CCDSALG Term 3, AY 2019 – 2020

Project 1 – Comparing Sorting Algorithms

Section	Names	Task 1	Task 2	Task 3	Task 4
<S15>	Lim, Christopher G.	X	X	X	X
<S11>	Lim, Ivan	X	X	X	X
<S12>	Sy, James	X	X	X	X

Fill this part with your section and names. For the tasks, put an X mark if you have performed the specified task. Please refer to the project specifications for the tasks.

LIST OF SORTING ALGORITHMS

Sorting Algorithm	Author (if available)	Downloaded From/Copy Pasted From
1. Bubble sort	GeeksforGeeks	https://www.geeksforgeeks.org/bubble-sort/
2. Insertion sort	GeeksforGeeks	https://www.geeksforgeeks.org/insertion-sort/
3. Selection sort	GeeksforGeeks	https://www.geeksforgeeks.org/selection-sort/
4. Merge sort	GeeksforGeeks	https://www.geeksforgeeks.org/merge-sort/
6. Quick sort	GeeksforGeeks	https://www.geeksforgeeks.org/quick-sort/
5. Radix sort	GeeksforGeeks	https://www.geeksforgeeks.org/radix-sort/

COMPARISON TABLE

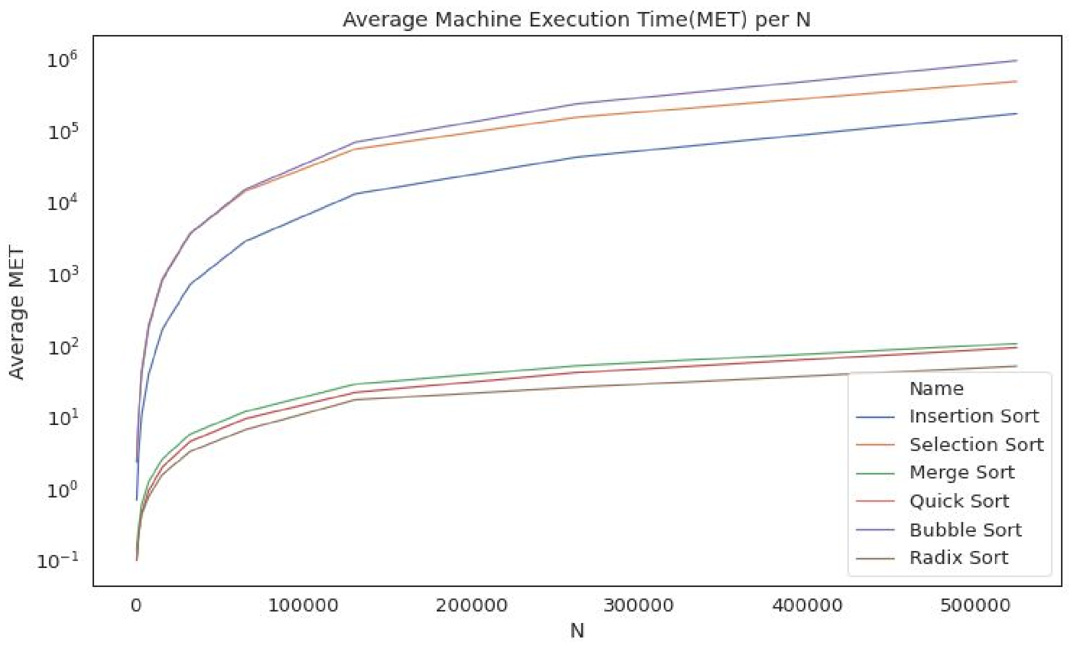
N = 10, M = 25

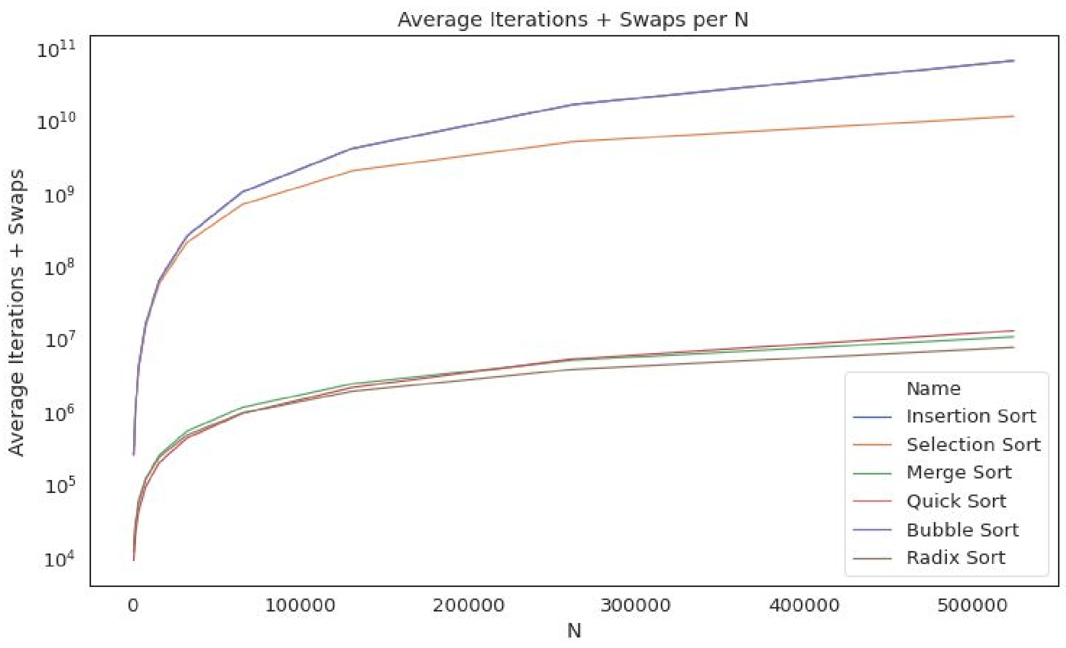
Size N	Average Machine Execution Time (milliseconds)					
	Bubble $O(n^2)$	Insertion $O(n^2)$	Selection $O(n^2)$	Merge $O(n \log n)$	Quick $O(n^2)$	Radix $O(nk)$
1024	2.363955	0.704316	2.368381	0.141708	0.100678	0.103824
2048	8.846344	2.587275	9.454002	0.284963	0.210359	0.20109
4096	40.380803	10.555163	44.163441	0.598471	0.455142	0.439045
8192	183.860378	40.384209	197.446497	1.27158	0.951894	0.787564
16384	830.435517	170.951239	877.871219	2.633427	2.033615	1.603994
32768	3637.597676	714.502013	3695.017597	5.826049	4.605925	3.337504
65536	15177.406779	2856.760789	14323.86917	12.025149	9.544604	6.715105
131072	69047.183346	13119.679616	55570.80337	29.072486	22.343117	17.569152
262144	236129.661678	42683.497168	153623.9945	52.462741	42.023032	26.320142
524288	945600.287781	172686.431336	489191.2488	107.159727	93.791742	51.952815

Size N	**Average Counter Value**					
	Bubble $O(n^2)$	**Insertion** $O(n^2)$	**Selection** $O(n^2)$	**Merge** $O(n\ logn)$	**Quick** $O(n^2)$	**Radix** $O(nk)$
1024	261953.04	261953.04	259913.48	12287.0	9252.48	15410.0
2048	1048761.36	1048761.36	1033742.16	26623.0	19906.4	30770.0
4096	4198883.36	4198883.36	4084588.28	57343.0	43705.72	61490.0
8192	16750353.96	16750353.96	15888569.32	122879.0	96352.84	122930.0
16384	67149982.76	67149982.76	60425985.12	262143.0	207278.36	245810.0
32768	268257468.8	268257468.8	218456428.48	557055.0	446802.12	491570.0
65536	1073258550.6	1073258550.6	725394370.44	1179647.0	973214.56	983090.0
131072	4293419017.76	4293419017.76	2116052820.6	2490367.0	2217265.68	1966130.0
262144	17174156576.64	17174156576.64	5310514086.72	5242879.0	5445535.28	3932210.0
524288	68743549408.52	68743549408.52	11859402810.92	11010047.0	13355166.68	7864370.0

GRAPHS

Copy/paste the graphs here, make sure it is big enough to see the trend in the increase of the average Machine Execution Time (MET) and the average counter value.





DISCUSSION

Explain interesting findings based on your experiments.

The six different sorting algorithms that were used in this analysis are Bubble sort, Insertion Sort, Selection Sort, Merge Sort, Quick Sort, and Radix Sort. The program created array sizes from 1024, 2048, ..., 524288 (N = 10) with random numbers from 1 to 50000 to be sorted 25 times each (M = 25). The experiment took around 11 hours to finish sorting all the arrays as the first three sorting algorithms took a long time due to their nature of having a worst case scenario of O(nˆ2). Bubble sort, Insertion Sort, and Selection sort were fairly the slowest among the six with average METs almost halved by the latter three. It can be seen in the graph Average MET against the array size N, Bubble sort and Selection sort had the worst times and closely beside them was Insertion Sort. Moreover, by looking at the average counter values of these three sorting algorithms, they also took up a lot more iterations and swaps in order to sort the array. The graph Average Iterations and Swaps against the array size N also showed similar results with the first graph.

On the other hand, the latter three sorting algorithms performed better and faster than what we expected. Merge Sort, Quick Sort, and Radix Sort were the faster sorting algorithms that came out of this experiment as their Average METs produced much less. In comparison, Bubble Sort had an average MET at N = 524288 of 945600.29 ms and Radix Sort only had 51.95 ms. In the same way when looking at the Average Counter, compared to the first three algorithms, they had a big difference. The graphs Average MET against the array size N and Average Iterations and Swaps against the array size N showed the three sorting algorithms in the bottom which pertained to a shorter time and less counts to finish.

Based on this experiment, the group has found that despite having a relatively small amount of array size, the faster algorithms were surprisingly almost instantaneous in arriving at a sorted array. It is interesting to note however that the average counter values for Bubble Sort and Insertion Sort were identical, but Insertion Sort had less MET to sort arrays. The sheer difference in results that the experiment produced showed that the use of these faster algorithms can have a big impact on the execution times on a machine. Interestingly, despite Merge Sort is O(n logn) and Quick Sort is O(n^2), Quick Sort still recorded a faster time and lesser counter value. However, looking at Radix Sort, though it can only sort numerical values, it shines at sorting with bigger array sizes and looking at its time with bigger arrays, it doubles the speed of Merge Sort and Quick Sort. Having their own unique uses, Quick Sort can still be seen as the reliable, fastest and most used sorting algorithm.

To add, this project/experiment has made the group realize how advanced the computers really are. Being able to sort 1024 items in merely a thousandth of a second is simply amazing in what computers are capable of doing. This project has broadened the viewpoint on what computers could really do and what efficiency could bring. For example, at N = 8192, Insertion Sort took almost 200x as much time as Merge Sort did. Putting it into perspective, it has cemented the idea of efficient code is a very core skill needed to be developed, especially moving forward in the future.