International Journal of Civil and Structural Engineering– IJCSE
Volume 1 : Issue 3 *[ISSN 2372 – 3971]*
Publication Date : 30 September, 2014

On preprocessing large data sets by the use of triple merge sort algorithm

[Zbigniew Marszałek, Dawid Połap, Marcin Woźniak]

***Abstract*—This paper illustrates preprocessing large data sets by the use of triple merge sort algorithm. Examined algorithm is oriented on large data sets and as research results have shown the version is about 15% faster than classic one. This feature may be crucial for efficiency in NoSQL database systems or other intelligent application operating on large data sets. In the paper is presented and discussed examined version. There are presented theoretical discussion and practical verification.**

***Keywords*—computer algorithm, data mining, sorting algorithm, analysis of computer algorithms**

I. Introduction

Computer Science is one of most developing sciences in recent years. This development is outcome of technology improvements. Nowadays electronic devices are capable of more new features and functions. We use electronics to help in medicine, engineering, economics, transport and other. Therefore database systems collect more and more information. However these mean that computers must operate on very large sets of information. Sorting algorithms are very important when one need to work on big data sets. There are known different versions, however many of them can be improved for special purposes. Here, the authors would like to present and discuss research results on organizing and sorting strategies in NoSQL database systems and intelligent applications using selected algorithm. Preprocessing large data sets is examined using triple merge sort algorithm. Merge sort algorithm uses structure of the stack. Such structure is set of elements that are placed in order. Input data is divided into stacks that are merged into one single sorted output. Sort by merging can be done in many ways.

Zbigniew Marszałek
Institute of Mathematics, Silesian University of Technology
Gliwice, Poland

Dawid Połap
Institute of Mathematics, Silesian University of Technology
Gliwice, Poland

Marcin Woźniak
Institute of Mathematics, Silesian University of Technology
Gliwice, Poland

The authors of [1, 4, 9, 18] presented first versions of merge sort algorithm. Some other versions are presented in [8, 14, 15, 20, 26, 27]. In [2, 7, 12, 22, 23] are examples of other special algorithms for sorting large data sets. In [3, 5, 6, 8, 17, 24] is presented parallel approach to programming selected methods. In [5, 7, 10, 11, 16, 19, 21, 24, 28] are presented solutions to improve sorting. In this paper we propose merge sort algorithm dedicated for large data sets. Presented modifications perform sorting without recursion, what increases stability and reduces sorting time. Proposed algorithm consists of two parts. The first procedure is merging sorted items into stacks. The second procedure sorts stacks. In classic version, presented in [1, 9, 20], are only merged two stacks. This paper provides extended procedures dedicated for large data sets. Extension of procedure is based on merging multiple stacks, what can be very efficient in parallel systems. The authors examined proposed extensions. Research results show that multiple merge algorithm can be expanded to increase stability and make it sort faster. This allows no recursive optimization of sorting for large data sets.

II. Triple Merge Sort Algorithm For Large Data Sets

Let us suppose we have very large input sequence. We can sort it by dividing it into subsequences then merge sorted substrings. Double merge procedure in first step begins with comparison of pairs on the input. In this way we obtain two component stacks. In second step, we obtain stacks containing doubled number of elements. We merge until we have only one stack. If the initial sequence contains 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 output. Modified version of double merge dedicated for large data sets was presented in [20]. Some other versions for large data sets with important improvements in double merge sort algorithm are shown in [14, 15, 20]. A number of modifications was described in [26, 27, 28]. Authors of [6, 8, 24] showed possibility of introducing multithreading. In papers [5, 10, 12] is described possibility of adaptation to special initial conditions. While in [13, 22, 23] are presented applications of selected sorting methods in NoSQL database systems and intelligent applications. The authors of present paper propose non recursive version dedicated to large data sets. It is based on extended structure of stack, what increases stability and efficiency.

Let us now discuss possibility of merging three stacks. The algorithm will merge three sequences X, Y and Z. While

sorting, we are having *2*3m-1* comparisons, where *3m* is number of elements in *X*, *Y* and *Z*. Merging is in some way similar to classic version, however in the research we have examined extended stacks. We used three instead of two. While performing operations, algorithm compares three stacks. First it compares every three elements. Than the number of elements in compared stacks is tripled by merging. This multiplication is preformed until all elements are merged and sorted. If on input there are elements which have no filling to triple stack structure, we rewrite them in similar way as in [20]. Thus sorting procedure shall be more efficient and faster. Sample sorting by triple merge is presented in figure 1.

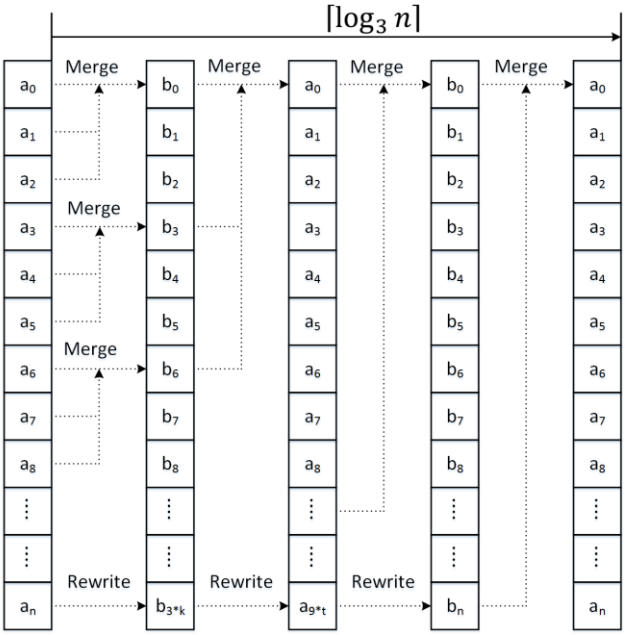

Figure 1. Sample triple merge sorting

This method was described theoretically and results were analyzed in practice. Tests were performed to examine the method and compare it with other methods dedicated for large data sets.

A. *Time Complexity In Theory*

Let us now discuss time complexity of the algorithm. This factor describes how fast the algorithm may perform in common use. Mainly we understand it as expected (average) value, which with some changes (described by standard deviation) shall be equal to computers with comparative numerical capacities, please see [19] and [24]. Presented algorithm of triple merge sorting has time complexity described in the following theorem.

Theorem. Triple merge sort algorithm for large data sets has time complexity $2n \cdot \log_3 n$.

Proof. Starting triple merge sorting algorithm, we merge three one element strings in three piece string. Such merging for *n*-element sequence can be carried out using *n* comparisons (for three elements one need to make three comparisons). In next steps we merge three element strings into structured sequence. This operation needs for *n*-element sequence no more than *2n* comparisons (to organize three *m* elements sequences into structured sequence one need to make *2*3m-1* comparisons). Each time merging three sorted strings, we get one structured sequence. Therefore to sort input *n* element sequence we merge in *k* steps and each of them makes no more than *2n* comparisons. In conclusion, without loss of generality, we can assume that sorted sequence has 3^k elements. Thus, we estimate the following assumption

$$\min_{k \in N} 3^k \geq n. \tag{1}$$

Logarithms both sides in (1) we have

$$\min_{k \in N} \log_3 3^k \geq \log_3 n. \tag{2}$$

Thus, on basis of logarithmic function we obtain the following

$$\min_{k \in N} k \cdot \log_3 3 \geq \log_3 n. \tag{3}$$

What means that (3) is equal to

$$\min_{k \in N} k \geq \log_3 n. \tag{4}$$

Finally, we can assume that number of operations performed by sorting will be

$$k = \lceil \log_3 n \rceil. \tag{5}$$

Therefore time complexity is

$$T_{avg} = 2n \cdot k = 2n \cdot \log_3 n. \tag{6}$$

Presented method was implemented. The implementation was then examined in research. Algorithm was compiled as CLR C++ in MS Visual Studio 2012 Ultimate. Tests were carried out on quad core amd opteron processor 8356 8p.

In analysis of examined methods we are looking for solutions of best time complexity and high stability. To describe operations were measured characteristics of CPU (Central Processing Unit) clock cycles and timing. CPU clock cycle (CPU clock rate) is calculated as the rate in cycles per second at which tested processor performs basic operations like moving values between registers. On its basis, we can estimate performance. Execution time is time in which CPU executes procedures. However it can reflect not only execution of the algorithm but also some other operations performed in the system. These characteristics should be measured for representative number of samples. The study of extended methods was carried out for 100 test input files in each of described classes. In the research have been tested random arrangements, reversely sorted sequences and sequences in correct order. In the analysis we used statistical methods as

arithmetic mean, standard deviation, mean deviation and coefficient of variation. In [19] are described examples of application of descriptive statistics to analyze procedures, for more details please see also [2, 4, 9].

Arithmetic mean, also called expected value, for *n* values from sample set of numbers $a_1,\ldots,a_n$ is based on formula

$$\overline{a} = \frac{a_1 + \ldots + a_n}{n}. \quad (7)$$

We also estimated possible deviation from expected value. Standard deviation is statistical measure, describing variability of characteristic. This also helps to estimate stability and performance. In statistics, there are several types of standard deviation. In analysis of selected algorithms we used concept of standard deviation for sample. This estimates standard deviation of population using knowledge of some of its objects. Standard deviation of random variable is in general denoted by formula

$$\sigma = \sqrt{E((X - E(X))^2)} = \sqrt{E(X) - (E(X))^2}, \quad (8)$$

where *E(X)* represents expected value of random variable *X*. In the research, results of each sampling are discrete variables. Thus, standard deviation is calculated for discrete random variables. A general formula for this type is

$$\sigma = \sqrt{\sum_{i=1}^{n}\left(a_i - \sum_{i=1}^{n} a_i \cdot p_i\right)^2 \cdot p_i}, \quad (9)$$

where symbols mean that random variable *X* can have *n* values $a_1,\ldots,a_n$ with corresponding probabilities $p_1,\ldots,p_n$. However to accurately describe the test one should determine standard deviation of entire population, based no formula

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}\left(a_i - \sum_{i=1}^{n} a_i \cdot p_i\right)^2}{n}}, \quad (10)$$

where symbols mean the same as in (9). In our research, (10) is estimated by entire sample standard deviation in (11). Approximation depends on information we have about observation. Therefore in study of large data sets we used formula

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(a_i - \overline{a})^2}{n-1}}, \quad (11)$$

where symbols mean:
n – number of elements in the sample,
$a_1,\ldots,a_n$ - random variables in the sample,
$\overline{a}$ - arithmetic mean of the sample determined in (7).

Standard deviation estimator in (11) is unbiased variance estimator with very slight error. Interpretation of standard deviation is possible distance from the average. With increase in standard deviation increases possibility of major differences in performance compared to average. At the same time, the smaller standard deviation, the greater certainty that results will be close to average. An important aspect of the analysis is to identify algorithms stability. Stability is best described by coefficient of variation. Coefficient of variation is measure that allows determining diversity in population. In the research it is defined by formula

$$V = \frac{\sigma}{\overline{a}}, \quad (12)$$

where the symbols mean:
σ - standard deviation of random variables in the tests, formula (11),
$\overline{a}$ - arithmetic mean of the sample, formula (7).

Based on its analysis, combined with analysis of standard deviation, we estimated whether the algorithm is stable. The higher are standard deviation and coefficient of variation the greater is potential instability of the algorithm. Comparison of these values will help to identify best of examined algorithms with good stability which run in short time. In analyzes as particular object we understand sample sort operation. We examined random sets and as example plotted results for sets of 100, 1 000, 10 000, 100 000, 1 000 000, 10 000 000 and 100 000 000 elements. In each of examined cardinalities, sorting tests were performed for different layouts of elements on input. Sorting tests were performed for randomly selected 100 sequences of each cardinality. Let us now present examined algorithm of triple merge sort for large data sets.

B. *Examined Algorithm*

When one implements solution for large data sets it is crucial to think of possible problems. Procedure must be ready to operate on large data sets of any pose on the input. There cannot be stack overflow as they are for other algorithms (see [21] and [22]). Moreover as the input data is large, the method shall use as little memory as possible. There is also need to perform sorting fast enough for even simple computer systems (see [13], [19] and [23]). All these aspects make it important to verify solution in different tests and examinations. In section III we present research results, which prove that examined method is well organized and efficient for large data sets. We have implemented proposed algorithm using experience from other tests and research described in [13] and [19]-[24]. Examined method performs sorting without recursion, what increases stability and reduces sorting time. Triple merge sort algorithm dedicated for large data sets consists of two parts. The first procedure is merging, as presented in figure 2. It is used to merge sorted items into

stacks. The second procedure sorts those stacks, as presented in figure 3.

```
Start
Load pointer  to table a
Load pointer  to table b
Load pointer  to table p
Construct table q size of 3
For i=0, i<3, i++ do
Remember value q[i] in p[i]
End
Set Boolean variable  bb as true
While variable bb is true, do
   Set Boolean variable as false
    For i=0, i<3, i++ do
        If index q[i] is less than p[i+1], then
             If variable bb is true, then
                  If element a[q[i]] is less than z, then
                       Remember value a[q[i]] in z
                       Remember index i in t
                  End
              Else
                  Set Boolean variable  bb as true
                  Remember value a[q[i]] in z
                  Remember index i in t
         End
    End
   If variable bb is true, then
        Remember value z in b[pb]
        Increase index pb by one
        Increase index q[i] by one
   End
End
Stop
```

Figure 2. Algorithm to merge three sequence

Presented solution has similar composition to method described in [20]. It was also used here, as the research have shown validity of improvements in solution for large data sets, for more detail see [19]-[23]. Examined methods are composed of two algorithms. These algorithms operate on input data to perform sorting. Algorithm presented in figure 3 is master one, which calls algorithm presented in figure 2 to help organize processed information. Algorithm to merge sequences makes the data in the stack to have appropriate structure. This structure is then used for process of sorting. Let us now present results of examinations.

III. Research results

Theoretical analysis of examined version shows that it shall be similar in efficiency to method presented in [20]. To verify this conclusion we have compared results of numerical experiments. Described algorithm was examined for large data sets. Algorithm was implemented using CLR C++ in MS Visual Studio 2012 Ultimate on MS Windows server 2008 R2. To test were taken random samples of 100 series in each class of frequencies, including unfavorable positioning. Tests were carried out on quad core amd opteron processor 8356 8p.

```
Start
Load pointer  to table a
Load size of  the data into n
Construct table b size of n
Construct table p size of 4
Set Boolean variable b1 as true
For m=1, m<n, m*=3 do
    For i=0, i<n, i+=(3*m) do
          For k=0, k<4, k++ do
               Set index p[k] as i +m
               If index p[k] is greater n, then
                    Set index p[k] as n
               End
          End
          If variable b1 is true, then
               Proceed Algorithm to merge three
               sequence  from array a to array b setting
               indexes  as table p
          Else
               Proceed Algorithm  to merge three
               sequence  from array b to array a setting
               indexes as table p
          End
     End
     Set b1 as negative b1
End
 If variable b1 is false, then
       Copy elements from array b to array a
Stop
```

Figure 3. Algorithm to sort merged sequence

The aim of analysis and comparison is to verify if extended triple merge is better to sort large data sets than classic version. In examinations and tests were compared characteristics presented in section II.

1) Double merge for large data sets

Let us first present research results for double merge sort, for more details about this algorithm please see [20]. In table 1 are presented research results for CPU usage.

TABLE I. DOUBLE MERGE CPU USAGE

CPU [ti]				
Number of elements	*avg*	*standard deviation*	*avg deviation*	*coefficient of variation*
100	2577,4	61,25	51,68	0,023
1000	3866,4	201,70	170,48	0,052
10000	21208,2	2494,63	2129,84	0,117
100000	140215,6	25280,54	19570,72	0,180
1000000	1356215,6	201155,10	175020,72	0,148
10000000	15937998	2369059,54	2064752,4	0,148
100000000	179688651,6	27556710,6	23970838,32	0,153

Research results on quad core amd opteron processor 8356 8p

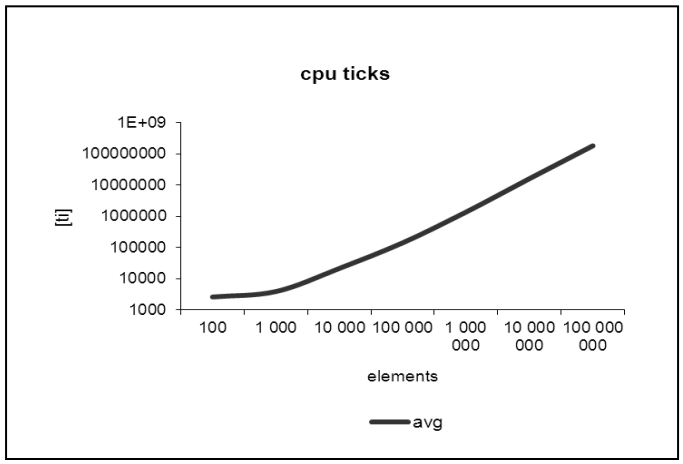

Figure 4. Double merge CPU usage

In figure 4 is shown average CPU usage while sorting. In table 2 are presented research results for sorting time.

TABLE II. DOUBLE MERGE SORTING TIME

Time [hh]				
Number of elements	*avg*	*standard deviation*	*avg deviation*	*coefficient of variation*
100	0:0:0.00165	0:0:0.00003	0:0:0.0003	0,023
1000	0:0:0.00248	0:0:0.00012	0:0:0.0010	0,052
10000	0:0:0.01360	0:0:0.00160	0:0:0.0136	0,117
100000	0:0:0.08996	0:0:0.01622	0:0:0.1255	0,180
1000000	0:0:0.88657	0:0:0.53971	0:0:0.3199	0,173
10000000	0:0:10.2259	0:0:1.52000	0:0:1.2475	0,148
100000000	0:1:55.28923	0:0:17.6805	0:0:15.3798	0,153

Research results on quad core amd opteron processor 8356 8p

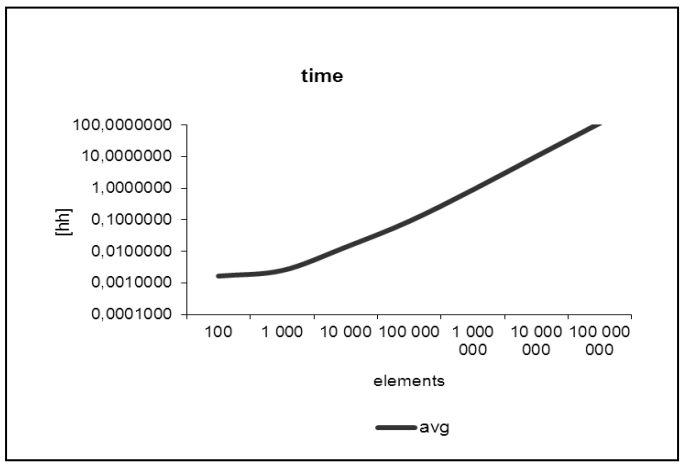

Figure 5. Double merge sorting time

In figure 5 is shown expected sorting time. However in the research we try to examine different algorithms to find fastest and most stable one in the sense described in section II. Let us now discuss stability based on coefficient of variation presented in section II.

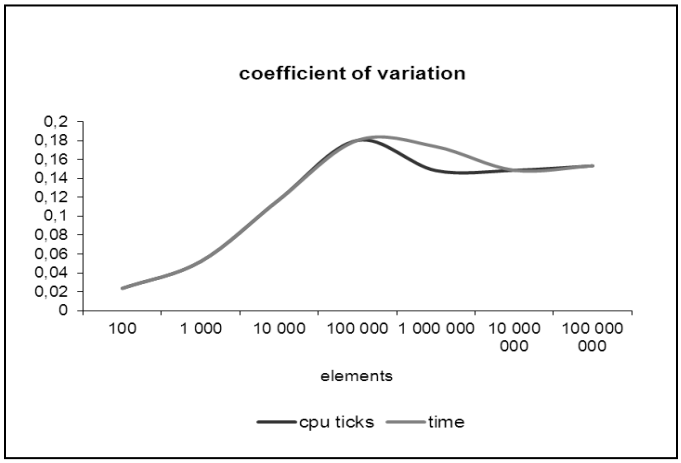

Figure 6. Triple merge - coefficient of variation

Coefficient of variation is presented in figure 6. We can see that for large data sets (above 1 000 000 elements on the input) double merge sort method has CPU usage coefficient of about 0.15. This means that usage of processor is stable and the algorithm performance for large data sets is good. Similar situation is for sorting time. In figure 6 it is marked with grey color. Coefficient of variation for sorting time is similar to CPU usage. However here we see more possible fluctuations. This results will be compared with similar values describing triple merge sort.

2) Triple merge for large data sets

Triple merge sort uses extended stacks. Elements are merged in triple stacks to sort and further processing. In table 3 are presented research results for CPU usage.

TABLE III. TRIPLE MERGE CPU USAGE

CPU [ti]				
Number of elements	*avg*	*standard deviation*	*avg deviation*	*coefficient of variation*
100	2532	14,68	12	0,006
1000	3694,2	172,58	147,44	0,047
10000	18167,2	2337,31	2006,64	0,129
100000	134077,4	17648,22	14153,68	0,132
1000000	1164992,2	211194,94	182129,04	0,181
10000000	13366027	2434017,52	2101512,8	0,182
100000000	149713305,6	28582137,68	24764383,92	0,191

Research results on quad core amd opteron processor 8356 8p

Values of average CPU usage are shown in figure 7. The chart of average CPU usage for triple merge sort is similar to double merge sort in figure 4. Therefore both methods shall have similar features, however in presented research most

important is efficiency and coefficient of variation to compare them.

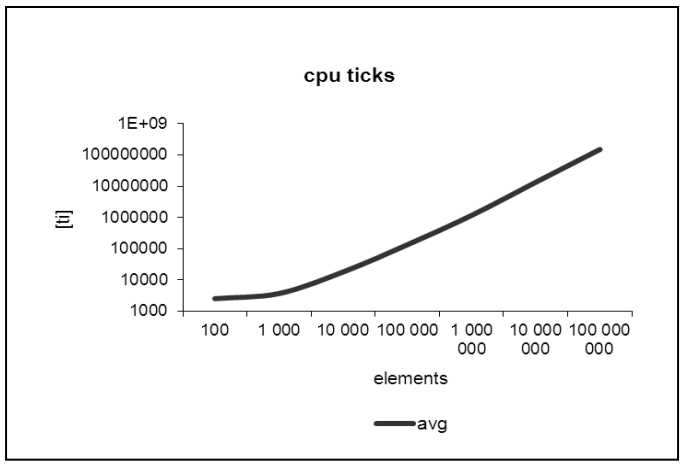

Figure 7. Triple merge CPU cycles

In table 4 are presented research results for sorting time.

TABLE IV. TRIPLE MERGE SORTING TIME

Time [hh]				
Number of elements	***avg***	***standard deviation***	***avg deviation***	***coefficient of variation***
100	0:0:0.00162	0:0:0.00001	0:0:0.00008	0,006
1000	0:0:0.00237	0:0:0.00011	0:0:0.0009	0,047
10000	0:0:0.01165	0:0:0.00149	0:0:0.0128	0,129
100000	0:0:0.08602	0:0:0.01132	0:0:0.0908	0,132
1000000	0:0:0.74300	0:0:0.13003	0:0:0.1149	0,175
10000000	0:0:8.57571	0:0:1.56167	0:0:1.4834	0,182
100000000	0:1:36.05688	0:0:18.33843	0:0:15.8896	0,191

Research results on quad core amd opteron processor 8356 8p

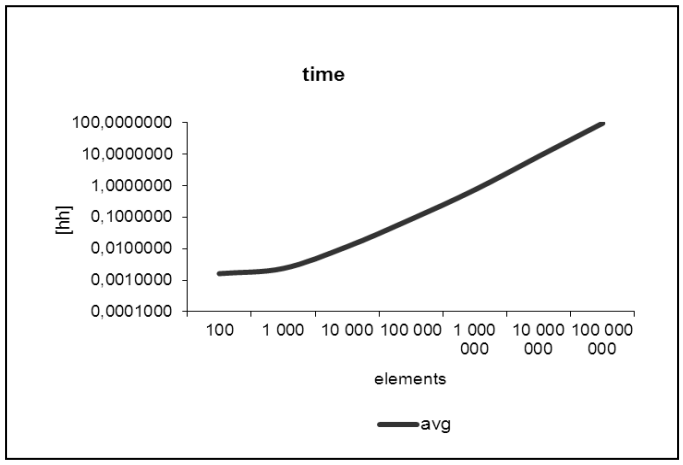

Figure 8. Triple merge sorting time

In figure 8 is shown expected sorting time. In this paper we want to compare classic merge sort with extended version for large data sets. Therefore in figure 9 we present chart of coefficient of variation for triple merge sort.

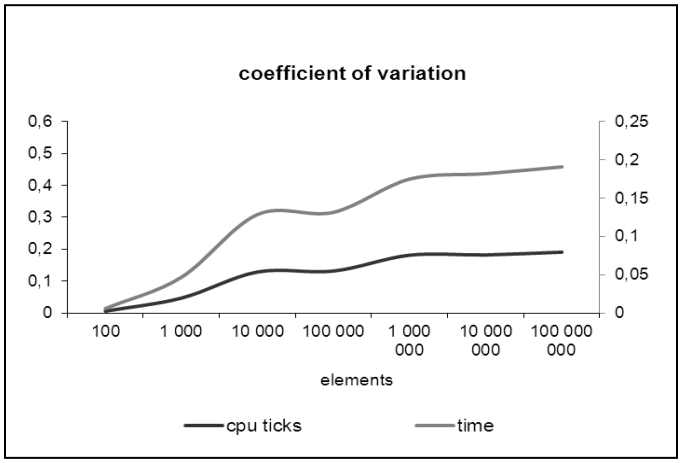

Figure 9. Triple merge - coefficient of variation

Coefficient of variation for triple merge sort is presented in figure 9. We can see that this method shall be more stable in statistic sense, as chart of coefficient of variation has less fluctuations. Both lines are almost flat for large data sets (above 1 000 000 elements on input). CPU usage coefficient of variation is lower than 0.2. This means that usage of processor is stable and the algorithm performance for large data sets is good. Coefficient of variation for sorting time is similar to CPU usage. However here we see more possible fluctuations. This results will be compared with similar values describing double merge sort.

Once we have examined presented extended merge sorting and classic method, we can compare them. The aim of analysis and comparison is to verify if extended triple merge is better to sort large data sets than classic version. We will try compare efficiency for large data sets and coefficient of variation.

IV. Analysis And Comparison

Analysis and comparison will describe efficiency for sorting large data sets. We will compare CPU usage and measured sorting time and coefficient of variations for both examined methods. Let us first compare statistical stability. In table 5 is presented CPU usage.

TABLE V. CPU USAGE COMPARISON

Coefficient of variation		
Number of elements	***merge 2***	***merge 3***
100	0,023	0,006
1000	0,052	0,047
10000	0,117	0,129
100000	0,180	0,132
1000000	0,148	0,181

10000000	0,148	0,182
100000000	0,153	0,191

Values calculated using formula (12)

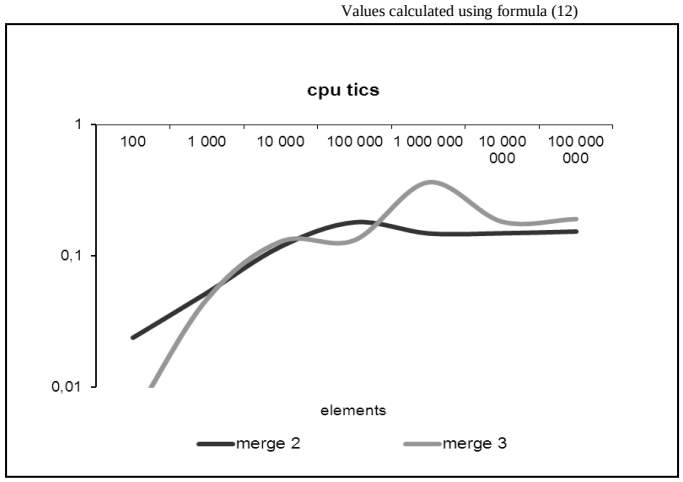

Figure 10. Coefficient of variation cpu usage caomparison chart

Analyzing figure 10 we see that for large data sets (above 10 000 000 elements on input) double merge and triple merge should have similar features. Let us now compare sorting time.

TABLE VI. SORTING TIME COMPARISON

Coefficient of variation		
Number of elements	***merge 2***	***merge 3***
100	0,023	0,006
1000	0,052	0,047
10000	0,117	0,129
100000	0,180	0,132
1000000	0,173	0,175
10000000	0,148	0,182
100000000	0,153	0,191

Values calculated using formula (12)

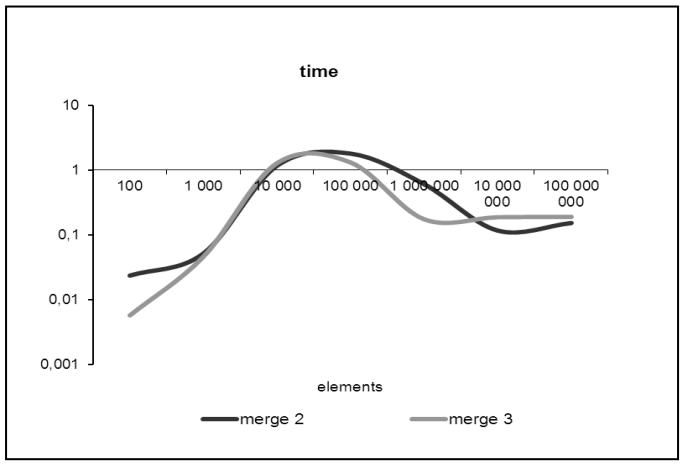

Figure 11. Cofficient of variation comparison chart for time

Analyzing figure 11 we see that both algorithms are similar in statistical stability for large data sets. Similar conclusions come from analysis of figure 10. Therefore in comparison most important will be efficiency in sorting large data sets. In figure 12 and figure 13 we present comparison of both examined methods for CPU usage and sorting time, respectively. Triple merge sort is marked in grey while double merge sort is marked in dark.

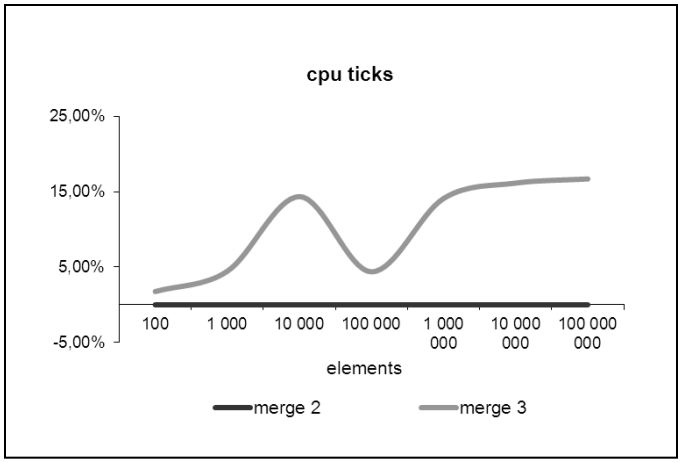

Figure 12. Comparison of cpu usage efficiency

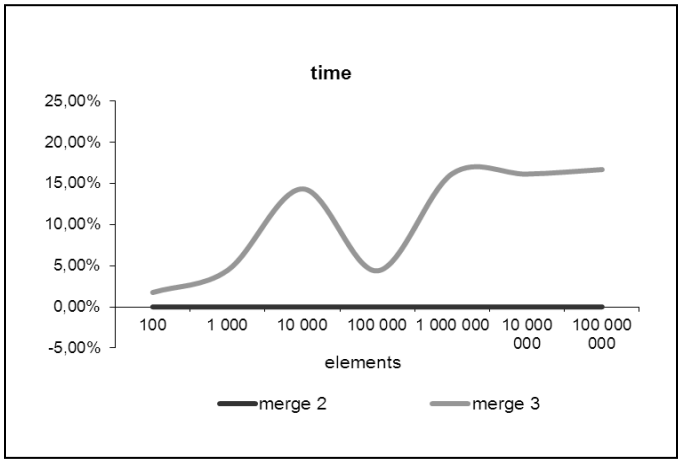

Figure 13. Comparison of time efficiency

Analyzing figure 12 and figure 13 we see that for large data sets (above 1 000 000 elements on input) triple merge sort is more efficient. Efficiency of triple merge sort in comparison to double merge sort is about 15% better. This means that presented in section II algorithm is efficient for preprocessing large data sets.

v. Conclusions

In the research we have compared double merge and triple merge algorithms in versions dedicated for large data sets. As

the research have shown triple merge sort is about 15% more efficient in preprocessing large data sets. Therefore extended merge sort is better sorting method for large data sets. In the future research some other extensions of merge sort will be examined and compared.

References

[1] A. Aho, J. Hopcroft and J. Ullman, The design and analysis of computer algorithms. Addison-Wesley Publishing Company, USA 1975.

[2] G.E. Blelloch, C.E. Leiserson, B.M. Maggs, C.G. Plaxton, S.J. Smith and M. Zagha, "A comparison of sorting algorithms for the connection machine CM-2", [in:] Proceedings of the 3rd annual ACM Symposium on Parallel Algorithms and Architectures (SPAA91), Hilton Head, South Carolina, July, 1991, pp. 3-16.

[3] R. Cole, "Parallel merge sort", SIAM Journal Comput. No. 17, 1988, pp. 770-785.

[4] T.H. Cormen, C.E. Leiserson, R.L. Rivest and C. Stein, Introduction to algorithms. The MIT Press and McGraw-Hill Book Company, Cambridge, 2001.

[5] P. Crescenzi, R. Grossi and G.F. Italiano, "Search data structures for skewed strings. Experimental and Efficient Algorithms", Second International Workshop, WEA 2003, Ascona, Switzerland, May 26-28, 2003, No. 2647, Lecture Notes in Computer Science, Springer-Verlag, New York-Berlin Heidelberg, USA-Germany , 2003, pp. 81-96.

[6] R. Dlekmann, J. Gehring, R. Luling, B. Monien, M. Nubel and R. Wanka, "Sorting large data sets on a massively parallel system", [in:] Proceedings of the 6th Symposium on Parallel and Distributed Processing, IEEE, Los Alamitos, CA, USA, Oct. 1994, pp. 2-9.

[7] G. Gedigaa and I. Duntschb, "Approximation quality for sorting rules", Computational Statistics and Data Analysis, No. 40, 2002, pp. 499-526.

[8] M.S. Jeon and D.S. Kim, "Parallel Merge Sort with Load Balancing", International Journal of Parallel Programming, No.1 Vol.31, February 2003, pp. 21-33.

[9] D.E. Knuth, Sorting and Searching, volume 3 of The Art of Computer Programming. Addison-Wesley, Reading, MA, USA, second edition, 1998.

[10] A. LaMarca and R.E. Ladner, "The inuence of caches on the performance of sorting", [in:] Proceedings of the 8th Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, Louisiana, 5-7 January, 1997, pp. 370-379.

[11] A. LaMarca and R.E. Ladner, "The Inuence of Caches on the Performance of Sorting", [in:] Proceedings of Eighth Ann. ACM-SIAM Symposium on Discrete Algorithms, 1997.

[12] P. Larson, "External Sorting: Run Formation Revisited", IEEE Transactions on Knowledge and Data Engineering, No.4 Vol.15, 2003, pp. 961-972.

[13] Z. Marszałek and M. Woźniak, "On Possible Organizing Nosql Database Systems", International Journal of Information Science and Intelligent System, the Martin Science Publishing LTD., Vol.2, No.2, USA, 2013, pp. 51-59.

[14] V.S. Pai and P.J. Varman, "Prefetching with Multiple Disks for External Mergesort:Simulation and Analysis", [in:] Proceedings of Internetional Conference no Data Engineering, 1992, pp. 273-282.

[15] B. Salzberg, "Merging Sorted Runs Using Large Main Memory", Acta Informatica, No.3 Vol.27, 1989, pp. 195-215.

[16] R. Sinha and J. Zobel, "Cache-conscious sorting of large sets of strings with dynamic tries", J. Exp. Algorithmics, No.9, 2004, pp. 1-5.

[17] R. Trimananda and C.Y. Haryanto, "A Parallel Implementation of Hybridized Merge-Quicksort Algorithm on MPICH", [in:] Proceedings of 2010 International Conference on Distributed Framework for Multimedia Applications (DFmA).

[18] M.A. Weiss, Data Structure & Algorithm Analysis in C++, 2nd ed. Addison Wesley Longman, 1999.

[19] M. Woźniak and Z. Marszałek, Selected Algorithms for Sorting Large Data Sets, Silesian University of Technology Press, Gliwice, Poland, 2013.

[20] M. Woźniak, Z. Marszałek, M. Gabryel and R.K. Nowicki, "Modified Merge Sort Algorithm for Large Scale Data Sets", ICAISC 2013, Lecture Notes in Artifcial Intelligence, No. 7895, Part II, Springer-Verlag, New York-Berlin Heidelberg, USA-Germany, 2013, pp. 612-622.

[21] M. Woźniak, Z. Marszałek, M. Gabryel and R.K. Nowicki, "On quick sort algorithm performance for large data sets", Advances and Innovations in Computer Science, Springer-Verlag, New York-Berlin Heidelberg, USA-Germany, 2014, pp. (accepted-in press).

[22] M. Woźniak, Z. Marszałek, M. Gabryel and R.K. Nowicki, "On quick sort algorithm performance for large data sets", [in:] Proceedings of International Conference on Knowledge, Information and Creativity Support Systems, 7-9 November, Cracow, Poland, 2013, pp. 647-656.

[23] M. Woźniak, Z. Marszałek, M. Gabryel and R.K. Nowicki, "Triple heap sort algorithm for large data sets", [in:] Proceedings of International Conference on Knowledge, Information and Creativity Support Systems, 7-9 November, Cracow, Poland, 2013, pp. 647-656.

[24] M. Woźniak and Z. Marszałek, Extended Algorithms for Sorting Large Data Sets, Silesian University of Technology Press, Gliwice, Poland, 2014.

[25] M. Woźniak, W. M. Kempa, M. Gabryel, R. K. Nowicki and Z. Shao, "On applying evolutionary computation methods to optimization of vacation cycle costs in finite-buffer queue", ICAISC'2014, Lecture Notes in Artificial Intelligence, Springer-Verlag, PART I, vol. 8467, 2014, pp. 480-491.

[26] W. Zhang and P.A. Larson, "Dynamic Memory Adjustment for External Mergesort", Proc. Very Large Data Bases Conf., 1997, pp. 376-385.

[27] L. Zheng and P.A. Larson, "Buffering and Read-Ahead Strategies for External Merge-sort", [in:] Proceedings of Very Large Data Bases Conference, 1998, pp. 523-533.

[28] L. Zheng and P.A.Larson, "Speeding Up External Mergesort", IEEE Transactions on Knowledge and Data Engineering, Vol.8, No.2, 1996, pp. 322-332.