking dinosort to 'perfection'. By the end of the day, I could sort 20,000 doubles to ~90% accuracy. Not bad! Fast forward to the next morning and I was at it again. This is when I realized my mistake. I had spent approximately a day and a half working on a sorting algorithm based on averages. How could I be so dense as to try and create a precision based algorithm centered around approximations? I was heart-broken. Visions of my name in textbooks as the inventor of dinosort faded, and I scrapped the idea altogether. Crestfallen, I chose to implement a tree-sort instead and finished the assignment swiftly.

III. ASSIGNMENT ANALYSIS

In the end, dinosort ran a mere 1 ms slower than quicksort, and about the same speed as tree sort. Insertion sort was magnitudes slower than the rest, just as mathematical analysis would suggest.

Writing and timing the algorithms was a valuable learning experience overall, but I would have to say that I learned the most from trying to create a new sorting algorithm altogether.