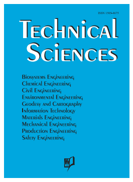
Technical Sciences, 2018, 21(1), 19–35

UNIWERSYTET WARMIŃSKO-MAZURSKI W OLSZTYNIE

Quarterly peer-reviewed scientific journal
ISSN 1505-4675
e-ISSN 2083-4527
TECHNICAL SCIENCES
Homepage: www.uwm.edu.pl/techsci/



PERFORMANCE TESTS ON MERGE SORT AND RECURSIVE MERGE SORT FOR BIG DATA PROCESSING

Zbigniew Marszałek

Institute of Mathematics
Silesian University of Technology

Received 22 August 2016, accepted 13 September 2017, available online 6 November 2017.

K e y w o r d s: Software, Dependability, Workflow, Analysis of Computer Algorithms, Big Data, Merge Sort.

A b s t r a c t

Merge sort algorithm is widely used in databases to organize and search for information. In the work the author describes some newly proposed not recursive version of the merge sort algorithm for large data sets. Tests of the algorithm confirm the effectiveness of the method and the stability of the proposed version.

Introduction

In recent years we have noted very fast development of computers and their applications. In storage, management and processing the amount of data is increasing. Dedicated algorithms used in the processing of large information volumes require an optimal strategy for classification (ARTIEMJEW et al. 2016, WILD et al. 2016). Similarly new possibilities for the development in methodological approaches for data handling help on new improvements in data systems (MLECZKO et al. 2016). By the use of intelligent solutions it is possible to use even incomplete data for information retrieval (NOWICKI et al. 2016, ŻMUDZIŃSKI et al. 2017). The new architectures of data base systems support

Correspondence: Zbigniew Marszałek, Instytut Matematyki, Politechnika Śląska, ul. Kaszubska 23, 44-100 Gliwice, e-mail: Zbigniew.Marszalek@polsl.pl

various methods of information retrieval and processing (GABRYEL 2016, GRYCUK et al. 2017). The information and models help to build the systems that support people in daily routine (DAMASEVICIU et al. 2016, DAMASEVICIU et al. 2016). However still one of the important issues is the order of the data. A special role is played here by the sort methods of the large data sets (GABRYEL et al. 2016, MARSZAŁEK et al. 2014), which enable to create indexes needed to search and organize data sets in the desired way. In the recent year we can see various approaches to analyze sorting methods. Sorting methods are reported to be very efficient in NOSQL data systems, where instead of complex solutions we use efficient sorting algorithms (WOŹNIAK et al. 2016, WOŹNIAK et al. 2013, WILD et al. 2016, MARSZAŁEK 2017).

Related works

Collations play a special role in the databases when searching for information. Methods of sorting are developed in various versions for multiple and standard architectures to efficiently compare the data. There were many tests on efficiency of the quick sort (AUMULLER et al. 2013, AUMULLER et al. 2016, NEBEL et al. 2016, WOŹNIAK et al. 2013, WILD et al. 2016), from which we can see that this sorting algorithm although fast still has important drawbacks. Therefore we started to search for other possibilities to improve sorting methods to be efficient but still fast enough for big data systems. Various versions of the heap sort appeared to be a good solution (MARSZAŁEK et al. 2014, MARSZAŁEK 2017, WENGER et al. 1989, WOŹNIAK et al. 2013). Merge sort was also analyzed and discussed in case of efficiency for data systems (MARSZAŁEK et al. 2015, MARSZAŁEK et al. 2014, WOŹNIAK et al. 2013). Even some new methods as derivatives from these classic approaches were composed (MARSZAŁEK 2016, WENGER et al. 1989). However, there is still an open question whether the use of recursive methods produces better results than direct programming methods.

In this work is presented not recursive version of this sorting algorithm and additionally to prove efficiency a comparison with the traditional recursive algorithm is given. Experimental tests allow us to find the best solution with the smallest possible complexity. The tests show the effectiveness and stability of the presented method.

Large databases and collected information

Currently in the database are collected enormous amounts of information from different sources and for different areas. This information is serialized and classified. Sample organization of NoSQL database is shown in Figure 1. A variety of statistics in order to improve production processes and decision-making creates a possibility for development in the research on sorting methods with a view to their improvement. For serializing information in NOSQL databases are used stable algorithms of low complexity. To compare the algorithms we run tests comparing used resources by the usage of CPU (Central Processing Unit). In this way, we can compare the performance of algorithms and determine their suitability for use in the analysis of large data sets.

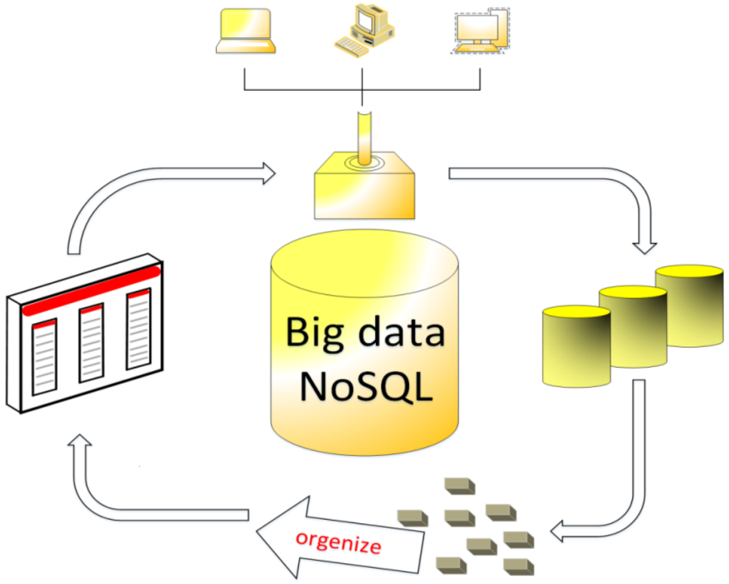


Fig. 1. Organization of NoSQL database

Statistical studies of algorithms

The surveys we run are performed on 100 tests for each desired dimension of the sample item. The statistical tests were done by the use of methods such as in systems (MARSZAŁEK 2017, MARSZAŁEK 2016, NOWICKI et al. 2016). A statistical average of n – element set of samples a_1, ... a_n is defined by the formula

$$\bar{a} = \frac{\sum_{i=1}^{n} \bar{a}_i}{n}.$$

The standard deviation is defined by the formula

$$\bar{a} = \sqrt{\frac{\sum_{i=1}^{n}(a_1 - \bar{a}_i)}{n-1}}.$$

where n is the number of elements in the sample, a is value of the random variable in the sample, $\bar{a}$ is the arithmetic mean of the sample. The standard deviation is characterized by the dispersion between time sorting. If we can determine the worst-case time sorting then its magnitude is the same as the average time of sort. We can say that statistical studies reflect the behavior of the algorithm in practice.

Another important factor in statistical surveys is the coefficient of variation presenting the stability of the algorithm. It is determined by formula

$$V = \frac{\sigma}{\bar{a}}.$$

Where σ is standard deviation of random variables in tests, $\bar{a}$ is the arithmetic mean of the sample. The analysis methods for sorting sets of random samples were taken for 10, 100, 1,000, 10,000, 100,000, 1,000,000 and 10,000,000 elements. The results are presented in graphs.

Merge sort

One of the most appropriate method for serializing information in database NoSQL is the merge sort algorithm. In the literature we can find many versions of this algorithm. The work shows a comparison of the recursive method with direct method presented in (WOŹNIAK et al. 2013).

No recursive merge sort algorithm

Let us suppose we have a sequence of numbers $a_0, a_1, ..., a_{n-1}$. We can sort it by dividing into subsequences then merging sorted substrings. Double merge procedure in the first step begins with comparison of pairs of input sequence. In this way as a result of the first step, we obtain two-component stacks. In second step, we merge received from previous step strings. As a result of this operation we obtain stacks containing doubled number of elements. We merge until we have only one stack. If initial sequence contains an odd number of items we rewrite last element until last step in the algorithm. In the last step we merge it and get completely sorted input. Method on input receives two sorted in previous step sequences $x_0 \le x_1 \le ... \le x_{m-1}$ and $y_0 \le y_1 \le ... \le y_{m-1}$. It returns sorted sequence $z_0 \le z_1 \le ... \le z_{2m-1}$. We merge two sequences X and Y having comparisons, where *2m* is number of elements in X and Y. Merge sort algorithm uses two components.

Figure 2 shows merging two sorted sequences 6, 8 and -1, -7. We compare first elements. Element -1 is smallest therefore it goes to output string.

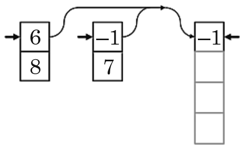


Fig. 2. Comparison and selection of minimum in first step

We compare other items placed on top of the stack, as shown in Figure 3. In this case, smallest element is found in first string. Thus, it goes to the output sequence.

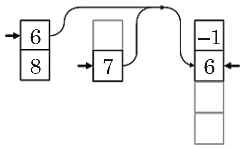


Fig. 3. Comparison and selection of minimum in second step

Third step is shown in Figure 4. Smaller element of 8 and -7 goes to output sequence.

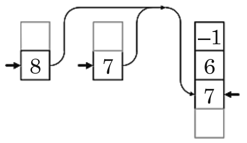


Fig. 4. Comparison and selection of minimum in third step

Finally biggest element goes to output sequence, as show in Figure 5.

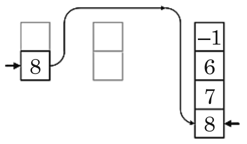


Fig. 5. Comparison and selection of minimum in forth step

No recursive merge sort merges elements in pairs without division. First, elements are merged in pairs, then in fours and so on. Continuing to do so, in each step we get organized doubled stacks. If n is not power of two, merging continues leaving at the end odd element. It will be merged in last step, as show in Figure 6.

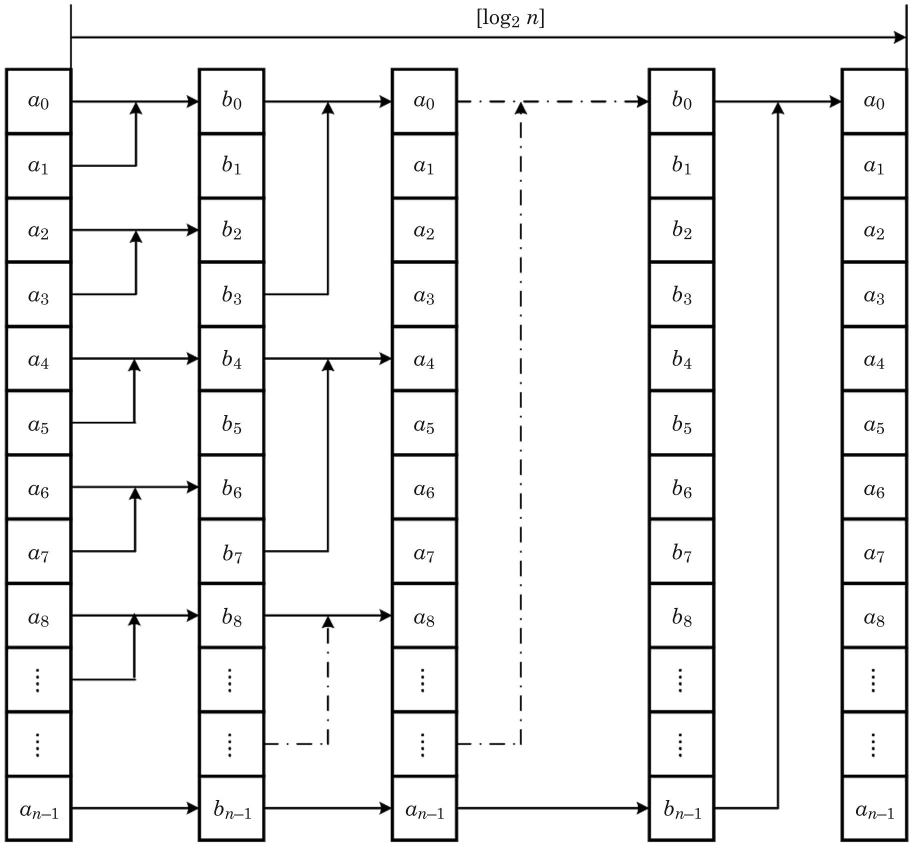

[log₂ n]
a₀ a₁ a₂ a₃ a₄ a₅ a₆ a₇ a₈ ⋮ ⋮ aₙ₋₁
b₀ b₁ b₂ b₃ b₄ b₅ b₆ b₇ b₈ ⋮ ⋮ bₙ₋₁

Fig. 6. Merge sorting of n elements

THEOREM 1. Merge Sort Algorithm has time complexity

$$T_{\max} = n \cdot \log_2 n - n + 1 \tag{1}$$

Proof. We are limiting deliberations to $n = 2^k$, where $k = 1,2, ...$

Inductive proof. For $k = 1$ the dimension of sorting sequence is $n = 2$. At the beginning algorithm merge two one element strings into one string. We can merge two strings with u and v elements making $u + v - 1$ operations of comparisons. To the formula (1), we get $n \cdot \log_2 n - n + 1 = 2 \cdot \log_2 2 - 2 + 1 + 1$. So for $k = 1$ the theorem is true.

We assume the true of the theorem for k. Hence $n = 2^k$ and we can sort a sequence doing no more comparison than

$$2 \cdot \log_2 2^k - 2^k + 1 \tag{2}$$

We have to prove that for $k + 1$ (the sequence is multiple by two and $n = 2^{k+1}$) the statement $2 \cdot \log_2 2^{k+1} - 2^{k+1} + 1$ is true. In step k + 1 we have two sequences with 2^k elements. Each one of two sequences, by the induction hypothesis, was sorted in no more comparisons then $2^k \cdot \log_2 2^k - 2^k + 1$. Now we merge 2 sequences of 2^k elements making no more than $2 \cdot 2^k - 1$ comparisons to sort. So estimating is:

$$2 \cdot (2^k \cdot \log_2 2^k - 2^k + 1) + 2 \cdot 2^k - 1$$

$$2^{k+1} \cdot \log_2 2^k - 2^{k+1} + 2 + 2^{k+1} - 1$$

$$2^{k+1} \cdot (\log_2 2^k + 1) - 2^{k+1} + 1$$

$$2^{k+1} \cdot \log_2 2^{k+1} - 2^{k+1} + 1$$

Which was to prove.

Presented method was implemented in C++ CLR Visual Studio Professional 2013. A simplified functional diagram no recursive method of sorting by merging is presented in Figure 7. The algorithm is divided into parts shown in Figure 8 and Figure 9. Sorting algorithm is invoked by specifying the array with number to sort.

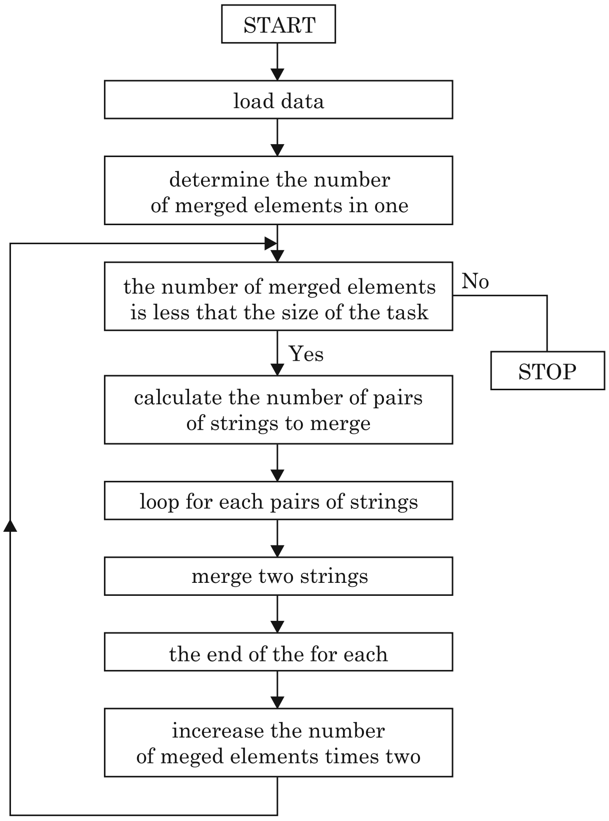

START
load data
determine the number of merged elements in one
the number of merged elements is less that the size of the task
No
STOP
Yes
calculate the number of pairs of strings to merge
loop for each pairs of strings
merge two strings
the end of the for each
incerease the number of meged elements times two

Fig. 7. No reclusive merge sort algorithm

```
Start
Load table a
Load table b
Load variable p1
Load variable c1
Load variable p2
Load variable c2
Load variable pb
While c1 greater than 0 and c2 greater than 0 then do
  If a[p1] less or equal a[p2] then do
  Begin
    Remember a[p1] in b[pb]
    Add to index p1 one
    Add to index pb one
    Subtract one from c1
  End
  Else
  Begin
    Remember a[p2] in b[pb]
    Add to index p2 one
    Add to index pb one
    Subtract one from c2
  End
While c1 greater than 0 then do
Begin
  Remember a[p1] in b[pb]
  Add to index p1 one
  Add to index pb one
  Subtract one from c1
End
While c2 greater than 0 then do
Begin
  Remember a[p2] in b[pb]
  Add to index p2 one
  Add to index pb one
  Subtract one from c2
End
Return pb
Stop
```

Fig. 8. Merge function two sorted numeric strings into a single sorted sequence of numbers

```
Start
Load table a
Load dimension of table a into n
Remember true in t
Create an array of b of dimension n
Remember 1 in m
While m is less than n then do
Begin
  Remember 0 in pb
  Remember 0 in i
  While i is less than n then do
  Begin
    Remember i in p1
    Remember i + m in p2
    If p2 greater than n then do
    Begin
      Remember n in p2
    End
    Remember n – p1 in c1
    If c1 greater than m then do
    Begin
      Remember m in c1
    End
    Remember n – p2 in c2
    If c2 greater than m then do
    Begin
      Remember m in c2
    End
    If t is true then do
    Begin
      Proceed function two sorted numeric strings into a single
      sorted sequence of numbers merging elements from
      array a in array b
    End
    Else
    Begin
      Proceed function two sorted numeric strings into a single
      sorted sequence of numbers merging elements from
      array b in array a
    End
    Add to index i the value 2 * m
  End
  Remember the negation of t in t
  Multiply variable m by two
End
If t is false then do
Begin
  Remember 0 in i
  While i is less than n then do
  Begin
    Remember b[i] in a[i]
    Add to index i one
  End
End
Stop
```

Fig. 9. Sorting function string of numbers by using the merge sort algorithm

Recursive merge sort algorithm

In the recursive merge method, a ternary division was used to share over two strings. Sharing is performed until we get two strings of single elements. Then the algorithm merges and passes them to second division as consecutive substrings for merging. Relevant here is how to make the merge of two strings. This can be done e.g. as shown in (MARSZAŁEK 2016) to select the smallest element and saving it merged within or act like (WOŹNIAK et al. 2013). A simplified functional diagram recursive method of sorting by merging is presented in Figure 10. The whole process of sorting sequence of numbers is shown in Figure 11.

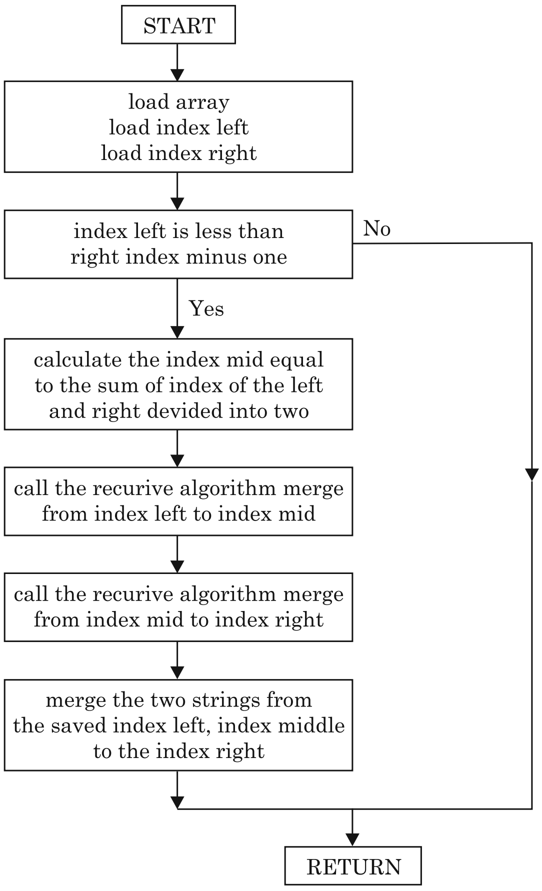

START
load array
load index left
load index right
index left is less than
right index minus one
No
Yes
calculate the index mid equal
to the sum of index of the left
and right devided into two
call the recurive algorithm merge
from index left to index mid
call the recurive algorithm merge
from index mid to index right
merge the two strings from
the saved index left, index middle
to the index right
RETURN

Fig. 10. Recursive merge sort algorithm

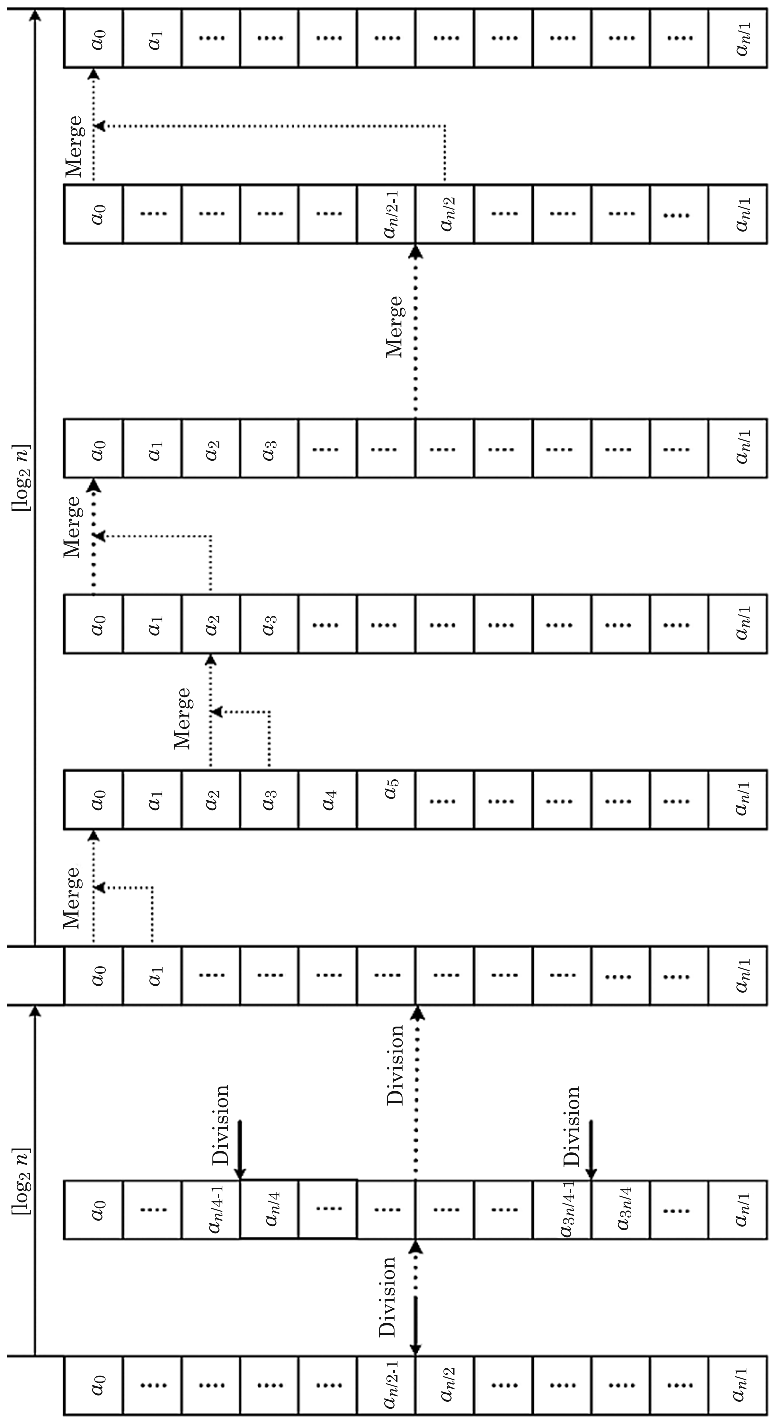


Fig. 11. Recursive merge sort algorithm

The study of the merge sort

The analysis of the tests presented for the algorithms was carried out for large data sets. Methods are implemented in C++ CLR in Visual Studio it 2013 Professional MS Windows Server 2012. Studies have been made on samples of 100 randomly generated for each desired dimension tasks using amd quad core processor 8356 8 p. The aim of the analysis was to compare the time of action for merge sort algorithm with recursive version of this algorithm. For determining the time of sorting have been selected samples of 10, 100, 1,000, 10,000, 100,000, 1,000,000, 100 million elements. Each sorting operation by examined methods was measured in time [ms] and CPU (Central Processing Unit) usage represented in track visitor interactions of CPU clock.

Table 1

Sorting results for recursive merge sort and no recursive merget sort

Elements	Method – average time sorting for 100 samples and special settings sorted the numbers			
	recursive merge sort algorithm		no recursive merge sort algorithm	
	ms	ti	ms	ti
10	1	42	1	28
100	1	642	1	383
1,000	6	8,871	3	5,417
10,000	55	85,459	37	58,158
100,000	576	897,853	349	543,702
1,000,000	6,665	10,388,724	4,105	6,398,619
10,000,000	75,007	116,909,369	46,979	73,224,239

Table 2

Coefficient of variation for recursive merge sort and no recursive merge sort

Coefficient of variation		
number of elements	recursive merge sort	no recursive merge sort
100	0.4266	0.4145
1,000	0.4086	0.4732
10,000	0.3163	0.4634
100,000	0.1675	0.3896
1,000,000	0.1255	0.1989
10,000,000	0.1263	0.1966
100,000,000	0.1322	0.2010

These results are averaged for 100 sorting samples and for a given dimension size added a sample consisting of numbers, ascending and descending, as well as samples containing numbers, which is a critical situation for the quick sort algorithm (WOŹNIAK et al. 2016). Benchmark comparison for recursive merge sort algorithm and no recursive merge sort algorithm in this paper are describe in Table 1 and Figure 12 and Figure 13.

Comparison of coefficient of variation for recursive mere sort and no recursive merge sort algorithm for large data sets is presented in Table 2.

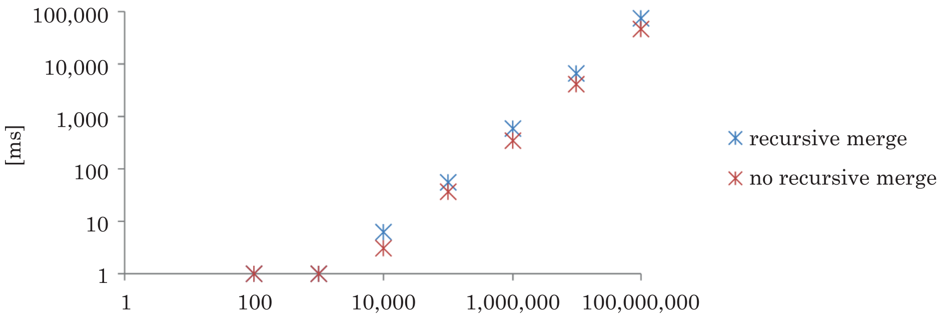


Fig. 12. Comparison of benchmark time [ms]

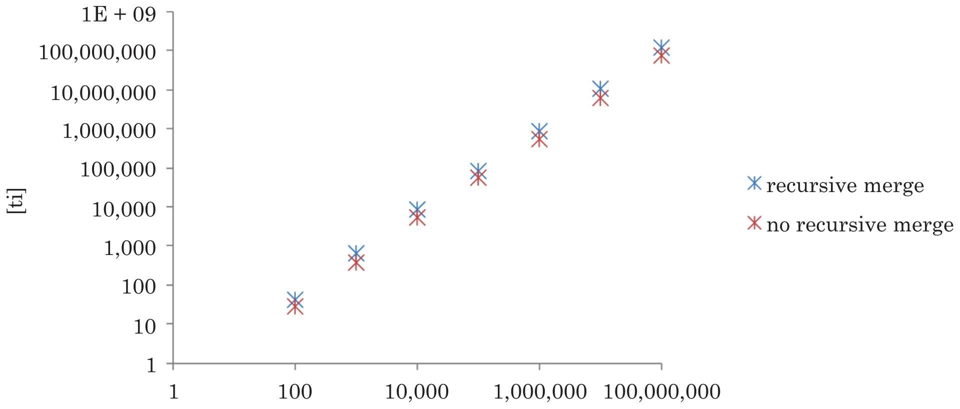


Fig. 13. Comparison of benchmark CPU operations [ti]

Both algorithms have almost identical statistical stability, which for large data sets is approximately more than 20% better for no recursive version. With an increase in the coefficient of variation task dimension stabilizes, which guarantees a repeatability of the results obtained in the work on any computer.

Comparison of time complexity algorithms

Comparison of time complexity allows to determine which algorithm transfer practical significance. Let us compare both methods of assuming the duration of the recursive merge sort and let us examine if the percentage is a longer duration of action for no recursive merge sort. The results are shown in the graphs Figure 14 and Figure 15.

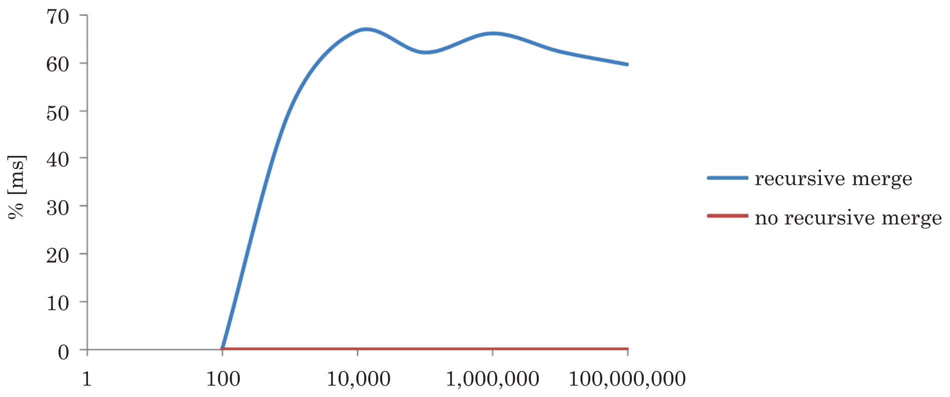


Fig. 14. Comparison of the two methods in terms of operational time [ms]

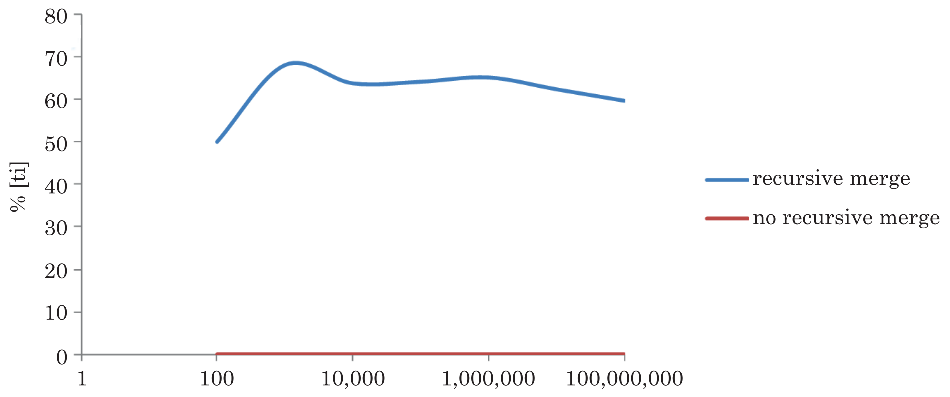


Fig. 15. Comparison of the two methods in CPU operations [ti]

Analysis of sorting times shows that the no recursive method of sorting is faster than recursive method for all tested dimensions. However both, recursive merge sort algorithm (WOŹNIAK et al. 2013) and no recursive version are a stable methods for sorting large data sets. The methods are also stable and effective for the small dimension of the task.

Final Remarks

The article presented merge sort algorithm for rapid sorting of large data sets. Studies have shown the effectiveness of the presented method for large data sets. Additional advantage of the proposed method is no deadlocks and the independence of the method from sorted strings. Sort analysis shows linear increase of sorting time. This is a very big advantage of the presented method and gives an opportunity to use it to sort data for any size task. The article compares the time complexity for a no recursive sorting algorithm by merging with the recursive merge sort algorithm. Tests confirm stability of both methods and theoretical complexity. Method of the no recursive version is faster than recursive merge sort algorithm. In addition, the method can be a simple way to write in the chipset which in turn determines the matter about the possibility of its practical application in the NOSQL databases.

References

ARTIEMJEW P., NOWAK B., POLKOWSKI L. 2016. *A New Classifier Based on the Dual Indiscernibility Matrix*. In: *Information and Software Technologies*. Eds. G. Dregvaite, R. Damasevicius. Communications in Computer and Information Science, 639: 380–391.

AUMULLER M., DIETZFELBINGER M. 2013 *Optimal Partitioning for Dual Pivot Quicksort*. ICALP'13 Proceedings of the 40th international conference on Automata, Languages, and Programming. Part I. Eds. F.V. Fomin, R. Freivalds, M. Kwiatkowska, D. Peleg. Springer-Verlag Berlin, Heidelberg.

AUMULLER M., DIETZFELBINGER M., KLAUE P. 2016. *How Good Is Multi-Pivot Quicksort?* ACM Trans.Algorithms, 13(1): 47.

DAMASEVICIUS R., MASKELIUNAS R., VENCKAUSKAS R., WOŹNIAK M. 2016a. *Smartphone User Identity Verification Using Gait Characteristics*. Symmetry 8(10): 100. doi: 10.3390/sym8100100.

DAMASEVICIUS R., VASILJEVAS M., SALKEVICIUS J., WOZNIAK M. 2016b. *Human Activity Recognition in AAL Environments Using Random Projections*. Comp. Math. Methods in Medicine, 2016(ID4073584): 17. http://dx.doi.org/10.1155/2016/4073584.

GABRYEL M. 2016. *The Bag-of-Features Algorithm for Practical Applications Using the MySQL Database*. Lecture Notes in Computer Science – ICAISC, 9693: 635–646. doi: 10.1007/978-3-319-39384-1.

GRYCUK R., GABRYEL M., SCHERER R., VOLOSHYNOVSKIY S. 2015. *Multi-layer Architecture For Storing Visual Data Based on WCF and Microsoft SQL Server Database*. Lecture Notes in Computer Science – ICIST, 9119: 715–726. doi: 10.1007/978-3-319-19324-3.

MARSZAŁEK Z., WOŹNIAK G., BOROWIK M., WAZIRALI R., NAPOLI C., PAPPALARDO G., TRAMONTANA E. 2015. *Benchmark tests on improved merge for big data processing*. Asia-Pacific Conference on Computer Aided System Engineering APCASE'2015, 14–16 July, Quito, Ecuador, pp. 96–101. IEEE. doi: 10.1109/APCASE.2015.24.

MARSZAŁEK Z., POŁAP D., WOŹNIAK M. 2014. *On preprocessing large data sets by the use of triple merge sort algorithm*. Proceedings of International Conference on Advances in Information Processing and Communication Technologies – IPCT'2014, 7–8 June, Rome, Italy, pp. 65–72. The IRED, Seek Digital Library. doi: 10.15224/978-1-63248-021-7-78.

MARSZAŁEK Z. 2017. *Performance test on triple heap sort algorithm*. Technical Sciences, 20(1): 49–61.

MARSZAŁEK Z. 2016. *Novel Recursive Fast Sort Algorithm*. Communications in Computer and Information Science – ICIST, 639: 344–355. doi: 10.1007/978-3-319-46254-7.

MARSZAŁEK Z. 2017. *Parallelization of Modified Merge Sort Algorithm*. Symmetry, 9(9): 176:1-176:18. doi: 10.3390/sym9090176.

MLECZKO W.K., NOWICKI R.K., ANGRYK R.A. 2016. *Rough Restricted Boltzmann Machine – New Architecture for Incomplete Input Data*. Lecture Notes in Computer Science-ICAISC, 9692: 114–125. doi: 10.1007/978-3-319-39378-0.

NEBEL M.E., WILD S., MARTINEZ C. 2016. *Analysis of Pivot Sampling in Dual-Pivot Quicksort: A Holistic Analysis of Yaroslavskiy;s Partitioning Scheme*. Algorithmica, 75(4): 632–683.

NOWICKI R.K., SCHERER R., RUTKOWSKI L. 2016. *Novel rough neural network for classification with missing data*. 21st International Conference on Methods and Models in Automation and Robotics, MMAR, Miedzyzdroje, August 29 – September 1, IEEE, p. 820–825. doi: 10.1109/MMAR.2016.7575243.

WENGER L.M., TEUCHOLA J.I. 1989. *The External Heapsort*. IEEE Transactions on Software Engineering, 15(7).

WOŹNIAK M., MARSZAŁEK Z., GABRYEL M., NOWICKI R.K. 2013. *On quick sort algorithm performance for large data sets*. In: *Looking into the Future of Creativity and Decision Support Systems*. Ed. A.M.J. Skulimowski. Progress & Business Publishers, 7–9: 647–656.

WOŹNIAK M., MARSZAŁEK Z., GABRYEL M., NOWICKI R.K. 2016. *Preprocessing Large Data Sets by the Use of Quick Sort Algorithm*. Advances in Intelligent Systems and Computing – KICSS, 364: 111–121. doi: 10.1007/978-3-319-19090-7.

WOŹNIAK M., MARSZAŁEK Z., GABRYEL M., NOWICKI R.K. 2013. *Triple heap sort algorithm for large data sets*. In: *Looking into the Future of Creativity and Decision Support Systems*. Ed. A.M.J. Skulimowski. Progress & Business Publishers, 7–9: 657–665.

WOŹNIAK M., MARSZAŁEK Z., GABRYEL M., NOWICKI R.K. 2013. *Modified merge sort algorithm for large scale data sets*. Leure Notes in Artificial Intelligence – ICAISC'2013, 7895, 612–622. doi: 10.1007/978-3642-38610-7 56.

ŻMUDZIŃSKI L., ARTIEMJEW P. 2017. *Path Planning Based on Potential Fields from Rough Mereology*. IJCRS, 2: 158–168.

WILD S., NEBEL M.E., MAHMOUD H. 2016. *Analysis of Quickselect Under Yaroslavskiy's Dual-Pivoting Algorithm*. Algorithmica, 74(1): 485–506.