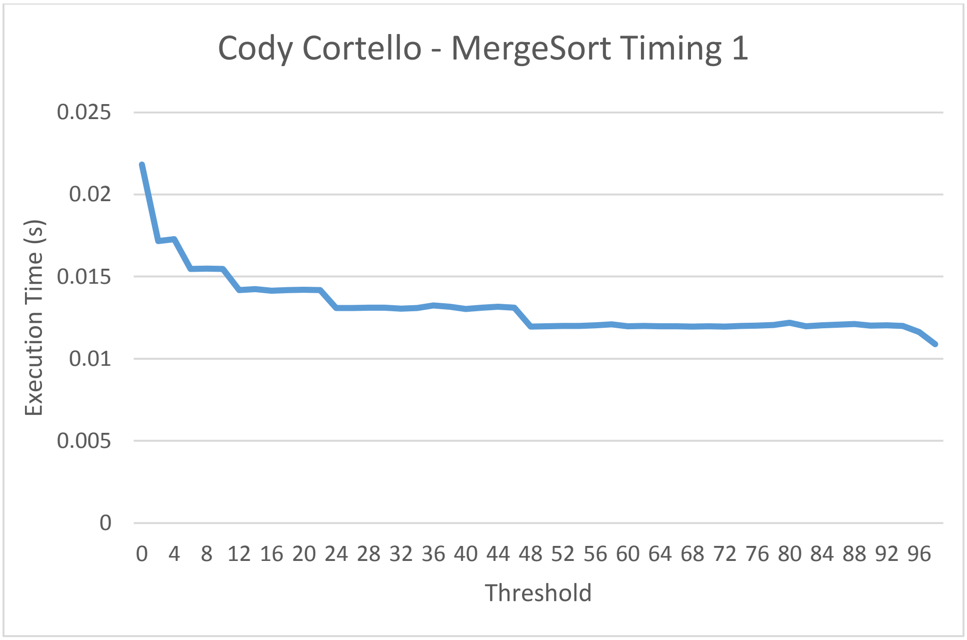
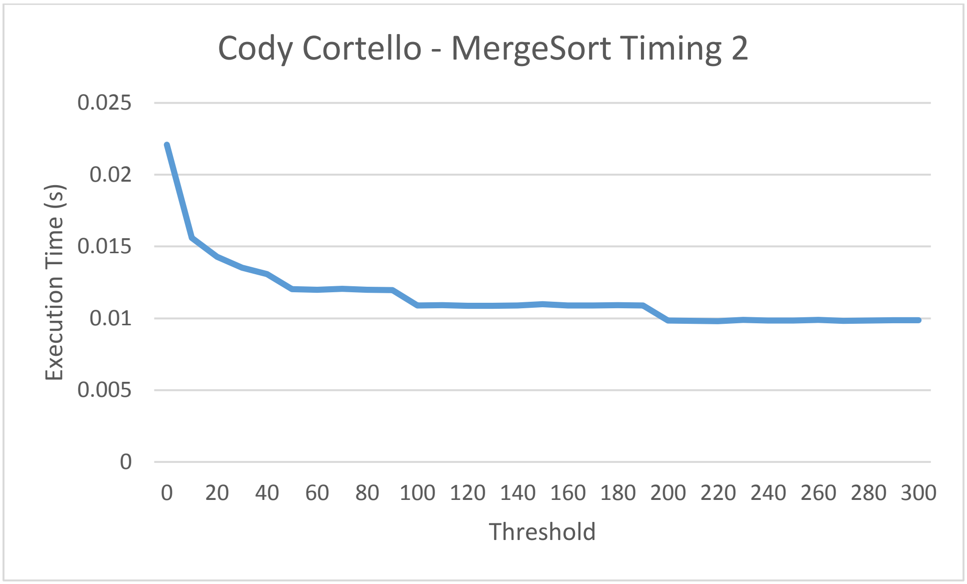
Cody Cortello
U0781604
CS 2420 – Summer 2014

Assignment 4 Analysis

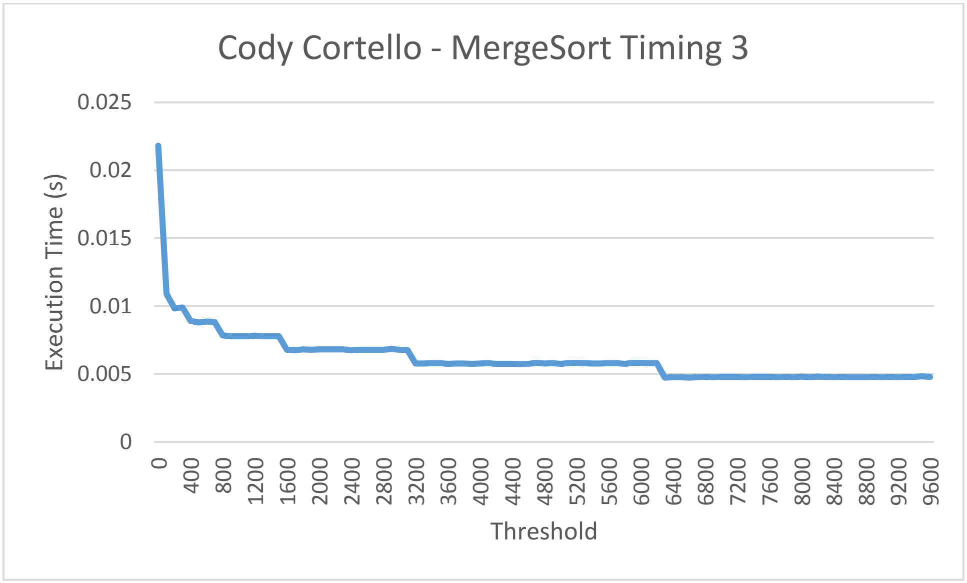
1. My programming partner is Casey Nordgran, but I will be submitting the source code for our project.
2. Casey is fun to work with, he understands Java and implementation very well, and he seems to have a great grasp of the concepts we're learning. I do plan on working with him again because he is a fantastic teammate as well as a knowledgeable one.
3. In finding the best threshold for MergeSort I copied the timing code and only had to slightly change the function timed. However, I find myself at a loss to explain my data in a meaningful way. I started testing an array of size 10,000 with 200 loops for a small threshold, between 0 and 96, and that experiment resulted in the following graph:



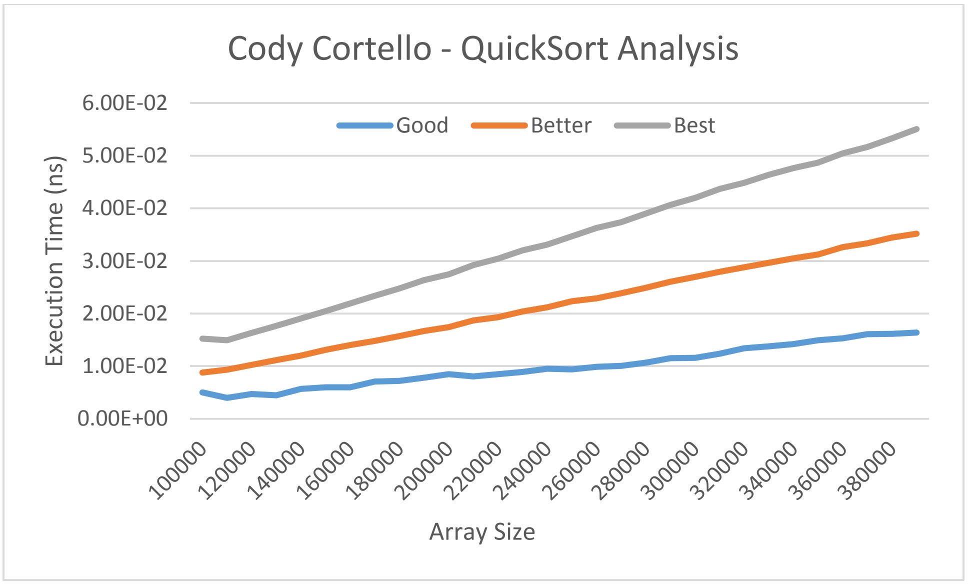
Which, as a monotonically decreasing function, clearly shows that my desired threshold is greater than the thresholds I tested. I increased my threshold size and experimented a second time, producing graph 2:



And again increased my tested thresholds and obtained a third graph:



4. To find the best pivot strategy I only had to slightly change the MergeSort timer class to implement the QuickSort algorithm and have it iterate though array sizes and manually change the pivot selection rather than iterating through a threshold and manually change the array size. Doing so with the three different pivot selection methods and a loop count of 100 resulted in the following graph:



Interestingly, this graph shows the exact opposite of what I would've expected. I would've imagined that the Best implementation would've run the fastest with the Better implementation in the middle and the Good implementation running the slowest. The only explanation I can devise for my data is that finding the pivot slows the computation down significantly depending on the implementation. As shown in the lab, creating a Random Object and especially using a seed and doing computation on that Object takes quite a bit longer than simply returning an integer, which would explain why the Better implementation took longer than the Good implementation. Furthermore, the Best implementation iterates through the entire array, making its operation O(N) rather than O(1) as is the case with the Good and Better implementations. I'd imagine that the reason that it's called the Best is because it minimizes the number of total comparisons and recursive calls made, and that the data I obtained reflects the execution time of the algorithm as a whole instead of just the comparisons.

5.

6. The running times do <u>not</u> reflect what I'd expected to see. First, the monotonic decreasing nature of my MergeSort implementation shows that a best threshold doesn't exist. I had expected the graph to dip to some minimum at the optimal threshold then rise once larger arrays were sorted with the less efficient InsertionSort algorithm. However, the data I obtained (and tested and debugged at great length) shows that no such threshold exists. Though I only tested one array size I know that such a threshold should exist for an array of any size as the threshold is simply the equality point of computation complexity between MergeSort and InsertionSort, and the nature of my graphs shows that a local minimum cannot exist since I tested all values from 0 to the array size 10,000. My graph shows that the complexity of my InsertionSort implementation is infuriatingly <u>less</u> than the complexity of my MergeSort implementation.

Second, it almost appears that the QuickSort algorithm is linear rather than linearithmic. This discrepancy is explained by the fact that a lot about the algorithm's exact execution time formula is unknown, e.g. all constants and linear overhead as applicable.

Furthermore, the array sizes which I tested might've not been sufficient to accurately portray the end behavior of the function as they were limited by the memory and performance efficiency of my laptop.

7. In total Casey and I spent about seventeen hours total on the assignment throughout the week.