International Journal of Scientific Research in Computer Science, Engineering and Information Technology
© 2017 IJSRCSEIT | Volume 2 | Issue 6 | ISSN : 2456-3307

Look Back Sort : A Backward Comparison Algorithm for Internal Sorting

Jyoti Lakhani[1], Kirti Shrimali[2], Dharmesh Harwani[3*]

[1,2]Department of Computer Science, Maharaja Ganga Singh University, Bikaner, Rajasthan, India
[3]Department of Microbiology, Maharaja Ganga Singh University, Bikaner, Rajasthan, India

ABSTRACT

Present communication is an attempt to develop a novel algorithm "Look-Back Sort" (LB Sort) for internal sorting. The objective of the present research work was to reduce the space and time complexities of the sorting process in the worst cases. The performance of all internal sorting algorithms in worst cases are bound to O(n log n). To address the issue, the proposed algorithm is implemented first followed by its benchmarking with some other internal sorting algorithms. The empirical analysis of algorithms using C++ function is performed on the numeric samples vary in size. It was observed that the runtime of the LB-sort algorithm is highly comparable with the Quick sort algorithm in the worst case analysis. The overall time complexity of the proposed algorithm is observed to be O(n log k) when k<<n where n is denoting the current index. The correlation and covariance analyses also revealed that the proposed algorithm improves the performance of sorting process in the worst case highly significantly without compromising its simplest execution. Moreover, the completion of the sorting process in a single run is its additional advantage.

Keywords : Internal Sorting, Backward Comparison, Benchmarking, Empirical Study, Space Complexity, Time Complexity

I. INTRODUCTION

Sorting is the process to arrange given items in a numerical or lexicographical order [1], [4]. There are mainly two types of sorting algorithms known as internal and external sorting algorithms. Internal sorting algorithms work on the small size data that held in primary memory. On the other hand, if data is large as compared to the primary memory then the internal sorting algorithm does not work. In these cases external sorting algorithms are used where data is kept in the external disk during the sorting process. The present study highlights an improved version of sorting algorithm to execute internal sorting. There are several other simple internal sorting algorithms detailed out in the literature such as Insertion sort, Selection sort, Bubble sort, Quick sort, Merge sort etc. [2],[8],[9],[10],[11],[12],[13],[14],[15]. These sorting algorithms function as the platform for other complicated sorting algorithms. The efficiency of an algorithm can be estimated by comparing its execution time. The number of comparisons performed by a sorting algorithm is directly proportional to the complexity [5] and execution time. Therefore the number of comparisons can also be considered as another criterion to address complexity of a sorting process. It is important to mention here that the programming languages used for implementation of the sorting algorithms do not have any impact on the complexity of algorithm [3]. The proposed algorithm tracks the previously sorted data held in an array. The performance of the algorithm has been investigated using asymptotic complexity analysis and the time complexity has been evaluated using empirical study.

Internal Algorithms

None of the internal sorting algorithm can perform faster than $O(n \log n)$ in an average and worst case [4].

The reason behind this is that all of these algorithms are in-place or comparison sorts [1]. The Selection sort

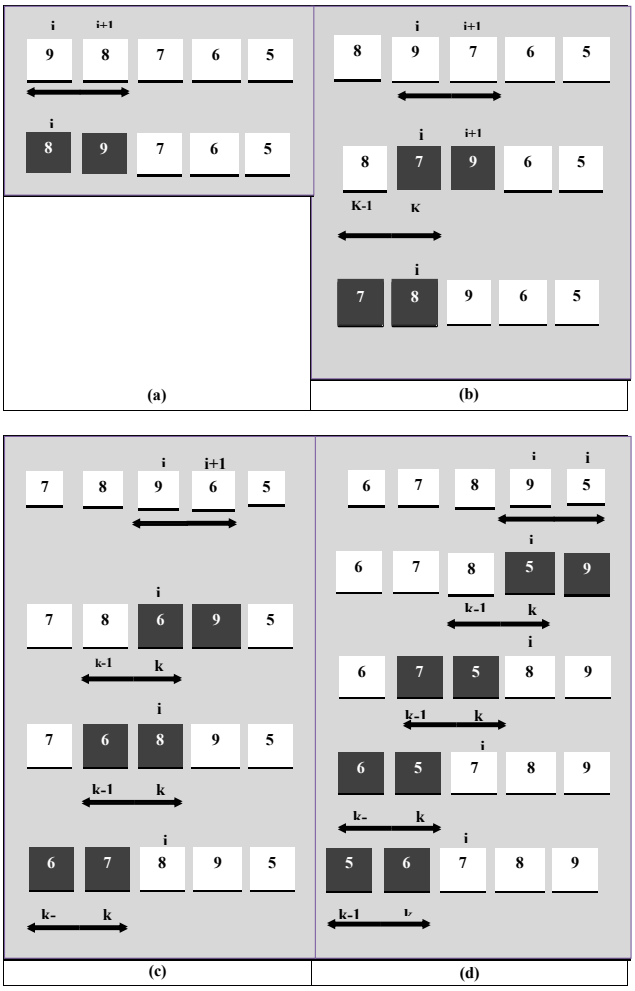

Figure 1. The process of sorting using the proposed LB-sort algorithm

is suitable for short length arrays with $O(n^2)$ complexity. The algorithm matches nearest items to perform in-place comparisons. It is preferred over the other complex algorithms for its simplicity which is its main advantage [6]. Insertion sort is another sorting algorithm which is the most suitable for small length arrays. It efficiently works for average cases. Whereas, the internal sort algorithm is used as a basic supporting algorithm for many other complex sorting algorithms. The main idea behind implementing the insert sort algorithm is to pick items one by one and insert them at the appropriate place in an array. The Insertion sort algorithm is an expensive algorithm in terms of space and time issues as it shifts elements in the array several times. The worst case running time of Insertion sort algorithm is $O(n \log n)$. The main feature of the Insertion sort algorithm is that it does not require random access in an array. The Bubble sort algorithm can function very efficiently on short length ordered arrays. It compares two adjacent elements and swaps them if required. Bubble sort requires $O(n^2)$ time to sort an array. Therefore the algorithm is not suitable for the large and unsorted data sets. The Quick sort algorithm is the fastest in all algorithms due to its divide and conquers nature [7]. It partitions an array and finds its pivot. The pivot is assumed to take already sorted place and located centrally. Once the pivot is found, all the other elements in the array are compared with it. The array elements smaller than the pivot is moved to the left and those which are large enough are moved to the right. These lesser and greater sub lists are then recursively sorted by using the same process which yields average runtime complexity of $O(n \log n)$. The Quick sort is complex but considered as the fastest sorting algorithm amongst other internal sorts.

II. METHODS AND MATERIAL

The proposed LB-sort algorithm is based on the backward comparison method. Algorithm targets items held in an array. Let n be the size of an array, i is the current index and the order is ascending. The algorithm starts with scanning items of the array from index i=0 and continue with length until it reaches at index n-2. The algorithm compares i^{th} item with i+1th item. In case of inappropriate order, the algorithm swaps the i^{th} and i+1th item and scan the array backwardly from i^{th} index to 0th index. The swap occurs, if an incompatibility is found in the sorted order at any position otherwise algorithm continues further. The process is shown in figure 1.

Algorithm: LB-sort (array)
Input: an array with size n
Output: sorted array
Steps:

1. For each index i from 0 to n-2
2. Compare i and its next item at i+1 position
3. If require swap i^{th} and i+1th item
4. For each index k=i to k>0
5. Compare k^{th} and k-1th item
6. If require swap k^{th} and k-1th item
7. End

III. RESULTS AND DISCUSSION

The proposed algorithm has been implemented in C++ programming language on the windows 64bit platform

with dual core 2.10GHz processor. To perform empirical analysis, we have implemented four other internal sort algorithms on the same machine. The four implemented internal sorting algorithms are Insertion sort, Selection sort, Bubble sort and Quick sort. Data samples of size from 10 to 10000 were inserted in an array and then all four above mentioned as well as LB-sort sorting algorithms were run. Running time and number of comparisons were used as the criteria for empirical analysis. Results, as shown in figure 2, clearly indicates that LB-sort require the minimum number of comparisons as compared to other algorithms. It also revealed that the Quick sort algorithm is the second best algorithm for the implementation of the same query followed by Insertion and Bubble sort algorithms**.** The Insertion and Bubble sort algorithms require almost the same number of comparisons for the inputs of short length of size less than 5000. It is clear from figure 2 that the Insertion sort algorithm performs better than the Bubble for the larger size inputs. The Selection sort algorithm discouraged Insertion and Bubble sort algorithms in terms of the number of comparisons made whereas it is least effective than the LB-sort and Quick sort algorithms.

Asymptotic analysis of the running time of the different sorting algorithms has been presented in Table 1. It is evident here that LB-sort algorithm has $\Omega(n)$ complexity for the best case. The same appears also for Insertion and Bubble sort algorithms. For Quick sort and Selection sort algorithms Ω(n log n) and Ω(n2) complexities were observed respectively. For average case, the complexity of LB-sort was observed to be θ(n log k)(k<<n) whereas for Insertion, Bubble and Selection sort θ(n2) complexity were observed. The complexity of Quick sort algorithm θ(n log n) is an exception here. It is clear from the above data that LB-sort performs relatively well in the worst case with the complexity observed as same as for the average case. All other four internal sort algorithms have worst case complexity O(n2) which is at the higher side than the LB-sort algorithm. Intriguingly, a Quick sort algorithm requires auxiliary space whereas other internal sort algorithm including LB-sort does not require any additional auxiliary space.

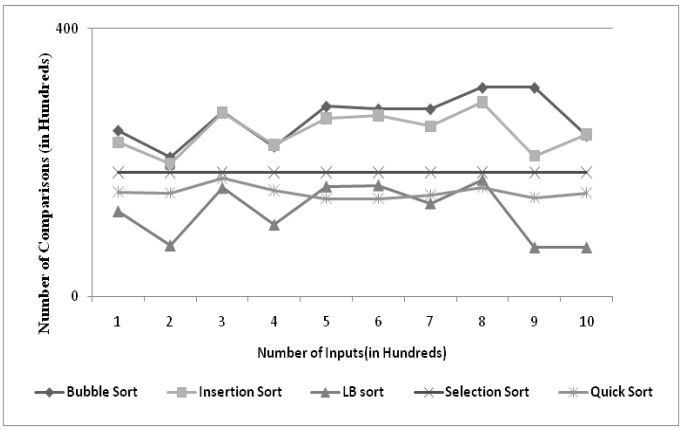

Figure2. Benchmarking of various internal sorting algorithms in terms of number of comparisons

TABLE I

TABLE1. TIME AND SPACE COMPLEXITIES OF INTERNAL SORT ALGORITHMS

Algorithms	Time complexity			Auxiliary Space complexity
	Best case	Average Case	Worst Case	
Insertion	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$	$O(1)$
Selection	$\Omega(n^2)$	$\theta(n^2)$	$O(n^2)$	$O(1)$
Bubble	$\Omega(n)$	$\theta(n^2)$	$O(n^2)$	$O(1)$
Quick	$\Omega(n \log n)$	$\theta(n \log n)$	$O(n^2)$	$O(n)$
LB	$\Omega(n)$	$\theta(n \log k)$ $k<<n$	$O(n \log k)$ $k<<n$	$O(1)$

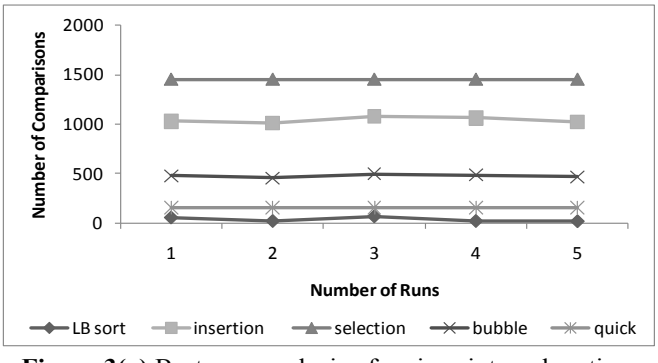

Figure 3(a) Best case analysis of various internal sorting algorithms

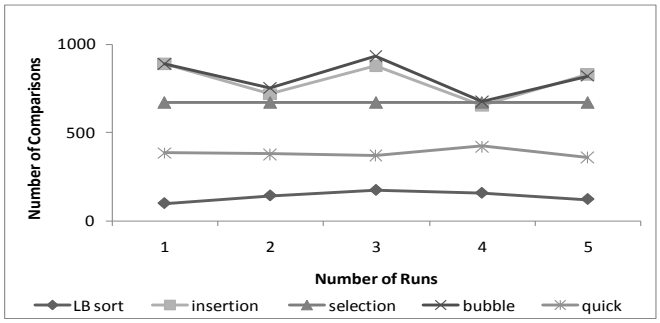

Figure 3(b) Average case analysis of various internal sorting algorithms

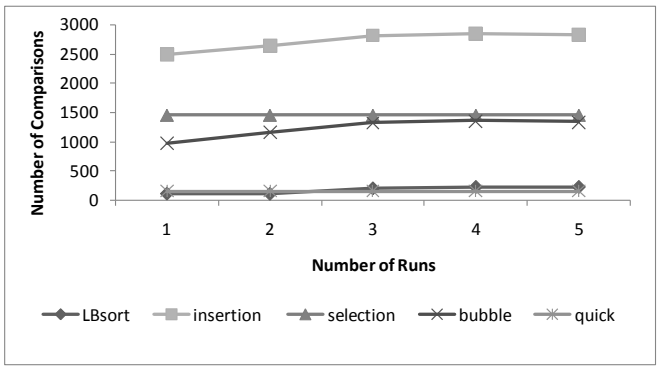

Figure 3(c).Worst case analysis of various internal sorting algorithms

The time complexity issue of the algorithms used in the present investigation has also been tested, results from which have been furnished in figure 3(a), 3(b) and 3(c). All the five algorithms have been run with best, average and worst case datasets. The results clearly revealed that for the best and average cases, LB-sort algorithm performs relatively better than the other four algorithms used. However for the worst case dataset, the performance of LB and Quick sort algorithms is comparable.

To find out, if any correlation exists between experimental algorithms, statistical analysis was performed. The analysis revealed a positive relation between of LB-sort algorithm with Insertion sort (0.0373), Bubble sort (0.0350) and Quick sort (0.0361) algorithms while a negative correlation with Selection (-0.070) algorithm was observed. The negative correlation value clearly indicates that there is no statistically significant relationship between the LB-sort algorithms with the Selection sort algorithm. In addition to that a covariance analysis was also performed between LB-sort and Insertion sort, Bubble sort and Quick sort algorithms (Figure 4(a), 4(b), 4(c) and 4(d)). The results from this analysis further confirms that the LB sort algorithm is more near to the Quick sort algorithm (86.98) in performance than the Insertion (982.8) and Bubble sort (661.36) algorithms.

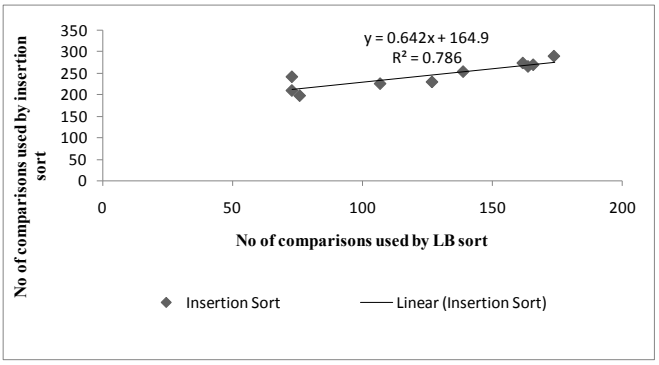

Figure 4(a) Scatter plot for the correlation analysis between LB sort and Insertion sort algorithms

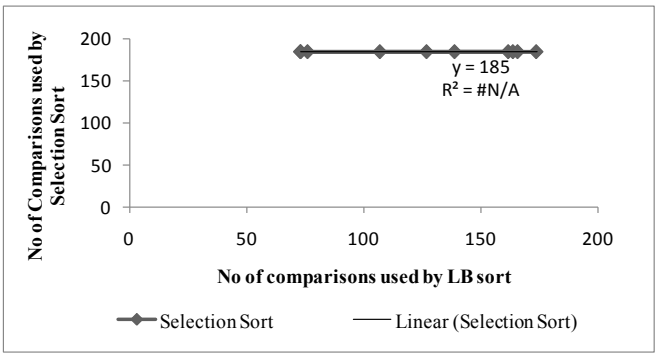

Figure 4(b) Scatter plot for the correlation analysis between LB sort and Selection sort algorithms

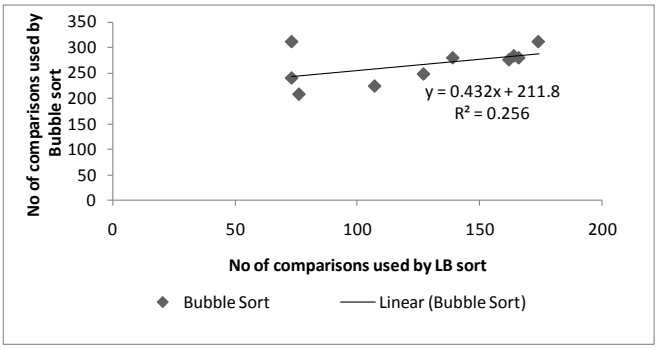

Figure 4(c). Scatter plot for the correlation analysis between LB sort and Bubble sort algorithms

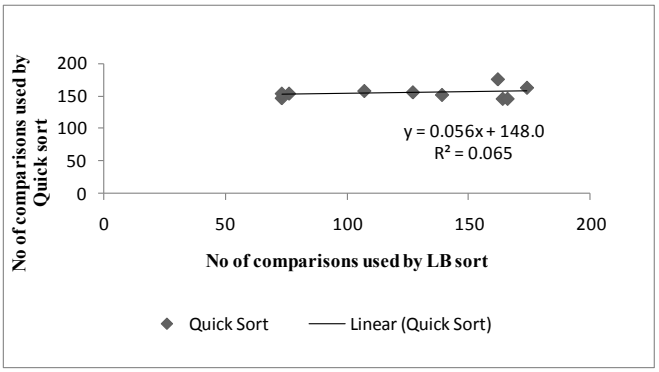

Figure 4(d). Scatter plot for the correlation analysis between LB sort and Quick sort algorithms

IV. CONCLUSION

The proposed LB-sort algorithm is found to be the best performer in our study with relation to its least space and time complexities than the Insertion, Selection,

Bubble and Quick sort algorithms for the best, average and worst cases. Although, the LB-sort algorithm is an in-space comparison algorithm similar to the Selection sort algorithm yet sorts the entire array in a single run. The worst case time complexity of the algorithm is O(n log k) where $k<<n$. Here, k is the current index. The statistical tests performed in the present study also confirmed that the LB-sort algorithm is able to execute much faster sorting followed by Quick, Insertion, Bubble and Selection sort algorithms. Moreover, it does not require any additional space as it is required in Quick sort algorithm. Moreover, an additional run to execute sorting is not needed in the LB sort algorithm as it is needed in the Bubble and Insertion sort algorithms.

V. REFERENCES

[1]. https://en.wikipedia.org/wiki/Sorting_algorithm downloaded on December 20, 2017

[2]. T.H. Cormen, C.E. Leiserson, R.L. Rivest. C. Stein, 2001. "Introduction to Algorithms", 2nd edition, MIT press,

[3]. A.M. Aliyu and Dr. P.B. Zirra, 2013. "A Comparative Analysis of Sorting Algorithms on Integer and. Character Arrays", The International Journal Of Engineering And Science (IJES), vol. 2, no. 7, (2013), pp. 25-30.

[4]. I. Flores. 1960. "Analysis of Internal Computer Sorting", ACM, vol. 7, no. 4, (1960), pp. 389-409.

[5]. C.A.R. Hoare. 1961. Algorithm 64: Quick sort. Comm. ACM, vol. 4, no. 7 (1961), pp. 321.

[6]. Selection Sort, http://www.algolist.net/Algorithms/ Sorting/Selection_sort

[7]. E. Horowitz, S. Sahni and S. Rajasekaran, first edition, 1987. "Computer Algorithms",Computer Science Press, New York.

[8]. S. Lipschutz, 1986. "Theory and Problems of Data Structure", McGraw Hill Book Company.

[9]. Md. Khairullah, 2013. International Journal of Advanced Science and Technology Vol. 56, July, 2013.

[10]. Sengupta et al., 2007. "Algorithms in Java", 3rd ed., Part 1-4, Addison-Wesley Professional Publisher, New York.

[11]. J. Hammad, 2015. "A Comparative Study between Various Sorting Algorithms", International Journal of Computer Science and Network Security, VOL.15 No.3.

[12]. Jr P.W., C.A. Brown. 1985. "The analysis of algorithms". Holt, Rinehart & Winston, New York.

[13]. S. Baase. 1988. "Computer algorithms: Introduction to analysis and design", Addison Wesley, Reading, Massachusetts,1988.

[14]. R. E. Neapolitan, K. Naimipour. 2010. "Foundations of algorithms", Jones & Bartlett Learning.

[15]. U. Manber. 1989. "Introduction to algorithms: A creative approach", Addison-Wesley Longman Publishing Co., Inc.