ISSN:0975-9646

CSB Sorting Algorithm

RinkleAswani, Anjali Arora, PallaSujitha

Department of Computer Science, G. B. Pant Engineering College, Okhla, Delhi, India.

***Abstract*- Sorting is a term which is very commonly used in computer science. It is a process of arranging items according to a certain order or in a particular sequence. The algorithm which are used for sorting generally consist of operations that includes comparison, swapping and assignment, as well as it may facilitate some other operations such as searching, arranging and merging of data. In this paper we proposed a new algorithm which is a combination of selection sort and bubble sort. During each iteration the smallest and largest elements are placed at its appropriate location. In our proposed approach, we reduces the running time complexity of selection sort in best case scenario and also improving the running time of bubble sort in worst cases .Observations and results are reported in support of the proposed idea.**

Keywords: **Combined Selection Bubble Sort, Bubble sort, Selection sort, Best Case, Run time Complexity.**

I. INTRODUCTION

Sorting is a term which refers to the arrangement of elements into a sequence or in some kind of order from a collection of unsorted data is termed as Sorting. For example, a list of names can be arranged in their lexicographic order; a list of eligible candidates can be arranged according to their experiences.

Sorting has been the major area of research in computer science as it is one of the important factors that helps in optimization of other algorithms such as linear search & binary search[1] etc. Some algorithms are simple which uses less number of resources and are fast. Some are spontaneous but complex. Before going into details of specific algorithms, we should know some of the operations that are used to analyze a sorting process i.e. in order to sort a collection of unsorted elements, it is necessary to have some systematic way to arrange the elements. First of all, it is necessary to compare two elements to check which element is smaller (or larger). Therefore the total number of comparisons will be the most common way to measure a sorting process. Secondly when elements are not in their proper order with respect to one another, then it is necessary to exchange them because it incurs a lot of cost to perform these operations. So there is the requirement of an algorithm which minimizes the number of exchanges to improve the overall efficiency of the algorithm [3]. However there exists a direct relationship between the algorithm's complexity and its effectiveness[8].

We have abundant of searching and sorting algorithms that have been developed to sort and search the elements in different fashion. The number of swaps in any sorting algorithms defines the computational complexity. If we compute an ample number of elements than it will take essential amount of resources which leads to increase the computing time and directly accelerate the overhead. Similarly if we take smaller list then the computing resources and time takes less overhead [3].

There are different cases of measuring the performance of sorting algorithms i.e. best case, average case & worst case. For example, the best case scenario for a simple linear search on an array occurs when the desired element is the first element in the list. Average case is similar to worst case, but in worst case scenario where the element to be searched is present at the end of the list than the number of swaps are comparatively greater than the average case [1].

Before selecting the algorithm we have to look up on certain factors such as memory space required for storing elements, computational resources, computational time and number of comparisons etc. As we know that their does not exist a single algorithm which can overcome all the problems, so to select the algorithm we have to consider all the factors and choose the best algorithm accordingly[2].

II. RELATED WORK

A. SELECTION SORT

As the name implies selection sort selects the elements on the basis of comparison and swapping. The working principal of selection sort algorithm in first phases that it selects the first element in the array assigning as the first position(assuming the arrangement of elements are in ascending order)and then it compares the value of the first position with remaining elements in the list. If the first element is smaller than it sets first position in the array else it swaps the value of initial position with the smallest element present in the array and it sets the first position.

Similarly in the second pass it compares the elements from the second position (i.e. First element is already being sorted), and set the position accordingly.

The selection sort algorithm:

Algorithm: Selection Sort (a[], n)

Here a is the unsorted input list and n is the size of the array.

1. Repeat steps 2 to 4 from i=0 to n-1
2. Set min=a[i]
3. Repeat for count=i+1 to n
   If (a[count]<a[min])
   Set min=count
   End if
4. Interchange data at location a[count] and min

The above algorithm takes two loops one within the other, the outer loop contains n number of comparisons and the inner loops takes n-1 comparisons. Adding the number of comparison we get:-

(n-1)+(n-2)+.....+3+2+1=n (n-1) /2 …… (i)

The comparison of shaded positions is done and the smallest of the compared values is placed at the desired location. After this phase the first element is the smallest one placed at appropriate location shown below (***Figure 1***).

24	36	**5**	18	2	35	9	7
24	36	**5**	18	2	35	9	7
5	36	24	18	2	35	9	7
5	36	24	18	**2**	35	9	7
2	36	24	18	5	35	9	7
2	36	24	18	5	35	9	7
2	36	24	18	5	35	9	7
2	36	24	18	5	35	9	7

FIGURE 1: SHOWS THE FIRST PHASE OF SELECTION SORT.

The best case scenario of selection sort gives the complexity of **$O(n^2)$.** Nonetheless when the list is in sorted order as it has to perform n(n-1)/2 comparisons[4]. In average and worst cases the number of comparisons remains the same so its complexity remains the same because of which, it is not suitable for handling large files[5]. Complexity of the selection sort algorithm is shown in table I.

Best Case	Average Case	Worst Case
$O(n^2)$	$O(n^2)$	$O(n^2)$

TABLE I: COMPLEXITY OF SELECTION SORT.

B. BUBBLE SORT

Bubble sort algorithm works on the basis of comparison and swapping. In the first phase of bubble sort, starting element of the list is compared with the very next element of the list and so on. It arranges the list with continuous interchanging of elements if necessary due to this the largest element is placed in its appropriate location means it will be the last element in the list(assuming the list is sorted in ascending order). The above method is repeated until all the elements of the list are sorted[3].

Algorithm: Bubble Sort (a[], n)
Here a is the unsorted input list and n is the size of array.
1. Repeat step 2 for i = 0 to n-2
2. Repeat step 3 for j= 0 to n-i-2
3. If (a[j]>a[j+1])
   Interchange a[j] and a[j+1]
   End if

Above algorithm takes n-1 comparisons in one iterations as well as n-1 passes to sort the unsorted input list. The best case scenario of bubble sort gives the complexity of **$O(n^2)$.**Nonetheless when the list is in sorted order then also it has to perform (n-1) comparisons. In average and worst cases the number of comparisons remains the same so its complexity remains the same[3][7].

The comparison of shaded positions is done and the largest of the compared values is placed at the latter location, so that they are in proper order. After this phase the last element is the largest one bubbled to its appropriate location shown below (***Figure 2***):

24	**36**	5	18	2	35	9	7
24	**36**	5	18	2	35	9	7
24	5	**36**	18	2	35	9	7
24	5	18	**36**	2	35	9	7
24	5	18	2	**36**	35	9	7
24	5	18	2	35	**36**	9	7
24	5	18	2	35	9	**36**	7
24	5	18	2	35	9	7	**36**

FIGURE 2: BUBBLE SORT: THE FIRST PHASE

After the first phase the111111 largest value will be in its appropriate location. In second phase we are left with n-1 elements to sort and so on . After completing entire phases, the smallest element will be placed at its appropriate location[4],[6]. Complexity of the bubble sort algorithm is shown in table II.

Best Case	Average Case	Worst Case
$O(n^2)$	$O(n^2)$	$O(n^2)$

TABLE II: COMPLEXITY OF BUBBLE SORT.

III. PROPOSED SYSTEM: CSB SORT ALGORITHM

In this algorithm we are combining two sorts that is selection and bubble sort. The CSB (Combined selection bubble sort) is based on the collective use of selection and bubble sort in one pass so that during each pass two elements are arranged at a time, the smallest one is placed to its appropriate place while bubbling up the maximum element to its appropriate place in the array.

By using the combination of both ,the number of swaps are reduced considerably, as well as it overcomes the best time complexity problem of selection sort by having the best time complexity of O(n) instead of O (n^2) .

Algorithm: CSB Sort (a[],n)

Here a is the unsorted input list and n is the length of array. After completion of the algorithm array will become sorted:-

1. fswap = true[Set the flag to initiate the loop]
2. Repeat step 3 to 7 for i= 1 to n/2 and while fswap is true.
3. swap = false;
   Min = a[i-1]
   Pos = i-1
4. Repeat step 5 to 7
   For j=i-1 to n-i-1
5. If a[j]>a[j+1]
   Swap a[j] and a[j+1]
   fswap =true.
   End If
6. If (min> a[j])
   min =a[j] and pos = j;
   Else
   If (min> a[j+1])
   min = a[j+1]
   pos = j+1
   End If
   End If
7. If pos! = i-1
   Swap(a[i-1]and a[pos])
   End If

The above algorithm takes two loops one within the other and the total number of comparisons are**:**

(n-1)+(n-3)+(n-5)+……..+1 = $n^2/4$ ……..(ii)

The comparison of shaded positions is done and the largest of the compared values is placed at the latter location, the minimum value is placed in the min variable and its location in the pos variable. At the end of the phase the minimum value is placed at its appropriate location and the largest element is bubbled to its appropriate location. Thus setting up two locations in each phase as shown below (***Figure 3***) :

Pos 1	Min 24	24	36	5	18	2	35	9	7
Pos 1	Min 24	24	36	5	18	2	35	9	7
Pos 2	Min 5	24	5	36	18	2	35	9	7
Pos 2	Min 5	24	5	18	36	2	35	9	7
Pos 4	Min 2	24	5	18	2	36	35	9	7
Pos 4	Min 2	24	5	18	2	35	36	9	7
Pos 4	Min 2	24	5	18	2	35	9	36	7
2	5	18	24	35	9	7	36		

FIGURE 3: CSB SORT: THE FIRST PHASE

After completing entire phases, all the elements will be placed at its appropriate location and complexity of the CSB sort algorithm is shown in table III.

Best Case	Average Case	Worst Case
O(n)	$O(n^2)$	$O(n^2)$

TABLE III: COMPLEXITY OF CSB SORT.

IV. RESULTS AND ANALYSIS

The following graphs demonstrate the performance of selection, bubble, and CSB sort with respect to sorted list(best case), random list (average case) and reverse sorted list(worst case) The comparisons of iterations. The CSB Algorithm has the complexity of O(n) i.e. in the best case, $O(n^2)$ i.e. in the average case and $O(n^2)$ i.e. in the worst case. The time complexity is the same but it reduces the number of swaps operations (as compared to bubble sort) and number of comparisons (as compared to bubble sort and selection sort).

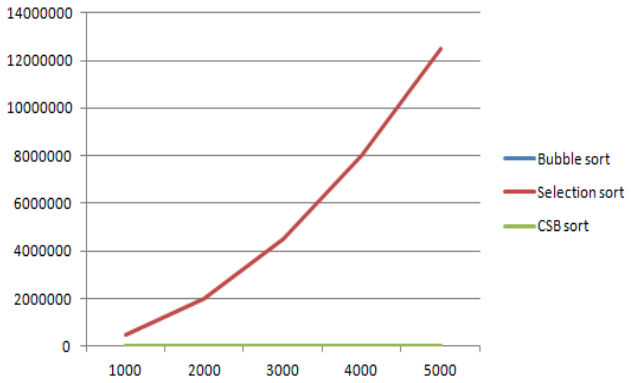

FIGURE 4: PERFORMANCE OF ALGORITHMS (SORTED LIST)

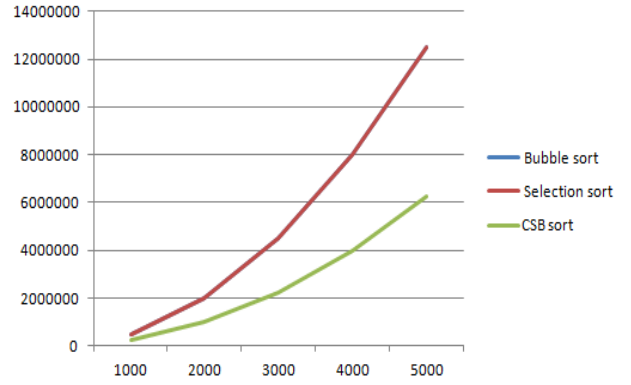

FIGURE 5: PERFORMANCE OF ALGORITHMS (RANDOM LIST)

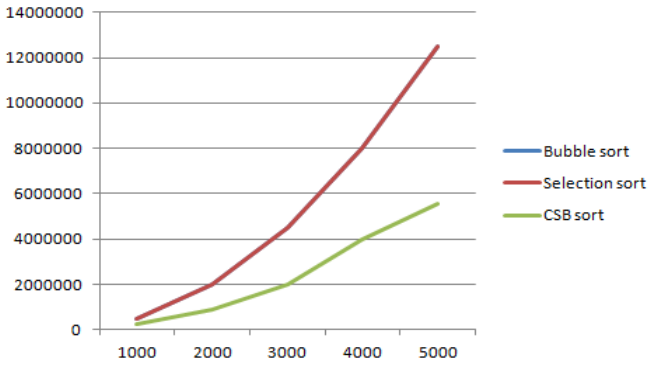

FIGURE 6: PERFORMANCE OF ALGORITHMS (REVERSELY ORDERED LIST)

V. CONCLUSION

The proposed algorithm has a significant enhancement over the original sorting algorithms. In best case scenario of selection sort, it includes unnecessary comparisons of elements when the list is in sorted order and it takes n-1 iterations, but in case of CSB Algorithm no. of iterations are reduced as well as unnecessary comparisons are avoided. This paper shows substantial advantage over classical algorithms it actually improves the running time complexity of selection sort in best case scenario and also improves the running time of bubble sort in worst cases by doing so we are actually minimizing the number of comparisons or the number of exchange which enhances the inefficient algorithms.

REFERENCE

[1] P. Dhivakar et al, Dual Sorting Algorithm Based on Quick Sort, International Journal of Computer Science and Mobile Applications, Vol.1 Issue. 6, December- 2013, pg. 1-10,SSN: 2321-8363

[2] Md. Khairullah , Enhanced Worst Sorting Algorithm, International Journal of Advanced Science and Technology Vol. 56, July, 2013.

[3] Brad Miller and David Ranum, Welcome to Problem Solving with Algorithms &Data Structures , Available at http://interactivepython.org /courselib/static/pythonds/index.html(accessed on 1st Feb 2014).

[4] Selection Sort, Available at http://www.personal.kent.ed ~rmuhamma / Algorithms/My Algorithm/Sorting/Selectionsort.htm. (accessed on 24rd March 2014).

5] The Art of Computer Programming, Volume 3: Sorting and Searching, Third Edition. Addison–Wesley, 1997.ISBN 0-201-89685-0. Pages 138– 141 of Section 5.2.3: Sorting by Selection.

[6] Sorting by Insertion", The Art of Computer Programming, 3. Sorting and Searching (second ed.), Addison-Wesley, 1998, pp. 80–105, ISBN 0-201- 89685-0.

[7] Paul E. Black and Bob Bockholt, "bidirectional bubble sort", in Dictionary of Algorithms and Data Structures (online), Paul E. Black, ed., U.S. National nstitute of Standards and Technology. 24 August 2009. (accessed: 15 Feb 2014).

[8] R. S. Salaria, Data Structure & Algorithms 4rd Edition 2006.

AUTHORS PROFILE

Ms. Rinkle Aswani is working as a Assistant Professor in computer Science Department in G. B. Pant Engineering College, Delhi, India. She completed her ALCCS(Eqv-M.Tech.) from IETE, Delhi in 2013. Her Area of interest are Algorithms, Computer Graphics, Data Structure, Cyber Crime & laws, Networking & Agile. She has an experience of over 10 years as an academician.

Ms. Anjali Arora is working as a Assistant Professor in G. B. Pant Engineering College, Delhi, India. She completed her M.Tech. from Banasthali University, Jaipur in 2012. Her Area of interest are Algorithms, Data warehouse & mining, DBMS, Computer Networks. She has published various papers in journals and conferences.

Ms. Palla Sujitha is working as a Assistant Professor in G. B. Pant Engineering College, Delhi, India. She completed her M.Tech. from Indraprastha Engineering College, NCR Delhi in 2012. Her Area of interest are Algorithms, Data Structures, Computer Networks, Operating System. She has published various papers in international & national conferences.