PA REPORT

ID: 11867592942

Name: Poyraz KOCA

Section: 2

Assignment Number: 2

Problem Statement and Code Design

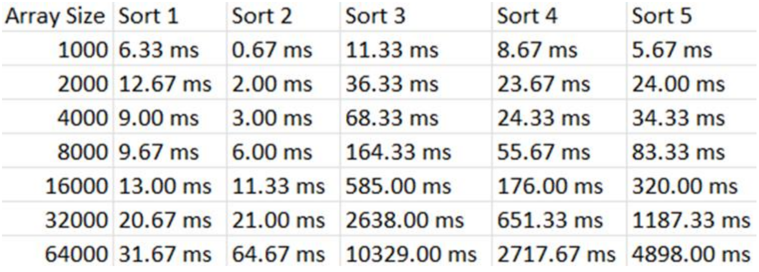
Problem Statement:

In this assignment, Task 2, I identified sorting algorithms implemented in the Java package “cmpe242-sort01.jar”. I differentiated between five sorting algorithms: MergeSort, QuickSort, BubbleSort, InsertionSort and SelectionSort. These sorts are implemented in the SortingAlgorithm Class, and by analysing the execution times I inferred which sorting algorithm corresponds to the method.

Code Design:

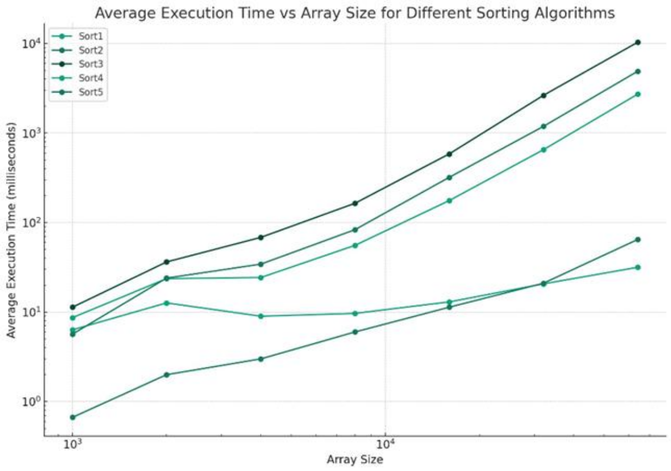
I utilized Eclipse IDE with the package provided by the instructor. Firstly, I generated 3 types of input arrays: Ascending Arrays, Descending Arrays, Random Arrays. Secondly, I implemented the SortingAlgorithmTester class to test the methods for sorting algorithms which are named as sort1, sort2, sort3, sort4 and sort5. To get their execution times, I used ‘System.currentTmeMillis()’.

Here is the data of my output which I obtained by changing the array size, shown by the table below:



Array Size	Sort 1	Sort 2	Sort 3	Sort 4	Sort 5
1000	6.33 ms	0.67 ms	11.33 ms	8.67 ms	5.67 ms
2000	12.67 ms	2.00 ms	36.33 ms	23.67 ms	24.00 ms
4000	9.00 ms	3.00 ms	68.33 ms	24.33 ms	34.33 ms
8000	9.67 ms	6.00 ms	164.33 ms	55.67 ms	83.33 ms
16000	13.00 ms	11.33 ms	585.00 ms	176.00 ms	320.00 ms
32000	20.67 ms	21.00 ms	2638.00 ms	651.33 ms	1187.33 ms
64000	31.67 ms	64.67 ms	10329.00 ms	2717.67 ms	4898.00 ms

To determine the types of sorts, I also generated a graph:



Implementation and Functionality:

Sort1: I observed that this method's time of execution is increasing logarithmically with the size of the array, which suits the time complexity of MergeSort, which is O(n logn).

Sort2: When I look at the behavior of this method, I see that the execution time is increasing logarithmically with the size of the array and it has the average time complexity of QuickSort O(n logn).
It's similar to Mergesort, but I chose QuickSort because QuickSort is generally faster than MergeSort.
As we can understand from the graph, sort2 is faster so it's more suitable for QuickSort.

Sort3: This method is suitable for BubbleSort because execution time is increasing quadratically with the size of the array, which corresponds to O(n^2).

Sort4: This method is kind of like Sort3, however it's more suitable for InsertionSort.
According to my calculations, I got the result O(n^2) again.
I chose Insertion Sort instead of Bubble Sort because InsertionSort is faster than BubbleSort in general.

Sort5: I chose SelectionSort for this method.
Again, I think that it has the same time complexity: O(n^2), however, it's not InsertionSort or BubbleSort.
The best option for this method is SelectionSort because SelectionSort is known to be slower than BubbleSort and InsertionSort.

Testing:

The code is working correctly without any error or problem. I was able to finish the assignment in a successful way by identifying each method and matching them with the suitable sort.
Thanks to this assignment, I was able to develop my sorting and algorithm knowledge and practice.

Final Assessments:

Trouble Points:

- One of the most difficult aspects was ensuring accurate linkage between vehicles, packages, and their respective cities, especially when processing large input files.
- Handling file input/output operations and addressing formatting inconsistencies in the provided input files required significant time and attention.

Most Challenging Parts:

- Designing and implementing appropriate data structures, such as Queues, Stacks, and Doubly Linked Lists, while ensuring their efficiency was particularly demanding.
- Establishing and displaying correct relationships between cities, packages, and vehicles in alignment with the requirements posed another challenge.

What I Liked & What I Learned:

- I found implementing custom data structures and integrating them into the system both challenging and rewarding.
- I enjoyed tackling the problem of parsing and managing data from input files, as well as structuring relationships among cities, packages, and vehicles.
- This project enhanced my understanding of modular program design and the practical application of data structures.
- Additionally, I learned valuable lessons in handling file I/O operations and implementing effective error-handling mechanisms to ensure program reliability.