Performance Measurement for Bubble Inc

By Lavrov, Ioffe, Biletskyi

Experiment 1, Experimentation & Evaluation 2024

Abstract

In the following experiment report you will find the results of testing 4 sorting algorithms:

- BubbleSortUntilNoChange
- BubbleSortWhileNeeded
- QuickSortGPT
- SelectionSortGPT

The results include:

- Comparative results of all algorithms included
- Analysis of testing environment
- Analysis of forces that can affect the results
- Situational analysis of used algorithms

1. Introduction

This experiment is targeted towards identifying the best sorting algorithm among 4 algorithms that participate in this experiment. The algorithms will be tested on simple data types that are present for each sorted object:

- Data types presented in experiment are int, long, float, double, char, string
- The data object will always hold a non-null value

Hypotheses:
The QuickSortGPT algorithm is expected to perform better than others, due to its average time complexity being O(nlogn) in comparison to average time complexity of O(n^2) of other presented algorithms. Experiment assumes that testing in different environments can yield highly different result times, but the difference will be due to available processing powers of environments and tendencies of algorithm performance will be the same.

2. Method

In this experiment, the testing was performed on 2 different machines:

- HP Omen 14 - Intel Core Ultra 9 185H processor, 32 GB RAM
- Macbook pro 14 - Apple M2 processor, 16 GB RAM

It is assumed that not 100% of processing power was available for a machine as it was running a corresponding OS and it together with necessary background processes could have consumed part of available resources (not more than 25%). The code for experiment and IDE versions will be available in Appendix section B.

It should also be noted, used machines are assumed to have more processing power than it is needed to perform the testing in a scale used for experiment, therefore their performance will be limited not by processing powers, but by speed with which information can travel inside a machine. Therefore HP Omen 14 and Macbook pro 14 will yield results close to each other. If the scale of experiment will be further increased, at some point the Macbook pro 14 may significantly fall back due to lower available resources.

2.1 Variables

Independent variable	Levels
Type of sorting algorithms	BubbleSortUntilNoChange, BubbleSortWhileNeeded, QuickSortGPT, SelectionSortGPT
Number of elements in tested array	100, 1 000, 10 000
Data type	Int, long, float, double, char, string

Dependent variable	Measurement Scale
Execution time	Execution time is measured in milliseconds and represents time it took for identified machine to complete sorting

Control variable	Fixed Value
Processing power	HP Omen 14: 32 GB RAM, 5.1 GHz processor, 16 cores / 22 threads Macbook 14 pro: 16 GB RAM, 3.2 GHz processor, 8 cores / 8 threads
Software environment	HP Omen 14: Java JDK 21, Intellij Idea, Windows OS Macbook 14 pro: Java JDK 21, Intellij Idea, Mac OS
Execution conditions	Minimal background processes and unlimited resource usage for algorithms

2.2 Design

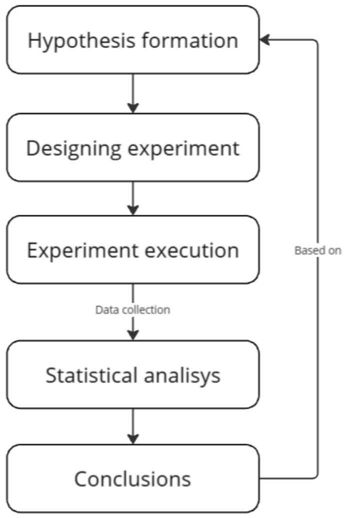
Type of Study (check one):

Observational Study	Quasi-Experiment	✓ Experiment

Number of Factors (check one):

Single-Factor Design	✓ Multi-Factor Design	Other

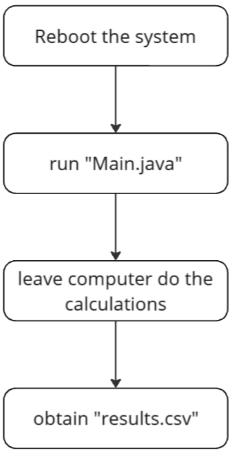
The experiment performed is a controlled experiment in a full controlled environment. Process can be illustrated on the image below



2.3 Apparatus and Materials

	Experiment 1 (Windows)	Experiment 2 (MacOs)
Computer and specifications	HP Omen 14, Ultra 9 185H processor and 32GB RAM	Macbook Pro 14 with an M2 processor and 16GB RAM.
Software	Java JDK 21, Intelij Idea	Java JDK 21, Intelij Idea
Time tracking	System.nanoTime()	System.nanoTime()

2.4 Procedure



Before each session reboot of the system must be done. Participant must unsure their computer is plugged to source the whole duration of the experiment.
No other heavy processes must be running on the system.
Experiment runs automatically by executing a “Main.java” file.
After script is executed, participant receives results in file “results.csv”.

Session took 22.4 minutes on HP and 20.2 on Mac.

3. Results

3.1 Visual Overview

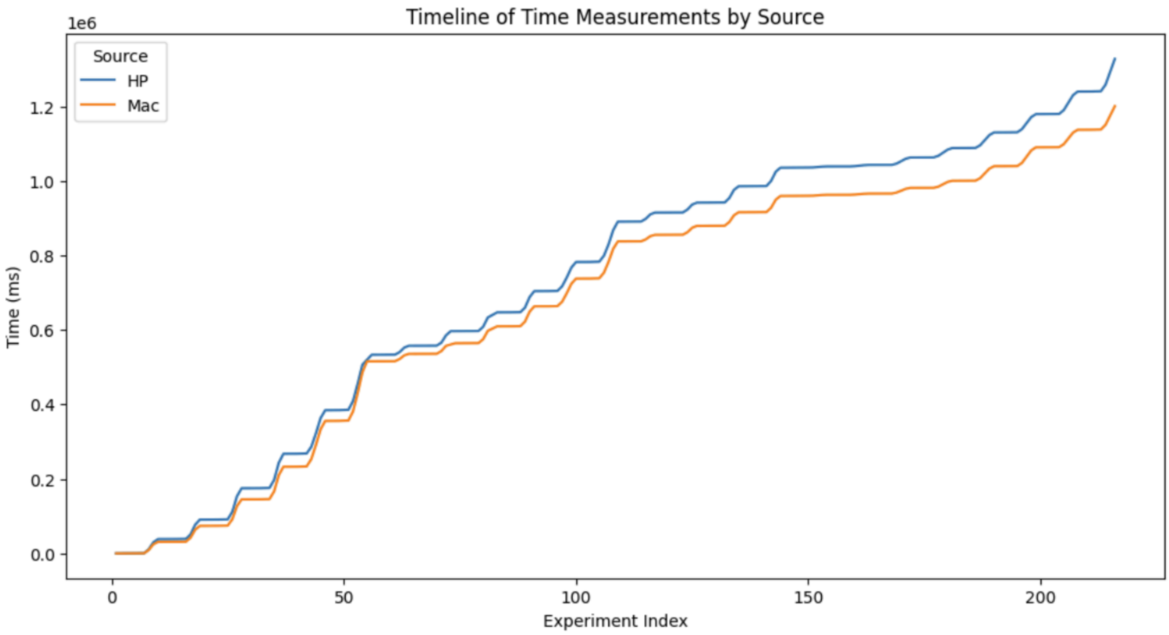
In the outcomes of the experiment, QuickSortGPT (later QSgpt) and SelectionSortGPT (later SSgpt) appeared to be superior for all tested data types. However they were best performing at different data types.

	Best data types	Analysis
QSgpt	Char, String	The QSgpt showed the fastest times to perform sorting on arrays of larger scales with non-numerical data types. While some algorithms can be faster in certain cases like low amount of elements to sort, or when the initial array is almost sorted, the QSgpt showed a great scalability with growth in time consumed severely lower than other algorithms. For arrays of extraordinary sizes, the QSgpt has absolute superiority over 3 other algorithms. It proved to be a reliable and scalable algorithm that can handle all array sizes in an efficient manner.
SSgpt	Int, Long, Foat, Double	The SSgpt showed the fastest times to perform sorting in arrays of larger scales with numerical data types. It was slightly slower than other presented algorithms for small size arrays, however at 10 000 elements it already outperforms other algorithms by a significant margin. Its growth of time consumed in relation to sorted elements is severely slower than other algorithms, which means with growth of elements in array, its efficiency will only grow in relation to other 3.

3.2 Descriptive Statistics

For comparison of actual outcomes of 4 sorting algorithms, their results were concluded in the tables in sections 3.2.1 - 3.2.4. The comparing metrics is ms - milliseconds it took the algorithm to finish sorting a given array. Each algorithm ran 300 tries of different cases for each N - number of elements in the array, after which statistics was created. All the statiscs based on results from laptop HP Omen 14*.

* - While the results from Macbook may have different times, they hold the same tendency in relative comparison. Therefore HP was chosen for slightly richer information as it is using the most common OS for any settings - Windows OS.


Timeline of Time Measurements by Source
1e6
Source
HP
Mac
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Time (ms)
0
50
100
150
200
Experiment Index

3.2.1 BubbleSortUntilNoChange

For all the tables below:
N - represents number of elements in array
Quartile - represents the value below which and including it, the given % of data falls into.
The table is measured in milliseconds (ms)

Int

N \ quartile	Min	25%	50%	75%	Max
100	0.01	0.01	0.02	0.06	0.65
1000	0	0	1.5	1.79	4.29
10000	0.01	0.01	169.9	199.0575	247.58

Long

N \ quartile	Min	25%	50%	75%	Max
100	0.01	0.01	0.06	0.06	0.35
1000	0	0	1.99	2.24	5.43
10000	0.01	0.03	235.95	270.5025	389.94

Float

N \ quartile	Min	25%	50%	75%	Max
100	0	0	0.06	0.08	0.15
1000	0	0.01	3.08	3.61	7.18
10000	0.03	0.05	396.685	425.3	530

Double

N \ quartile	Min	25%	50%	75%	Max
100	0	0	0.04	0.04	0.07
1000	0	0	3.72	4.07	8.28
10000	0.03	0.03	425.51	479.08	536.39

Char

N \ quartile	Min	25%	50%	75%	Max

	100	0.02	0.03	0.03	0.03	0.08
	1000	2.93	3.17	3.36	3.8025	7.87
	10000	313.7	354.4575	381.405	415.565	511.81

	N \ quartile	Min	25%	50%	75%	Max
String	100	0.03	0.03	0.04	0.05	3.01
	1000	4.21	4.52	4.65	5.085	14.39
	10000	416.07	451.14	482.105	526.4675	713.84

3.2.2 BubbleSortWhileNeeded

Int

N \ quartile	Min	25%	50%	75%	Max
100	0	0	0.03	0.04	0.11
1000	0	0	0.99	1.3	15.93
10000	0.01	0.01	104.21	129.16	160.31

Long

N \ quartile	Min	25%	50%	75%	Max
100	0	0	0.02	0.02	0.22
1000	0	0	1.24	1.4225	3.12
10000	0.01	0.01	156.81	210.8	306.92*

* - In the experiment, there was a single entity of 236478.5 ms. It is unsure how a single sample reached such a significant time, but it was removed from statistics, however it should be taken into account that there is a possible flow of an algorithm that caused it.

Float

N \ quartile	Min	25%	50%	75%	Max
100	0	0	0.03	0.03	0.18
1000	0	0	1.89	2.0525	6.34
10000	0.02	0.03	218.805	267.68	348.25

Double

N \ quartile	Min	25%	50%	75%	Max
100	0	0	0.02	0.03	0.08
1000	0	0	2.13	2.34	5.77
10000	0.03	0.03	240.24	302.49	418.82

Char

N \ quartile	Min	25%	50%	75%	Max
100	0.02	0.02	0.02	0.02	2.05

1000	1.98	2.05	2.2	2.35	6.85
10000	207.18	229.8575	246.74	283.6675	359.41

String

N \ quartile	Min	25%	50%	75%	Max
100	0.02	0.02	0.03	0.04	0.05
1000	2.61	2.85	3.12	3.7825	25.83
10000	272.28	297.005	318.775	439.3325	693.48

3.2.3 QuickSortGPT

Int

N \ quartile	Min	25%	50%	75%	Max
100	0.01	0.02	0.03	0.17	0.63
1000	0.05	0.12	0.96	1.27	3.76
10000	0.71	0.78	93.905	133.4075	218.36

Long

N \ quartile	Min	25%	50%	75%	Max
100	0.01	0.01	0.02	0.03	0.48
1000	0.06	0.07	0.92	1.46	3.03
10000	0.82	0.9	99.485	148.65	217.64

Float

N \ quartile	Min	25%	50%	75%	Max
100	0.01	0.01	0.03	0.03	0.06
1000	0.07	0.18	1.77	2.16	9.51
10000	1.04	1.2175	193.89	227.1675	277.98

Double

N \ quartile	Min	25%	50%	75%	Max
100	0	0.01	0.02	0.02	0.06
1000	0.08	0.09	1.94	2.44	4.1
10000	1.08	1.795	210.45	264.19	338.25

Char

N \ quartile	Min	25%	50%	75%	Max
100	0	0	0	0.01	0.01
1000	0.13	0.13	0.14	0.15	2.77
10000	9.67	10.02	10.3	11.14	21.59

String	

N \ quartile	Min	25%	50%	75%	Max
100	0	0	0.01	0.01	0.03
1000	0.15	0.16	0.165	0.18	0.61
10000	11.26	11.9175	12.39	13.565	38.63

3.2.4 SelectionSortGPT

Int

N \ quartile	Min	25%	50%	75%	Max
100	0.02	0.02	0.02	0.02	0.26
1000	0.5	0.54	0.58	0.59	1.62
10000	57.87	60.41	64.535	69.91	97.4

Long

N \ quartile	Min	25%	50%	75%	Max
100	0.02	0.02	0.02	0.02	1.63
1000	0.69	0.7	0.71	0.73	2.15
10000	66.19	75.7575	81.28	88.6825	141.24

Float

N \ quartile	Min	25%	50%	75%	Max
100	0.02	0.02	0.02	0.02	0.07
1000	1.19	1.23	1.26	1.31	3.17
10000	120.95	128.305	134.38	145.205	203.51

Double

N \ quartile	Min	25%	50%	75%	Max
100	0.02	0.02	0.02	0.02	0.18
1000	1.44	1.49	1.515	1.56	5.87
10000	142.26	152.2775	160.64	168.2225	209.75

Char

N \ quartile	Min	25%	50%	75%	Max
100	0.02	0.02	0.02	0.02	0.02
1000	1.73	1.74	1.77	1.8525	6.47
10000	178.02	189.23	196.855	207.9175	247.41

String	<table><tr><td>N \ quartile</td><td>Min</td><td>25%</td><td>50%</td><td>75%</td><td>Max</td></tr><tr><td>100</td><td>0.03</td><td>0.03</td><td>0.03</td><td>0.03</td><td>0.07</td></tr><tr><td>1000</td><td>2.83</td><td>3</td><td>3.12</td><td>4.0525</td><td>21.29</td></tr><tr><td>10000</td><td>305.41</td><td>329.35</td><td>343.15</td><td>359.18</td><td>703.09</td></tr></table>

4. Discussion

4.1 Compare Hypothesis to Results

The hypothesis was partially proven by the outcomes of the experiment. It was proven to be true that QuickSortGPT will perform best for all non-numerical data types, and proven to be untrue for numerical data types. It is unsure how the SelectionSortGPT algorithm with average time complexity of O(n^2) may surpass the QuickSortGPT algorithm with average time complexity of O(nlogn). However, it is not an exception that occurred once, as it is a tendency for all the numerical data types.

It is possible due to the SelectionSorGPTt algorithm going through an array to find the smallest automatically, which is easy to do for numerical data types. Other possible reason is imperfect coding for given algorithms.

4.2 Limitations and Threats to Validity

- It is possible that algorithms presented in the experiment may have flaws, as the results didn’t line up with hypotheses based on the mathematical research.
- The * case of BubbleSortedWhenNeeded with long data types (a single element had a significant execution time) may represent the undetected flaw.
- The data used for analysis may be not full which will damage or invalid the results of this experiment

4.3 Conclusions

For low scale arrays it is a low difference in which algorithm to use, because O(nlogn) O(n) O(n^2) time complexities are very close to each other and end result differences are insignificant taking into consideration average processing powers of modern PCs and professional working stations. However with increasing in scale, only QuickSortGPT for non-numerical data types and SelectionSortGPT for numerical data types are viable, because their growth in average consumed time is massively less exponential than other algorithms.

Appendix

A. Materials

This document uses algorithms provided in attached “Algorithms” folders.
Hand-out is in “experiment-01.pdf”.

B. Reproduction Package (or: Raw Data)

This document supports and uses the data in the attached “Experiment-1-Performance-Measurement.zip” file.

To start experiment file “Main.java” should be executed. It uses “ArrayGenerator.java” and files found in “Algorithms” folder.

After finishing the task results can be found in “results.csv” file.

Results for hpo14 is in “results-hpo14.csv”.

Results for m2pro is in “results_m2pro.csv”.

“Results hpo14 PROCESSED.xlsm” has the same content as Results hpo14, additionally processed for ease of representation in report. Was processed using macros found in file “MacrosForProcessing.txt”

Various graphs and results of data analysis, among with explanations can be found in Jupiter notebook file “analysis.ipynb”. It has useful insights, but not everything made it to the report due to lack of relevancy in context of the experiment.