Technical Sciences, 2017, 20(1), 49–61

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

PERFORMANCE TEST ON TRIPLE HEAP SORT ALGORITHM

Zbigniew Marszałek

Institute of Mathematics
Silesian University of Technology

Received 19 November 2016, accepted 16 January 2017, available online 19 January 2017.

K e y w o r d s: computer algorithm, data sorting, data mining, analysis of computer algorithms.

A b s t r a c t

Rapid information search in large data sets is one of the most important issues. Quite often it leads sorting strings stored in different cultures, languages. In this work the author presents a modified triple heap algorithm to sort strings for large data sets. Triple heap algorithm is the subject of research and demonstrating its usefulness in applications.

Introduction

Recent years have seen a substantial increase in computer capacity. Computers have become faster and began to store large data sets. The result is that we are forced to organize data sets in such a way that they can be efficiently processed. A special role is played here sorting algorithms giving the opportunity to operate on large data sets. Development of database structures enforces new versions of algorithms for specific applications. A modified version of the sort algorithm (WOZNIAK et al. 2013, 2016) allows the sorting of large data sets for specific issues. It is particularly convenient if we do not have a sufficiently large memory to duplicate the data, as required by the triple merge algorithm (WEGNER, TEUHOLA 1989) and derivatives (MARSZALEK 2016).

Correspondence: Zbigniew Marszałek, Zakład Matematyki Obliczeniowej i Informatyki, Instytut Matematyki, Politechnika Śląska, ul. Kaszubska 23, 44-100 Gliwice, phone: 32 237 13 41, e-mail: Zbigniew.Marszalek@polsl.pl

Also various approaches to other sorting algorithms are widely examined in the resent years, i.e. multi-pivot quicksort was examined for efficiency of pivot operations during sorting (AUMÜLLER et al. 2016). Similarly effects of partitioning were discussed for applications of quicksort (AUMÜLLER, DIETZFELBINGER 2013). Other research (WILD et al. 2016, NEBEL et al. 2016) proposed interesting approaches to improve some strategies for preprocessing input data before sorting. This paper presents a modification of the triple heap algorithm for sorting strings, and compares its efficiency to the traditional heap algorithm.

Related work

Sorting of search results is one of the most commonly used methods for the preparation of reports and quick information retrieval. In many of the works presented are algorithms for sorting of large data sets. In practical solutions we meet the need of effective memory management and use of algorithms that do not use additional resources i.e. for distributed gaming system (POLAP et al. 2015a, b). Some system models need efficient technology to store the data (DAMASEVICIUS et al. 2016, GABRYEL 2016) of different types, what can be of further processing in expertise (DAMASEVICIUS et al. 2016a, b). Experimental tests allow you to find the best solutions with the best possible computational complexity and apply them in practice. There are various methods useful in data mining (ARTIEMJEW et al. 2016, ARTIEMJEW 2014, 2015, NOWICKI et al. 2016), which are implemented in some special system data architectures like Boltzmann machine (MLECZKO et al. 2016). The paper presents a modified algorithm to sort through mounds used for sorting strings.

Big data sets

In large databases stored information is collected from the respective structures of different sources. These data are then processed so that it would be possible to quickly search information and efficient processing in order to produce the required reports. Information processing illustrated by Figure 1. It shows that the collected information is gathered mostly on external media and are given the appropriate classification and ordering. To organize large sets of data are used for stable algorithms with low computational complexity, allowing efficient operation in NoSQL databases. For comparison methods are carried out tests to check the operating time using a CPU (Central Processing Unit) and clock cycles (clock rate). This enables an efficient comparison algorithms.

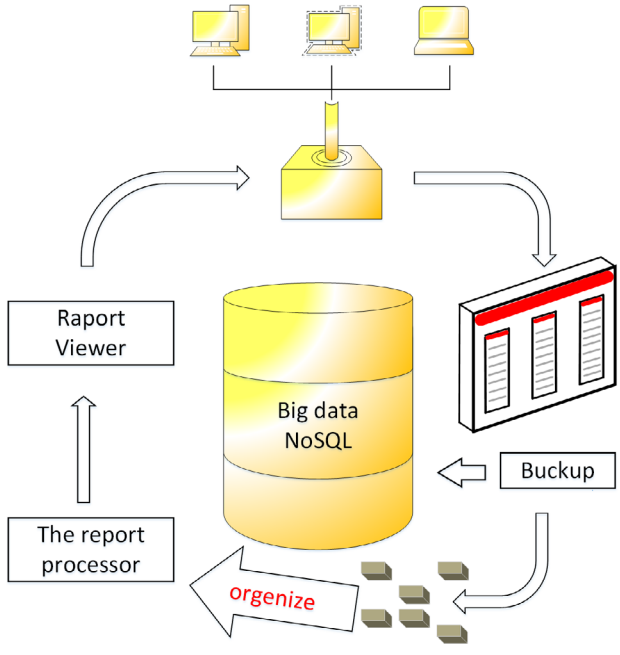

Fig. 1. Big data NoSQL database management system

Algorithms and statistical research

The statistical tests of time algorithms carried out tests for 100 samples for each dimension of the task of sorting. In each sample for a given dimension of the first tasks are random length strings of up to 8 characters, and then are inserted into chains of random capital letters of the English alphabet. For example, generated 10 sample string is shown in Figure 2.

The statistical tests were used methods such as in NOWICKI et al. (2016). A statistical average of n-element set of samples a_1, ..., a_n defined by the formula

$$\bar{a} = \frac{1}{n}(a_1 + \ldots + a_n).$$

The standard deviation is defined by the formula:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(a_i - \bar{a})^2}{n-1}},$$

where:
n – the number of elements in the sample,
a_1 – value of the random variable in the sample,
$\bar{a}$ – the arithmetic mean of the sample.

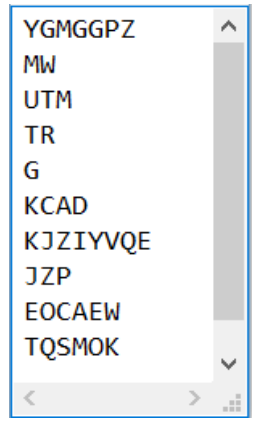

Fig. 2. Randomly generated sample for 10 strings

To determine the algorithms with the lowest time complexity they will be compared to the average time they work for large data sets. The standard deviation is characterized by the dispersion between time sorting. If we can determine the worst-case time sorting and 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 talking about the stability of the algorithm. It is determined by formula:

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

where:
σ – standard deviation of random variables in tests,
$\bar{a}$ – the arithmetic mean of the sample.

The analysis methods for sorting sets of random samples taken 10, 100, 1,000, 10,000, 100,000, 1,000,000 and 10,000,000 elements. The results are presented in graphs.

Triple Heap Sort Algorithm – THSA

The algorithms used in NoSQL databases as possible require low computational complexity. The methods of handling the data source and do not require additional resources are of particular interest. Let us now to present modified triple heap sort algorithm for sorting strings stored in Unicode. Strings a_i $i = 0, ..., n - 1$ present in the form of triple tree where the node number is the index of the string Figure 3.

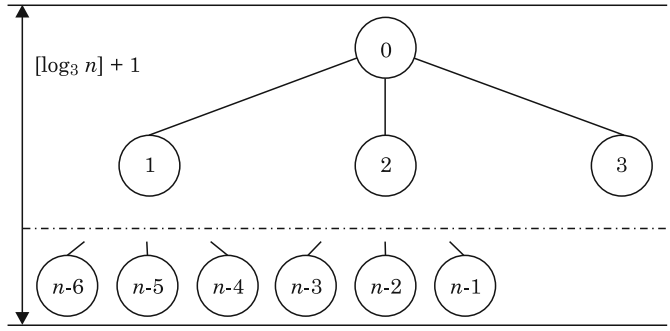

Fig. 3. Triple tree of strings

Sorting strings start of construction of the heap, i.e. Strings arrangement in such a way as to satisfy the conditions:

$$a[i] \geq a[3 \cdot i + 1] \; i = 0, ..., \left\lceil \frac{n-2}{3} \right\rceil \tag{1}$$

$$a[i] \geq a[3 \cdot i + 2] \; i = 0, ..., \left\lceil \frac{n-2}{3} \right\rceil \tag{2}$$

$$a[i] \geq a[3 \cdot i + 3] \; i = 0, ..., \left\lceil \frac{n-2}{3} \right\rceil \tag{3}$$

The relationship most equality is understood the same as the stacking order of words in the dictionary of a language of culture. Starting to build the heap from node $\left\lceil \frac{n-2}{3} \right\rceil$, we know that elements indexed from $n - 1$ down to $\left\lceil \frac{n-2}{3} \right\rceil + 1$ and they don't have descendants. We begin construction of the heap

with the $\left[\frac{n-2}{3}\right]$ having at least one descendant node. Next we are passing all nodes values up to the node with the index zero each time applying Algorithm 1. This algorithm is pushing the element off the root of the parent into the bottom as far as possible. It is held in this way. We sort the biggest value to be in a node from descendant nodes of the parent chosen according to (1) – (3). Next a value of the node of the parent is being compared with the nodes of the descendant. If the descendant node stores the value greater than the node of the parent, the value is changed and the algorithm gets up to the chosen descendant node. Than the same algorithm is applied for the parent to the moment when the parent isn't storing the greater value from a child or the parent doesn't have any children. Passing all nodes values up we receive a sorted heap. In the root the biggest value is sorted. We change this top node value with the value of last element of the heap and then reduce the number of elements in the heap by one. Again we apply the Algorithm 2 pushing the element off the root possibly far into the bottom to receive all elements sorted. Therefore the sequence on the stack is sorted by proposed THSA in the same table of strings, what make the method very efficient for big data system.

THEOREM 1. Height of triple heap, built of n elements is

$$k = [\log_3 n] + 1 \tag{4}$$

Where heap height k is undesrstood as number of levels in the heap and n is a number of all elements in the heap.

Proof. Each heap level k has from $1 = 3^0$ to 3^k elments. Therefore according to heap divisions we can estimate number of all elements in the heap as

$$1 + 3 + \ldots + 3^{k-1} + 3^k = \frac{1}{2}(3^{k+1} - 1) \tag{5}$$

In the last level of the heap do not have to be a complete number of leaves. Nevertheless we can write down that $r \leq 3^k$ that satisfy the equation

$$1 + 3 + \ldots + 3^{k-1} + r = n \tag{6}$$

Now we can estimate n

$$n = r + 1 + 3 + \ldots + 3^{k-1} \leq \frac{1}{2}(3^{k+1} - 1) \tag{7}$$

$$\log_3 (2n + 1) \leq (k + 1) \log_3 3 = (k + 1) \tag{8}$$

$$\log_3 (2n + 1) - \log_3 3 \leq k \tag{9}$$

$$\log_3 \left(\frac{2n+1}{3}\right) \leq \min_{k \in N} k \tag{10}$$

Given that $n \geq \frac{2n+1}{3}$ for $n \in N$ we obtain an estimate for $k \in N$

$$\log_3 \left(\frac{2n+1}{3}\right) \leq \log_3 n \leq [\log_3 n] + 1 = k \tag{11}$$

which proves the theorem.

THEOREM 2. Triple heap is created in a linear time.

THEOREM 3. Presented Triple Heap Sort Algorithm is sorting n elements in time

$$\upsilon (n \cdot \log_3 n) \tag{12}$$

which we give without proof.

Presented method was implemented in C++ CLR. The algorithm is divided into algorithm to re-arrange elements into triple heap Figure 4 and triple heap sort algorithm Figure 5.

```
Start
Load table a,
Load index t,
Load index r,
Remember 3 · t + 1 in k0,
Remember k0+1 in k1,
Remember k0+2 in k2,
Remember a[t] in x,
While r is greater than or equal to k2 then do
Begin
  Remember k0 in z,
  If a[k1] is greater than a[z] then do
    Remember k1 in z,
  If a[k2] is greater than a[z] then do
    Remember k2 in z,
  If a[z] is greater than x then do
  Begin
    Remember a[z] in a[t],
    Remember z in t,
    Remember 3 · t+1 in k0,
    Remember k0+1 in k1,
    Remember k0+2 in k2,
  End
  Else
  Begin
    Remember r+1 in k0,
    Remember r+2 in k1,
    Remember r+3 in k2,
  End
End
If r is greater than or equal to k1 then do
Begin
  Remember k0 in z,
  If a[k1] is greater than a[z] then do
    Remember k1 in z,
  If a[z] is greater than x then do
  Begin
    Remember a[z] in a[t],
    Remember z in t,
  End
End
Else
Begin
  If r is greater than or equal to k0 then do
  Begin
    If a[k0] is grater then x then do
    Begin
      Remember a[k0] in a[t],
      Remember k0 in t,
    End
  End
End
Remember x in a[t],
Return
Stop
```

Fig. 4. Algorithm to re-arrange into triple heap

```
Start
Load table a,
Load size of table a into n,
Remember (n-2)/3 in t,
While t is greater than or equal to zero then do
Begin
    Proceed Algorithm to re-arrange into triple heap with table a and
    setting index of initial heap element t and index of final heap element as n-1,
    Decrease the index t by one,
End
Remember n-1 in t,
While t is greater than zero then do
Begin
  Remember a[t] in x,
  Remember a[0] in a[t],
  Remember x in a[0],
  Proceed Algorithm to re-arrange into triple heap with table a and
  setting index of initial heap element 0 and index of final heap element as t-1,
  Decrease the index t by one,
End
Return
Stop
```

Fig. 5. Triple heap sort algorithm

The study triple heap sort algorithm

Performance analysis presented methods has been tested for sorting large data sets. The algorithm was implemented in C ++ CLR in Visual Studio 2015 Enrprice on MS Windows Server 2008 R2. The study was conducted on 100 samples randomly generated for each dimension of the task. Tests were carried out on quad core amd opteron processor 8356 8p. The purpose of the analysis and comparison is to check whether triple heap sort algorithm is better than traditional heap sort of sorting large data sets. For the benchmark we have applied input samples of 10, 100, 1,000, 10,000, 100,000, 1,000,000 and 10,000,000 elements. Each sorting operation by examined methods was measured in time [ms] and CPU (Central Processing Unit) usage represented in tics of CPU clock.

Summary of time sorting the method is binary heap sort and triple heap sort were placed in Table 1. These results are averaged for 100 sorting samples for each of BSHA and THSA in Figures 6 and 7.

Compare the current coefficient of variation methods are binary heap and triple heap sorting of large data sets.

Table 1

Sorting results binary heap sort and triple heap sort algorithm

Elements	Method – average time sorting for 100 samples			
	binary heap sort algorithm – BHSA		triple heap sort algorithm – THSA	
	ms	ti	ms	ti
10	1	33	1	30
100	1	689	1	499
1,000	8	11,612	5	6,940
10,000	122	189,626	63	98,877
100,000	1,479	2,305,583	846	1,317,943
1,000,000	19,425	30,276,976	11,217	17,483,931
10,000,000	246,460	384,143,716	143,035	222,939,977

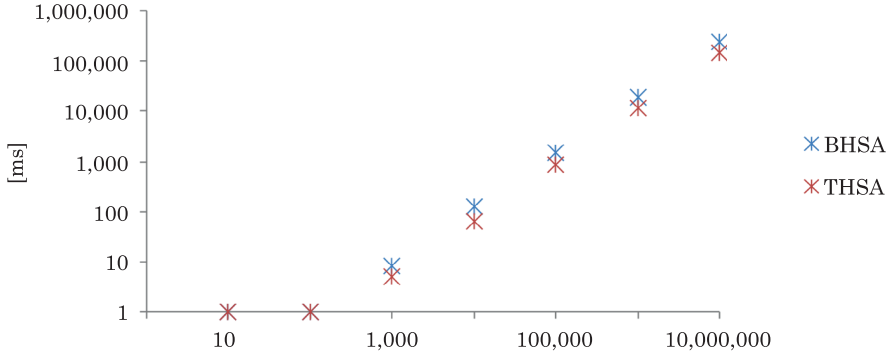

Fig. 6. Comparison of benchmark time [ms] for presented in Sec. 2 methods

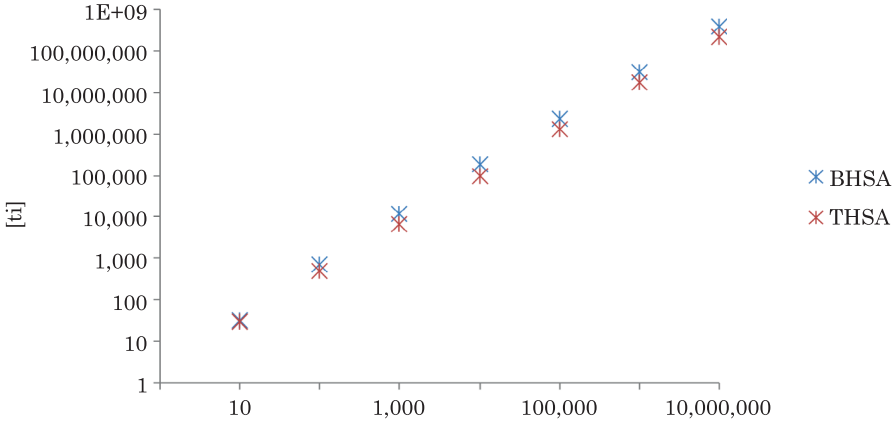

Fig. 7. Comparison of benchmark CPU operations [ti] for presented in Sec. 2 methods

Table 2

Coefficient of variation binary heap fast sort and triple heap sort

Coefficient of variation		
Number of elements	BHSA	THSA
10	0.925820099772551	1.24162020748728
100	0.843720298643621	0.338541807177754
1,000	0.232357160225841	0.0237000571071293
10,000	0.0232357160225841	0.0244774504038522
100,000	0.00792931906914819	0.0078376227129250
11,000,000	0.00667148990227573	0.0066610364196879
110,000,000	0.00487603595021875	0.00487596419925631

Analyzing Table 2 we see that both algorithms are similar in statistical stability for large data sets. Low stability of the algorithm for a small size job due to the fact that the system operation execution exceed the sorting algorithm. As size increases the coefficient of variation task is stabilizing.

Analysis of time sorting and comparison

Analysis and comparison will describe efficiency for sorting large data sets. Let us compare both methods of assuming the duration of the binary heap sort and let us examine if the percentage is a longer duration of action triple heap sort. The results are shown in the graphs Figures 8 and 9.

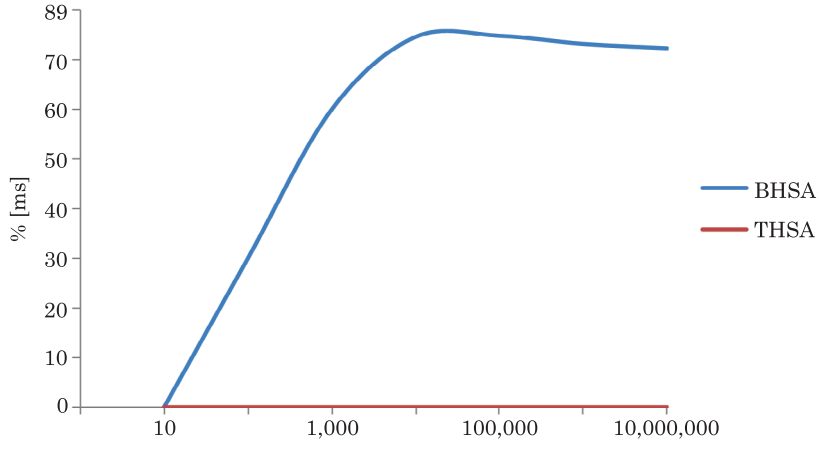

Fig. 8. Comparison of the two methods in terms of operational time [ms]

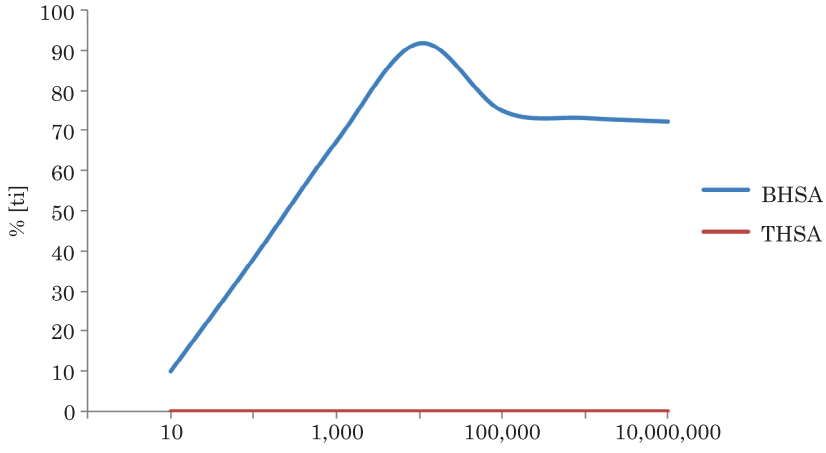

Fig. 9. Comparison of the two methods in CPU operations [ti]

The study shows that the triple heap sort method operate in a much shorter time measuring tasks from 1,000 to 10,000,000. Triple heap sorting algorithm is stable method allows sorting of large data sets stored in the form of strings in Unicode. Because it does not require additional resources, it is particularly the case when the entire job must do in the computer's memory.

Final Remarks

The article presented triple heap sort algorithm for rapid sorting of large data sets. The tests performed demonstrate the stability of the method and confirm the theoretical time complexity. Both Triple Heap Sort Algorithm and Binary Heap Sort Algorithm for fast sorting may find practical application in NoSQL databases.

References

Artiemjew P. 2014. *Rough mereology classifier vs. simple DNA microarray gene extraction methods*. International Journal of Data Mining, Modelling and Management, 6(2): 110–126. doi: 10.1504/IJDMMM.2014.063193.

Artiemjew P. 2015. *The Boosting and Bootstrap Ensemble for Classifiers Based on Weak Rough Inclusions*. Lecture Notes in Computer Science – RSFDGrC, 9437: 267-277. doi: 10.1007/978-3-319-25783-9.

Artiemjew P., Nowak B.A., Polkowski L.T. 2016. *A New Classifier Based on the Dual Indiscernibility Matrix*. Communications in Computer and Information Science – ICIST, 639: 380–391. doi: 10.1007/978-3-319-46254-7.

AUMÜLLER 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.

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

DAMASEVICIUS R., MASKELIUNAS R., VENCKAUSKAS A., WOZNIAK 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(ID 4073584): 17. doi: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.

MARSZALEK 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.

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., MARTÍNEZ 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.

POLAP D., WOZNIAK M., NAPOLI CH., TRAMONTANA E. 2015a. *Is Swarm Intelligence Able to Create Mazes?* International Journal of Electronics and Telecommunications, 61(4): 305–310. doi: 10.1515/eletel-2015-0039.

POLAP D., WOZNIAK M., NAPOLI CH., TRAMONTANA E. 2015b. *Real-Time Cloud-based Game Management System via Cuckoo Search Algorithm*. International Journal of Electronics and Telecommunications, 61(4): 333–338. doi: 10.1515/eletel-2015-0043.

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

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

WOZNIAK M., MARSZALEK Z., GABRYEL M., NOWICKI R.K. 2013. *Modified Merge Sort Algorithm for Large Scale Data Sets*. Lecture Notes in Computer Science – ICAISC, 7895: 612–622. doi: 10.1007/978-3-642-38610-7.

WOZNIAK M., MARSZALEK 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.