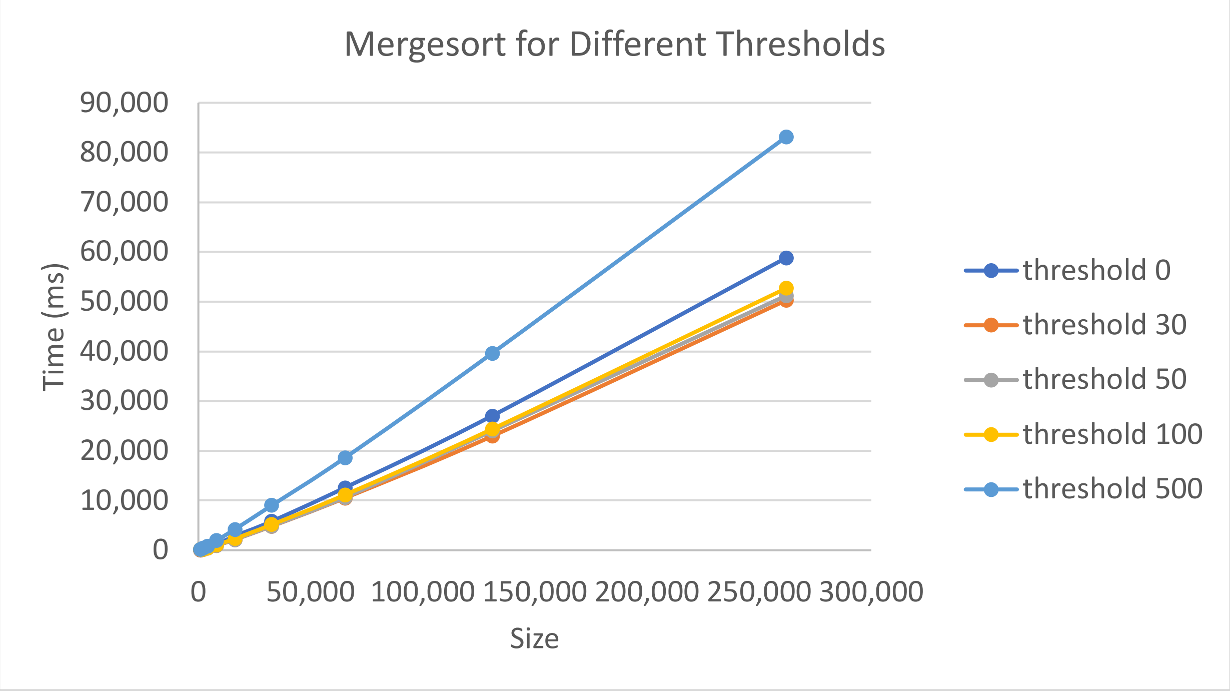
Assignment 4 Quicksort and Mergesort – Analysis Doc

U1426365 Jin-Ching Jeng

1. Who are your team members?
Chun-Hao Hsu

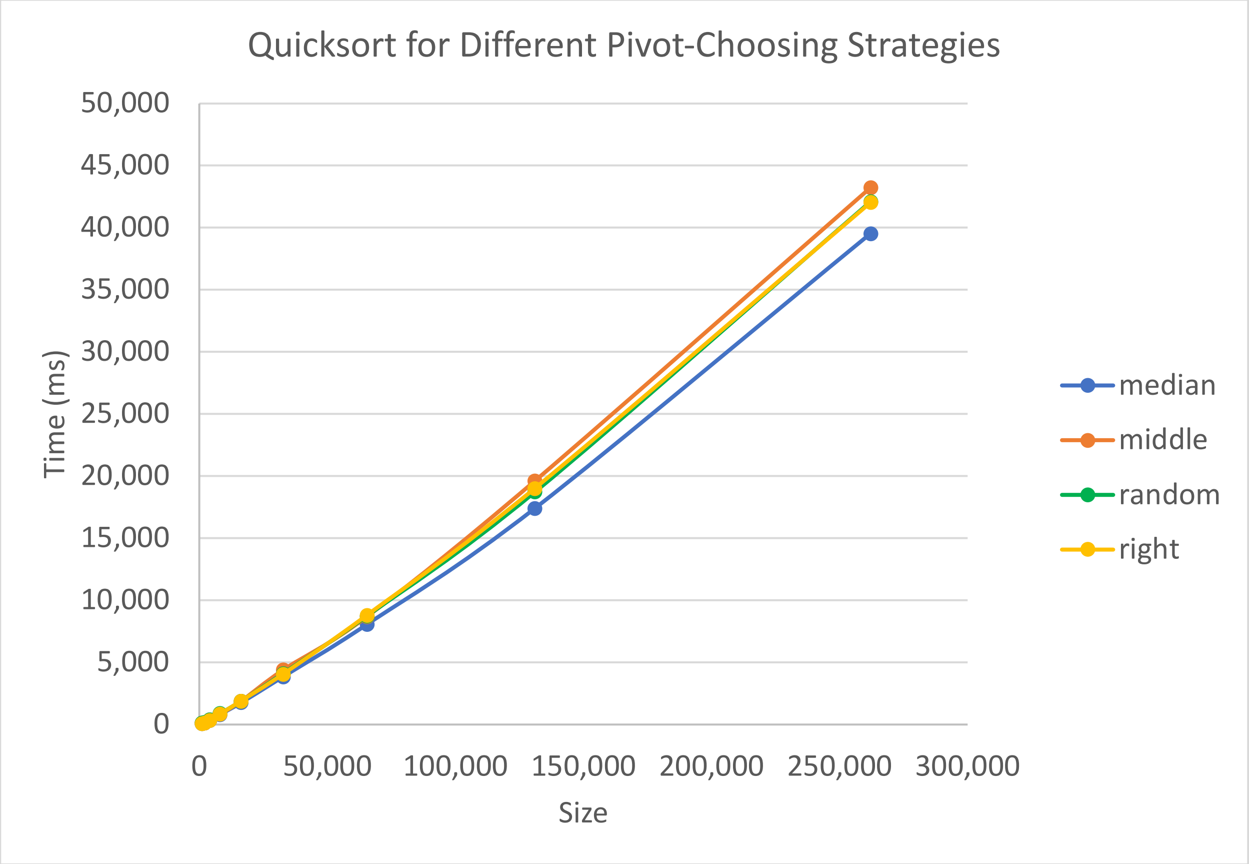
2. Mergesort Threshold Experiment: Determine the best threshold value for which mergesort switches over to insertion sort. Your list sizes should cover a range of input sizes to make meaningful plots, and should be large enough to capture accurate running times. To ensure a fair comparison, use the same 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).



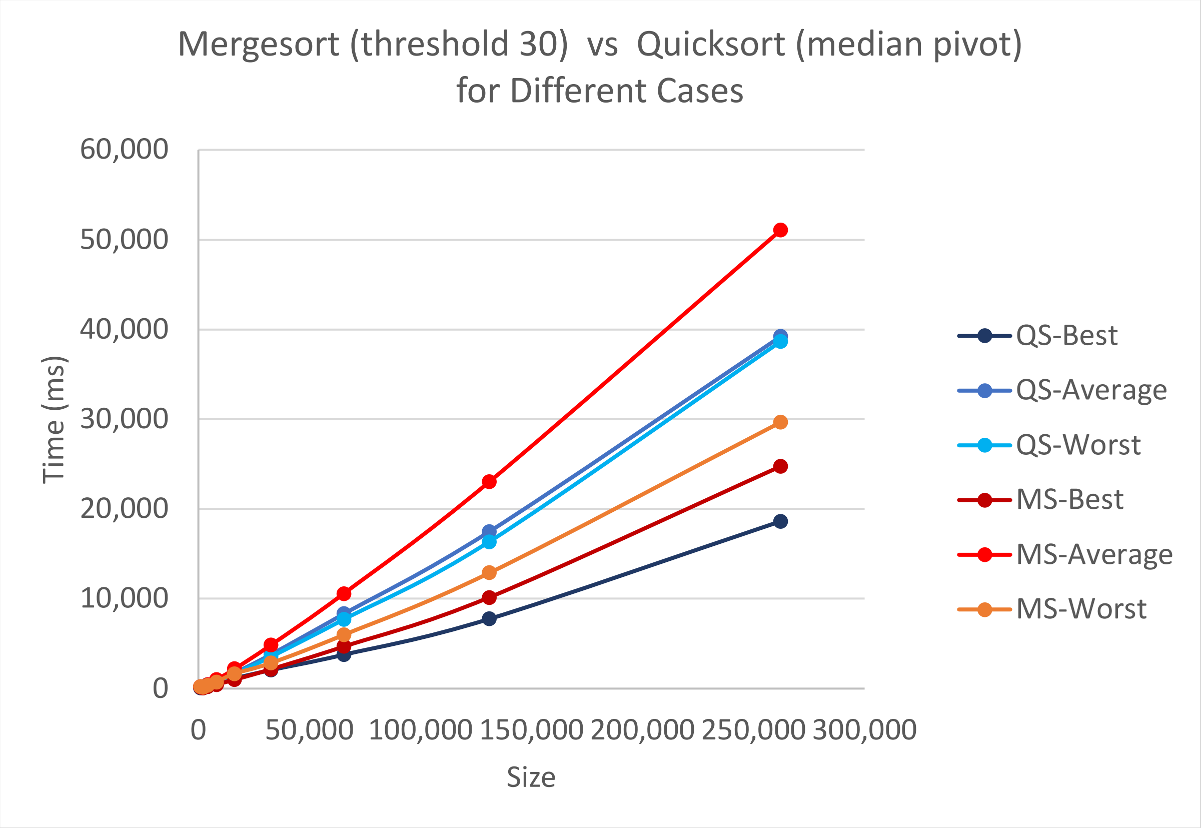
3. Quicksort Pivot Experiment: Determine the best pivot-choosing strategy for quicksort. (As in #2, use large list sizes, the same set of permuted-order lists for each strategy, and the timing techniques demonstrated before.) Plot the running times of your quicksort for three different pivot-choosing strategies on permuted-order lists (one line for each strategy).
Three pivot-choosing strategies:

1. Median 3: Get the value of the first, middle and last elements, then choose the middle value as the pivot.
2. Middle: Get the value of the middle element as the pivot.
3. Random: Get the value of the random element as the pivot.
4. Right: Get the value of the last element as the pivot.

The most time efficient: median 3 (for average case)



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).



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.

Yes.

For quicksort, we use the median 3 method to choose the pivot, and swap the pivot with the last element. The median 3 performs well than others because it is guaranteed that we won't choose the biggest or smallest number as the pivot. Then in the partition method, we traverse elements from the first one to the one before the pivot (the last element) and classify these elements into two groups. If the element is smaller than the pivot, it will be swapped to the left group(smaller). Otherwise, it will stay in the right group(bigger), and we use an iterator pointing to the first element that is bigger than the pivot (which is the first element of the right group). Because the partition method we use is different from the two-sided version in class, it may take more time to swap.

For merge sort, we test different threshold values from 0, 30, 50, 100, to 500. It shows that it is more efficient when the threshold is 30. The reason might be that the complexity of merge sort O(NlogN) and insertion sort O(N^2) cross when N is around 30. We found that the value will depend on how we write our merge sort and insertion sort because every group has a different answer, usually between 20-50.

Comparing quick sort using median 3 with merge sort using threshold 30, we found that quick sort is more efficient than merge sort both in the best and worst case. We assume this is because our merge sort spends more time creating new arrays. Also, the complexity of the quick sort in the worst case is not O(N^2) here because the median 3 will never choose the biggest or smallest element as a pivot.