2XB3 Lab 4

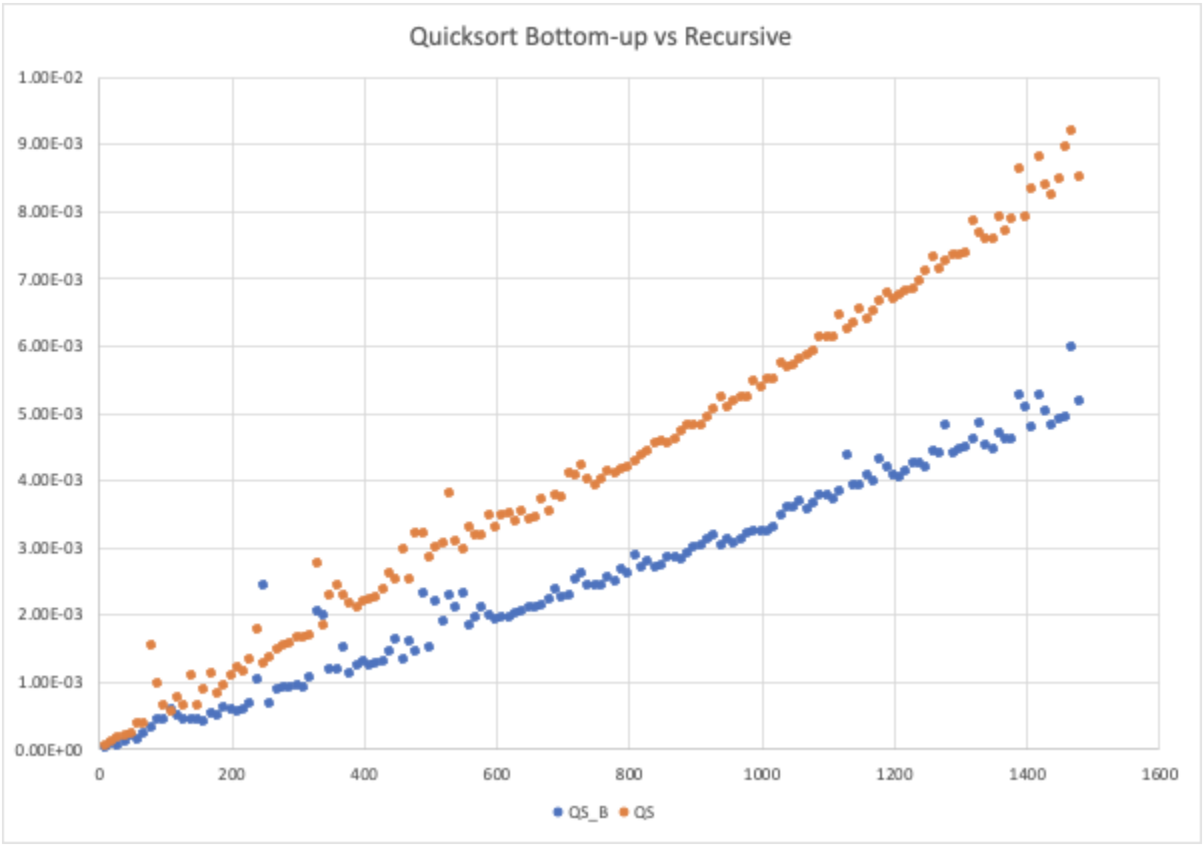
Casey Doede – 001223493 – doedecd@mcmaster.ca – L01

Amaan Ahmad Khan – 400230523 – khana251@mcmaster.ca – L01

Alex Axenti – 400268807 – axentia@mcmaster.ca - L04

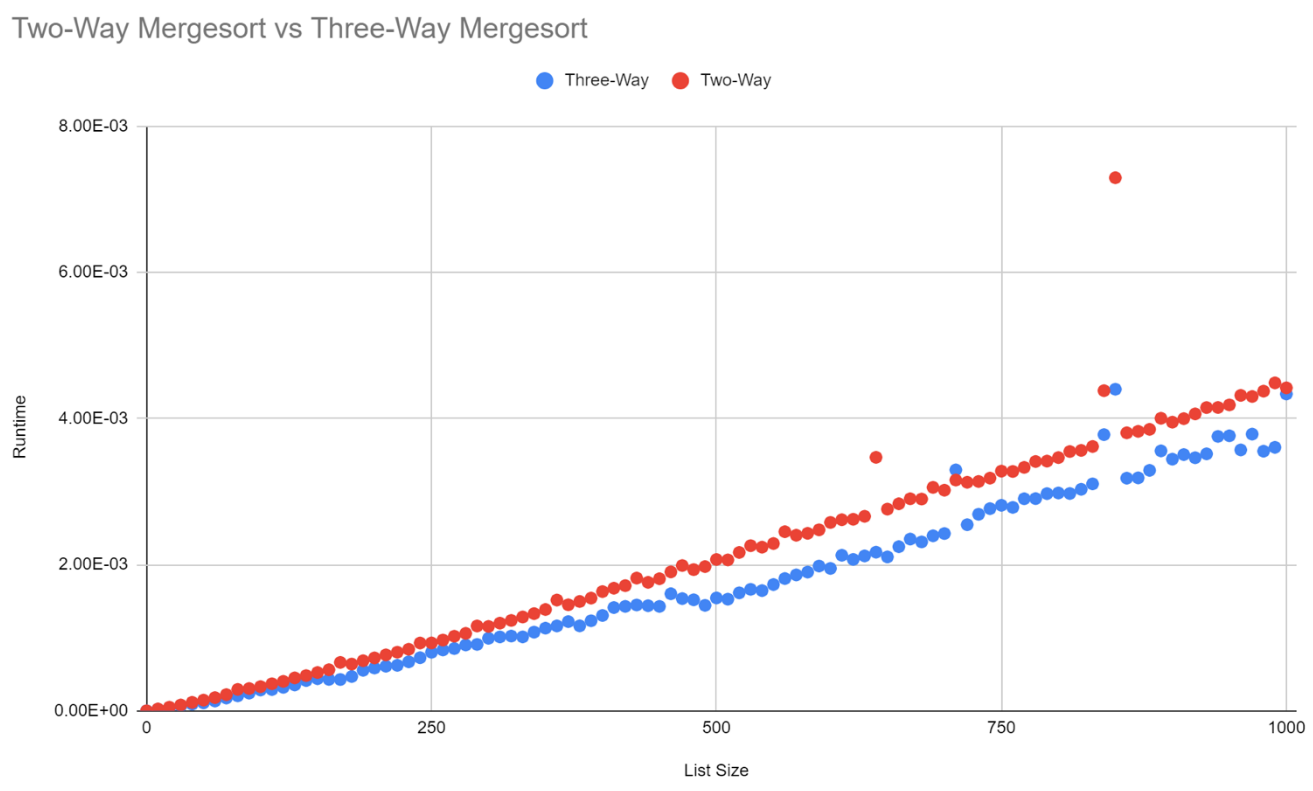
Bottom up Mergesort

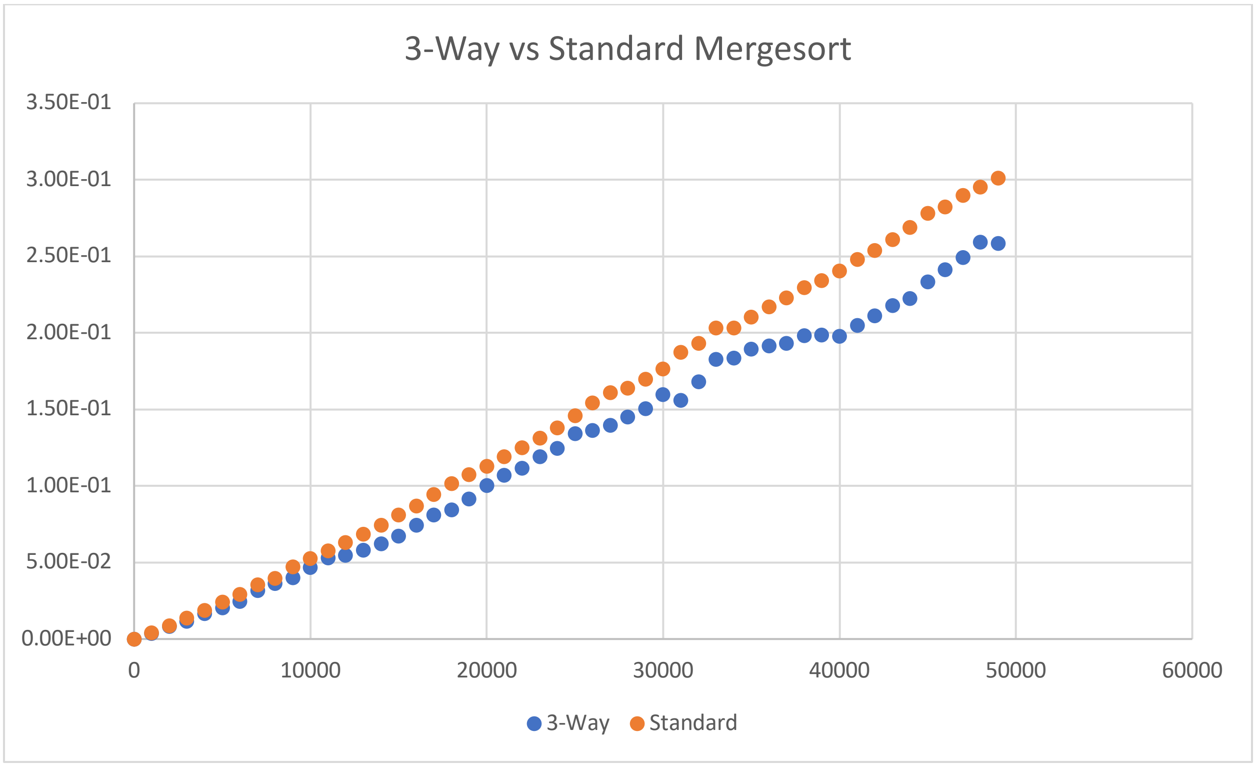
Bottom up mergesort has been implemented and compared to the traditional recursive top-down approach. It was found bottom-up mergesort does indeed perform better by the factor of a constant. Although both are still O(nlogn). The implementation of bottom up mergesort is of a greater complexity in the actual implementation aspects of design, as one cannot recursively call any functions for dividing the lists.



Three-Way Mergesort

We believed that three-way mergesort would be worse than two-way mergesort because it requires a greater number of comparisons when merging lists back together. For example, in two way, if one of the two lists being merged has reached its end, then the remainder of the other list gets automatically added. Meanwhile for three-way mergesort, you have to check if only one of the three lists has reached its end, or maybe some combination of two lists have reached their end but not the other. Overall, we believed this would lead to more comparisons and reduced performance.


Two-Way Mergesort vs Three-Way Mergesort
● Three-Way ● Two-Way
Runtime
8.00E-03
6.00E-03
4.00E-03
2.00E-03
0.00E+00
0
250
500
750
1000
List Size

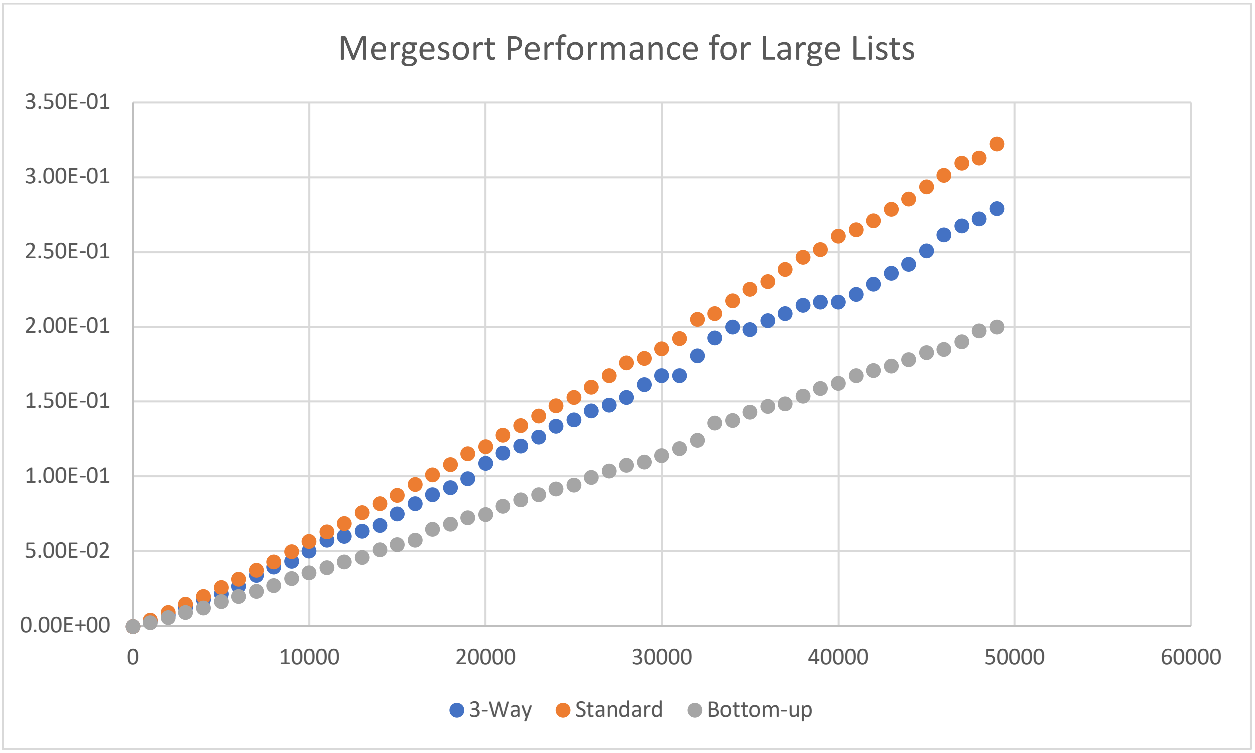

3-Way vs Standard Mergesort
3.50E-01
3.00E-01
2.50E-01
2.00E-01
1.50E-01
1.00E-01
5.00E-02
0.00E+00
0
10000
20000
30000
40000
50000
60000
● 3-Way ● Standard

After running the experiment, it was found that three-way mergesort was faster by a small factor. Clearly, our initial belief that three-way would be slower was incorrect. Although the difference in performance between a standard merge sort algorithm and three way merge sort algorithm is minor.

This can be explained by the fact that traditional mergesort utilizes $\log_2 n$ divisions whereas a three way mergesort would utilize $\log_3 n$ divisions. The tradeoff is that each merge is of a greater complexity as it must perform two comparisons instead of one.

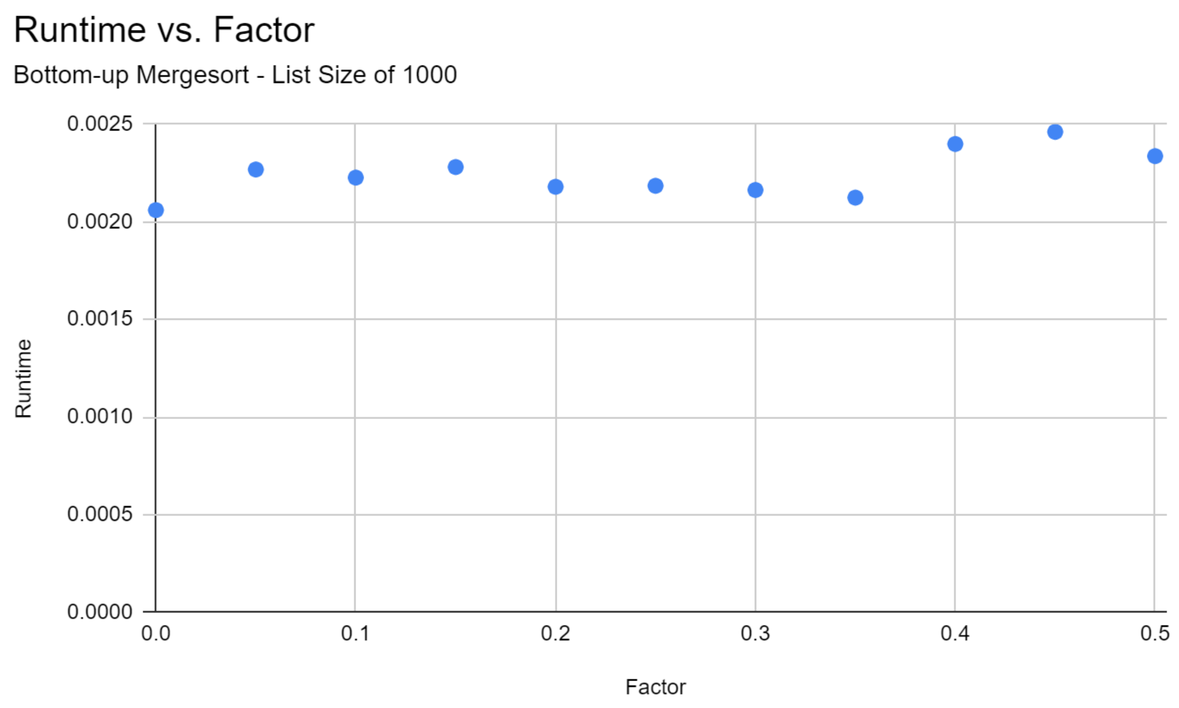
We performed the experiment a second time with large lists up to 50000 with similar results, 3-way seemed to perform better than traditional mergesort as the size of list grew larger. We deduce this could be due to the greater number of divisions allowing for one to get to the base case and begin merging and sorting faster. Though this brings the question of how many divisions would be optimal for mergesort. Overall this small increase in performance does not seem to be entirely worth the great increase in code complexity.

Finally all three versions of mergesort were compared and the bottom-up implementation was better than either version by a greater margin than that between traditional and three way mergesort.



Worst Case

After our previous analysis, it was determined that bottom up mergesort was the best implementation, and therefore it will be used for this section.



This was the result for executing bottom up mergesort on lists of size 1000, with a factor from 0 to 0.5. Comparing this to the previous graph of bottom up mergesort, where at a size of 1000 it took a time of approximately 0.00325, being near sorted clearly resulted in a faster runtime. This must be because when merging two sub lists, the left list will always be less than the elements in the right list, meaning that when the left list reaches its end, then the entire right list can be inputted without the extra comparison. Therefore, being near sorted is not a worse case scenario for merge sort, and is instead beneficial.

Another factor is that while quicksort splits the lists based on a pivot, mergesort divides lists into even portions of size n/2 recursively, or in the case of bottom up starts from size 1 and grows *2 each time. This means that there will not be the case of unbalanced list divisions due to an improperly chosen pivot. One can then conclude that mergesort retains its nlogn complexity for random, sorted and reverse-sorted lists.