Your Title Here

Experiment 1, Experimentation & Evaluation 2024

# Abstract

Short (120-130 words) summary of your entire report. Give the reader a quick idea of what you did and what the main findings were (if you prepare this report ahead of time, leave out the findings until after you finish the analysis).

# 1. Introduction

The topic of investigation of our experiment is the performance, more specifically the execution time required by four different sorting algorithms (the lower the better). The latter are: BubbleSortUntilNoChange, BubbleSortWhileNeeded, QuickSortGPT and SelectionSortGPT.

The motivation for this study stems from the need of a company to decide which implementation of sorting algorithm to include in the Java library they are developing.

**# TODO: OUTLINE OUR PROPOSED EXPERIMENT**

|  |
| --- |
| **Hypotheses:** |
| QuickSortGPT is the sorting algorithm that delivers the best performance across all array types, sizes and data orderings, since it is the one with the better time complexity. |

# 2. Method

In the following subsections, describe everything that a reader would need to replicate your experiment in all important details.

## 2.1 Variables

The independent variables (i.e. the values changed during the experiment) are the following: the sorting algorithm, the type of data, its orderings in the array and the array size.

|  |  |
| --- | --- |
| **Independent variable** | **Levels** |
| Sorting algorithm | BubbleSortUntilNoChange, BubbleSortWhileNeeded, QuickSortGPT and SelectionSortGPT |
| Data Type | Integer and String |
| Data Orderings in the array | Best case: the data in the array are already sorted  Average case: the data in the array are in random ordering  Worst case: the data in the array are reverse sorted |
| Array Size | Small: 100  medium: 10’000  Large: 100’000 |

The dependent variable (i.e. what is measured in the experiment) is the execution time of the sorting algorithms.

|  |  |
| --- | --- |
| **Dependent variable** | **Measurement Scale** |
| Execution time | Ratio scale (in ns) |

The control variable(s) (i.e., what is kept constant during the experiment) are the following:

hardware, Operating System, running applications, JDK, and warmup.

|  |  |
| --- | --- |
| **Control variable** | **Fixed Value** |
| Hardware | Model: Dell Inc. Precision 5570  Memory: 16.0 GiB  Processor: 12th Gen Intel® Core™ i7-12700H × 20  Graphics: 1: Intel® Graphics (ADL GT2)  Graphics: NVIDIA RTX A1000 Laptop GPU  Disk Capacity: 512.1 GB |
| Operating System | OS Name: Ubuntu22.04.5 LTS  OS Type: 64-bit  GNOME VERSION: 42.9  Windowing System: X11 |
| Running applications | IDE and the shell |
| JDK | Java version: OpenJDK 17.0.12  IDE: IntelliJ 2024.2.4 |
| Warmup | 10 |

## 2.2 Design

Check off the characteristics of your experimental design:

**Type of Study** (check one):

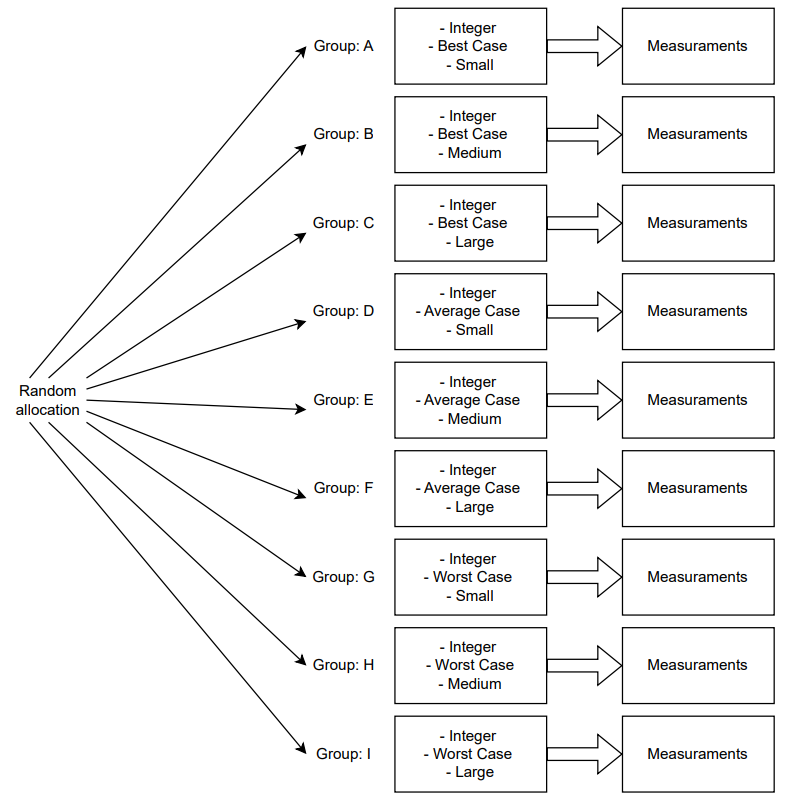
|  |  |  |
| --- | --- | --- |
| **⃞ Observational Study** | ⃞ **Quasi-Experiment** | ⃞ **Experiment** |

**Number of Factors** (check one):

|  |  |  |
| --- | --- | --- |
| ⃞ **Single-Factor Design** | ⃞ **Multi-Factor Design** | ⃞ Other |

The experiment we designed is neither an Observational Study nor a Quasi-Experiment, as we are not looking at a phenomenon in a systematic and scientifically rigorous way in its environment, and we have complete control over manipulation of the independent variables. Therefore, it is an Experiment.

Moreover, it follows a Multi-Factorial Design, since we have more than one independent variable in the study.

Figure 1: Experiment considering one algorithm on one data type

## 2.3 Apparatus and Materials

The relevant “props” used in this experiment are the following:

* A Dell laptop, hardware model Dell Inc. Precision 5570, with Ubuntu20.04.6 LTS as Operating Systems
* OpenJDK 17.0.12
* IntelliJ 2024.2.4
* A background process on the computer that gets automatically triggered (used to measure the time)

## 2.4 Procedure

To perform the experiment, we start our Dell laptop, we make sure all applications are closed, and we open OpenJDK and the shell.

Then, using IntelliJ as IDE and OpenJDK 17.0.12 as java version, we execute each sorting algorithm on every permutation (e.g., Figure 1) and capture the execution time through a background process. For each group, we collect 50 measurements after 30 warmup rounds. This is because relevant statistics can be computed from 30 or more data points and we observed that after the first 30 runs the execution time of the algorithms becomes stable.

After that, for each permutation, we compute the minimum, first quartile, median, third quartile, and maximum values.

Lastly, the sorting algorithm that demonstrates the best performance across all array types, sizes, and data orderings is the one with the highest number of smaller medians across all permutations.

# 3. Results

## 3.1 Visual Overview

Provide an insightful overview of the data you collected. This requires some engineering from your part, to find a good degree of summarization: On one end of the spectrum, you don't summarize, and report hundreds of raw measurement values in a block of text. On the other end of the spectrum, you report a single number (like a mean value). Both approaches are bad.

Instead, use appropriate visual summaries (such as **scatter plots**, **histograms**, **box plots**, or **empirical cumulative distribution functions**) to show the distribution of your data. If you have a very small number of measurement values, then report all of them in a **well organized table** (where rows and/or columns correspond to different levels of different factors).

## 3.2 Descriptive Statistics

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | BubbleSortUntilNoChange | BubbleSortWhileNeeded | QuickSortGPT | SelectionSortGPT |
| Integer  Best case  Small | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| Integer  Best case  Medium | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| Integer  Best case  Large | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| Integer  Average case  Small | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| Integer  Average case  Medium | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| Integer  Average case  Large | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| Integer  Worse case  Small | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| Integer  Worse case  Medium | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| Integer  Worse case  Large | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| String  Best case  Small | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| String  Best case  Medium | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| String  Best case  Large | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| String  Average case  Small | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| String  Average case  Medium | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| String  Average case  Large | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| String  Worse case  Small | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| String  Worse case  Medium | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |
| String  Worse case  Large | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: | Minimum:  First quartile:  Median:  Third quartile:  Maximum: |

The statistics we compute are the following: minimum, first quartile (Q1), median, third quartile (Q3) and maximum.

The minimum is the smallest value in a data set. The first quartile (also known as Q1) is the value below which 25% of the data falls. The median is the middle value of the data (when it is ordered from smallest to largest). The third quartile (also known as Q3) is the below which 75% of the data falls, and the maximum is the highest value in the set.

# 4. Discussion

## 4.1 Compare Hypothesis to Results

Our measurements shows that for the majority of the groups, QuickSortGPT has the better performance (i.e. execution time), as expected in our hypothesis.

The only permutations that are not consistent with our initial assumption are all the ones containing the already sorted array (i.e. best case). That’s because QuickSortGPT’s best case is equal to its time complexity.

## 4.2 Limitations and Threats to Validity

The main limitation in our study coincides with the values we use to represent the dimensions of the array considered (i.e. small, medium and large). Indeed, to really reflect the order of magnitude of the different sizes, our values are not so precise. The correct ones should be e.g. 100, 10’000, 100’000. Our choice was forced by memory constraints.

Despite that, our results are still reliable since the difference between 100 and 10’000 provides an effective size difference.

This problem can be solved in an environment that provides an adequate amount of memory.

## 4.3 Conclusions

This experiment underlines that, among the four candidate algorithms, QuickSortGPT is the best one considering execution time as only criterion for performance.

Despite that, this study shows that QuickSortGPT is not the best for all the input arrays, as it performs badly on already sorted arrays.

End with the main conclusions that can be drawn from your study.

Appendix

# A. Materials

Any documents you used for your informed consent (information sheets, consent) or as part of your apparatus (e.g., manual, hand-out), please include them here.

# B. Reproduction Package (or: Raw Data)

Before, during, and after the experiment you collected all kinds of data. Don't ever throw such data away! Any plots, tables, summaries, and statistics provided in this report should be recreatable from the raw data you have.

If you only collected a small amount of data, put it in this Appendix right here.

If you collected data in forms that are better kept in separate files, then zip up those files, and submit them as a "reproduction package" supporting this report.