Team Shuffle

Alex Vidal: avidal@csu.fullerton.edu

Project 2: Sorting and Time Differences Within the Same Order

The same caveat that was given in project 1 must be given here: when discussing the efficiency of these algorithms we are assuming that the size of the words is immaterial. Any *n's* in the efficiency statements refer only to the *number* of words in the data set. The raw data is in data.csv and was created by tee'ing the experiment's output. The file dataAnalysis.xlsx was made with Excel and contains various bits of computation and graphing. The experiment took roughly 15 minutes to run on my terrible laptop.

The two top-level algorithms covered in this project are Merge Sort and Quick Sort (hereafter referred as MS and QS, respectively). MS is based on the idea of leveraging the efficiency of merging two data sets that are independently sorted into a larger, sorted, set. This merge has an efficiency of O(n), so the idea is to split the data set until you have sets of 1 element each and then merge the sets into progressively larger, sorted sets. This splitting and merging has efficiency of O(logn), so the efficiency of the algorithm as a whole is O(n*logn). What is interesting about merge sort is that its efficiency is stable, that is: O, Θ, and Ω are all n*logn.

QS is conceptually more difficult. The act that is being leveraged by quicksort is partitioning. A partitioning algorithm will place all values in the set that are smaller or equal to the partition point before the partition point and all values larger than the partition point after. These two partitions are not sorted amongst themselves, they are only sorted relative to the partition. QS will then recursively partition these smaller sets until each recursion reaches a set of only 1 element. At that point the total set is sorted. The efficiency of QS is not straightforward, though. If the partitions are roughly half the size of the data set then QS has efficiency O(n*logn). If, however, the partitions place all the elements to one side of the partition point you have effectively reduced the problem size only by 1 element, this case has efficiency O(n^2). The latter case occurs if the set is already sorted or mostly sorted, leading to a counter-intuitive situation where QS performs better on random data!

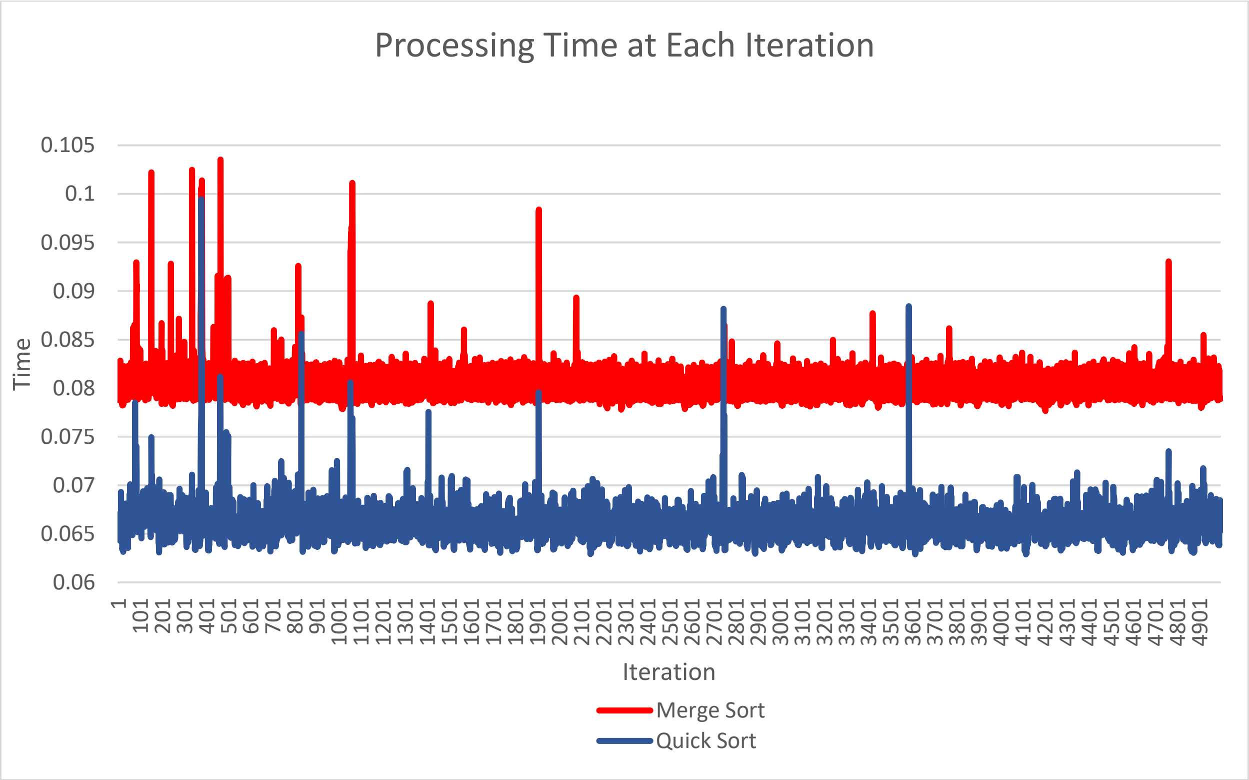
Ok, so analysis says that MS always has efficiency O(n*logn), but QS ranges from O(n*logn) to O(n^2). Prevailing wisdom (and really difficult analysis) states that QS will almost always outperform MS, despite their being (mostly) in the same order of efficiency. So this project is to get empirical evidence and show whether this assertion is correct.

The experiment is run like so: we have a set of roughly 100,000 words, we grab 50,000 of them, randomize their order, run MS and record the time it takes. We grab a new set of 50,000 words, randomize them, and this time run it through QS, recording its time as well. We repeat this process 5000 times and then look at the resulting time data. Randomizing the data makes the sets more realistic for typical scenarios. Running the algorithms back to back each iteration hopefully will keep lag spikes (mostly out of our control) from affecting one algorithm more than the other.

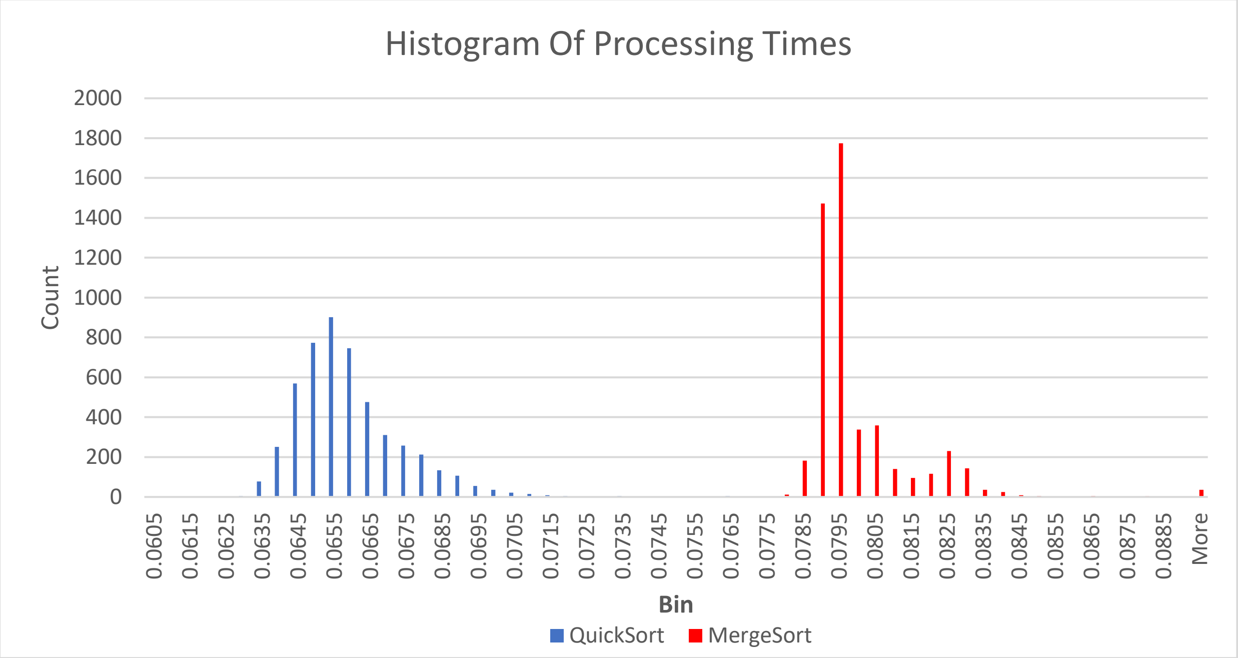
10,000 points of data is a little much for this report, so if you'd like to see it please reference dataAnalysis.xlsx. Below we have some summarizations:

	MergeSort	QuickSort
Mean From Code	0.0797571	0.0657995
Mean From Excel	0.079758052	0.0657995
Std From Code	0.00181967	0.0017934
Std From Excel	0.001819484	0.0017932
Total Time	398.7902581	328.99756
Min Time	0.0776473	0.0628876
Max Time	0.103536	0.0994566
Range	0.0258887	0.036569
Range/Mean	0.324594299	0.5557641

It seems like the analysis is correct. Mean and total time are both less for QS, but QS can range much further. Interestingly the standard deviation for both methods is roughly the same, but the range for QS is a much greater proportion of its mean than for MS. Here is a visual for time vs. iteration:



There's lots of interesting features in this graph. It clearly shows that the average time QS took to execute was almost always lower than MS and was pretty close to our calculated values. There are lots of time spikes throughout, wherever spikes line up, like iteration 500 or 1900, we might be able to explain it as a time of heavy load on the processor. In such times its feasible the OS would slice processor time in a way less favorable for the experiment, adding lag overall. There are some spikes in QS with no corresponding spike in MS (~2700 and 3600), which may be due to bad data sets pushing QS closer to the $O(n^2)$ territory, or poor time slicing. There are some spikes in MS that are not present in QS (~300), the only explanation I can think of is poor time slicing that only affected the MS algorithm. Also, QS seems in general to be noisier than MS.



This graph is very interesting for three reasons. 1) It reinforces that the average QS time is below the average MS time. 2) QS looks reasonably close to a normal distribution, just with a long tail to the right indicating it can take a relatively large range of values. 3) MS has a very strong peak at its mean with small counts in other bins. It really likes executing at a predictable time. The “more” bin does have some representation, but keep in mind that it contains values from .0885 up to .103, i.e. a range 50% of the size of what is being displayed.