Dom Brugioni

German

CPSC350: Data Structures

May 21, 2023

Crude Sorting Algorithm Analysis

For my analysis of the various sort methods I created, I used a randomly generated list of 1000 doubles with values between 0 and 1000. The methods were timed using the ctime package and the clock() function. I assumed that QuickSort would be fastest, followed by MergeSort and then thought InsertionSort, BubbleSort, and SelectionSort would be essentially interchangeable as the slowest functions. However, after running the program, I saw MergeSort was the fastest, which makes sense since my method for choosing the pivot for QuickSort was simply to divide the array in half, rather than using a more optimized pivot. In the end, the start and end times were as follows: **Quick Sort:** Start Time: 9141, End Time: 11461; **Merge Sort:** Start Time: 11491, End Time: 11491; **Selection Sort:** Start Time: 11497, End Time: 95200; **Insertion Sort:** Start Time: 95217, End Time: 95246; **Bubble Sort:** Start Time: 95248, End Time: 95261. Since the time measurements were taken using the clock function, the times were measured in computer ticks. This increment is extremely fast, but for the most part was able to show the discrepancy in runtimes for the different algorithms. The one downside of this empirical analysis is that it doesn't work for extremely fast functions like MergeSort (start and end times were the same.) Using c++ was frustrating, but in the end beneficial, because I was able to swap pointers to memory allocations which was very low level and efficient. Once I sorted out some of the segmentation faults that that produced, I was pretty happy with the results.