Assignment 6: Sorting

Crittenden, Mary Kate
2278514
critt102@mail.chapman.edu
CPSC 350-01

This following document was written using the IEEE template in LaTeX.

I. Introduction

Using C++, Docker, and Atom, five different sorting algorithms were coded and implemented into the files sort.h and main.cpp. An input of 10, 100, 100,000, 200,000, and 800,000 doubles were used to test the runtime of each of the following five algorithms: Bubble Sort, Selection Sort, Insertion Sort, Quick Sort, and Merge Sort.

II. Results

When running with inputs of 10 and 100 doubles, all of the sorting algorithms ran instantly (less than a second. Did not show a change in the start and end time). Change began to show when the number of doubles was increased to 100,000. With 100,000 doubles, the runtimes were:

Bubble Sort: 52 seconds
Selection Sort: 58 seconds
Insertion Sort: 8 seconds
Merge Sort: <1 second
Quick Sort: <1 second

Seeing that this large number still did not affect Merge or Quick Sort runtimes, the number of doubles was increased and the test was run again. With 200,000 doubles, the runtimes were:

Bubble Sort: 3 min 30 seconds
Selection Sort: 3 min 31 seconds
Insertion Sort: 35 seconds
Merge Sort: <1 second
Quick Sort: 1 second

Doubling the amount of numbers almost quadruples the runtimes of the first three algorithms, but still did not significantly effect Merge and Quick Sort. So those two algorithms were run again with a much larger number. With 800,000 doubles, the runtimes were:

Merge Sort: 1 second
Quick Sort: 8 seconds

III. Conclusions

When testing large amounts of data, there were clear runtime differences became each of these sorting algorithms. Bubble Sort and Selection Sort had the worst runtimes, with Bubble Sort performing only slightly better than Selection Sort. Insertion Sort was the next best, having a runtime about 5 times better than Bubble and Selection Sort. But these three were all significantly slower than Merge and Quick Sort. With 800,000 numbers, both of these algorithms ran in under 10 seconds, with Merge Sort barely changing with just a 1 second runtime. So from this we can rank these sorting algorithms from fastest to slowest as follows:

1. Merge Sort
2. Quick Sort
3. Insertion Sort
4. Bubble Sort
5. Selection Sort

IV. Thoughts

A. Were the time differences more drastic than you expected? What tradeoffs are involved in picking one algorithm over another?

Using asymptotic analysis and Big O predictions, I did expect these to be pretty drastically different, so I wasn't too shocked by the results. The tradeoff between complexity of implementation to runtime is the main factor when choosing an algorithm. When dealing with small sets of data, the difference in runtimes of these five algorithms is so minimal that it makes the most sense to chose the sorting algorithm that is easiest to implement. But as data gets into the hundreds of thousands, it is best to choose the algorithm with the shortest runtime.

B. How did your choice of programming language affect the results?

My knowledge on how different coding programs affect runtime is limited. But from doing some research, it seems C++ is one of the most efficient programming languages when it comes to runtime. In an article by Martin Stein and Andreas Geyer-Schulz, they tested the runtime, memory usage, and other performance factors of various common coding programs, such as Java, C++, and Python [1]. The results of their experiments stated that "The C++ implementation emerges as the fastest one for both algorithms and on all datasets." From this information, choosing C++ as my programming language had a positive affect on my results.

C. What are some shortcomings of this empirical analysis?

The shortcomings of empirical analysis are time and power. To run all five sorting algorithms with 200,000 doubles took over 7 1/2 minutes. As for power, when I ran the program, my computer's CPU usage went from 91% idle to 55% idle. If I were to test hundreds of millions of numbers, the runtime and CPU usage would be astronomical, and likely make it impossible for the average computer to handle.

References

[1] Stein, Martin, and Andreas Geyer-Schulz. A Comparison of Five Programming Languages in a Graph Clustering Scenario. Journal of Universal Computer Science, 2 Jan. 2013, pdfs.semanticscholar.org/8a29/f5468e075f32484ce80025fbef9fdcec934e.pdf.