Journal of Applied Information Science
6 (1), June 2018, 39-41
http://www.publishingindia.com/jais

An Enhancement of Selection Sorting

V. Bothra[1] and S. Bhalerao[2*]
[1]Acropolis Institute of Technology and Research, Indore, Madhya Pradesh, India.
[2]Professor, Department of Computer Science Engineering, Acropolis Institute of Technology and Research, Indore, Madhya Pradesh, India. Email: shilpabhalerao@acropolis.in
*Corresponding Author

Abstract: One of the fundamental issues in computer science is ordering a list of items. Although there are many sorting algorithms, different sorting algorithms can be used in different scenarios. This paper presents an enhanced version of selection sort. Efficient selection sort is an enhancement on selection sort by making it a slightly faster sorting algorithm because it overcomes the limitations of the existing selection sorting algorithm. The new algorithm is analyzed, implemented, tested, and compared and the results are promising.

Keywords: Bubble sort, Complexity, Efficient Selection Sort (∃SS), Selection sort, Sorting algorithm.

I. Introduction

In today's era, exponential growth of data, generate tremendous information. This information is widely used in various domain such as health care, entertainment, financial investments etc. Huge data is available in various formats that require use of various techniques to extract information. The basic information from the data is to visualize data in sorted order. However, there is not always a compulsion that data should be sorted but it makes it easy to be processed and used.

Sorting is one basic concept of computer science and used in many applications as subroutine. However, many sorting algorithm already exist for getting optimum results. Still there is scope of improvement in existing sorting algorithm for dealing huge structured data. In this paper, we have proposed a new sorting algorithm i.e. Efficient Selection Sort (∃SS). Proposed algorithm is enhanced version of selection sort. The study shows that the proposed algorithm is more efficient, theoretically, analytically, and practically as compared to the original sorting algorithm.

Section 2 presents the concept of Efficient Selection Sort (∃SS) algorithm and its pseudo code. In section 3, implementation, analysis, and comparison of proposed algorithm with selection sort is briefly described. Also, the performance of ∃SS with the selection sort, bubble sort and quick sort is compared in form of various case studies. Finally, conclusions are presented in Section 4.

II. Enhanced Selection Sort (∃SS)

A. Concept

Input: unsorted array

Proposed sorting algorithm is based on applying divide and conquer method in selection sort. In selection sort after a single iteration smallest element reaches to its proper position. In proposed algorithm, traverse the array from both ends to find the smallest element and exchanging it with the first element, and then decreasing the size of the array be one for the next call. In fact, the Efficient Selection Sort (∃SS) algorithm is an enhancement to the selection sort algorithm in decreasing the number of swap operations as well as the time for processing all the elements to place the smallest element at the top.

B. Algorithm

The algorithms can be described as follows:

1. Read all the elements in array.
2. Set i to initial index.
3. Traverse the array from both side i.e. from one pointer from i index in incremental order and second pointer will point to last element of array from and traverse in reverse direction.
4. Algorithm finds the smallest element in the array traversing from both sides at the same time and swapping it with the a[i] of the same array.
5. Increment I by 1.
6. Repeat step 3 and until I equal to total number of element.

C. Pseudocode

In simple pseudo code, efficient selection sort algorithm is expressed as:

```
Function efficient selection sort (array, size)
1       vari, j
2       var min1,min2,back_index, temp
3       for i := 1 to size-1 do
4               min1 := i
5               min2 := size
6               back_index := size
7               for j := i+1 to (size+i)/2 do
8                       if array(j) < array(min1)
9                               min1 := j
10                      end if
11                      if array(back_index) < array(min2)
12                              min2 := back_index
13                      end if
14              back_index=back_index-1
15                      j=j+1
16              end for j
17      if array(min1)<=array(min2)and min1!=i
18              temp := array(i)
19              array(i) := array(min1)
20              array(min1) := temp
21      end if
22      else
23              temp := array(i)
24              array(i) := array(min2)
25              array(min2) := temp
26      end else
27      i=i+1
28      end for i
```

D. Analysis

For loop, in line 7 iterates (*n-1)/2* times in the first call then *n* keeps decreasing by one after each pass. We may say that:

$$T(n) = (n-1)/2 + T(n-1)$$
$$= (n-1)/2 + (n-2)/2 + T(n-2)$$
$$= (n-1)/2 + (n-2)/2 + (n-3)/2 + T(n-3)$$
$$= (n-1)/2 + (n-2)/2 + (n-3)/2 + (n-4)/2 + T(n-4)$$
$$= \ldots$$
$$= (n-1)/2 + (n-2)/2 + (n-3)/2 + \ldots + (1/2)$$

So, $T(n) = (n^2+n)/4 = O(n^2)$

Although, ∃SS algorithm has same complexity of selection sort but it reduces the time required for processing all the elements by checking 2 elements in each single pass of the internal loop.

The number of swaps in the proposed sorting algorithm may be summarized as:

(a) In the best-case scenario: if we have a sorted array, then there is no swapping in that case.

(b) In the average-case scenario: if the input array is in reversed order (elements in descending order) then minimum n/2 swapping operations are required to be performed since we are traversing the array from both sides.

For example, if we have a descending sorted array as follows:

10	8	6	4	2

Then during the first pass the first element will be swapped with the last element:

2	8	6	4	10

Hence, the minimum value is placed at the first index of the processed array.

In the next comparison, the second element now is the minimum value and it will be swapped with the first element of the resulting array.

2	4	6	8	10

The array is now sorted hence, we would no longer require a swapping operation. Therefore, n/2 number of swapping operations are performed in case of a reversed array.

(c) In the worst case; if the input array is unsorted i.e., neither ascending nor descending, then the required swapping operations are approximately *n* operations.

III. Comparison with Selection Sort Algorithm

Selection Sort algorithm, identify of smallest element from unsorted array and place it to proper position. Therefore, number of comparison in each iteration is n-i in i^{th} iteration. Therefore, selection sort has a complexity of $O(n^2)$ [2, 3].

The main advantage of efficient selection sort over selection sort algorithms are as follows:

1. Selection sort always perform O(n) swaps whereas the ∃SS algorithm performs swapping operation depending on the input array.
2. If the input array is already sorted, the ∃SS does not perform any swap operation, but selection sort performs *n* swap operations. Writing in memory is more expensive in time than reading. However, there are many similarities between ∃SS and SS algorithms, as shown in Table I.

Table I: ∃SS vs SS Algorithms

Criteria	Enhanced Selection Sort	Selection Sort
Best Case	$O(n^2)$	$O(n^2)$
Average Case	$O(n^2)$	$O(n^2)$
Worst Case	$O(n^2)$	$O(n^2)$
Stability	Yes	Yes
Number of Swaps	Depends on data	Always n

To prove that ∃SS algorithm is relatively faster than selection sort, we implemented proposed algorithm and measured the execution time for large amount of date in numeric form. We have also compared proposed algorithm with other efficient sorting techniques such as bubble sort with case studies.

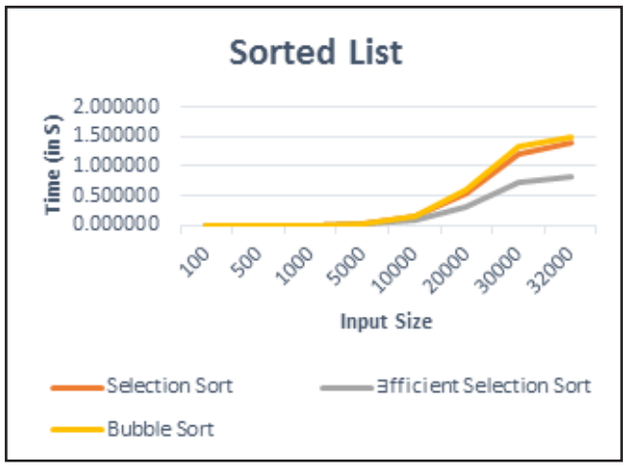

Fig..1: Comparison of ∃SS Algorithm for Sorted List

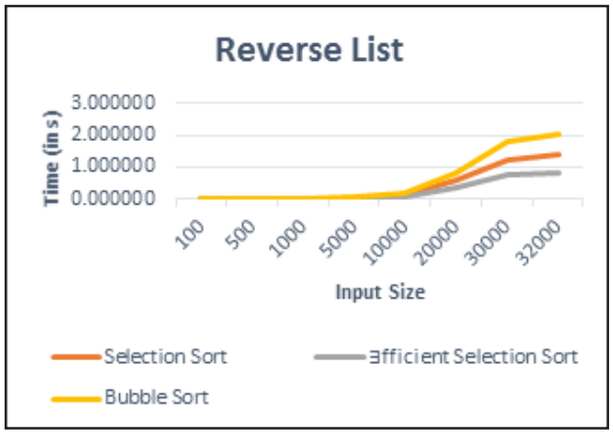

Fig. 2: Comparison of ∃SS Algorithm for Decreasing Order List

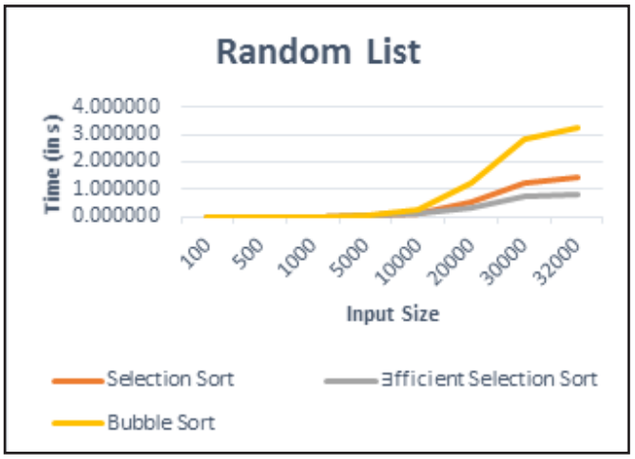

Fig. 3: Comparison of ∃SS Algorithm for Random List

We have considered 3 case studies: Sorted numbers, decreasing order and random number.

Dataset: numeric data with variable size of input. We have extended our data from 100 row to 32000 and observed the time required to sort by ∃SS algorithm, selection sort and bubble sort. Following are observations of our three case studies shown in Fig. 1, Fig. 2, Fig. 3.

1. The proposed sorting technique perform at almost same efficiency in all the cases whereas there is a significant difference in time required by other sorting techniques on changing the input array. For example, time increases drastically in selection and bubble sort in case of random value and size.
2. The proposed algorithm can be seen as the best option in all three cases due to divide and conquer algorithm.
3. The proposed algorithm is the most viable option in comparison with the selection sort and the bubble sort when dealing with the random array as shown in Fig. 3.

IV. Conclusions

In this paper, a new sorting algorithm has been presented. Although, ∃SS has $O(n^2)$ complexity, but case studies depicts better performance in comparison of selection sort.

Selection sort can be specially programmed to be stable sort whereas ∃SS is stable without the need to this special implementation.

Moreover, ∃SS is definitely faster than Bubble sort, because of the fact that bubble sort performs $O(n^2)$ operations but ∃SS performs operations based on the input array.

References

[1] J. Alnihoud, and R. Mansi, "An enhancement of major sorting algorithms," *The International Arab Journal of Information Technology*, vol. 7, no. 1, pp. 55-62, January 2010.

[2] I. Flores, "Analysis of internal computer sorting," *Journal of the ACM*, vol. 8, no. 1, pp. 41-80, January 1961.

[3] J. W. J. Williams, "Algorithm 232: Heap sort," *Comm. ACM*, vol. 7, no. 6, pp. 347-348, June 1964.

[4] A. Andersson, and S. Nilsson, "A new efficient radix sort," In *Proceeding of the 35th Annual IEEE Symposium on Foundation of Computer Science,* pp. 714-721, 1994.

[5] V. E. Castro, and D. Wood, "A survey of adaptive sorting algorithms," *Computing Surveys*, vol. 24, pp. 441-476, 1992.