1. Who are your team members?
Ben Trueman & Erik Poole (Me) – Ben should be uploading our zipped java files

2. Mergesort Threshold Experiment: Determine the best threshold value for which mergesort switches over to insertion sort. Your list sizes should cover a range of input sizes to make meaningful plots, and should be large enough to capture accurate running times. To ensure a fair comparison, use the same set of permuted-order lists for each threshold value. Keep in mind that you can't resort the same ArrayList over and over, as the second time the order will have changed. Create an initial input and copy it to a temporary ArrayList for each test (but make sure you subtract the copy time from your timing results!). Use the timing techniques we already demonstrated, and be sure to choose a large enough value of timesToLoop to get a reasonable average of running times. Note that the best threshold value may be a constant value or a fraction of the list size. Plot the running times of your threshold mergesort for five different threshold values on permuted-order lists (one line for each threshold value). In the five different threshold values, be sure to include the threshold value that simulates a full mergesort, i.e., never switching to insertion sort (and identify that line as such in your plot).
My partner Ben Trueman identified five different insertion sort threshold values, 0, 50, 100, 150, & 200. The '0' threshold should simulate running a mergesort through in its entirety and the linear dataset shown supports that. The other threshold values split into two subsets – 50, 100 & 150, 200. The subset with smaller threshold values more closely aligned with mergsesort running through to completion, which makes sense since Insertion sort is handling less of the overall sorting. All four non-0 strategies appear to have a similar effect on the algorithm, however. They all beat the '0' strategy by a small amount as the dataset grows. See figure 1 for the complete graph.

3. Quicksort Pivot Experiment: Determine the best pivot-choosing strategy for quicksort. (As in #2, use large list sizes, the same set of permuted-order lists for each strategy, and the timing techniques demonstrated before.) Plot the running times of your quicksort for three different pivot-choosing strategies on permuted-order lists (one line for each strategy).
I (Erik Poole) compared three pivots - one based on the first index, one based on the middle index, and one based on the median value of the first, middle, and last index. When comparing on a permuted-order (average case) list the sampling pivot strategy beat both of the others, but only narrowly. A close second was choosing the middle index and then choosing the first index came in last. See figure 2 for details.

4. Mergesort vs. Quicksort Experiment: Determine the best sorting algorithm for each of the three categories of lists (best-, average-, and worst-case). For the mergesort, use the threshold value that you determined to be the best. For the quicksort, use the pivot-choosing strategy that you determined to be the best. Note that the best pivot strategy on permuted lists may lead to O(N^2)

performance on best/worst case lists. If this is the case, use a different pivot for this part. As in #2, use large list sizes, the same list sizes for each category and sort, and the timing techniques demonstrated before. Plot the running times of your sorts for the three categories of lists. You may plot all six lines at once or create three plots (one for each category of lists).
We determined the sampling strategy to be the best way to choose a pivot for selection sort and a threshold of 100 to be most appropriate for mergesort. When we compare these algorithms across our best, average, and worst cases selection sort emerges as a clear winner. In the best and worst cases selection sort with a sampling based pivot will always choose the ideal pivot (since one of our three options for choosing the pivot is the center index) and that leads to incredibly rapid sorting. Even in the average case where selection sort is not guaranteed to choose the ideal pivot it still matched mergesort acting on our best and worst cases and greatly outperformed mergesort in its average case. See figure 3 for the data.

5. Do the actual running times of your sorting methods exhibit the growth rates you expected to see? Why or why not? Please be thorough in this explanation.
They do! While variation can be seen between different pivot strategies for selection sort and different thresholds for mergesort they all approximate O(nlog(n)) strategies and notably greatly outperform our plot of O(n^2) (see figure 3). It's interesting to note that a pure mergesort also ran as expected (O(nlog(n)), see figure 2) though I'm not convinced that we found a truly ideal threshold for when we swap to insertion sort, since I would expect a good threshold to consistently outperform O(nlog(n)). It's also interesting that selection sort utilizing a pivot based around the center index consistently outperformed selecting a pivot based on the first index for a randomized set. This was contrary to my expectations (I would have thought they would be the same) and this leads me to believe that some inversions are undone during every pass of quick sort even beyond the guarantee that both sides of the pivot will be divided correctly.

Figure 1 -

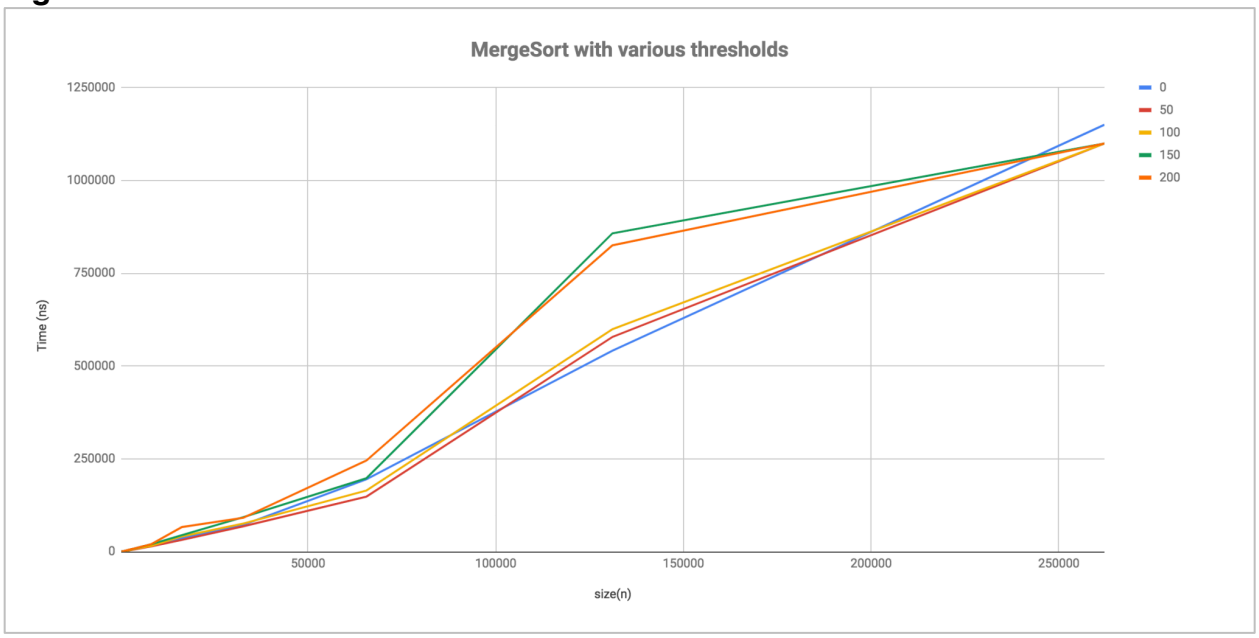


Figure 2 -

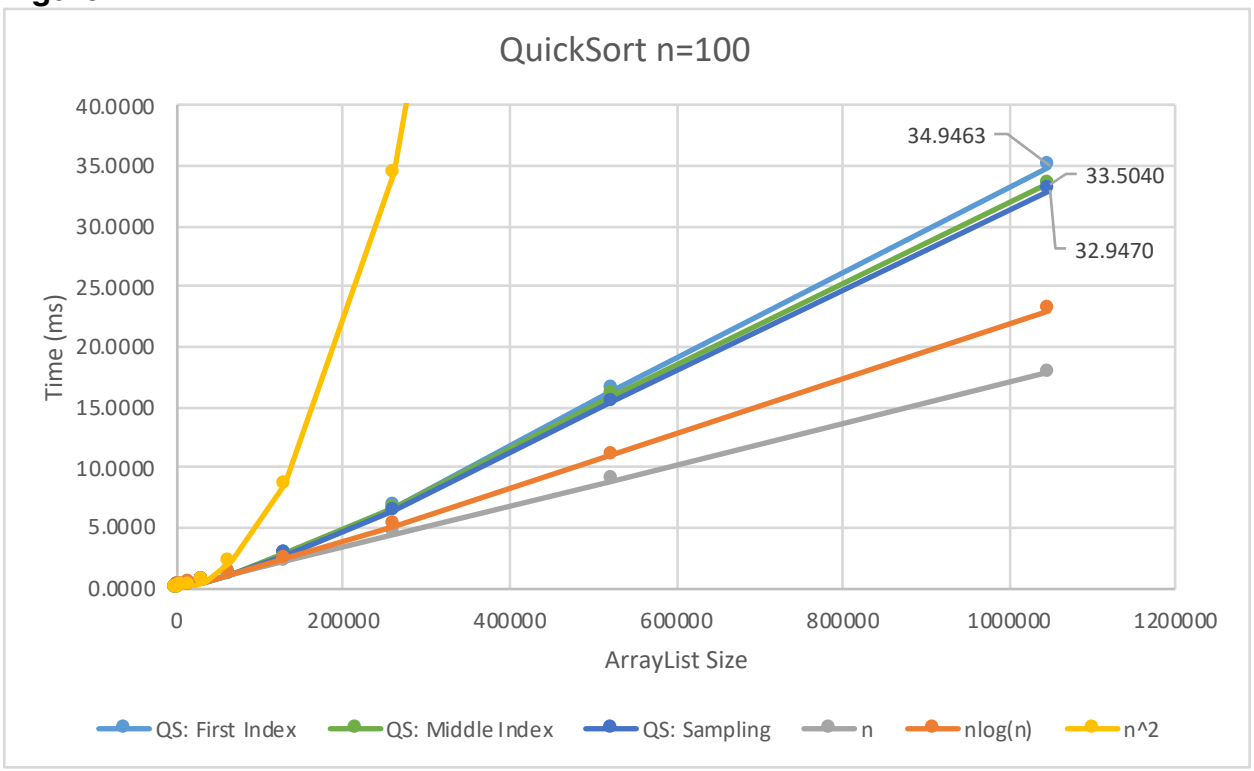
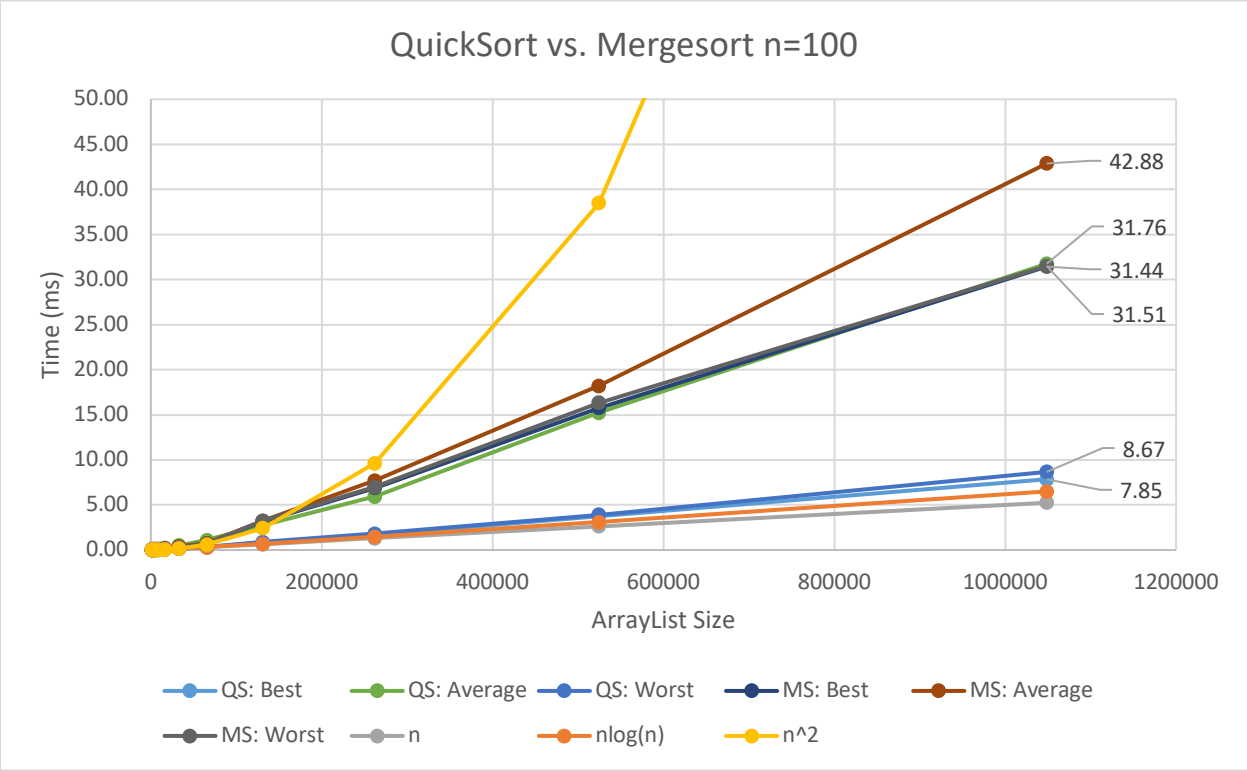


Figure 3 -


QuickSort vs. Mergesort n=100
Time (ms)
50.00
45.00
40.00
35.00
30.00
25.00
20.00
15.00
10.00
5.00
0.00
0
200000
400000
600000
800000
1000000
1200000
ArrayList Size
42.88
31.76
31.44
31.51
8.67
7.85
QS: Best
QS: Average
QS: Worst
MS: Best
MS: Average
MS: Worst
n
nlog(n)
n^2