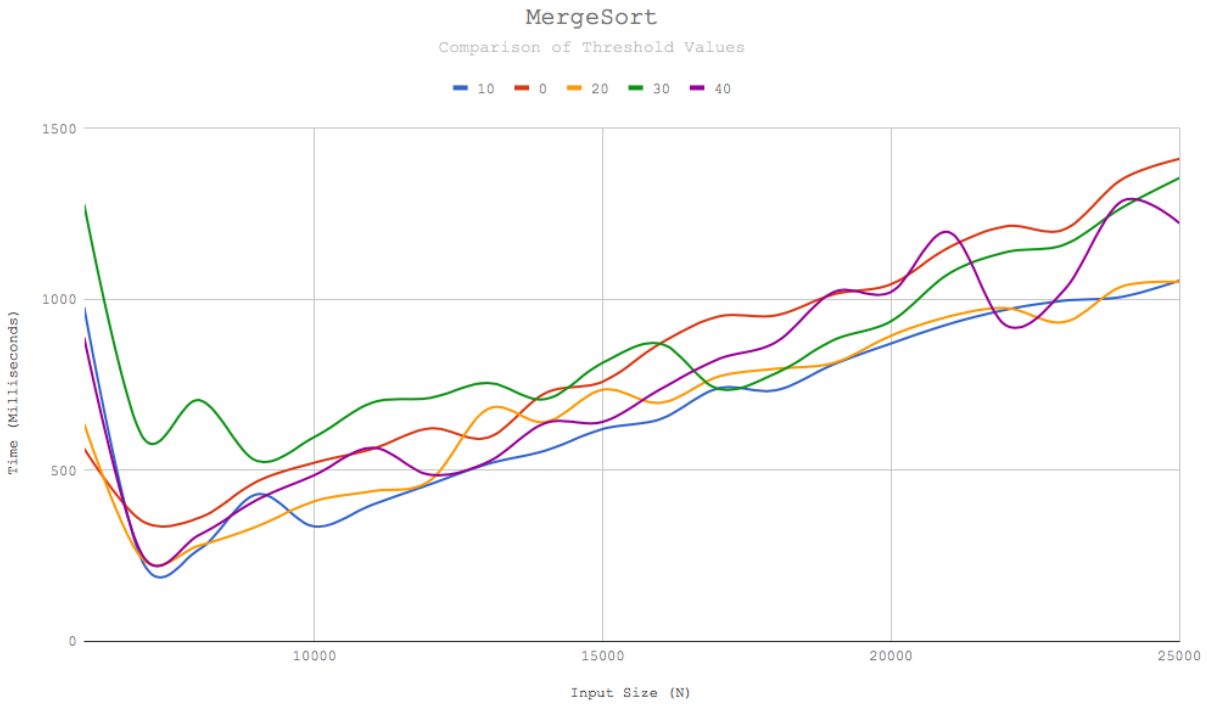
Assignment04

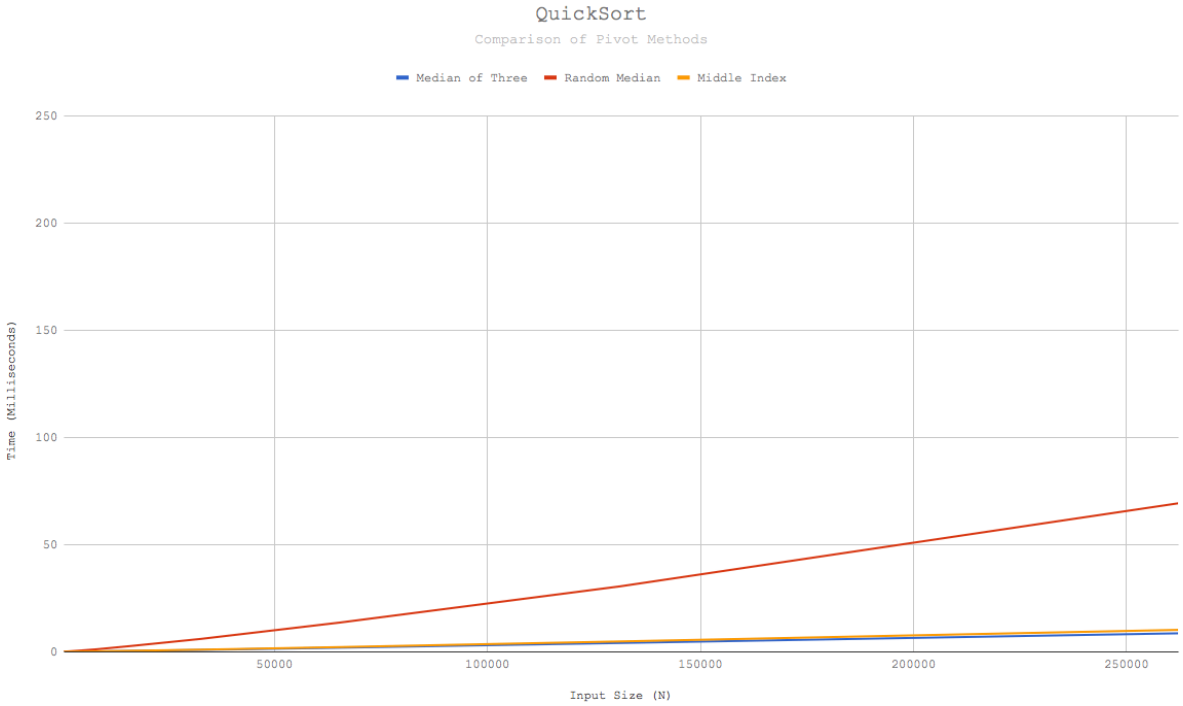
Quicksort vs. Mergesort Analysis

By Matt Josse & Junchen Zhang

Mergesort Threshold Experiment. Five threshold values were analyzed for the most accurate subarray size to switch from mergesort to insertion sort. As shown below, a value of 10 appears to be the ideal threshold. Of course, further experimentation may reveal a more efficient value but with the variance already present in differing processors and other factors, this seems to be a reasonable real-world estimate for the proper threshold.



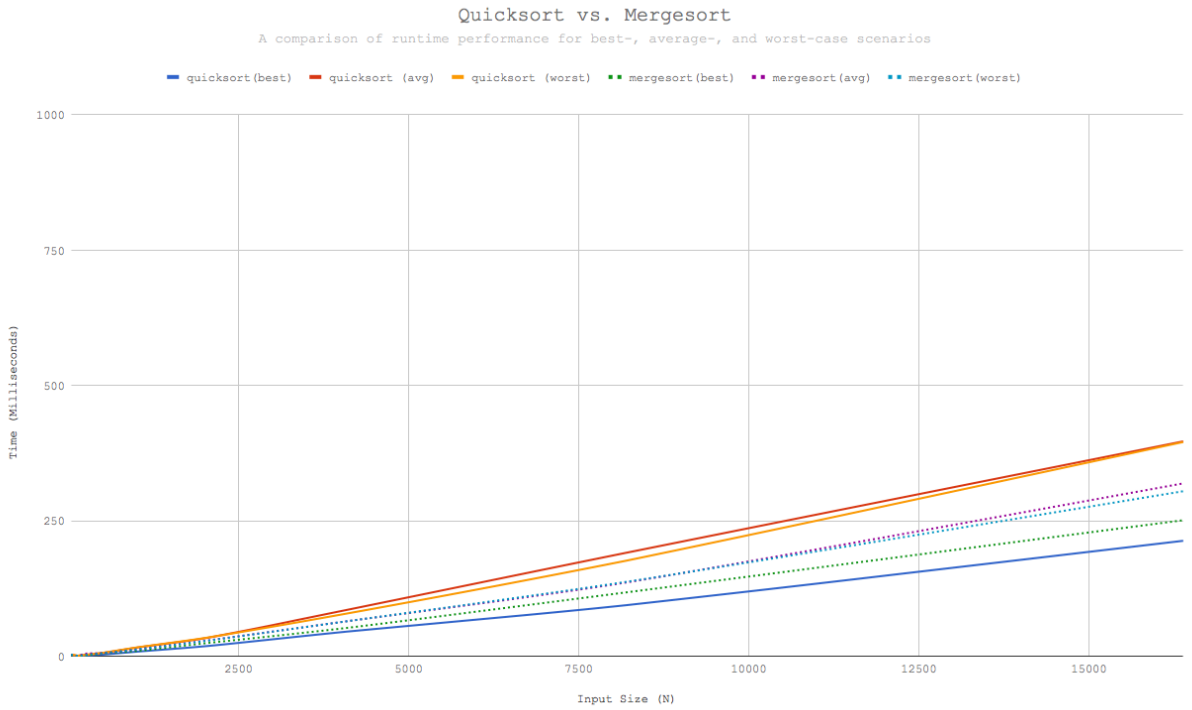
Quicksort Pivot Experiment. We analyzed three distinct pivot-choosing strategies over several input sizes: (1) Median-of-three, (2) Randomized Median-of-three, and (3) Middle index. As demonstrated by the graph below, we found the most effective of these methods to be the median-of-three. Note that all pivot analyses were conducted on equivalent average case scenarios (identical randomized lists). The input size is the number of Integer values sorted within an ArrayList object.



We believe the cost of running the randomization code accounts for the excessive time gap between the randomized median-of-three and the other two methods. Interestingly, randomizing the pivot selection decreases the effectiveness of quicksort (likely negating any benefit intended by randomization in the first place).

Mergesort vs. Quicksort Experiment. We performed the following analysis using a threshold value of 40 for the mergesort algorithm and the median-of-three pivot selection method for that of quicksort (as discussed above). In this case the input size remains the number of Integer values sorted within an ArrayList object but the sample size is relatively smaller than the quicksort experiment. This is because the quicksort succumbed to a stack overflow error at an approximate input size of 65000 for the worst-case scenario.

In this experiment each sort routine was measured for runtime performance using each of three conditions: best-, average-, and worst-case scenarios. As demonstrated in the graph below, the mergesort implementation appears to be the most efficient for all but the best-case scenario for quicksort. Of course, this could be attributed to a variety of factors, such as machine, power supply, and processor handling. Overall, mergesort is certainly the most efficient under the provided conditions.



Summary. The growth rates of both mergesort and quicksort do appear to align with the expected big-O behavior. A detailed analysis of these behaviors has been attempted in the individual experiment descriptions above. However, the most interesting outcome has been the substantial difference between the randomized median-of-three pivot selection and its standard counterpart (as discussed previously). Furthermore, it is interesting to note mergesort has either a more effective implementation or additional analyses of further input sizes is required to show a more accurate comparison.