Austin Bolstridge

COP3530 – Data Structures 1

4/18/15

Merge sort vs. Quicksort – A real world analysis

Introduction

In classes, we're taught that, theoretically, merge sort and quicksort's complexity is the same. However, how does that relate to real-world performance, and how different is their performance? After all, Merge sort has the overhead of extra memory allocation – how does that affect our performance? In this paper, we will be exploring the performance difference of merge sort versus quicksort on a random, unsorted file of integers.

Setup and Environment

For the loading and sorting of integers, we wrote custom quicksort and merge sort programs, called quicksort.c and mergesort.c respectively. These two programs take two command line arguments – the number binary file generated by genNums, and the amount of numbers to take from that file.

The results were generated on a Lenovo Thinkpad Twist (model 3347-2GU) running Mac OS X 10.10. The laptop has an i7-3517U with 8GB of on-board DDR3-1333 RAM, compiled with Apple LLVM 6.0, based on libraries for OS X 10.9. All compilation was done with all errors and warnings turned on, and with debug information saved to the binaries (gcc –Wall –g)

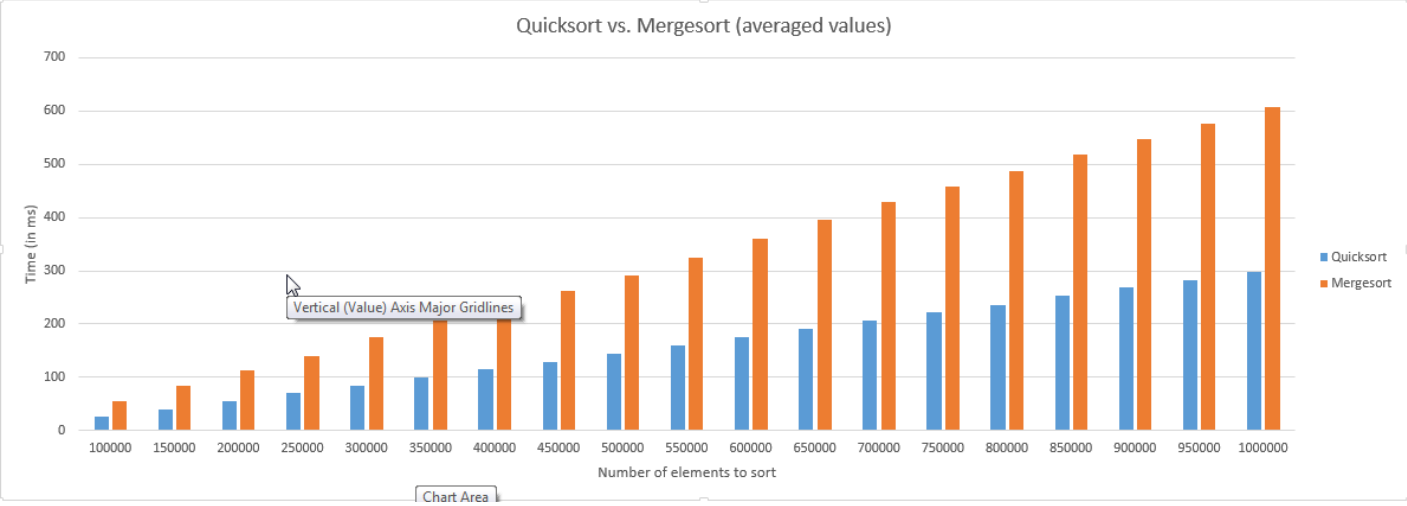
Testing

For gathering our statistics, we ran test.sh – this shell script runs though the quicksort and mergesort programs 19 times, starting with 100,000 numbers, and increasing in increments of 50,000 to 1,000,000. Before each run of each program, it generates a new numbers.bin file, in order to remove possible bias from an accidental semi-sorted or otherwise biased random number file. It then repeats this 10 times, and stores every run in the respective results file in a tab-delimited format – mergesort_results.txt for merge sort, quicksort_results.txt for quicksort.

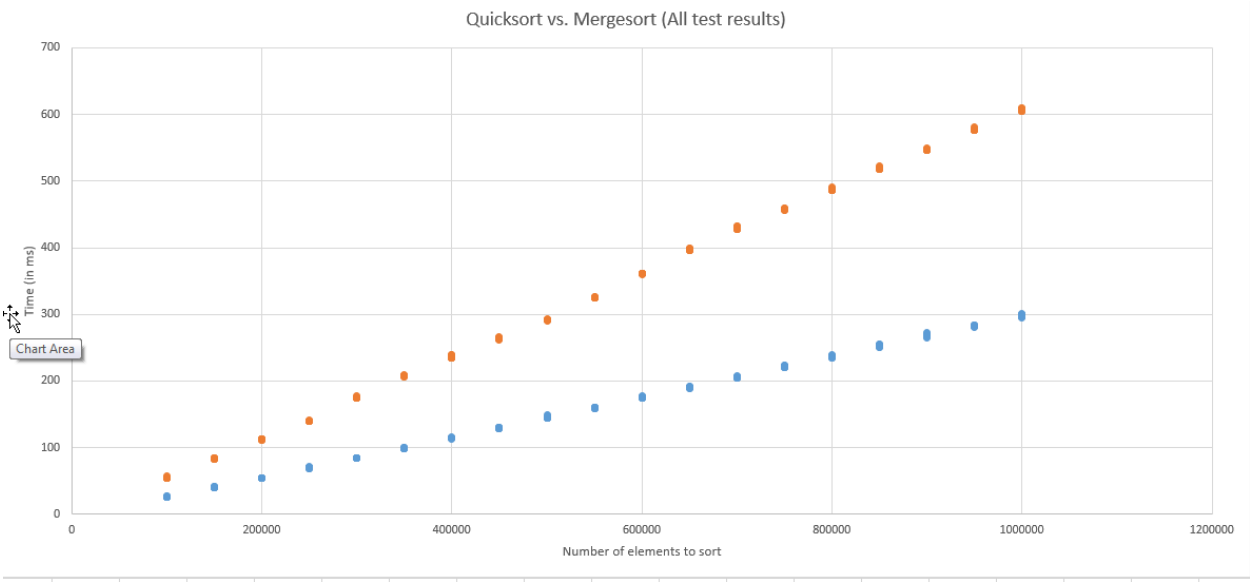
After we ran the test, we imported the data into Excel, and averaged each common run where the number of numbers and the algorithm were the same. We then created a histogram for the averaged data, and a scatter plot for all data.

Analysis

Initial analysis of the data suggested that the quicksort algorithm was indeed faster than the merge sort algorithm. Further analysis of the data proved this suggestion.



As you can see, as the data set grows bigger, on average, quicksort is more and more performant, and the difference between quicksort and merge sort is more apparent. Towards the end, there's a difference of nearly 300ms – almost one third of a second.



This statement continues to be proven, even when we take into account all of the data points. In fact, there are no substantial outliers, and every data point collected for quicksort was faster when compared to merge sort.

Conclusion

When we take a pseudo-random data set, and sort it using both merge sort and quicksort, quicksort is truly the faster choice. As an added bonus, in theory, the memory requirements of quicksort are much less than merge sort – however, that particular data point was not tracked.