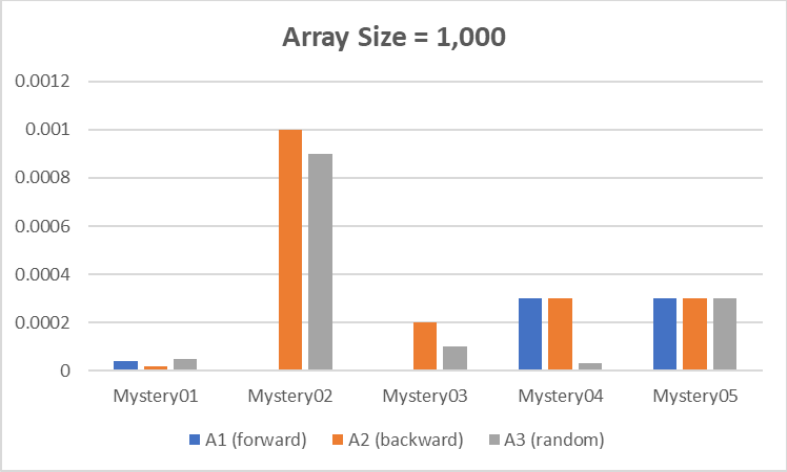
Deuce Palmer
Dr. Fontenot
CS 2341

PA 03 - Mystery Sorter

Findings

Mystery01 = Merge Sort
Mystery02 = Optimized Bubble Sort
Mystery03 = Insertion Sort
Mystery04 = Quicksort (with last element as pivot)
Mystery05 = Selection Sort



Write-up

My name is Deuce Palmer and I am a junior here at Southern Methodist University majoring in finance and minoring in computer science. My objective is to properly connect each sorting algorithm given (optimized bubble sort, insertion sort, merge sort, quicksort (with last element as pivot), and selection sort) to its corresponding mystery function.

My strategy was to test various datasets against the mystery functions in order to compare the results. While everything is based on time, that is the only variable being compared. The first step was to generate datasets that would be given to the mystery functions to be sorted. I created arrays that were forward, backward, and randomly sorted. I did this because I discovered that different sorting algorithms might perform differently based on the beginning state of the array. The three main array types were then duplicated for different sized arrays (1000, 10000, 100000, 500000). This is important in order to verify the results and make sure that the findings are sound no matter the size. After each mystery function was compared to each dataset created, the findings were analyzed to understand how each function did with different datasets, and how the functions performed against each other.

As the findings are above, it is important to understand the analysis behind them. The bar graph shows the findings from arrays containing 1,000 elements. This is because as the size of the arrays got larger, the different runtimes between O(n), O(n lg n), and O(n^2) grew and some data was no longer visible. Although not all graphs are shown, the complete findings can be found at the bottom of the report. While collecting the data, it was important to test multiple times in order to further solidify the findings for each runtime calculation. The first two mystery functions that were the easiest to identify were Mystery01 and Mystery05. As seen on the graph, Mystery01 and Mystery05 were the only two functions that had consistent runtimes no matter the initial state of the array. From the sorting algorithms given, the only two that behave like this are merge sort and selection sort. As merge sort is always O(n lg n) and selection sort is always O(n^2), it was easy to link Mystery01 to merge sort and Mystery05 to selection sort since Mystery01 was consistently performing faster than Mystery05.

The next one that was identified was Mystery04 as being quicksort. Mystery04 consistently had slower times for both forward and backward sorted arrays, but faster times for the randomly sorted arrays. As discussed in class, quicksort's worst case occurs when the pivot is either the largest or smallest element and its best case is when the pivot is the median. When an array is forward or backward sorted, the last element is either going to be the largest or smallest element. This explains why forward and backward sorted arrays have similar and slow time performance, since the quicksort given uses the last element as a pivot. On the other hand, a randomly sorted array is extremely likely to have a last element closer to the median than that of a forward or backward sorted array. Although it is possible to randomly select the largest or smallest element, it is highly unlikely especially with the size of the datasets being used. In addition, quicksort is not a stable sorting algorithm which may explain the errors I received when testing the algorithm against large datasets.

The last two algorithms to identify are bubble sort and insertion sort. These were the hardest since both of them have O(n^2) for average case and worst case and O(n) for the best case. For all of the forward sorted arrays, being the best case, Mystery02 and Mystery03 performed almost identically. In addition, both of which slowed down for the backward and randomly sorted arrays, but the dropoff was always more significant for Mystery02. This led me to understand that Mystery02 was bubble sort. Although the Big O's are the same, bubble sort will take longer because the algorithm performs more swaps than the insertion sort. As Mystery02 has been identified as bubble sort, the last connection was to link Mystery03 to insertion sort.

Data
(~ = rounded average based on multiple tests)

Size = 500,000	D1 (forward)	D2 (backward)	D3 (random)
01	~.02	~.02	~.06
02	~.0006	~280	~460
03	~.0007	~54	~28
04	Exit 11	Exit 11	~.04
05	~63	~66	~63

Size = 100,000	C1 (forward)	C2 (backward)	C3 (random)
01	~.002	~.002	~.009
02	~.0001	~11	~16
03	~.0001	~2.5	~1.2
04	Exit 11	Exit 11	~.006
05	~2.5	~2.5	~2.5

Size = 10,000	B1 (forward)	B2 (backward)	B3 (random)
01	~.0003	~.0003	~.0007
02	~.00001	~.12	~.08
03	~.00001	~.02	~.01
04	~.03	~.02	~.0004
05	~.025	~.025	~.025

Size = 1,000	A1 (forward)	A2 (backward)	A3 (random)
01	~.00004	~.00002	~.00005
02	~.000001	~.001	~.0009

03	~.000001	~.0002	~.0001
04	~.0003	~.0003	~.00003
05	~.0003	~.0003	~.0003