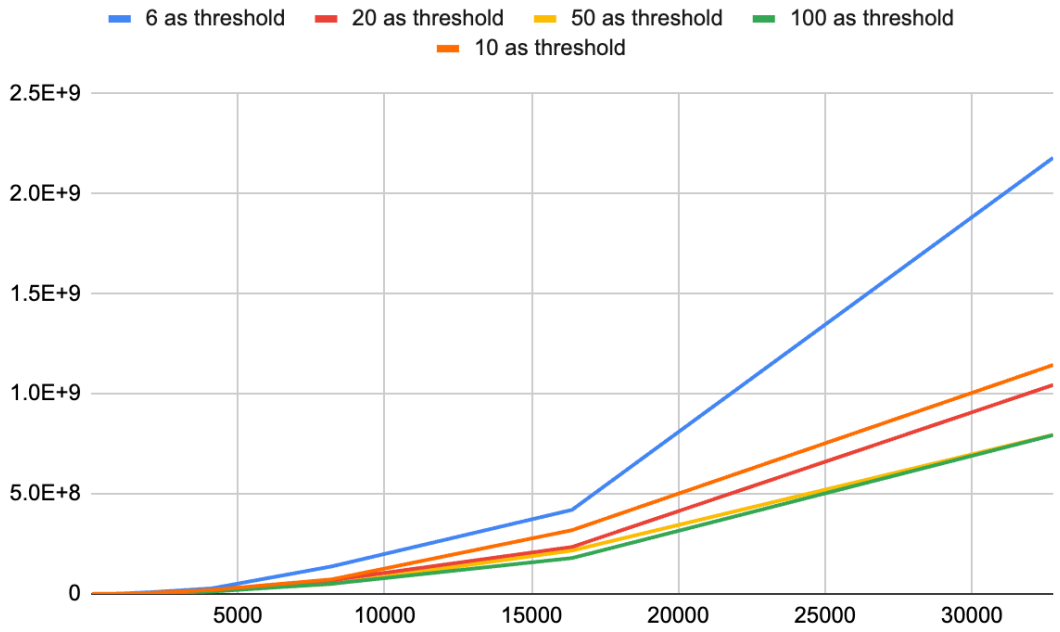
Analysis Document on Assignment 04

Tailang (Terry) CAO u1480633

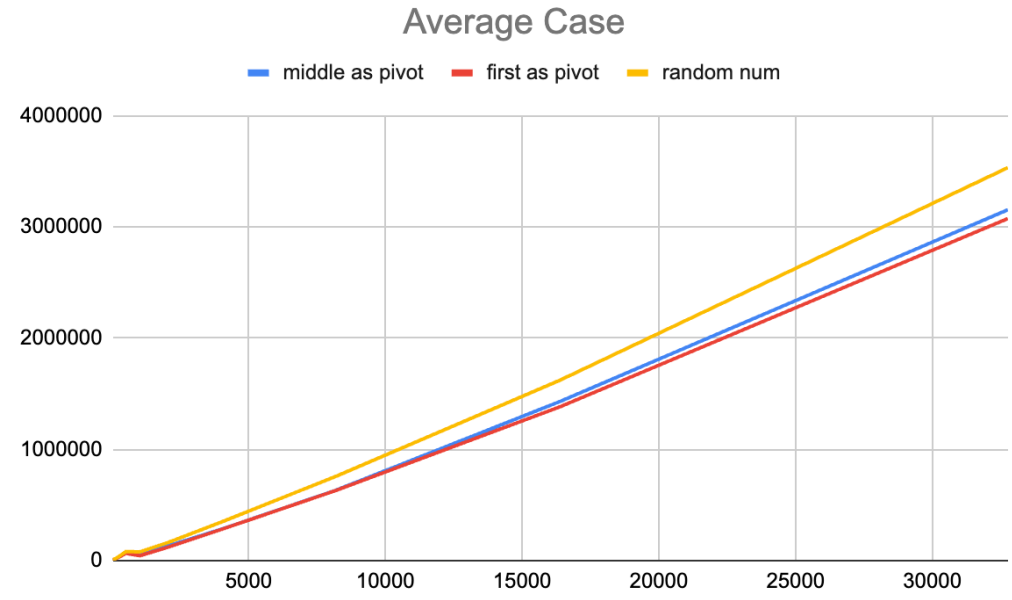
Team member: Yuyao (Spencer) TU

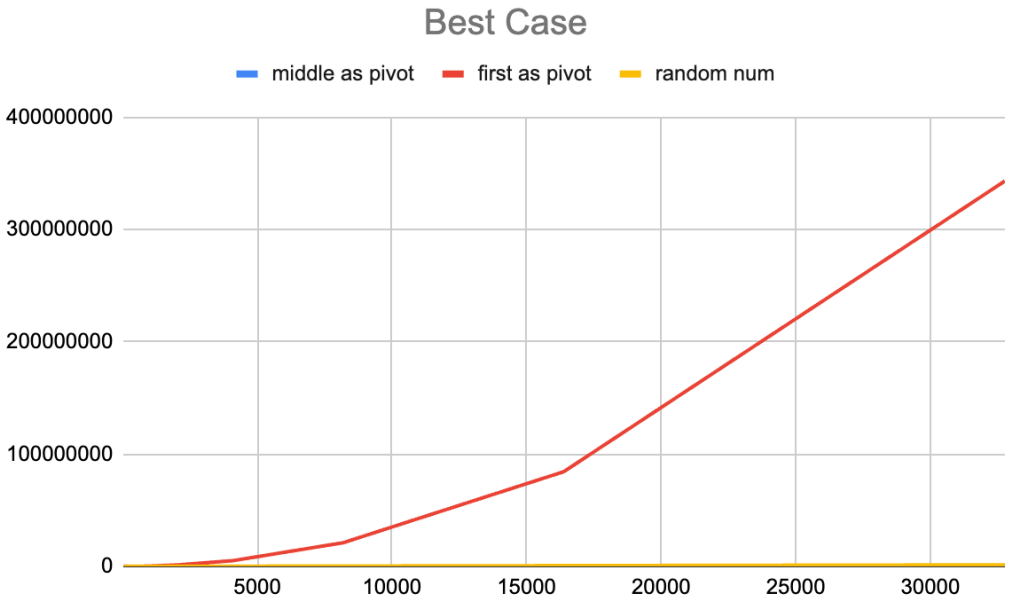
1. Mergesort Threshold Experiment



The experiment used 4 different threshold and the result is based on the average taken over 100 iterations and covers size from 2^5 to 2^15. As is shown in the graph, taking 6 as the threshold to switch from insertion sort to merge sort has the worst runtime performance, while taking 50 and 100 as threshold has the similar performance but taking 100 as the threshold is slightly better than 50.

2. Quicksort Pivot Experiment



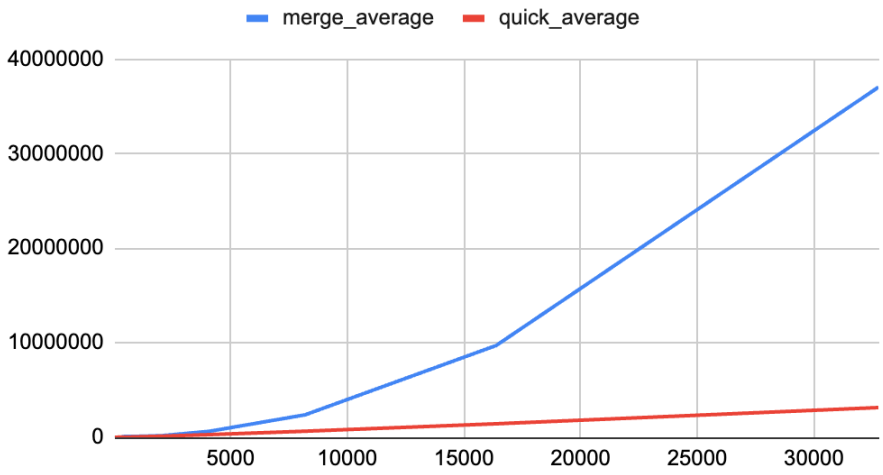


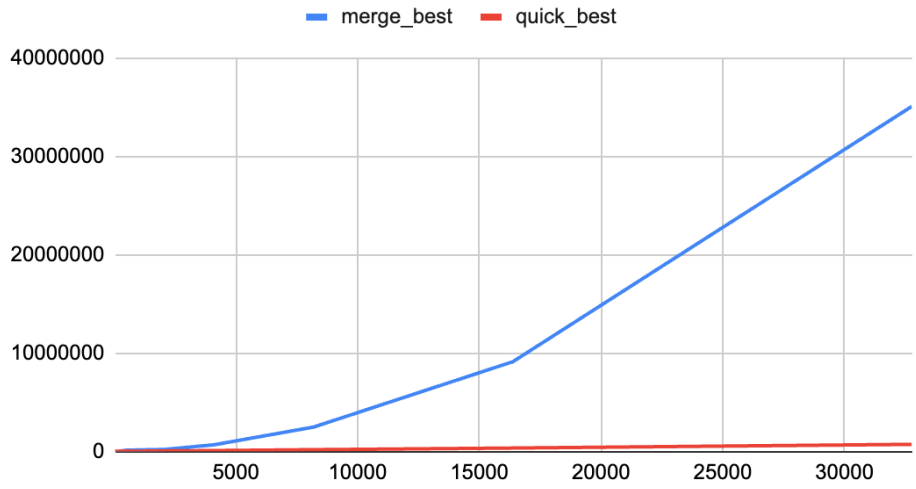
Above is the runtime plot of choosing the first element, the middle element, and a random element as the pivot in the quick sort. (the last element is not considered to be chose as a pivot as it is likely to have the same result of choosing the first element as the pivot). The data is the average result of 100 iterations with the size of the list being from 2^5 to 2^15.

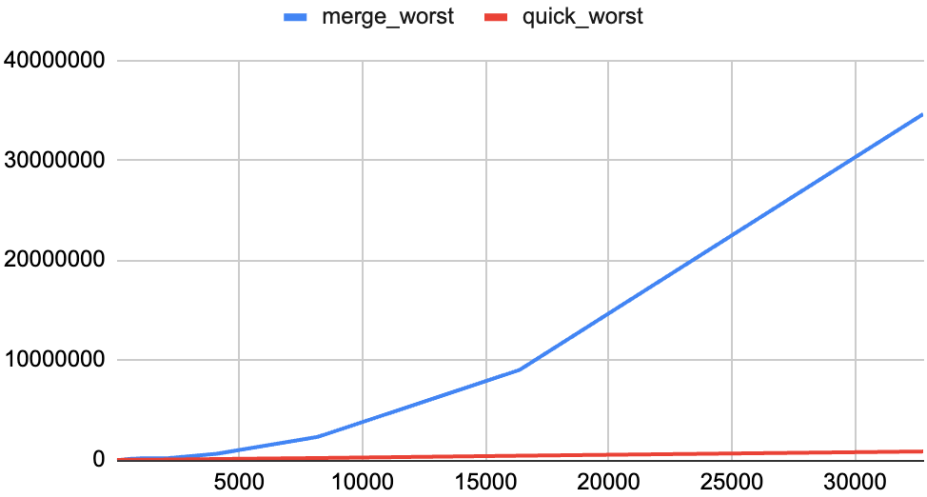
As is shown in the plot, the three methods has similar performance in the case of sorting the average case (a randomly generated arrayList), while taking the first as pivot seems to have a slightly better performance.

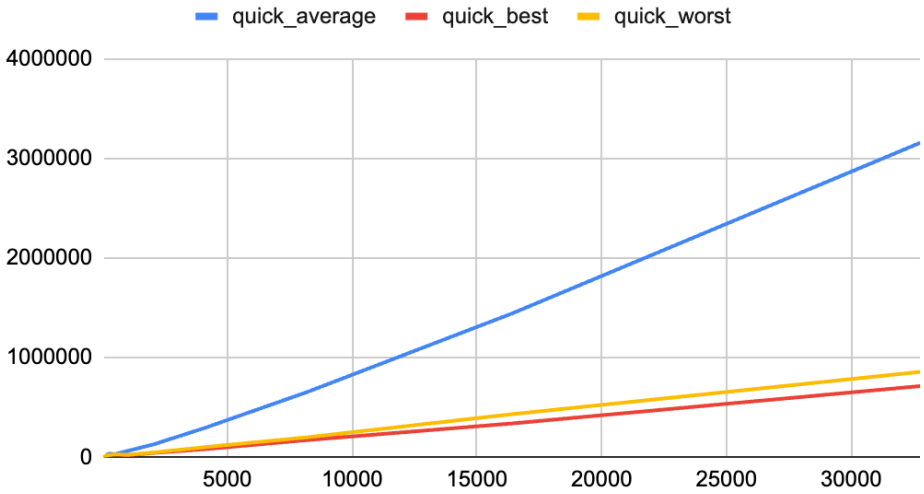
However, in the case of best case (an arrayList of integer 1 to required size in ascending order), taking the first element as the pivot has a significantly worse performance comparing with the other 2 pivot choice. This is because in the already sorted list, taking the first element as the pivot means always choosing the smallest number as the pivot, which result in a highly unbalanced partition, and each recursion can only reduce the size of the partition by one element. In the best case scenario, taking the middle element as the pivot has a slightly better performance comparing with taking a random element as the pivot, which is the same in the average case scenario. Therefore, taking the middle element as the pivot is considered to be the best pivot choice and will be implemented in following experiments.

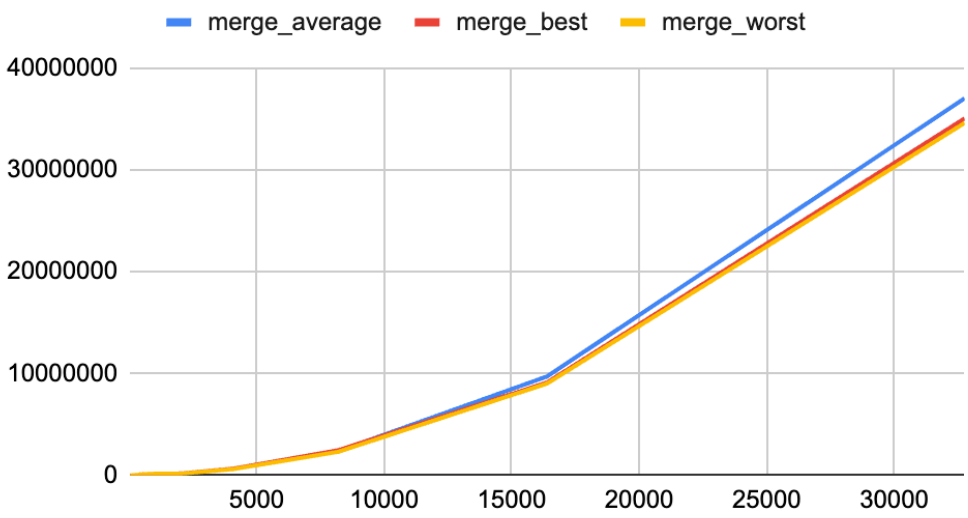
3. Mergesort vs. Quicksort Experiment




merge_best
quick_best
40000000
30000000
20000000
10000000
0
5000
10000
15000
20000
25000
30000


merge_worst
quick_worst
40000000
30000000
20000000
10000000
0
5000
10000
15000
20000
25000
30000


quick_average
quick_best
quick_worst
4000000
3000000
2000000
1000000
0
5000
10000
15000
20000
25000
30000



Above is the runtime plots of using merge sort and quick sort with the previously determined best threshold and pivot choice to sort the same array. The data is collected over taking the average of 100 iterations. As is shown in the plots, quick sort has a better performance in all three scenarios comparing with merge sort.

In all three scenarios, the runtime performance of quick sort is better than the performance of merge sort. This is resulted by the logic of merge sort having to create a new arraylist to store the data every time.

In the last two graphs, it can be seen that both merge sort and quick sort has a time complexity of nlogn, which meets the expectation. The significant runtime difference is still considered to be resulted by the newly created arraylists of merge sort.

For both methods, it is clear that the average case is the slowest condition. This is because branch predictor performs better on sorted data comparing with random data, which is the case of the average case.