Recursive Sorting Research

Quick Sort Vs. Merge Sort

I. Introduction

Recursive sorting methods follow the divide and conquer method to sort through data sets. While sorting can be performed iteratively, using recursive methods brings down the temporal complexity significantly, in other words faster. This is because the algorithm breaks large data sets into smaller and smaller parts, and then sorts the smaller parts repeatedly until the whole data set is sorted.

There are multiple efficient recursive sorting methods, but the two common ones are the Quick sort and Merge sort. Quick sort was developed by Tony Hoare in 1960. This algorithm works by selecting a pivot point, and then all the smaller elements are put before the pivot and the larger elements after it. The data is broken down recursively and this methodology of partitioning is applied to each smaller portion of the data set. This is done until the data is sorted accordingly.

The Merge sort, introduced by Herman Goldstine and John Von Neumann in the 1940s, follows a different method. Here the data set is recursively broken down into many one element sets, and then every two elements are compared and sorted. These two elements are then taken, and compared to another two-element set, which then become a sorted 4-element set and so on. All the single elements are merged and sorted like this until it completes the whole data set.

The goal of this research is to assess the process time both algorithms take. The results of the tests, for each method, will be compared by the average value, the best and the worst value. The Quick sort method will generally perform better than Merge sort.

II. Argument

The reasoning behind the hypothesis that the Quick sort will be faster than Merge sort lies in the spatial complexity of the sorting methods. As seen with iterative sorting methods, in-place sorting creates faster processes then sorting into a new array. One of the key differences between quick sort and merge sort is the fact that Quick sort uses in-place sorting while merge sort uses a temporary array to sort the data set. Therefore, Merge sort uses extra space and memory to process datasets. If the systems CPU is busy, this will most probably cause larger processing time for merge sort as there will be more limited resources.

Merge sort is better with larger data sets and more stable compared to Quick sort, due to the fact that the worst-case scenario for Quick Sort is quadratic (O(N2)). Otherwise the both have a logarithmic relationship (O(N Log N)) when it comes to temporal complexity. However, Quick sort's worst case can be avoided by choosing the right pivot. A bad pivot can cause more comparisons then needed. This is avoidable by randomizing pivot choice, therefore minimizing the change of a worst-case scenario. Choosing a middle element as the pivot is also acceptable. Therefore, choosing a correct pivot point will minimize quick sort's worst-case scenario and making it as stable as merge sort.

III. Test Process

A test program was created to facilitate the research. The program runs 100 different sorts for each trial and records the time taken to process each sort. Each sort takes an array of 10 million random 6-digit integers and sorts them in ascending order. There are five trials for each sorting method. The first two trials are conducted under the condition where the computer is just running the test software and about 10% of the CPU usage. The third trial is conducted in similar condition except the network adapter is turned off. The last two trials were conducted with the CPU at 90% usage. This condition was attained by running several programs, including a YouTube stream, Spotify music, Excel, the computer game Witcher 3 at ultra-settings, and the test software. For the test, a Lenovo Legion laptop was used. The system configuration is below.

System Configuration		
Make:		**Lenovo**
Model:		**Provemce-5R1**
Processor:		**Intel Core i5 7300HQ**
RAM:		**DDR4 16GB**
Op. Sys.:		**Windows 10, 64 bit**
IDE:		**IntellJ IDEA 2019.1.13**
Language:		**Java 11**

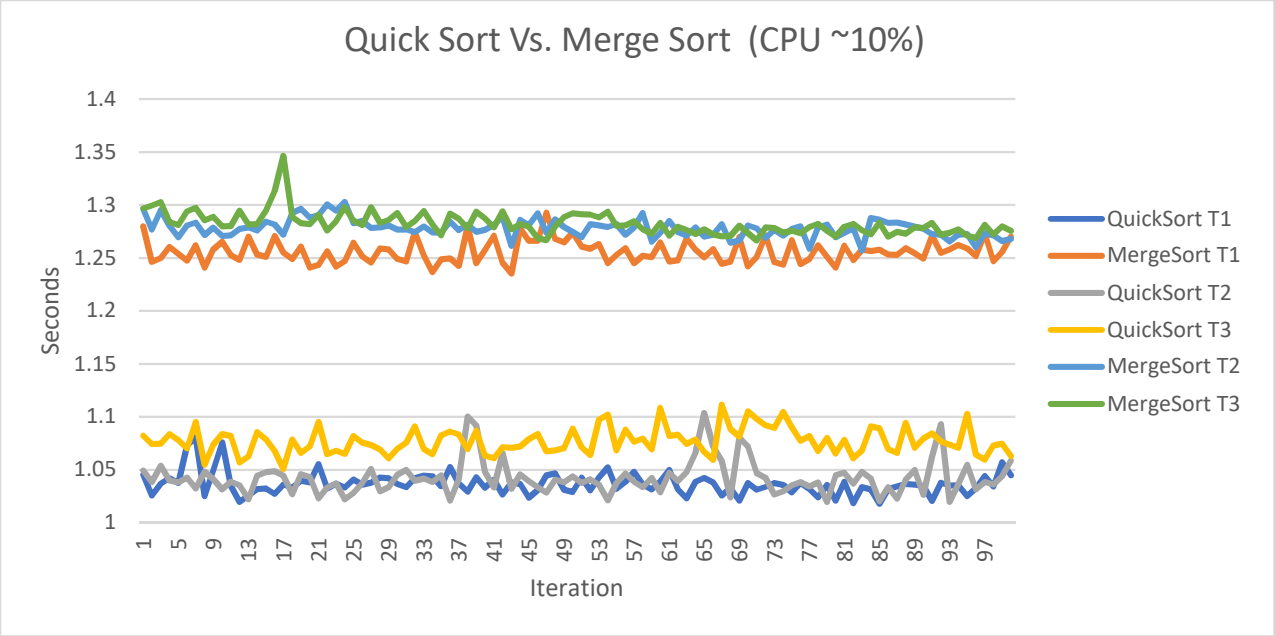
IV. Results

Quick Sort					
	CPU ~ 10% (Min Usage)			**CPU ~ 90% (Max Usage)**	
	T1	**T2**	**T3 (No Net)**	**T4**	**T5**
Average	1.03672	1.04237	1.07708	1.58402	1.99453
Best	1.01784	1.01946	1.05059	1.25011	1.86131
Worst	1.08068	1.10361	1.11154	2.25668	2.28525

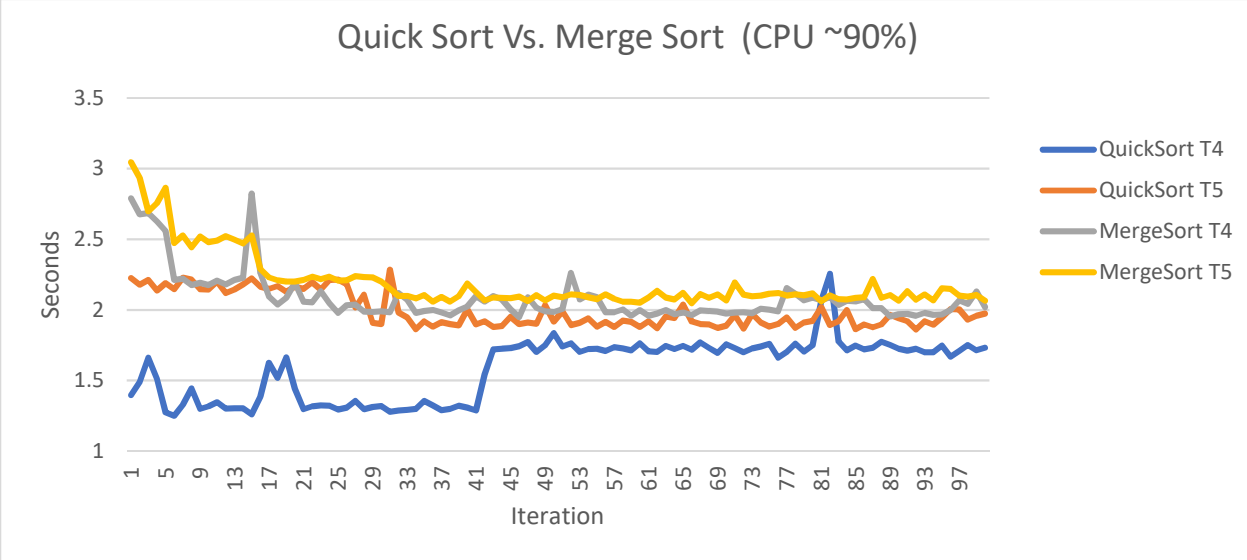
From the table above, it shows that the average processing time for Quick sort was about 1.05 seconds for the first 3 trials. The average went up, as expected, when the CPU was at 90%. Also, it shows how vastly different the worst cases were at 10% and 90% of the CPU usage. This again was also expected.

Merge Sort					
	CPU ~ 10% (Min Usage)			**CPU ~ 90% (Max Usage)**	
	T1	**T2**	**T3 (No Net)**	**T4**	**T5**
Average	1.25587	1.27808	1.28247	2.08485	2.19444
Best	1.23514	1.25695	1.26649	1.94923	2.04742
Worst	1.29280	1.30289	1.34641	2.82487	3.04584

Merge sort average was about 0.2 seconds more than the average of Quick sort, at 10% CPU usage, and much greater when the CPU was max usage. The best and worst case for merge sort were also greater than that of Quick sort.



When comparing just the trials that were conducted at low usage, it clearly shows the processing time difference between the two. What is interesting to see is that Trail 3 for both algorithms, with the network interface turned off, was higher when it was on. This could be due to the fact that some system background processes were trying to connect to the network continuously, and therefore increasing the CPU usage.

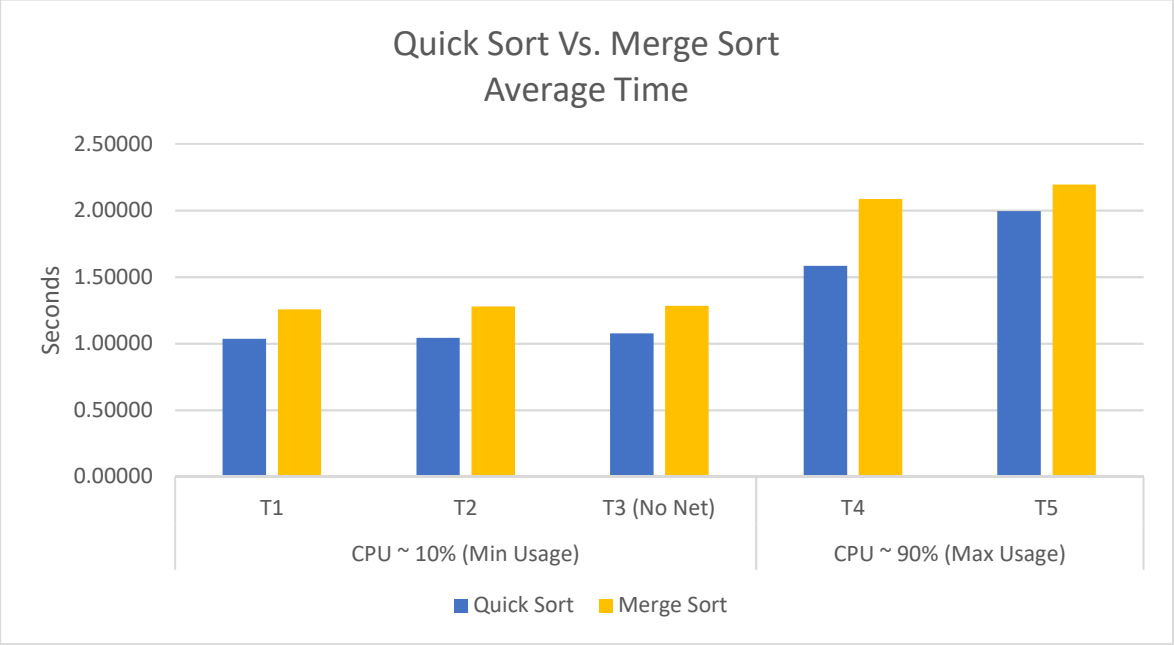


When running the test with high CPU usage, on average the Quick sort was faster than Merge sort again. There is some fluctuation where the processing time cross each other but in

general Quick sort was faster. It can be observed that Quick sort Trail 4 starts lower and then jumps up nearer to the rest. This could be due to a change in CPU processing due to another program.

V. Conclusion

The following bar graph summarizes the average processing time for each trial and sorting method.



As seen from the results, on average the Quick sort algorithm was faster than the Merge sort algorithm. Due to Merge sorts use of an additional array, it gives a higher spatial complexity compared to Quick sorts in-place sort. For the test program, a pivot was not chosen at random but at the middle of the data set. A random pivot might have faster results, but even then, Quick sort was faster.

Even though the results show that the Quick sort was faster, the difference is not substantial. This is because both algorithms are both efficient and similar. This is way they are both common and popular sorting methods used in programs. Merge sort is more stable and consistent, especially when data sets get bigger, and a preferred method for Linked lists. Quick

sort however, is faster for arrays and has a better spatial complexity and its worst-case complexity can be avoided by using randomized pivots.

For future tests, it might be better to test different data sets and sizes. Giving more scenarios and conditions would give more robust results and therefore a better comparison. In conclusion, the test program shows that Quick sort is faster than Merge sort.