Write Up

- About Me

  My name is Leilani, and I am a freshman majoring in computer science.

- Problem Description

  I was tasked with identifying the type of sorting algorithm 5 functions labeled randomly, all while given no access to the source code.

- Strategy

  The strategy is to generate different types of data sets of varying sizes to test the efficiency of each function. Each algorithm would be tasked with sorting each data set and using the chrono library to record their runtimes in a table. Based off their runtimes with data sets that are: sorted, reverse sorted, and randomly sorted, I will determine which algorithm they are.

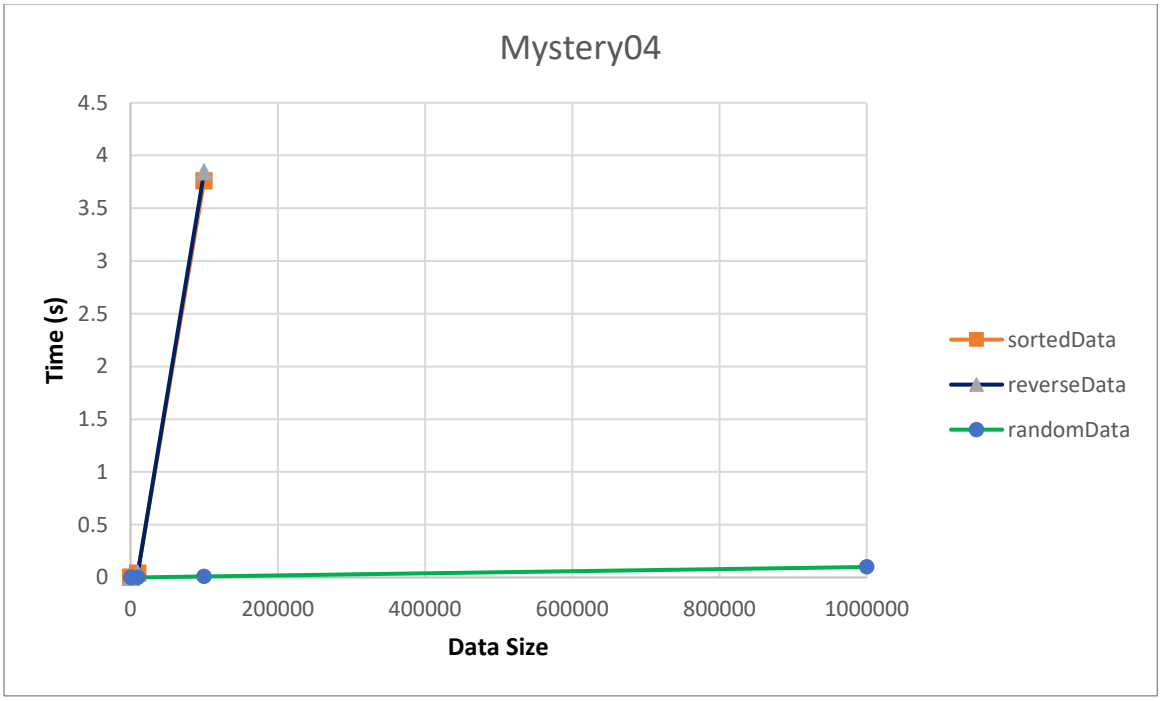
- Solution

  After analyzing their runtimes with different data sets, I have determined the following:
  - **Mystery01**: Merge Sort
  - **Mystery02**: Optimized Bubble Sort
  - **Mystery03**: Insertion Sort
  - **Mystery04**: Quick Sort
  - **Mystery05**: Selection Sort

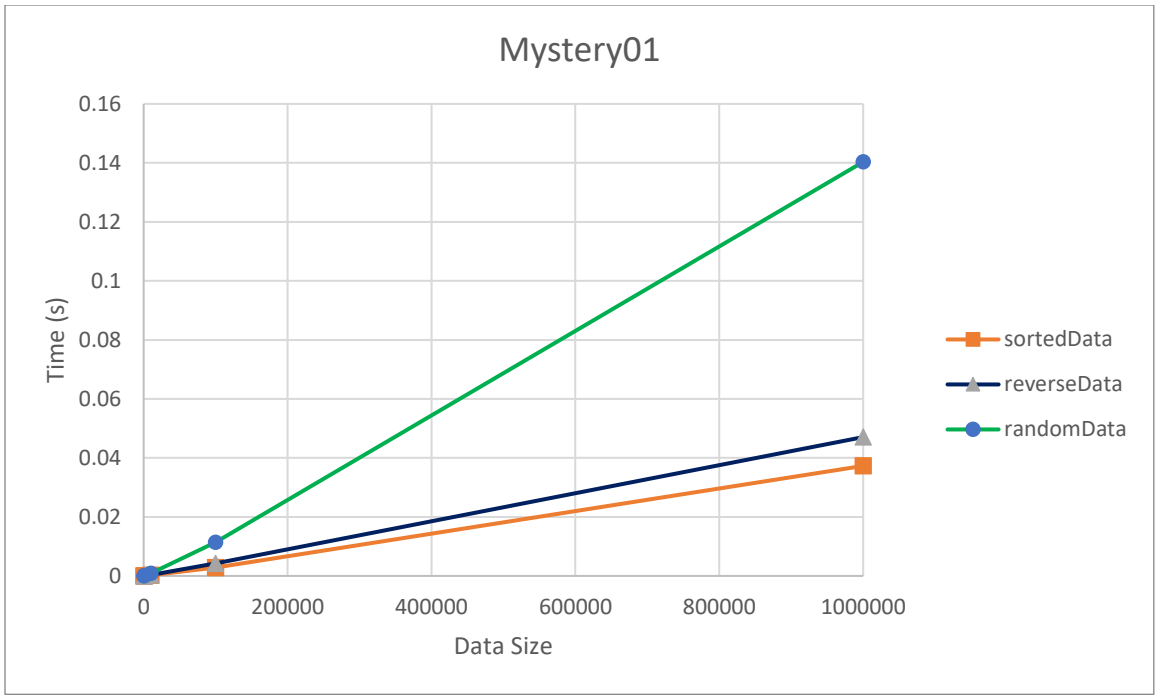
- Explanation (for me it was easier to look at the table)

	Mystery01		Mystery02		Mystery03		Mystery04		Mystery05	
	DataSize	Time(s)	DataSize	Time(s)	DataSize	Time(s)	DataSize	Time(s)	DataSize	Time(s)
sortedDataTimes	100	0.0000081	100	0.0000027	100	0.0000017	100	0.0000065	100	0.0000056
	1000	0.0000262	1000	0.0000034	1000	0.0000015	1000	0.0003908	1000	0.0002882
	10000	0.0002243	10000	0.0000209	10000	0.0000108	10000	0.0393753	10000	0.0287351
	100000	0.0027862	100000	0.0001447	100000	0.000104	100000	3.760184	100000	2.82035
	1000000	0.0372896	1000000	0.0012038	1000000	0.0010695	NaN		1000000	302.3057953
reverseDataTimes	100	0.0000074	100	0.0000136	100	0.0000036	100	0.0000059	100	0.0000054
	1000	0.0000307	1000	0.0011016	1000	0.000228	1000	0.0004241	1000	0.0002766
	10000	0.0003874	10000	0.1301004	10000	0.0193061	10000	0.0391507	10000	0.0269979
	100000	0.0042953	100000	8.3214737	100000	1.9995042	100000	3.8433239	100000	2.7811119
	1000000	0.0470527	1000000	991.543506	1000000	225.614879	NaN		1000000	305.2306938
randomDataTimes	100	0.0000106	100	0.0000159	100	0.0000032	100	0.0000056	100	0.0000063
	1000	0.0000709	1000	0.0009882	1000	0.0001204	1000	0.0000506	1000	0.0003101
	10000	0.0009081	10000	0.1628896	10000	0.0114149	10000	0.0006819	10000	0.0266227
	100000	0.0113704	100000	15.0989935	100000	1.0887219	100000	0.0079647	100000	2.6845645
	1000000	0.1403431	1000000	1793.7336	1000000	116.0787207	1000000	0.0989838	1000000	301.3780773
Guess	Merge Sort		Optimized BubbleSort		Insertion Sort		QuickSort		Selection Sort	

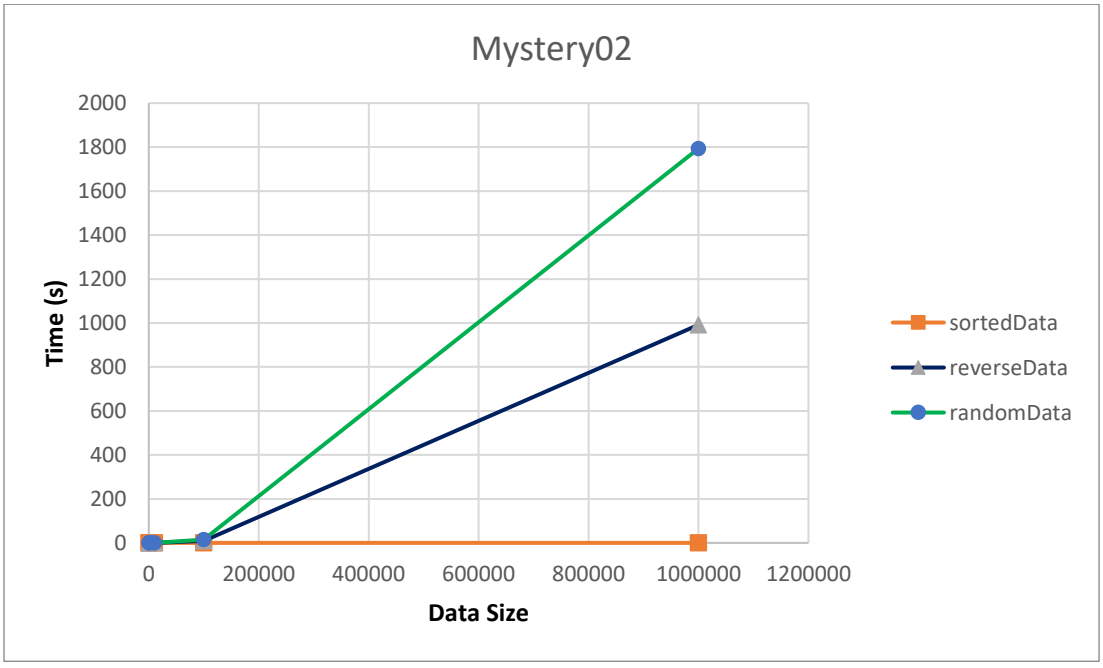
- The easiest to identify was **mystery04**, since it was not able to sort a very large data set of sorted or reverse sorted numbers (i.e., it gave an error code). For a randomized data set the runtime was incredibly fast compared to the other functions. From these observations I concluded that mystery04 was quicksort, specifically with the last element as the pivot. This makes sense because quicksort is known for performing as O(n^2) whenever it picks the biggest or smallest number as the pivot, which it would do if the data was sorted ascending or descending, and it performed great when the data was random.



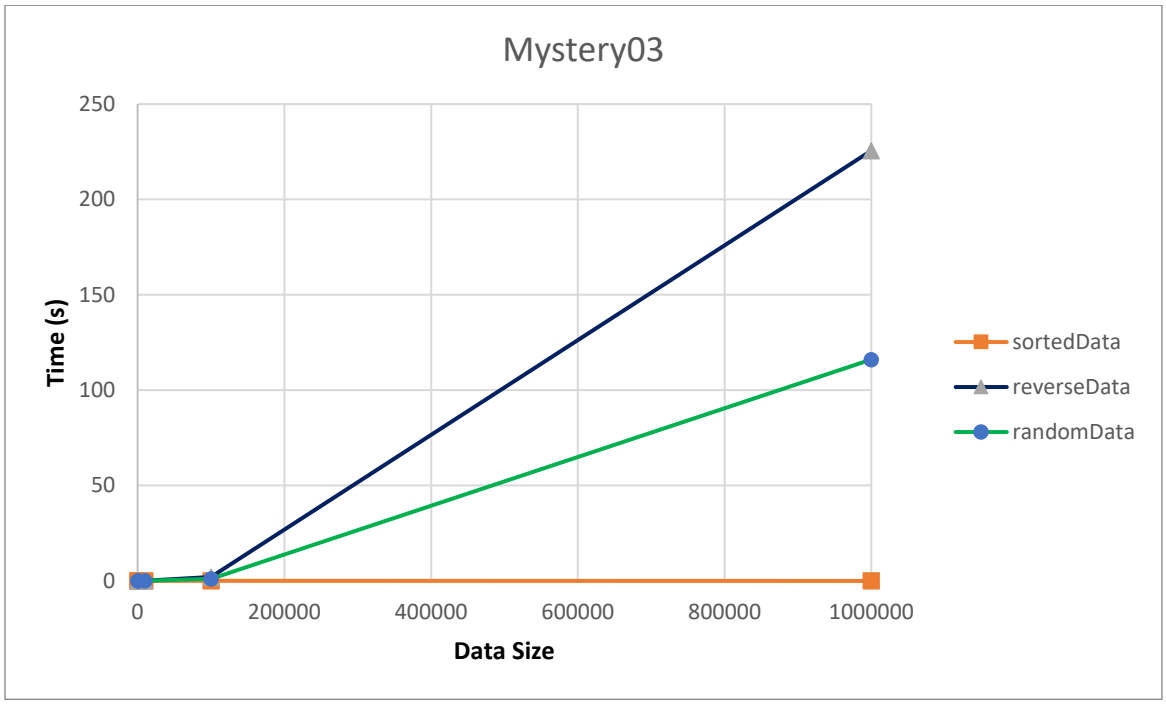
- Next I identified **mystery01**, since it is the only other sorting algorithm with O(n log n), and the recorded runtimes for each data size grew logarithmically and were generally faster than the other functions. Since quicksort was already taken, that lead me to believe that mystery01 was merge sort, the only other function that did *not* behave like O(n^2).



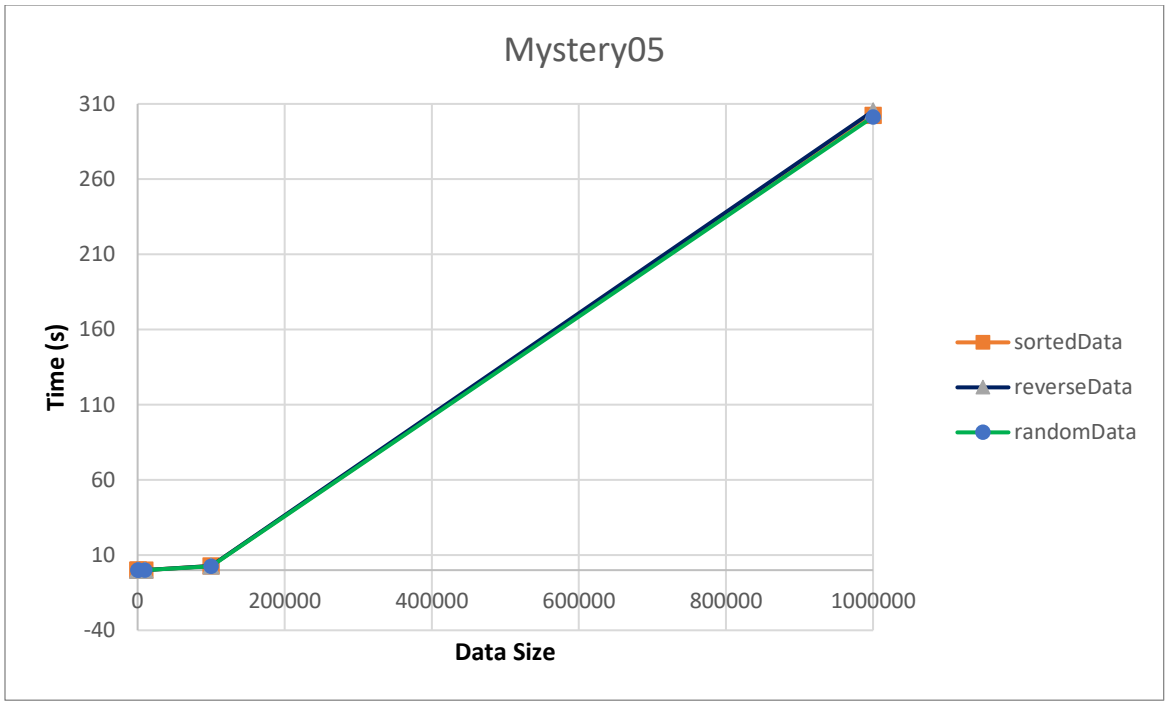
- Next, I identified **mystery02** since it had the worst runtimes for some of its datasets. When researching optimized bubble sort, I found that it was generally classified as O(n^2) but its best runtime (O(n)) was when given an already sorted list, and mystery02 performed much better than most of the other functions when given a sorted list. However, the runtime for the other datasets grew exponentially, and for the large data sets the runtimes were very large for mystery02 compared to all the functions, which lead me to believe that mystery02 was optimized bubble sort.



- Then I identified **mystery03**. The only sorting algorithms left are insertion and selection, which are both O(n^2). During research I found that insertion sort typically has its best case of O(n) when the data is already sorted (better than most other algorithms) and worst when the data is reversed. Mystery03 performed well like the other efficient algorithms when given a sorted data set, but it performed far worse for reverse sorted data, which is expected of insertion sort. This led me to believe that mystery03 was insertion sort.



- By process of elimination, and just generally slow and exponentially growing runtime (regardless of data order), I was led to the conclusion that **mystery05** was selection sort.


Mystery05
310
260
210
160
110
60
10
-40
Time (s)
0
200000
400000
600000
800000
1000000
Data Size
sortedData
reverseData
randomData