УДК 004.04;004.043 DOI: 10.32626/2307-4507.2022-28.76-80

Tetiana Pylypiuk[1], Valeria Sukmaniuk[2]

Kamianets-Podilskyi National Ivan Ohiienko University

e-mail: [1]pylypyuk.tetiana@kpnu.edu.ua, [2]kn1b20.sukmaniuk@kpnu.edu.ua; ORCID: [1]0000-0002-4676-9830

STUDY OF ALGORITHMS FOR SORTING INFORMATION IN DIFFERENT TYPES ARRAYS

The article is devoted to the research of sorting algorithms for different types of arrays orderliness: unordered, almost ordered, reverse-ordered.

The authors formulated the sorting problem and presented its mathematical basis.

The authors considered the several popular algorithms for sorting arrays of information and their modifications: "bubble" sorting, "odd-even" sorting, "comb" sorting, insertions, inclusion, selection; provided a brief description of the selected algorithms for a better understanding of the principle of their work; conducted a study of the applicability of these algorithms for different types of arrays orderliness and performed their comparative analysis in execution time, number of mileages and iterations.

For a better comparison, different arrays dimensions were used: 10, 100, 500, 1000, 2000, 5000 and 10000 elements. The arrays were filled with random numbers, which were generated by the appropriate function, which allowed each of the algorithms to be evaluated fairly. The traditional sorting criterion was chosen – by growth.

The authors also carried out a comparative analysis of the applicability of one or another algorithm for different types of the initial (input) data orderliness. Relevant conclusions have been made.

Key words: array, algorithm, orderliness, random numbers, execution time, mileage, iteration, sorting.

Introduction. It is often necessary to work with information in the form of an arbitrary set of values, that is fields, in practice. Most often, the information is presented in an array. An array is a finite named sequence of values of the same type, which differ by a sequence number (array index). Array data processing is simplified if the data is ordered (sorted).

There are dozens of sorting algorithms today. It`s impossible to identify one perfect algorithm among all such algorithms because:

- different algorithms are optimal for different data sets and data types;
- some algorithms are easy to implement and are well suited for explaining sorting principles, others for practical implementation;
- some algorithms are better used for process large data sets, others have proven themselves better for small volumes of information;
- some algorithms are better to use for unordered data, others for almost ordered or reverse ordered data.

When we must choose one or another sorting algorithm, it`s also necessary to consider it optimization in processor cycles terms and speed, which is extremely important for processing large volumes of information.

The authors considered several sorting algorithms and conducted a comparative analysis of them in terms of speed, number of runs and iterations in this article. An analysis of the applicability of one or another algorithm for different types of initial (input) data ordering was also carried out.

The main part. For the most general case, the sorting poblem is formulated as follows: there is some unordered input set of keys, and it is necessary to obtain a set of this keys, sorted by increasing or decreasing order [1].

Let`s formulate the mathematical basis of sorting.

There is a sequence of n numbers $(a_1, a_2, \ldots, a_n)$ at the beginning of the algorithm (algorithm input).

The task is: to permute the input sequence in such a way that $a_{\pi(1)} \le a_{\pi(2)} \le \ldots \le a_{\pi(n)}$ (or $a_{\pi(1)} \ge a_{\pi(2)} \ge \ldots \ge a_{\pi(n)}$). Here π is the permutation of the sequence of numbers (algorithm output).

The input sequence is most often represented as an n-element array, although it can also have another representation, for example, as a linked list.

To implement sorting algorithms of various types arrays ordering let`s give a brief description of the selected algorithms to understand the principle of their operation. The program codes for the implementation of the main algorithms can be found, for example, in [2, c. 47-55; 3, c. 25-49].

1) *"Bubble" sort* (or exchange sort) is one of the simplest sorting algorithms. The principle of operation is to compare neighbouring elements, viewing them from left to right. In several passes, the algorithm sorts the array by increasing or decreasing key values, depending on the specified criterion, swapping the elements if they are not positioned correctly. In the second pass, these operations are performed on the elements starting from the first and ending with the (n-1)th element, in the third – from the first to the (n-2)th element, etc. Sorting of the array will be completed when there is no permutation of the array elements during the pass.

2) *Cocktail sort* algorithm is one of the varieties of "bubble" sorting, but with one difference that makes the work much easier. This difference is in the algorithm direction. The "bubble" algorithm proceeds only from left to right, starting over when the array runs out. Cocktail sort algorithm works in two directions. Viewing the array also starts from left to right, swapping elements if it`s necessary, but if is the end of the array, viewing does not start over, but reverses from right to left. So it speeds things up a bit.

3) An *odd-even sort,* also known as brick sort, is a relatively simple sorting algorithm. It is often compared to a "bubble" as it has many similar characteristics. The algorithm, just like the "bubble", reads data from left to right, swapping elements to the appropriate criterion, but has a larger step. The "bubble" compares the first element with the second, the second with the third, etc. In the odd-even sorting algorithm, the first pass compares an odd element with the adjacent even element (the first with the second, the third with the fourth, the fifth with the sixth,

etc.), in the second pass the even element with the next odd one (the second with third, fourth and fifth, etc.). Then the process starts over, odd with even, then even with odd, hence the name of the odd-even algorithm. The sorting algorithm ends when no change has occurred in two such passes. This means that the array is sorted.

4) *Comb sort* algorithm is a modification of the "bubble" algorithm, which was developed in 1980 by Wlodek Doboshevych and popularized in 1991 thanks to the Byte Magazine. The main idea is to prevent the situation where small values remain at the end of the array for a long time if you want to sort the array in ascending order. The essence of the algorithm is to take a sufficiently large distance between elements and gradually reduce it. For example, for a better understanding: in the first run it will be a comparison of the first and last element, in the second – the first and penultimate, second and last, etc. until the distance becomes such that neighbouring elements are compared and no permutation occurred during the algorithm run. It`s advised not to take any value of the gap between the elements, but taking into account a certain value, called the reduction factor, which is equal to 1.247. This is the optimal value that was found through many experiments with different values. At first, we need to take a step that will be equal to the rounded value of the array divided by the reduction factor. On the next run, we divide our previous step again by the reduction factor and it will be a new step. This will continue until the step becomes equal 1, that`s mean neighbouring elements.

5) *Selection sort* is a simple sorting algorithm based on insertions, that is, a certain element (the smallest or largest element of the array) is selected and inserted into the appropriate place in the array (at the beginning or at the end), which depends on the specified criterion. This continues until the array is sorted.

6) *Insertion sort* is a comparison-based algorithm. The essence of the algorithm is that at first we consider that the first element is in its place. We compare the second with the first and change places as necessary. If not, then we believe that it is in its place. We compare the next one with the correct elements and include it in the right place in the array. So the algorithm will work until the array runs out. The result is a sorted array.

The authors chose the sorting algorithms described above for researching arrays of different data ordering types: unordered, almost ordered, and reverse-ordered.

First of all, let's note a few points.

1. Since the speed (time) and the number of runs and iterations (permutations) of each of the algorithms are investigated, all algorithms were partially modified by several variables for such tracking. The traditional sorting criterion was chosen – by growth.

2. For better comparison, arrays of different dimensions were used: 10, 100, 500, 1000, 2000, 5000 and 10000 elements.

3. After each program code execution the program cache was cleaned to prevent inaccurate indicators.

4. The arrays were filled with random numbers generated using the corresponding function rand(). The same sequence of randomly generated numbers was generated every time with the help of this function. This made it possible to fairly evaluate each of the methods.

Let`s present the results of the research.

1. Let's consider the indicators of an unordered array and analyze them.

Indicators of time, number of runs and iterations of "bubble", cocktail sort , "odd-even", "comb", selection, insertion sorting algorithms for unordered array are presented in *tables 1-6*, respectively.

Table 1

"Bubble" sort algorithm, unordered array indicators

Number of elements / Measurement results	10	100	500	1000	2000	5000	10000
Number of runs	10	100	500	1000	2000	5000	10000
Time (in sec)	0.001	0.0011	0.0015	0.0027	0.0071	0.0373	0.1516
Number of iterations	13	2360	62 420	248 145	1 007 742	6 175 385	24 952 888

Table 2

Cocktail sort algorithm, unordered array indicators

Number of elements / Measurement results	10	100	500	1000	2000	5000	10000
Number of runs	5	50	250	500	1000	2500	5000
Time (in sec)	0.0008	0.0009	0.0014	0.0025	0.0068	0.0359	0.1439
Number of iterations	13	2360	62 420	248 145	1 007 742	6 175 385	24 952 888

Table 3

"Odd-even" sort algorithm, unordered array indicators

Number of elements / Measurement results	10	100	500	1000	2000	5000	10000
Number of runs	4	46	247	489	979	2473	4989
Time (in sec)	0.0009	0.0011	0.0013	0.0024	0.0049	0.0246	0.094
Number of iterations	13	2360	62 420	248 145	1 007 742	6 175 385	24 952 888

Table 4

Comb sort algorithm, unordered array indicators

Measurement results \ Number of elements	10	100	500	1000	2000	5000	10000
Number of runs	7	17	24	28	31	35	39
Time (in sec)	0.0008	0.0009	0.001	0.001	0.0012	0.0015	0.0018
Number of iterations	7	246	1 809	4 369	9 746	27 189	59 136

Table 5

Selection sort algorithm, unordered array indicators

Measurement results \ Number of elements	10	100	500	1000	2000	5000	10000
Number of runs	9	99	499	999	1999	4999	9999
Time (in sec)	0.0009	0.001	0.0011	0.0016	0.0032	0.0147	0.0533
Number of iterations	7	92	488	994	1 993	4 987	9992

Table 6

Insertion sort algorithm, unordered array indicators

Measurement results \ Number of elements	10	100	500	1000	2000	5000	10000
Number of runs	9	99	499	999	1999	4999	9999
Time (in sec)	0.0009	0.001	0.0011	0.0015	0.0025	0.0102	0.0372
Number of iterations	13	2360	62 420	248 145	1 007 742	6 175 385	24 952 888

The operation of various sorting algorithms for an unordered array type is shown in *Fig. 1* (number of runs) and *Fig. 2* (time).

From *Fig. 1* we can see that three algorithms out of six performed equally with a large number of runs. These are: "bubble" sorting, selection and insertion sort algorithms. Two algorithms out of six: cocktail sort and "odd-even" sorting performed moderately. The least number of runs is observed in the comb sorting algorithm.

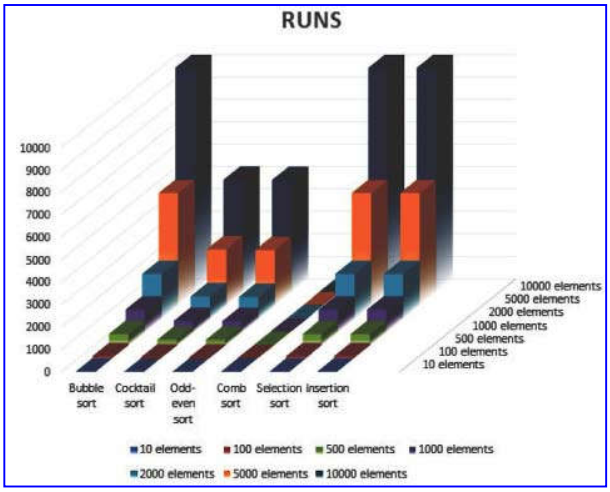

Fig. 1. *Indicators of runs for different sorting methods of unordered array*

From *Fig. 2* we visually observe the gradation of time to sort an increasing number of elements. We can see that "bubble" and cocktail sort on large arrays are very slow, while comb sort takes almost no time at all.

The four algorithms showed themselves exactly the same according to the iterations number. These are: "bubble" sorting, cocktail sort algorithm, "odd-even" and insertion. Fewer iterations are needed for comb sorting. The smallest number uses the selection sort algorithm.

2. Let's consider the indicators of an almost ordered array and perform their analysis.

An almost ordered array is an array that already has sorted elements. Therefore, element generation was divided into two cycles. The first composed consecutive numbers in half of the array, the other half – randomly generated numbers.

The relevant indicators of the time, number of runs and iterations of the sorting algorithms by "bubble", cocktail sort, "odd-even", "comb", selection and insertion for an almost ordered array are obtained and entered in the corresponding tables.

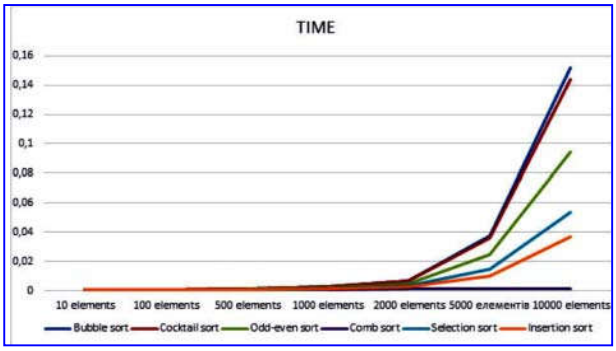

Fig. 2. *Indicators of times for different sorting methods of unordered array*

In order not to burden the article visually, we don't present such tables, but we present the results using appropriate diagrams (*Fig. 3-4*).

We can see from the diagram in *Fig. 3*, that there are no changes in the number of runs, although half of the elements are sorted in this type of array.

In *Fig. 4* we visually observe the gradation of time with the help of a graph in order to sort more and more elements. Compared to the previous graph (*Fig. 2*), we can see that the cocktail sort and insertion algorithms take half the time to sort an almost ordered array.

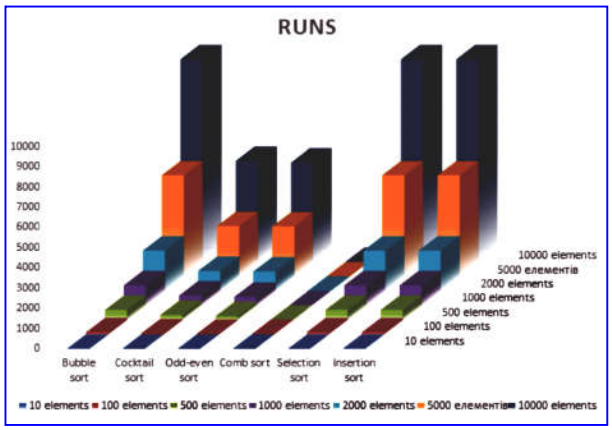

Fig. 3. *Indicators of runs for different sorting methods of almost ordered array*

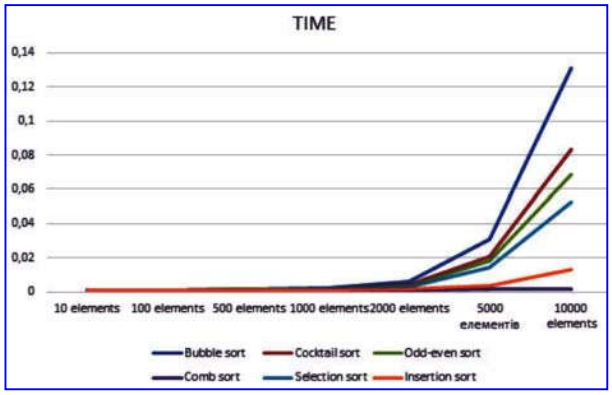

Fig. 4. *Indicators of times for different sorting methods of almost ordered array*

According to the number of iterations, the algorithms showed themselves in the same way as for the ordered array. Although in the four algorithms that act in the same way, three times less permutations are made.

3. Let's consider the indicators of reverse-ordered array and perform their analysis. Reverse-ordered array – array sorted in reverse. For some sorting methods, this is the worst case, because more permutations will have to be done.

The size of the array was used to generate the elements, which gradually decreased depending on its index.

The relevant indicators of the time, number of runs and iterations of the sorting algorithms by "bubble", cocktail sort, "odd-even", "comb", selection and insertion for reverse-ordered array are obtained and entered in the corresponding tables.

In order not to burden the article visually, we don't present such tables, but we present the results using appropriate diagrams (*Fig. 5-6*).

We can see the number of runs that were needed to sort the reverse-ordered array on the diagram (*Fig. 5*). We do not observe significant changes.

In *Fig. 6* we visually observe the gradation of time with the help of a graph in order to sort more and more elements. We can note that each of the sorting algorithms takes more time to implement, except for the "comb" algorithm, which remained unchanged. It is also interesting that for unordered and almost ordered arrays, the cocktail sort and insertion sort algorithms took more time than "bubble" and selection sort, respectively.

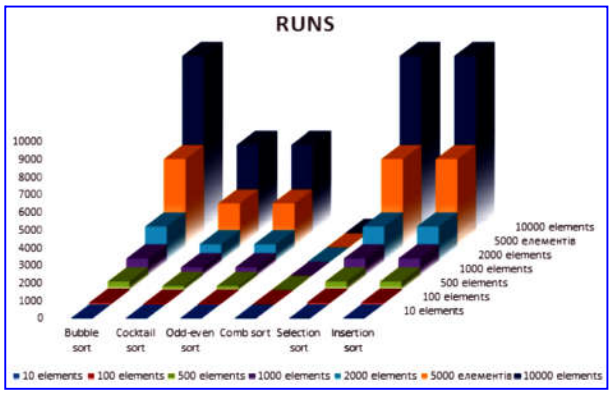

Fig. 5. *Indicators of runs for different sorting methods of reverse-ordered array*

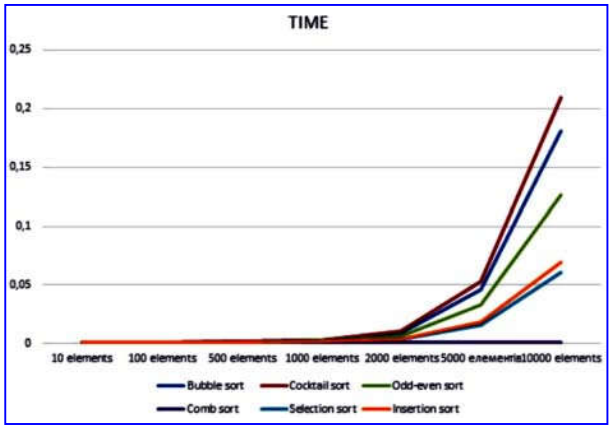

Fig. 6. *Indicators of times for different sorting methods of reverse-ordered array*

As for iterations, almost every method increased the number of permutations. The number of iterations in the selection sort algorithm increased insignificantly. In the "comb" sorting algorithm, the number of permutations decreased by about 2.7 times. The number of iterations in the remaining four algorithms is twice as large.

As for the run analysis, we are talking about comparing how many repetitions a particular algorithm needed to sort the array.

We count the run as repeating the operation of reading elements from the first element. We also take into account a control run that checks whether the array is sorted.

The "bubble" sorting algorithm needed the largest number of runs. This is natural, since only one element is put in its place in one run. Different array ordering type does not matter.

Following the "bubble", the sorting algorithms by selection and insertion sort have the same number of runs. Only one less than in the "bubble" algorithm. This is because these algorithms are based on finding an element that is put into place in one pass. And we believe at the beginning, that one element is already in its place without running.

The cocktail sort algorithm needs half as many runs as its predecessors. Such indicators are because in one run the algorithm read array twice: from beginning to end and from end to beginning.

The "odd-even" sorting algorithm is one of those algorithms for which the number of runs depends on the ordering type. The lowest number of runs is observed for almost ordered array.

The least number of runs is observed in the "comb" sorting algorithm. The algorithm showed the same number of runs for all three types of arrays ordering, but for the revers-ordered array, starting with two thousand elements, one run less was performed. The algorithm performed 1.4 views per run with ten elements and 256.4 views with ten thousand elements.

Thus, the best in terms of the number of runs is the "comb" sorting algorithm among all the considered sorting algorithms.

As for the time analysis. It is important to note that time is measured in seconds and that the values are too small to show graphically. Because of this, we will compare them using the arithmetic average of each method.

The "bubble" sorting algorithm turned out to be the slowest of all algorithms. The best performance of the execution time is shown in the almost ordered array – 0.0247 sec, the average indicator is in unordered array with value 0.0289 sec, and the worst indicator is in revers-ordered array – 0.0346 sec.

Next, according to the indicators, the cocktail sort algorithm. Although it shows the worst result – 0.04 sec in reverse-ordered array, but in almost ordered array the average sorting execution time is 0.0162 sec. In an unordered array, the indicator does not differ significantly from the "bubble" sorting algorithm – 0.0275 sec. This algorithm is not stable with respect to different types of arrays ordering.

As for the "odd-even" sorting algorithm, the best result was obtained for an almost ordered array with an indicator of 0.0138 sec. The worst – 0.0244 sec – for the reverse- ordered array.

One of the most stable was the selection sorting algorithm. We got the worst result for the reverse-ordered array (0.0132 sec), the best is for an almost ordered array (0.106 sec).

The insertion sort algorithm was found to be faster than the selection sort algorithm for different array types. For an unordered array – 0.0078 sec, for almost orderly – 0.0032 sec. The worst indicator is in the reverse-ordered array – 0.0138 sec.

The "comb" sorting algorithm turned out to be the most stable and the fastest. It has a stable indicator (0.0012 sec) in two types of array ordering: almost ordered and unordered arrays. The best time for a reverse-ordered array is 0.001 sec.

As for iterations, the four algorithms performed exactly the same for all types of array ordering. These are sorting algorithms by "bubble", cocktail sort algorithm, "odd-even", selection and insertion. The same number of permutations is performed in these algorithms, and this number is one of the largest.

The "comb" sorting algorithm performed much better than other algorithms, but the selection sorting algorithm has the lowest number of iterations.

Conclusions. A comparative analysis of six different sorting algorithms was done: "bubble", cocktail sort algorithm, "odd-even", "comb", selection and insertion for different types of arrays ordering: unordered, almost ordered, reverse-ordered.

By the volume of the software code, these algorithms can be divided from the smallest to the largest as follows: sorting by "bubble" and insertion, cocktail sort algorithm and selection, "odd-even", "comb". The most self-explanatory algorithms for beginners are, of course, "bubble" and cocktail sort algorithm, and "odd-even". They are simple to implement and easy to master the principles of sorting. Insertion and selection sorting methods can be used to explain the principle of element memorization. These methods are also optimal for small sizes arrays. The most difficult to understand is the "comb" method, because it uses a step variable that must be calculated. But this ordering algorithm showed itself in the best way: it does not perform many runs, that`s why it is the fastest of all considered algorithms (average speed is 0.0012 sec). This algorithm is optimal for any array type with a large number of elements.

References:

1. Wirt Niklaus. Algorithms and data structures: trans. with English. Kyiv: DMK Press, 2016. 272 p.
2. Krenevich A.P. Algorithms and data structures. Textbook. Kyiv: VOC "Kyiv University", 2021. 200 p. URL: http://www.mechmat.univ.kiev.ua/wp-content/uploads/2021/09/pidruchnyk-alhorytmy-i-struktury-danykh.pdf
3. Kuzmenko I.M., Datsyuk O.A. Basic algorithms and data structures : teaching. manual. Kyiv: Igor Sikorsky KPI, 2022. 137 p. URL: https://ela.kpi.ua/bitstream/123456789/48256/1/Bazovi.pdf

Т. М. Пилипюк, В. С. Сукманюк

Кам'янець-Подільський національний університет імені Івана Огієнка

ВИВЧЕННЯ АЛГОРИТМІВ СОРТУВАННЯ ІНФОРМАЦІЇ В МАСИВАХ РІЗНИХ ТИПІВ

Стаття присвячена дослідженню алгоритмів сортування для різного типу впорядкованості масивів: невпорядкованих, майже впорядкованих, обернено впорядкованих.

У статті сформульовано постановку задачі сортування та подано її математичну основу.

Автори розглянули декілька популярних алгоритмів сортування масивів інформації та їх модифікацій: сортування «бульбашкою», сортування «парне-непарне», сортування «гребінцем», вставками, включенням, вибором; подали коротку характеристику обраних алгоритмів для кращого розуміння принципу їх роботи; провели дослідження застосовності цих алгоритмів для різного типу впорядкованості масивів та здійснили їх порівняльний аналіз за швидкодією, кількістю пробігів та ітерацій.

Для кращого порівняння використано різну розмірність масивів: 10, 100, 500, 1000, 2000, 5000 та 10000 елементів. Масиви заповнювалися випадковими числами, які генерувалися за допомогою відповідної функції, що дозволило справедливо оцінювати кожний з алгоритмів. Критерій сортування обрано традиційний – за зростанням.

Також здійснено порівняльний аналіз застосовності того чи іншого алгоритму для різного типу впорядкованості початкових (вхідних) даних. Зроблено відповідні висновки.

Ключові слова: масив, алгоритм, впорядкованість, випадкові числа, швидкодія, пробіг, ітерація, сортування.

Отримано: 19.11.2022