ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2014

Seria: MATEMATYKA STOSOWANA z. 4 Nr kol. 1920

Zbigniew MARSZAŁEK, Dawid POŁAP, Marcin WOŹNIAK

Institute of Mathematics
Silesian University of Technology

ON CLASSIC BUBBLE SORT PERFORMANCE FOR LARGE DATA SETS

Summary. In the paper we discuss performance of classic bubble sort algorithm for large data sets. Research results discussed and described in this article help to evaluate computer methods used in NoSQL database systems for large amounts of the input data. Therefore we try to analyze one of the most common sorting algorithms and its properties for large data sets.

O WYDAJNOŚCI KLASYCZNEJ WERSJI SORTOWANIA BĄBELKOWEGO DLA DUŻYCH ZBIORÓW DANYCH

Streszczenie. Artykuł ma na celu przedstawienie analizy wydajności algorytmu sortowania bąbelkowego w postaci klasycznej dla dużych zbiorów danych. Podjęty temat ma duże znaczenie dla rozwoju współczesnej informatyki ze względu na to, że komputery muszą pracować na coraz większych ilościach danych.

2010 Mathematics Subject Classification: 68W40, 68Q25, 68P01, 68P10.
Keywords: computer algorithm, large data sets, NoSQL database systems.
Corresponding author: Z. Marszałek (Zbigniew.Marszalek@polsl.pl).
Received: 30.06.2014.
This project has been financed from the funds of Institute of Mathematics, Silesian University of Technology.

1. Introduction

Performance of sorting algorithms is discussed in many publications devoted to computing systems, NoSQL databases and computer simulations. In [22] is presented analysis of some dedicated versions of sorting algorithms for large data sets. While in [21] is discussed possible extension and improvements of sorting methods for large data sets and NoSQL database systems. There are many research on the performance of sorting methods for large data sets (please see [16, 17, 24]). Here we discuss some aspects of classic bubble sort performance for large data sets.

While implementing software we often encounter the problem of lack of order in the information. Natural solution is to use a sorting algorithm to sort the information we have. Algorithms and data sorting problem is an important matter discussed in [1, 2, 11, 12, 19]. In the papers [3, 4, 7, 8] authors described various modifications and changes to sorting algorithms, which allow more efficient operating. Moreover, the authors of [5, 6, 9, 10] present new sorting methods with some interesting properties. The solutions and examinations presented in the following sections will help to analyze classic bubble sort algorithm.

2. Classic bubble sort

Classic bubble sort is one of sorting algorithms. Modifications of the method and the references to it's interesting derivatives are also described in [1, 2, 11, 12, 16, 19]. Authors of [5, 8, 13–15, 27] show also possibility of constructing adaptive or parallel algorithms to improve sorting and data mining. At the same time in [3, 6, 7, 18] are presented interesting solutions for special data structures.

Classic bubble sort algorithm is converting elements in pairs. The procedure swaps compared elements. The biggest one is moved to the end of sequence in following iterations, for more details on this method please see [21]. Therefore, in subsequent iterations, each time we consider decreased sequence. Described classic procedure is also presented in [1,2,11,12,19]. Many interesting implementations of his method and its derivatives are presented by the authors of [1, 2, 11, 12, 16, 19]. However by now there are no research made on it's performance for large data sets. Let us now present the examined classic bubble sort algorithm. Implementation of the algorithm was done using CLR standard for MS Visual Studio 2012. For more interesting examples of implementations of similar sorting methods see [1,2, 11,12,19]. In Figure 1 is presented block diagram of implemented version.

```
  Start
  Load data
  Set starting index $i$
1 if element with index $i$ is the last one then
      Return to 4
  else
      Return to 2
  end if
2 Start sorting
  Take next element with index $j$ to compare
  if element with index $j$ is not the last one then
      if next element is bigger then
          Swap elements
          Return to 3
      else
          Return to 3
      end if
3     Increase index $j++$
      Return to 2
  else
      Increase index $i++$
      Return to 1
  end if
4 Write sorted sequence
  Stop
```

To determine theoretical time complexity of the classic bubble sort algorithm, one should note that to sort n-element sequence we need to make $n-1$ comparisons in pairs. After all operations, the largest element is placed at the end of the sequence. Let us now consider the theoretical time complexity of the presented method.

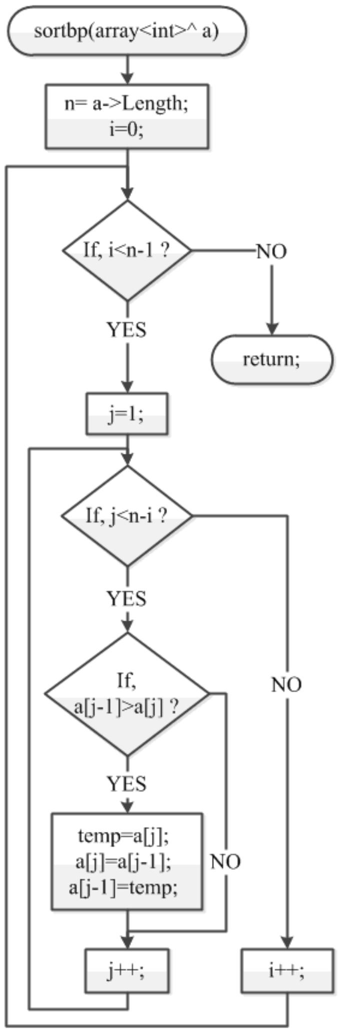

Fig. 1. The block diagram of implemented classic bubble sort algorithm
Rys. 1. Schemat blokowy zaimplementowango algrytmu sortowania bąbelkowego w postaci klasycznej

Theorem 1. *Classic bubble sort algorithm has theoretical time complexity $\vartheta(n^2)$.*

Proof. In subsequent steps, the number of necessary comparisons is reduced. Recording the number of comparisons, and therefore the complexity, we have the formula

$$\vartheta(n^2) = (n-1) + (n-2) + \ldots + 2 + 1 = n \cdot \frac{n-1}{2}, \tag{1}$$

where n means number of elements in the sequence. We can limit formula (1)

$$\frac{n^2}{4} \leqslant n \cdot \frac{n-1}{2} \leqslant \frac{n^2}{4}, n \geqslant 2 \tag{2}$$

We have estimated time of sorting by polynomials of equal degree. Thus theoretical time complexity is $\vartheta(n^2)$. $\square$

3. Experimantal results

Time complexity of this algorithm is similar to insertion sort algorithm, please see [23]. However, these two algorithms differ in the way of sorting and have a different sort time constant. Thus, it is important to test their performance in practice and verify theoretical assumptions. Methods for validity tests and analysis of algorithms that were used in this article are described by the authors of [3, 4, 10]. Some other methods are also presented in [16, 21, 22]. Bubble sort algorithm in classical form, as well as insertion sort, were tested to the level of 1000000 elements (see [23]). Above this number of elements using classic bubble sort takes considerable amount of time. Let us first analyze values of CPU clock cycles measured during the tests. The research results are plotted in Figures 2–3.

Chart of average CPU clock cycles in Figure 2 shows that it increases practically linear with the number of sorted elements. Thus, classic bubble sort algorithm in many situations may behave similarly to insertion sort described in [21, 23].

Shown in Figure 3 standard deviation and average deviation of CPU clock cycles estimate possible changes in the examined values. Analysis of charts shows that for large data sets, the algorithm can slow down process of sorting. We conclude that for sequences of over 1000000 elements one should use other algorithms. Presented statistical study of CPU clock cycles is not sufficient description of classic bubble sort algorithm. Therefore for a broader description of studied algorithm authors of [3, 4, 10, 16, 21, 22] propose some other factors. One of them is sorting time. Measured results were plotted in Figures 4–5.

Standard deviation and average deviation shown in Figure 5 confirm that the bubble sort algorithm for large data sets can behave with potential variability of execution time. Moreover the algorithm behaves unstable for sets of size close to 10000. Coefficients of variation of classic bubble sort algorithm, as described in [3,4,10], are shown in Figure 6. The values of the coefficients of variation shown in Figure 6 were approximated by polynomials. Characteristic curves derived from the variation of classic bubble sort algorithm are shown in Figure 6. The lowest volatility is characterized by the time, while the largest changes may in CPU clock cycles. Such properties are related to the possibility of more efficient use of the processor or code optimization. The result curves of variation indicate the possibility of efficient use of this algorithm in the range of 100 to 10000 elements.

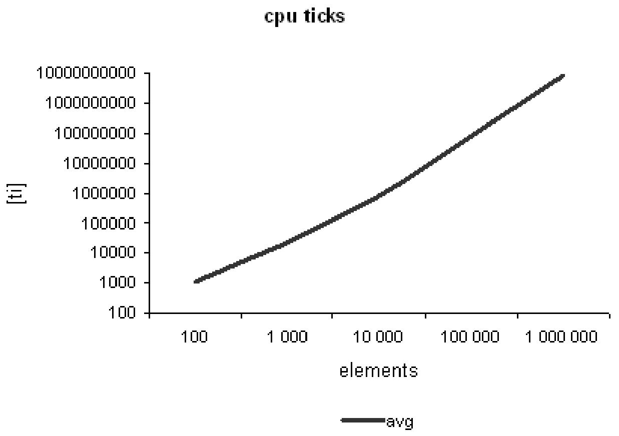

Fig. 2. The average number of CPU clock cycles during classic bubble sort

Rys. 2. Średnia liczba cykli zegarowych procesora w trakcie sortowania bąbelkowego w postaci klasycznej

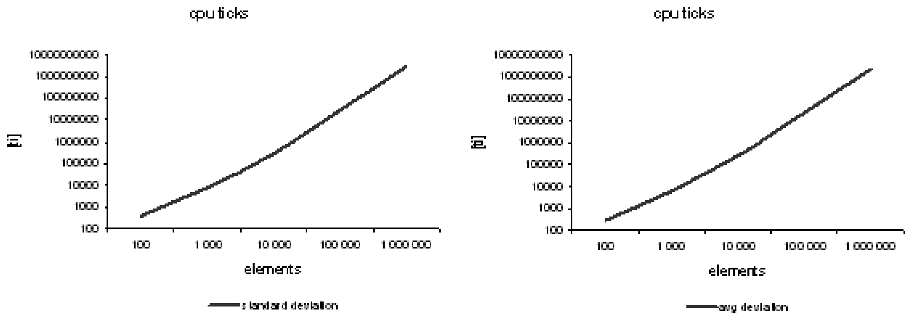

Fig. 3. Chart of the standard deviation and the average deviation of the results

Rys. 3. Wykres odchylenia standardowego oraz odchylenia średniego wyników przeprowadzonych badań

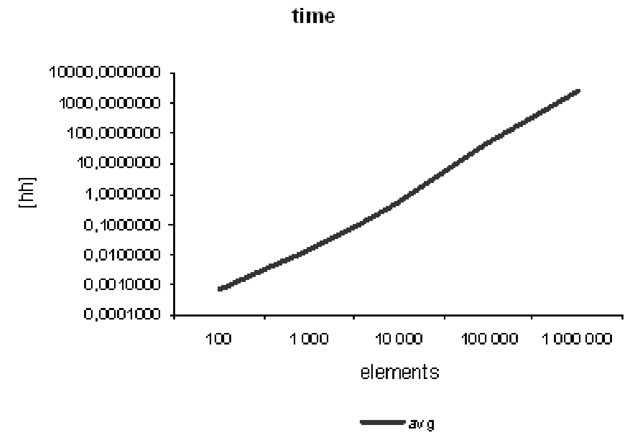

Fig. 4. The average time of classic bubble sort

Rys. 4. Średnia liczba cykli zegarowych procesora w trakcie sortowania bąbelkowego w postaci klasycznej

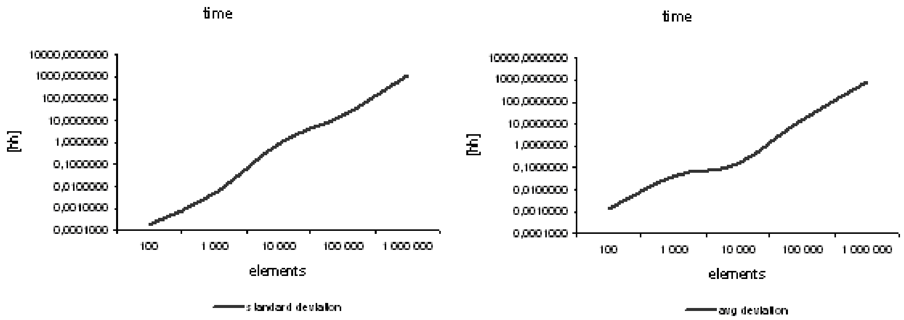

Fig. 5. Charts of the standard deviation and the average deviation of results

Rys. 5. Wykres odchylenia standardowego oraz odchylenia średniego wyników badań

4. Conclusions

In conclusion, we see that the presented in [20, 21] version of the algorithm with logic control of order can be more efficient for large data sets and allows acceleration of the sorting. This was verified in tests and comparisons. There are also some dedicated methods designed for large data sets and NoSQL database systems. In [24] is discussed special version of modified merge sort. While in [16] is presented fast merging for NoSQL systems. moreover in [25] and [26] are presented dedicated quick sort and heap sort respectively. Further research will concentrate on extensions of sorting methods, similar to these presented in [22] and performance improvements like these discussed in [16,17].

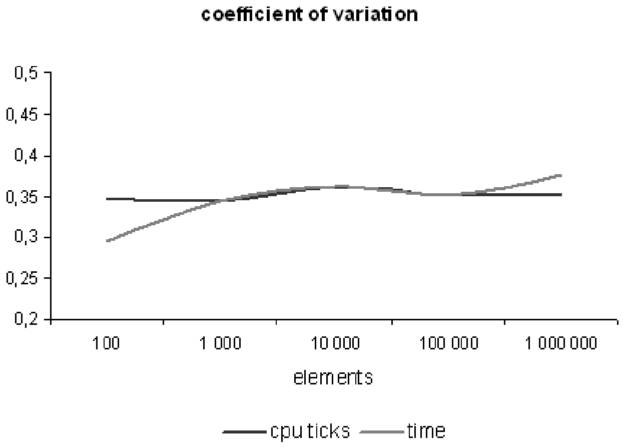

Fig. 6. Polynomial approximation of coefficient of variation for classic bubble sort
Rys. 6. Aproksymacja wielomianowa współczynnika zmienności dla sortowania bąbelkowego w postaci klasycznej

References

1. Aho I.A., Hopcroft J., Ullman J.: *The Design and Analysis of Computer Algorithms.* Addison-Wesley, Indianapolis 1974.
2. Banachowski L., Diks K., Rytter W.: *Algorithms and Data Structures.* WNT, Warszawa 1996 (in Polish).
3. Bentley J.L., Stanat D.F., Steele J.M.: *Analysis of a randomized data structure for representing ordered sets.* In: Proceedings of the 19th Annual Allerton Conference on Communication, Control and Computing, University of Illinois, 364–372.
4. Brown M.R., Tarjan R.E.: *Design and analysis of data structures for representing sorted lists.* SIAM J. Comput. **9** (1980), 594–614.
5. Carlsson S., Chen J.: *An optimal parallel adaptive sorting algorithm.* Inform. Process. Lett. **39** (1991), 195–200.
6. Cook C.R., Kim. D.J.: *Best sorting algorithms for nearly sorted lists.* Comm. ACM **23** (1980), 620–624.

7. Dinsmore R.J.: *Longer strings for sorting.* Comm. ACM **8** (1965), 48–65.
8. Dlekmann R., Gehring J., Luling R., Monien B., Nubel M., Wanka R.: *Sorting large data sets on a massively parallel system.* In: Proceedings of the 6th IEEE Symposium on Parallel and Distributed Processing, Dallas 1994, 29–38.
9. Estivill-Castro E., Wood D.: *A genetic adaptive sorting algorithms.* Comput. J **35** (1992), 505–512.
10. Islam T., Lakshman K.B.: *On the error sensitivity of sort algorithms.* In: Proceedings of International Conference on Computing and Information, Toronto 1990, 81–85.
11. Knuth D.: *The Art of Computer Programming*, vol. 1–3. Addison-Wesley Professional, Indianapolis 2006.
12. Knuth D.E., Greene D.H.: *Mathematics for the Analysis of Algorithms*, Birkhäuser, Boston 2007.
13. Levcopolos C., Petersson O.: *A note on adaptive parallel sorting.* Inform. Process. Lett. **33** (1985), 187–191.
14. Levcopolos C., Petersson O.: *An optimal parallel algorithm for sorting presorted files.* Lecture Notes in Comput. Sci. **338** (1988), 154–160.
15. Levcopolos C., Petersson O.: *Splitsort an adaptive sorting algorithm.* Inform. Process. Lett. **39** (1991), 205–211.
16. Marszałek Z., Połap D., Woźniak M.: *On preprocessing large data sets by the use of triple merge sort algorithm.* Proceedings of the International Conference on Advances in Information Processing and Communication Technologies (IPCT 2014), The IRED – Digital Seek Library, Santa Barbara 2014, 65–72.
17. Marszałek Z., Woźniak M.: *On possible organizing Nosql database systems.* Int. J. Information Science and Intelligent System **2**, no. 2 (2013), 51–59.
18. Sinha R., Zobe J.: *Cache-conscious sorting of large sets of strings with dynamic tries.* J. Exp. Algorithmics **9** (2004), article no. 1.5.
19. Weiss M.A.: *Data Structures and Algorithm Analysis in C++.* Prentice Hall, Indianapolis 2013.
20. Woźniak M., Marszałek Z.: *On some properties of bubble sort with logic control of order for large scale data sets.* Zesz. Nauk. PŚl., Mat. Stosow. **3** (2013), 47–58.
21. Woźniak M., Marszałek Z.: *Selected Algorithms for Sorting Large Data Sets.* Wyd. Pol. Śl. (Silesian University of Technology Press), Gliwice 2013.
22. Woźniak M., Marszałek Z.: *Extended Algorithms for Sorting Large Data Sets.* Wyd. Pol. Śl. (Silesian University of Technology Press), Gliwice 2014.

23. Woźniak M., Marszałek Z., Gabryel M.: *The analysis of properties of insertion sort algorithm for large data sets.* Zesz. Nauk. PŚl., Mat. Stosow. **2** (2012), 45–55.
24. Woźniak M., Marszałek Z., Gabryel M., Nowicki R.K.: *Modified merge sort algorithm for large scale data sets.* Lecture Notes in Artificial Intelligence **7895** (2013), 612–622.
25. Woźniak M., Marszałek Z., Gabryel M., Nowicki R.K.: *On quick sort algorithm performance for large data sets.* In: Looking into the Future of Creativity and Decision Support Systems, Skulimowski A.M.J. (ed.), Progress & Business Publishers, Cracow 2013, 647–656.
26. Woźniak M., Marszałek Z., Gabryel M., Nowicki R.K.: *Triple heap sort algorithm for large data sets.* In: Looking into the Future of Creativity and Decision Support Systems, Skulimowski A.M.J. (ed.), Progress & Business Publishers, Cracow 2013, 657–665.
27. Zheng S.Q., Calidas B., Zhang Y.: *An efficient general in-place parallel sorting scheme.* J. Supercomput. **14** (1999), 5–17.