Project Two Report

Team members:

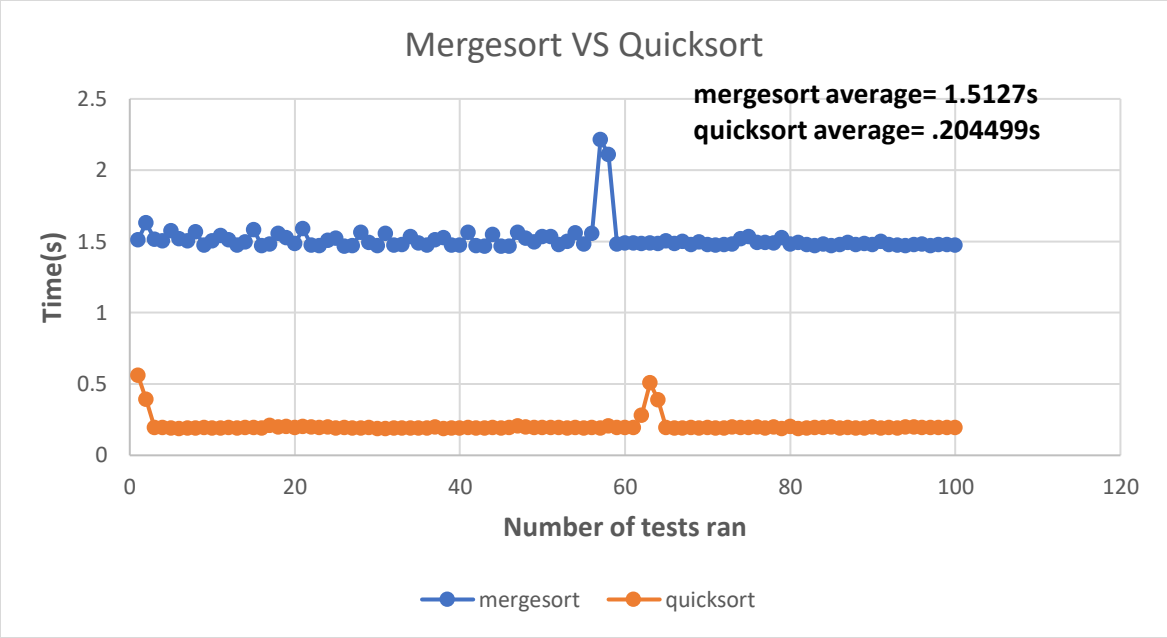
Areg Tevanyan atevanyan@csu.fullerton.edu

Chris Baroni baroni128@csu.fullerton.edu

For this project we wrote code to analyze the difference between Mergesort and Quicksort. From class we have learned that mergesort breaks down a list into smaller lists until there are only one element which means each individual sort is inherently sorted since it is only one item, and then it is put back together. We learned that a mergesort runs in O(n * logn) time, and in worst case, O(n * logn) again. Quicksort on the other hand, utilizes the hoare partition to separate the list in two using a pivot to compare against repeatedly. From class we have been taught that this is also on best case O(n * logn) however with Quicksort, the worst case is O(n^2). Judging from this we would expect that the Mergesort would consistently produce the better result, however that was not the case.

As seen from the graph below, mergesort average roughly 1.51 seconds per sort on a list of words 400,000 long. Seeing as this was the first test we ran I was quite impressed with the results. There were two outliers as seen in the graph which was not expected since we imagined that mergesort would run at nearly the same speed every time. Those outliers must have been caused by an extra load on the processor at that moment.

In contrast, Quicksort averaged .205 seconds on the same list of 400,000 words. Well, not exactly the same, since before each iteration of the test, the list of words was put in a random order. It is amazing that the quicksort ran so much faster every time when the worst case scenario for quicksort is O(n^2). Because the way the hoare partition runs, it is possible that quicksort actually runs faster than mergesort even though both cases are nlogn. Though according to the data not a single test took longer that a second. This shows that the slowest quicksort was still faster than the quickest mergesort. In the quicksort data we can see a few outliers at both the start and the middle which could be caused by the program choosing an “unlucky” pivot point that did not partition the list well.



According to our data, the standard deviation of the mergesort was 0.025, while the standard deviation of quicksort was 0.258. This means that the probability of an outlier for mergesort is much lower than that for quicksort. Therefore, mergesort runs at the expected efficiency more frequently which is consistent with what we are taught. However we should not forget, that even though mergesort may be more consistent, quicksort still runs much more efficiently. This can be seen by that fact that range of mergesort is roughly 1.4 seconds to 2.21 seconds, whereas the range for quicksort is 0.18 seconds to 0.56 seconds.

Overall it seems that quicksort is a better option between the two sorts, however this test was run on a list of only 400,000. This may seem big, but in the cases where a sort is running on a list of 400 million, the risk of getting too many “unlucky” pivots may not be worth the additional hours of time. Sometimes consistency is far more valuable.

Also….