Best Algorithm for Java Developers

Experiment 1, Experimentation & Evaluation 2024

Abstract

The following experiment evaluates the performance for the four Java sorting algorithms: BubbleSortUntilNoChange, BubbleSortWhileNeeded, QuickSortGPT, and SelectionSortGPT. This set of algorithms will be evaluated under several factors that may influence their execution time, namely:

data type (integer and string);
data ordering (best case, average case, and worst case); and
array size (small, medium, and large).

Each of the sorting algorithms was run across all test cases, and data was gathered for the thorough comparison of its performance. This experiment aims to identify the algorithm with the fastest execution.

The results are based on median performance of many repetitions so that every condition gets a fair evaluation, and the efficiency of each algorithm was evaluated across many dimensions.

This experiment will confirm QuickSortGPT to be the best algorithm based on execution time.

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.

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. Thus, it will outperform the other algorithms.

To run this experiment, we first consider the identification of variables that can be used to influence the outcome, including those of specifications for the test machine itself.

We proceed by testing each algorithm across various data permutations and comparing the results through data analysis and graphical comparison.

The best algorithm is then determined based on median execution times across all configurations, ensuring a fair and consistent performance.

2. Method

The following subsections provide all essential details required to replicate the experiment accurately.

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: 1’000 Large: 10’000

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: Ubuntu24.04.1 LTS

	OS Type: 64-bit GNOME VERSION: 46 Windowing System: Wayland
Running applications	IDE and the shell
JDK	Java version: OpenJDK 17.0.12 IDE: IntelliJ 2024.2.4
Warmup	30

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 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.

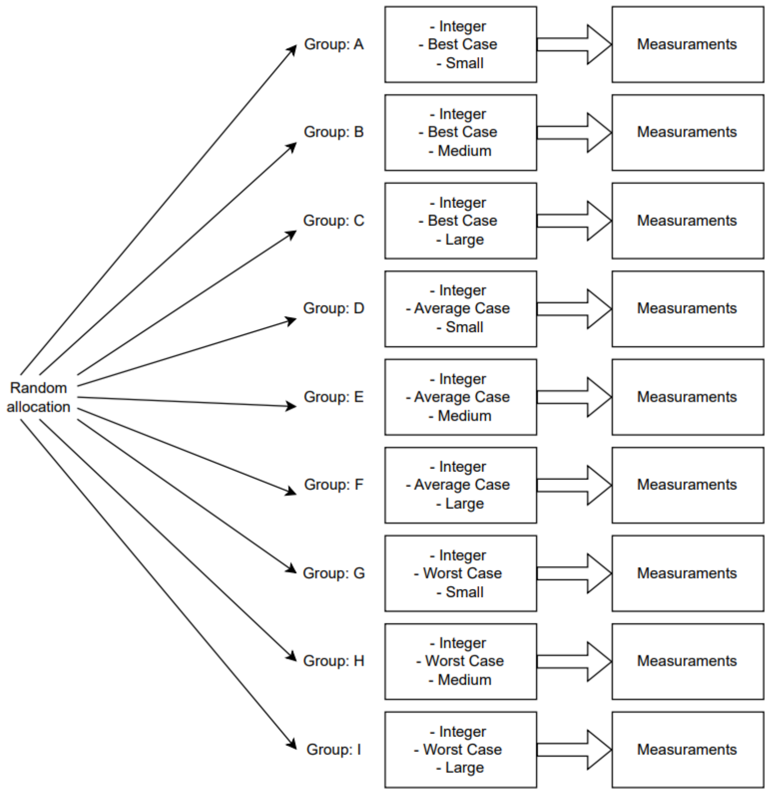


Figure 1: Experiment considering one algorithm on one data type

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 Ubuntu24.04.1 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.

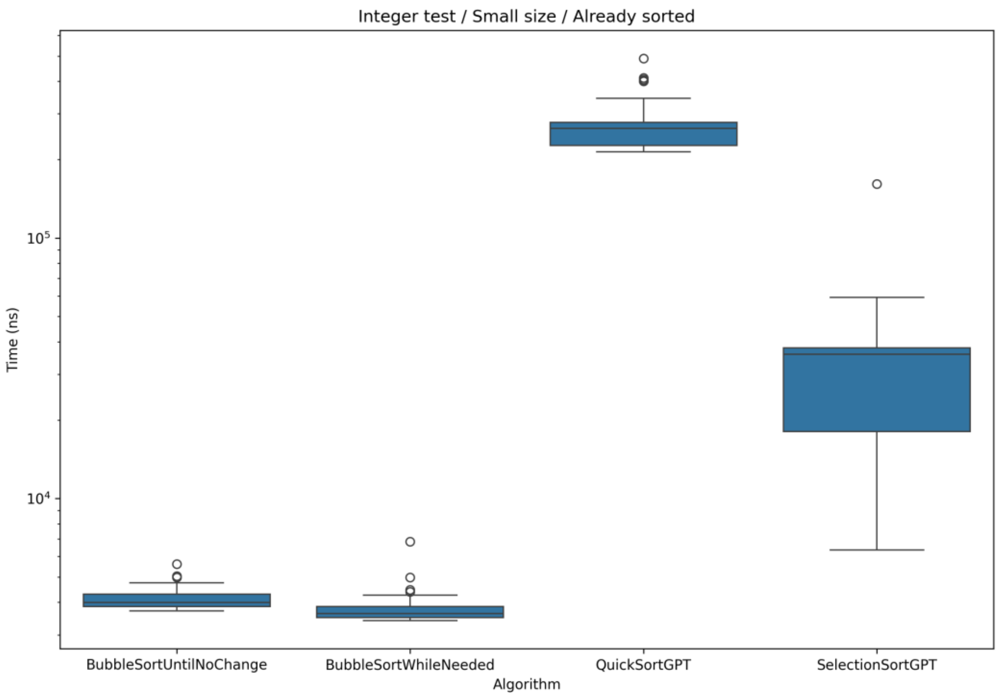
Then, using IntelliJ as IDE and OpenJDK 17.0.12 as java version, we execute each sorting algorithm on every permutation (e.g., Figure 1) and capture the execution time through a background process. For each group, we collect 50 measurements after 30 warmup rounds. This is because relevant statistics 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.

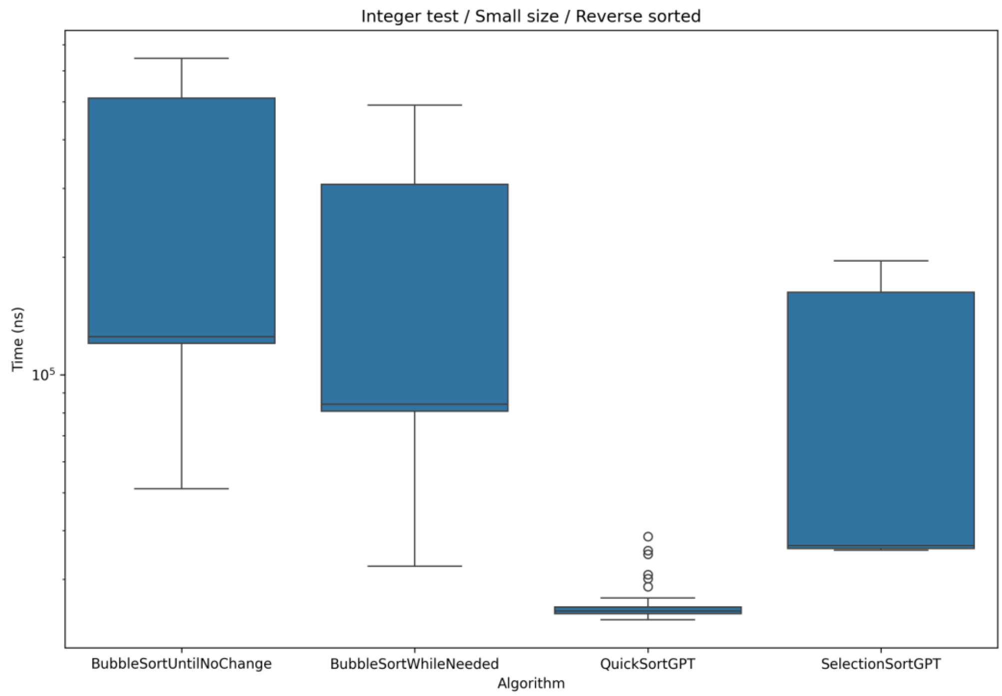
Lastly, the sorting algorithm that demonstrates the best performance across all array types, sizes, and data orderings is the one with the highest number of smaller medians across all permutations.

3. Results

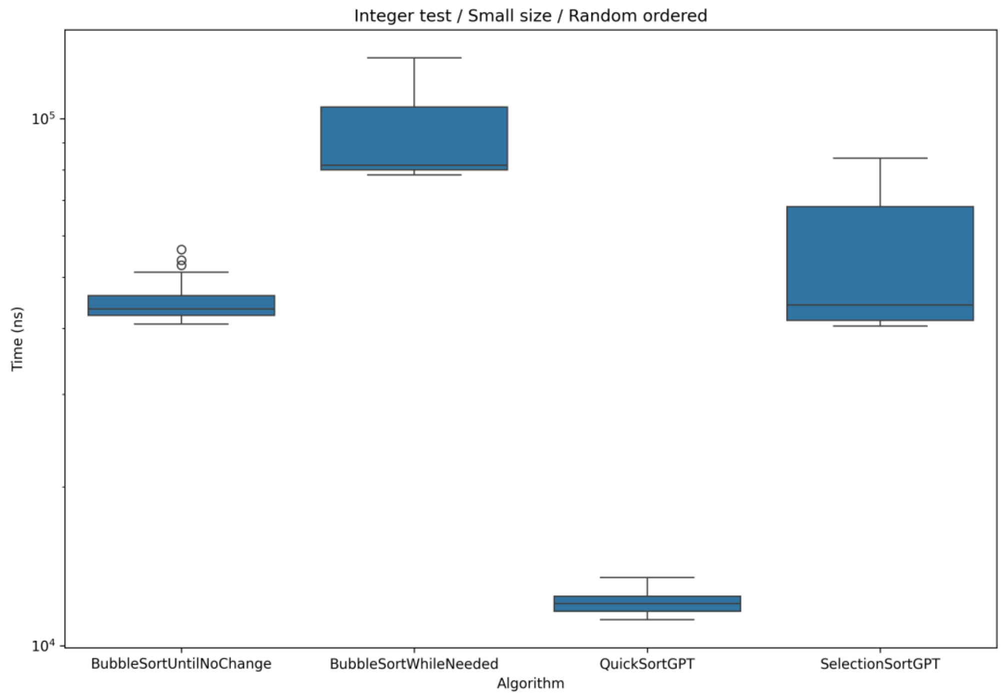
3.1 Visual Overview

Here are reported the box plot summarizing the results of all the permutations.

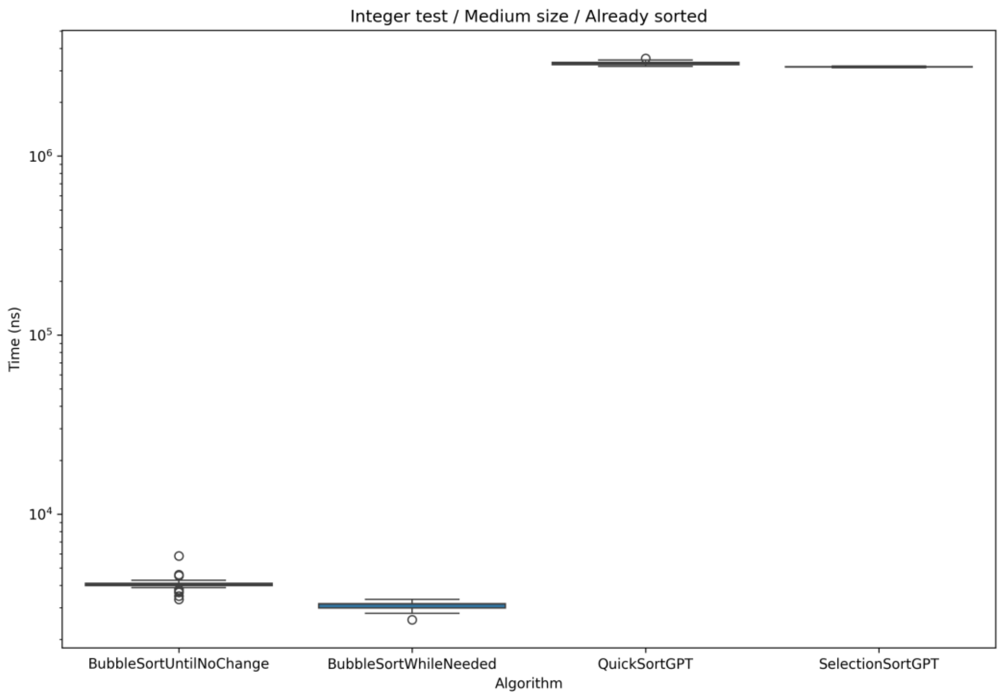




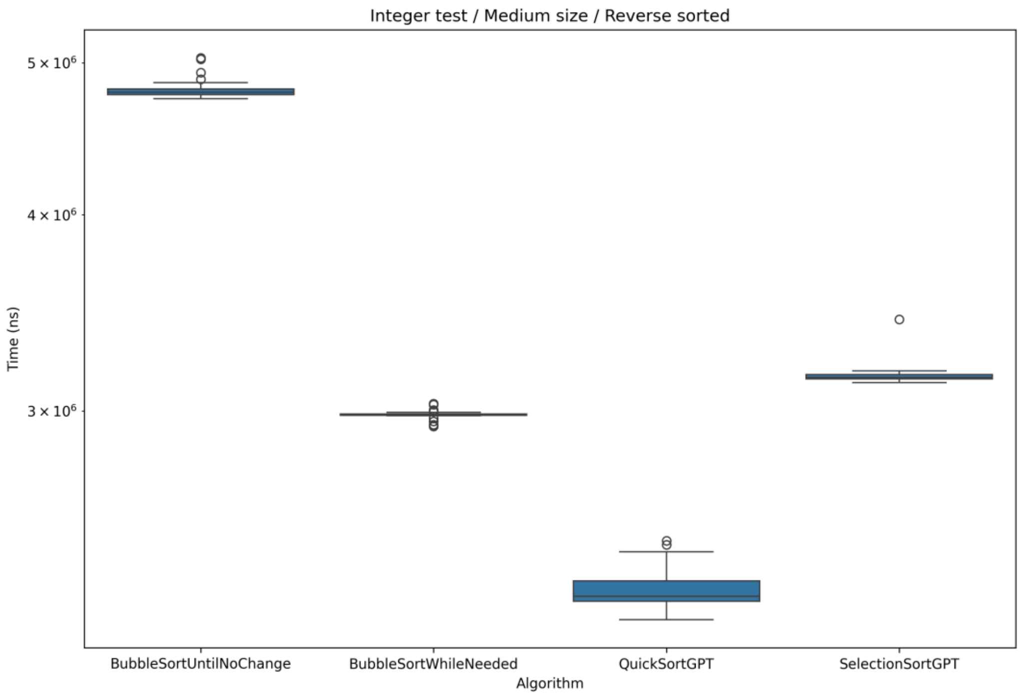
Integer test / Small size / Reverse sorted
Time (ns)
10^5
BubbleSortUntilNoChange
BubbleSortWhileNeeded
QuickSortGPT
SelectionSortGPT
Algorithm



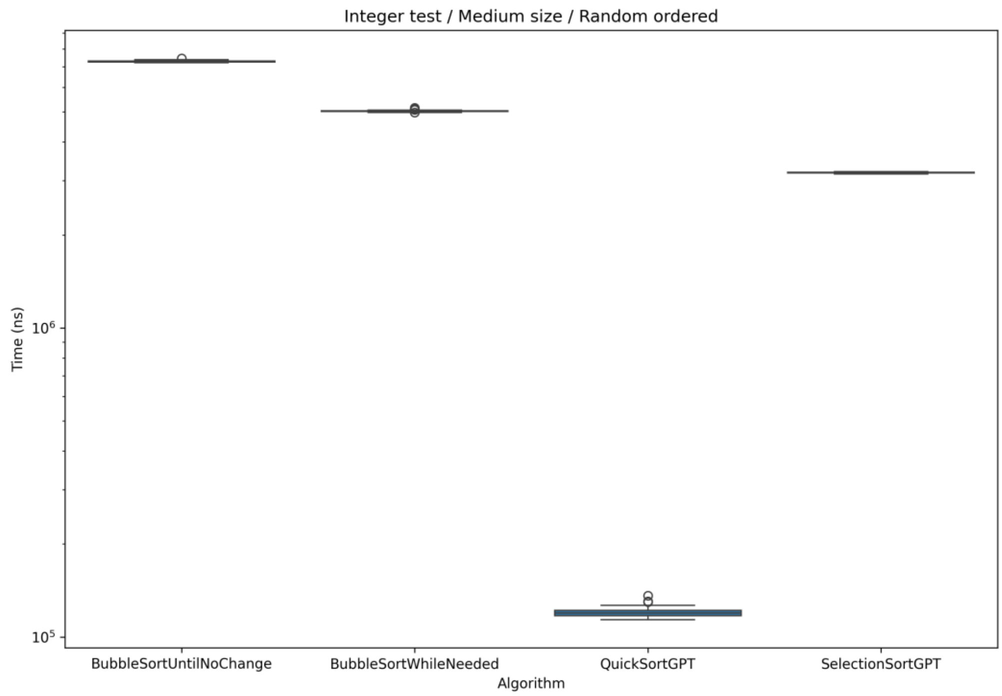
Integer test / Small size / Random ordered
Time (ns)
10^5
10^4
BubbleSortUntilNoChange
BubbleSortWhileNeeded
QuickSortGPT
SelectionSortGPT
Algorithm

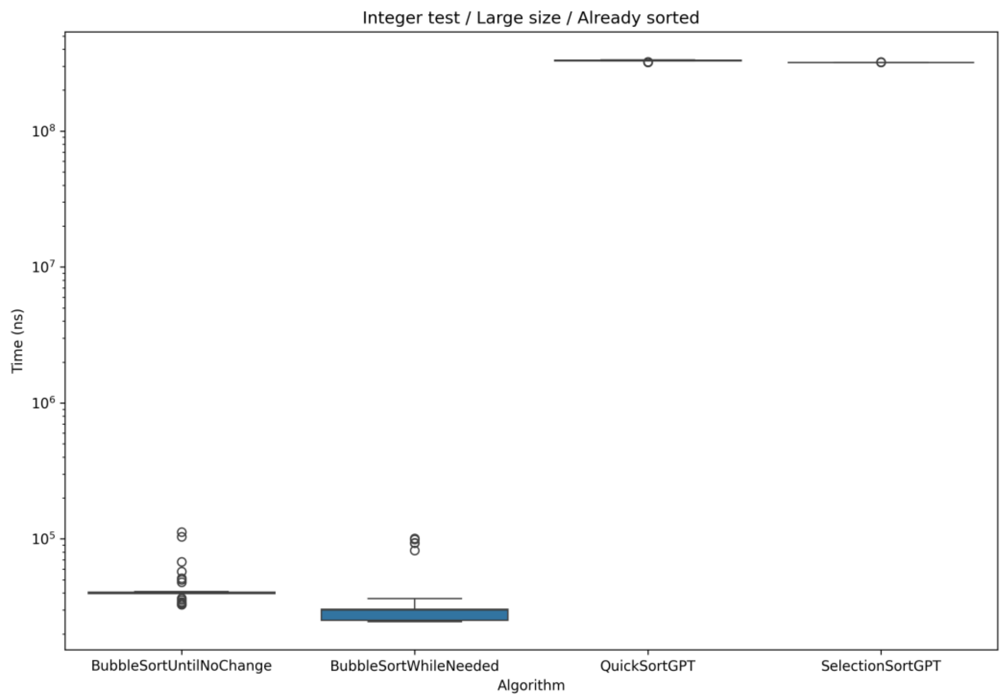


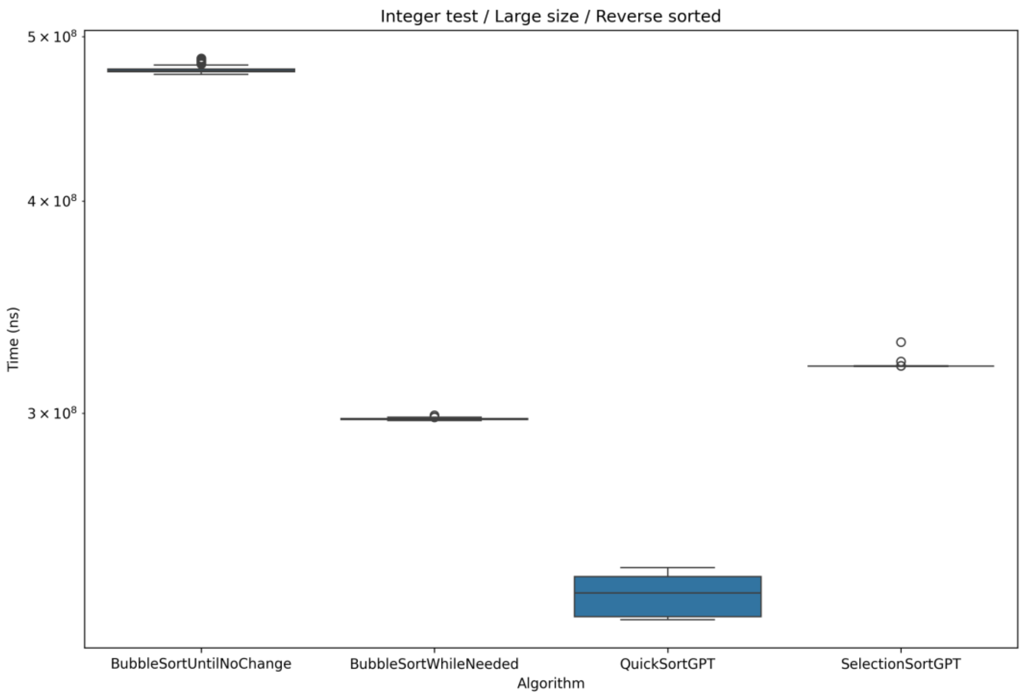
Integer test / Medium size / Already sorted
Time (ns)
10^6
10^5
10^4
BubbleSortUntilNoChange
BubbleSortWhileNeeded
QuickSortGPT
SelectionSortGPT
Algorithm

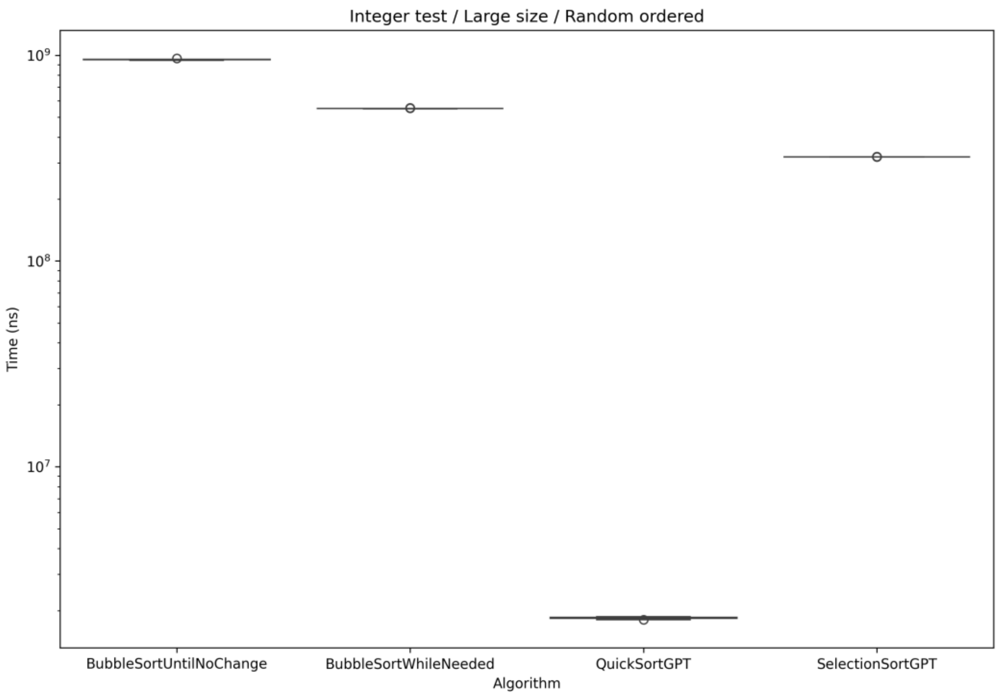


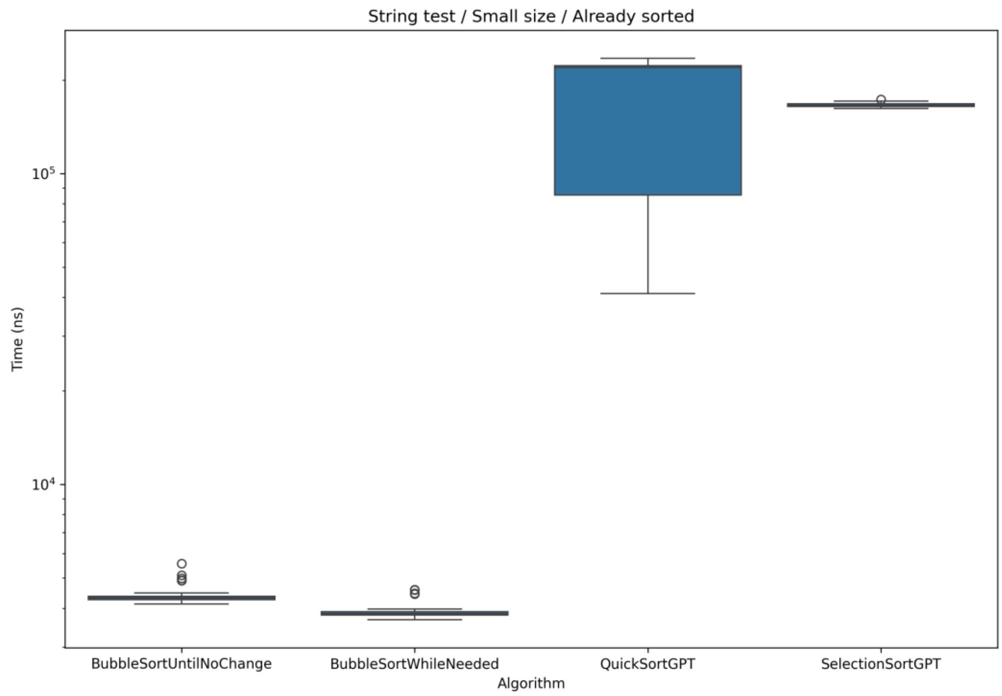
Integer test / Medium size / Reverse sorted
Time (ns)
5×10^6
4×10^6
3×10^6
BubbleSortUntilNoChange
BubbleSortWhileNeeded
QuickSortGPT
SelectionSortGPT
Algorithm

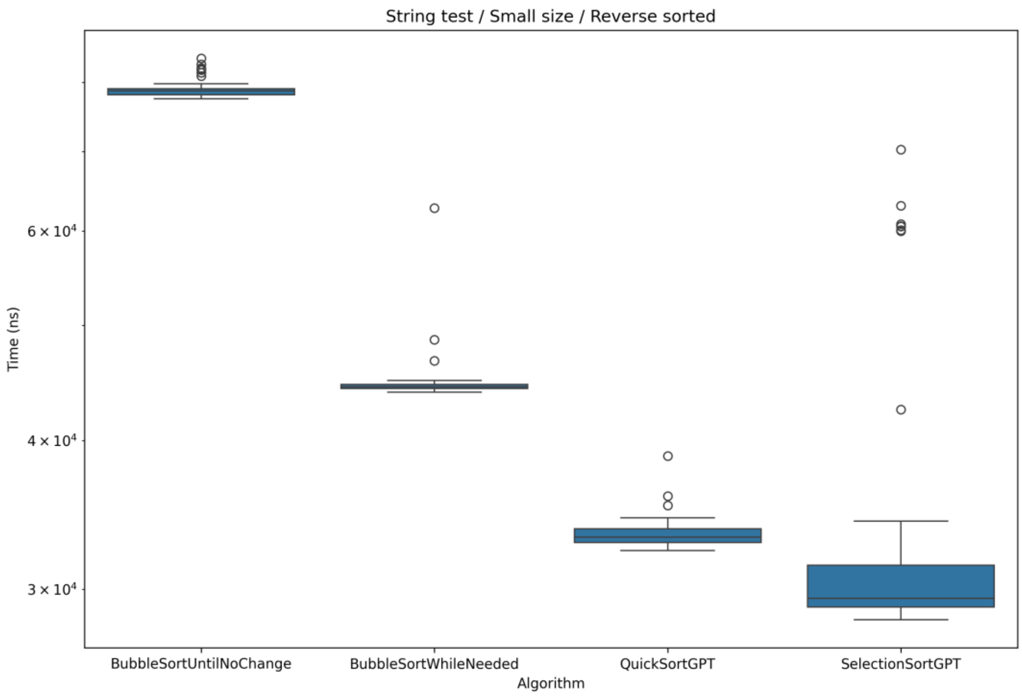

Integer test / Medium size / Random ordered
Time (ns)
10^6
10^5
BubbleSortUntilNoChange
BubbleSortWhileNeeded
QuickSortGPT
SelectionSortGPT
Algorithm

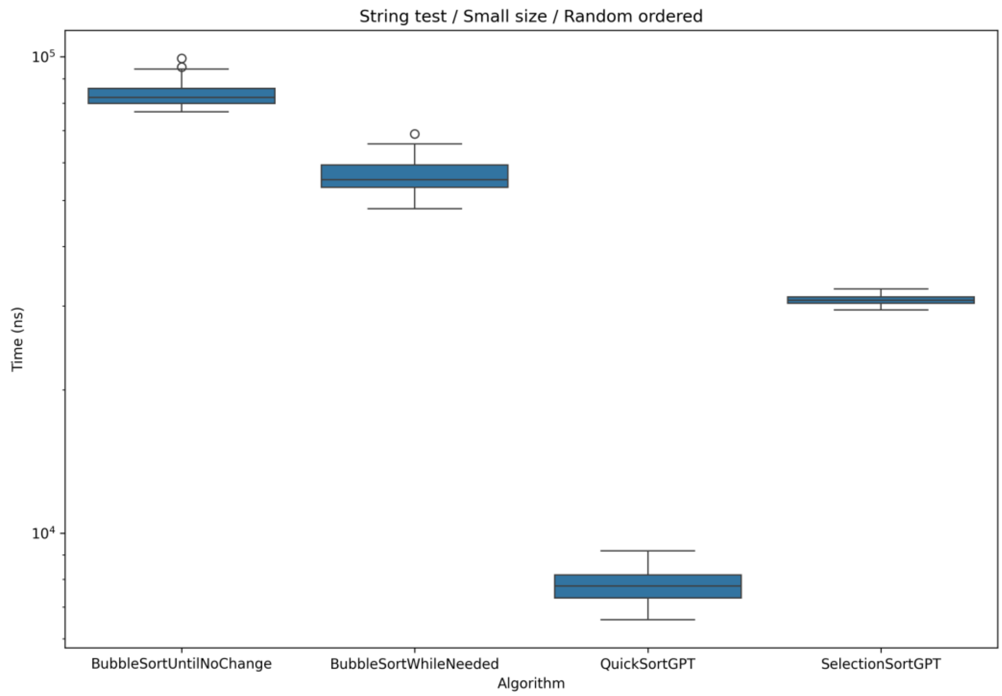

Integer test / Large size / Already sorted
Time (ns)
10^8
10^7
10^6
10^5
BubbleSortUntilNoChange
BubbleSortWhileNeeded
QuickSortGPT
SelectionSortGPT
Algorithm

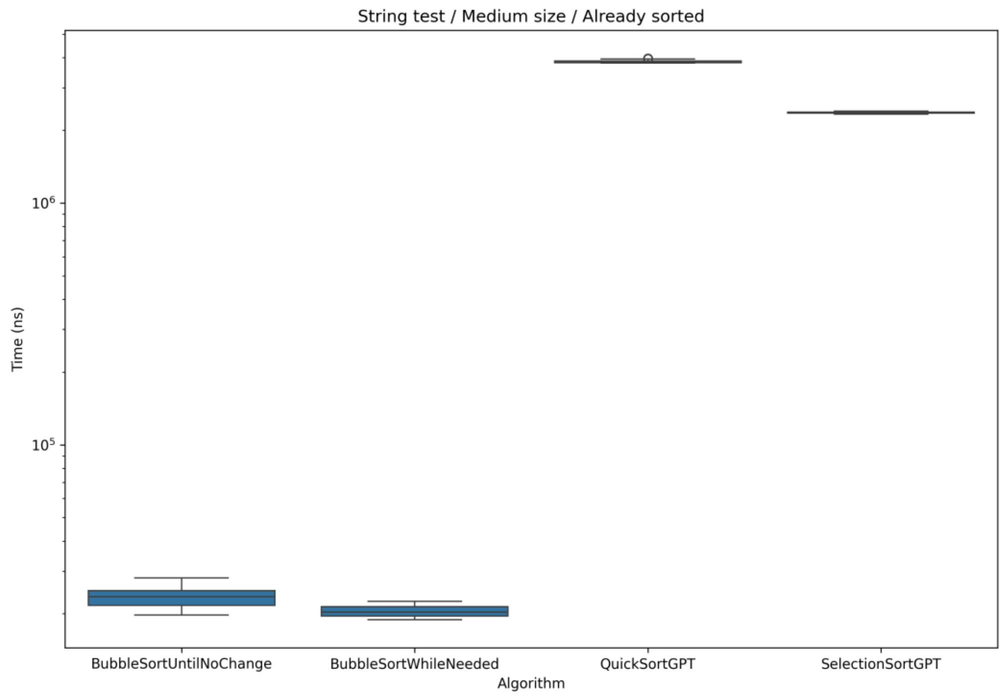

Integer test / Large size / Reverse sorted
5 × 10^8
4 × 10^8
3 × 10^8
Time (ns)
BubbleSortUntilNoChange
BubbleSortWhileNeeded
QuickSortGPT
SelectionSortGPT
Algorithm

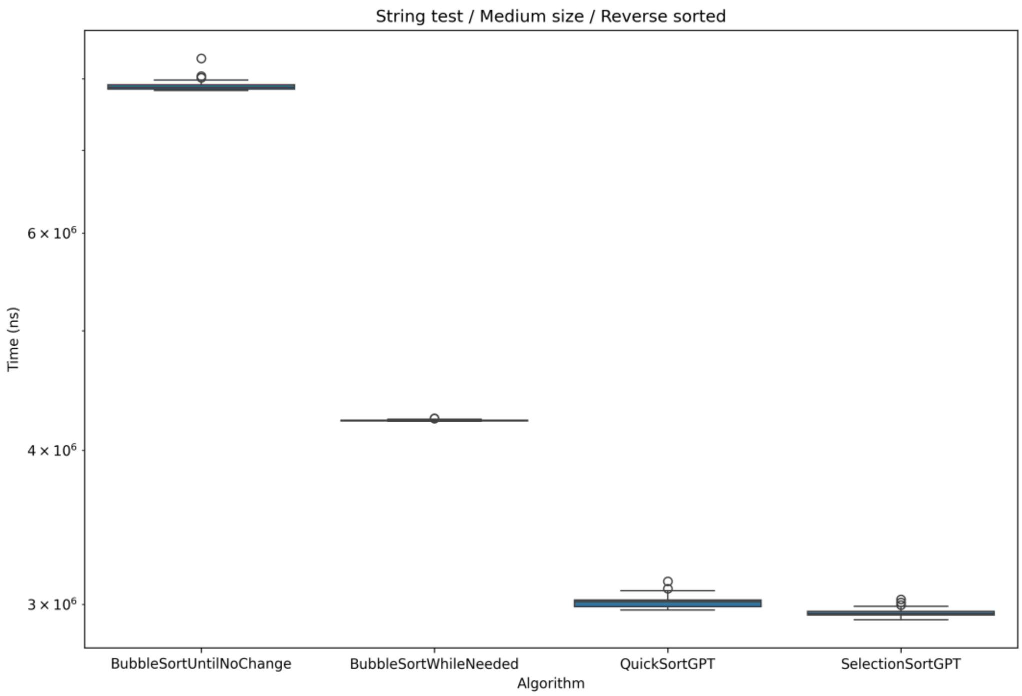

Integer test / Large size / Random ordered
10^9
10^8
10^7
Time (ns)
BubbleSortUntilNoChange
BubbleSortWhileNeeded
QuickSortGPT
SelectionSortGPT
Algorithm

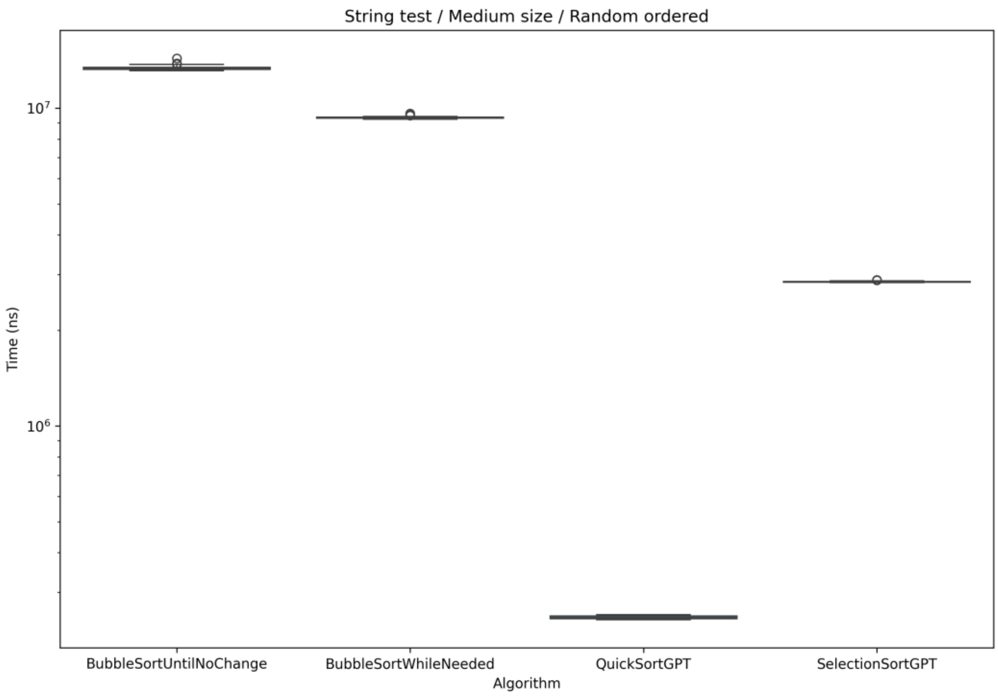

String test / Small size / Already sorted
Time (ns)
10^5
10^4
BubbleSortUntilNoChange
BubbleSortWhileNeeded
QuickSortGPT
SelectionSortGPT
Algorithm

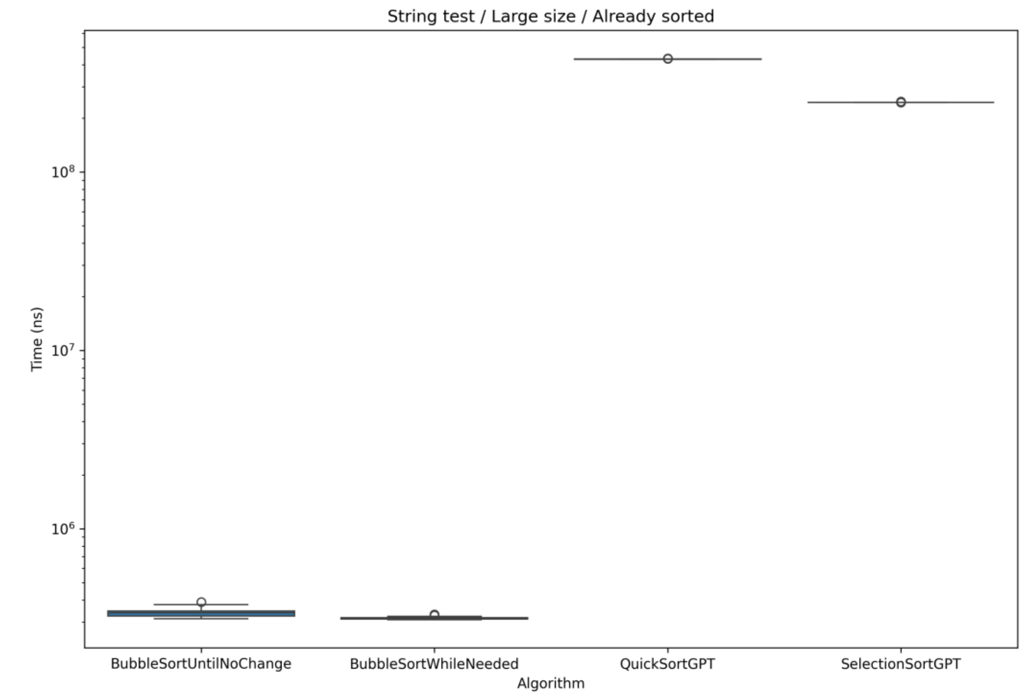

String test / Small size / Reverse sorted
Time (ns)
6×10^4
4×10^4
3×10^4
BubbleSortUntilNoChange
BubbleSortWhileNeeded
QuickSortGPT
SelectionSortGPT
Algorithm

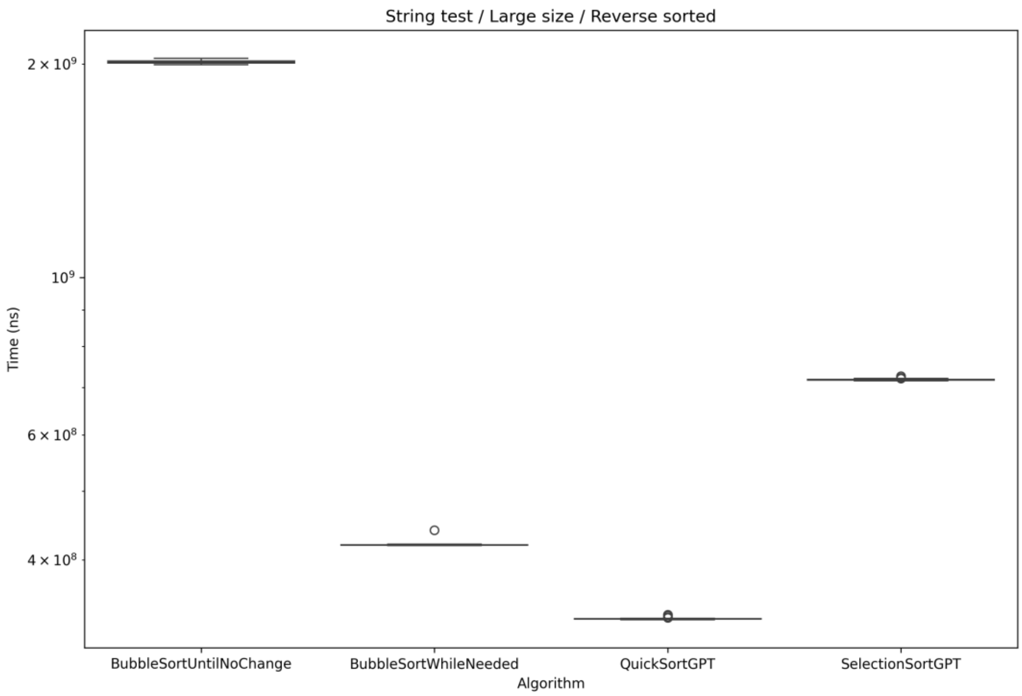

String test / Small size / Random ordered
10^5
10^4
Time (ns)
BubbleSortUntilNoChange
BubbleSortWhileNeeded
QuickSortGPT
SelectionSortGPT
Algorithm

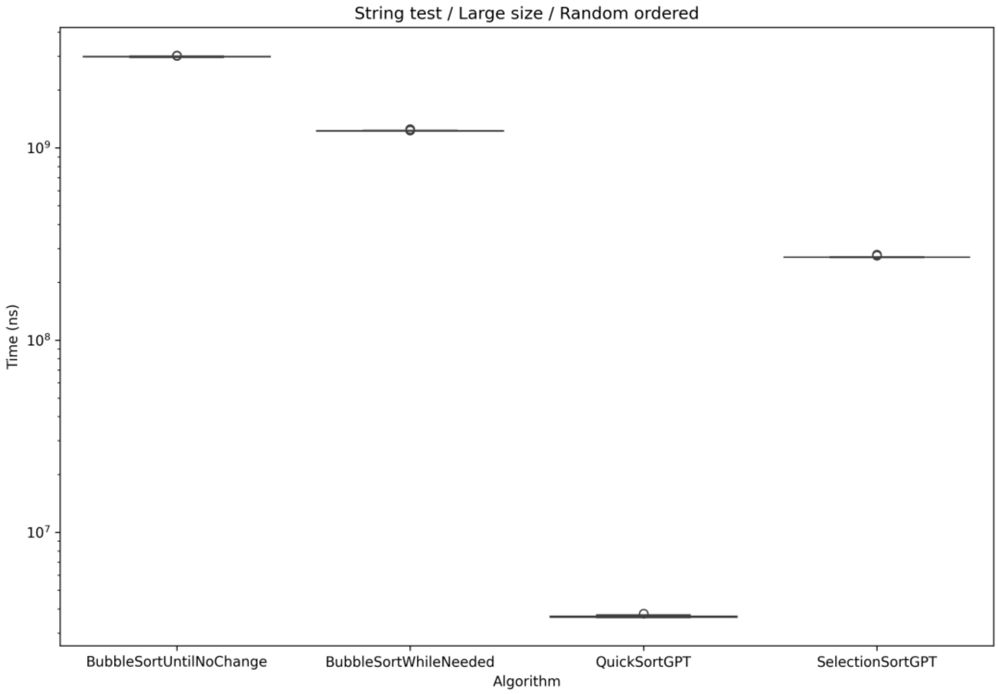

String test / Medium size / Already sorted
10^6
10^5
Time (ns)
BubbleSortUntilNoChange
BubbleSortWhileNeeded
QuickSortGPT
SelectionSortGPT
Algorithm


String test / Medium size / Reverse sorted
Time (ns)
6 × 10^6
4 × 10^6
3 × 10^6
BubbleSortUntilNoChange
BubbleSortWhileNeeded
QuickSortGPT
SelectionSortGPT
Algorithm


String test / Medium size / Random ordered
Time (ns)
10^7
10^6
BubbleSortUntilNoChange
BubbleSortWhileNeeded
QuickSortGPT
SelectionSortGPT
Algorithm


String test / Large size / Already sorted
Time (ns)
10^8
10^7
10^6
BubbleSortUntilNoChange
BubbleSortWhileNeeded
QuickSortGPT
SelectionSortGPT
Algorithm


String test / Large size / Reverse sorted
Time (ns)
2×10^9
10^9
6×10^8
4×10^8
BubbleSortUntilNoChange
BubbleSortWhileNeeded
QuickSortGPT
SelectionSortGPT
Algorithm



3.2 Descriptive Statistics

Below are reported the statistics for each permutation.

	BubbleSortUntilNoChange	BubbleSortWhileNeeded	QuickSortGPT	SelectionSortGPT
Integer Best case Small	Minimum: 3706 First quartile: 3848 Median: 3992 Third quartile: 4261 Maximum: 5610	Minimum: 3402 First quartile: 3495 Median: 3628 Third quartile: 3829 Maximum: 6846	Minimum: 214626 First quartile: 225859 Median: 263756 Third quartile: 275306 Maximum: 489096	Minimum: 6348 First quartile: 12888 Median: 35869 Third quartile: 37739 Maximum: 161116
Integer Best case Medium	Minimum: 3338 First quartile: 3968 Median: 4067 Third quartile: 4103 Maximum: 5854	Minimum: 2573 First quartile: 2981 Median: 3146 Third quartile: 3161 Maximum: 3349	Minimum: 3161830 First quartile: 3226823 Median: 3290154 Third quartile: 3328139 Maximum: 3510895	Minimum: 3119080 First quartile: 3138862 Median: 3146323 Third quartile: 3154399 Maximum: 3179118
Integer Best case Large	Minimum: 32916 First quartile: 39555 Median: 40063 Third quartile: 40438 Maximum: 111983	Minimum: 24614 First quartile: 25192 Median: 30096 Third quartile: 30382 Maximum: 100638	Minimum: 320964236 First quartile: 328384234 Median: 330161760 Third quartile: 331295721 Maximum: 333778866	Minimum: 319449939 First quartile: 319547036 Median: 319595375 Third quartile: 319681854 Maximum: 321996684
Integer Average case Small	Minimum: 40750 First quartile: 42321 Median: 43577 Third quartile: 45658 Maximum: 56443	Minimum: 78206 First quartile: 79965 Median: 81707 Third quartile: 105033 Maximum: 130469	Minimum: 11204 First quartile: 11608 Median: 12031 Third quartile: 12397 Maximum: 13475	Minimum: 40394 First quartile: 41391 Median: 45848 Third quartile: 68070 Maximum: 84115
Integer Average case Medium	Minimum: 7216353 First quartile: 7248829 Median: 7274641 Third quartile: 7307326 Maximum: 7454786	Minimum: 4977575 First quartile: 5012925 Median: 5027040 Third quartile: 5037130 Maximum: 5156499	Minimum: 113776 First quartile: 117234 Median: 119774 Third quartile: 121864 Maximum: 136070	Minimum: 3148598 First quartile: 3173368 Median: 3184306 Third quartile: 3192070 Maximum: 3202753
Integer Average	Minimum: 945156201 First quartile: 949892659	Minimum: 549391511 First quartile: 550036569	Minimum: 1805513 First quartile: 1833020	Minimum: 320345765 First quartile: 320428365

case Large	Median: 952158101 Third quartile: 952812825 Maximum: 965151741	Median: 550422494 Third quartile: 550705820 Maximum: 555707871	Median: 1843384 Third quartile: 1851597 Maximum: 1870782	Median: 320492252 Third quartile: 320540993 Maximum: 321361416
Integer Worse case Small	Minimum: 51180 First quartile: 120329 Median: 125789 Third quartile: 508259 Maximum: 645547	Minimum: 32405 First quartile: 80667 Median: 84328 Third quartile: 306569 Maximum: 490241	Minimum: 23645 First quartile: 24495 Median: 24872 Third quartile: 25365 Maximum: 38613	Minimum: 35599 First quartile: 35946 Median: 36690 Third quartile: 162469 Maximum: 195820
Integer Worse case Medium	Minimum: 4744419 First quartile: 4771992 Median: 4789770 Third quartile: 4807446 Maximum: 5035942	Minimum: 2933181 First quartile: 2980955 Median: 2984164 Third quartile: 2986981 Maximum: 3032064	Minimum: 2208503 First quartile: 2268868 Median: 2287465 Third quartile: 2336306 Maximum: 2479269	Minimum: 3126921 First quartile: 3144908 Median: 3151546 Third quartile: 3164305 Maximum: 3431527
Integer Worse case Large	Minimum: 475056238 First quartile: 476561603 Median: 477334915 Third quartile: 478230921 Maximum: 485408135	Minimum: 296931038 First quartile: 297336011 Median: 297535559 Third quartile: 297729802 Maximum: 299102934	Minimum: 226688798 First quartile: 227601658 Median: 239457925 Third quartile: 240254318 Maximum: 243336325	Minimum: 319622658 First quartile: 319715665 Median: 319771466 Third quartile: 319793914 Maximum: 330395820
String Best case Small	Minimum: 4128 First quartile: 4255 Median: 4317 Third quartile: 4369 Maximum: 5570	Minimum: 3673 First quartile: 3797 Median: 3837 Third quartile: 3895 Maximum: 4589	Minimum: 41095 First quartile: 45411 Median: 219521 Third quartile: 222180 Maximum: 234847	Minimum: 162017 First quartile: 164207 Median: 165533 Third quartile: 167476 Maximum: 173080
String Best case Medium	Minimum: 19778 First quartile: 21665 Median: 23756 Third quartile: 24866 Maximum: 28109	Minimum: 18903 First quartile: 19567 Median: 20379 Third quartile: 21354 Maximum: 22514	Minimum: 3794231 First quartile: 3812966 Median: 3827709 Third quartile: 3865023 Maximum: 3970244	Minimum: 2336162 First quartile: 2354546 Median: 2365754 Third quartile: 2379697 Maximum: 2402021
String Best case Large	Minimum: 313539 First quartile: 323815 Median: 338631 Third quartile: 346211 Maximum: 388585	Minimum: 309491 First quartile: 312166 Median: 314639 Third quartile: 317782 Maximum: 331508	Minimum: 427456747 First quartile: 428280992 Median: 428775108 Third quartile: 429358620 Maximum: 433590994	Minimum: 245224519 First quartile: 245292563 Median: 245362692 Third quartile: 245431628 Maximum: 248413062
String Average case Small	Minimum: 76676 First quartile: 79747 Median: 82424 Third quartile: 85691 Maximum: 99265	Minimum: 48008 First quartile: 53225 Median: 55442 Third quartile: 59327 Maximum: 68924	Minimum: 6583 First quartile: 7300 Median: 7795 Third quartile: 8136 Maximum: 9184	Minimum: 29410 First quartile: 30366 Median: 30824 Third quartile: 31149 Maximum: 32590
String Average case Medium	Minimum: 13115330 First quartile: 13268145 Median: 13345388 Third quartile: 13431233 Maximum: 14357974	Minimum: 9246458 First quartile: 9302672 Median: 9333492 Third quartile: 9371653 Maximum: 9634803	Minimum: 245845 First quartile: 247559 Median: 249893 Third quartile: 251986 Maximum: 255237	Minimum: 2827600 First quartile: 2838592 Median: 2843645 Third quartile: 2849873 Maximum: 2885195
String Average case Large	Minimum: 2956010469 First quartile: 2973015185 Median: 2980427441 Third quartile: 2986446741 Maximum: 3026133031	Minimum: 1222328567 First quartile: 1224690506 Median: 1226000519 Third quartile: 1227157639 Maximum: 1252971557	Minimum: 3600710 First quartile: 3618260 Median: 3640104 Third quartile: 3660832 Maximum: 3773153	Minimum: 268789306 First quartile: 269424929 Median: 269693126 Third quartile: 269959132 Maximum: 278637050
String Worse case Small	Minimum: 77494 First quartile: 78106 Median: 78752 Third quartile: 79015 Maximum: 83826	Minimum: 43929 First quartile: 44235 Median: 44403 Third quartile: 44576 Maximum: 62758	Minimum: 32339 First quartile: 32828 Median: 33191 Third quartile: 33717 Maximum: 38831	Minimum: 28286 First quartile: 28995 Median: 29561 Third quartile: 31277 Maximum: 70238
String Worse case Medium	Minimum: 7828365 First quartile: 7849651 Median: 7877101 Third quartile: 7902842 Maximum: 8311323	Minimum: 4221896 First quartile: 4224543 Median: 4225490 Third quartile: 4230753 Maximum: 4245713	Minimum: 2967740 First quartile: 2986436 Median: 3016197 Third quartile: 3023169 Maximum: 3132358	Minimum: 2915356 First quartile: 2941025 Median: 2949147 Third quartile: 2961433 Maximum: 3027745
String Worse case Large	Minimum: 1995038205 First quartile: 2006161706 Median: 2014484412 Third quartile: 2019364755 Maximum: 2036314834	Minimum: 419072375 First quartile: 419359576 Median: 419556266 Third quartile: 419818804 Maximum: 440260711	Minimum: 329300749 First quartile: 329839117 Median: 330129917 Third quartile: 330300313 Maximum: 334829769	Minimum: 714739939 First quartile: 716416187 Median: 717251293 Third quartile: 717941155 Maximum: 725715441

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 most efficient algorithms considering execution time as only criterion for performance, with consistently lower execution times across the majority of tested configurations.

Despite that, this study shows that QuickSortGPT is not the best for all the input arrays, as it performs badly on already sorted arrays due to its time complexity in the best-case scenario.

In summary, this study indicates that QuickSortGPT is the most effective algorithm for general applications where execution time is the only performance criterion.

Appendix

See our GitHub repository to check for extra materials and raw data.