Optimized Selection Sort Algorithm for Two Dimensional Array

Sultan Ullah, Muhammad A. Khan
Department of Information Technology,
University of Haripur,
Haripur, Pakistan.
Email: (sultan,aamir)@uoh.edu.pk

Mudasser A. Khan, H. Akbar, and Syed S. Hassan
Computer and Communication Research Lab
University of Haripur,
Haripur, Pakistan.
Email: (mudasser,habib,shabih)@uoh.edu.pk

***Abstract*—The main idea of Optimized Selection Sort Algorithm (OSSA) is based on the already existing selection sort algorithm, with a difference that old selection sort; sorts one element either smallest or largest in a single iteration while optimized selection sort, sorts both the elements at the same time i.e smallest and largest in a single iteration. In this study we have developed a variation of OSSA for two-dimensional array and called it Optimized Selection Sort Algorithms for Two-Dimensional arrays OSSA2D. The hypothetical and experimental analysis revealed that the implementation of the proposed algorithm is easy. The comparison shows that the performance of OSSA2D is better than OSSA by four times and when compared with old Selection Sort algorithm the performance is improved by eight times (i.e if OSSA can sort an array in 100 seconds, OSSA2D can sort it in 24.55 Seconds, and similarly if Selection Sort takes 100 Seconds then OSSA2D take only 12.22 Seconds). This performance is remarkable when the array size is very large. The experiential results also demonstrate that the proposed algorithm has much lower computational complexity than the one dimensional sorting algorithm when the array size is very large.**

Keywords*—Algorithm analysis, Sorting algorithms, Two Dimensional Arrays, Computational Complexity.

I. Introduction

Sorting and Searching are the tasks that are repeatedly encountered in various computer applications. Since they reveal basic tasks that must be deal with quite frequently, researchers have tried in the past to develop algorithms efficient in terms of least memory requirement and least time requirement. Searching together with sorting is perhaps the most common operation used in Computing. It was always a matter of great attraction for the researchers that to reduce the time a computer takes in sorting and searching data [1], [2]. That's why, the researcher tried to develop speedy, well-organized and economical algorithms for sorting and ordering lists and arrays. This was one of the fundamental fields of research in computing for decades. The development of optimized sorting algorithms will leads to efficient computing as whole. It is observed that, the time to complete the task for most sorting algorithms depends on the amount of data. Consequently to analyze an algorithm, it is required to find a relationship showing how the time needed for the algorithm depends on the amount of data. It is observed that, for an algorithm, when the amount of data is doubled, the time needed is also doubled. Similarly from the analysis of another algorithm it was cleared that when the amount of data is increased the time to complete the task of sorting is increased by a factor of four. It is also a matter of great concern that, the nature of the data also has an effect on the efficiency of the algorithm. A sorting algorithm may behave differently if the original data set is almost sorted rather than if it is random or it may be in the reverse order. The creation of spatial data structures that are important in computer graphics and geographic information systems is necessarily a sorting procedure. A well-organized sort routine is also a useful unit in implementing algorithms like sparse matrix multiplication and parallel programming patterns like Map Reduce [3], [4]. The idea of optimized selection is based on the old selection sort [5]. Unlike the old selection sort which sort data item one at a time, the optimized selection sort algorithm (OSSA) sort two data elements (smallest as well largest) in a single iteration. It reduced the execution time of the selection sort by almost fifty percent, although this algorithm also has a computational time complexity of Big O$((n)^2)$ [5], [6]. Enormous data in many fields is normally represented by the use of Matrices. It is felt by the computer professional to have efficient sorting algorithms for sorting the two-dimensional arrays. There is plenty of algorithms which mainly focuses on one-dimensional array sorting, but a few algorithms have been proposed for two-dimensional array sorting [7], [8], [9]. In [8], the result is extended from one-dimensional array to two-dimensional by the use of interpolation and filtering algorithms which results in a complexity of O$((mn)^2)$. Two-dimensional array AR(M,N) is treated as one dimensional array by most of the earlier sorting algorithms to sort these mn elements [10], [11], [12]. Simple selection sorting is the widely sorting algorithm used due to its simplicity and less implementation complexity. Although it has a time complexity of O$((mn))^2$) on sorting one dimensional array with mn elements as (n-1)nm/2 comparison operations are required. The time required for sorting large arrays (e.g. arrays with 1000000*1000000 elements) is practically much time consuming. In this article, we propose an optimized sort algorithm for two dimensional arrays, to solve the above mentioned problem of sorting. The hypothetical and experimental results proved that this proposed algorithm has a high efficiency level [10], [13]. The paper is organized in the following manner. In section 2 the algorithm for two dimensional arrays is proposed. The proposed algorithm (OpSSA2D) is analyzed in section 3. In Section 4, the approach of sorting single dimensional arrays is presented for the proposed algorithm. Section 5, Experimental analysis and discussion is presented and Section 6 contain conclusion.

II. Proposed Optimized Selection Sort Algorithm For Two Dimensional Arrays

The main idea of Optimized Selection sort is based on already existing selection sort algorithm, with a difference that selection sort, sorts one element either smallest or largest in a single iteration while optimized selection sort, sorts the two elements at a time i.e smallest and largest in a single iteration. We have developed a variation of the already existing selection sort for two-dimensional array and called it Optimized Selection Sort Algorithms for Two-Dimensional array.

We have a matrix in the form of AR(M,N) as follows

$$AR(M,N) = \begin{pmatrix} AR_{11} & AR_{12} & . & AR_{1n} \\ . & . & . & . \\ . & . & . & . \\ AR_{1m} & AR_{m2} & .... & AR_{mn} \end{pmatrix}$$

Before going to the stepwise details of the algorithm we will create a new array Y(M,N) which will store sorted elements. The working of our proposed optimized selection algorithm for two-dimensional array is depicted in the following steps:

STEP 1: Each row of the array should be sorted using the Optimized Selection Sorting Algorithm (Let assume that the elements are sorted in descending order in each row).
STEP 2: Initialize the index values of the target array i.e Y such that (row,col)as $row = col = 1$
STEP 3: Find the Largest element from the first column and placed it into Y(row,col) the successor the largest will now be the first element of the row.
STEP 4: Update the index values for row,col $col = col + 1$
If $col > n$then $col = 1, row = row + 1$
STEP 5: Goto step (3) if AR is not empty, else goto Exit.
STEP 6: EXIT.

III. Analysis of OpSSA2D

The Most of the time in sorting process the algorithms major cost is on the comparisons of the data items to be sorted. Therefore we analyzed the time complexity of by computing the number of comparison which is performed. As we know for sorting one dimension array the number of comparison required by the selection sort is $\frac{(n(n-1))}{2}$but we have introduced the OSSA which performs less number of comparisons and it becomes $\frac{(n(n-1))}{4}$ [1]. Let assume that there are m rows in a two-dimensional array. So $\frac{(nm(n-1))}{4}$ times comparison will be required to sort all the rows of the array. Now after sorting each row of the array in descending order by the above mechanism we will now search the maximum element in the first column. For this purpose the number of comparison needed is $(m-1)$ and there are m rows and n columns. So for all the comparison in the array it needs $mn(m-1)$. After finding the element in the first column it will be transferred to the target array. Therefore the total number of comparison needed for the two-dimensional array to be sorted using OSSA for two dimensional array will be:

$$f(m,n) = \frac{nm(n-1)}{4} + nm(m-1) \tag{1}$$

$$f(m,n) = \frac{((mn^2 - mn) + 4(nm^2 + nm))}{4}$$

The time complexity for the algorithm after ignoring the constant and lower terms will be: $O(mn^2 + nm^2)$

In [6] the efficiency of old selection sort algorithm for one-dimensional array having the same number of elements as that with the two-dimensional array is found as follows. For one dimensional array the number of comparison needed by selection sort;

$$\frac{(mn-1)mn}{2} \tag{2}$$

For two-dimensional array with equal number of elements as that of one-dimensional array the number of comparisons are;

$$\frac{nm(n-1)}{2} + mn(m-1) \tag{3}$$

The efficiency Calculated for efficient two dimensional sorting algorithms in [10]is;

$$\frac{\frac{mn}{2}(mn-1)}{\frac{mn(n-1)}{2} + mn(m-1)}$$
$$= \frac{(mn-1)}{(n+2m-3)}$$
$$\approx \frac{mn}{n+m} \tag{4}$$

Now calculating the efficiency of optimized selection sort algorithm for two-dimensional array, against the efficiency of optimized selection sort for one dimension array;

$$\frac{\frac{mn}{4}(mn-1)}{\frac{mn(n-1)}{4} + mn(m-1)}$$
$$= \frac{(mn-1)}{(n+4m-5)}$$
$$\approx \frac{mn}{n+4m} \tag{5}$$

Again calculating the efficiency of optimized selection sort for two-dimensional array against old selection sort for the same number of data items;

$$\frac{\frac{mn}{2}(n-1)}{\frac{mn(n-1)}{4} + mn(m-1)}$$
$$= \frac{2(mn-1)}{(n+4m-5)}$$
$$\approx \frac{2mn}{n+4m} \tag{6}$$

Similarly calculating the efficiency of optimized selection sort of two-dimensional against the efficient selection sort for two-dimensional array; from equation and we have:

$$\frac{\frac{mn}{2}(n-1) + (m-1)mn}{\frac{mn(n-1)}{4} + mn(m-1)}$$
$$\frac{2(n+2m-3)}{(n+4m-5)}$$

$$\approx \frac{2(n+2m)}{n+4m} \tag{7}$$

Now consider if we have the values of $n = 10000$ and the value of $m = 10000$ After putting the values in equation we

$$\frac{2mn}{n+4m}$$
$$= \frac{2(10^8)}{10^4 + 4(10^4)}$$
$$= 400$$

The above result shows that our algorithm is 400 times efficient than the ordinary selection sort for one dimensional array having same amount of data. if we put the values in the equation:

$$\frac{2(n+2m)}{n+4m}$$
$$= \frac{2(10^4 + 2(10^4)}{10^4 + 4(10^4)}$$
$$= 1.2$$

The above result shows that this proposed algorithm is still more efficient than the already existing algorithm for two-dimensional array having same amount of data[10].

IV. Conversion of Single Dimensional Array to Two Dimensional Array

It has been observed in the previous section that the sorting of two-dimensional array is far more efficient than the one-dimensional array for the same number of elements. In this section we have tried to convert a one-dimensional array with constant elements K to a two-dimensional array having (m n) data elements. According to equation:

$$f(m,n) = \frac{nm(n-1)}{4} + nm(m-1)$$
$$f(m,n) = \frac{nm}{4}(n + 4m - 5)$$
$$And K = (n \times m)$$
$$f(m,n) = \frac{K}{4}(n + 4m - 5)$$

Since K is a constant and $m, n >= 1$ so according to geometric inequality;

$$\frac{K}{4}(4\sqrt{(n \times 4m - 5)})$$

When $n = 4m = \sqrt{4K}$ then the values of $f(m,n)$is the minimum value.

$$\frac{K}{4}(4\sqrt{(4K-5)}) \tag{8}$$

From the above equation we can calculate the number of comparison of two-dimensional arrays and found that it has the lowest value. As the number of comparison of one-dimensional array is given by $K(K-1)/4$, so if the number is increased by 100 times so we need $100(100-1)/4$, almost 2500 times more comparisons as that of the two dimensional array which is $100/4(4(\sqrt{((4100))} - 5)$, 1875 times which is much more less than the Optimized Selection Sort Algorithm for one dimensional array. We can calculate the values of m,and n by the following method.

$$m = \sqrt{\frac{(4K)}{4}}$$

$$m = int(m)$$

$$n = 4m$$

$$If(m > (int)m)$$

$$m = (int)m + 1$$

$$If(m > (int)m)$$

$$m = (int)m + 1$$

$$n = \frac{K}{m}$$

$$If(n > (int)n)$$

$$n = (int)n + 1$$

Some times it might happen that the calculated value of $(mn) > K$, so we need to fill the empty cells of the array with imaginary values depending upon the data to be sorted. If the data is sorted in the descending then we will fill the empty cells with infinitesimal values and with infinite values otherwise.

V. Results and Discussion

In this section, Optimized Selection Sort Algorithm for Two Dimensional Arrays is compared with other existing algorithm of the same computational complexity. In the Table 1, the comparison between Simple Selection Sort, Optimized Selection Sort for Single Dimensional Arrays and Optimized Selection Sort Algorithm for Two Dimensional Arrays are presented which are as under:

A. Comparison of Selection, Optimized Selection and Optimized Selection Sort Algorithm for Two Dimensional Arrays

This section represents the comparison of old selection sort with the optimized variant of the selection sort algorithms as presented in Table 1.

When we plot the graph of Selection Sort Algorithm (SSA) and Optimized Selection Sort Algorithm for Two Dimensional Array (OpSSA2D), we can see that the performance is improved. The analysis further shows that SSA take 100 seconds to complete a sorting job, while the same is done in 12.41 seconds by OpSSA2D. These results are presented in figure 1.

TABLE I. Results of Selection Sort, OSSA for One Dimenional Array and OpSSA

K	Slection Sort	OSSA 1D	OpSSA
1000.0	499500.0	249750	61995.55
5000.0	12497500.0	6248750.0	700856.8
10000.0	49995000.0	24997500.0	1987500
15000.0	1.12E+08.0	56246250.0	3655485
20000.0	2E+08.0	99995000.0	5631854.0

TABLE II. Results of Insertion Sort, and Optimized Selection Sort Algorithm for 2D Array

K	Insertion Sort	OpSSA
1000.0	499400.0	61995.55
5000.0	12496500.0	700856.8
10000.0	49994000.0	1987500
15000.0	1.10E+08.0	3655485
20000.0	1.9E+08.0	5631854.0

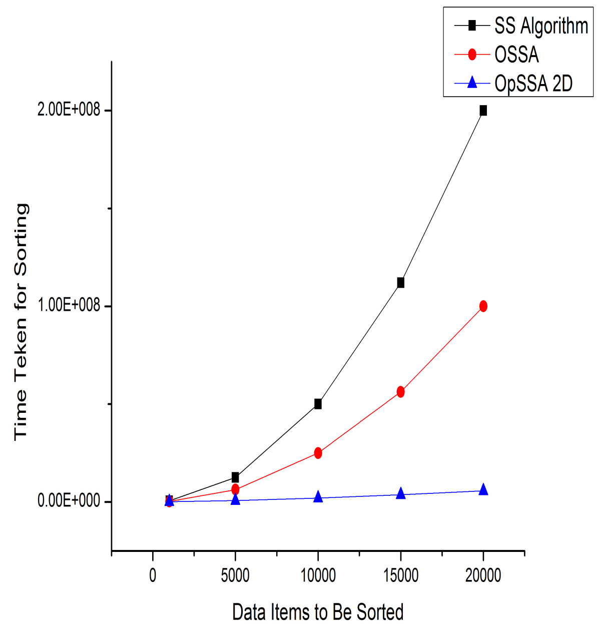

Fig. 1. Results of Selection Sort, OSSA for One Dimensional Array and OpSSA2D

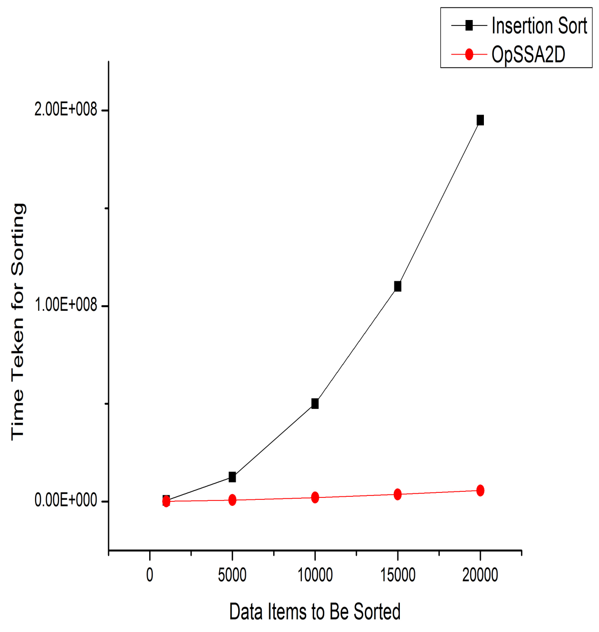

Fig. 2. Results of Insertion Sort, and Optimized Selection Sort Algorithm for 2D Array

B. Comparison of Insertion Sort and Optimized Selection Sort Algorithm for Two Dimensional Arrays

The comparison of Insertion Sort and optimized selection sort algorithm for two dimensional arrays is presented in this section. These results are shown in Table 2.

The results of insertion sort and OpSSA2D are plotted and presented in figure 2. These results show that the time taken by OpSSA2D is much lesser as compared with insertion sort. Both these algorithms belongs to the same family of comparison based sorting algorithms. It is cleared from the figure that the performance of Optimized Selection for Two Dimensional array is much better in comparison with insertion sort.

C. Comparison of Efficient Selection Sort and Optimized Selection Sort Algorithm for Two Dimensional Arrays

The Section describes the comparison of proposed and already existing algorithm. Table 3 contains the results.

The efficient selection sort is another implementation for selection sort. The graph contains the comparison of efficient selection sort (EfSSA) and with OpSSA2D. The results shows that OpSSA2D is much better that than EfSSA.

Although the computational complexity of both the algorithms are $O(n^2)$, but the performance of OpSSA2d even get better and better when the data volume is high. Despite the computational complexity the results of Optimized Selection Sort Algorithm for two Dimensional Array are promising and support our claim to be better than other comparison based sorting algorithms.

TABLE III. Results of Efficient Selection Sort, and Optimized Selection Sort Algorithm for 2D Array

K	Efficient Sort	OpSSA
1000.0	123991.1	61995.55
5000.0	1401714	700856.8
10000.0	3975000	1987500
15000.0	7310969	3655485
20000.0	11263708	5631854

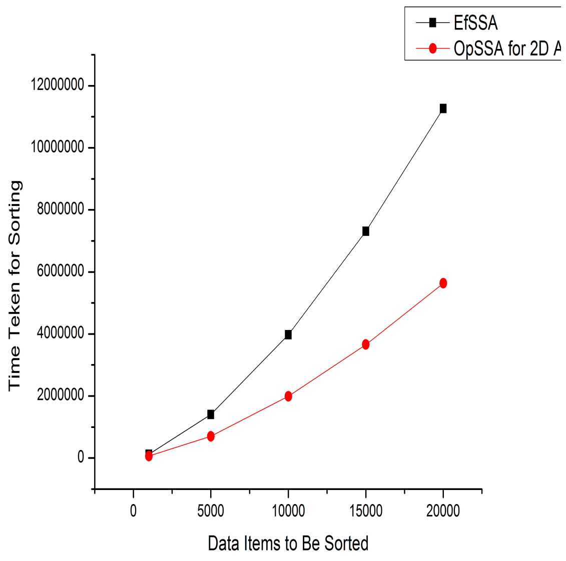

Fig. 3. Comparison of Efficient Selection Sort and Optimized Selection Sort Algorithm for Two Dimensional Arrays

VI. Conclusion

Optimized Selection Sort Algorithm for two dimensional array belongs to the family of comparison based sorting algorithms. It is stable and very simple to implement. It is evident from the above results that optimized selection algorithm for two-dimensional arrays is more efficient than the other sorting algorithm of the same complexity for the same amount of data. Hereafter proved our claim to be an efficient algorithm of the order of $O(n^2)$. It could be useful algorithm when it needs to solve the problem of sorting huge volume of data in reasonably easy manner and efficiently.

References

[1] D. Kitabı, “Th cormen, ce leiserson, rl rivest and c. stein: Introduction to algorithms,” 2001.

[2] C. L. Muller and C. Kidd, “Debugging geographers: teaching programming to non-computer scientists,” *Journal of Geography in Higher Education*, vol. 38, no. 2, pp. 175–192, 2014.

[3] S. Matsuoka, H. Sato, O. Tatebe, M. Koibuchi, I. Fujiwara, S. Suzuki, M. Kakuta, T. Ishida, Y. Akiyama, T. Suzumura *et al.*, “Extreme big data (ebd): Next generation big data infrastructure technologies towards yottabyte/year,” *Supercomputing frontiers and innovations*, vol. 1, no. 2, pp. 89–107, 2014.

[4] F. Kaashoek, R. Morris, and Y. Mao, “Optimizing mapreduce for multicore architectures,” 2010.

[5] S. Jadoon, S. F. Solehria, S. Rehman, and H. Jan, “Design and analysis of optimized selection sort algorithm,” *International Journal of Electric & Computer Sciences (IJECS-IJENS)*, vol. 11, no. 01, pp. 16–22, 2011.

[6] M. Khan, A. Khan, M. Khan, and S. Anwar, “A novel learning method to classify data streams in the internet of things,” in *Software Engineering Conference (NSEC), 2014 National*, Nov 2014, pp. 61–66.

[7] S. Jadoon, S. F. Solehria, and M. Qayum, “Optimized selection sort algorithm is faster than insertion sort algorithm: a comparative study,” *International Journal of Electrical & Computer Sciences IJECS-IJENS*, vol. 11, no. 02, pp. 19–24, 2011.

[8] S. Sumathi, A. Prasad, and V. Suma, “Optimized heap sort technique (ohs) to enhance the performance of the heap sort by using two-swap method,” in *Proceedings of the 3rd International Conference on Frontiers of Intelligent Computing: Theory and Applications (FICTA) 2014*. Springer, 2015, pp. 693–700.

[9] J. Singh and R. Singh, “Merge sort algorithm,” *International Journal of Research*, vol. 1, no. 10, pp. 1203–1207, 2014.

[10] M. Zhou and H. Wang, “An efficient selection sorting algorithm for two-dimensional arrays,” in *Genetic and Evolutionary Computing (ICGEC), 2010 Fourth International Conference on*. IEEE, 2010, pp. 853–855.

[11] A. Chadha, R. Misal, T. Mokashi, and A. Chadha, “Arc sort: Enhanced and time efficient sorting algorithm,” *arXiv preprint arXiv:1406.2262*, 2014.

[12] W. Min, “Analysis on 2-element insertion sort algorithm,” in *Computer Design and Applications (ICCDA), 2010 International Conference on*, vol. 1. IEEE, 2010, pp. V1–143.

[13] M. F. Umar, E. U. Munir, S. A. Shad, and M. W. Nisar, “Enhancement of selection, bubble and insertion sorting algorithm,” 2014.