ZESZYTY NAUKOWE POLITECHNIKI ŚLĄSKIEJ 2012
Seria: MATEMATYKA STOSOWANA z. 2 Nr kol. 1874

Marcin WOŹNIAK[1], Zbigniew MARSZAŁEK[1], Marcin GABRYEL[2]
[1]Institute of Mathematics
Silesian University of Technology
[2]Department of Computer Engineering
Czestochowa University of Technology

THE ANALYSIS OF PROPERTIES OF INSERTION SORT ALGORITHM FOR LARGE DATA SETS

Summary. Insertion sort algorithm is one of the sorting algorithms. It is characterized by the computational complexity and time complexity, which represent the possibility of using it for large data sets. The present work is to describe this algorithm and describe it's performance when sorting large scale data sets.

ANALIZA WŁASNOŚCI ALGORYTMU SORTOWANIA PRZEZ WSTAWIANIE DLA DUŻYCH ZBIORÓW DANYCH

Streszczenie. Algorytm sortowania przez wstawianie jest jednym z algorytmów opisywanych w literaturze. Omawiana metoda została scharakteryzowana poprzez złożoność czasową i obliczeniową algorytmu, która opisuje możliwość stosowania tego algorytmu do sortowania dużych zbiorów danych. Praca ta ma na celu opisanie zachowania algorytmu i jego wydajności dla dużych zbiorów danych.

2010 Mathematics Subject Classification: 68W40, 68Q25, 68P01, 68P10.
Wpłynęło do Redakcji (received): 14.07.2012 r.

1. Defining the algorithm

Insertion sort algorithm compares data by inserting the individual characters sorted. It selects found item and inserts it at the end of the string. Then, the sorting is subjected to a new, remaining string. As the largest element is already at the end of the string, the procedure comparison is only for $n-1$ elements of the original string. This procedure is repeated until the one element remain. One-piece string is understood as the smallest of all the elements, in a similar way the authors describe it in the literature [2, 3, 1]. The insertion sort algorithm is:

1. Start.
2. Load sequence to be sorted.
3. Assume i as the first place from the end of the sorted.
4. Make sure if the loaded string is empty.
   (a) If you loaded the string is empty, then:
      i. go to Step 4,
   (b) f you loaded the string is not empty, then:
      i. go to Step 5.
5. Start sorting.
   (a) Take as the maximum the first element of the sorted string.
   (b) Get the next element.
   (c) Make sure if the downloaded item is the last in the tested string.
      i. If you are on the last element in the sorted sequence, then:
         A. Create a new string to sort of length t which is less than the loaded $i+1$.
         B. Found maximum swap with last element in the sorted string.
         C. Increase the index i, which denotes the location of the next maximum from the end of the main string.
         D. Forward to sort the newly created string.
         E. Go to Step 4.
      ii. If you are the last element in the sorted sequence, then:

A. Check if another element is greater than the accepted maximum.

B. If so, then:
   - Accept this as a maximum element.
   - Receive t as a value index, where found a new maximum.

C. If not, then:
   - Go to Step 5.b.

6. Print the sorted string.

7. Stop.

Now we consider the complexity of the insertion sort algorithm. If we consider duration of the program on the size of the given task we can determine a time complexity of the algorithm, as described in literature [2, 1, 5]. The time that we need for the implementation of the program is written as the symbol $\vartheta\left(n^2\right)$. Such a record means that if the dimension of the task will increase two times, it will need four times more for it's execution. In practice we are not interested in the exact runtime function containing a constants dependent on the computer, but only estimation of the time of the algorithm from the bottom and top. We define only factor that determines how long the algorithm is executed. Of course, this estimation is valid for sufficiently large n and we define it as asymptotic complexity. In fact, it may be that for the small dimension of the tasks constants imposed by the algorithm can play a crucial role, what is discussed in literature [4, 1, 5]. The algorithm of the lower asymptotic complexity can be running longer than the algorithm with higher computational complexity expressed as $\vartheta\left(n^2\right)$. To determine the time complexity of the insertion sort algorithm should be noted that the search for the greatest numbers in the n-element string needs to make $n-1$ comparisons. After shifting the largest element at the end in the next iteration the number of necessary comparisons is reduced by one. Recording the number of necessary comparisons in subsequent iterations - the complexity of the time, we get the formula

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

where n means the number of elements in the present task, so it's dimension. We restrict now the time complexity function (1) as follows

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

In this way, we estimate the time of the algorithm from below and from above by polynomials of the same degree. The operating time factor determines n^2. We infer from the fact that this sort algorithm has time complexity $\vartheta\left(n^2\right)$. Thus, time complexity is an increasing function in the same way as ordinary parabola with the increase of data. Overall, we are interested in reducing the time of the algorithm and finding algorithms with complexity of $\vartheta\left(n \cdot log_2 n\right)$, which are described as the fastest in literature [2, 4, 5]. Similarly, the complexity of the memory footprint will be a function in memory, depending on the dimension of the task. The complexity of memory is a linear function of the form

$$f\left(n\right) = a \cdot n + c, \tag{3}$$

where the following symbols mean: n - dimension of the task. In our case the dimension of the task is the amount of numbers that we have to be sorted, ie. n elements of the sorted string, a - constant for the number of bytes required to remember the integer. For integers a long int is equal to 4, c – constant dependent on the size of the program, which for different compilers running under different operating systems is different, as a constant for our purpose we can ignore it.

The function described by the formula (3) is a description footprint in memory by the insertion sort algorithm. It stems from the need to memorize a vector of numbers to be sorted and fixed number of variables used to construct an algorithm. The shape of this function is based on the declaration of the array and variables. Both the complexity can be treated as independent attributes of this algorithm for characterizing the rate of work and the amount of required space on your computer. For example, we can show how to reduce the duration of the program by using more memory.

2. The analysis of properties

The insertion sort algorithm is not highly efficient algorithm, which also draws attention to literature [2, 4, 3, 1, 5]. It's performance has been tested only to the level of 1000000 elements. Above this number of elements using insertion sort takes a considerable amount of time. We can talk about a considerable increase in time complexity. The present algorithm has been described by the characteristics of the CPU clock cycles, real-time and time. These characteristics have been plotted on a logarithmic scale to facilitate the analysis of properties of the insertion sort algorithm.

Table 1
Table of values of the characteristics of the CPU clock cycles for the insertion sort algorithm

cpu tics	Number of sorted elements		
[ti]	100	1000	10000
avg	1000,6	10722,4	404794
standard deviation	213,6124996	2142,837325	65894,69603
avg deviation	149,04	1717,28	45036
coefficient of variation	0,2134844	0,1998468	0,1627858
variation area upper	786,9875004	8579,562675	338899,304
variation area lower	1214,2125	12865,23732	470688,696
[ti]	100000	1000000	
avg	37104280,8	3694885205	
standard deviation	6769240,094	661175967,3	
avg deviation	5386017,36	533498159,9	
coefficient of variation	0,1824383	0,1789436	
variation area upper	30335040,71	3033709237	
variation area lower	43873520,89	4356061172	

The results presented in Table 1 were plotted for the calculated characteristic values describing the results, which show Figures 1–2.

Figure of arithmetic average number of CPU clock cycles shows that the number is increasing as almost linear increment with the number of sorted elements.

The size of the average number of clock cycles is shown in Figure 1 is related to the standard deviation and the deviation of the average shown in Figure 2. The results shown in Figure 1 suggest as shown in formula (1) that the algorithm is to harvest bigger than 1 million elements can significantly prolong the duration of it's action. In Table 2 is shown to develop real-time statistics on how the items were sorted during the tests performed.

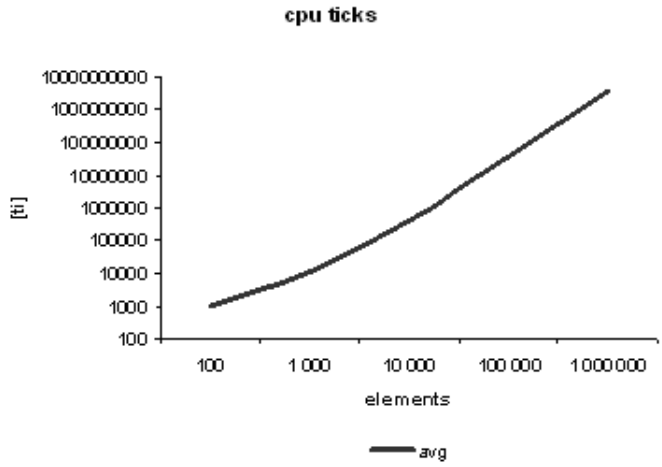

Fig. 1. Average number of CPU clock cycles during the sorting
Rys. 1. Średnia ilość cykli zegarowych procesora w trakcie operacji sortowania

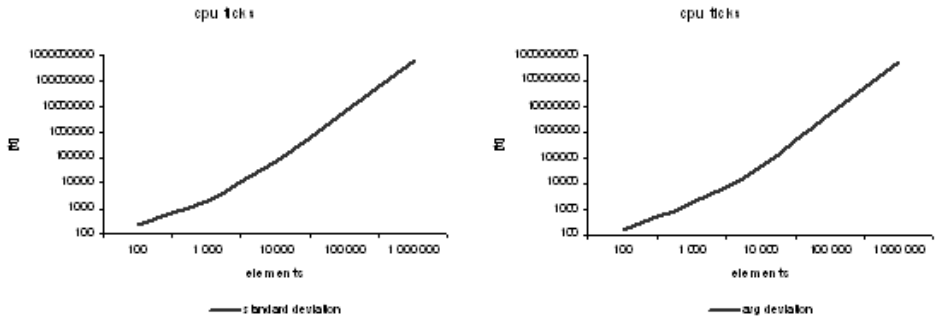

Fig. 2. Graph of the avg deviation and the standard deviation of the results obtained during testing
Rys. 2. Wykres odchylenia standardowego i średniego ilości cykli zegarowych procesora jakie uzyskano w trakcie testów

Table 2
Table of values of real-time characteristics for the insertion sort algorithm

real time	Number of sorted elements		
[hh]	100	1000	10000
avg	0:0:0.0006419	0:0:0.0068791	0:0:0.2596993
standard deviation	0:0:0.0001371	0:0:0.0013747	0:0:0.0422757
avg deviation	0:0:0.000956	0:0:0.011016	0:0:0.288934
coefficient of variation	0,2135847	0,1998372	0,1627871
variation area upper	0:0:0.000779	0:0:0.0082538	0:0:0.6824563
variation area lower	0:0:0.0005048	0:0:0.0055044	0:0:0.1630577
[hh]	100000	1000000	
avg	0:0:23.8045918	6:39:30.4953151	
standard deviation	0:0:4.3428855	1:7:4.2004747	
avg deviation	0:0:3.554729	0:5:42.823114	
coefficient of variation	0,182439	0,1789501	
variation area upper	0:0:28.1474773	7:46:34.6957898	
variation area lower	0:0:19.4617063	5:32:26.2948404	

The results presented in Table 2 were plotted with the calculated values of characteristic, as shown in Figure 3–4.

Graph of real-time arithmetic average shows that the number is increasing almost linear increment with the number of sorted elements. This phenomenon is similar in nature to the clock cycles as shown in Figure 1.

Standard and avg deviation is shown in Figure 4. Graphs of these characteristics show that the algorithm for larger strings can be characterized by variability of execution time. This confirms the beginning suggestion. Therefore, for strings larger than 1000000 items should be used other sorting algorithm. In Table 3 is shown statistical study of the results for the time clock in which elements of strings were sorted.

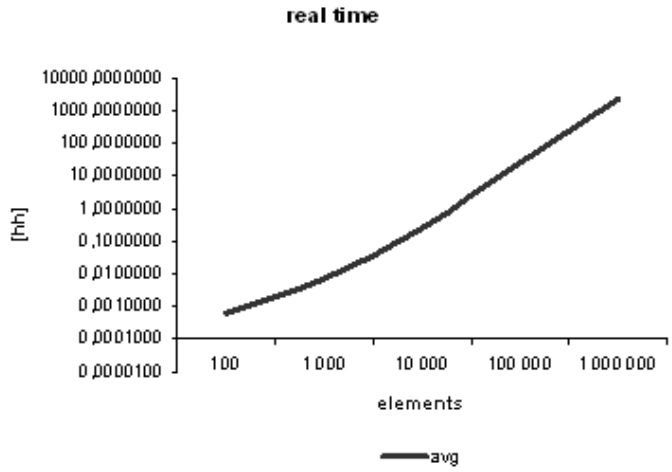

Fig. 3. The average real-time during the sorting
Rys. 3. Średni czas rzeczywisty w trakcie operacji sortowania

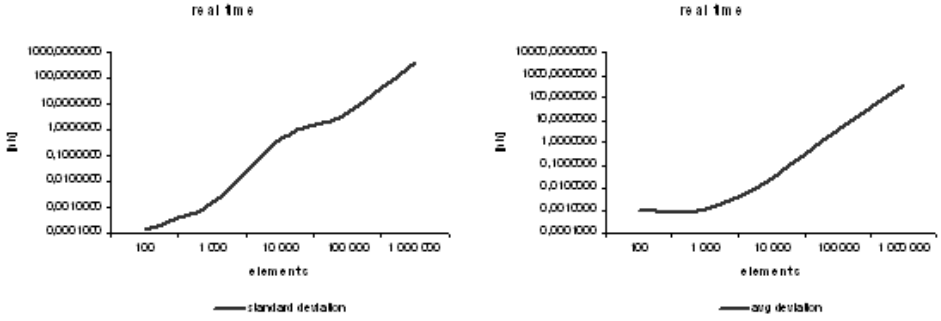

Fig. 4. Graph of the avg deviation and the standard deviation of the results obtained during testing
Rys. 4. Wykres odchylenia standardowego i średniego czasu rzeczywistego jaki uzyskano w trakcie testów

Table 3
Table of values of the characteristics of the clock time for the insertion sort algorithm

clock time	Number of sorted elements		
[ms]	100	1000	10000
avg	0	6,4	259,2
standard deviation	0	1,1401754	42,038078
avg deviation	0	0,88	28,72
coefficient of variation	0	0,1781524	0,1621839
variation area upper	0	7,5401754	301,238078
variation area lower	0	5,2598246	217,161922
[ms]	100000	1000000	
avg	23784,2	2370494,8	
standard deviation	4363,292301	424200,3286	
avg deviation	3471,44	342282,16	
coefficient of variation	0,1834534	0,1789501	
variation area upper	28147,4923	2794695,129	
variation area lower	19420,9077	1946294,471	

The coefficients of variation of the characteristics of insertion sort algorithm described in Tables 1–3 are shown in Figure 5.

3. Conclusions

Analysis of the coefficient of variation shows that the algorithm is able to perform in an acceptable time of sorting up to 1,000 items. However, in practice we are dealing with such small collections. The coefficients of variation brought closer by using polynomial approximations. The resulting curves show the characteristics of Figure 6, where the approximation characteristics of the CPU clock cycles and real-time practically coincide. Variability characteristic curves confirm the possibility of effective application of the algorithm by putting in the range up to 1,000 items. The results that were obtained during the tests suggest that for the larger sets use other sorting algorithms.

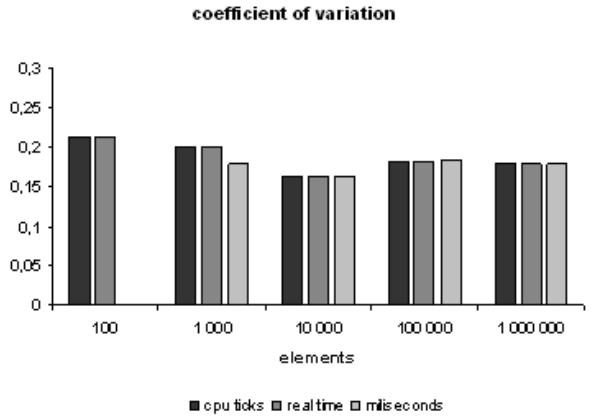

Fig. 5. Coefficient of variation for the number of CPU clock cycles, real-time and milliseconds that were obtained during performed tests
Rys. 5. Współczynnik zmienności dla cykli zegarowych procesora, czasu rzeczywistego oraz czasu zegarowego jakie uzyskano w trakcie wykonanych testów

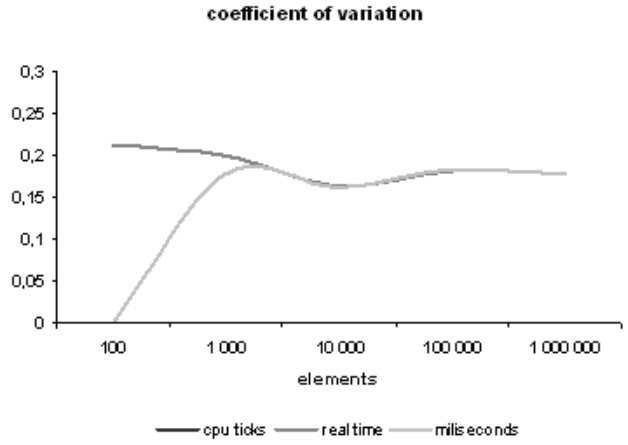

Fig. 6. Characteristics of coefficient of variation for number of CPU clock cycles, real-time and milliseconds based on polynomial approximation
Rys. 6. Wykres przybliżenia krzywej współczynnika zmienności dla cykli zegarowych procesora, czasu rzeczywistego oraz czasu zegarowego poprzez aproksymację wielomianową

References

1. Aho A.V., Ullman J.D., Hopcroft J.E.: *Data Structures and Algorithms.* Addison Wesley, Boston 1983.
2. Cormen T.H., Leiserson Ch.E., Rivest R.L., Stein C.: *Wprowadzenie do algorytmów.* WNT, Warszawa 2007.
3. Mehlhorn K., Sanders P.: *Algorithms and Data Structures.* Springer, Berlin 2008.
4. Sedgewick R.: *Algorithms in C, Parts 1-5 (Bundle).* Addison-Wesley, Boston 2001.
5. Shaffer C.A.: *Data Structures and Algorithm Analysis in C++.* Dover Publ., New York 2011.

Omówienie

W artykule przedstawiono metodę sortowania poprzez wstawianie. Opisano algorytm pozwalający wykonać sortowanie oraz określono złożoność omówionej metody. Metoda została przetestowana dla dużych zbiorów danych. Na podstawie wyników jakie uzyskano w trakcie przeprowadzonych badań wykonano analizę własności metody sortowania poprzez wstawianie. Wyniki opracowania statystycznego pozwoliły określić szybkość omówionej metody dla dużych zbiorów danych i zweryfikować jej stabilność. Wyniki analizy pozwoliły wprowadzić modyfikacje zwiększające efektywność.