International Research Journal of Computer Science (IRJCS) ISSN: 2393-9842
Issue 6, Volume 2 (June 2015) www.irjcs.com

A New Approach to Improve Worst Case Efficiency of Bubble Sort

Vipul Sharma*
Department of Computer Science & Engineering
National Institute of Technology (Srinagar, J&K)

***Abstract*— Sorting is one of the most common operations in computer science, being a fundamental operation, sorting is defined as an activity which involves rearrangement of elements in either ascending order or descending order. There are a number of sorting algorithms that have been proposed for one reason or another, but this paper focuses on bubble sort algorithm. Here in this paper I have proposed a new technique that will enhance the performance of normal bubble sort, when used in worst case scenario. Further the experimental results that have been produced by using this new technique has been compared with normal bubble sort and insertion sort and it has been found, that the performance of the proposed algorithm is much better than the previous ones.**

***Keywords*— bubble sort, complexity, worst case, algorithm, performance, comparisons.**

I. INTRODUCTION

Sorting is defined as the process of arranging a given set of elements in either ascending order or descending order. Let us consider a set ‘A’ of ‘n’ elements, A_1, A_2, A_3……….A_n in memory. Sorting of ‘A’, refers to rearranging the elements of the set in increasing order, that is, $A_1 <= A_2 <= A_3 <=$ ………….. $<= A_n$.[1] There are a variety of algorithms available for performing this kind of rearrangement. Although a large number of algorithms have been developed but no single sorting algorithm is best suited for all the applications. Sorting algorithms find their usage in a variety of applications in real life and many other computer applications. This reason has widened the scope of research in this field. Sorting algorithms can broadly be classified into internal sorting and external sorting. In case of internal sorting the elements to be rearranged are present in the main memory. Where as in case of external sorting the elements to be sorted are large in number hence they reside in external sources other than the main memory.[1]

Here in this paper I have proposed a new approach to improve the worst case performance of bubble sort by reducing the number of comparisons. The rest of the paper is organized as follows: Section II describes the bubble sort algorithm followed by section III where performance of simple bubble sort has been analyzed. Section IV would be presenting the new approach to enhance the worst case performance of bubble sort. Section V & VI describes the performance analysis of new approach and comparison of proposed algorithm with normal bubble sort and insertion sort. Finally section VII would include the conclusion followed by references.

II. NORMAL BUBBLE SORT (WORST CASE)

A. Working of Bubble sort.

Consider a list of numbers A[1], A[2], A[3],…………..A[N] in memory. Bubble sort algorithm would work as follows:

Step 1: - Compare A[1] with A[2] and rearrange them in desired fashion such that A[1] < A[2]. After this compare A[2] with A[3] and arrange them so that A[2] < A[3]. Continue in this fashion until you compare A[N-1] with A[N] and rearrange them such that A[N-1] < A[N]. [1] After step 1 A[N] would contain the largest element. It should be noted that step 1 involves (n-1) comparisons.

Step 2: - Repeat step 1with one comparison less. This means that here in this step, we will stop when A[N-2] is compared with A[N-1]. [1] After step 2 A[N-1] would hold the second largest element in the list. It should be noted that step 2 involves (n-2) comparisons.

Step 3: - Repeat step 1 with two less comparisons. This means that here in this step, we will stop after rearranging A[N-3] and A[N-2]. [1]

Follow the above given sequence until you compare A[1] with A[2].
After (n-1) steps bubble sort would produce the sorted list.

B. Pseudocode for Bubble sort.

```
Bubble Sort (A[],n)
Here A is the list of unsorted elements and n is the length of the array. After completion of the algorithm array A will be arranged in ascending order (sorted).
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 [2]

C. Worst Case Example of Bubble sort.

Consider the following list of elements in reverse order (The worst case scenario): -
78, 68, 58, 48, 38, 28, 18
Here number of elements to be sorted is '7'. If we apply normal bubble sort the algorithm will sort the elements after 6th pass i.e. (n-1) steps.

Pass 1: -

78	**68**	58	48	38	28	18
68	**78**	**58**	48	38	28	18
68	58	**78**	**48**	38	28	18
68	58	48	**78**	**38**	28	18
68	58	48	38	**78**	**28**	18
68	58	48	38	28	**78**	**18**
68	58	48	38	28	18	78

End of Pass 1, Bold elements indicates the elements to be compared. (Number of comparisons 6 i.e. (n-1))

Pass 2: -

68	**58**	48	38	28	18	78
58	**68**	**48**	38	28	18	78
58	48	**68**	**38**	28	18	78
58	48	38	**68**	**28**	18	78
58	48	38	28	**68**	**18**	78
58	48	38	28	18	68	78

End of Pass 2, (Number of comparisons 5 i.e. (n-2))
Pass 3:

58	**48**	38	28	18	68	78
48	**58**	**38**	28	18	68	78
48	38	**58**	**28**	18	68	78
48	38	28	**58**	**18**	68	78
48	38	28	18	58	68	78

End of Pass 3, (Number of comparisons 4 i.e. (n-3))
Pass 4:

48	**38**	28	18	58	68	78
38	**48**	**28**	18	58	68	78
38	28	**48**	**18**	58	68	78
38	28	18	48	58	68	78

End of Pass 4, (Number of comparisons 3 i.e. (n-4))

Pass 5:

38	**28**	18	48	58	68	78
28	**38**	**18**	48	58	68	78
28	18	38	48	58	68	78

End of Pass 5, (Number of comparisons 2 i.e. (n-5))
Pass 6:

28	**18**	38	48	58	68	78
18	28	38	48	58	68	78

End of Pass 6, (Number of comparisons 1 i.e. (n-6)).

After Pass 6th i.e. (n-1th) Bubble sort will produce the sorted list.
18, 28, 38, 48, 58, 68, 78.

III. PERFORMANCE ANALYSIS OF NORMAL BUBBLE SORT (WORST CASE)

The performance of any sorting algorithm can be measured on two parameters, first being the number of comparisons and second being number of swaps.
Keeping the worst case into consideration the performance results are as: -

A. Number of comparisons:

Total number of comparisons for worst case scenario
- ⇨ (n-1) + (n-2) + (n-3) +…………………………………………….
- ⇨ (n-1) + (n-2) +…………………………………………..+ 3 + 2 +1.
- ⇨ n*(n-1)/2
- ⇨ O(n^2).

In above example n= ‘7’.
Therefore, number of comparisons: - 7*(7-1)/2 = 21(comparisons).

B. Number of swaps:

Total number of comparisons for worst case scenario
- ⇨ (n-1) + (n-2) + (n-3) +…………………………………………….
- ⇨ (n-1) + (n-2) +…………………………………………..+ 3 + 2 +1.
- ⇨ n*(n-1)/2
- ⇨ O(n^2).

In above example n= ‘7’.
Therefore, number of comparisons: - 7*(7-1)/2 = 21(swaps).

IV. PROPOSED ALGORITHM

A. Working of proposed algorithm

The working of proposed algorithm to enhance the performance of bubble sort (in worst case) is as given below:

Step no. 1: - Begin with first element of the reversed array and compare it with the last element, if **A[front] > A[last]**, swap them. After this consider the second element and the second last element, again if **A[front+1] > A[last-1]**, swap them. Continue in this fashion until you have either a single element remaining in the list or when two consecutive middle elements have been compared.

Step no. 2: - Apply normal bubble sort algorithm on the list produced as a result from step no. 1.

B. Pseudo code for proposed algorithm.

```
improved_bubble_sort(A[0…..n-1])
front=0, last=n-1;
while(front < last)
{
if(A[front] > A[last])
{
interchange(A[front], A[last]);
}
front++;
last--;
}
Call normal_bubble_sort(A[0………..n-1]). //Call normal bubble sort.
```

C. Worst case example of improved bubble sort

Consider the following list of elements in reverse order (The worst case scenario): -
78, 68, 58, 48, 38, 28, 18

Step no. 1:

a)	**78**	68	58	48	38	28	**18**

Compare 78 with 18, since 78 > 18 so swap them.

b)	18	**68**	58	48	38	**28**	78

Compare 68 with 28, since 68 > 28 so swap them.

c)	18	28	**58**	48	**38**	68	78

Compare 58 with 38, since 58 > 38 so swap them.

d)	18	28	38	**48**	58	68	78

Since, only one element remains in the middle. So, go to step no. 2
(No of comparisons & swaps = n/2)

Step no. 2: - Apply normal bubble sort on the above generated list.

Pass 1:

18	**28**	38	48	58	68	78

Pass 2:

18	**28**	38	48	58	68	78

Pass3:

18	**28**	38	48	58	68	78

Pass4:

18	**28**	38	48	58	68	78

Pass5:

18	**28**	38	48	58	68	78

Pass6:

18	**28**	38	48	58	68	78

Bold elements indicates the elements to be compared.
(No. of comparisons = n-1 & no. of swaps = 0).

V. PERFORMANCE ANALYSIS OF PROPOSED ALGORITHM.

Performance of proposed algorithm is also analyzed keeping into consideration the worst case scenario.

A. Number of comparisons

As indicated by the above example. It has been found that the number of comparisons in step no. 1 of the algorithm is |n/2| and that in step no. 2 is (n-1). Combining both step 1 & step 2 we get:

- ⇨ n/2 + (n-1)
- ⇨ (2(n-1) + n)/2
- ⇨ (2n-1+n)/2
- ⇨ 3n-1/2
- ⇨ O(n).

B. Number of swaps

The number of swaps in step no. 1 is n/2 & that in step no. 2 is 0. Combining both we get:

- ⇨ n/2 + 0
- ⇨ n/2
- ⇨ O(n).

VI. COMPARISON OF PROPOSED ALGORITHM WITH EXISTING ALGORITHMS.

The proposed algorithm has been compared with normal bubble sort and insertion sort on different size of input elements in reverse order. Results obtained are as shown in the table given below: -

TABLE 1
Comparison of proposed algorithm with existing ones.

Input size	**Bubble sort**	**Insertion sort**	**Proposed Algorithm (Improved bubble sort)**
	No. of comparisons	**No. of comparisons**	**No. of comparisons**
15	105	105	22
55	1485	1485	82
220	24090	24090	329
350	61075	61075	524
512	130816	130816	767
405	81810	81810	607
850	360825	360825	1274
600	179700	179700	899
750	280875	280875	1124
890	395605	395605	1334
910	413595	413595	1364
1200	719400	719400	1799

VII. CONCLUSION

So, it is clear from the results that the algorithm that I have proposed here, gives better results than the normal bubble sort and insertion sort when applied on large input size taken in reverse order.

REFERENCES

[1] Data Structures by Seymour Lipschutz, Schaum's outlines, The MacGraw Hill Companies.
[2] Khairullah Md., "Enhancing worst sorting algorithms", International Journal of Advanced Science & Technology, Vol. 56, July, 2013.
[3] Levitin A., "Introduction to the Design & Analysis of Algorithms", 2nd Ed. Pearson Educational, 2007.
[4] Oyelami Olufemi Moses, "Improving the performance of bubble sort using a modified diminishing increment sorting" Covenant University, P. M. B. 1023, Ota, Ogun State, Nigeria.
[5] Ming Zhou, Hongfa Wang, "An Efficient Selection Sorting Algorithm for Two – Dimensional Arrays", Zhejiang Water Conservancy and Hydropower College Hangzhon, China.
[6] Sartaj S. (2000). Data Structures, Algorithms & Applications in Java, McGraw-Hill.
[7] E. Kapur, P. Kumar and S. Gupta, "Proposal of a two way sorting algorithm and performance comparison with existing algorithms", International Journal of Computer Science, Engineering & Applications (IJCSEA), Vol. 2, No. 3, (2012), pp. 61-78.