**CMPE223 – Data Structures and Algorithms**

**Assignment 2 Report**

**Experimental Setup & Procedure**

Firstly, to guess the sorts from 1-5, I had to implement the jar file which has already been uploaded to LMS. Then, I created a class named **SortingAlgorithmTester** as given in the homework format. After that, I created a Random object inside my class to fill high-sized arrays with random variables. I used **random.ints(streamSize, randomNumberOrigin, randomNumberBound).toArray()** method. Also, I had to fill high-sized arrays with ascending and descending numbers; to see sorting algorithms in running times on best case and worst cases. I have done that with 2 for loops, with a simple array-filler algorithm. Below the codes I explained above:

metin içeren bir resim

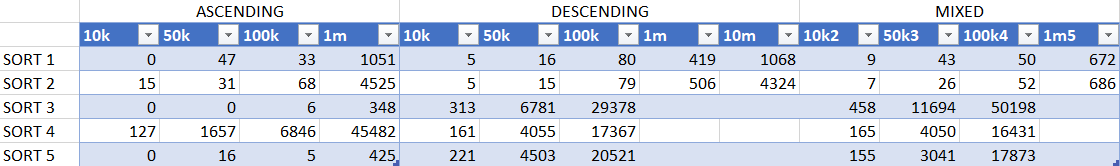
Açıklama otomatik olarak oluşturuldu

After these lines of codes, I ran sorting algorithms with different sized arrays. I specified arrays’ size as 10.000, 50.000, 100.000 and 1 million. However, in order to differentiate Merge and Quick sorts, I had to use 10 million sized descending arrays, as its worst case.

metin içeren bir resim

Açıklama otomatik olarak oluşturulduTo find out the running times of each algorithm, I used these code lines:

This was the output:

After these, my aim is to find out which sorting algorithm represents which one. With the running times I had, it was not hard to do it. Let me first give the table I created with the running times.

I left empty some lines because the running times for those lines were too high.

**Experimental Results**

Easiest sort to guess was selection sort because of its time complexity, it is **O(N2)** for all cases. Therefore, I could easily understand that **Sort-4 was Selection Sort.** It is also can be seen in the graph below:

It may seem like a linear graphic, which is not. The reason it seems like linear is the lack of data. If I did my experiment with more data, the graph would be quadratic.

After that, I understood that Sort1-2 are Merge and Quick sorts. To separate them, I did my experiment on 10 million sized descending arrays. Results were consistent: Sort-1 was Merge and Sort-2 was Quick. Worst case time complexity for Merge Sort is **O(NlogN)**, for Quick Sort it is **O(N2)**. Therefore, worst running time of Quick Sort must have been higher than Merge Sort. These were the results:

I had only 2 sorts left, which were Insertion and Bubble Sorts. One of them was Sort-3, the other was Sort-5. This comparison was the hardest one to make because of the time complexities these sorts have. Both have **O(N), O(N2) and O(N2)** for best – average and worst cases. However, Bubble Sort runs 2x slower than Insertion Sort with Tilda Notation. With the usage of this information, I understood that Sort-5 was Insertion and Sort-3 was Bubble.