Assignment 6

Bubble, Selection, Insertion, and Quick Sort

Lloyd Black
229598
CPSC 350, Section 2
lblack@chapman.edu

***Abstract*—This is a brief report on the performances of Bubble, Selection, Insertion, and Quick Sorts in sorting random numbers. Algorithms were implemented in C++, and random number lists were generated using Python 3.5.3.**

I. Introduction

For the purposes of testing the four algorithms, I generated 4 lists of numbers (sets 1, 2, 3, and 4). Sets 1-3 consisted of a sequence of numbers between zero and ten million in as random an order as the Python 3 random module provides. Set 1 consisted of 50,000 numbers, Set 2 of 100,000, and Set 3 of 200,000. Set 4 consisted of 200,000 semi-sorted random numbers (generated by writing two random numbers in a range guaranteed to both be less than the next two numbers).

II. Results

A. Bubble Sort

Mathematically, we know that Bubble Sort runs in O(n^2) time, but what does that look like practically? Running the Bubble Sort algorithm against Set 1 yielded a run time of 8.6819 seconds. Thus, doubling the set size should yield a run time 4 times as long, or in this case, 34.7276 seconds. In reality, Set 2 took 35.9679 seconds to sort. Similarly, Set 3 took 144.609 seconds compared to an expected 143.872. All of these small differences are likely due to processor clutter and aren't out of the realm of the expected. Set 4 took a mere 14.487 seconds (ten times as fast as the same number of data randomly sorted), likely due to the significantly reduced number of data swaps occurring.

B. Selection Sort

Mathematical analysis of Selection Sort also yields a O(n^2) performance. This is corroborated by the empirical performance times recorded. Set 1 took 3.497 seconds to sort, Set 2 took 13.374 seconds, and Set 3 took 52.855 seconds. Each Set takes roughly 4 times as long as the last, as expected. The thing to note is that, despite also being O(n^2), Selection sort sorted the data in less than half the time that Bubble Sort did. Set 4 took 13.182 seconds, which, while significantly faster than Set 3, is just about equal with Bubble Sort's Set 4. It seems, having lost the advantage of not swapping as often, Selection Sort becomes about equal to Bubble Sort.

C. Insertion Sort

Insertion Sort is in a similar situation to Selection Sort. Mathematically, it runs O(n^2). Practically, Set 1 took 1.950 seconds, SEt 2 took 7.603 seconds, and Set 3 took 30.799 seconds. Again, all three of these times are roughly 4 times the length of the set prior, as expected. Also, they are shorter still than the previous O(n^2) algorithm. Set 4 took all of .692 seconds. The difference between performance of Insertion and the other two O(n^2) algorithms is more pronounced with semi-sorted data, it seems.

D. Quick Sort

Unlike the three previous algorithms, mathematical analysis of Quick Sort shows a O(nlog(n)) performance. In practice, Quick Sort sorted Set 1 in .00842 seconds, Set 2 in .0165 seconds, and Set 3 in .0335 seconds. This is a stark contrast to the previous three algorithms, as Quick Sort sorted 4 times the data of the most efficient O(n^2) algorithm about 60 times quicker. One point to note, however, is that Quick Sort sorted Set 4 in .0180 seconds. Still faster than its comparable Set 3, but only by a factor of 2 or so, as compared to the 10 times or greater performance boost the other algorithms received.

III. Conclusion

While I'm unsurprised in the amount of time each algorithm took relative to itself when sorting more and more data, I am shocked at the difference in performance even within the O(n^2) algorithms. Of course, Quick Sort sorting 200,000 numbers in three hundreths of a second was impressive, but thinking of algorithms purely in terms of Big Oh oversimplifies things quite a bit. For instance, it's much more apparent much semi-sorted data affects Bubble and Insertion sort now, where simply labelling them all O(n^2) didn't account for those sorts of differences.