Maggie Asare

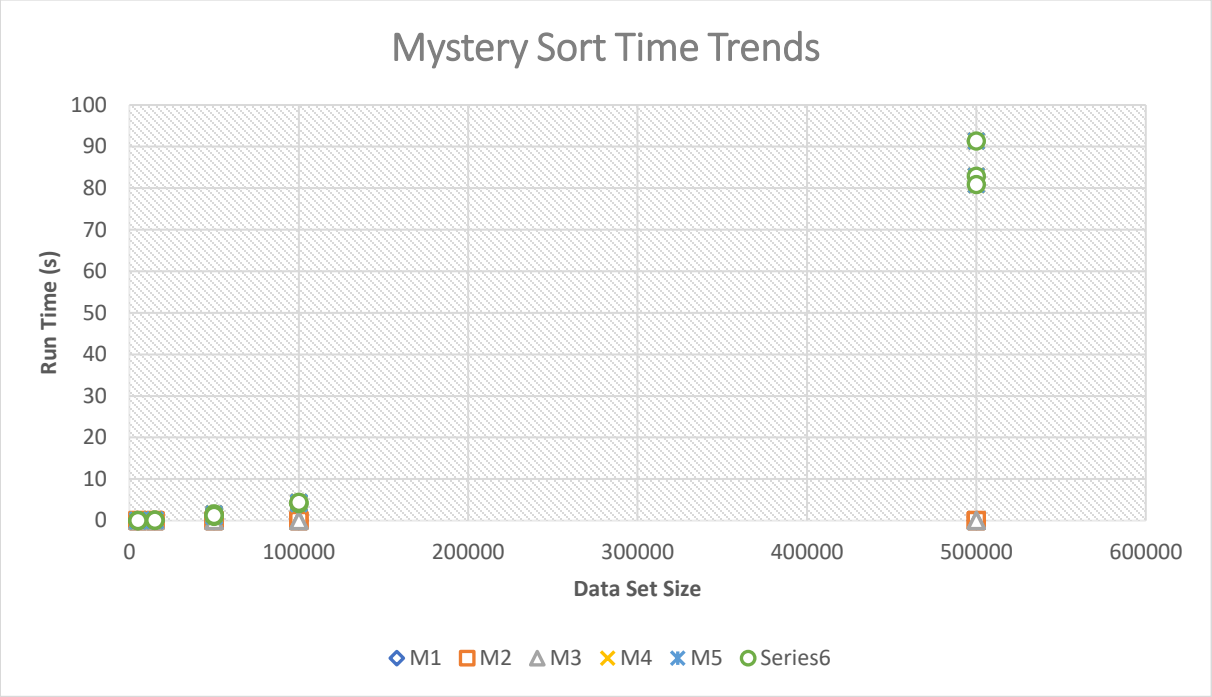
CSE 2341

PA03

After the digital troublemakers hacked the repository for Project 3 and made some distasteful edits to the function names, no one could differentiate between the different sort functions that were implemented. Consequently, I, Maggie Asare, an amateur C++ programmer was put on the job of analyzing the functions to determine which sorting algorithm was which. The options for this mystery were optimized bubble sort, insertion sort, merge sort, quicksort (with last element as pivot), and selection sort.

To detangle this sorting function mess, I needed to determine how quickly each function ran for data sets of varying sizes. I made a data tester class that created randomized integer data sets of the following sizes: 5000, 15000, 50000, 100000, 500000. I needed there to be a very small size and a very large size because some sorts do not seem so slow until they get to larger data set sizes, such as bubble sort. My data tester class then recorder the data into a csv format that I could export into Excel and reformat and graph.

Figure 1: Graph of Mystery Sort Run Times



I graphed the runtimes and for the most part, they were pretty close to each other. Most of the sorting algorithms could run in under a second. It was only upon closer analysis at the decimal level that I could differentiate between them.

According to analysis that I will go into more detail about momentarily, here are my identities for the mystery algorithms:

Mystery01: Optimized Bubble Sort
Mystery02: Quick Sort
Mystery03: Merge Sort
Mystery04: Selection Sort
Mystery05: Insertion Sort

Figure 2: Raw Data for Mystery Sorting

	A	B	C	D	E	F	G	H	I	J	K
1	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time	
2	Testing Mystery 1		Testing Mystery 2		Testing Mystery 3		Testing Mystery 4		Testing Mystery 5		
3	5000	0.000384	5000	0.000015	5000	0.000006	5000	0.012729	5000	0.00017	
4	15000	0.001254	15000	0.000038	15000	0.000015	15000	0.113639	15000	0.000449	
5	50000	0.00706	50000	0.000131	50000	0.000102	50000	1.047697	50000	0.001638	
6	100000	0.010717	100000	0.000241	100000	0.000126	100000	4.193941	100000	0.003168	
7	500000	0.052194	500000	0.001193	500000	0.000415	500000	91.36422	500000	0.016474	
8	5000	0.000418	5000	0.000008	5000	0.000016	5000	0.014275	5000	0.000143	
9	15000	0.001199	15000	0.000037	15000	0.000037	15000	0.125404	15000	0.000382	
10	50000	0.005553	50000	0.000075	50000	0.000129	50000	1.608323	50000	0.001467	
11	100000	0.009596	100000	0.000269	100000	0.000168	100000	4.083987	100000	0.002825	
12	500000	0.055989	500000	0.001221	500000	0.000673	500000	82.82541	500000	0.014183	
13	5000	0.000391	5000	0.000014	5000	0.000021	5000	0.012007	5000	0.000141	
14	15000	0.00148	15000	0.000032	15000	0.000017	15000	0.121109	15000	0.000391	
15	50000	0.004449	50000	0.000113	50000	0.00005	50000	1.213972	50000	0.001402	
16	100000	0.010738	100000	0.000262	100000	0.0001	100000	4.440545	100000	0.002872	
17	500000	0.055439	500000	0.001532	500000	0.000459	500000	80.82593	500000	0.015033	
18											

Let's start with the easiest to determine, the slowest and fastest algorithms, selection sort and quick sort respectively. I concluded that selection sort was Mystery04 because of how embarrassingly slow it was. Since selection sort has a complexity of O(n^2) in all cases, it is the slowest of all the algorithms. As you can see, I had to sit there for over a minute for it to complete the size 500000 data set, while all the other sorts completed that size in less than a second. Conversely, for quick sort, Mystery02, its speed is most apparent in the smaller data sizes. When it comes to smaller data sizes it is faster than Mystery03 a lot of the time, but once it gets to the 100000 and 500000 size data sets it is consistently slower than Mystery03.

This similarity between Mystery03 and Mystery02 checks out because I believe that Mystery03 is merge sort. Quick sort and merge sort are both O(nlogn) for their best and average case, but quick sort is O(n^2) in its worst case, while Merge sort is still O(nlogn). According to my research, in comparison to merge sort, quick sort is inefficient for larger arrays even though it works faster than every other sorting algorithm for smaller and medium data sets. Meanwhile,

merge sort is a consistent speed for any size data set (Bisht, 2019). The two sorts hold data differently as quick sort uses in-place sorting, but merge sort uses not in-place so it takes more storage. All of this is apparent in the fact that it gets to the point where merge sort is a whole tenth of a second faster than Mystery02 at the 500000 size data set.

Lastly, I needed to differentiate optimized bubble sort from insertion sort. Although both algorithms have complexities of O(n) for their best case and O(n^2) for their average and worst cases, insertion sort actually requires less swaps than optimized bubble (Sohail, 2020). As a result, Mystery01 and Mstery05 are generally not wildly off from each other in terms of speed; however, Mystery05, insertion sort, is consistently faster than Mysetry01, bubble sort.

Works Cited

Bisht, A. (2019, August 23). Quick Sort vs Merge Sort. Retrieved from https://www.geeksforgeeks.org/quick-sort-vs-merge-sort/#:~:text=Merge sort is not in place because it,case of larger array size or datasets. whereas

Sohail, A., MD. (2020, October 08). Comparison among Bubble Sort, Selection Sort and Insertion Sort. Retrieved from https://www.geeksforgeeks.org/comparison-among-bubble-sort-selection-sort-and-insertion-sort/