Profiling Research Paper

Analysis of Sorting Algorithms Using a Code Execution Profiler

Author: Hiren Patel

Profiling Research Paper

Table of Contents

Profiling Research Paper

Introduction

When using sorting algorithms, there are multiple factors that can be examined. These factors include the amount of time it takes to sort a list (if the data is completely random, already in order, reversed order, or random order) and the amount of times the method is being called or being called by itself. The purpose of this paper is to determine the differences between nine distinct sorting algorithms: sink sort, selection sort, insertion sort, merge sort, quicksort (original), quicksort (median-of-three), shell sort, counting sort, and radix sort.

The Problem

Nine sorting algorithms were analyzed to determine an algorithm's strong points, the worst-case scenario for each algorithm (i.e., Big-Oh Analysis), and which algorithms are most and least efficient overall. To determine the performance of each algorithm, a list of N values was sorted (where $N = 100, 1000, 10000$). These values were randomly generated, and then sorted in increasing order, decreasing order, and increasing order with 10% out-of-order (or left random).

Solution Plan

In order to solve the problem, I used Microsoft Visual Studio and C#. With the performance profiler, the code can be examined after execution for various statistics such as runtime, number of method calls, the amount of time spent inside the method only (exclusive time), and the amount of time spent in the method and its children (inclusive time). By using the performance profiler, these statistics can be exported into an Excel spreadsheet for analysis and visualization.

Tasks Completed

Nine different method were developed, one for each sorting algorithm. Each method was executed on four types of lists: sorting a simple random list, sorting an already sorted list, sorting a list in reverse order, and sorting a list with 10% of the data out of order. Additionally, each method was ran for each list type with 100 items, 1000 items, and 1000 items. The program was executed through the VS performance profiler to collect the data, which was them transferred to an Excel spreadsheet. The data was then analyzed in the spreadsheet and exported into tables and figures.

Results and Conclusions

After analyzing the results of the program, various conclusions can be made about the sorting algorithms. I can conclude that the sink sort algorithm is the worst sorting algorithm, because as the value of N goes up, the execution time of the sorting algorithm starts to drastically increase. Sink sort is an algorithm with an approximated runtime of $O(N^2)$, and thus performs as one of the worst algorithms.

The original quick sort and quick sort median-of-three algorithms tend to get slower and slower as the value of N goes up. Unlike the sink sort algorithm, the original quick sort and quick sort median-of-three algorithms were not drastically affected by the distribution of data in the lists.

The counting sort algorithm is the best sorting algorithm because it has the lowest execution time no matter what the *N* value is or what distribution the data is in. Even for lists with data distributed in simple random order, counting sort performs the fastest.

The other algorithms (shell sort, radix sort, selection sort, insertion sort, etc.) work best for specific scenarios, such as when *N* is expected to be small or when the distribution of the list's data is expected a certain way. For example, selection sort algorithms are most beneficial when the number of data points is small or when the data can be expected to be almost in order (approximately 10% out of order), while radix sort algorithms are most beneficial when the number of data points is large.

Summary

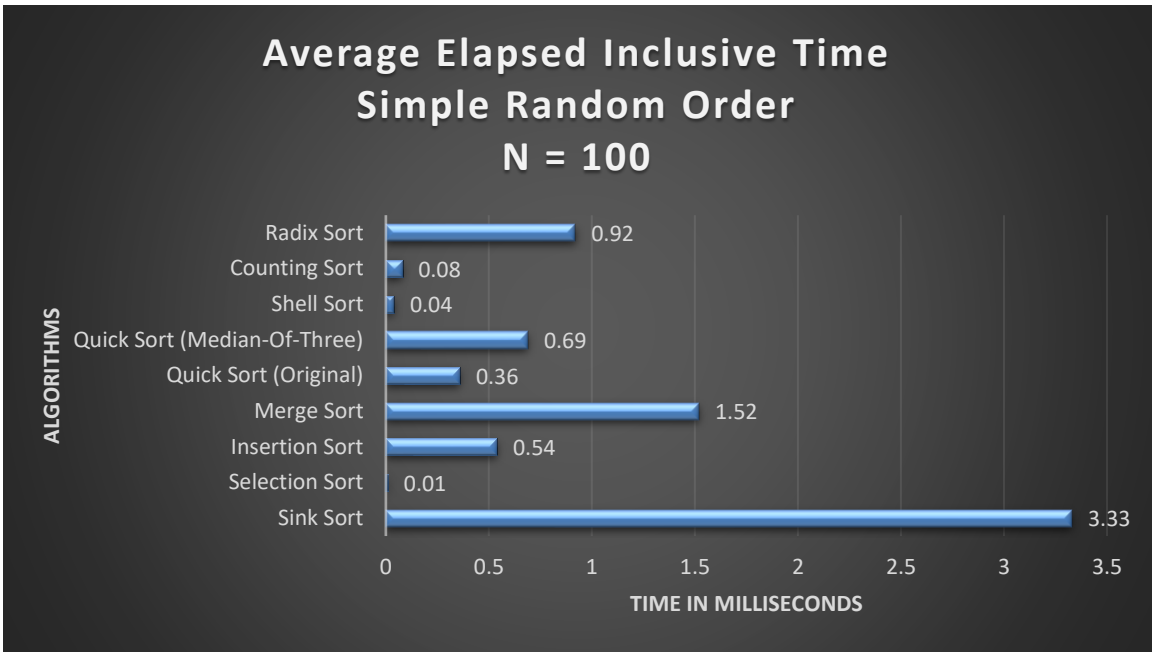
In summary, certain algorithms outperform other algorithms in all cases. For example, sink sort performs the worst than any other algorithm, in almost every case. Other algorithms perform better when the value of *N* is low or the data distribution is in the algorithm's advantage. Certain algorithms are better to use for general use, such as counting sort, radix sort, and shell sort, as they have the best performance in all cases.

Appendix 1 – Data and Graphs

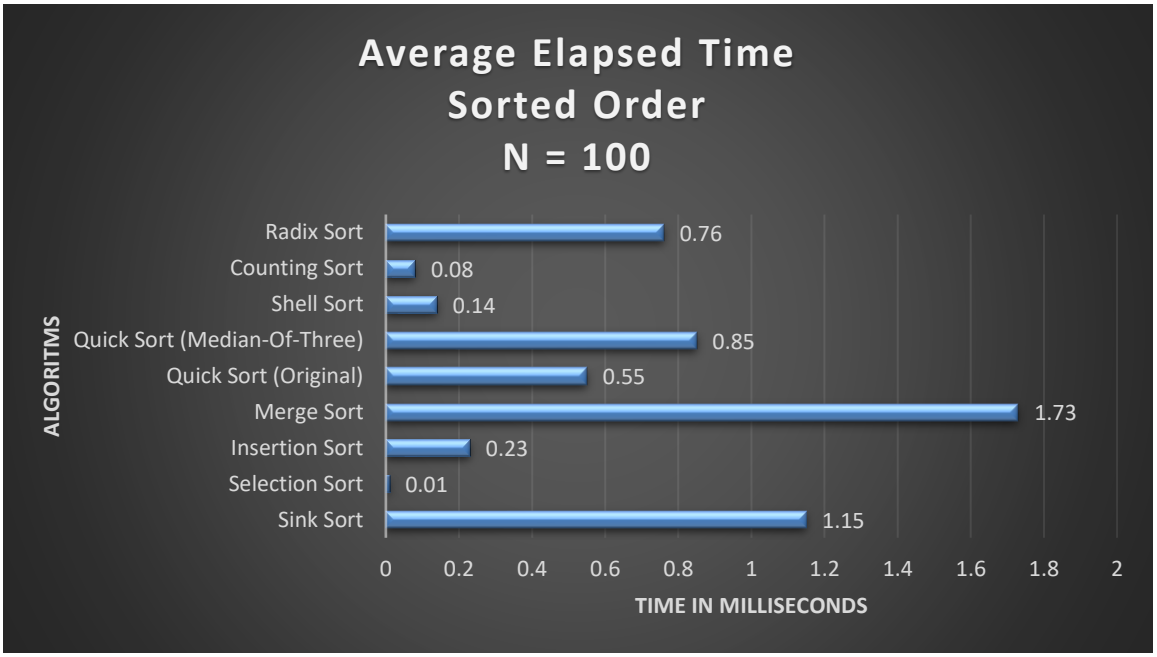
Table 1:

Algorithm	Number of Calls	Average Elapsed Inclusive Time	Value of N	Type of Execution
Sink Sort	1	3.33ms	100	Random Order
Selection Sort	200	0.01ms	100	Random Order
Insertion Sort	1	0.54ms	100	Random Order
Merge Sort	199	1.52ms	100	Random Order
Quick Sort (Original)	133	0.36ms	100	Random Order
Quick Sort (Median-Of-Three)	1	0.69ms	100	Random Order
Shell Sort	1	0.04ms	100	Random Order
Counting Sort	1	0.08ms	100	Random Order
Radix Sort	1	0.92ms	100	Random Order
Sink Sort	1	1.15ms	100	In Order
Selection Sort	200	0.01ms	100	In Order
Insertion Sort	99	0.23ms	100	In Order
Merge Sort	199	1.73ms	100	In Order
Quick Sort (Original)	131	0.55ms	100	In Order
Quick Sort (Median-Of-Three)	1	0.85ms	100	In Order
Shell Sort	1	0.14ms	100	In Order
Counting Sort	1	0.08ms	100	In Order
Radix Sort	1	0.76ms	100	In Order
Sink Sort	1	4.50ms	100	Reversed Order
Selection Sort	200	0.03ms	100	Reversed Order
Insertion Sort	2	0.47ms	100	Reversed Order
Merge Sort	99	2.81ms	100	Reversed Order
Quick Sort (Original)	1	0.72ms	100	Reversed Order
Quick Sort (Median-Of-Three)	1	1.36ms	100	Reversed Order
Shell Sort	1	0.10ms	100	Reversed Order
Counting Sort	1	0.16ms	100	Reversed Order
Radix Sort	1	1.32ms	100	Reversed Order
Sink Sort	1	0.45ms	100	10% Not in Order
Selection Sort	200	0.02ms	100	10% Not in Order
Insertion Sort	2	0.21ms	100	10% Not in Order
Merge Sort	199	3.37ms	100	10% Not in Order
Quick Sort (Original)	1	2.05ms	100	10% Not in Order
Quick Sort (Median-Of-Three)	1	2.31ms	100	10% Not in Order
Shell Sort	1	0.02ms	100	10% Not in Order
Counting Sort	1	0.11ms	100	10% Not in Order
Radix Sort	1	3.35ms	100	10% Not in Order

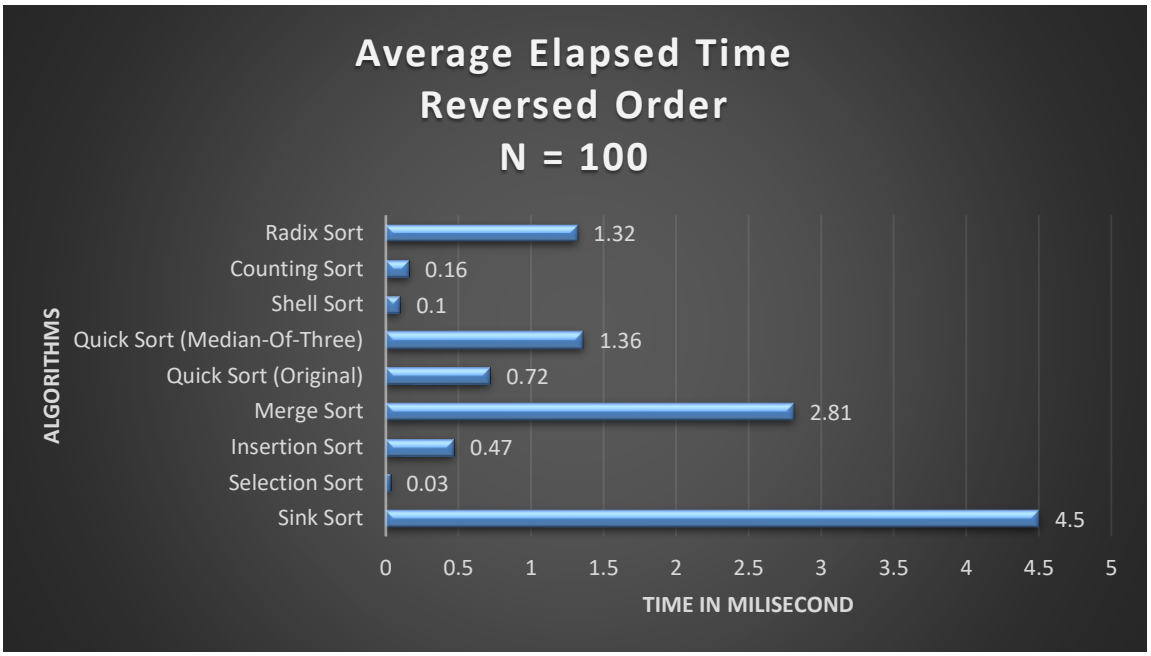
Graph 1.1:



Graph 1.2:



Graph 1.3:



Graph 1.4:

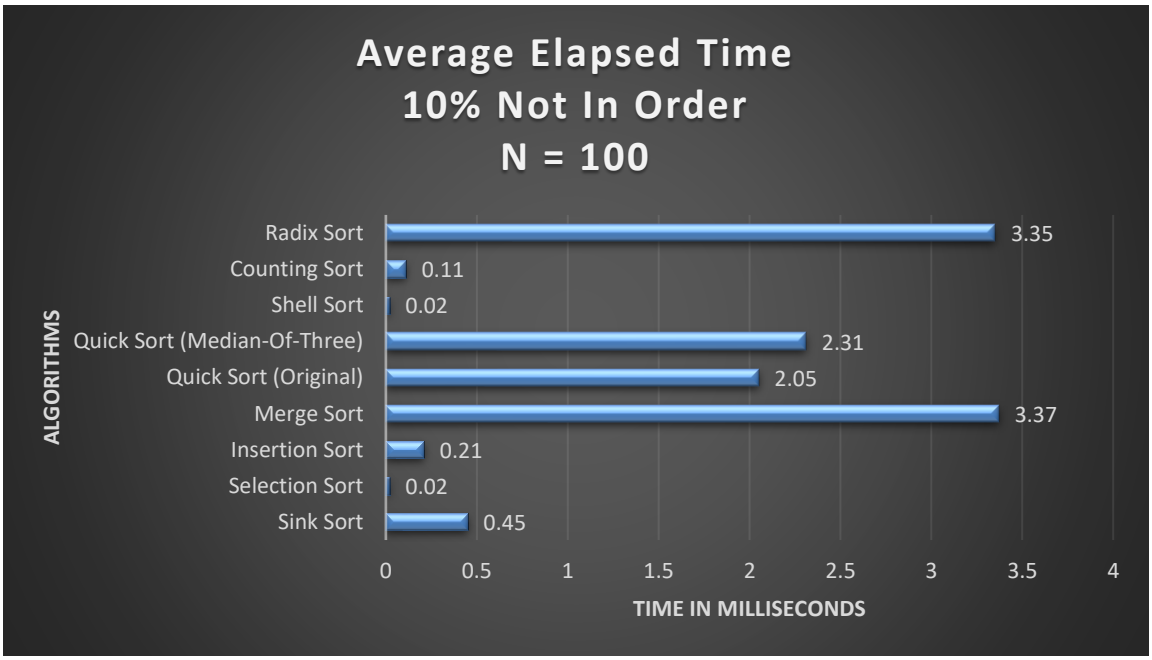
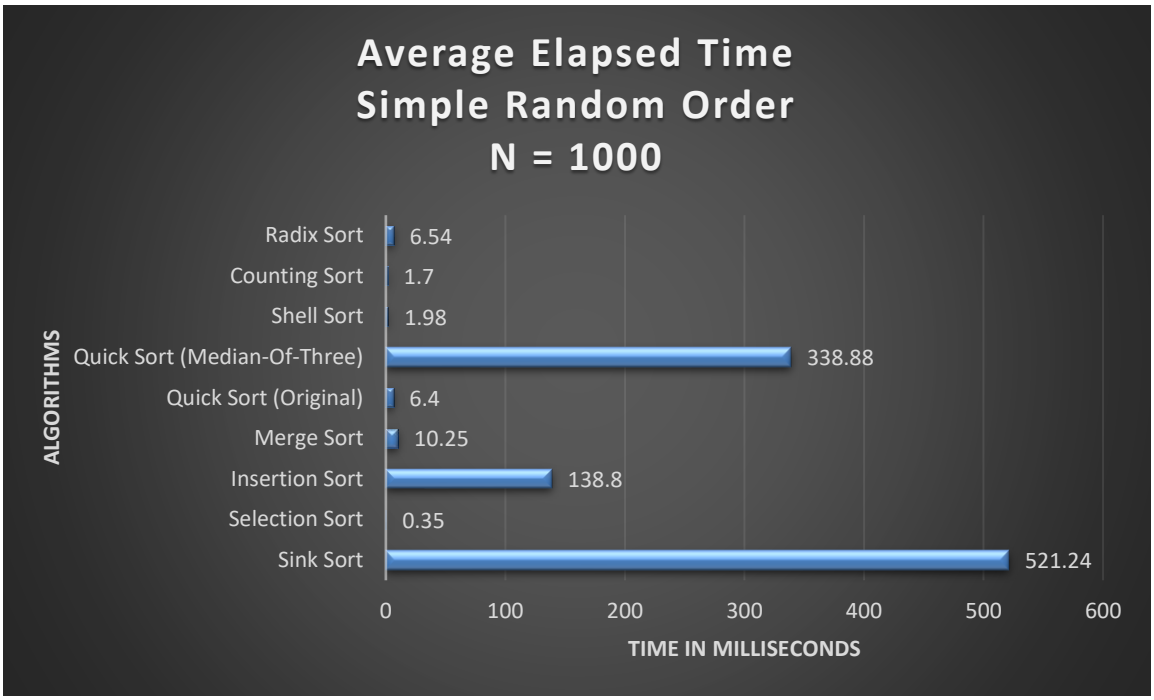


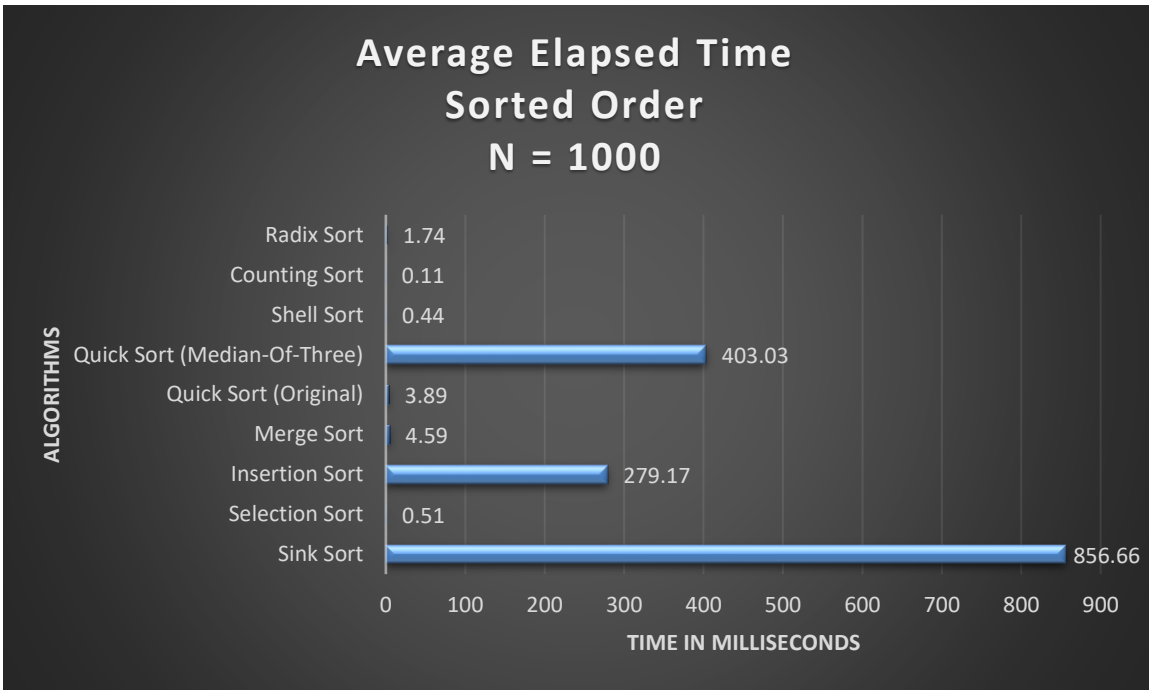
Table 2:

Algorithm	Number of Calls	Average Elapsed Inclusive Time	Value of N	Type of Execution
Sink Sort	1	521.24ms	1000	Random Order
Selection Sort	2000	0.35ms	1000	Random Order
Insertion Sort	1	138.80ms	1000	Random Order
Merge Sort	1999	10.25ms	1000	Random Order
Quick Sort (Original)	1	6.40ms	1000	Random Order
Quick Sort (Median-Of-Three)	1	338.88ms	1000	Random Order
Shell Sort	1	1.98ms	1000	Random Order
Counting Sort	1	1.70ms	1000	Random Order
Radix Sort	1	6.54ms	1000	Random Order
Sink Sort	1	856.66ms	1000	In Order
Selection Sort	2000	0.51ms	1000	In Order
Insertion Sort	2	279.17ms	1000	In Order
Merge Sort	1999	4.59ms	1000	In Order
Quick Sort (Original)	1	3.89ms	1000	In Order
Quick Sort (Median-Of-Three)	1	403.03ms	1000	In Order
Shell Sort	1	0.44ms	1000	In Order
Counting Sort	1	0.11ms	1000	In Order
Radix Sort	1	1.74ms	1000	In Order
Sink Sort	1	866.59ms	1000	Reversed Order
Selection Sort	2000	0.28ms	1000	Reversed Order
Insertion Sort	2	76.95ms	1000	Reversed Order
Merge Sort	1999	5.96ms	1000	Reversed Order
Quick Sort (Original)	1	2.10ms	1000	Reversed Order
Quick Sort (Median-Of-Three)	1	66.64ms	1000	Reversed Order
Shell Sort	1	1.03ms	1000	Reversed Order
Counting Sort	1	0.01ms	1000	Reversed Order
Radix Sort	1	1.47ms	1000	Reversed Order
Sink Sort	1	862.96ms	1000	10% Not in Order
Selection Sort	2000	0.19ms	1000	10% Not in Order
Insertion Sort	2	593.39ms	1000	10% Not in Order
Merge Sort	1999	13.87ms	1000	10% Not in Order
Quick Sort (Original)	1	8.52ms	1000	10% Not in Order
Quick Sort (Median-Of-Three)	1	257.81ms	1000	10% Not in Order
Shell Sort	1	8.49ms	1000	10% Not in Order
Counting Sort	1	0.24ms	1000	10% Not in Order
Radix Sort	1	11.80ms	1000	10% Not in Order

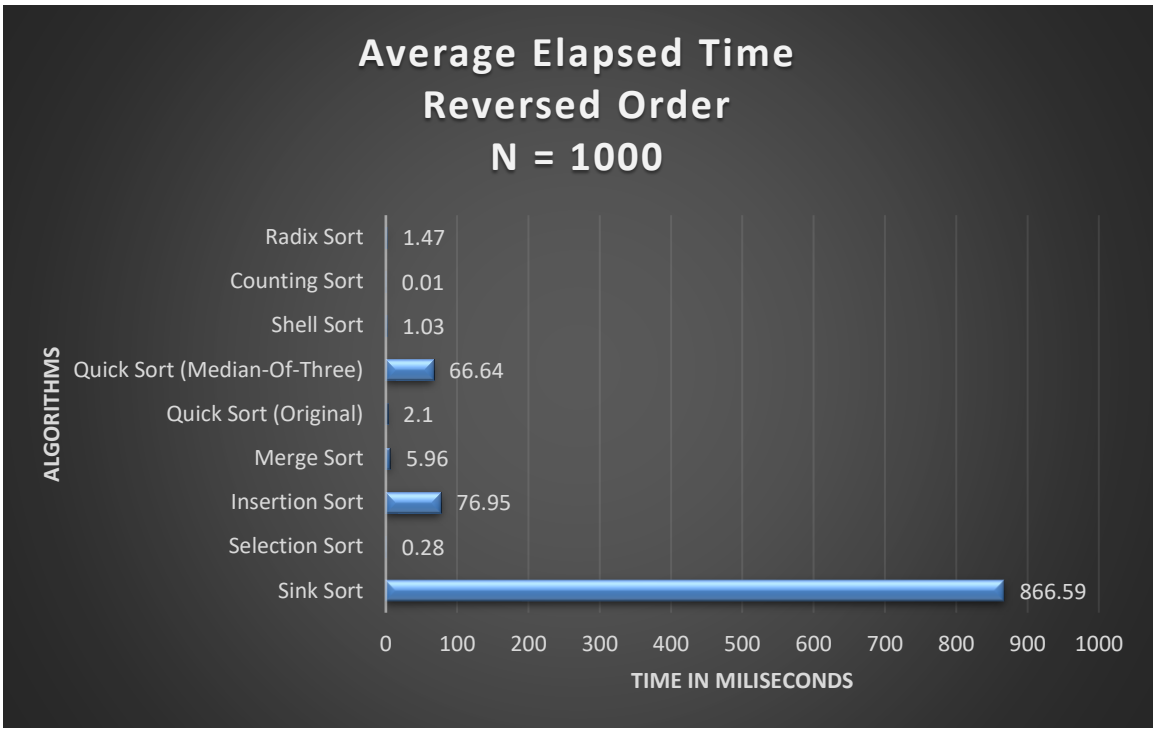
Graph 2.1:



Graph 2.2:



Graph 2.3:



Graph 2.4:

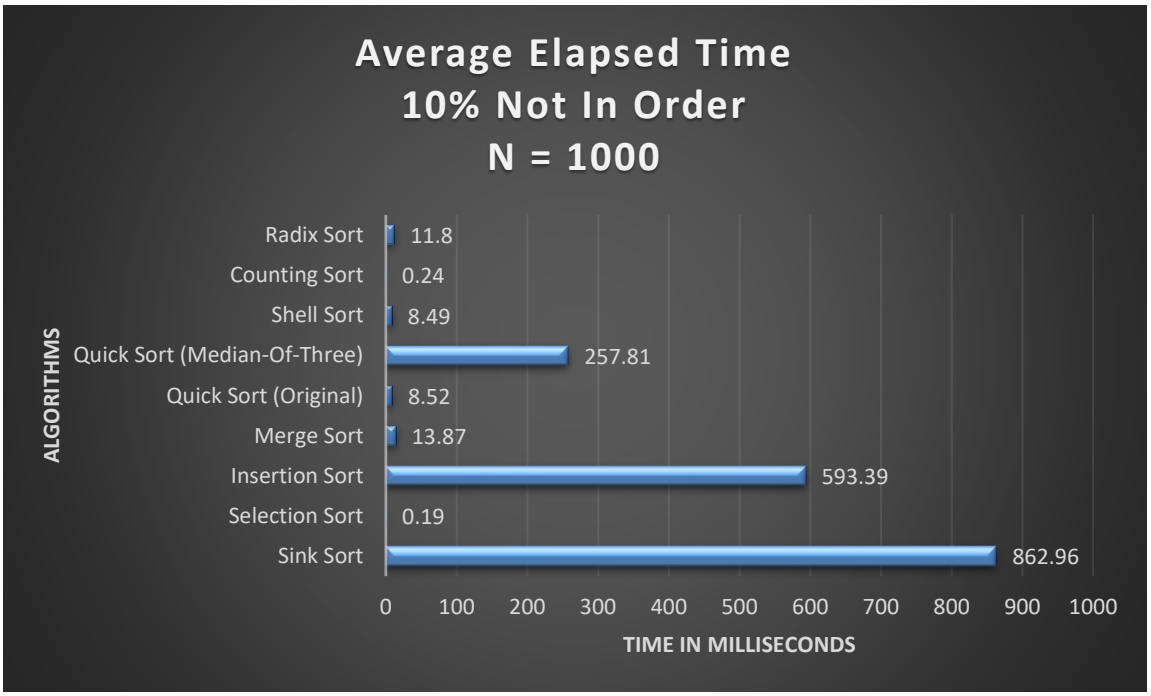
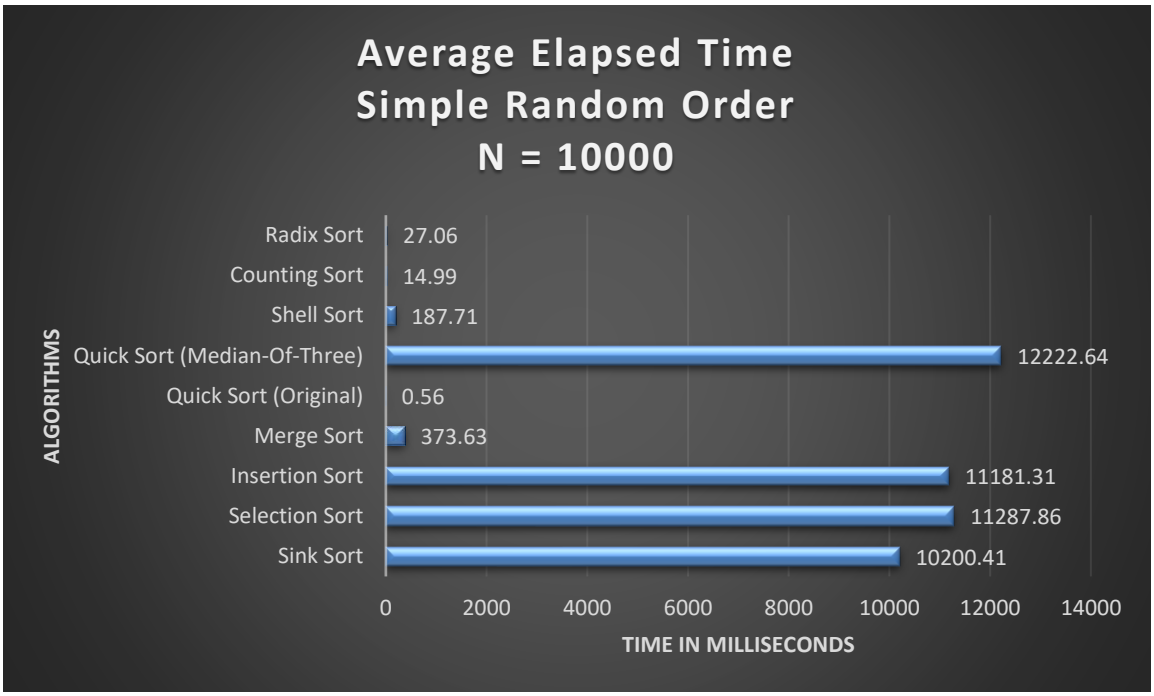


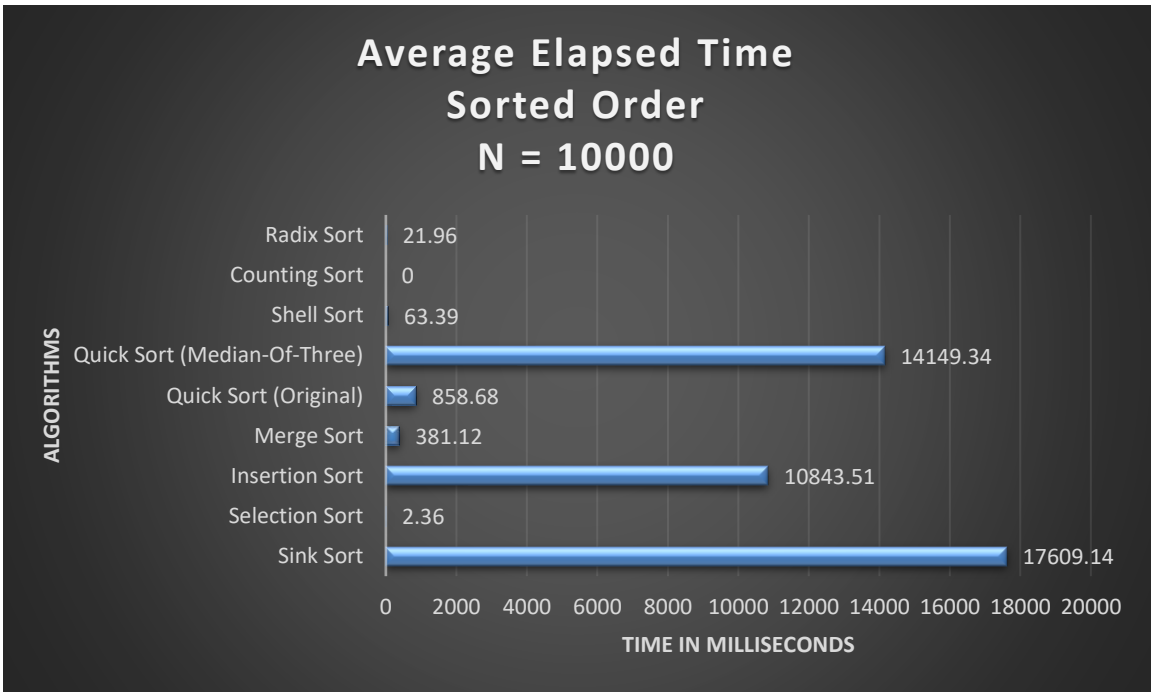
Table 3:

Algorithm	Number of Calls	Average Elapsed Inclusive Time	Value of N	Type of Execution
Sink Sort	1	10200.41ms	10000	Random Order
Selection Sort	2801	11287.86ms	10000	Random Order
Insertion Sort	1	11181.31ms	10000	Random Order
Merge Sort	19999	373.63ms	10000	Random Order
Quick Sort (Original)	1	0.56ms	10000	Random Order
Quick Sort (Median-Of-Three)	1	12222.64ms	10000	Random Order
Shell Sort	1	187.71ms	10000	Random Order
Counting Sort	1	14.99ms	10000	Random Order
Radix Sort	1	27.06ms	10000	Random Order
Sink Sort	1	17609.14ms	10000	In Order
Selection Sort	20000	2.36ms	10000	In Order
Insertion Sort	1	10843.51ms	10000	In Order
Merge Sort	19999	381.12ms	10000	In Order
Quick Sort (Original)	1	858.68ms	10000	In Order
Quick Sort (Median-Of-Three)	1	14149.34ms	10000	In Order
Shell Sort	1	63.39ms	10000	In Order
Counting Sort	1	0.00ms	10000	In Order
Radix Sort	1	21.96ms	10000	In Order
Sink Sort	1	8593.89ms	10000	Reversed Order
Selection Sort	2858	10180.83ms	10000	Reversed Order
Insertion Sort	1	12447.47ms	10000	Reversed Order
Merge Sort	19999	674.67ms	10000	Reversed Order
Quick Sort (Original)	1	766.91ms	10000	Reversed Order
Quick Sort (Median-Of-Three)	1	12551.39ms	10000	Reversed Order
Shell Sort	1	177.46ms	10000	Reversed Order
Counting Sort	1	2.21ms	10000	Reversed Order
Radix Sort	1	16.25ms	10000	Reversed Order
Sink Sort	1	56258.50ms	10000	10% Not in Order
Selection Sort	20000	2.41ms	10000	10% Not in Order
Insertion Sort	1	15301.84ms	10000	10% Not in Order
Merge Sort	19999	529.65ms	10000	10% Not in Order
Quick Sort (Original)	1	985.35ms	10000	10% Not in Order
Quick Sort (Median-Of-Three)	1	14979.70ms	10000	10% Not in Order
Shell Sort	1	46.74ms	10000	10% Not in Order
Counting Sort	1	0.88ms	10000	10% Not in Order
Radix Sort	1	23.46ms	10000	10% Not in Order

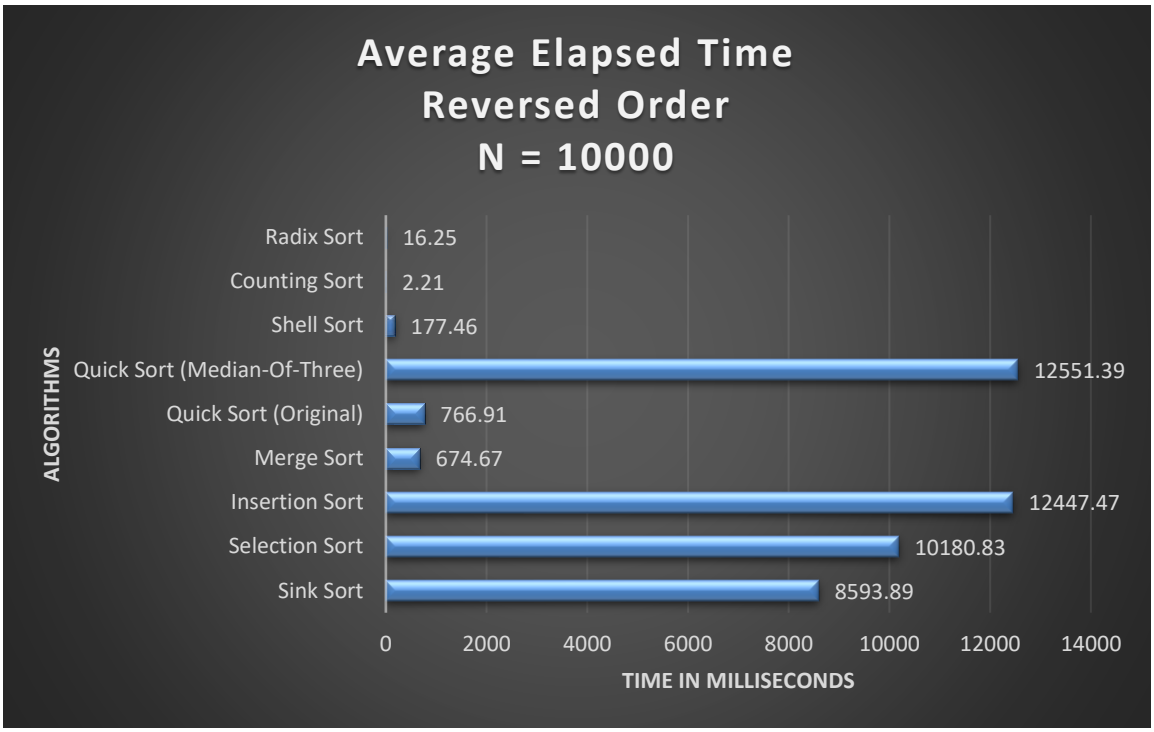
Graph 3.1:



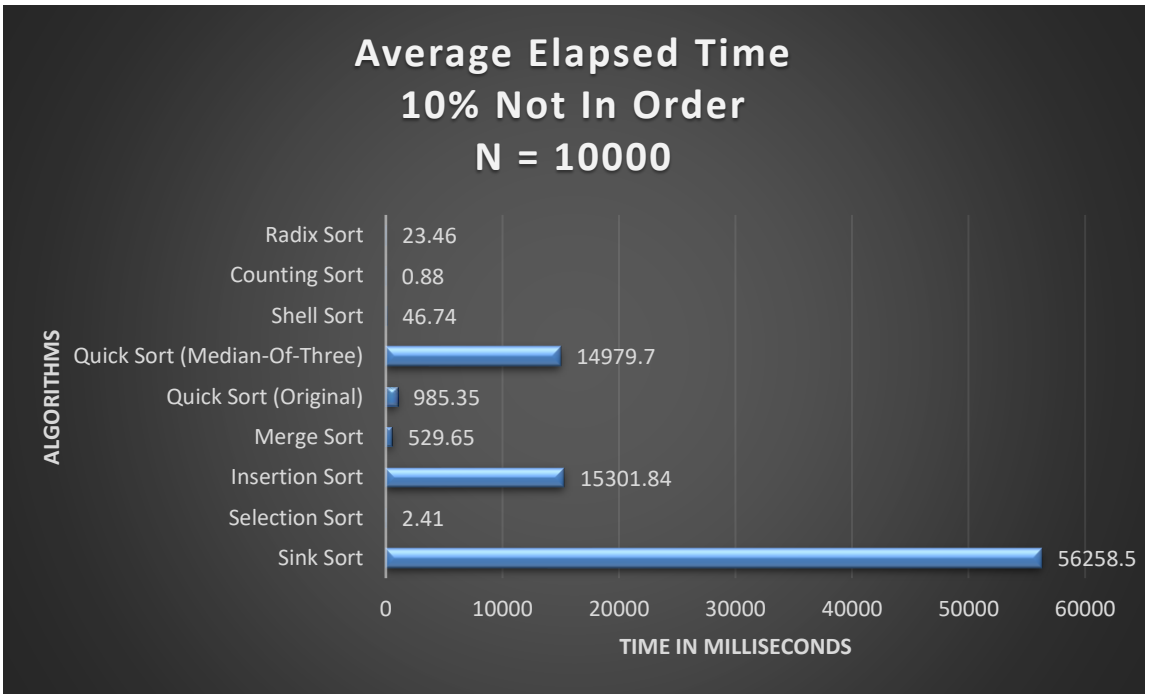
Graph 3.2:



Graph 3.3:



Graph 3.4:



References

Bell, R. (2008-2020). *Rob Bell*. Retrieved from A beginner's guide to Big O notation: https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/

Moore, K., Pilling, G., & Khim, J. (2016). *Brilliant*. Retrieved from Sorting Algorithms: https://brilliant.org/wiki/sorting-algorithms/

Wesley, A., Morgan, K., Sedgewick, R., & Bentley, J. (2010-2020). *Toptal*. Retrieved from Sorting Algorithms Animations: https://www.toptal.com/developers/sorting-algorithms