Sort Algorithms

Daus(Davis) Carmichael

December 2019

Bubble Sort

The Bubble Sort algorithm is based on proceeding through each index and determining if there needs to be a swap performed. This has a run-time of $O(n^2)$. This function is a very simple and easy algorithm to implement. The run-time of this function for an array of 64 doubles 28 was microseconds, very similar to the selection sort.

Insertion Sort

This algorithm is Todd Carmichael's[1] favorite, a when given partially sorted data the run-time approaches O(n). Even with random data it is likely that some data will be sorted in small chunks. Nevertheless, the run-time on the worst case is still the same as the other two simple algorithms, performing at $O(n^2)$. The comparison between Insertion, Selection and Bubble shows relatively similar run-times with the run-time of Selection at 11 microseconds, returning the fastest run-time.

Selection Sort

Selection Sort was implemented in a simple manner as well. This algorithm is characterized by placing the minimum value at the beginning of the array repeatedly until the algorithm is sorted. The run time is similar to the run-time of bubble sort, however, there are usually fewer swaps. The official run-time of Insertion sort is $O(n^2)$. The run-time of this program with the same data set of 64 doubles was 26 microseconds.

Merge Sort

The final sorting algorithm is merge sort with the most efficient algorithm according to its big-Oh run-time. Merge sort has an asymptotic run-time of O(n log(n)). This function performs this way without regard to the inputted data making it a very versatile sorting algorithm. However, it is more computationally taxing than the others. The function ran in 17 microseconds falling in second place, however, this may have been due to the sorted nature of the data randomly.

Comparison

When a larger data set was used of size 1024, the performance of the merge sort shined at only 157 microseconds, while the insertion sort fell behind at 706. Bubble and Selection still ran far slower while selection had a slower performance of 3449 microseconds and bubble had slightly less at 3011 microseconds. Observing the activity monitor, with a set of 2^{14} or 16384 doubles the activity used by each algorithm is as follows. Merge used 2.0%, insertion spiked to 2.8% but main was around 2.2%. The other two selection and bubble had 2.5% and 1.32% respectively.

Comparison

I would like to thank Todd Carmichael, the number one resource for this project. He lent me his book and ear so I could walk through the algorithms I found

online to understand them, and slightly rewrite them in a way that was more coherent to the both of us. I used several discussions on stack overflow, the ones impacting my code are below:
-https://stackoverflow.com/questions/15749482/printing-time-in-seconds
-http://www.cplusplus.com/doc/tutorial/files/
-https://www.geeksforgeeks.org/merge-sort/ My algorithm is very similar, however, I was writing the algorithm with the other as a reference and derived why certain parts of the function are the way they are. I also used the book from Todd Carmichael:
-Sedgewick, Robert. Algorithms. Addison-Wesley, 1989.