Comparing running time for three sorting algorithms

Daniela Gjorgjieva, Elisa Spinelli

Experiment 1, Experimentation & Evaluation 2022

Abstract

In this experiment we monitor the execution times of three sorting algorithms using a few arrays as input. We decided to create arrays of Integers, Strings and Doubles to check how the three algorithms would perform with different data types, and to run the code with and without an IDE to check if the performance would change.
When we analyzed the results we discovered that the execution time performs better when running with an IDE like Visual Studio code than only with a Shell terminal like the Dell's. After examining those results visually and on different computers we came to the conclusion that out of all the three algorithms, the one named "BubbleSortUntilNoChange" has the best execution time on every machine we ran Sorter.java.

1. Introduction

Bubble Inc. is a company developing a library providing utilities to Java developers. Because it is a big company it must give efficient and accurate algorithms. It needs to find the most suitable algorithm between the three sorting algorithms that we are given.
Rearranging an array or list of elements according to a comparison operator on the elements is done using a sorting algorithm.There are many ways to complete this experiment but we used the following method.We are comparing the execution times of each method using 18 arrays of various sizes, of which 6 contain Integers, 6 contain Strings, and another 6 contain Doubles.

Hypotheses:

We presume that all three algorithms have the same execution time. We decide to consider the size and type of the arrays as independent variables; whether the array is already sorted, is inverted, or is filled at random; executing the algorithms with an IDE — VisualStudioCode — or without one — Shell Terminal
We will use the performance of the three algorithms in terms of nanoseconds as a dependent variable and with the help of these independent variables

2. Method

First we declared our arrays, and we populated the ordered integers arrays of 100 elements and 1000 elements and the reversed integers arrays of 100 and 1000 elements by using for loops.

In order to get the ordered and reversed String arrays we used the built in function sort, calling it with the comparator reverseOrder() for the reversed ones.
Then we added them in an array allIntegers that is filled with all the arrays of Integers, and another array with the array of Strings.
We used this approach in order to pass more easily all the arrays to the functions we will use later for testing the three algorithms.

2.1 Variables

Independent variable	Levels
Algorithms	3
Array data type	3 - Integer, String and Double
IDE	VisualsStudio
Size of an array	100 and 1000
Array Sorted	Reversed- Ordered- Random
IDE	Visual Studio Code

Dependent variable	Measurement Scale
3 algorithms	Ns - nanoseconds

Control variable	Fixed Value
Type of Laptop	Dell Precision 5540
Size of Strings	4 characters - 5 characters
Precision of Doubles	2 integers after the comma
NanoTime Execution Time	Default it’s 100 nanoseconds
Warm up cycles	50

2.2 Design

Type of Study :

☐ **Observational Study**	☐ **Quasi-Experiment**	✓ **Experiment**

Number of Factors :

☐ **Single-Factor Design**	✓ **Multi-Factor Design**	☐ Other

2.3 Apparatus and Materials

For the array of Integers and Strings of 100 and 1000 elements we used the website https://www.random.org/ to get the values and then we used the website https://arraythis.com/ to get those values in the form of an array.

For the array of Doubles we used the same approach, but with a different website (https://www.meridianoutpost.com), as https://www.random.org/ was not generating doubles.

We used the terminal of our PCs which are Dell with Ubuntu as an operating system- version 20.04.3 and version 20.04.5. For running the program with an IDE we used Visual Studio code with version 1.67.1, and version 1.73.1.
We observe that with more updated versions we get better results for the execution time of all three sorting algorithms.

The full links for the websites can be found in the appendix.

2.4 Procedure

In order to start our program you need to write java Sorter in the terminal.

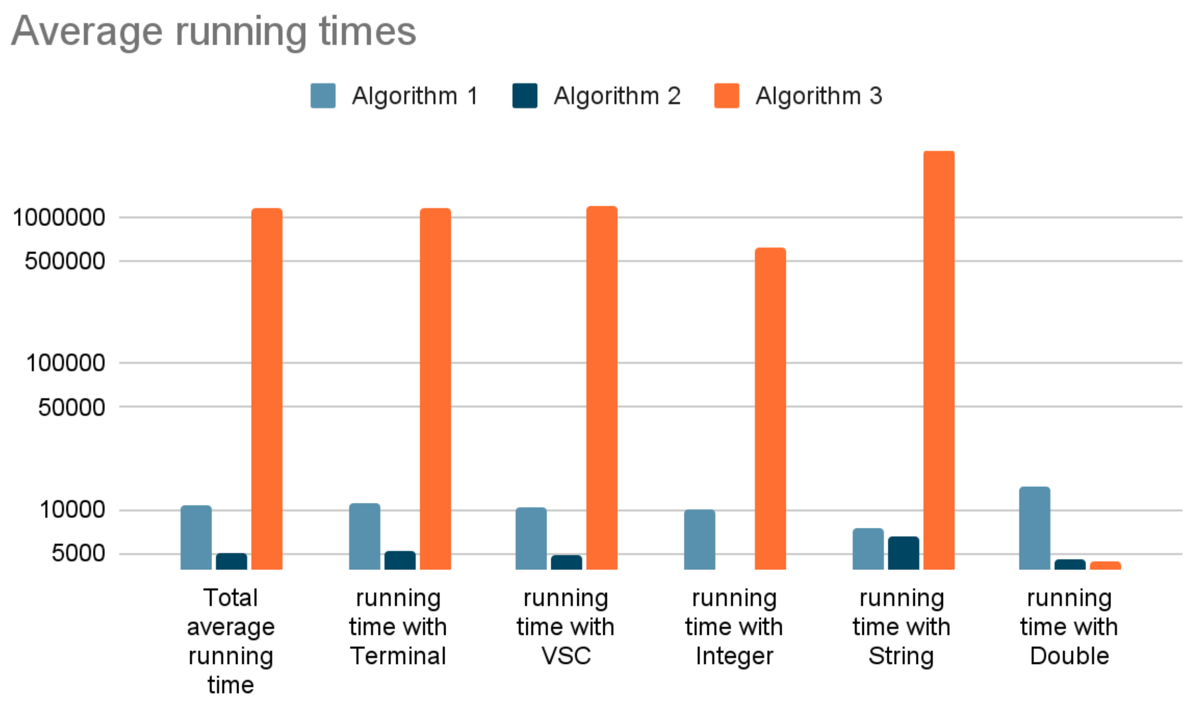
For our procedure first of all we declared three types of an Array, as we previously mentioned that are populated with random integers, strings and doubles.
We created an algorithm that accounts for the various data and ran the three distinct algorithms for each of them while keeping track of the execution times. The algorithms run in the following order:
BubbleSortWhileNeeded()
BubbleSortPassPerItem()

BubbleSortUntilNoChange()
The methods for each of these algorithms run the sort function on each of the arrays in the allInteger, allString, and allDouble arrays and store the average time it took to run the sort function in the timeA1 array.
The number of iterations is set to 500, and before beginning the execution time measurements for each method, there is a warm-up cycle of 50 iterations. To record the duration of each algorithm's execution, we made use of three separate arrays, one for each algorithm. At the conclusion of the execution, the method publishes all the data discovered in the same order as the combinations in the Google Sheet, making it simple to relate each result to its associated set of independent variables.

3. Results

3.1 Visual Overview

The following graphs were computed using Excel, and they represent the average time of the algorithms using different versions of the same PCs and different machines.
Algorithm 1 is "BubbleSortWhileNeeded", Algorithm 2 is "BubbleSortUntilnoChange", Algorithm 3 is "BubbleSortPassPerItem".



3.2 Descriptive Statistics

The best algorithm that executes with the fastest execution time produces the minimum result which in this case is “BubbleSortUntilNoChange”. The algorithm that takes the maximum execution time is “BubbleSortPassPerItem”. And the “BubbleSortWhileNeeded” coincides with the median between these algorithms.

4. Discussion

4.1 Compare Hypothesis to Results

The results show that the algorithms run more quickly on the IDE Visual Studio Code than on the Shell terminal, some of them run faster when given an array of strings as input, and—most importantly the algorithm which is BubbleSortUntilNoChange() runs faster on average than the other two algorithms. Our hypothesis was incorrect because we see visually that one of the algorithms performs better in terms of execution time compared to the others. This led to the conclusion that one algorithm must perform better in terms of execution time while the other two perform worse.

4.2 Limitations and Threats to Validity

First limitation that we came to notice is that while using the random generator website, we had some faults getting the results we wanted for integers, Strings and doubles. There are distances between where certain values occur and are distributed differently from those in a random sequence distribution.
There was also a slight problem we encountered in the excel, when we were trying to compute the average time for the doubles, we didn’t get in the graph exactly what we wanted for the data type double as you can see.

Second limitation is that while we were running Sorter.java more than 10 times. After a specific number of times we were getting numbers for the execution time that didn’t make any sense. We can say that we tried running on more machines but the execution time varies, and for this we can say that hardware of the computer may be a factor.

4.3 Conclusions

We managed to complete the experiment and we may declare as a conclusion that the algorithm “BubbleSortUntilNoChange” has the best execution time running on almost every machine. It is followed by “BubbleSortWhileNeeded” and “BubbleSortPassPerItem” performing as the worst out of all three.

Appendix

A. Materials

https://www.random.org/
https://arraythis.com/
https://www.meridianoutpost.com/resources/etools/calculators/generator-random-integer-real-numbers.php

B. Reproduction Package (or: Raw Data)

We created a github repository, and it’s included in this report:
https://github.com/DanielaGjorgjieva/Evaluation_and_Experimentation_Experiment1