An Approach to Improve the Performance of Insertion Sort Algorithm

Partha Sarathi Dutta
Department of Computer Application,
Gyan Jyoti College, University of North Bengal
Dagapur, Near Pintail Village, Siliguri, West Bengal, India
E-mail: dpartha.sjc@gmail.com

Abstract— Sorting is a fundamental operation in computer science. Sorting means rearranging data in some order, such as ascending, descending with numerical data or alphabetically with character data. There are many sorting algorithm that have been proposed to meet the particular application, among which is insertion sort. It is known that insertion sort runs faster when the list is 'nearly' sorted but it runs slow when the list is in reverse order. This paper shows a way to improve the performance of insertion sort technique by implementing the algorithm using a new approach of implementation. This new approach has compared with the original version of insertion sort algorithm and bubble sort and showed that the proposed approach performed better in the worst case scenario.

Keywords- sorting; insertion sort; algorithm; new approach; bubble

I. INTRODUCTION

The sorting algorithm is used to rearrange the items of a given list in some order. Though dozens of sorting algorithm have been developed, no single sorting technique is best suited for all applications. Great research went into this category of algorithm because of its importance. These types of algorithm are very much used in many computer algorithms for example searching an element, database algorithm and many more. Sorting algorithm can be classified into two categories: internal sorting and external sorting. In internal sorting all items to be sorted are kept in the main memory. External sorting, on the other hand, deals with a large number of items, hence have to be reside in auxiliary storage device such as tape or disk.

This paper presents a new idea to improve the performance of insertion sort algorithm by reducing the number of comparison in worst case scenario.

The paper is organized as follows: Section II describes the insertion sort algorithm. Section III describes the analysis of insertion sort algorithm. Proposed algorithm is described with Example and pseudocode in section IV. Section V and Section VI describes the performance analysis and comparison of proposed algorithm respectively. Conclusion is described in section VII and finally used references has described.

II. STRAIGHT INSERTION SORT

The insertion sort algorithm works as follows: Consider an array A[0..n-1] with n- elements. This algorithm scans A from A[0] to A[n-1], inserting each element A[k] into its proper position in the previously sorted subarray A[0], A[1], …….A[k-1]. This can be accomplished by comparing A[k] with A[k-1], A[k] with A[k-2] and so on, until we found an element A[j] such that A[j]<=A[k]. Then each of the elements A[k-1],A[k-2]…..A[j+1] is moved forward one location , and A[k] is the inserted in the j+1st position in the array.

Here is a pseudocode of this algorithm:

```
Algorithm InsertionSort(A[0..n-1)
// Input: An array A[0..n-1] with n- elements
//output: An array A[0..n-1] sorted in ascending order
for(i=1;i<=n-1;i++)
{
   t=A[i];
   j=i-1;
While(j >= 0 && t < A[j])
    {
       A[j+1] = A[j];      //Move element forward
        j=j-1;
     }
A[j+1] = t ;                // Insert element in proper place
}
```

Fig 1. Pseudocode of insertion sort

III. ANALYSIS OF STRAIGHT INSERTION SORT

A. Number of comparisions

The basic operation of this algorithm is the key comparison. In this algorithm, the number of key comparisons depends on the nature of the input. The worst case occurs, when input of an array are of reverse order. The number of key comparisons for such an input is

$$C(n) = \sum_{i=1}^{n-1} \sum_{j=0}^{i-1} 1$$

$$= \sum_{i=1}^{n-1} (i - 1) - 0 + 1$$

$$= \sum_{i=1}^{n-1} i$$

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

$$= n(n-1) / 2 = O(n^2)$$

B. Number of movements

The number of key movement is , $M(n) = 1 + 2 + 3 + \ldots\ldots + (n-1) = n(n-1) / 2$. Since, the number of keys that needs to be moved in i-th iteration is i.

C. Number of swaps

The total number of swaps when the array are of reverse order, $S(n) = 0$

IV. THE PROPOSED ALGORITHM

A. Working of proposed algorithm

The proposed algorithm to sort the list of elements in an array A[0..n-1] works as follows:-

Step 1:- Start with the first element A[0] and compare it with the last element A[n-1]. If A[0] > A[n-1], then swap A[0] and A[n-1] . Next compare A[1] with A[n-2] and swap if A[1] > A[n-2]. This process is repeated until the last two consecutive middle elements are compared or until only one element remains in the middle.

Step 2:- Now, insertion sort algorithm is applied to sort the elements in the array A.

B. Example

Consider the following list of elements in the reverse order (worst case scenario)

		55	44	33	22	11
Step 1:-	(a)	55	44	33	22	11 [Compare 55 with 11. Since 55 > 11 so, swap]
	(b)	11	44	33	22	55 [Compare 44 with 22. Since 44 > 22 so, swap]
	(c)	11	22	33	44	55 [only one element in the middle, go to step 2]
Step 2:-	(a)	**11**	22	33	44	55
	(b)	11	**22**	33	44	55
	(c)	11	22	**33**	44	55
	(d)	11	22	33	**44**	55
	(e)	11	22	33	44	**55**
	(f)	11	22	33	44	55

Fig 2. Illustration of proposed algorithm. The elements being inserted is in bold.

C. Pseudocode of proposed algorithm

```
Algorithm ProposedSort(A[0..n-1])
// Input- an array A[0..n-1] with n-elements
//Output- an array A[0..n-1] , sorted in ascending order.
beg=0,end=n-1;          //initialize lower and upper bound of  A
while( beg < end)
 {
    if( A[beg] > A[end] )
        swap(A[beg], A[end])
```

```
  beg =beg +  1;                    //update lower and upper bound of A
  end =end - 1 ;
}//end of while loop
Call  InsertionSort(A[0..n-1])      // call insertion sort algorithm (given in fig(1)) to sort the elements.
```

Fig 3. Proposed algorithm

V. PERFORMANCE ANALYSIS OF PROPOSED ALGORITHM

Generally, the time for a sorting algorithm is determined in terms of the number of comparisons. In this paper, the number of comparisons of the proposed algorithm has measured when the elements are in the reverse order. Total number of data movement and swap operations taken by the proposed algorithm has also been measured.

A. Number of comparisons

In step 1 of the algorithm, the number of comparisons is |_n/2_|. In step2 of the algorithm, the number of comparisons in each iteration is 1. The total number of iteration is n-1, so the number of comparisons is = n-1, where n is the input size of the array A.

Therefore, total number of comparison to sort array A of size n by the proposed algorithm is

$$C(n) = |\_n/2\ \_| + (n-1)$$
$$= |\_(3n-2) / 2\_|$$
$$= O(n)$$

B. Number of movements

There is no data movement when the elements are in the reverse order. So, M(n) = 0

C. Number of swaps

The total number of swap, when the elements are in the reverse order is , S(n) = |_n / 2_|

VI. COMPARISONS OF PROPOSED ALGORITHM WITH EXISTING ALGORITHM

The following table shows the performance of the proposed algorithm by comparing with the existing algorithm (Insertion and Bubble) when the elements are in the reverse order.

TABLE I. COMPARISONS OF PROPOSED ALGORITHM WITH EXISTING SORT

Size of Input	Insertion Sort	Bubble Sort	Proposed Algorithm
	Number of Comparisons	*Number of Comparisons*	*Number of Comparisons*
5	10	10	6
32	496	496	47
64	2016	2016	95
128	8128	8128	191
256	32640	32640	383
512	130816	130816	767

VII. CONCLUSION

Table I shows that the proposed algorithm gives better performance in terms of number of comparisons. The result also shows that, the proposed algorithm is more efficient then both insertion sort and bubble sort for large values of input. Finally, the author has reached the conclusion that the proposed algorithm can be used for large number of input to get the better result.

REFERENCES

[1] Levitin A., “Introduction to the Design and Analysis of Algorithms,” 2nd ed, Pearson Education, 2007.

[2] Knuth D.,”The Art of Computer Programming Sorting and Searching,” 2nd ed., vol. 3, Addision Wesley, 1998.

[3] Wang Min, “Analysis on 2-Element Insertion Sort Algorithm,” International Conference on Computer Design and Applications (ICCDA), 2010.

[4] Oyelami Olufemi Moses, “Improving the Performance of Bubble Sort Using a Modified Diminishing Increment Sorting,” Scientific Research and Essay Vol.4 (8), pp. 740 -744 (2009).

[5] Vandana Sharma, Satwinder Singh and Dr.K.S.Kahlon, “Performance Study of Improved Heap Sort Algorithm and Other Sorting Algorithm on Different Platforms,” IJCSNS International Journal of Computer Science and Network Security, Vol.8 No.4 (2008)

[6] Rupesh Srivastava, Tarun Tiwari, and Sweetes Sing, “Bidirectional Expansion—Insertion Algorithm for Sorting,” Second International Conference on Emerging Trends in Engineering and Technology, ICETET-09. (2009).