FRI-2G.303-1-CST-03

SORTING DATA WITH TABLES OF INVERSIONS WITHOUT MOVES

Assoc. Prof. Nayden Vasilev, PhD
Department of Computer Systems and Technologies,
Technical University of Sofia, Plovdiv Branch, Bulgaria
Tel.:
E-mail: mnvasilev@yahoo.com

Assoc. Prof. Atanaska Bosakova-Ardenska, PhD
Department of Computer Systems and Technologies, University of Food Technologies, Plovdiv, Bulgaria
Phone: 032-603 860
E-mail: a_bosakova@uft-plovdiv.bg

Assoc. Prof. Nikolay Shopov, PhD
Department of Computer Systems and Technologies, University of Food Technologies, Plovdiv, Bulgaria
Phone: 032-603 775
E-mail: nikshop@abv.bg

Abstract: *The paper discuses tables of left and right inversions. These tables could be used to define ascending or descending order of data. It is proposed one algorithm for sorting which uses tables of left and right inversions without moves. This algorithm is named SortLRT. The theoretical evaluation of time complexity of SortLRT is made. This algorithm is implemented in C and an experimental evaluation of time complexity is also made. The results show that the proposed algorithm SortLRT is faster than Bubble sort and Selection sort but slower than Insertion sort and Quick sort algorithms.*

Keywords: *Sorting without moves, Tables of invertions (Invertion vector), Sorting algorithms.*

INTRODUCTION

Search operation could be defined as an answer of the question "Is this value part of this set of values?"There are two possible answers: Yes or No (binary logic). Depending on the searched operation result (answer), a lot of operations could be performed more effectively. Performance of searching operation depends on the type of data sets (it's structure) and data ordering. If data are sorted in ascending or descending order then the searching could be performed in fastest way. Because of this statement, the effective sorting algorithms are very important. The most popular data structure is array (static or dynamic) and therefore a lot of searching and sorting algorithms in arrays are developed (Knuth D. 1973; Stoichev St. 2008; Cormen T. H., Leiserson Ch. E., Rivest R. L., Stein C. 2001; Wirth N. 1976; Sedgewick R. 1984).

This paper is part of a research in the field of using tables of inversions in sorting operations. Four algorithms for sorting which use table of left inversions are developed (Vasilev N. & Bosakova-Ardenska A. 2012). These algorithms use table of left inversions and move elements from left to right or from right to left. Current research continues working over sorting with tables of inversions but introduces some new elements. These elements are:

- table of right inversions;
- sorting without move operations.

EXPOSITION

Sorting With Table of Left Inversions Without Moves

Table of left inversions (by positions) contains integers which present numbers of elements (d_j) bigger than j-th element and their indices are smaller than j (j= 1, 2,…n) (Vasilev N. & Bosakova-Ardenska A. 2012).

Statement 1. Table of left inversions by positions for row a_j, j = 1, 2,…, n defines unambiguously ascending or descending order for the row.

Proof. The value d_n defines position p_n of element a_n in the sorted row. If the row has ascending order then $p_n = n - d_n$. If the row has descending order then $p_n = d_n + 1$. The position p_n should be excluded and positions after it should be renumberd. The last position is n - 1.

The value d_{n-1} defines positon p_{n-1} of element a_{n-1} in the sorted row. If the row has ascending order then $p_{n-1} = n - 1 - d_{n-1}$. If the row has descending order then $p_{n-1} = d_{n-1} + 1$. The position p_{n-1} should be excluded and positions after it should be renumberd. The last position is n - 2.

The value of d_j defines position p_j of element a_j, j = n, n-1, …, 2, 1 in the sorted row. If the row has ascending order then $p_j = j - d_j$. If the row has descending order then $p_j = d_j + 1$. The position p_j should be excluded and positions after it should be renumbered. The last position is n – j.

By this way every element will appear in the exact position of the sorted row.

Example: Let it is given the row 50, 60, 10, 12, 18, 33, 92, 16 and let implement sorting in ascending order. The table 1 presents table of left inversions for the given row.

Table 1. Table of left inversions by positions

Starting indices (**j**)	1	2	3	4	5	6	7	8
Values of $\mathbf{a_j}$	50	60	10	12	18	33	92	16
Number of left elements which are bigger than $\mathbf{a_j}$ - $\mathbf{d_j}$	0	0	2	2	2	2	0	5

The element a_8 is written in the position $p_8 = 8 - d_8 = 8 - 5 = 3$. The value in the new position 3 is excluded of sorting procedure. Therefore the indices are 7 and indices bigger than 3 should be decremented.

The element a_7 is written in the position $p_7 = 7 - d_7 = 7 - 0 = 7$. The value in the new position 7 is excluded of sorting procedure. Therefore the indices are 6 and indices are the same as in previous step.

The element a_6 is written in the position $p_6 = 6 - d_6 = 6 - 2 = 4$. The value in the new position 4 is excluded of sorting procedure. Therefore the indices are 5 and indices bigger than 4 should be decremented.

The element a_5 is written in the position $p_5 = 5 - d_5 = 5 - 2 = 3$. The value in the new position 3 is excluded of sorting procedure. Therefore the indices are 4 and indices bigger than 3 should be decremented.

The element a_4 is written in the position $p_4 = 4 - d_4 = 4 - 2 = 2$. The value in the new position 2 is excluded of sorting procedure. Therefore the indices are 3 and indices bigger than 2 should be decremented.

The element a_3 is written in the position $p_3 = 3 - d_4 = 3 - 2 = 1$. The value in the new position 1 is excluded of sorting procedure. Therefore the indices are 2 and indices bigger than 1 should be decremented.

The element a_3 is written in the position $p_2 = 2 - d_2 = 2 - 0 = 2$. The value in the new position 2 is excluded of sorting procedure. The index is only one and it is the same as in previous step.

The element a_1 is written in position $p_1 = 1 - d_1 = 1 - 0 = 1$.

Table 2. Sorting in ascending order for row 50, 60, 10, 12, 18, 33, 92, 16

End positions (**j**)	1	2	3	4	5	6	7	8
Writing of $\mathbf{a_8}$			16					
Writing of $\mathbf{a_7}$			16					92
Writing of $\mathbf{a_6}$			16		33			92
Writing of $\mathbf{a_5}$			16	18	33			92
Writing of $\mathbf{a_4}$		12	16	18	33			92
Writing of $\mathbf{a_3}$	10	12	16	18	33			92
Writing of $\mathbf{a_2}$	10	12	16	18	33		60	92
Writing of $\mathbf{a_1}$	10	12	16	18	33	50	60	92
Sorted values	10	12	16	18	33	50	60	92

Sorting With Tables of Left and Right Inversions Without Moves

Table of right inversions (by positions) contains integers which present numbers of elements (d_j) bigger than j-th element and their indices are smaller than j (j= 1, 2,…n) (Vasilev N. & Bosakova-Ardenska A. 2012).

Statement 2. The element a_j, j=1, 2, …, n of the given row is in position $\mathbf{p_{ja} = j + r_{ja}}$ in the ascending order and $\mathbf{r_{ja} = c_j - d_j}$.

Proof. Let c_j-th elements of the given row, which are smaller than a_j is moved to the left side of a_j and d_j-th elements, which are biger than a_j is moved to the right side of a_j. It is obviously that the element is on $\mathbf{r_{ja} = c_j - d_j}$ positions from j-th position and it's position in ascending order is $\mathbf{p_{ja} = j + r_{ja}}$ (j= 1, 2,…**n**).

Statement 3. The element a_j, j=1, 2, …, n of the given row is in position $\mathbf{p_{jd} = j + r_{jd}}$ in the descending order and the shift is $\mathbf{r_{jd} = n - j - c_j - (j - 1 - d_j)}$.

The proof is similar to proof of Statement 3.

Therefore position of element a_j after building of inversions tables is:

- for ascending order: $\mathbf{p_{ja} = j + c_j - d_j}$;
- for descending order: $\mathbf{p_{jd} = n + 1 - j - c_j + d_j}$.

The statements 2 and 3 present a way of building rows in ascending or descending order.

Example: Let it is given the row 60, 70, 50, 30, 20, 100, 40, 110, 10, 80 and let implement sorting in ascending and descending order. The next two tables demostrate sorting.

Table 3. Tables of left and right inversions, shifts ($\mathbf{r_{ja}}$, $\mathbf{r_{jd}}$) and positions $\mathbf{p_{ja}}$, $\mathbf{p_{jd}}$ for elements in sorted rows

Start positions (**j**)	1	2	3	4	5	6	7	8	9	10
Value of $\mathbf{a_j}$	60	70	50	30	20	100	40	110	10	80
$\mathbf{d_j}$ – number of left elements, bigger than $\mathbf{a_j}$	0	0	2	3	4	0	4	0	8	2
$\mathbf{c_j}$ – number of right elements smaller than or equal to $\mathbf{a_j}$	5	5	4	2	1	3	1	2	0	0
Shift value $\mathbf{r_{ja} = c_j - d_j}$	5	5	2	-1	-3	3	-3	2	-8	-2
New position $\mathbf{p_{ja} = j+c_j-d_j}$ (ascending order)	**6**	**7**	**5**	**3**	**2**	**9**	**4**	**10**	**1**	**8**
Shift value $\mathbf{r_{jd}=n-j-c_j-(j-1-d_j)}$	4	2	3	4	4	-4	0	7	1	-7
New position $\mathbf{p_{jd} = n+1-j-c_j+d_j}$ (descending order)	**5**	**4**	**6**	**8**	**9**	**2**	**7**	**1**	**10**	**3**

Table 4. Sorting in ascending and descending order for row 60, 70, 50, 30, 20, 100, 40, 110, 10, 80

Start positions (**j**)	9	5	4	7	3	1	2	10	6	8
New positions	1	2	3	4	5	6	7	8	9	10
Ascending row	10	20	30	40	50	60	70	80	100	110
Start positions (**j**)	8	6	10	2	1	3	7	4	5	9
New positions	1	2	3	4	5	6	7	8	9	10
Descending row	110	100	80	70	60	50	40	30	20	10

Evaluation of Sorting Algorithms with Inversions Tables without Moves

An evaluation of time complexity for the proposed algorithms is made.

Evaluation of algorithm for sorting with table of left inversions without moves

The proposed algorithm uses next shown operations:

1) Comparisons – for table of left inversions building (for counting elements bigger than current element);

2) Writes – for counting the elements bigger than current element;

3) Arithmetics – for calculation the new positions (indices) for elements;

4) Writes – for building the sorted row.

The number of comparisons for counting elements bigger than current element is $\mathbf{C = n(n-1)/2}$. The number of writes for counting the elements bigger than current element is:

$$\mathbf{R} = \sum_{\mathbf{j=1}}^{\mathbf{n}} \mathbf{d_j} \tag{1}$$

The number of arithmetic operations is:

- minimal $\mathbf{N_{min} = 0}$ when the sorting is in ascending order;

- maximal $\mathbf{N_{max} = (n–2)+(n–3)+…+3+2 = (n+1)(n–2)/2}$ when the sorting is in descending order.

The number of write operations for building the sorted row is **n**. If t_1 is time for execution an operation compare, t_2 – time for execution an operation write, t3 – time for calculation a new position for j-th element, t4 – time for execution an operation write into the new row, then time for building the sorted row is:

$$\mathbf{T_{ot} = t_1 n\,(n-1)/2 + t_2 \sum_{j=1}^{n} d_j + \sum_{j=1}^{n} t_{3j} + t_4 n} \tag{2}$$

The times t_{3j}, j=1, 2, …, n are different for different elements. Calculations for finding new position of current element contain:

- mark occupied positions in the row;

- counting free positions to find new position for current element.

It is obvious that number of operations for calculating the new position for current element is big and various. Because of this the algorithm is unacceptable in comparison with well known good sorting algorithms.

Evaluation of algorithm for sorting with tables of left and right inversions without moves

The discussed algorithm uses next shown operations:

1) Comparisons – for tables of inversions building (for counting elements bigger and smaller than current element);

2) Writes – for counting the elements bigger and smaller than current element;

3) Sums – for calculation the shifts for elements and its positions in sorted row;

4) Writes – for building the sorted row.

The number of operations compare (C) and write (W) for counting of bigger and smaller elements than current element is $\mathbf{C = n(n-1)/2}$. The c_j and d_i are incremented when the element a_j is compared with element a_i only if $a_j > a_i$, i=1, 2, …, n; j = i+1, i+2, …, n.

The number of operations increment (R) for counting the elements bigger and smaller than current element is:

$$\mathbf{R} = \sum_{\mathbf{j=1}}^{\mathbf{n}} \mathbf{d_j} + \sum_{\mathbf{j=1}}^{\mathbf{n}} \mathbf{c_j}. \tag{3}$$

The number of sums for finding positions of elements in sorted row is: $\mathbf{S = kn}$, where k=2 for ascending row and k=4 for descending row.

The number or write (move) operations for building the sorted row is **n**.

If t_1 is time for executing the operation compare, t_2 – time for executing the operation increment, t3 – time for executing the operation sum, t4 – time for executing the operation move, then time for building the sorted row is:

$$\mathrm{T} = \mathbf{t_1}\frac{\mathbf{n(n-1)}}{\mathbf{2}} + \mathbf{t_2}\left(\sum_{\mathbf{j=1}}^{\mathbf{n}} \mathbf{d_j} + \sum_{\mathbf{j=1}}^{\mathbf{n}} \mathbf{c_j}\right) + \mathbf{t_3 kn} + \mathbf{t_4 n} \tag{4}$$

The values of sums for d_j and c_j are equal:

$$\sum_{\mathbf{j=1}}^{\mathbf{n}} \mathbf{d_j} = \sum_{\mathbf{j=1}}^{\mathbf{n}} \mathbf{c_j} \tag{5}$$

The equation 5 is true because when $a_j \leq a_k$, k = j+1, j+2, …, n then some value of inversions table is incremented.

When n is constant, the first, third and fourth member of formulae are also constant. The value of second member of the formulae depends on inversions number. It's value is small when the number of inversions is small too. Finally - the time for building the sorted row has small value when the number of inversions of the given row is small.

Experimental Results

Discussed algorithm for sorting with tables of left and right inversions without moves is implemented on C in MS Visual Studio. The experimental scenario is as follows:

- random generated numbers are used by sorting (standard C function rand() is used for numbers generation);

- different sized arrays are sorted (the size of array for sorting is between 1 000 and 10 000);

- every size of array is used for the execution and time for sorting is presented as an averaged value;

- five sorting algorithms are used for examination: four well known algorithms (Bubble sort, Insertion sort, Selection sort and Quick sort) and presented algorithm for sorting with tables of left and right inversions without moves.

Computer system with CPU- Intel Celeron 3300 2,5GHz (with 2 cores) and 3GB RAM is used for experiments.

On figure 1 are presented results which are measured by experiments according above scenario. The algorithm for sorting with tables of left and right inversions without moves on figure 1 is named SortLRT (LRT- Left and Right inversions Tables).

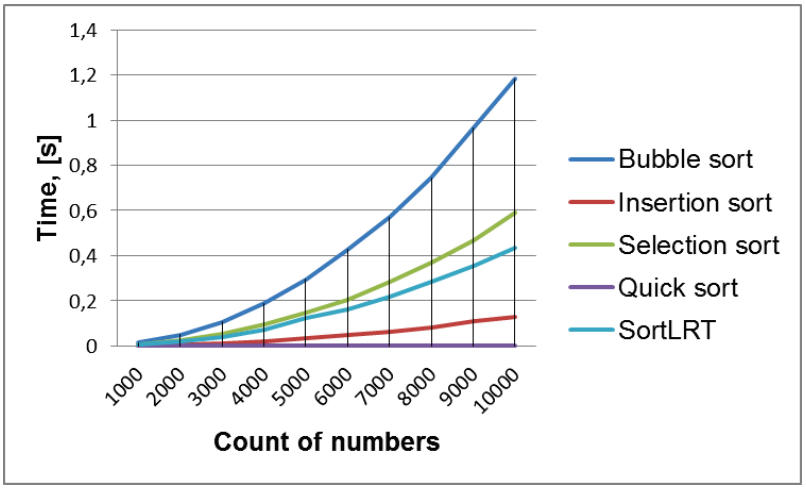

Fig. 1 Time for sorting with five different algorithms

The results show that proposed algorithm for sorting with tables of left and right inversions without moves is faster than Bubble sort and Selection sort but it is slower than Insertion sort and Quick sort.

CONCLUSION

This paper discuses two algorithms for sorting without moves. First algorithm uses tables of left inversions and second algorithm uses tables of left and right inversions. Analytical evaluation shows that second algorithm has better time complexity than first one. Because of this only second algorithm has a program implementation and an experimental evaluation. The results show that proposed sorting algorithm is not better than popular Quick sort algorithm but it is better than other well known sorting algorithms. One important advantage of discussed algorithm is that indices of sorted order is saved in additional array and it is not obligatory to change original data. The algorithm allow us to have together sorted and unsorted data. When it is necessary to have sorted data the array with indices that is produced by algorithm could be used to create new array with sorted data.

REFERENCES

Cormen T. H., Leiserson Ch. E., Rivest R. L. & Stein C. (2001). *Introduction to Algorithms Second Edition*. MIT Press.

Knuth D. (1973). *The art of computer programming, V3. Sorting and Searching*. Addison Wesley Publishing Company.

Sedgewick R. (1984). *Algorithms*. Addison Wesley Publishing Company.

Stoichev St. (2008). *Sintezi analiz na algoritmi*. BPS (***Оригинално заглавие**: Синтез и анализ на алгоритми (2008). БПС*).

Vasilev N. & Bosakova-Ardenska A. (2012). Algorithms for sorting by left inversions table, *International Review on Computers and Software (IReCoS)*, 7(2), 642-650.

Vasilev N. & Bosakova-Ardenska A. (2012), *Sorting by left inversions table with filling from left to right*, In proceedings of Automatics and Informatics, 353-356.

Wirth N. (1976). *Algorithms+data structures=programs*. Prentice-Hall.

ACKNOWLEDGMENT

This research was supported by University of Food Technologies according 12/16-H UFT Plovdiv research item.