Write-up created by Kenny Zhang for CS 2341 (4649) at Southern Methodist University
Github: https://github.com/KZhang2002
Repository: https://github.com/SMUCSE2341/22s-pa02-KZhang2002

PA03 Write-Up

TL;DR

- mystery01 = merge sort
- mystery02 = optimized bubble sort
- mystery03 = insertion sort
- mystery04 = quicksort
- mystery05 = selection sort

Introduction

There are five functions entitled mystery01, mystery02, ... , mystery05. We know that the five algorithms are merge sort, optimized bubble sort, insertion sort, quicksort, and selection sort. But, we do not know which function corresponds to which algorithm.

Strategy

In order to find out which function is which algorithm, I will be timing how long it takes for the function to complete. I will be using the std::chrono library to do this.

The function will be tested by sorting various sizes and types of arrays. The sizes and structures of these arrays is elaborated upon in the key on the next page.

I will compare the time complexities of the algorithms for different types and sizes of arrays to the data I have collected to match the functions to their algorithms.

Methodology

For the ascending and descending arrays, seven trials are conducted for every array size. The mean of these trial sizes is then calculated. This is the value shown in the tables below and is the y value of the points plotted.

For the random array, the same process described previously is performed. However, this process is performed seven times using a different randomly shuffled array. The mean of the means of the trials is then used in the table and graphs below.

Multiple trials are conducted to ensure accuracy of results and prevent the randomness of computer processing times and std::random_shuffle from impacting the data.

Key for Data

Array Order Structure

- Ascending
  - (1, 2, 3, ... , n)
  - An already sorted list.
- Descending
  - (n, n - 1, n - 2, ... , 1)
  - A reverse sorted list.
- Random
  - Digits are random *and* unique. It is an ascending list that has been shuffled using std::random_shuffle. This means that every array has all the digits in the range [0, n).
  - All functions in the test use the *same* randomized dataset.

Legend

- 🔵 Mystery01
- 🔴 Mystery02
- 🟡 Mystery03
- 🟢 Mystery04
- 🟠 Mystery05

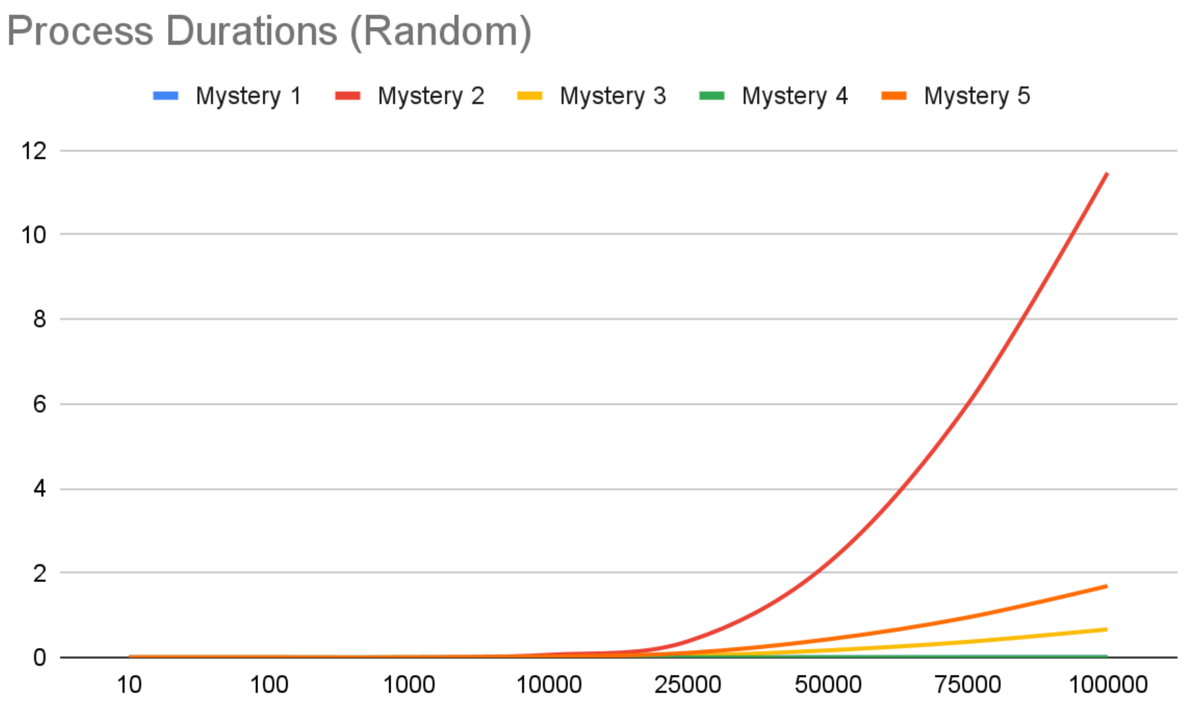
Note:

- The x-axis measures dataset size.
- The y-axis measures the time in seconds the function takes to sort the array.
- The x-axis has been standardized across all graphs.
- Line colors have been standardized across all graphs for your viewing convenience. The legend above is always right.
- Leading zeroes in the table have been grayed out for easier comparison.
- The data collected is discrete, however the data points have been connected for better visibility and analysis.
- The TA reading this is doing a great job. Your work is appreciated. Your presence is valued. *You* are valued.
- Taking breaks is good for you. Take one if you need one.

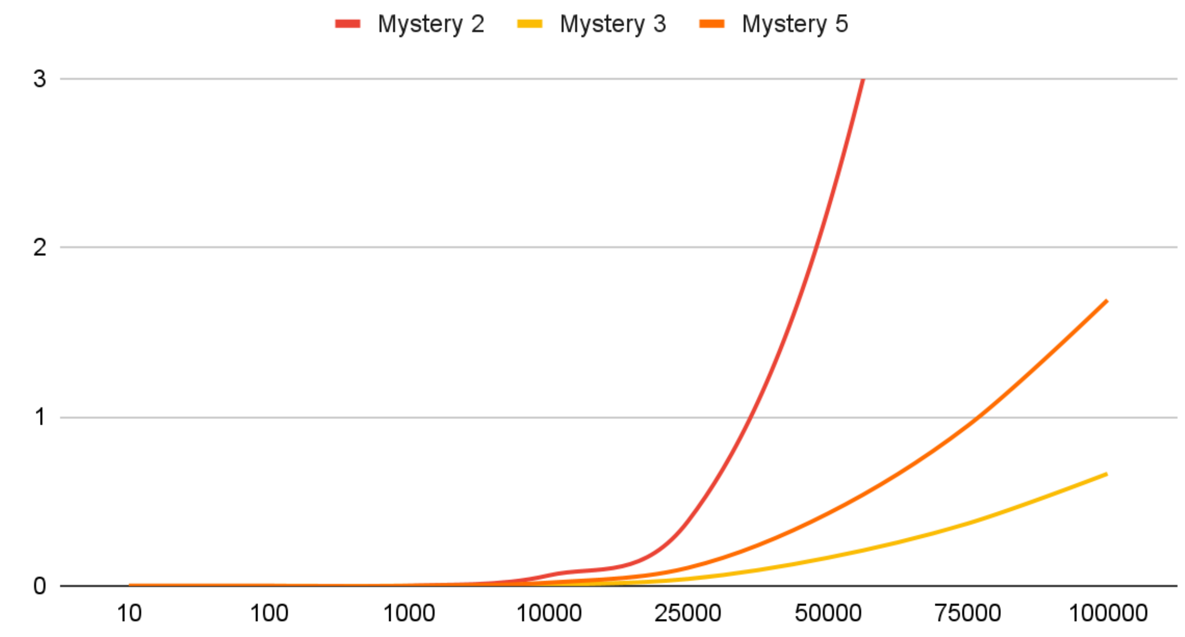
Dataset (Random)

Array Size	Mystery 1	Mystery 2	Mystery 3	Mystery 4	Mystery 5
10	0.000000449	0.000000172	0.000000117	0.000000157	0.000000196
100	0.000004640	0.000007027	0.000001489	0.000001191	0.000003670
1000	0.000033331	0.000649193	0.000078413	0.000018317	0.000221940
10000	0.000584790	0.061424831	0.006698225	0.000393831	0.018988753
25000	0.001604419	0.383511012	0.041037168	0.001134001	0.107113813
50000	0.003492662	2.225726510	0.166513718	0.002449537	0.427873133
75000	0.005315871	5.987623465	0.368675191	0.003801673	0.946600326
100000	0.007165204	11.467410542	0.662966269	0.005219691	1.689691841

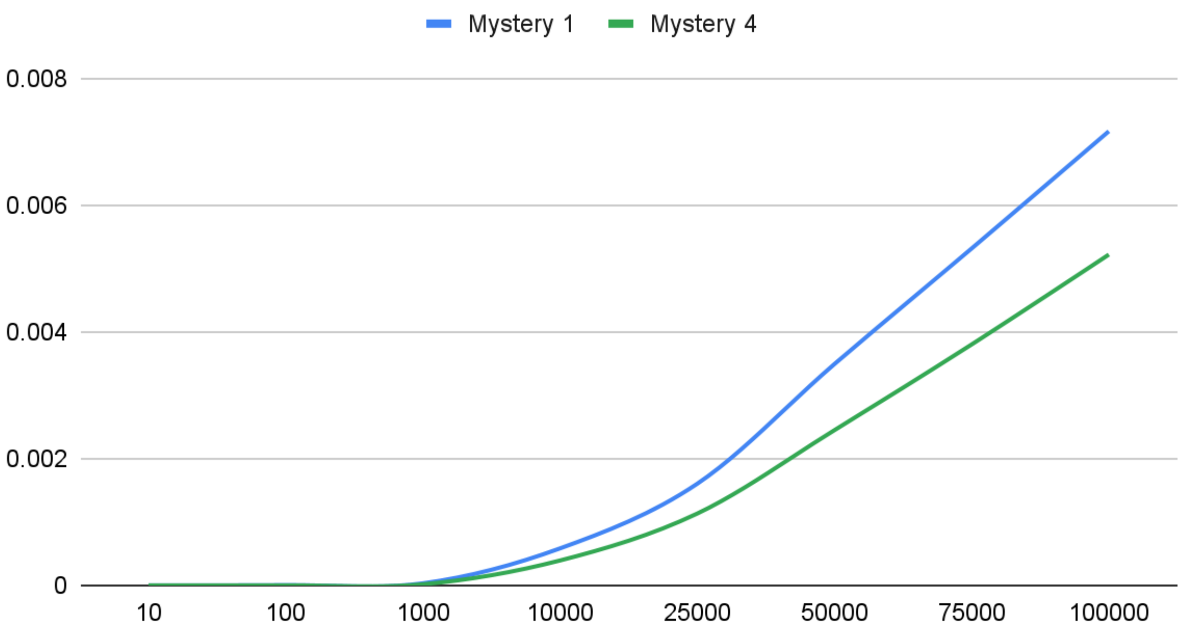
Graphs



O(n^2) Process Durations (Random)



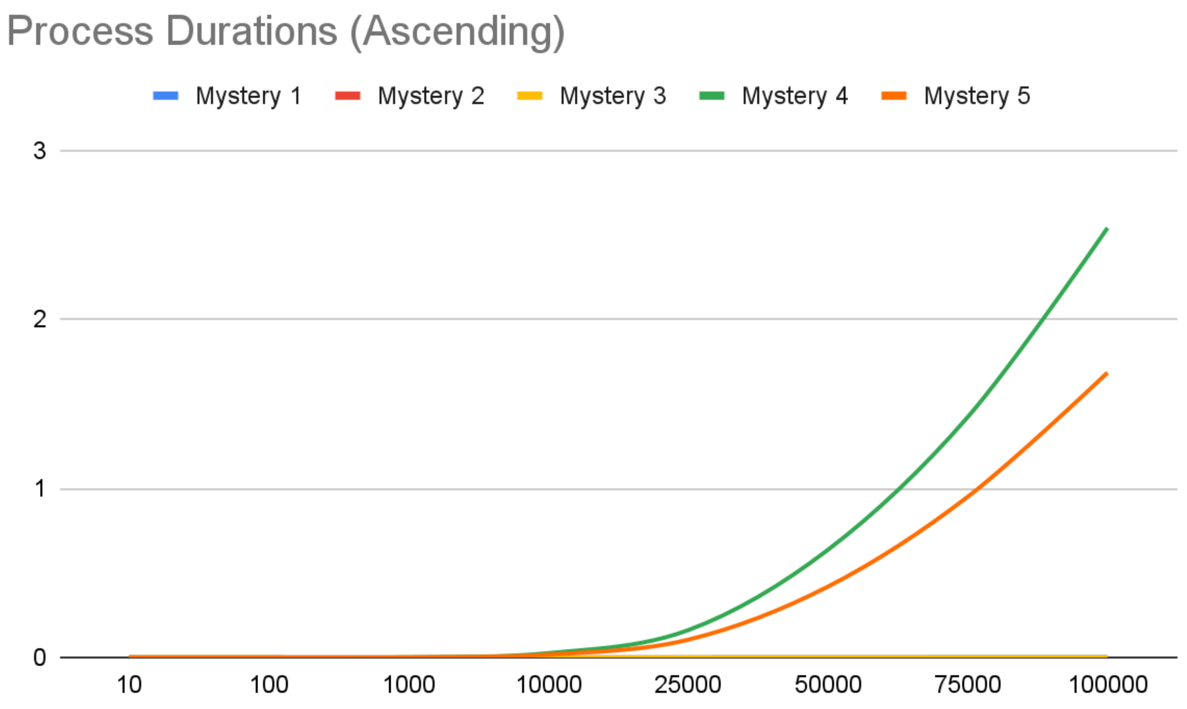
O(n log n) Process Durations (Random)



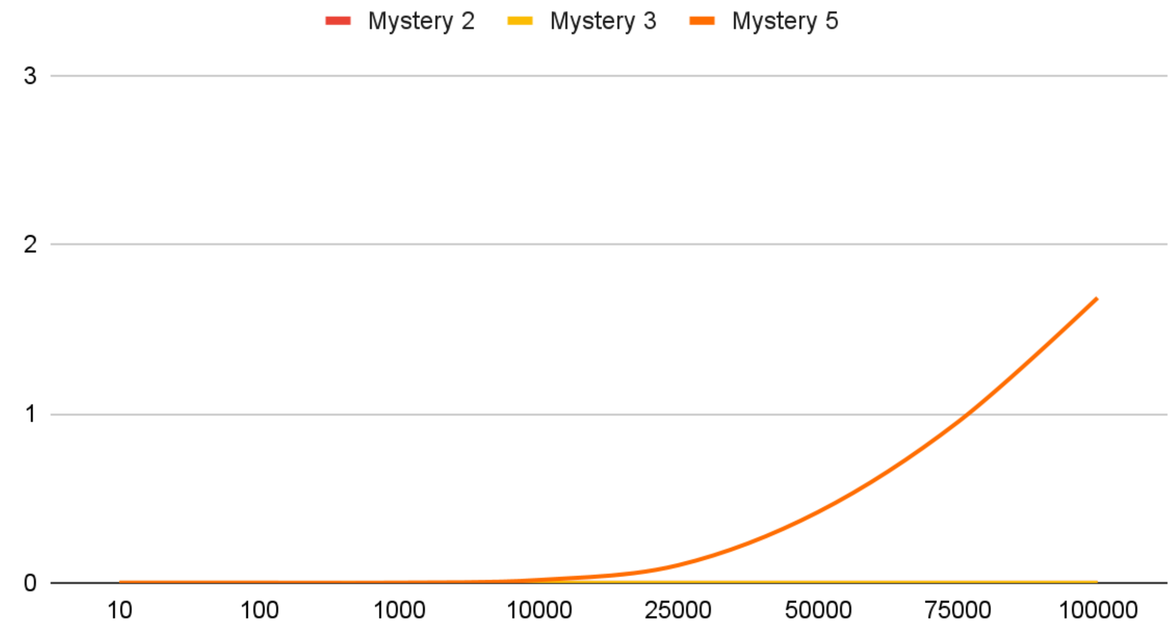
Dataset (Ascending)

Array Size	Mystery 1	Mystery 2	Mystery 3	Mystery 4	Mystery 5
10	0.000000261	0.000000039	0.000000052	0.000000103	0.000000086
100	0.000002331	0.000000090	0.000000118	0.000003130	0.000002525
1000	0.000017335	0.000000532	0.000000855	0.000258557	0.000175769
10000	0.000181829	0.000005004	0.000008197	0.025252198	0.016940570
25000	0.000458972	0.000012469	0.000019512	0.161363619	0.104744336
50000	0.000938315	0.000024892	0.000038966	0.636830343	0.418940699
75000	0.001448041	0.000037046	0.000058085	1.422121938	0.950464702
100000	0.001969797	0.000052831	0.000079678	2.541233654	1.685062329

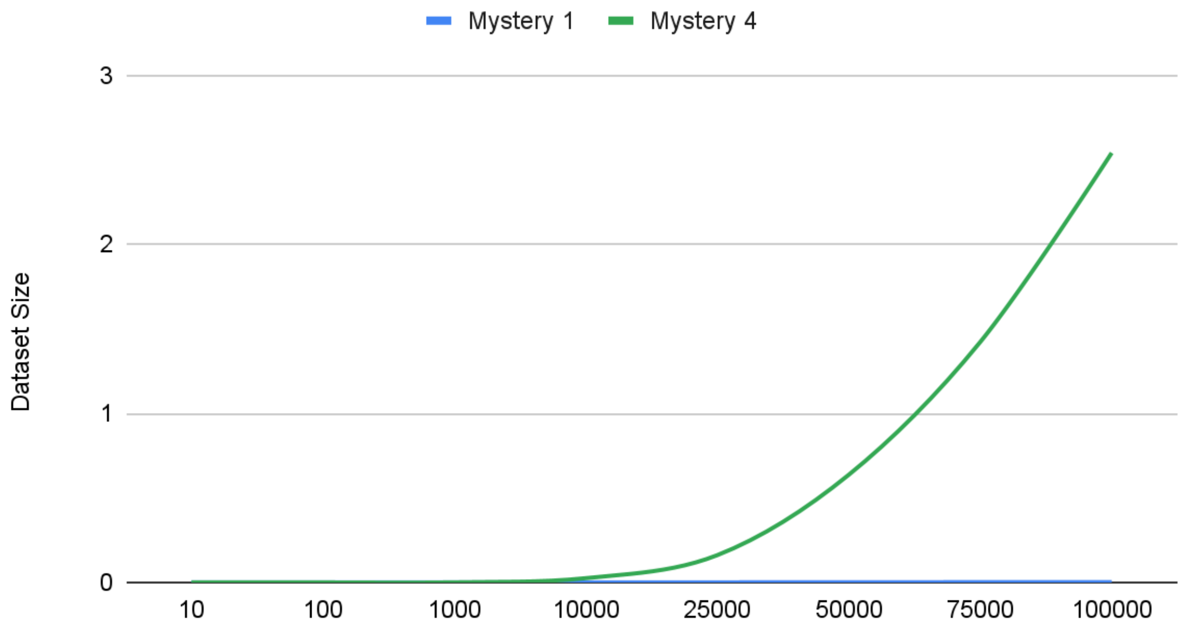
Graphs



O(n^2) Process Durations (Ascending)



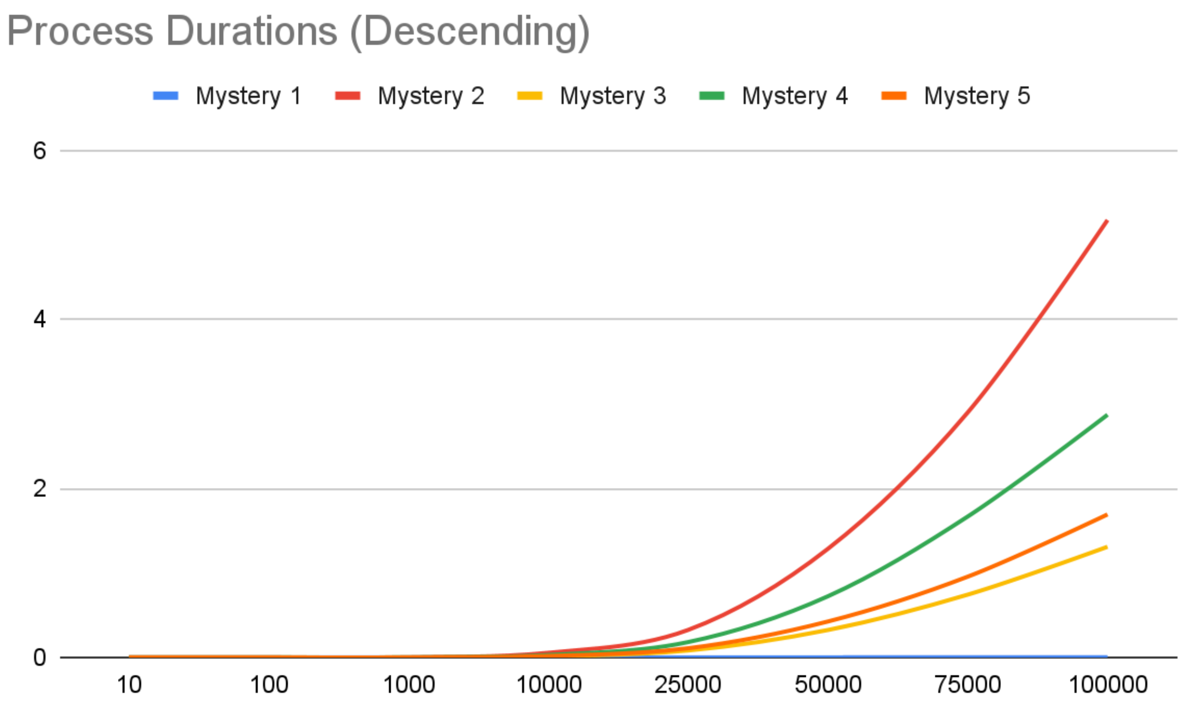
O(n log n) Process Durations (Ascending)



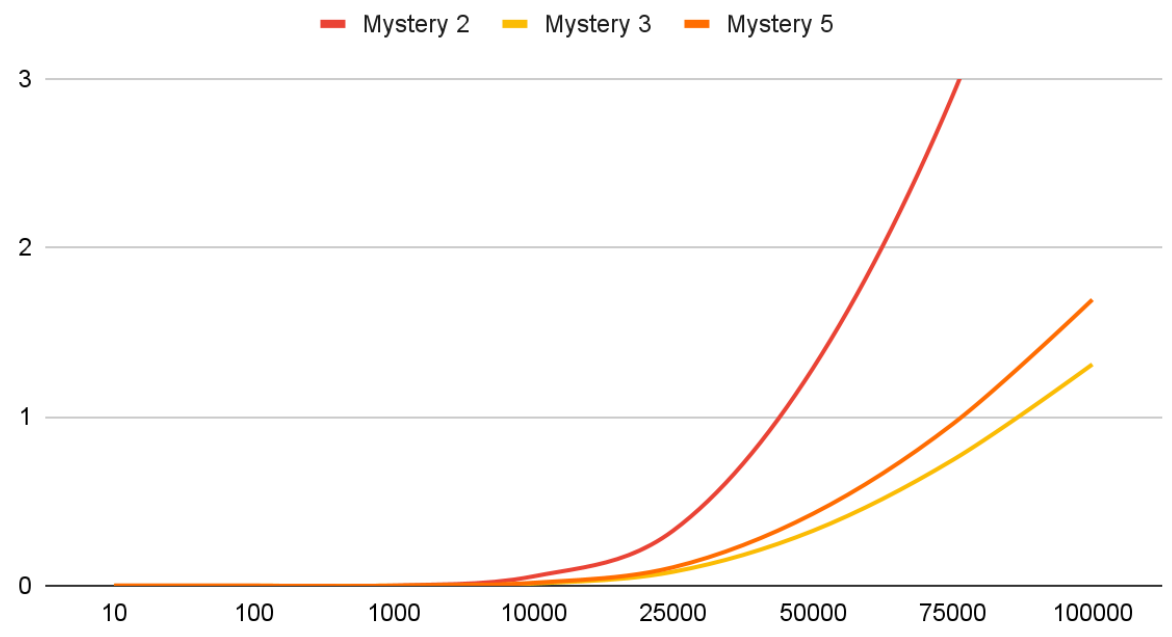
Dataset (Descending)

Array Size	Mystery 1	Mystery 2	Mystery 3	Mystery 4	Mystery 5
10	0.000000245	0.000000156	0.00000092	0.000000124	0.000000116
100	0.000002310	0.000006250	0.000001783	0.000003593	0.000002687
1000	0.000021698	0.000669907	0.000129109	0.000313496	0.000188825
10000	0.000203718	0.055021867	0.012717473	0.028948956	0.017196873
25000	0.000597105	0.325571999	0.082014942	0.181682191	0.108793553
50000	0.001016870	1.285482237	0.324615227	0.723805033	0.425339859
75000	0.001563160	2.898521898	0.742991980	1.667228896	0.955706954
100000	0.002149238	5.175492397	1.309260818	2.871568190	1.692012878

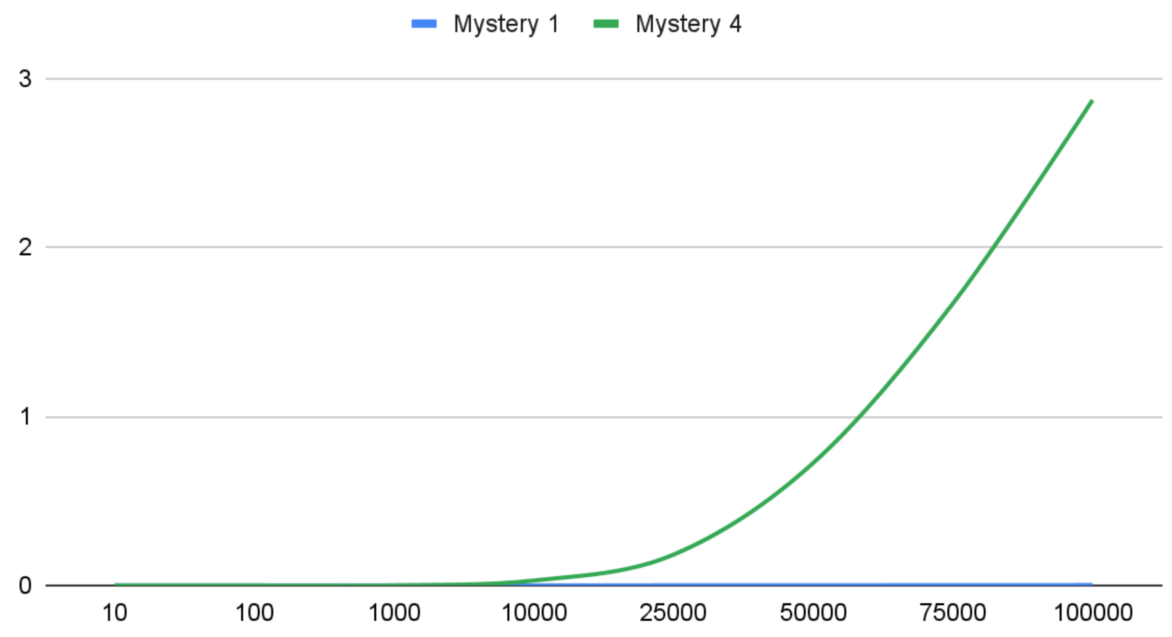
Graphs



O(n^2) Process Durations (Descending)



O(n log n) Process Durations (Descending)



Analysis

(Sources for information are in the footnotes)

Statement	Reasoning
Mystery two, three, and five are O(n^2) sorts.	• In the random chart, m2, m3, and m5 have a much greater duration ramp up as array size increases.
Mystery one and four are O(nlogn) sorts.	• In the random chart, m1 and m4 have a much lower duration ramp up as array size increases.
Mystery two is bubble sort.	• Bubble sort consistently has the average worst case time complexity for large arrays out of all the sorting algorithms except when sorting sorted lists.[1] ○ M2 is extremely slow when sorting the descending and random arrays. • Bubble sort has an O(n) complexity when sorting already sorted lists.[2] ○ M2 is extremely fast when sorting the ascending array.
Mystery three is insertion sort.	• Insertion sort has an O(n) complexity when sorting already sorted lists.[3] ○ M3 is extremely fast when sorting the ascending array. • Insertion sort is generally faster than bubble sort.[4] ○ M3 is faster than M2.
Mystery five is selection sort.	• Selection sort's time complexity is the same, no matter how the array is arranged.[5] ○ M5's durations are similar for all array tests. • Selection sort has an O(n^2) complexity. ○ M5 has an O(n^2) complexity as established earlier.

[1] LaMont, Michael. "Bubble Sort." Computer Science Department of University of Bath, http://web.archive.org/web/20071011001112/http://linux.wku.edu/~lamonml/algor/sort/bubble.html. Accessed 18 March 2022.
[2] Gupta, Shivam. "Time and Space Complexities of all Sorting Algorithms." *Interview Kickstart*, https://www.interviewkickstart.com/learn/time-complexities-of-all-sorting-algorithms. Accessed 17 March 2022.
[3] Ibid.
[4] LaMont, Michael. "Insertion Sort." Computer Science Department of University of Bath, http://web.archive.org/web/20071011001112/http://linux.wku.edu/~lamonml/algor/sort/bubble.html. Accessed 18 March 2022.
[5] Gupta, Shivam. "Time and Space Complexities of all Sorting Algorithms." *Interview Kickstart*, https://www.interviewkickstart.com/learn/time-complexities-of-all-sorting-algorithms. Accessed 17 March 2022.

Mystery four is quick sort.	<ul><li>Quicksort, when choosing the rightmost element as pivot, has an O(n^2) complexity for sorted and reverse-sorted arrays.[6]<ul><li>M4 performs poorly for ascending and descending arrays. Its curve is comparable to the O(n^2) functions.</li></ul></li><li>Quicksort has an O(nlogn) on average.<ul><li>M4 has an O(nlogn) complexity as established earlier.</li></ul></li></ul>
Mystery one is merge sort.	<ul><li>Merge sort's time complexity is the same, no matter how the array is arranged.[7]<ul><li>M1's durations are similar for all array tests.</li></ul></li><li>Merge sort has an O(nlogn) time complexity.<ul><li>M1 has an O(nlogn) complexity as established earlier.</li></ul></li></ul>

TL;DR (again)

- mystery01 = merge sort
- mystery02 = optimized bubble sort
- mystery03 = insertion sort
- mystery04 = quicksort
- mystery05 = selection sort

[6] Gupta, Shivam. "Time and Space Complexities of all Sorting Algorithms." *Interview Kickstart*, https://www.interviewkickstart.com/learn/time-complexities-of-all-sorting-algorithms. Accessed 17 March 2022.
[7] Ibid.