The Bubble Algorithm Library: A Performance Evaluation

Group 5

June 2, 2022

Erick Garro Elizondo
Cindy Guerrero Toro

Contents

1 Introduction

Algorithm (in honour to mathematician *Muhammad ibn Mūsā al-Khwārizmī*)is the term used to refer to a set of ordered and finite well-defined instructions that enclose a process to do or achieve something specific. Algorithms, for example, can be seen as simple as a recipe for cooking a delicious meal, as complex as a flight collision avoidance system, or as beautiful as the creation of life. So when you think about algorithms, you realize that are simply part of everyday life. When we face the pragmatic side of an algorithm, we notice that is process or a technique for solving a given problem based on completing a set of tasks [9].

Programming languages serve as a natural-like language interface for programmers to encode algorithms that allow a general-purpose computer to execute the tasks designed to solve the proposed problems [8, 7]. In this way, these programs enable humans to simplify things while expanding their capabilities.

In computer science, the main question is how much time algorithms take to solve a problem. Generally, computer scientists determine the answer to this question based on the set of components that the algorithm acts on. For example, the processing time of an algorithm that finds the largest set in a database is relative to the size of the list. However, for the "slow" formula for sorting the numbers in the database from the smallest to largest, the execution time would be relative to the shape of the size of the list. This variation depends on whether the list is pre-sorted from the smallest to largest element, which is the worst case if the list is sorted from largest to smallest. The most representative scenario in real life is when the list is randomly sorted. In such a case, the average case scenario is considered. However, there is a general tendency among computer scientists to consider the complexity of an algorithm as the worst-case scenario, based on the concept of Murphy's Law, which states, "Anything that can go wrong will go wrong."

There are several ways to evaluate the execution of an algorithm, and the following categories are relevant to this research: Completeness, Optimality, Experience Quality, and Infinite Quality. An algorithm that is *complete* usually promises to obtain the solution, if there is one. An *optimal* algorithm, is not only capable to find a solution, but to perform within the lowest cost (i.e. smallest Big O notation) [1, 3].

The minute *quality* of the algorithm quantifies the time it takes to finish the formula as an absolute number/length of the sequence. The *infinite quality* of the algorithm quantifies the amount of space it takes for the algorithm to successfully complete the formula.

Algorithmic efficiency refers to the features of an algorithm that are related to the number of computational resources required by the approach. It is possible to optimize a computer program such that it runs faster while using less memory, power, and other resources. Algorithmic efficiency is the same as technical productivity in a recurrent or continuous operation [5].

The biggest drawback of the algorithm formula is the hardware that has to hold all those nodes, compared to the level-first search algorithm, which has a minimal footprint. This leads to an exponentially infinite quality that can exhaust the free hardware on the computer within a few hours. Thus, the formula is impractical for significantly large problems.

Bubble, Inc. has requested us to analyze the execution time performance of their three in-place Bubble Sort algorithm candidates to choose which unique implementation we recommend to include in the library they are creating for Java developers.

This report describes our experiment's design decisions, the process we followed, the results we obtained, and our final recommendation.

2 Hypotheses

- Data sorting: When consider the input data, the complexity of the sorting of the data has an influence in the performance (running time) in any of the sorting algorithms being the pre-sorted data in an *ascending*- the fastest, *random*- the average, and *descending*-data the slowest.

- Array size: When considering the length of the arrays, the running time will be directly proportional to the size of the input data. Being a large input-data set (*10,000 elements*) the slowest, compared to the smallest input

data set (*100 elements*). A middle size input-data with *1,000 elements* is also to be considered.

- Data type: There are two major considerations regarding data-type: Size and Nature. When considering the nature of data, the data with highest complexity (as a reference type), *string*, will take the slowest running time, whereas data-type of a number nature will run faster. Among these *floating-point*-number (4 Bytes) will have the slowest running time, followed by *integer*-number (4 Bytes) and *short*-number (2 Bytes) as the fastest number. Thus, the performance (time) will proceed as: string > float > integer > short.

- Algorithm (complexity): When considering the coding and algorithm complexity, the *Bubble sort pass per item* algorithm falls in the $O(n^2)$ will take the longest time, whereas a more refined *Bubble sort until no change* algorithm will run faster. The *Bubble sort while needed* algorithm will be the fastest, as it implements only if it is necessary to run a bubble sorting.

3 Materials & Methods

All experiments were performed in accordance with instructions from the course in Experimentation and Evaluation 2022 as part of the curriculum program for bachelor's in informatics of the Università della Svizzera italiana, Lugano, Switzerland.

3.1 Variables

3.1.1 Independent variables

Three major bubble sorting algorithms, provided by Prof. Dr. Gabriele Bavota, were evaluated and compared one another in this study (*See* Appendix Bubble Sort Algorithms):

1. Bubble sort pass per item
2. Bubble sort until no change
3. Bubble sort while needed

Variable	Levels
Algorithms	BubbleSortPassPerItem BubbleSortUntilNoChange BubbleSortWhileNeeded
Data Type	Floating-point number Integer Short String
Array Size	100 elements 1,000 elements 10,000 elements
Data Sorting	Ascending Descending Random

Table 1: Independent variables. Four major independent variables where including in the study (left panel). Notice that input data of Floating-point number contains four decimals (i.e. 123.0000); integer contains numbers from -2,147,483,648 to 2,147,483,647; short contains numbers from -32,768 to 32,767). Strings contains only alphabetical words with length of five characters (Data Type, right panel).

3.1.2 Dependent variables

Variable	Measurement scale
Running time	nanoseconds (ns)

Table 2: Dependent variables. The performance of an algorithm is measured by the computation running time in ns (10^{-9} s) .

3.1.3 Control variables

The control variables can be found in table 3. Different factors within the machinery can be considered during the implementation of the sorting algorithms and measurement of the processing time (ns).

Each algorithm was run equally with a number of 1000 iterations times prior to data acquisition in order to prepare the system. All algorithms were executed using a Mac Mini Apple M1 chip (M1, 2020) with 8-Core GPU Family Apple 7 and 16-core Neural Engine, LPD DDR4 and 16 GB memory macOS Monterey v. 12.3 (21E230).

Any active software was reduced to contain only the necessary programs for the basic computer functioning. Data acquisition and data collection were performed without additional software and by us-

ing Java Virtual Machine with implementation of the version 17.0.1 of the Java SE Platform as specified by JSR 392. No internet connection of any kind was supplied during data acquisition. Data was collected and exported for off-line statistical analysis. A full repository of the implementation and supplementary material can be found on GitHub.

Variable	Fixed value
Machine	Apple Mac Mini
CPU	Apple M1 (2020) 8-core CPU
Controller	iBoot-7459.101.2
Graphics	Apple M1 8-core GPU, GPU Family Apple 7 Built-in bus
Memory	16 GB LPDDR4
Operating System	macOS Monterrey v.12.3 (21E230)
Java version	17.0.1, 2021-10-19 LTS
Java™ SE Runtime Environment	Build 17.0.1+12-LTS-39
Java HotSpot™	64-Bit Server VM, mixed mode, sharing

Table 3: Control variables. The hardware and software properties of the machine needs to be considered when running the algorithms in different machines. Some machines have *per se* a better performance than others. A general description of the hardware and software used during experimentation is described above.

3.2 Design

Type of study: Experiment
Number of factors: Multi-Factor Design

We chose to analyze the Bubble Sort implementations with four different *data types* in distinct combinations of *array sizes*. This allowed us to measure our independent variable in three different configurations per stage, by reorganizing the *data ordering* each time.

The experiment was executed in sequence, one group after the other, as no multi-threading was implemented.

These groups could be considered dissimilar because of their technical nature of belonging to different types of objects. Nonetheless, randomized generation was our source of each group's population, and then, by configuration, we achieved homogenization by synchronizing the other independent variables. Figure 1 shows a simplified view of the groups configuration.

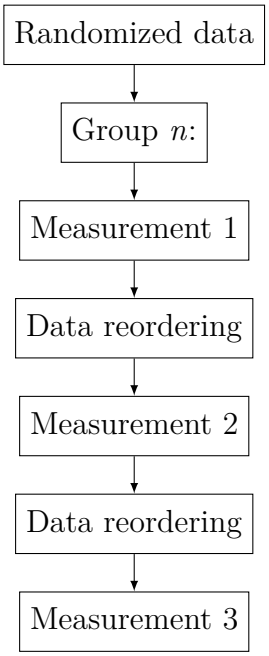

Figure 1: Simplified view of a multiple level experiment with multiple independent variables. Each group represent a combination of *data type* and *array size*. Every *measurement* represents the evaluation of the dependent variable with each of the algorithms. Each *data reordering* represents the order reorganization prior to each measurement.

3.3 Apparatus

The experimentation software was programmed and compiled on a MacBook Pro (15-inch, 2019) running macOS Catalina 10.15.7 (19H1715), with a 2.6 GHz 6-Core Intel Core i7 CPU and 6 GB 2400 MHz DDR4 of memory; wiht IntelliJ IDEA 2021.3.2 (Ultimate Edition) Build #IU-213.6777.52, built on January 28, 2022, and Java's JDK v.17 was used for the compilation.

All algorithms were executed using a Mac Mini Apple M1 chip (M1, 2020) with 8-Core GPU Family Apple 7, 16 GB LPD DDR4 memory, and macOS Monterrey v.12.3 (21E230).

3.4 Procedure

To run the experiment, we created an application that automates the initialization of our data and the execution of the measurements, results collection, and data processing.

The program allows the experiments to be fully customized by parameters-passing on invocation within the restriction of a fixed set of *data types*.

The user can set the following variables declaring flags for:

- the number of iterations to run each instance of source data for each Bubble Sorting algorithm to evaluate
- the seed to initialize the randomized data generator
- the length of the strings to assess, the array sizes to use
- the last n number of results from the executed iterations to be considered for the data evaluation

The usage of the application is as follows:

```
java -jar BubbleSortExperiments.jar
   [-i <iterations>] [-s <seed>] [-l
   <string length>] [-a <array sizes>]
   [-n <last number of results>] [-h]
   [-r]
```

If the user does not provide any of those values, the system will execute with these default values:

- iterations: 1,000
- seed: Java standard random seed
- string length: 5 characters
- last number of results: 30

The user can also call the program with the *-r* flag to get a basic overview of the statistics of every iteration, or if needed, with the *-h* flag to get the usage details.

When invoked, the program looks on the disk for all pre-filtered files available, and if they match the string length, it will load it into memory. If no matching file exists, it will process an external file containing 370,103 unique strings and generate the corresponding filtered version. The files is located in *data/all_strings.txt* inside the working directory. The original file is publicly available[1].

The program then initializes the arrays that will hold the data to sort for each array size and type and populate them with randomly generated data. In the case of the strings, it will randomly take the words from the pre-filtered data.

After all randomized data arrays are generated according to the current array size and data type, the program is ready to enter the next phase, where data ordering comes to play.

One ordering type at a time, the program will create clones of the corresponding random array and run them sequentially through the provided Bubble Sort algorithms. Because the Enums are also sorted by default, the ascending array is processed first, the descending second, and the random third.

When finished, it continues to the following data type. When all data types for that array size are done, it will iterate again with the next array size if there is more than one.

Before the sorter in evaluation processes each configuration, an arbitrary total of seven garbage collection invocations are performed, aiming at clearing up the memory heap of unused objects [6].

Then a timer in nanoseconds precision is started and stopped right after the sorter finishes. Execution time is then calculated and collected in a Result class instance, which is pushed into a Result Stack. The measurements are collected for further processing to extract statistical data in the next stage.

The entire process concludes when a CSV file is generated and persisted on the hard drive.

3.4.1 Project structure

The source code of the experimentation program is structured inside the main package *ch.usi.ee*, where the *Main.java* class resides.

[1] https://github.com/dwyl/english-words/blob/master/words_alpha.txt

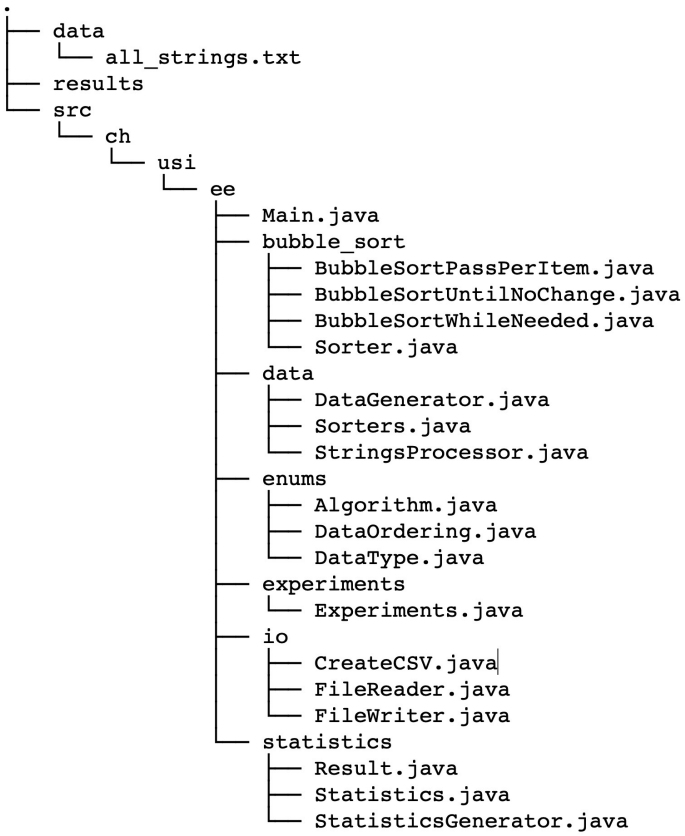

Figure 2: Package and classes structure of the project

The rest of the project was structured in six sub packages, considering future extensibility and maintainability. Here is their high level description:

- **bubble_sort**: contains the *sort* interface and the three implementations provided of Bubble Sort.
- **data**: contains the strings source file for the String processor, and the data generators and pre-sorter
- **enums**: contains Enum classes to describe the algorithms that can be tested, and the available data orderings and data types
- **experiments**: holds the class that executes the experiments
- **io**: manages all input and output data from and to the hard disk, as well as the creation of CSV files
- **statistics**: handles result holders and statistics generation

3.4.2 Source code, artifacts, and statistical files

When all the results are gathered, a custom-built statistics package is used to generate statistical data that is dumped in a CSV file that is timestamped and written in the folder *results/* inside the working directory.

The source code of the experimentation software is available on GitHub[2]. It contains also the project files for the IDE described in 3.2.

A pre-compiled JAR file can be downloaded from there[3] as well.

We made available on the repository all the files used for data analysis[4], including the original output CSV file and a set of ten runs to corroborate for results consistency of the application.

3.4.3 Results reproduction steps

To reproduce the results the following instructions must be followed:

a) Compile the source code according to the hardware specifications, IDE, and JDK described in 3.2. Alternatively, use the provided JAR file from 3.4.1.

b) Place the JAR file in the desired folder.

c) Create a subfolder named *data/* and copy into it the file *all_strings.txt*, also described in 3.4.1.

d) Invoke the JAR file with the following command on a system with the characteristics described in:

```
java -jar
   BubbleSortExperiments.jar -i
   1000 -s 17072021 -l 5 -a
   100,1000,10000 -n 30
```

e) After the experiment is finished, a timestamped CSV results file will be available in the subfolder named *results/*

[2] https://github.com/erickgarro/EE-Bubble-Sort-Experiments

[3] https://github.com/erickgarro/EE-Bubble-Sort-Experiments/blob/main/artifacts/BubbleSortExperiments.jar

[4] https://github.com/erickgarro/EE-Bubble-Sort-Experiments/tree/main/statistical_files

3.5 Input data array size

Each bubble sort algorithm requires input of an array of type Comparable, an interface that makes the comparison between objects more effortless [4]. As such, they can contain objects of other data types. For example, our case had String (with a predefined length of five with alphabetical characters), floating-point-, integer-, and short-numbers.

Three representative lengths of the array were chosen (100, 1000, and 10000).

In addition, the data within the array was presorted following an ascending, descending or random order (java.util.Random).

3.6 Processing time

The performance of each Bubble algorithm is determined by using Java System.nanoTime() . During data acquisition, to avoid external added time, the time was calculated by the difference between processing time preceding and after running each Bubble sorter algorithm. Thus, an actual total processing time per iteration (*See* Class Experiments line 201-206, *elapsedTime*) was considered for the data analysis.

Each bubble sorting algorithm was run ∼1,000 each time (total iterations) before collecting data. A total of thirty processing time acquisitions (last thirty per experimental set) were collected per experimental set. The collected data was summarized in a table (*See* Summary Results) and exported a single CSV file for offline analysis.

3.7 Data Entry

The Class Experiments contributed to the processing and collecting of the data with the performance (processing time, ns) of the multiple combination of each of the independent variables.

Each of the bubble sort algorithms took as input an array of elements belonging to a combination of one of each of the following categories: Data type, array size, and data sorting (*See* Fig. 3).

We constructed multiple combinations or configurations for the data input iteratively for each measurement as follows: first, we choose data type, then array size, and last data ordering. After each data configuration was complete, we ran it through each sorter in order, as described in 3.3.

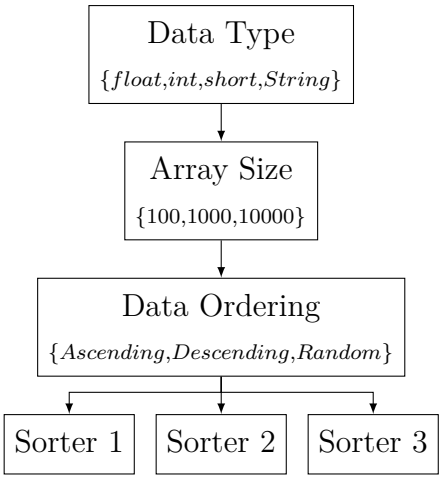

Figure 3: Input data. Each of the input data was the product of a combination of the main variables (*See* Independent variables). *Data type* consisted of floating-point-, integer- or, short-number or String of length five. *Array size* has 100, 1,000 or 10,000 elements. *Data sorting* arranges the data arrays in ascending, descending or random order. *Sorter* is a comparable option among three algorithms BubbleSortPassPerItem, BubbleSortUntilNoChange or BubbleSortWhileNeeded

3.8 Data analysis

The data was analysed by using functions within Java Statistical Packages and IBM® SPSS® Statistics version 27, 64-bit edition. The paired-sample t-test was used to compared data within the same experimental group input, whereas the Wilcoxon-Mann-Whitney test were used for comparison between different groups. The results are presented as mean ±SEM (standard error of the mean) with the level of significance set as $*p < 0.05$.

4 Results

4.1 Data Sorting and Array Size

Data sorting has a fundamental role when considering the size of the array that will be sorted in each algorithm. As shown in figure 4, the array data inputed into the algorithms performs better on data pre-sorted on an ascending manner (n=36 experiments, $3.85\text{x}10^7$ ns ± $1.61\text{x}10^8$ ns). This behaviour was observed for each algorithm, particularly when array data size was 1000 and 10000 elements (*See* Data Sorting vs Array Size). The second most efficient array-sorting-data input was the descending sorting (n=36 experiments, $1.086\text{x}10^8$ ns ± $1.92\text{x}10^8$ ns), followed by random sorting (n=36

experiments, 1.39x10^8 ns ± 2.78x10^8 ns).

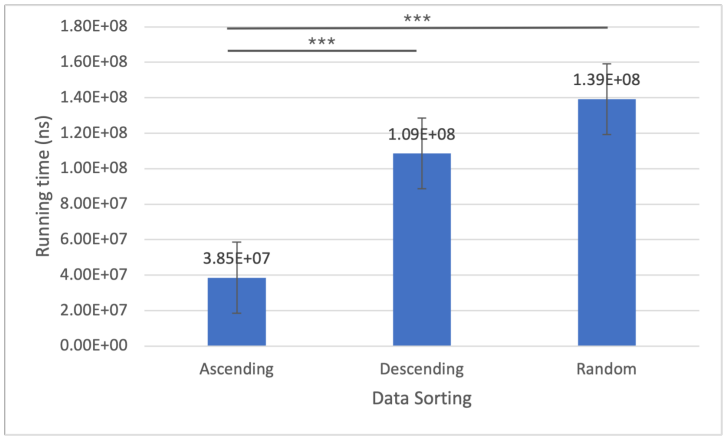

Figure 4: Data sorting. The processing time (ns) of three main array input sorted by ascending, descending or random (n=36 total experiments. 12 experiments per case). All data including array size, data type and algorithm type were added in the analysis. Notice that the ascending sorting has a better performance compared to descending or random sorting arrays. Mean±SEM. ***, P<0.001 by Mann-Whitney test.

When taking into consideration each bubble sort algorithm and the pre-ordering of data, we observe once more the advantage that ascending pre-sorted array presented compared to descending or random array inputs. The processing time of the bubble sort until no change (n=12 experiments, 1.65x10^4± 3.13x10^4 ns) and bubble sort while needed (n=12 experiments, 1.63x10^4± 3.39x10^4 ns) algorithms, which were relatively similar each other, were significantly different compared to its pair-set algorithm, bubble sort pass per item, or any other combination for the descending or random array. In the case of descending- vs random-sorted arrays, as shown in figure 5, the data presented a similar behaviour. Despite the relatively rapid processing time in the descending-bubble sort while needed algorithm (Fig. 5 *center, WN*, descending) compared with its pair algorithms, the values were not significantly different one another.

4.2 Data Type

The leading six worst performance, as running time (ns), experiments were strings. In fact, the worst performance observed among of all data series was an array random-sorted of strings running the *bubble sort until no change* algorithm, followed by string random *bubble sort pass per item* algorithm. We observed that String data type (n=27 experiments, 3.66x10^5±7.47x10^5 ns) has the worst processing time

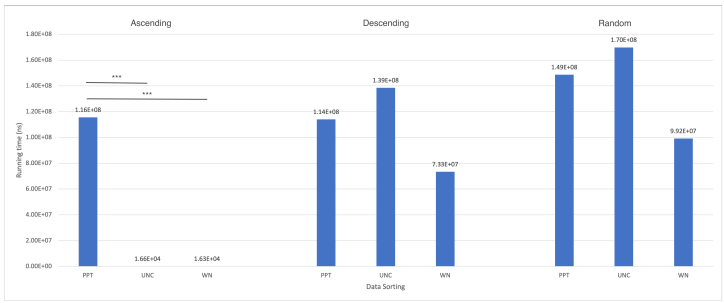

Figure 5: Data sorting per algorithm. The processing time (ns) resulting of the input-array data pre-sorted as ascending, descending or random where compared with each of the three bubble sort algorithms (BS) 1) BS per item (PPT) 2) BS until no change (UNC) 3) BS while needed (WN). The processing time (ns) of three main array input sorted by ascending, descending or random (n=36 total experiments). Four experiments were grouped in each set, each corresponding to data type:floating-point-, integer-, short-number, or String. The ascending sorting has a better performance compared to descending or random sorting arrays. Mean±SEM. *, P<0.05; ***, P<0.001 by Mann-Whitney test.

comparing to its pairs floating-point number (n=27 experiments, 2.49x10^4±5.68x10^4 ns), integer number ((n=27 experiments, 2.47x10^4±6.88x10^4 ns)), short number (n=27 experiments, 5.83x10^4± 1.40x10^5 ns). As shown in figure 6, string was significantly different from any of its pair sample data types, discriminating array size or type of algorithm.

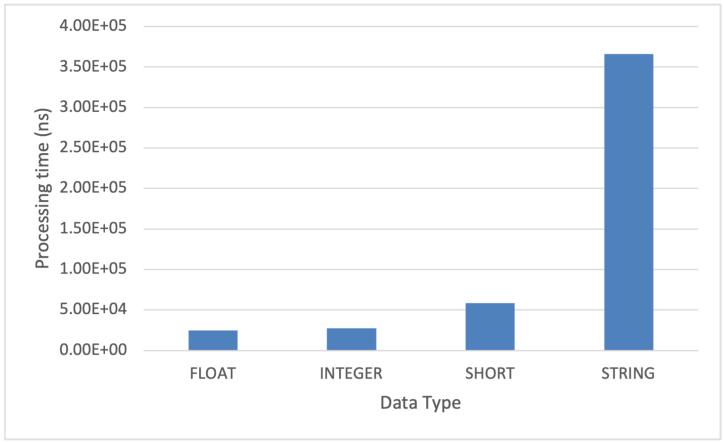

Figure 6: Data Type. The processing time (ns) resulting of the data type floating-point-, integer-, short-number, and string (n=27 total experiments). Four experiments were grouped in each set, each corresponding to data type:floating-point-, integer-, short-number, or String. The ascending sorting has a better performance compared to descending or random sorting arrays. The string has the worst performance, followed by short.

Once we performed a more refined sectioning of the data (*See* Fig. 7), we clearly observed that the main algorithms contributing to a bad processing time are *Bubble Sort Pass Per Item* (n=9 experiments,
5.89x10^5±1.40x10^5 ns) and *Bubble Sort Until No Change* ((n=9 experiments, 4.13x10^5±8.26x10^5 ns)

algorithms, whereas the *Bubble Sort while needed* algorithm (n=9 experiments, $4.13x10^5 \pm 8.26x10^5$ ns) was significantly faster than its pairs.

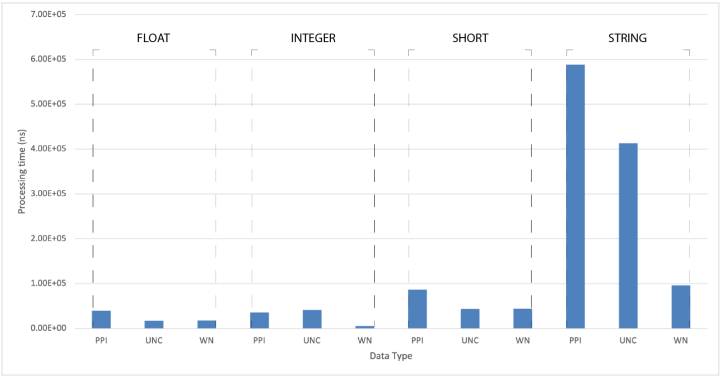

Figure 7: Data Type per algorithm. The processing time (ns) resulting of the data type four data types floating-point-, integer-, short-number, and string where compared with each of the three bubble sort algorithms (BS) 1) BS per item (PPT) 2) BS until no change (UNC) 3) BS while needed (WN). The processing time (ns) of three main array (n=9 per experimental group).

5 Discussion

5.1 Data Sorting

As initially hypothesized, our data (*See* Data Sorting and Array Size) proves that the data sorted in an ascending manner will have a better performance regardless of the type of algorithm since this is the best case-scenario were the bubble sort algorithm, theoretically, does not have to sort any single element.

One of the aspects we clearly observed is the inefficacy of the the Bubble Sort Pass Per Item algorithm. Despite of having a pre-sorted array of elements, we observed that the processing time was the worst among the three algorithms regardless of data type or size. Another interesting aspect that needs to be highlighted when determine the array sorting is the random sorting.

Opposite to what we initially considered, where the worst complexity would be given by a descending order array, the random sorting presented the slowest running time. A possible attribution to this behaviour is that after several iterations of the $a < b$ comparison will continue to be false, leading to a branch prediction heuristic that will assume the same outcome for the upcoming comparisons[2].

5.2 Array size

In regards to array sizes, we confirmed our hypothesis where the running time would be proportional to the length of the arrays. Larger arrays take longer to be processed. Nonetheless, is notable the repetition of the patter, where in the cases of size 1000 and 10000, Bubble Sort Pass Per Item has the worst performance, and the Bubble Sort Until No Change and Bubble Sort While Needed perform similarly.

5.3 Data type

Recapitulating our hypothesis related to data types, we originally said that in terms of execution time,

string > float > integer > short

but we found out that they perform as follows:

string > short > float > integer

String: takes the longest processing time since it is a complex data type (a reference kind, not a number as all the others). It takes longer to fetch information from memory and process it.

Short: short also takes long processing times as the computer architecture (64-bit) is designed to pass data in chunks of 4 Bytes. Then, the fact that short has 2 bytes, does not make it more efficient. On the contrary, it makes the machine (possibly) to save the information in chunks of 4 bytes, take that and extract the (first or second) 2 bytes to pass it. This process is tedious and occupies memory.

Floating-point: either floating or integer contain 4 bytes, however floating point has extra information. It needs to process information (depending on the machinery) by using i.e. one's complement to store data, re transform it as a floating, hence, this processing takes time.

Integer: this is the fastest since the machinery is designed to pass and get numbers.

6 Conclusions

The main focus of this report, presented three algorithm based solutions *Bubble Sort Pass Per Item*

(PPT), *Bubble sort until no change* (UNC) and *Bubble Sort While Needed* (WN, *See* Bubble Sort Algorithms) for complex array data input containing a variation in data types, array sizes and array sorting. We have concluded that the best algorithm that the *Bubble Inc.* company can use to implement its library is the *Bubble Sort While Needed*.

Despite the poor execution time presented when including data of either *String* or *short* number type, it shows the least poor performance among the different bubble sort algorithms between all data set.

First of all, when we observed the performance of pre-sorted data, the test cases data clearly indicates that the ascending data arrays irrespective of the data type or the pattern length presented a reduced processing data cost.

Second, the array size has a detrimental role in each of the bubble sort algorithms. The input data with a higher array size has a the worst performance. Particularly, this behaviour is clearly observed in the PPI algorithm.

During this study, we acknowledge that we compared small data sets with less than 10,000 elements. However, we noticed a clear pattern when increasing array data size, particularly for PPI. We infer that really complex data input with exponentially increased values would have a low performance, however further experiments with bigger that sets would need to be implemented to conclude this.

Furthermore, when applying complex data types, bigger size arrays perform the least. Therefore, we do not recommend to include this PPI algorithm in the library.

Finally, the data type with integer number has the highest-throughput computational experimental screening. Possibly due to hardware architecture and JVM that favours this performance by facilitating resource allocation and memory processes. We suspected that the memory size demands increase with the size of the input array.

Evidently the implementation of multi-threading and memory allocation would increase the performance of each algorithm. Further studies would be suggested on this regards, possibly by implementing the given bubble sorting algorithms in a language such as C-programming.

References

[1] R. Singh Chaha, "A* Algorithm, A* Admissibility," The Polymath Blog, Aug. 10, 2020. https://neelshelar.com/a-algorithm-a-admissibility/. Accessed: Mar 24, 2022.

[2] G. Cox, "Branch Prediction in Java — Baeldung," Dec. 31, 2019. https://www.baeldung.com/java-branch-prediction (accessed Mar. 29, 2022).

[3] D. J. Songhurst, Ed., "Chapter 5 - Agents," in Charging Communication Networks, Amsterdam: Elsevier Science B.V., 1999, pp. 109–147. Elsevier Science. https://doi.org/10.1016/B978-044450275-9/50006-5

[4] Oracle, "Comparable (Java Platform SE 8)," Java™ Platform, Standard Edition 8 API Specification. https://docs.oracle.com/javase/8/docs/api/java/lang/Comparable.html (accessed Mar. 22, 2022).

[5] Singh R., Easy Data Structure Using C Language: For BCA, B-tech and Others. Rana Books India, India, 2021, p. 11.

[6] Oracle, "Java Garbage Collection Basics," Java Garbage Collection Basics. https://www.oracle.com/webfolder/technetwork/tutorials/obe/java/gc01/index.html (accessed Mar. 22, 2022).

[7] D. Price, E. Rilofff, J. Zachary, and B. Harvey, ("NaturalJava: a natural language interface for programming in Java," in Proceedings of the 5th international conference on Intelligent user interfaces, New York, NY, USA, Jan. 2000, pp. 207–211. https://doi.org/10.1145/325737.325845 Accessed: Mar 23, 2022.

[8] W. Ballard and A. W. Biermann, "Programming in natural language: 'NLC' as a prototype," in Proceedings of the 1979 annual conference, New York, NY, USA, Jan. 1979, pp. 228–237. https://doi.org/10.1145/800177.810072 Accessed: Mar 22, 2022.

[9] Y. N. Moschovakis (2001), "What Is an Algorithm?," in Mathematics Unlimited — 2001 and Beyond, B. Engquist and W. Schmid, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 919–936. https://doi.org/10.1007/978-3-642-56478-9_46 Accessed: Mar 24, 2022.

7 Appendix

7.1 Bubble Sort Algorithms

Interface Sorter

```
3  public interface Sorter<T extends Comparable<T>> {
4     public abstract void sort(final T[] items);
5  }
```

Class Bubble sort pass per item

```
3  public final class BubbleSortPassPerItem<T extends Comparable<T>> implements
      Sorter<T> {
4
5     public void sort(final T[] items) {
6
7        for (int pass = 0; pass < items.length; pass++) {
8           for (int i = 0; i <items.length-1; i++) {
9              if (items[i].compareTo(items[i + 1]) > 0) {
10                final T item = items[i];
11                items[i] = items[i + 1];
12                items[i + 1] = item;
13             }
14          }
15       }
16    }
17 }
```

Class Bubble sort until no change

```
3  public final class BubbleSortUntilNoChange<T extends Comparable<T>> implements
      Sorter<T> {
4
5     public void sort(final T[] items) {
6        boolean changed;
7        do {
8           changed = false;
9           for (int i = 0; i < items.length-1; i++) {
10             if (items[i].compareTo(items[i + 1]) > 0) {
11                final T item = items[i];
12                items[i] = items[i+1];
13                items[i + 1] = item;
14                changed = true;
15             }
16          }
17       } while (changed);
18    }
19 }
```

Class Bubble sort while needed

```
3  public final class BubbleSortWhileNeeded<T extends Comparable<T>> implements
       Sorter<T> {
4
5      public void sort(final T[] items) {
6          int n = items.length;
7
8          do {
9              int maxIndex = 0;
10             for (int i = 1; i < n; i++) {
11                 if (items[i - 1].compareTo(items[i]) > 0) {
12                     final T item = items[i - 1];
13                     items[i - 1] = items[i];
14                     items[i] = item;
15                     maxIndex = i;
16                 }
17             }
18             n = maxIndex;
19         } while (n > 0);
20     }
21 }
```

7.2 Class Experiments

Class Experiments

```
29 public class Experiments {
30    /**
31     * This function executes the experiments and returns the results.
32     *
33     * @param rand The random number generator.
34     * @param arraySizes The array sizes to be tested.
35     * @param totalIterations The total number of iterations.
36     * @param dataTypes The data types to be tested.
37     * @param dataOrderings The data orderings to be tested.
38     * @param stringsSourceFile The source file for the strings to be selected
           randomly.
39     * @return The results of the experiments.
40     * @throws IOException If the source file is not found.
41     */
42    public static Stack<Result>[] runExperiments(Random rand, Stack<Long>
          arraySizes, int totalIterations, DataType[] dataTypes, DataOrdering[]
          dataOrderings, String stringsSourceFile) throws IOException {
43       int numberOfAlgorithms = 3;
44       String[] filteredStrings = readData(stringsSourceFile);
45
46       Stack<Result> BubbleSortPassPerItemResults = new Stack<Result>();
47       Stack<Result> BubbleSortUntilNoChangeResults = new Stack<Result>();
48       Stack<Result> BubbleSortWhileNeededResults = new Stack<Result>();
49       Stack<Result>[] allResults = new Stack[numberOfAlgorithms];
50       Stack<Result> results;
51
52       Sorter[] sorters = new Sorter[numberOfAlgorithms];
53       sorters[0] = new BubbleSortPassPerItem();
54       sorters[1] = new BubbleSortUntilNoChange();
55       sorters[2] = new BubbleSortWhileNeeded();
56
57       System.out.println("\nRunning experiments...");
58
59       Comparable[] randomIntegers = null, sortedIntegers = null, reversedIntegers
            = null;
60       Comparable[] randomShorts = null, sortedShorts = null, reversedShorts =
            null;
61       Comparable[] randomFloats = null, sortedFloats = null, reversedFloats =
            null;
62       Comparable[] randomStrings = null, sortedStrings = null, reversedStrings =
            null;
63
64       // Guards to avoid an error due to the available filtered strings being
            less than the max array size
65       long maxArraySize = 0;
66
67       for (int i = 0; i < arraySizes.size(); i++) {
68          if (maxArraySize == 0) {
```

```
69              maxArraySize = arraySizes.get(i);
70           } else {
71              if (arraySizes.get(i) > maxArraySize) {
72                 maxArraySize = arraySizes.get(i);
73              }
74           }
75        }
76
77        if (arraySizes.lastElement() > filteredStrings.length) {
78           System.out.println("\nThe filtered strings source file does not have
                 enough strings to fill your largest array size.");
79           System.out.println("There are only " + filteredStrings.length + "
                 strings in the filtered strings source file.");
80           System.out.println("Please, decrease the array sizes of your largest
                 array or try a different string size.");
81           exit(1);
82        }
83
84        Comparable[] toSort = null;
85
86        for (DataType type : dataTypes) {
87           for (Long arraySize : arraySizes) {
88              int size = arraySize.intValue();
89              randomIntegers = generateRandomIntegers(rand, size);
90              randomShorts = generateRandomShorts(rand, size);
91              randomFloats = generateRandomFloats(rand, size);
92              randomStrings = generateRandomStrings(rand, size, filteredStrings);
93
94              System.out.println("\nArray size: "+ size );
95
96              for (DataOrdering ordering : dataOrderings) {
97                 // Get the data to be sorted according to the type ordering and
                       array size
98                 switch (type) {
99                    case INTEGER:
100                      switch (ordering) {
101                         case RANDOM:
102                            toSort = randomIntegers.clone();
103                            break;
104                         case ASC:
105                            sortedIntegers = (Comparable[])
                                  Sorters.quickSort(randomIntegers.clone(), 0, size -
                                  1);
106                            toSort = sortedIntegers.clone();
107                            break;
108                         case DESC:
109                            reversedIntegers =
                                  Sorters.reverseArray(sortedIntegers.clone(),
                                  size).clone();
110                            toSort = reversedIntegers.clone();
111                            break;
112                         default:
```

```
                        System.out.println("Invalid ordering");
                        exit(1);
                        break;
                }
                break;
            case SHORT:
                switch (ordering) {
                    case RANDOM:
                        toSort = randomShorts.clone();
                        break;
                    case ASC:
                        sortedShorts = (Comparable[])
                            Sorters.quickSort(randomShorts.clone(), 0, size -
                            1);
                        toSort = sortedShorts.clone();
                        break;
                    case DESC:
                        reversedShorts =
                            Sorters.reverseArray(sortedShorts.clone(),
                            size).clone();
                        toSort = reversedShorts.clone();
                        break;
                    default:
                        System.out.println("Sorting order not supported");
                        exit(1);
                        break;
                }
                break;
            case FLOAT:
                switch (ordering) {
                    case RANDOM:
                        toSort = randomFloats.clone();
                        break;
                    case ASC:
                        sortedFloats = (Comparable[])
                            Sorters.quickSort(randomFloats.clone(), 0, size -
                            1);
                        toSort = sortedFloats.clone();
                        break;
                    case DESC:
                        reversedFloats = Sorters.reverseArray(sortedFloats,
                            size).clone();
                        toSort = reversedFloats.clone();
                        break;
                    default:
                        System.out.println("Invalid ordering");
                        exit(1);
                        break;
                }
                break;
            case STRING:
                switch (ordering) {
```

```
158                     case RANDOM:
159                         toSort = randomStrings.clone();
160                         break;
161                     case ASC:
162                         sortedStrings = (Comparable[])
                                Sorters.quickSort(randomStrings.clone(), 0, size -
                                1);
163                         toSort = sortedStrings.clone();
164                         break;
165                     case DESC:
166                         reversedStrings =
                                Sorters.reverseArray(sortedStrings.clone(),
                                size).clone();
167                         toSort = reversedStrings.clone();
168                         break;
169                     default:
170                         System.out.println("Invalid ordering");
171                         exit(1);
172                         break;
173                 }
174                 break;
175             default:
176                 System.out.println("Type not implemented");
177                 exit(1);
178                 break;
179         }
180
181         int allResultsArrayIndex = 0;
182         for (Sorter sorter : sorters) {
183             String className =
                        sorter.getClass().getName().split("\\.")[sorter.getClass().getName
                        - 1];
184             Result iteration_result = new Result(type, ordering, className,
                        size, totalIterations);
185
186             if (sorter instanceof BubbleSortPassPerItem) {
187                 iteration_result.setAlgorithm(BUBBLE_SORT_PASS_PER_ITEM);
188                 results = BubbleSortPassPerItemResults;
189             } else if (sorter instanceof BubbleSortUntilNoChange) {
190                 iteration_result.setAlgorithm(BUBBLE_SORT_UNTIL_NO_CHANGE);
191                 results = BubbleSortUntilNoChangeResults;
192             } else {
193                 iteration_result.setAlgorithm(BUBBLE_SORT_WHILE_NEEDED);
194                 results = BubbleSortWhileNeededResults;
195             }
196
197             System.out.print(iteration_result.getAlgorithm() + " | " +
                        iteration_result.getDataType() + " | " +
                        iteration_result.getDataOrdering() + "... ");
198             forceGarbageCollection();
199
200             // Run the sorting algorithms
```

```
                    for (int i = 0; i < totalIterations; i++) {
                        long startTime = System.nanoTime();
                        sorter.sort(toSort.clone());
                        long endTime = System.nanoTime();
                        long elapsedTime = endTime - startTime;
                        iteration_result.addElapsedTime(i, elapsedTime);
                    }

                    System.out.println("\t\t\tDone!");
                    results.push(iteration_result);
                    allResults[allResultsArrayIndex++] = results;
                }
            }
        }
    }

    return allResults;
  }

  /**
   * Runs garbage collection to free up memory
   */
  private static void forceGarbageCollection() {
    for (int i = 0; i < 6; i++) {
      System.gc();
    }
  }
}
```

7.3 Summary Results

results

Algorithm	Data Type	Data Ordering	Array size	Standard deviation	Standard error	Mean	Minimum time	Maximum time	Median	N last iterations	First quartile	Third quartile	Interquartile range	First result	Last result	v_0	v_1	v_2	v_3	v_4
BUBBLE_SORT_PASS_PER_ITEM	INTEGER	ASC	100	232.3114982565910	42.414082654319000	6548.633333333330	6417	578959	6500.0	30	6500.0	6500.0	0.0	578959.0	6459.0	6500	6500	6541	6500	7792
BUBBLE_SORT_PASS_PER_ITEM	INTEGER	RANDOM	100	1254.6163419769200	229.06055717179200	14125.033333333300	13791	356792	13875.0	30	13834.0	13917.0	83.0	356792.0	13833.0	13833	13875	13875	20875	13958
BUBBLE_SORT_PASS_PER_ITEM	INTEGER	DESC	100	48.98702776132560	8.943766710003280	17797.066666666700	17708	22916	17791.0	30	17750.0	17833.0	83.0	20417.0	17750.0	17708	17792	17792	17875	17833
BUBBLE_SORT_PASS_PER_ITEM	INTEGER	ASC	1000	17378.17543373170	3172.806231112320	872045.8333333330	863583	984375	866459.0	30	863834.0	870291.0	6457.0	870042.0	866583.0	866333	901084	943875	917417	866459
BUBBLE_SORT_PASS_PER_ITEM	INTEGER	RANDOM	1000	4318.086806291260	788.3711830237220	1365011.0	1355709	1475083	1364459.0	30	1362666.0	1368291.0	5625.0	1365083.0	1368291.0	1366166	1362125	1363917	1363833	1366708
BUBBLE_SORT_PASS_PER_ITEM	INTEGER	DESC	1000	5184.3019035031800	946.5196991552210	1638734.8	1633042	1777958	1636834.0	30	1636250.0	1638750.0	2500.0	1638792.0	1636834.0	1634042	1633834	1638708	1646041	1638750
BUBBLE_SORT_PASS_PER_ITEM	INTEGER	ASC	10000	749024.5210529490	136752.5421017350	9.0484857E+07	88916875	95182500	9.081675E+07	30	9.074675E+07	9.090575E+07	159000.0	8.9725292E+07	9.0891625E+07	90909833	90845709	90798917	90794125	90829625
BUBBLE_SORT_PASS_PER_ITEM	INTEGER	RANDOM	10000	560539.268303964	102340.0005391740	1.531036513E+08	151513541	165465625	1.5296175E+08	30	1.52671417E+08	1.53479E+08	807583.0	1.52721708E+08	1.53462625E+08	153731500	153205458	153597583	153479000	154543458
BUBBLE_SORT_PASS_PER_ITEM	INTEGER	DESC	10000	417690.14720494700	76259.43855726760	1.648164139E+08	164607334	169377083	1.64729708E+08	30	1.64695042E+08	1.64765417E+08	70375.0	1.65311916E+08	1.64865958E+08	164772750	164630875	164723958	164734584	164908375
BUBBLE_SORT_PASS_PER_ITEM	FLOAT	ASC	100	189.6646748577310	34.62787359382080	11872.333333333300	11208	392125	11792.0	30	11708.0	12083.0	375.0	392125.0	12167.0	12084	11667	12083	11834	11958
BUBBLE_SORT_PASS_PER_ITEM	FLOAT	RANDOM	100	41.54510266632590	7.585063294738230	18497.266666666700	18416	356625	18500.0	30	18459.0	18541.0	82.0	356625.0	18459.0	18500	18541	18458	18542	18542
BUBBLE_SORT_PASS_PER_ITEM	FLOAT	DESC	100	31.0439831350439	5.66782994595783	23579.266666666700	23500	29875	23583.0	30	23583.0	23584.0	1.0	29875.0	23584.0	23584	23583	23583	23584	23542
BUBBLE_SORT_PASS_PER_ITEM	FLOAT	ASC	1000	3265.585032663450	596.2115286136720	1100043.1666666700	1098125	1143042	1099458.0	30	1098916.0	1099959.0	1043.0	1143042.0	1099292.0	1098792	1099708	1098834	1099125	1098917
BUBBLE_SORT_PASS_PER_ITEM	FLOAT	RANDOM	1000	3345.0540264695300	610.7205154569480	1847948.6	1838333	1891541	1848584.0	30	1846458.0	1850208.0	3750.0	1857041.0	1848875.0	1850583	1848041	1852000	1851792	1844458
BUBBLE_SORT_PASS_PER_ITEM	FLOAT	DESC	1000	1478.5581794437400	269.9465558217030	2201636.1	2200667	2343917	2200917.0	30	2200833.0	2202167.0	1334.0	2343917.0	2200875.0	2202292	2200833	2201000	2200833	2200875
BUBBLE_SORT_PASS_PER_ITEM	FLOAT	ASC	10000	1026931.2911365100	187491.14438779	1.124211848E+08	111242542	118545917	1.1192075E+08	30	1.1159675E+08	1.13560334E+08	1963584.0	1.14233833E+08	1.11641042E+08	113677084	113701792	113520750	113655875	113514625
BUBBLE_SORT_PASS_PER_ITEM	FLOAT	RANDOM	10000	829328.0657304720	151413.89639090200	2.30328354233333E+08	223750958	233334292	2.30102959E+08	30	2.29883666E+08	2.30734583E+08	850917.0	2.30377334E+08	2.30174708E+08	229773625	230063333	229699292	230789417	230800791
BUBBLE_SORT_PASS_PER_ITEM	FLOAT	DESC	10000	82121.40957821060	14993.249493368900	2.24304140233333E+08	219933292	224677250	2.24320292E+08	30	2.24227583E+08	2.24359667E+08	132084.0	2.20330375E+08	2.24227583E+08	224344083	224241875	224400417	224330000	224520333
BUBBLE_SORT_PASS_PER_ITEM	SHORT	ASC	100	48.36119886391940	8.829506508587290	22720.833333333300	22541	355000	22709.0	30	22708.0	22750.0	42.0	355000.0	22792.0	22750	22708	22750	22709	22750
BUBBLE_SORT_PASS_PER_ITEM	SHORT	RANDOM	100	46.45265211900060	8.481055140505590	28926.533333333300	28750	40375	28917.0	30	28916.0	28958.0	42.0	39625.0	28875.0	28917	28917	28917	28916	29000
BUBBLE_SORT_PASS_PER_ITEM	SHORT	DESC	100	42.809773027507000	7.815959456280610	31627.7	31417	40208	31625.0	30	31584.0	31666.0	82.0	35000.0	31666.0	31625	31708	31666	31666	31625
BUBBLE_SORT_PASS_PER_ITEM	SHORT	ASC	1000	3940.707635268910	719.4714881232070	2241957.0	2204083	2385167	2240334.0	30	2239125.0	2245125.0	6000.0	2204083.0	2239042.0	2247584	2239125	2243750	2246208	2239750
BUBBLE_SORT_PASS_PER_ITEM	SHORT	RANDOM	1000	4802.795574338860	876.8664917171240	2759943.066666670	2745750	2919959	2759917.0	30	2756417.0	2764500.0	8083.0	2761750.0	2760459.0	2765042	2756708	2759334	2759166	2752125
BUBBLE_SORT_PASS_PER_ITEM	SHORT	DESC	1000	6321.951400916930	1154.2251299112000	3121255.533333330	3081625	3284208	3119250.0	30	3117417.0	3124917.0	7500.0	3081625.0	3121417.0	3117250	3115416	3114417	3118208	3114667
BUBBLE_SORT_PASS_PER_ITEM	SHORT	ASC	10000	96083.37279213410	17542.34355981	2.28043227766667E+08	227906541	228841958	2.28016125E+08	30	2.27996416E+08	2.280445E+08	48084.0	2.28265042E+08	2.28426084E+08	228011417	228006875	228130208	228026042	228022917
BUBBLE_SORT_PASS_PER_ITEM	SHORT	RANDOM	10000	3036316.7770154700	554353.0635009200	3.365711944E+08	330380292	511501333	3.3627325E+08	30	3.33549625E+08	3.39266458E+08	5716833.0	3.43793E+08	3.35915375E+08	341383833	338670166	339239583	341309250	340612167
BUBBLE_SORT_PASS_PER_ITEM	SHORT	DESC	10000	1123270.0486358600	205080.1146025950	3.149269263E+08	311741584	333089833	3.14674958E+08	30	3.14083667E+08	3.159175E+08	1833833.0	3.16466166E+08	3.15960042E+08	315987042	316003125	315869875	315903958	316291375
BUBBLE_SORT_PASS_PER_ITEM	STRING	ASC	100	33.39860276518570	6.097722707882200	38357.0	37583	309250	38375.0	30	38333.0	38375.0	42.0	309250.0	38334.0	38333	38333	38375	38333	38375
BUBBLE_SORT_PASS_PER_ITEM	STRING	RANDOM	100	474.2966500221380	86.594324722086	54591.566666666700	54375	73375	54500.0	30	54458.0	54542.0	84.0	73375.0	54500.0	54458	54458	54500	54458	57125
BUBBLE_SORT_PASS_PER_ITEM	STRING	DESC	100	493.67092165620700	90.1315665918234	47684.76666666670	47500	58000	47584.0	30	47583.0	47625.0	42.0	55709.0	47625.0	47625	47625	47542	47583	47583
BUBBLE_SORT_PASS_PER_ITEM	STRING	ASC	1000	31462.474841291300	5744.235728504680	4700395.833333330	4558000	6640458	4696416.0	30	4683875.0	4713791.0	29916.0	5178167.0	4714125.0	4720958	4720333	4834375	4703917	4710333
BUBBLE_SORT_PASS_PER_ITEM	STRING	RANDOM	1000	31820.730209444000	5809.643910666190	7635329.266666670	7449542	8015125	7633875.0	30	7618542.0	7656708.0	38166.0	7518750.0	7710917.0	7630500	7678917	7630000	7633875	7590250
BUBBLE_SORT_PASS_PER_ITEM	STRING	DESC	1000	13522.708615469300	2468.897515754010	4939333.366666670	4910000	5320167	4938417.0	30	4929834.0	4949042.0	19208.0	5160250.0	4964458.0	4931750	4949791	4919417	4941458	4942875
BUBBLE_SORT_PASS_PER_ITEM	STRING	ASC	10000	1.44594922152433E+07	2639930.0187863600	9.47196080566667E+08	904143417	1022278542	9.52158667E+08	30	9.313625E+08	9.59228583E+08	2.7866083E+07	1.022278542E+09	9.26985084E+08	931362500	937516750	938995958	931792125	930572167
BUBBLE_SORT_PASS_PER_ITEM	STRING	RANDOM	10000	9258255.274261630	1690318.4189514200	1.05015707776667E+09	1026217625	1365386583	1.054430583E+09	30	1.045139667E+09	1.056175375E+09	1.1035708E+07	1.045924041E+09	1.067714209E+09	1055389458	1046385167	1058116916	1056175375	1059265500
BUBBLE_SORT_PASS_PER_ITEM	STRING	DESC	10000	5219019.279548160	952858.1958209630	6.52845247266667E+08	646248291	701548750	6.51061417E+08	30	6.49277833E+08	6.55652542E+08	6374709.0	6.76112583E+08	6.53633625E+08	658944833	655652542	648435000	648773500	649277833
BUBBLE_SORT_UNTIL_NO_CHANGE	INTEGER	ASC	100	23.54242034191800	4.2982382265123300	619.4333333333330	583	11666	625.0	30	625.0	625.0	0.0	11666.0	709.0	625	583	583	625	625
BUBBLE_SORT_UNTIL_NO_CHANGE	INTEGER	RANDOM	100	35.18276093012980	6.4234639322478400	16352.8	16250	201666	16375.0	30	16333.0	16375.0	42.0	201666.0	16375.0	16334	16417	16292	16375	16375
BUBBLE_SORT_UNTIL_NO_CHANGE	INTEGER	DESC	100	42.35680451697090	7.733259099260560	22234.633333333300	22125	31708	22250.0	30	22208.0	22250.0	42.0	27125.0	22166.0	22250	22166	22291	22250	22250
BUBBLE_SORT_UNTIL_NO_CHANGE	INTEGER	ASC	1000	833.7165179017520	152.21511447315000	1848.6333333333300	1625	6834	1708.0	30	1667.0	1708.0	41.0	6834.0	1750.0	1667	1667	1708	1666	1708
BUBBLE_SORT_UNTIL_NO_CHANGE	INTEGER	RANDOM	1000	3690.130271864850	673.7225300110160	1691910.9	1685542	1813833	1691125.0	30	1689166.0	1694416.0	5250.0	1721875.0	1689375.0	1688416	1689166	1689583	1698250	1694708
BUBBLE_SORT_UNTIL_NO_CHANGE	INTEGER	DESC	1000	3392.7593581364200	619.4302775460210	2143031.933333330	2137666	2305959	2141042.0	30	2140500.0	2147083.0	6583.0	2147291.0	2147375.0	2141667	2140334	2140458	2147083	2140334
BUBBLE_SORT_UNTIL_NO_CHANGE	INTEGER	ASC	10000	663.2546821043430	121.09318358648900	18426.4	14625	284083	18500.0	30	17875.0	18875.0	1000.0	17917.0	18500.0	18250	19000	18083	19042	18083
BUBBLE_SORT_UNTIL_NO_CHANGE	INTEGER	RANDOM	10000	1776292.9831422600	324305.2452017210	1.84092254233333E+08	181340291	187497750	1.8528125E+08	30	1.82005709E+08	1.85527125E+08	3521416.0	1.83048917E+08	1.81557167E+08	185281250	185708167	185527125	185315542	185361333
BUBBLE_SORT_UNTIL_NO_CHANGE	INTEGER	DESC	10000	243098.49468344200	44383.50974455700	2.16991190333333E+08	212844292	218688666	2.16934208E+08	30	2.16909291E+08	2.16976375E+08	67084.0	2.13347292E+08	2.16928584E+08	216954625	216993917	216914333	216918042	216916333
BUBBLE_SORT_UNTIL_NO_CHANGE	FLOAT	ASC	100	84.42530557967930	15.413881430085700	2716.633333333330	2625	21625	2708.0	30	2667.0	2709.0	42.0	21625.0	2666.0	2708	2666	2667	2709	2708
BUBBLE_SORT_UNTIL_NO_CHANGE	FLOAT	RANDOM	100	42.6590618222618	7.788443480686740	20656.933333333300	20542	324750	20666.0	30	20625.0	20667.0	42.0	324750.0	20708.0	20625	20667	20625	20667	20709
BUBBLE_SORT_UNTIL_NO_CHANGE	FLOAT	DESC	100	46.65953516937580	8.518826644990930	27994.566666666700	27875	34292	28000.0	30	27958.0	28000.0	42.0	34292.0	27958.0	28042	27959	28000	27959	27958
BUBBLE_SORT_UNTIL_NO_CHANGE	FLOAT	ASC	1000	31.12806879115160	5.683181816162100	1861.1	1791	6084	1875.0	30	1833.0	1875.0	42.0	6084.0	1833.0	1834	1875	1959	1833	1833
BUBBLE_SORT_UNTIL_NO_CHANGE	FLOAT	RANDOM	1000	1570.8165463294000	286.7905520423240	2126815.3333333300	2121333	2193833	2127083.0	30	2125417.0	2127708.0	2291.0	2134875.0	2127208.0	2128375	2127375	2126709	2126625	2125209
BUBBLE_SORT_UNTIL_NO_CHANGE	FLOAT	DESC	1000	1030.62751111479	188.16597874099200	2608961.0	2608125	2749791	2608292.0	30	2608250.0	2609708.0	1458.0	2616375.0	2608250.0	2609666	2608250	2608167	2608208	2609708
BUBBLE_SORT_UNTIL_NO_CHANGE	FLOAT	ASC	10000	741.4988581694960	135.37855032791900	19756.9	16583	225459	19792.0	30	19125.0	20333.0	1208.0	20917.0	20167.0	20375	20333	19375	20458	19500
BUBBLE_SORT_UNTIL_NO_CHANGE	FLOAT	RANDOM	10000	769150.5134421000	140427.02877630600	2.52781626433333E+08	250395959	263767500	2.52482791E+08	30	2.52267334E+08	2.52912E+08	644666.0	2.56830209E+08	2.52167125E+08	254247584	253866958	252086958	253992334	252344208
BUBBLE_SORT_UNTIL_NO_CHANGE	FLOAT	DESC	10000	81056.48131752880	14798.821083202200	2.65158047233333E+08	259975041	266280000	2.65127417E+08	30	2.65109583E+08	2.65241875E+08	132292.0	2.6025775E+08	2.65128041E+08	265129833	265112584	265261958	265114167	265104334
BUBBLE_SORT_UNTIL_NO_CHANGE	SHORT	ASC	100	17.693564429538300	3.230388120243050	657.0666666666670	584	13708	667.0	30	666.0	667.0	1.0	13708.0	667.0	667	667	667	667	625
BUBBLE_SORT_UNTIL_NO_CHANGE	SHORT	RANDOM	100	187.95449803964100	34.31563945362390	32304.2	31792	106417	32292.0	30	32125.0	32500.0	375.0	106417.0	32541.0	32500	32042	32125	32500	32292
BUBBLE_SORT_UNTIL_NO_CHANGE	SHORT	DESC	100	534.3584117633240	97.56005197180220	36223.566666666700	35375	55625	36125.0	30	36083.0	36167.0	84.0	39417.0	36083.0	36041	36292	36208	36208	36209
BUBBLE_SORT_UNTIL_NO_CHANGE	SHORT	ASC	1000	22.883302986141600	4.177900411911690	2790.233333333330	2708	5375	2792.0	30	2791.0	2792.0	1.0	5375.0	2792.0	2791	2792	2792	2791	2834
BUBBLE_SORT_UNTIL_NO_CHANGE	SHORT	RANDOM	1000	11587.77599642540	2115.62876818637	3748383.3	3722292	3919625	3747125.0	30	3740958.0	3753750.0	12792.0	3770041.0	3750875.0	3744834	3747541	3752375	3747125	3735250
BUBBLE_SORT_UNTIL_NO_CHANGE	SHORT	DESC	1000	4507.082701334280	822.8769546873670	3483615.3	3426333	3646625	3483791.0	30	3479292.0	3486250.0	6958.0	3426333.0	3486250.0	3480917	3479250	3485750	3483584	3486333
BUBBLE_SORT_UNTIL_NO_CHANGE	SHORT	ASC	10000	882.7877944330680	161.17426283374200	30206.9	27041	216834	30208.0	30	29583.0	30542.0	959.0	31041.0	29917.0	30250	29083	30458	29333	30292
BUBBLE_SORT_UNTIL_NO_CHANGE	SHORT	RANDOM	10000	1816784.4257808000	331697.9373747380	4.19288336166667E+08	414477625	428032458	4.19128708E+08	30	4.17734542E+08	4.20285375E+08	2550833.0	4.19954667E+08	4.22150125E+08	418102167	419514542	417242958	419049583	419943917
BUBBLE_SORT_UNTIL_NO_CHANGE	SHORT	DESC	10000	334449.5219997320	61061.849182024400	3.50167963866667E+08	349862458	363880500	3.50090959E+08	30	3.50014E+08	3.50146291E+08	132291.0	3.50607334E+08	3.50121875E+08	351415833	350174000	349991291	349982417	350272083
BUBBLE_SORT_UNTIL_NO_CHANGE	STRING	ASC	100	20.69836279086400	3.778986734660600	601.3333333333330	542	16042	584.0	30	583.0	625.0	42.0	16042.0	584.0	625	625	584	584	584
BUBBLE_SORT_UNTIL_NO_CHANGE	STRING	RANDOM	100	52.15392815716020	9.521960971393490	62501.36666666670	62291	83792	62500.0	30	62458.0	62542.0	84.0	78917.0	62500.0	62458	62416	62542	62541	62458
BUBBLE_SORT_UNTIL_NO_CHANGE	STRING	DESC	100	796.7245927336920	145.4613438531200	60008.3	59666	70125	59875.0	30	59833.0	59875.0	42.0	67833.0	59875.0	59833	59875	59875	59875	59833
BUBBLE_SORT_UNTIL_NO_CHANGE	STRING	ASC	1000	51.21636023338200	9.350785271044630	7980.533333333330	7833	16792	7959.0	30	7958.0	8000.0	42.0	16792.0	8041.0	7917	8000	8000	7959	8041
BUBBLE_SORT_UNTIL_NO_CHANGE	STRING	RANDOM	1000	46710.357509467000	8528.105492355310	8886290.266666670	8810500	9415459	8879458.0	30	8855042.0	8904459.0	49417.0	9077458.0	8995000.0	8882875	8879458	8840292	8864958	8832916
BUBBLE_SORT_UNTIL_NO_CHANGE	STRING	DESC	1000	24126.306385262400	4404.840745496390	6515695.933333330	6469792	7101584	6516584.0	30	6498750.0	6529667.0	30917.0	6721583.0	6592375.0	6529666	6521625	6479083	6516584	6501334
BUBBLE_SORT_UNTIL_NO_CHANGE	STRING	ASC	10000	781.7761607746530	142.73214605868800	111306.96666666700	110875	362125	111083.0	30	111000.0	111167.0	167.0	132917.0	111000.0	111084	111000	110959	111291	111084
BUBBLE_SORT_UNTIL_NO_CHANGE	STRING	RANDOM	10000	1.15341004036627E+07	2105828.990538500	1.16551690143333E+09	1143492000	1242305167	1.16447525E+09	30	1.154396792E+09	1.175263625E+09	2.0866833E+07	1.163454709E+09	1.150147083E+09	1158408334	1164475250	1150450375	1156897500	1153339083
BUBBLE_SORT_UNTIL_NO_CHANGE	STRING	DESC	10000	8757210.2310639	1598840.5281229100	8.14977881966667E+08	799788250	1010318625	8.11089792E+08	30	8.07518375E+08	8.21685375E+08	1.4167E+07	8.29415417E+08	8.31904084E+08	819919167	833343583	823664667	825680208	819680542
BUBBLE_SORT_WHILE_NEEDED	INTEGER	ASC	100	20.090793911640200	3.668060341197600	651.4	583	10042	666.0	30	625.0	667.0	42.0	8459.0	625.0	666	625	667	667	667
BUBBLE_SORT_WHILE_NEEDED	INTEGER	RANDOM	100	32.87053729743740	6.001311585040450	9995.833333333330	9916	149459	10000.0	30	9959.0	10041.0	82.0	60000.0	10000.0	10000	9958	9958	10041	10000
BUBBLE_SORT_WHILE_NEEDED	INTEGER	DESC	100	27.26864296016380	4.978550287278720	14223.566666666700	14125	21584	14209.0	30	14208.0	14250.0	42.0	17083.0	14250.0	14208	14250	14250	14167	14250
BUBBLE_SORT_WHILE_NEEDED	INTEGER	ASC	1000	26.198897347450000	4.783242352986870	1530.5333333333300	1458	3875	1541.0	30	1500.0	1542.0	42.0	3875.0	1583.0	1500	1542	1542	1500	1584
BUBBLE_SORT_WHILE_NEEDED	INTEGER	RANDOM	1000	3362.6643959812600	613.9357143328070	1040548.4	1034750	1137000	1040209.0	30	1038166.0	1042250.0	4084.0	1063125.0	1042667.0	1043916	1040541	1039666	1049375	1038917
BUBBLE_SORT_WHILE_NEEDED	INTEGER	DESC	1000	4144.2659965306300	756.6359902004840	1393137.5	1388666	1510625	1391542.0	30	1391417.0	1396000.0	4583.0	1484125.0	1398375.0	1396000	1391417	1388750	1391584	1391458
BUBBLE_SORT_WHILE_NEEDED	INTEGER	ASC	10000	102.57807866314400	18.728109196447800	13913.933333333300	13667	162667	13875.0	30	13833.0	13958.0	125.0	20042.0	13958.0	14125	13958	13959	13958	13875
BUBBLE_SORT_WHILE_NEEDED	INTEGER	RANDOM	10000	190946.22305082000	34861.85121178230	1.030349166E+08	100876250	104393750	1.03010417E+08	30	1.02914459E+08	1.03068833E+08	154374.0	1.0316875E+08	1.03010417E+08	103015958	103013250	103052042	102917167	103055291
BUBBLE_SORT_WHILE_NEEDED	INTEGER	DESC	10000	72674.58632318150	13268.503428854300	1.38600515233333E+08	138558042	143366583	1.38572208E+08	30	1.38566417E+08	1.38574916E+08	8499.0	1.41321E+08	1.38874291E+08	138566791	138572000	138571125	138567000	138566292
BUBBLE_SORT_WHILE_NEEDED	FLOAT	ASC	100	100.55280536447800	18.358346572850400	2568.0	2458	29875	2542.0	30	2500.0	2542.0	42.0	29084.0	2500.0	2541	2541	2833	2500	2500
BUBBLE_SORT_WHILE_NEEDED	FLOAT	RANDOM	100	215.62318211794100	39.367226922346900	14711.1	14333	210417	14666.0	30	14625.0	14708.0	83.0	210417.0	14667.0	14667	14708	14708	14667	14667
BUBBLE_SORT_WHILE_NEEDED	FLOAT	DESC	100	34.72072707892060	6.339108478035090	17754.266666666700	17666	22959	17750.0	30	17750.0	17750.0	0.0	22959.0	17750.0	17750	17709	17750	17792	17708
BUBBLE_SORT_WHILE_NEEDED	FLOAT	ASC	1000	24.3804338672542	4.451237863619350	1845.8333333333300	1708	4708	1834.0	30	1833.0	1834.0	1.0	4708.0	1833.0	1833	1834	1833	1833	1834
BUBBLE_SORT_WHILE_NEEDED	FLOAT	RANDOM	1000	5266.33502162557	961.4968289079270	1476834.8	1462417	1751417	1476750.0	30	1472375.0	1480791.0	8416.0	1482166.0	1474375.0	1472375	1475209	1469292	1469375	1480417
BUBBLE_SORT_WHILE_NEEDED	FLOAT	DESC	1000	730.9417547854880	133.4510957728070	1721938.8666666700	1721458	1776500	1721625.0	30	1721583.0	1721709.0	126.0	1727375.0	1721709.0	1721542	1721667	1721625	1721625	1721542
BUBBLE_SORT_WHILE_NEEDED	FLOAT	ASC	10000	99.73253676152480	18.208586700500100	16355.566666666700	15875	170583	16333.0	30	16292.0	16375.0	83.0	30500.0	16375.0	16333	16333	16292	16334	16333
BUBBLE_SORT_WHILE_NEEDED	FLOAT	RANDOM	10000	819010.9072666730	149530.24958424300	2.15960808433333E+08	212605458	224306167	2.15872E+08	30	2.15580792E+08	2.16332292E+08	751500.0	2.18166459E+08	2.15699875E+08	217294667	216317666	215760500	215613834	217186250
BUBBLE_SORT_WHILE_NEEDED	FLOAT	DESC	10000	55178.6266709614	10074.192839947200	1.752410042E+08	174764833	175540666	1.75222041E+08	30	1.75210083E+08	1.75236833E+08	26750.0	1.74913958E+08	1.75222041E+08	175214583	175354000	175223167	175206167	175221209
BUBBLE_SORT_WHILE_NEEDED	SHORT	ASC	100	52.36900058452730	9.561227644715610	690.2333333333330	625	11958	667.0	30	666.0	708.0	42.0	11958.0	708.0	667	708	666	666	875
BUBBLE_SORT_WHILE_NEEDED	SHORT	RANDOM	100	31.128925455273900	5.683338220916770	19194.3	18791	84792	19208.0	30	19166.0	19208.0	42.0	52833.0	19208.0	19208	19167	19250	19166	19166
BUBBLE_SORT_WHILE_NEEDED	SHORT	DESC	100	29.937360530874400	5.465789224974910	22562.433333333300	22458	37041	22583.0	30	22542.0	22583.0	41.0	37041.0	22584.0	22583	22583	22500	22625	22541
BUBBLE_SORT_WHILE_NEEDED	SHORT	ASC	1000	28.440288324839500	5.192462485821800	2719.5	2625	4708	2708.0	30	2708.0	2709.0	1.0	4708.0	2750.0	2709	2709	2708	2708	2750
BUBBLE_SORT_WHILE_NEEDED	SHORT	RANDOM	1000	4417.574504056370	806.5350684437900	1834300.0333333300	1824375	1915791	1834042.0	30	1831125.0	1836291.0	5166.0	1836541.0	1826291.0	1840542	1831125	1833000	1834125	1836291
BUBBLE_SORT_WHILE_NEEDED	SHORT	DESC	1000	3409.437897790320	622.4753483442510	2257697.2333333300	2252333	2412709	2256334.0	30	2255084.0	2262000.0	6916.0	2264000.0	2262125.0	2262083	2255125	2255084	2255166	2259542
BUBBLE_SORT_WHILE_NEEDED	SHORT	ASC	10000	99.29247481836460	18.128242749510600	26131.933333333300	25916	192542	26125.0	30	26083.0	26167.0	84.0	34375.0	26459.0	26041	26125	26167	26125	26083
BUBBLE_SORT_WHILE_NEEDED	SHORT	RANDOM	10000	2024155.0321297500	369558.4569950200	2.45400586266667E+08	239462584	255398000	2.45950084E+08	30	2.44155084E+08	2.46936375E+08	2781291.0	2.41166042E+08	2.44596583E+08	243426084	249442500	247207417	245710584	246936375
BUBBLE_SORT_WHILE_NEEDED	SHORT	DESC	10000	146191.158969648	26690.731825166600	2.25302298666667E+08	224455125	233275292	2.252505E+08	30	2.25208959E+08	2.25361083E+08	152124.0	2.25429541E+08	2.25182625E+08	225165916	225208959	225351042	225226250	225386625
BUBBLE_SORT_WHILE_NEEDED	STRING	ASC	100	22.797270791819700	4.162193154077770	598.4666666666670	541	7208	583.0	30	583.0	625.0	42.0	7208.0	583.0	583	583	583	625	583
BUBBLE_SORT_WHILE_NEEDED	STRING	RANDOM	100	1312.2125204740600	239.57613256811800	38151.36666666670	37708	51625	37834.0	30	37792.0	37875.0	83.0	51625.0	37792.0	37875	37750	37917	37750	37875
BUBBLE_SORT_WHILE_NEEDED	STRING	DESC	100	66.92799613515010	12.219324403946800	33968.1	33166	44875	33959.0	30	33958.0	34000.0	42.0	38583.0	34000.0	33958	33875	33959	33916	33959
BUBBLE_SORT_WHILE_NEEDED	STRING	ASC	1000	147.2437170891250	26.882901766874200	7862.433333333330	7625	17875	7875.0	30	7709.0	7958.0	249.0	17875.0	7708.0	7791	7917	8041	8042	7750
BUBBLE_SORT_WHILE_NEEDED	STRING	RANDOM	1000	14121.93913360820	2578.3015397307200	5133422.2	5107916	5494750	5131375.0	30	5122375.0	5141250.0	18875.0	5144166.0	5125375.0	5130291	5134542	5115833	5151959	5116708
BUBBLE_SORT_WHILE_NEEDED	STRING	DESC	1000	6545.725093185290	1195.080429588400	3305490.2666666700	3246750	3887792	3305542.0	30	3300791.0	3309916.0	9125.0	3246750.0	3307000.0	3310458	3305542	3296875	3310458	3304750
BUBBLE_SORT_WHILE_NEEDED	STRING	ASC	10000	25334.49652086170	4625.425075837380	120968.03333333300	111250	362875	115875.0	30	115458.0	116333.0	875.0	134666.0	117208.0	115334	115459	115416	115583	115625
BUBBLE_SORT_WHILE_NEEDED	STRING	RANDOM	10000	1528227.6506405200	279014.9190863130	6.16295773566667E+08	612332708	633548125	6.163005E+08	30	6.14820333E+08	6.17332292E+08	2511959.0	6.31998084E+08	6.16927209E+08	614667541	613983583	614464084	614782000	614414875
BUBBLE_SORT_WHILE_NEEDED	STRING	DESC	10000	3177005.307756930	580039.1574573700	3.32219454166667E+08	327676042	337849125	3.3429425E+08	30	3.28619875E+08	3.35237708E+08	6617833.0	3.32443375E+08	3.36352209E+08	329368209	328559208	328388958	328078584	329169583

1

Algorithm	v_5	v_6	v_7	v_8	v_9	v_10	v_11	v_12	v_13	v_14	v_15	v_16	v_17	v_18	v_19	v_20	v_21	v_22	v_23	v_24	v_25	v_26	v_27	v_28	v_29
BUBBLE_SORT_PASS_PER_ITEM	6500	6500	6500	6500	6500	6500	6500	6500	6625	6500	6500	6500	6542	6500	6500	6500	6500	6500	6500	6500	6500	6500	6500	6500	6459
BUBBLE_SORT_PASS_PER_ITEM	13959	13958	13917	13833	14000	13958	13833	13917	14042	13875	13833	13834	13917	13917	13875	13834	13917	13875	13834	13875	13875	13833	13875	13916	13833
BUBBLE_SORT_PASS_PER_ITEM	17750	17750	17791	17791	17791	17791	17875	17750	17875	17750	17875	17792	17750	17791	17833	17709	17917	17792	17792	17833	17791	17791	17791	17791	17750
BUBBLE_SORT_PASS_PER_ITEM	863750	870417	863750	863834	866458	867208	870667	874917	866458	864875	863750	866459	863708	870291	863708	866417	870875	868125	863708	868125	866416	865416	866500	863792	866583
BUBBLE_SORT_PASS_PER_ITEM	1372667	1363042	1363709	1359083	1362666	1360708	1368291	1369541	1375916	1359083	1363875	1362666	1365416	1363833	1368959	1364667	1358917	1368958	1370708	1359667	1364459	1367417	1356667	1368375	1368291
BUBBLE_SORT_PASS_PER_ITEM	1636792	1634417	1636833	1648291	1636208	1636500	1636875	1634125	1643542	1641375	1635750	1636417	1638750	1643667	1636625	1637791	1636250	1636292	1658625	1633042	1636667	1638042	1636959	1644000	1636834
BUBBLE_SORT_PASS_PER_ITEM	91080250	90802375	91349667	90934125	89261500	89115250	89170167	89153917	89201042	89018792	89089500	90746750	90967583	90905750	90879583	90816750	90800209	90812000	90840000	90902708	90910209	90808583	91097666	90811500	90891625
BUBBLE_SORT_PASS_PER_ITEM	152756292	152923041	152715625	152532750	152613208	152563834	152772958	153116417	153522958	152777083	152615208	152187083	153591084	154080333	152961750	152594042	154263250	153217250	153436416	152386500	153125750	152671417	152747708	152917958	153462625
BUBBLE_SORT_PASS_PER_ITEM	164717167	164679792	164705958	164705833	164714959	164807708	167031916	164740167	164765417	164698000	164749500	164695042	164907291	164694875	164671500	164729708	164679333	164681792	164914417	164704625	164730750	164645667	164742625	164741875	164865958
BUBBLE_SORT_PASS_PER_ITEM	11708	11750	11708	11709	11875	11709	11750	12125	12042	11917	11709	11708	12166	12125	11708	11708	11667	11792	12167	12167	11709	11667	11708	12083	12167
BUBBLE_SORT_PASS_PER_ITEM	18459	18500	18459	18500	18584	18417	18500	18542	18500	18458	18500	18500	18458	18541	18500	18583	18500	18458	18458	18542	18417	18500	18500	18500	18459
BUBBLE_SORT_PASS_PER_ITEM	23625	23584	23542	23583	23583	23584	23583	23583	23584	23583	23583	23584	23500	23584	23625	23542	23625	23583	23542	23625	23583	23500	23625	23583	23584
BUBBLE_SORT_PASS_PER_ITEM	1098250	1099458	1100083	1099167	1099042	1099959	1100042	1099708	1098916	1098875	1102708	1098917	1099459	1100167	1099666	1098375	1098750	1099709	1098792	1099458	1099583	1117084	1100167	1100292	1099292
BUBBLE_SORT_PASS_PER_ITEM	1841833	1843000	1847125	1848417	1848167	1851333	1849208	1842209	1850208	1853583	1849542	1848542	1849042	1847584	1846458	1848708	1841000	1850708	1847542	1845709	1853208	1841583	1848584	1849416	1848875
BUBBLE_SORT_PASS_PER_ITEM	2203708	2200792	2200917	2200917	2202167	2200833	2201000	2200875	2200875	2202291	2206500	2200833	2200750	2200875	2202042	2200833	2200875	2200875	2202292	2200959	2206250	2200750	2201000	2203166	2200875
BUBBLE_SORT_PASS_PER_ITEM	113675833	113989167	113472625	113861583	113560334	114417375	111517625	111242542	111358125	111890750	111802209	111738250	111755709	112083000	111473250	111920750	111303666	111499584	111730000	111688125	111596750	111976083	111950916	111420125	111641042
BUBBLE_SORT_PASS_PER_ITEM	229361667	229899000	229975083	229969042	231339209	230381875	230562584	229938792	230102959	230213333	232016833	229230625	229883666	232821959	230101250	231006542	229342708	230112375	229952292	229798375	230190000	229453542	230734583	232161167	230174708
BUBBLE_SORT_PASS_PER_ITEM	224332625	224136625	224359667	224397958	224244709	224203750	224320292	224216500	224365292	224272875	224359584	224275166	224404875	224333250	224222333	224216833	224206125	224395083	224226667	224267750	224281833	224409708	224280625	224329791	224227583
BUBBLE_SORT_PASS_PER_ITEM	22709	22708	22750	22625	22792	22667	22709	22708	22791	22708	22750	22541	22708	22709	22750	22750	22708	22750	22708	22709	22708	22750	22708	22750	22792
BUBBLE_SORT_PASS_PER_ITEM	28958	28958	28917	28917	28875	29000	28875	28959	28875	28917	28875	29000	28917	28917	29000	28917	28875	28958	28834	29042	28917	28917	28917	28917	28875
BUBBLE_SORT_PASS_PER_ITEM	31542	31625	31667	31625	31625	31583	31708	31625	31584	31583	31666	31625	31708	31625	31625	31667	31584	31625	31625	31541	31625	31583	31625	31584	31666
BUBBLE_SORT_PASS_PER_ITEM	2239708	2240459	2246292	2239083	2238709	2239416	2251334	2239416	2240084	2247041	2240834	2241333	2238708	2245125	2239875	2240334	2250125	2236166	2236834	2244000	2248250	2239375	2238750	2242000	2239042
BUBBLE_SORT_PASS_PER_ITEM	2765750	2758334	2754833	2764500	2759917	2757541	2766334	2764750	2758625	2767750	2766625	2752791	2762959	2757875	2760916	2761500	2751750	2763000	2756417	2754250	2762375	2768333	2753583	2754750	2760459
BUBBLE_SORT_PASS_PER_ITEM	3114875	3116958	3143125	3119250	3115500	3119250	3118542	3125125	3126583	3122542	3117834	3119583	3133916	3117459	3128750	3124416	3117417	3128375	3117458	3118583	3124917	3120083	3128167	3117583	3121417
BUBBLE_SORT_PASS_PER_ITEM	228005625	227950500	228161375	228044500	228022791	228007834	228168583	228029083	227991958	228013417	228041875	228146041	227941917	228016125	227966375	228068542	228000375	227943917	228043083	228180500	227996416	227968125	228001542	227962791	228426084
BUBBLE_SORT_PASS_PER_ITEM	338715875	337503417	340300708	339150500	340847541	339266458	339436375	339442791	338006459	334190791	336273250	335408709	333008625	334218667	332834583	334112458	333158250	332429292	331940208	334658292	333387375	334999334	333165875	333549625	335915375
BUBBLE_SORT_PASS_PER_ITEM	315917500	315884625	316226708	312899208	312778292	313483791	313914417	313376625	314443958	314566250	314510083	314096541	314918292	314300625	314644625	314674958	313838375	313839291	314083667	314523208	317201833	315962542	315830792	315876166	315960042
BUBBLE_SORT_PASS_PER_ITEM	38334	38375	38292	38375	38375	38375	38375	38375	38292	38375	38375	38375	38375	38334	38333	38417	38333	38375	38333	38334	38375	38334	38458	38333	38334
BUBBLE_SORT_PASS_PER_ITEM	54458	54500	54500	54500	54458	54542	54416	54500	54500	54583	54500	54500	54542	54458	54500	54666	54667	54416	54417	54542	54458	54500	54542	54583	54500
BUBBLE_SORT_PASS_PER_ITEM	47583	47542	47625	47625	47584	47625	47542	47541	47667	47583	47583	47583	47584	47708	47625	47500	47583	47625	47584	47625	47584	47541	50334	47584	47625
BUBBLE_SORT_PASS_PER_ITEM	4722084	4688750	4707542	4711833	4731208	4686417	4696416	4725542	4713791	4660459	4688791	4668959	4694625	4659166	4673125	4680584	4697791	4670542	4686791	4692459	4702042	4672083	4692959	4683875	4714125
BUBBLE_SORT_PASS_PER_ITEM	7633334	7582333	7656708	7622834	7622750	7590750	7618542	7573500	7616417	7577041	7631250	7670000	7642417	7616250	7641792	7657125	7649625	7636917	7686667	7656084	7662750	7663208	7650250	7626875	7710917
BUBBLE_SORT_PASS_PER_ITEM	4959000	4921208	4935417	4929834	4966125	4920084	4937500	4935084	4949042	4922041	4941750	4935750	4959208	4925292	4944708	4928417	4950125	4934459	4942750	4935583	4957750	4914167	4938417	4946541	4964458
BUBBLE_SORT_PASS_PER_ITEM	927648375	927346167	956319208	964623292	967148792	964333375	961990041	957906417	971874208	962815209	955056666	960958459	954026500	959228583	958529500	953544083	952158667	949046958	951585583	943534875	931644167	929913292	929494208	927931208	926985084
BUBBLE_SORT_PASS_PER_ITEM	1035032375	1033169000	1038406500	1031076416	1045139667	1060388417	1054430583	1056737959	1054864583	1055929083	1055711125	1039710917	1035982375	1045440042	1036085583	1046382750	1055777750	1052264583	1052293709	1055267500	1056654041	1052391209	1054174625	1058354916	1067714209
BUBBLE_SORT_PASS_PER_ITEM	647045167	647557167	651282458	650535542	649981416	649354250	652281958	649755125	650673750	648031209	649947750	648584375	651889000	652449292	651061417	650740625	653853333	657026167	667030750	667938708	659795542	658523750	656635417	648665917	653633625
BUBBLE_SORT_UNTIL_NO_CHANGE	583	625	625	625	583	584	583	625	625	625	625	625	625	625	625	625	625	625	625	625	625	625	625	625	709
BUBBLE_SORT_UNTIL_NO_CHANGE	16375	16292	16333	16416	16333	16375	16333	16292	16375	16375	16333	16333	16375	16333	16334	16375	16375	16333	16334	16375	16375	16333	16417	16292	16375
BUBBLE_SORT_UNTIL_NO_CHANGE	22209	22250	22250	22208	22250	22166	22250	22292	22208	22250	22333	22208	22167	22250	22250	22208	22250	22250	22167	22292	22250	22209	22208	22291	22166
BUBBLE_SORT_UNTIL_NO_CHANGE	1709	1708	1666	1666	1708	1667	1709	1875	1709	1708	1666	1708	1667	1667	1667	1709	1667	1667	1667	1708	1708	1667	1667	6333	1750
BUBBLE_SORT_UNTIL_NO_CHANGE	1689791	1692416	1690083	1696000	1689916	1695791	1688583	1686584	1697208	1688166	1691125	1694083	1694416	1691416	1687208	1691917	1688584	1703208	1688166	1693625	1690625	1690834	1695042	1693042	1689375
BUBBLE_SORT_UNTIL_NO_CHANGE	2145083	2140625	2147250	2140375	2142291	2148125	2140292	2140792	2140500	2151542	2140500	2137708	2141042	2147417	2144750	2140667	2140375	2147416	2142333	2141000	2147666	2140541	2140583	2144834	2147375
BUBBLE_SORT_UNTIL_NO_CHANGE	19583	17834	18875	17917	17875	18959	19041	18625	17875	18583	17542	18833	17791	20500	18500	18959	17917	18625	17583	18542	17833	17625	18792	17625	18500
BUBBLE_SORT_UNTIL_NO_CHANGE	185908292	185323125	185172583	185341167	185276625	185756209	185355125	185538833	185057167	185852792	185418250	186300500	185793167	183034250	181722833	182037459	182277166	181755917	182005709	181570416	181800750	182099625	181901208	181717875	181557167
BUBBLE_SORT_UNTIL_NO_CHANGE	216961833	216912209	216934208	216906334	216949208	216935875	216909291	216912625	216980291	217116000	216906167	216961292	217119959	216981458	216908958	216915042	216903167	216987500	216907292	218268292	216941292	216908875	216976375	216906333	216928584
BUBBLE_SORT_UNTIL_NO_CHANGE	2667	3000	2708	2667	2709	2708	2708	2667	2667	2708	2666	2666	2917	2667	2666	2708	2709	2708	2666	2667	2709	2750	2959	2708	2666
BUBBLE_SORT_UNTIL_NO_CHANGE	20667	20583	20666	20666	20666	20625	20708	20625	20583	20709	20625	20625	20750	20667	20667	20625	20667	20709	20583	20666	20666	20708	20667	20584	20708
BUBBLE_SORT_UNTIL_NO_CHANGE	27959	28042	28000	27917	28000	27958	28084	27958	28000	27958	27958	28042	28084	28042	28125	28000	27958	28000	27958	27959	28000	27959	28000	28000	27958
BUBBLE_SORT_UNTIL_NO_CHANGE	1875	1833	1875	1916	1875	1875	1917	1875	1833	1834	1875	1875	1833	1875	1833	1875	1833	1875	1833	1875	1833	1834	1875	1834	1833
BUBBLE_SORT_UNTIL_NO_CHANGE	2127167	2126958	2127583	2124667	2126959	2128375	2130167	2127709	2124292	2124083	2124583	2125209	2125417	2127708	2130166	2126083	2129084	2127542	2126208	2127958	2127083	2127333	2125834	2124791	2127208
BUBBLE_SORT_UNTIL_NO_CHANGE	2608292	2608292	2609708	2608583	2611375	2608250	2609667	2608208	2608208	2608292	2609584	2608208	2610416	2612166	2609833	2608250	2608291	2608333	2609709	2608250	2608459	2608208	2609791	2608208	2608250
BUBBLE_SORT_UNTIL_NO_CHANGE	20416	19583	18958	20583	19292	20417	18833	20417	19167	20208	18833	22167	19125	20083	19042	19750	18792	19834	18833	18833	19875	20208	19792	19458	20167
BUBBLE_SORT_UNTIL_NO_CHANGE	252355084	251882334	252395125	252279084	252425000	252912000	251798541	252293625	253801500	252102416	251950250	254573625	254240666	252880709	252403542	252178000	252709750	253640250	252859500	252267334	252804000	252817459	252691041	252482791	252167125
BUBBLE_SORT_UNTIL_NO_CHANGE	265339375	265155125	265050000	265114417	265269000	265138625	265127417	265120292	265259500	265116708	265112042	265087792	265241875	265132750	265095625	265138458	265286375	265106000	265105959	265308333	265123625	265109583	265040583	265311041	265128041
BUBBLE_SORT_UNTIL_NO_CHANGE	625	667	667	667	667	625	625	667	625	666	666	666	666	625	667	667	667	667	667	625	667	667	667	667	667
BUBBLE_SORT_UNTIL_NO_CHANGE	32292	32750	32209	32125	32584	32500	32334	32208	32083	32417	32125	32291	32291	32041	32334	32042	32250	32459	32208	32541	32500	32292	32250	32000	32541
BUBBLE_SORT_UNTIL_NO_CHANGE	36083	39083	36042	36125	36125	36042	36166	36125	36125	36083	36125	36083	36167	36083	36125	36167	36167	36083	36166	36000	36167	36084	36125	36125	36083
BUBBLE_SORT_UNTIL_NO_CHANGE	2791	2834	2791	2792	2750	2791	2792	2792	2791	2834	2750	2791	2792	2792	2791	2750	2791	2792	2792	2791	2833	2792	2750	2750	2792
BUBBLE_SORT_UNTIL_NO_CHANGE	3748667	3740958	3739500	3753750	3742291	3731667	3780791	3728084	3745000	3739958	3773708	3737583	3741542	3745042	3774625	3744083	3754125	3752083	3740542	3744250	3756625	3754208	3754959	3749458	3750875
BUBBLE_SORT_UNTIL_NO_CHANGE	3484167	3478708	3481125	3488917	3479292	3479292	3498041	3479042	3486042	3483833	3486167	3479291	3480459	3486333	3478708	3490792	3479542	3489250	3483791	3480000	3487750	3479167	3486208	3480458	3486250
BUBBLE_SORT_UNTIL_NO_CHANGE	28958	29417	31000	29417	30500	29917	30542	29750	30375	29500	31334	30208	30708	29541	31834	29666	30792	29583	29708	30625	29791	33417	29833	30458	29917
BUBBLE_SORT_UNTIL_NO_CHANGE	422907583	419151625	418085875	418400042	416983708	419338833	419128708	417760375	417659833	421200542	418971375	422503959	421921084	420285375	417458334	417076500	418052250	420987250	417491083	417313917	417734542	419784500	419346208	423103292	422150125
BUBBLE_SORT_UNTIL_NO_CHANGE	349993708	350089583	350107125	349928500	350194333	350033083	350014000	350090959	350146291	350069834	350366459	351319334	349986792	350095958	350039792	350040292	350116750	350038541	350127500	350107208	350003459	350064708	350193833	349913375	350121875
BUBBLE_SORT_UNTIL_NO_CHANGE	625	625	583	583	583	583	583	583	583	625	625	625	583	583	583	583	625	625	583	625	583	625	625	625	584
BUBBLE_SORT_UNTIL_NO_CHANGE	62417	62459	62500	62541	62584	62500	62541	62458	62417	62500	62542	62500	62458	62542	62500	62500	62542	62583	62417	62500	62625	62500	62542	62458	62500
BUBBLE_SORT_UNTIL_NO_CHANGE	59833	59875	59750	59875	59875	59875	59833	59917	59833	59917	59875	59875	59834	59791	59875	59750	59833	59958	59875	59917	59917	59833	59875	64292	59875
BUBBLE_SORT_UNTIL_NO_CHANGE	7959	7875	8042	7916	7958	8000	8084	7958	7958	7959	7916	8000	8000	7917	8041	8083	7917	7959	8041	8000	7958	7958	8000	7959	8041
BUBBLE_SORT_UNTIL_NO_CHANGE	8904459	8848000	8919791	8866417	8881625	8814667	8837416	8863875	8901917	8825375	8855042	8876875	8897458	8868792	8868500	8900166	8905292	8849041	8975125	8921958	8865084	9008625	8953459	8884250	8995000
BUBBLE_SORT_UNTIL_NO_CHANGE	6523250	6496250	6540291	6509334	6535375	6483000	6496000	6507250	6515167	6469792	6522875	6495250	6519417	6498750	6527500	6510834	6529667	6479875	6506375	6510125	6537458	6532709	6537250	6546417	6592375
BUBBLE_SORT_UNTIL_NO_CHANGE	111166	111000	111167	111083	111041	110959	111000	111125	110916	110917	111583	111042	111042	111000	111125	111042	111917	114541	111250	110916	111167	113667	111125	111000	111000
BUBBLE_SORT_UNTIL_NO_CHANGE	1158793375	1173647583	1175263625	1176755292	1185779250	1186665042	1177910041	1186263792	1182828500	1168690625	1177966500	1170044833	1171900417	1162981292	1162674125	1161128958	1165117084	1151577166	1153087084	1150634375	1153989334	1167568375	1154396792	1156125958	1150147083
BUBBLE_SORT_UNTIL_NO_CHANGE	821685375	817470583	824511792	806564750	810044375	806570375	807897959	807518375	808973000	820109041	810691792	806522167	806601041	805578625	802677250	815621542	806882667	812073666	811089792	810423500	808443583	809670584	830086958	827435416	831904084
BUBBLE_SORT_WHILE_NEEDED	667	667	625	667	667	625	666	666	625	667	667	625	667	625	666	666	625	667	625	667	667	625	666	625	625
BUBBLE_SORT_WHILE_NEEDED	9958	10041	9958	10000	10041	9958	10042	9959	9958	10042	9959	10000	9959	10042	10000	10000	9958	10000	9959	10042	10042	10000	10000	10000	10000
BUBBLE_SORT_WHILE_NEEDED	14291	14208	14209	14208	14208	14250	14208	14209	14208	14250	14208	14209	14208	14250	14208	14291	14208	14209	14208	14208	14209	14250	14209	14208	14250
BUBBLE_SORT_WHILE_NEEDED	1500	1542	1500	1583	1542	1500	1541	1541	1500	1542	1500	1542	1500	1541	1541	1500	1542	1500	1542	1542	1542	1500	1541	1541	1583
BUBBLE_SORT_WHILE_NEEDED	1040208	1037916	1039208	1037167	1043375	1041083	1045625	1038958	1040458	1040209	1037208	1040541	1038166	1038666	1047625	1040792	1036250	1040209	1036917	1039541	1035666	1046791	1036541	1042250	1042667
BUBBLE_SORT_WHILE_NEEDED	1398708	1403000	1391541	1391458	1388709	1391542	1391542	1398708	1393000	1392041	1391541	1388750	1391458	1398959	1388792	1393167	1388875	1388667	1398500	1391500	1391542	1388708	1393083	1402750	1398375
BUBBLE_SORT_WHILE_NEEDED	13917	13875	13875	13792	13792	13833	13875	13959	13792	13875	13917	13833	13833	14250	13833	14042	13875	13792	13959	14083	13875	13958	13875	13875	13958
BUBBLE_SORT_WHILE_NEEDED	102914459	103114375	102961500	102840000	102967250	102834833	103200958	102893416	103111375	102953375	103068833	102936375	102887333	102939500	102886958	103107166	103034750	103202500	103048292	103550916	103745584	102911250	102989541	102882834	103010417
BUBBLE_SORT_WHILE_NEEDED	138572208	138566125	138573541	138571666	138566750	138565542	138572917	138572292	138565334	138574916	138577500	138566417	138572333	138578000	138570875	138566375	138573875	138564917	138566375	138572958	138741625	138720292	138731542	138693583	138874291
BUBBLE_SORT_WHILE_NEEDED	2542	2542	2542	2458	2542	2542	2500	2792	2542	2542	2542	2542	2500	2542	2625	2583	2625	2916	2541	2541	2541	2583	2500	2500	2500
BUBBLE_SORT_WHILE_NEEDED	14667	14708	14708	15250	14625	14542	15125	14625	14625	14667	14625	14625	14625	14625	14708	14666	15583	14625	14583	14625	14584	14625	14625	14583	14667
BUBBLE_SORT_WHILE_NEEDED	17791	17750	17750	17750	17666	17750	17792	17750	17709	17750	17834	17750	17750	17750	17750	17834	17792	17750	17792	17709	17750	17750	17750	17750	17750
BUBBLE_SORT_WHILE_NEEDED	1833	1833	1834	1917	1875	1875	1916	1833	1834	1875	1834	1833	1833	1834	1833	1875	1833	1875	1833	1833	1834	1833	1833	1834	1833
BUBBLE_SORT_WHILE_NEEDED	1485750	1480791	1485125	1485833	1480542	1467292	1471417	1474959	1475750	1483000	1472083	1476750	1476583	1475250	1482333	1473375	1478041	1476875	1469875	1481209	1477125	1470084	1485250	1478709	1474375
BUBBLE_SORT_WHILE_NEEDED	1723125	1721584	1721583	1721625	1721625	1721625	1723041	1721583	1721709	1721583	1721541	1721625	1723042	1721542	1721583	1721583	1721584	1722917	1724708	1721541	1721750	1721583	1721541	1722833	1721709
BUBBLE_SORT_WHILE_NEEDED	16417	16375	16375	16417	16375	16209	16291	16333	16459	16625	16709	16333	16250	16333	16250	16333	16333	16334	16291	16250	16292	16333	16417	16333	16375
BUBBLE_SORT_WHILE_NEEDED	217062625	216562833	215694458	215716584	217643500	216821917	216290750	214669834	216332292	215782750	216322625	215502542	215580792	215670958	214612667	217084000	215974041	215394417	214612750	216074917	216271792	214999292	215872000	214401125	215699875
BUBBLE_SORT_WHILE_NEEDED	175223333	175220166	175349709	175210083	175204292	175249917	175216959	175402625	175212125	175234542	175209750	175212083	175228000	175333542	175162542	175204125	175218792	175236833	175259833	175343833	175225625	175199333	175203417	175227500	175222041
BUBBLE_SORT_WHILE_NEEDED	708	666	666	708	709	667	667	875	709	667	667	667	708	666	666	667	667	667	667	667	667	667	666	666	708
BUBBLE_SORT_WHILE_NEEDED	19208	19209	19167	19208	19208	19167	19209	19208	19250	19166	19166	19208	19209	19167	19166	19166	19250	19167	19208	19208	19250	19125	19208	19166	19208
BUBBLE_SORT_WHILE_NEEDED	22542	22583	22583	22542	22583	22583	22542	22625	22541	22542	22583	22542	22584	22583	22583	22584	22541	22500	22583	22541	22542	22542	22541	22542	22584
BUBBLE_SORT_WHILE_NEEDED	2708	2834	2709	2708	2708	2750	2708	2708	2709	2709	2709	2708	2708	2708	2708	2709	2709	2708	2708	2708	2792	2709	2708	2708	2750
BUBBLE_SORT_WHILE_NEEDED	1832834	1837417	1828417	1835541	1845084	1833375	1834416	1827834	1841250	1835583	1830333	1834042	1839167	1828791	1830709	1832625	1829416	1841334	1832916	1832834	1833542	1841083	1834500	1834584	1826291
BUBBLE_SORT_WHILE_NEEDED	2255167	2255875	2262375	2258000	2262541	2256334	2262000	2255125	2259709	2253125	2255083	2252333	2257042	2262292	2255083	2255167	2256875	2265583	2255083	2255084	2262166	2259000	2255084	2255666	2262125
BUBBLE_SORT_WHILE_NEEDED	26125	26167	26125	26042	26166	26042	26167	26000	26042	26000	26250	26416	26167	26083	26125	26083	26125	26167	26083	26041	26125	26167	26125	26125	26459
BUBBLE_SORT_WHILE_NEEDED	245950084	247048416	244155084	244236000	246483708	247781417	245343292	243984750	246943250	245951334	248056375	247522500	245952000	245451709	244983083	246490542	246441959	246780917	245116500	241730625	243252416	242999042	241445167	240597875	244596583
BUBBLE_SORT_WHILE_NEEDED	225298584	225191959	225209375	225375709	225196292	225202750	225215083	225408708	225370833	225228791	225322209	225225750	225361083	225309458	225214042	225930917	225476166	225250500	225195000	225212459	225465625	225350125	225192750	225343375	225182625
BUBBLE_SORT_WHILE_NEEDED	583	625	625	584	584	625	583	625	583	583	583	625	583	583	583	625	584	625	583	583	666	625	583	583	583
BUBBLE_SORT_WHILE_NEEDED	37750	37834	37750	37916	37792	37791	37834	37875	37833	44708	37917	37958	37792	37792	37875	37875	40541	37833	37834	37833	37708	37833	37833	37875	37792
BUBBLE_SORT_WHILE_NEEDED	33916	33959	33959	34000	33916	34000	33959	33958	33958	34042	34000	34000	34000	33958	33917	33959	33916	34250	34000	33958	33959	33834	33958	34000	34000
BUBBLE_SORT_WHILE_NEEDED	7917	7708	7917	7958	7791	7792	8167	7791	7667	7875	7791	8084	7750	7875	8166	7709	8084	7667	7916	7708	7917	7958	7708	7708	7708
BUBBLE_SORT_WHILE_NEEDED	5141250	5127916	5145625	5121417	5141875	5129958	5151042	5118750	5143041	5122334	5138083	5122375	5138833	5112000	5184042	5134042	5132416	5127417	5150250	5121500	5131375	5124167	5137083	5131167	5125375
BUBBLE_SORT_WHILE_NEEDED	3309375	3296708	3314333	3299917	3301292	3304959	3305666	3307167	3306625	3298167	3309916	3303500	3311334	3301500	3327625	3313250	3300791	3296250	3303250	3308084	3303666	3296417	3310042	3299791	3307000
BUBBLE_SORT_WHILE_NEEDED	115375	115333	115458	115500	118125	115458	115291	115500	115500	115292	257209	117000	119334	116041	116042	115833	116084	116333	116000	116084	118541	115875	116083	121125	117208
BUBBLE_SORT_WHILE_NEEDED	614323166	615809542	614820333	614465250	615946625	617332292	616799167	616343375	616300500	615612125	617021416	615473167	614980917	615350708	615487875	617632875	617981792	618787000	619290041	618799500	618252833	618545625	617291708	616986083	616927209
BUBBLE_SORT_WHILE_NEEDED	328597000	328603500	329066208	327676042	328748792	329314916	328619875	328300042	333149208	334848583	335061709	335237708	334373958	334300417	334294250	335409833	335256542	335201792	335439666	335354583	335374542	335358667	334837208	334241833	336352209

7.4 Data Sorting vs Array Size

Data_Ordering	Array_Size	Mean	N	Std. Deviation
ASC	100	7.3834E+3	12	1.1807E+4
	1000	7.4524E+5	12	1.4278E+6
	10000	1.1488E+8	12	2.7164E+8
	Total	3.8543E+7	36	1.6183E+8
DESC	100	2.9638E+4	12	1.3408E+4
	1000	2.9442E+6	12	1.4945E+6
	10000	3.2296E+8	12	2.0592E+8
	Total	1.0865E+8	36	1.9223E+8
RANDOM	100	2.7501E+4	12	1.6699E+4
	1000	3.2956E+6	12	2.5980E+6
	10000	4.1438E+8	12	3.5180E+8
	Total	1.3923E+8	36	2.7899E+8
Total	100	2.1507E+4	36	1.7069E+4
	1000	2.3283E+6	36	2.1850E+6
	10000	2.8407E+8	36	3.0261E+8
	Total	9.5474E+7	108	2.1887E+8

Figure 8: Data sorting vs array size. Report containing the mean values of processing time in nanoseconds for given data-input as arrays sorted by ascending (ASC), descending (DESC) or random(RANDOM). The array size is included with number of elements varying from 100, 1000 or 10000 in each case.

7.5 Data Sorting vs Algorithm

Data_Ordering	Algorithm	Mean	N	Std. Deviation
ASC	BUBBLE_SORT_PASS_PER_ITEM	1.1559E+8	12	2.7131E+8
	BUBBLE_SORT_UNTIL_NO_CHANGE	1.6564E+4	12	3.1372E+4
	BUBBLE_SORT_WHILE_NEEDED	1.6320E+4	12	3.3931E+4
	Total	3.8543E+7	36	1.6183E+8
DESC	BUBBLE_SORT_PASS_PER_ITEM	1.1408E+8	12	2.0153E+8
	BUBBLE_SORT_UNTIL_NO_CHANGE	1.3852E+8	12	2.4754E+8
	BUBBLE_SORT_WHILE_NEEDED	7.3344E+7	12	1.1541E+8
	Total	1.0865E+8	36	1.9223E+8
RANDOM	BUBBLE_SORT_PASS_PER_ITEM	1.4866E+8	12	3.0563E+8
	BUBBLE_SORT_UNTIL_NO_CHANGE	1.6986E+8	12	3.4191E+8
	BUBBLE_SORT_WHILE_NEEDED	9.9188E+7	12	1.8569E+8
	Total	1.3923E+8	36	2.7899E+8
Total	BUBBLE_SORT_PASS_PER_ITEM	1.2611E+8	36	2.5596E+8
	BUBBLE_SORT_UNTIL_NO_CHANGE	1.0280E+8	36	2.4819E+8
	BUBBLE_SORT_WHILE_NEEDED	5.7516E+7	36	1.2976E+8
	Total	9.5474E+7	108	2.1887E+8

Figure 9: Data sorting vs Algorithm. Report containing the mean values of processing time in nanoseconds for given data-input as arrays sorted by ascending (ASC), descending (DESC) or random(RANDOM). The array size is included with number of elements varying from 100, 1000 or 10000 in each case.