Selected Two's Elements at a Time for Improving the Performance of Selection Sorting Algorithm

Anuradha Brijwal[1], Arpit Goel[2] ,Wajahat Gh mohd[3]
[1] Dept. Of Computer Science & Engineering, Himalayan Institute of Technology, Dehradun Uttarakhand (India)
[2] Dept. Of Computer Science & Engineering, Himalayan School of Engineering & Technology, Swami Rama Himalayan University, Dehradun Uttarakhand (India)
Dept. of computer science and engineering, JB institute of technogy Dehradun Uttrakhand.
[1]meetanubrijwal01@gmail.com, [2]arpitgoel29@gmail.com, wajahatiust040@gmail.com[3]

Abstract-
Sorting is an algorithm that arranges all elements of an array, orderly. Sorting involves rearranging information into either ascending or descending order. In computer science and mathematics, a sorting algorithm is an algorithm that puts elements of a list in a certain order, not necessarily in increasing order; it may be in decreasing order as well. Efficient sorting is important to optimizing the use of other algorithms that require sorted list to work efficiently; it is also useful for producing human-readable output. Most simple sorting algorithms involve two steps which are compare two items and swap two items or copy one item. In this paper we present a new sorting algorithm, named as Dual-Element Selection Sorting, which is faster than selection sort. After studying various sorting algorithms; I found that there are no such sorting algorithms which work on the basis of selecting two elements at a time, means selecting two elements simultaneously. We also compare Dual-Element Selection Sort algorithm with selection sort. We have used the MATLAB for implementation. The new algorithm is analyzed, implemented & tested.

Keywords: Algorithm, Selection and Dual-Element Selection Sort, Comparison.

I. INTRODUCTION

There are many fundamental and advance sorting algorithms. All sorting algorithm are problem specific means they work well on some specific problem and do not work well for all the problems. All sorting algorithm apply to specific kind of problems. Some sorting algorithm apply to small number of elements, some sorting algorithm suitable for floating point numbers, some are fit for specific range, some sorting algorithms are used for large number of data, some are used if the list has repeated values [6].

For instance, if the given input of numbers is (99, 61, 41, 51, 6, 78), then the output sequence returned by a sorting algorithm will be (6, 41, 51, 61, 78, 99).

One of the fundamental problems of computer science is ordering a list of items [9]. There is a plethora of solutions to this problem, known as sorting algorithms. Some sorting algorithms are simple and intuitive, such as the bubble sort. Others, such as the quick sort are extremely complicated, but produce lightning-fast results. The common sorting algorithms can be divided into two classes by the complexity of their algorithms. There is a direct correlation between the complexity of an algorithm and its relative efficiency.

Sorting is one of the most important and well-studied problems in computer science. Many good algorithms are known which offer various trade-offs in efficiency, simplicity, memory use, and other factors. However, these algorithms do not take into account features of modern computer architectures that significantly influence performance. A large number of sorting algorithms have been proposed and their asymptotic complexity, in terms of the number of comparisons or number of iterations, has been carefully analysed [10]. In the recent past, there has been a growing interest on improvements to sorting algorithms that do not affect their asymptotic complexity but never the less improve performance by enhancing data locality.

Sorting is a fundamental task that is performed by most computers. It is used frequently in a large variety of important applications. Database applications used by schools, banks, and other institutions all contain sorting code. Because of the importance of sorting in these applications, dozens of sorting algorithms have been developed over the decades with varying complexity. Slow sorting methods such as bubble sort, insertion sort, and selection sort have a theoretical complexity of O (n^2) [11]. Even though these algorithms are very slow for sorting large arrays, the algorithm is simple, so they are not useless. If an application only needs to sort small arrays, then it is satisfactory to use one of the simple slow sorting algorithms as opposed to a faster, but more complicated sorting algorithm [12]. For these applications, the increase in coding time and probability of coding mistake in using the faster sorting algorithm is not worth the speedup in execution time. Of course, if an application needs a faster sorting algorithm, there are certainly many ones available, including quick sort, merge sort, and heap sort. These algorithms have a theoretical complexity of O (n log n). They are faster than the O (n^2) algorithms and can sort large arrays in a reasonable amount of time. However, the cost of these fast sorting methods is that the algorithm is much more complex and is harder to correctly code. But the result of the more complex algorithm is an efficient sorting method capable of being used to sort very large arrays [13].

But sometimes question arises in front of us, whether there any way through selection sorting can be more

effective and how to convert that algorithm into code [14]. Then demonstrate a modification of this algorithm, and finally to assign the coding modification as a programming. This paper suggests one simple modification of sorting algorithm: Dual-Element Selection Sort. One can argue as to whether the use of Dual-element selection sort for these small array partitions will provide improvement to this critical algorithm.

Therefore, to understand important concepts and programming practice, a good programming exercise plays a crucial role i.e. for using Dual-element selection sort in place of normal selection sorting technique that raises the sorting skills.

An effort is done in positive direction and realizes coding technique for Dual sorting offer great improvements speed up over the single selection sorting.

II. SELECTION SORT

The selection sort works by selecting the smallest unsorted item remaining in the list, and then swapping it with the item in the next position to be filled. The selection sort has a complexity of O (n^2) [14].

The worst case as well as average case complexity of Selection sort is O(n^2), where n represents the total number of items in the given array to be sorted.
The selection sort is the unwanted step child of the n^2 sorts. It yields a 60% performance improvement over the bubble sort, but the insertion sort is over twice as fast as the bubble sort and is just as easy to implement as the selection sort. In short, there is not really any reason to use the selection sort-use the insertion sort instead [15].

The algorithm for selection sort having *ARRAY* as an array with N elements is as follows:

```
SELECTION (ARRAY, N)
for (i=1 to N-1)
{
        Min = ARRAY[i]
        for (k = i+1 to N)
        {
                if  (min > ARRAY [k])
                {
                        Min = A[k]
                        Loc = k
                }
        }
        Temp = ARRAY [Loc]
        ARRAY [Loc] = ARRAY [i]
        ARRAY [i] = Temp
}
```

III DUAL ELEMENT SELECTION SORT

A. Introduction

Various authors had made continuous attempts for increasing the efficiency and performance of the sorting process. The proposed algorithm is based on selection sort.
The proposed algorithm as:
Starts from two elements and searches the entire list until it finds the minimum value and second minimum value. The sorting places the minimum value in the first place and second minimum value in the second place, this process continues until the complete list is sorted. In other words, the proposed algorithm designed to minimize the number of passes/comparisons that are performed. It works by making N/2 passes over the shrinking unsorted portion of the array, each time selecting the smallest and second smallest value. Those values are then moved into their final sorted position in one pass.

B. Algorithm

```
DESS (ARRAY, n)
for (i = 1; i <= n-1; i = i+2)
{
        Min = ARRAY [i];
        Smin = ARRAY [i+1];
        Loc_Min = i;
        Loc_Smin = i+1;

        for (j = i+1; j <= n; j++)
        {
                If (ARRAY [j] < Min)
                {
                        Smin = Min;
                        Loc_Smin = Loc_Min;
                        Min = ARRAY [j];
                        Loc_Min = j;
                 }
                elseif (ARRAY [j] < Smin)
                {
                        Smin = ARRAY [j];
                        Loc_Smin = j;
                }
        }
        if (Loc_Min ~= i)
        {
                temp = ARRAY [i];
                ARRAY [i] = ARRAY [Loc_Min];
                ARRAY [Loc_Min] = temp;
        }
        if (Loc_Smin == i)
        {
                Loc_Smin = Loc_Min;
        }
        if (Loc_Smin ~= i+1)
        {
                temp1 = ARRAY [i+1];
```

```
                ARRAY [i+1] = ARRAY [Loc_Smin];
                ARRAY [Loc_Smin] = temp1;
        }
}
```

C. Complexity Analysis

The general working of the proposed algorithm is already discussed in detail. Now discuss its complexity analysis.

TABLE I
COMPLEXITY ANALYSIS

Line No.	Iteration
1.	for(i=1;i<=n-1;i=i+2)
2.	Min = ARRAY[i]
3.	Smin = ARRAY [i+1];
4.	Loc_Min = i;
5.	Loc_Smin = i+1;
6.	for(j=i+1;j<=n;j++)
7.	if(ARRAY [j] < Min)
8.	Smin = Min;
9.	Loc_Smin = Loc_Min;
10.	Min = ARRAY [j];
11.	Loc_Min = j;
12.	elseif(ARRAY [j]<Smin)
13.	Smin = ARRAY [j];
14.	Loc_Smin = j;
15.	if(Loc_Min ~= i)
16.	temp = ARRAY [i];
17.	ARRAY [i] = ARRAY [Loc_Min];
18.	ARRAY [Loc_Min] = temp;
19.	if(loc1 == i)
20.	Loc_Smin = Loc_Min;
21.	if(Loc_Smin ~= i+1)
22.	temp1 = ARRAY [i+1];
23.	ARRAY [i+1] = ARRAY [Loc_Smin];
24.	ARRAY [Loc_Smin] = temp1;

Line 1 execute n/2 + 1 time in a single execution of the algorithm.
Line 2-6 executes n/2 times in a single execution of the algorithm.
Line 7-14 executes

$$\sum_{K=0}^{n/2-1} (2k+1).t \quad \text{if n is even}$$

$$\sum_{K=0}^{n/2-1} (2k+2).t \quad \text{if n is odd}$$

times in a single execution of the algorithm.
Line 15-24 executes n/2 times in a single execution of the algorithm.

Note: t=1 when if statement is true, else t=0.

$$\sum_{K=0}^{n/2-1} (2k+1).t$$

Suppose for t=1 we have

$$\sum_{K=0}^{n/2-1} (2k+1) = 2\sum_{K=0}^{n/2-1} k + \sum_{K=0}^{n/2-1} 1$$

$$= 2(((n/2-1)*((n/2-1)+1))/2) + (n/2-1)$$

[By Applying $1+2+3+\ldots n = (n(n+1)/2)$]

$$= ((n^2-2n)/4) + (n/2-1)$$

Now calculating total time taking by proposed algorithm,

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

Now taking only the dominant term, i.e. n^2 the running time of the algorithm is,

$$T(n) = O(n^2)$$

III. WORKING

Let the given set of elements are 97, 43, 58, 84, 23, 76.

A. *Selection Sort*

TABLE II
WORKING OF SELECTION SORT

Passes	Elements									
Initial	3	5	6	8	2	9	4	1	7	0
1.	0	5	6	8	2	9	4	1	7	3
2.	0	1	6	8	2	9	4	5	7	3
3.	0	1	2	8	6	9	4	5	7	3
4.	0	1	2	3	6	9	4	5	7	8
5.	0	1	2	3	4	9	6	5	7	8
6.	0	1	2	3	4	5	6	9	7	8
7.	0	1	2	3	4	5	6	9	7	8
8.	0	1	2	3	4	5	6	7	9	8
9.	0	1	2	3	4	5	6	7	8	9

B. *DESS Sort*

TABLE III
WORKING of DESS SORT

Passes	Elements									
Initial	3	5	6	8	2	9	4	1	7	0
1.	0	1	6	8	2	9	4	5	7	3
2.	0	1	2	3	6	9	4	5	7	8
3.	0	1	2	3	4	5	6	9	7	8
4.	0	1	2	3	4	5	6	7	9	8
5.	0	1	2	3	4	5	6	7	8	9

IV. COMPARISION

A. *Comparison of Proposed Algorithm with Selection Sort*

TABLE IV
WORST CASE ANALYSIS (ON THE BASIS OF NUMBER OF COMPARISIONS)

Size of input	Selection Sort	DESSA	% Improvement
N=50	1225	625	48%
N=99	4851	2450	49%
N=150	11175	5625	49%
N=499	124251	62250	49.9
N=1000	499500	250000	50%
N=4999	12492501	6247500	50.1%
N=10000	49995000	25000000	50.3%

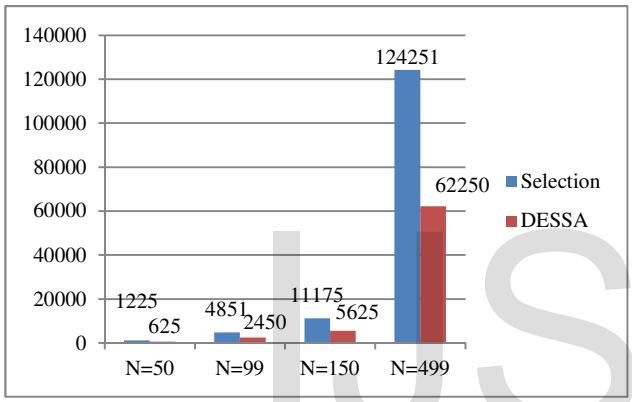

Fig. 1 Analysis On the Basis of Number of Comparisons

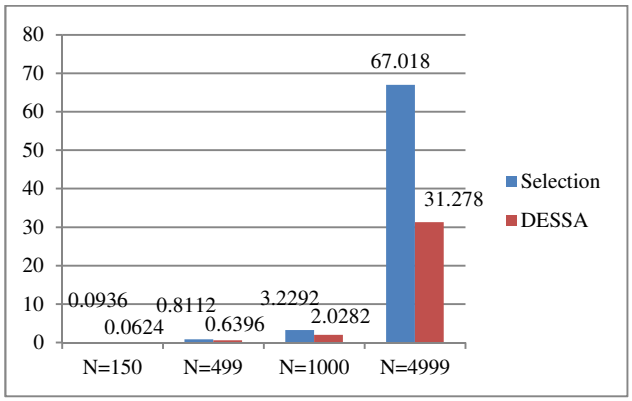

Fig. 2 Analysis On the Basis of Execution Time

B. *Comparison of Proposed Algorithm with Fundamental Sorting Technique*

TABLE VII
ON THE BASIS OF NUMBER OF PASSES

Size of Input	Selection	Bubble	Insertion	DESSA
N=50	49	49	49	2
N=99	98	98	98	49
N=150	149	149	149	75
N=499	498	498	498	249

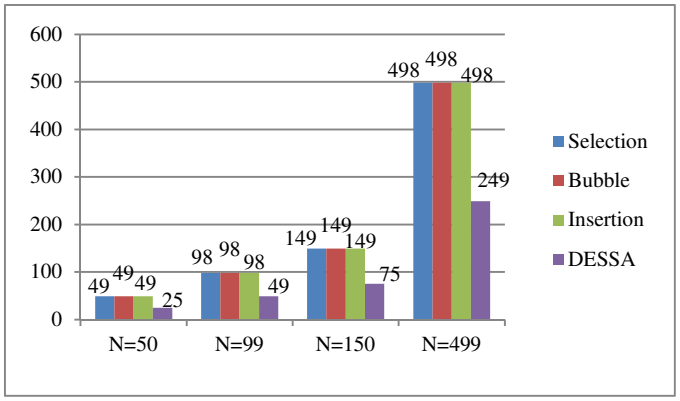

Fig. 3 Analysis On the Basis of Number of Passes

VI. CONCLUSION & FUTURE SCOPE

In this research paper we have studied about different sorting algorithms along with their comparison. Every sorting algorithm has advantage and disadvantage. The fundamental sorting algorithms are basic sorting algorithm and we have try to show this how disadvantage of fundamental sorting algorithm have removed in advance sorting algorithm. Various Sorting algorithms have been compared on the basis of different factors like complexity, number of passes, number of comparison etc. After the study of all various sorting algorithms we observed that there is no such algorithm, which works in this way that to sort the two elements at a time. So we have proposed sorting algorithm, which work on the basis of selecting two elements simultaneously. For implementation to the proposed algorithm we have to use MATLAB.
My first target is to remove the demerits of various sorting algorithms. It is also seen that many algorithms are problem oriented so we will try to make it global oriented. Hence we can say that there are many future works which are as follows.

- Remove disadvantage of various fundamental sorting and advance sorting.
- Make problem oriented sorting to global oriented.

In the end we would like to say that there is huge scope of the sorting algorithm in the near future, and to find optimum-sorting algorithm, the work on sorting algorithm will go on forever.

REFERENCES

[1] Y.Han "Deterministic sorting in O (nloglogn) time and linear space", Proceeding of the thirty-fourth annual ACM symposium on theory of computing, Monteral Quebec, Canada, **(2002)**, p.602-608.

[2] Y.Han, M.Thorup, "Integer Sorting in O(nloglogn) time and linear space" proceesings of the 43rd symposium on foundations of Computer Science, **(2003)**, p.135-144.

[3] J.L. Bentley and R.Sedgewick. "Fast Algorithms for Sorting and Searching Strings", ACM-SIAM SODA "**(2003)**, 360–369.

[4] G. Franceschini and V. Geffert, "An In-Place Sorting with O(n log n) Comparisons and O(n) Moves", Proceedings of 44th Annual IEEE Symposium on Foundations of Computer Science, **(2003)**, pp. 242-250.

[5] M. A. Bender, M. Farach-Colton and M. A. Mosteiro, "Insertion Sort is O(n log n)", Proceedings of the Third International Conference on Fun With Algorithms (FUN), **(2004)**, pp. 16-23.

[6] A. D. Mishra and D. Garg, "Selection of the best sorting algorithm", International Journal of Intelligent Information Processing, vol. 2, no. 2, **(2008)** July-December, pp. 363-368.

[7] O. O. Moses, "Improving the performance of bubble sort using a modified diminishing increment sorting", Scientific Research and Essay, vol. 4, no. 8, **(2009)**, pp. 740-744.

[8] J. Alnihoud and R. Mansi, "An Enhancement of Major Sorting Algorithms", International Arab Journal of Information Technology, vol. 7, no. 1, **(2010)**, pp. 55-62.

[9] Sultanullah Jadoon et al., "Design and Analysis of Optimized Selection Sort Algorithm", International Journal of Electric & Computer Sciences IJECS-IJENS, **(2011)**Vol: 11 No: 01.

[10] Savina & Surmeet Kaur, "Study of Sorting Algorithm to Optimize Search Results", International Journal of Emerging Trends & Technology in Computer Science, **(2012)** Volume 2, Issue 1.

[11] Md. Khairullah, "Enhancing Worst Sorting Algorithms", International Journal of Advanced Science and Technology, **(2013)** Vol. 56.

[12] Ibrahim M. Al Turani, Khalid S. Al-Kharabsheh & Abdallah M. AlTurani, "Grouping Comparison Sort", Australian Journal of Basic and Applied Sciences, **(2013)**, 7(7): 470-475.

[13] Nitin Arora, Anil Kumar &Pramod Mehra, "Two Way Counting Position Sort", International Journal of Computer Application **(2013)**.

[14] Partha Sarathi Dutta , " Design and Analysis of Hybrid Selection Sort Algorithm", International Journal of Applied Research and Studies **(2013)**.

[15] Surender Lakra & Divya, "Improving the performance of selection sort using a modified Dual-ended selection sorting", International Journal of Application or Innovation in Engineering & Management (IJAIEM) **(2013)**.

[16] Pankaj Sareen, "Comparison of Sorting Algorithms", International Journal of Advanced Research in Computer Science and Software Engineering **(2013)**.

[17] R.Srinivas & A.RagaDeepthi, "Novel Sorting Algorithm", International Journal on Computer Science and Engineering **(2013)**.

[18] Partha Sarathi Dutta, "An Approach to Improve the Performance of Insertion Sort Algorithm", International Journal of Computer Science & Engineering Technology **(2013)**.

[19] Khalid Suleiman Al-Kharabsheh et al., "Review on Sorting Algorithms A Comparative Study", International Journal of Computer Science and Security (IJCSS), **(2013)**, Volume (7) : Issue (3).

[20] Savina & Surmeet Kaur, "Study of Sorting Algorithm to Optimize Search Results", International Journal of Emerging Trends & Technology in Computer Science, **(2013)** Volume 2, Issue 1.