I J C T A, 9(6), 2016, pp. 3353-3357 ISSN: 0974-5572
© International Science Press

Schematize Sorting

R. Srinivas*, Shaik Vahid and K.A. Sireesha*****

ABSTRACT

One of the fundamental issues in computer science is Sorting. Even though there are a vast number of sorting algorithms to sort the given data, still the problem of sorting has become a major hurdle in research. Even to optimize other algorithms, a well-organized sorting is very important. There are a number of methods to measure the efficiency of the sorting algorithms, but we cannot approximately come to a conclusion that which algorithm is the best, because each has its own advantages and drawbacks. In this paper, a novel sorting algorithm named schematize sorting is proposed which uses only assignments and comparison operations to sort the given elements. This algorithm is efficient when compared with bubble, novel, insertion and selection sorting methods.

Keywords: sorting;novel sorting;insertion sort;selection sort; buble sort; best sort; time complexit.y

1. INTRODUCTION

The rapid growth of information in the world is to search data in connection to this information. It should be ordered based on some property and it was estimated that more than half the time on many commercial computers was spent in sorting. Fortunately this is no longer going to exist as many classy methods have been developed for organizing data in a sorted order. Many sorting algorithms make use of various techniques to accomplish the sorting task[1-2]. The performance of the algorithm can be measured in number of ways. Few of them are listed below [3-4].

a. Memory usage
b. Recursion.
c. Stability
d. Real execution time
e. Computational complexity of element comparisons.

The proposed method uses comparison and assignments only. So while evaluating performance of the algorithm only these operations are considered. The time taken by an algorithm to execute depends on a number of factors [5-6]

i. Speed of the Processor used
ii. RAM capacity
iii. Operating system
iv. number of statements and quality of the statements
v. Programming language used
vi. Compiler used etc

* Professor Department of CSE ACET, Surampalem Kakinada, India, *Email: viceprincipal@acet.ac.in*

** Assistant Professor Department of CSE ACET, Surampalem Kakinada, India, *Email: vahida.shaik@acet.ac.in*

*** Assistant Professor Department of CSE AEC, Surampalem Kakinada, India, *Email: sireesha.kanduri@gmail.com*

The execution time of the algorithm also depends on the number of inversions in the given data.

Inversion: Let $x_1, x_2, x_3, \ldots\ldots x_{n-1}, x_n$ be a permutation of the set {1, 2, ….., n}. If i < j and $x_i > x_j$. The pair (x_i, x_j) is called an ***inversion*** of the permutation [7].

Divide and conquer method is used in proposed algorithm. The divide and conquer strategy uses following methods to solve problem

i. Breaking (sub) problem into sub(sub) problems that are themselves smaller instances of the same type of (sub)problem.
ii. Solve these sub problems recursively.
iii. Combining the answers of sub problems to get final solution to the given problem.

2. PROPOSED METHOD

A key element is randomly selected from the list of elements and it is compared with other elements. If the element is less than the key element keep it in the left list, if the element is equal to key element increment the count value of the key otherwise keep it in the right list. Continue this process until both left and right list elements are in sorted order or contain only one element. Additional memory is used to store intermediate results.

For example consider the list of elements: 8 6 7 4 2 9 6 4 2 5

First select on the element as key element say 4

Now compare elements of the list with selected key element

8 compared with 4, 8 greater than 4 so this element kept in right sub list : left list empty, right list {8}, count = 0

6 compared with 4, 6 greater than 4 so this element kept in right sub list: left list empty, right list {6,8}, count = 0

7 compared with 4, 7 greater than 4 so this element kept in right sub list: left list empty, right list {7,6,8}, count = 0

4 compared with 4, 4 equal to 4 so count value is incremented by one: left list empty, right list {7,6,8}, count = 1

2 compared with 4, 2 less than 4 so this element kept in left sub list: left list {2}, right list {7,6,8}, count = 1

9 compared with 4, 9 greater than 4 so this element kept in right sub list: left list {2}, right list {9,7,6,8}, count = 1

6 compared with 4, 6 greater than 4 so this element kept in right sub list: left list {2}, right list {6,9,7,6,8}, count = 1

4 compared with 4, 4 equal to 4 so count value is incremented by one: left list {2}, right list {6,9,7,6,8}, count = 2

2 compared with 4, 2 less than 4 so this element kept in left sub list left list {2,2}, right list {6,9,7,6,8}, count = 2

5 compared with 4, 5 greater than 4 so this element kept in right sub list: left list {2,2}, right list {5,6,9,7,6,8}, count = 2

Once all the elements are compared with key elements we have two sub lists of elements, one list contain the elements less than key and other list contain elements greater than key and count value indicated number of values equal to key. At this stage the correct position of the key element is determined either based on number of elements in the left list or based on the right list. The position of the key is number of elements in left list plus one.

Now apply the same procedure to both sub lists until all the elements are in sorted order.

3. ALGORITHM

3.1 Sort(a[],l,h)

Input: a[] is an array of elements, l is lower bound(index) of elements in the list, h is the upper bound(index) for the list of element

Output: array of sorted elementsoutput[]

Step 1: Set count = 0, low = l, high = h, j = 0,k = h

Step 2: Select one element randomly from the list of elements mark it as key element

Step 3: Compare elements of the array a starting from a[low] to a[high] and perform the following operations.

a. If element of array less than key element then b[j] = array element, j++

b. If element of array greater than key element b[h] = array element, h—

c. If element equal to key count = count +1;

Step 4: Repeat step5 for k=0 to count where step size = 1

Step 5: Output [l + j + k] = key

Step 6: Test the elements in the left sub list

i. if j = 0 no elements, no operation is required

ii. if j = 1 only one element is there, keep this element in output [l] = b[0]

iii. if j>2 then call the sort algorithm with parameter array b, left index 0, right index j—1

Step 7: Test the elements in the right sub list

i. if k-h = 0 no elements, no operation is required

ii. if k-h = 1 only one element is there, keep this element in output [h] = b[h]

iii. if k-h>1 then call the sort algorithm with parameter array b, left index k + 1, right index h

4. ALGORITHM ANALYSIS

4.1. Best case [8-12]

The best case is the one in which list of elements are divided into two equal sub lists. This is true for all key elements.

The number of elements is n

The number of levels possible m = $\log_2 n$

The time complexity in best case is **O(nlogn)**

4.2. Unfavourable case

The least case is one in which except key element, all other elements kept in left list only or kept in right list only. This is true for all the key elements in list.

The list of elements n

In first iteration one element is selected as key, the rest of the elements i.e. n-1 belong to one set. The same procedure is repeated for n-1 elements.

The unfavorable case complexity is $O(n^2)$

5. RESULTS

The best case time complexity of proposed sorting algorithm is O(nlogn) and in worst case $O(n^2)$. Number of experiments were conducted to compare the actual run time. The set of same elements is used for all algorithms. The execution times are given in table 1 and corresponding graphs given in graphs 1 and 2. The space complexity of proposed sorting the algorithm is O(n).

Table 1
Average execution times of Bubble, Novel, Insertion, Selection and proposed sorting method

Algorithm/No.of Elements	1000	5000	10000	20000	30000	40000	50000	100000	200000
Bubble sort	4.42	90.3	359.68	1457.76	3525.34	6536.56	10311.56	36234.0	162110.0
Novel sorting method	17.44	61.8	199.12	770.04	1791.92	3104.36	4859.96	17469.0	87484.0
Selection sort	0.0	47.0	156.0	562.0	1250.0	2563.0	4000.0	16062.0	65531.0
Insertion sort	0.0	16. 0	62.0	265.0	594.0	1125.0	1766.0	7047.0	30094.0
Schematize sorting	0.0	0.0	0.0	15.0	16.0	16.0	16.0	16.0	47.0

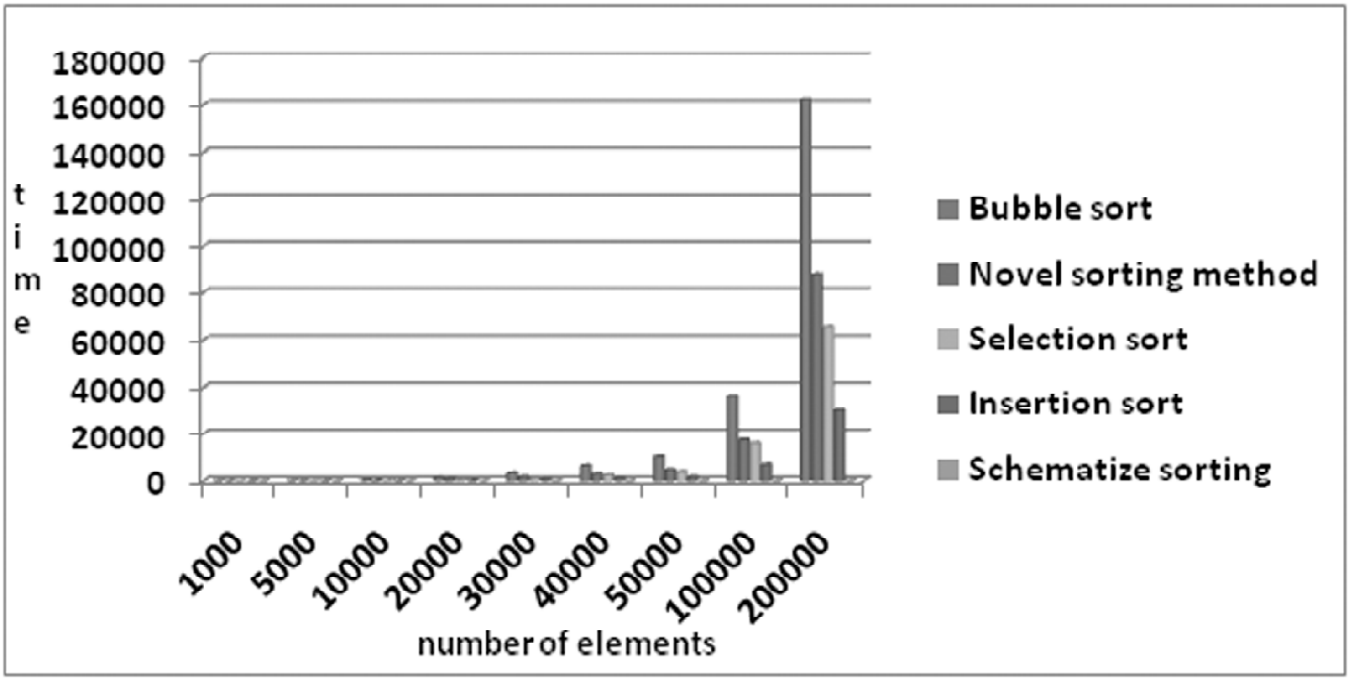

Figure 1: Bar Graph drawn for execution times

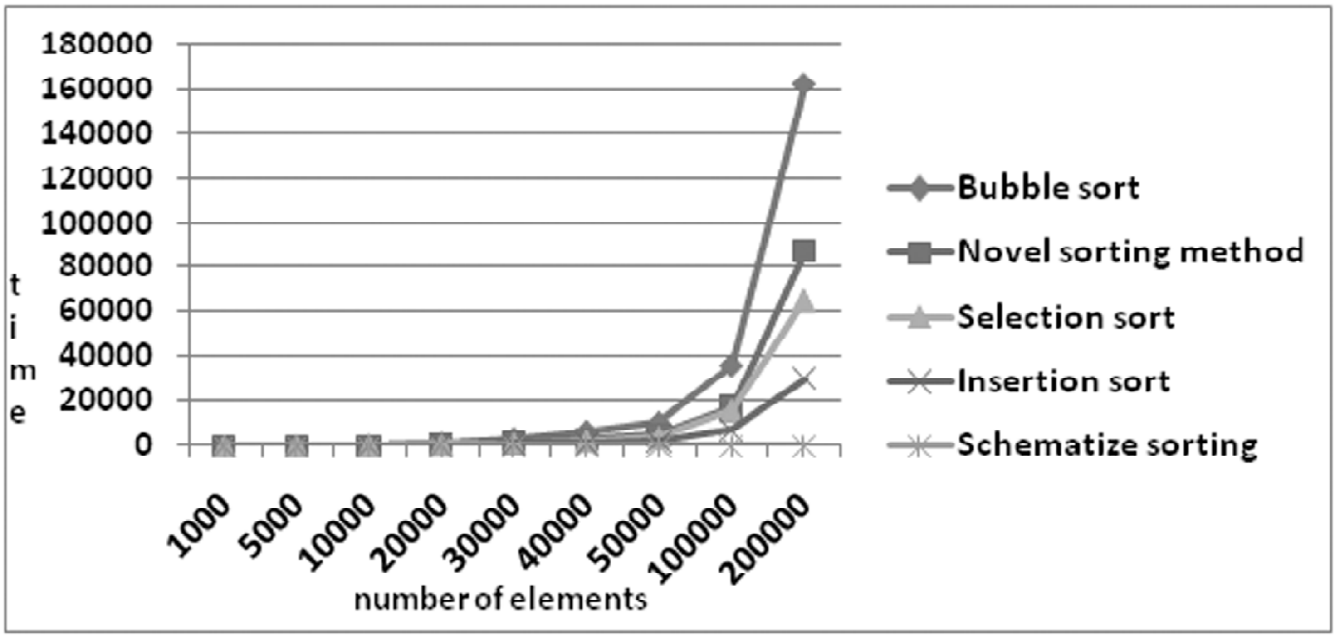

Figure 2: Line Graph drawn for execution times

6. CONCLUSION

To provide more proficient sorting, a new sorting algorithm was proposed in this paper. This algorithm is easy to understand and implement. This algorithm requires additional space to store intermediate results and to store the final output. It does not require any complex strategy to implement algorithm but requires intelligent methods to save memory.

REFERENCES

[1] D. Omar Khan, V. Shree lakshmi, S Sushma and D.C. Vinutha, " Analysis and Determination of Asymptotic Behavior Range For Popular Sorting Algorithms", Special Issue of IJCSI, ISSN : 2231–5292, Volume- II, Issue-1

[2] Alnihoud, Jehad. "An Enhancement of Major Sorting Algorithms", International Arab Journal of Information Technology Vol. 7, No. 1, January 2010.

[3] B. Parag, D.Nilesh, L.Sakharam, P. Santosh, "A Comprehensive Note on Complexity Issues in Sorting Algorithms", Advances in Computational Research, ISSN: 0975–3273, pp: 01-09, Volume- 1, Issue- 2, 2009.

[4] J. F. Francescri, Albert Jesus, "An Analysis of selection sort using recurrence relations", Questho, volume-20, pp. 111-119, 1996.

[5] Owen Astrachan, "Bubble Sort: An Archaeological Algorithmic Analysis", *SIGCSE '03*, February 19-23, PP:1-5, Reno, Nevada, ACM,Volume-35, Issue -1.

[6] S. Nadathur, H. Mark, G. Michael, "Designing Efficient Sorting Algorithms for Manycore GPUs", 23rd IEEE International Parallel and Distributed Processing Symposium, May 2009.

[7] Krung S., "The sorted list exhibits the minimum successive difference", The Joint Conference on Computer Science and Software Engineering, Nov- 17th-18th, 2005.

[8] R. Srinivas, A.Raga Deepthi, "Novel Sorting Algorithm", IJCSE, pp-1-4,Volume 5, Issue 01, Jan. 2013.

[9] R. Srinivas, "Enhanced Novel Sorting Algorithm", GJCSTSDE, Volume 13, Issue 11, Year 2013.

[10] D.S. Malik, "C++ Programming: Program Design Including Data Structures", Course Technology (Thomson Learning), 2002.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, "Introduction to Algorithms", MIT Press, Cambridge, MA, 2nd edition, 2001.

[12] C.L. Liu, "Analysis of sorting algorithms", Proceedings of Switching and Automata Theory, 12th Annual Symposium, East Lansing, pp: 207-215,MI, USA, 1971.