Enhanced Insertion Sort Algorithm

Tarundeep Singh Sodhi
Assistant Professor in CSE dept. at ARNI University, Kathgarh Indora-H.P.
H.No^{B-5}/$_{108,}$ Ganga Singh Nagar Jandiala road,Tarn-Taran.
Pincode-143401

Surmeet Kaur
Assistant Professor in CSE dept. at Lovely Professional University, Jalandhar.
H.No-170A , 2 Partap Nagar, Opposite Kamal Palace, Sangrur.
Pincode-148001

Snehdeep Kaur
Student of MCA at Guru Nanak Dev University, Regional campus, jalandhar.
H.No^{B-5}/$_{108,}$ Ganga Singh Nagar Jandiala road,Tarn-Taran.
Pincode-143401

ABSTRACT

Sorting is integral part of many computer based systems and applications, as it involves rearranging information into either ascending or descending order. There are many sorting algorithms like Quick sort, Heap sort, Merge sort, Insertion sort, Selection sort, Bubble sort and Freezing sort. However, efforts have been made to improve the performance of the algorithm in terms of efficiency, indeed a big issue to be considered. Major Emphasis has been placed on complexity by reducing the Number of comparisons, hence reducing complexity. This paper presents new sorting algorithm **EIS**, **"ENHANCED INSERTION SORT"**.It is basically an enhancement to**INSERTION SORT** (a kind of Hybrid sorting technique) by making it impressively faster algorithm with **O(n)**complexity as compared to **O(n^2)** of **insertion sort** in worst case and less than **O(n$^{1.585}$)** in average case which is much better than insertion sort **O(n^2)**. It works flawlessly with huge lists of elements. To prove the effectiveness of the algorithm, the new algorithm is analyzed, implemented, tested and results has been carried out and compared with other major sorting algorithms and the results were promising.

General Terms

Sorting Algorithm, Hybrid technique, EIS-Enhanced Insertion sort, NOC- Number of Comparisons, NOE-Number of elements.

Keywords

Enhanced Insertion sort, EIS, NOC, NOE, Freezing Sort, complexity, selection sort, bubble sort, transition element.

1. INTRODUCTION

Algorithm is a stepwise method to solve a problem, efficiently and expressed as a finite sequence of steps. Algorithms are used for calculation, data processing, and many other fields.

Sorting has been considered as a fundamental problem in the study of algorithms, that due to many reasons:

- The need to sort information is inherent in many applications.
- Algorithms often use sorting as a key subroutine and efficient sorting is important to optimize the use of other algorithms that require sorted lists to work correctly.

The output should satisfy two major conditions:

- The output is a permutation, or reordering, of the input. \
- The output is in non decreasing order.

Many researchers considered all sorting techniques had been discovered, but many useful new sorting algorithms are recently introduced, for example, library sort was first published in 2004.

Insertion sorting algorithm is another important algorithm, used for sorting small lists. But the study shows that the EIS is more efficient, theoretically, analytically, and practically as compared to the original (insertion) sorting algorithm and also good for sorting bigger lists. Section III presents the concept of EIS algorithm and its pseudo code. Furthermore, the implementation, analysis, and comparison with insertion sort and other algorithms are highlighted.

2. ABOUT INSERTION SORT

The insertion sort, as its name suggests, inserts each item into its proper place in the final list. The simplest implementation of this requires two list structures: the source list and the list into which sorted items are inserted.

2.1 Example

45	30	60	25	70	20	80	75	15	10
30	45	60	25	70	20	80	75	15	10
30	45	60	25	70	20	80	75	15	10
25	30	45	60	70	20	80	75	15	10
25	30	45	60	70	20	80	75	15	10
20	25	30	45	60	70	80	75	15	10
20	25	30	45	60	70	80	75	15	10
20	25	30	45	60	70	75	80	15	10
15	20	25	30	45	60	70	75	80	10
10	15	20	25	30	45	60	70	75	80

- Calculating the number of green elements, we can observe that no of comparisons in all are 31.
- Red elements show the transitioned elements

3. ENHANCED INSERTION SORT

3.1 Concept

Inserting a new element at desired place in already sorted part of an and decreasing the number of comparisons of the array by one for next call. In fact, the Enhanced

Insertion Sort(EIS) algorithm is an enhancement to the IS algorithm, but the difference is in the approach as it compares with the very first element i.e.A[0] in the sorted part of array, which in fact is the smallest element in the list at instant , after comparing i th element with (i-1)th. This is called as hit method; more we get hit more the efficiency increases.

Basically sometimes we have element which gets sorted after (n-1) comparisons i.e at first place A[0] in insertion sort. So for reducing these useless comparison, why not we compare the element to be sorted with the very first element A[0] in the part of list, which is already sorted i.e. before ith element, which we know is the smallest element up till now.

Further list is divided, selecting a middle element and comparing to part on its left or right based on the condition for middle comparison and then comparing after leaving one element in that particular part, hence reducing the no of comparisons. The technique is more efficiently suitable for bigger lists and efficiency increases when the ith is less than A[0] which gives O[n] in worst case.

3.2 Procedure

The whole procedure which shows the enhancement in the insertion sort technique as described below:-

1. Instead of comparing all the elements from right to left, we just compare the ith element with (i-1)th element.
2. If ith > (i-1)th then the element is simply added/appended to the list i.e no swapping. Else
3. Compare ith element with ptr, (i.e. first element which is smallest in current sorted list) as the elements from A[0]th to A[i-1]th are already sorted.
4. If ith <ptr then insert ith element before ptr and ptr to this element. (which is now smallest and first element in the list)and swap further list accordingly. Else
5. If the element lies between A[0]th and the (i-1)th element, then we further divide the total number of sorted elements or ith by 2 i.e k = (i-1)/2 or k=i/2. We take the later one.
6. Now compare the ith element with kth element and check again.
7. If ith < kth, then compare with k-2 and so on until we find an element kth < ith and then compare with (k+1)th and swap based on conditions.
8. If ith > kth then compare with k+2 and so on until we find an element kth > ith and then compare with (k-1)th and swap ith based on condition 2.

3.3 Pseudo code

```
1.  Calculate length n
2.  var i=1,j
3.  if (a[i] < a [i-1]), then
4.  if (a[i] < a [0]), then
5.  set j = 0 and goto-35
6.  else j = i/2 and goto-10
7.  end if
8.  else i++ and repeat-3
9.  end if
10. if (a[i]<a[j]), then goto-16
11. elseif (if (a[i] = = a[j]), then
12. set j = j+1 and goto-35
13. end if
14. else goto-25
15. end if
16. while((j-1)>=0), do
17. j = j-2 and if (a[i]>a[j]), then
18. if (a[i] < a [j+1]), then
19. set j = j+1 and goto-35
20. else set j = j+2 and goto-35
21. end if
22. else if (a[i] = =a[j]), then
23. Set j = j+1 and goto-35
24. Else return
25. While (((i-1)-j) > = 0 ), do
26. J = j+2 and (a[i] < a[j]), then
27. If (a[i] < a [j-1]), then
28. Set j=j-1 and goto-35
29. Else set j+j+0 and goto-35
30. End if
31. Else if (a[i]= = a[j]), then
32. Set j=j+1 and goto-35
33. Else return
34. End if
35. Swap a[i] and a[j]
36. j++
37. while(j=i-1), do
38. i++ and goto-3
39. END
```

3.4 Analysis and Comparisons

EIS algorithm is easy to analyze as compared to IS algorithm since the loop does not requires scanning all i-1 elements (this takes i-1 comparisons) and then swapping the ith element into its appropriate position as in IS algorithm.

3.4.1. Swaps

We can very well observe that there is no change in number of swaps in this technique as compared to insertion sort as the previous list is already sorted in both the techniques and we just have to find the place for the next element. For example:
For 10 Elements
5,60,37,28,50,20,160,7,89,10

- Number of swaps in Insertion sort 22
- Number of Swaps in Enhanced insertion sort 22

3.4.2. Complexity

3.4.2.1. Best Case

In the best case, when all the elements in the array are in increasing order, then there should be no comparisons for i=1, 2, 3……..n. So we get the running time in linear order i.e O(n) which is same as that of insertion sort.

3.4.2.2. Average Case

The average case of enhanced insertion sort is also quadratic as is the case with insertion sort when we have an unsorted array, but it reduces the number of comparison as compared to insertion sort, as it is observed that the average case of insertion sort can often be as bad as worst case i.e there may be a need to compare each element A[i] with each elements in the entire sorted sub array A[1],A[2],…….A[i-1] and thus the time can be expressed as a quadratic equation i.e O(n^2), but this is never the case with any unsorted array in EIS.

So in an unsorted array, when the ith element to be sorted lies at A[2]th or A[i-2]th position we have maximum number of comparisons which is much less than the worst case of insertion sort.

For example, consider the list of 106 elements as below, where we have maximum number of comparisons when sorted with EIS:
950,50,750,250,751,249,752,248,753,247,754,246,755,245,756,244,757,243,758,242,759,241,760,240,761,239,762,238,763,237,764,236,765,235,766,234,767,233,768,232,769,231,770,230,771,229,772,228,773,227,774,226,775,225,776,224,777,223,778,222,779,221,780,220,781,219,782,218,781,217,782,216,783,215,784, 214,785, 213,786, 212,787, 211,788,210,789,209,790,208,791,207,792,206,793,794,205,795,204,796,203,797,202,798,201,799,200,800
Let us include them one by one.(Green shows the element to be compared, red represent the current element which is being added and brackets represents the number of times it is compared)

Table 1: LISTING

LIST INPUT	**NUMBER OF Elements (NOE) = 106**	**950,50,750,250,751,249,752, 248,753, 247,754, 246,755,245,756, 244,757,243,758,242,759,241,760,240,761,239,762,238,763,237, 764,236,765, 235,766, 234,767, 233,768, 232,769, 231,770, 230,771, 229,772,228,773,227,774,226,775, 225,776, 224,777, 223,778, 222, 779, 221,780,220,781,219,782, 218,781, 217,782, 216,783, 215,784, 214,785, 213,786, 212,787, 211,788, 210,789, 209,790, 208,791,207, 792,206,793,794,205,795,204,796,203,797,202,798,201,799,200,800**

VALUE OF (i)	NOE (Sorted List)	ETI Element To INSERT	LIST AFTER SORTING	NOC Number Of Comparisons	(NMPC) Number of Possible Comparisons
i = 1	0	50	50,950	1	1
i =2	2	750	50,750,950	2	3
i =3	3	250	50,250,750,950	3	4
i =4	4	751	50,250,750,751,950	3	4
i =5	5	249	50,249,250,750,751,950	4	5
i =6	6	752	50,249,250,750,751,752,950	4	5
i =7	7	248	50,248,249,250,750,751,752,950	4	5
i =8	8	753	50,248,249,250,750,751,752,753,950	4	5
i =9	9	247	50,247,248,249,250,750,751,752,753,950	5	6
i =10	10	754	50,247,248,249,250,750,751,752,753,754,950	5	6
i =11	11	246	50,246,247,248,249,250,750,751,752,753,754,950	5	6
i =12	12	755	50,246,247,248,249,250,750,751,752,753,754,755,950	5	6
i =13	13	245	50,245,246,247,248,249,250,750,751,752,753,754,755, 950	6	7
i =14	14	756	50,245,246,247,248,249,250,750,751,752,753,754,755, 756,950	6	7
i =15	15	244	50,244,245,246,247,248,249,250,750,751,752,753,754, 755,756,950	6	7
i =16	16	757	50,244,245,246,247,248,249,250,750,751,752,753,754, 755,756,757,950	6	7
i =17	17	243	50,243,244,245,246,247,248,249,250,750,751,752,753, 754,755,756,757,950	7	8
i =18	18	758	50,243,244,245,246,247,248,249,250,750,751,752,753, 754,755,756,757,758,950	7	8
i =19	19	242	50,242,243,244,245,246,247,248,249,250,750,751,752, 753,754,755,756,757,758,950	7	8
i =20	20	759	50,242,243,244,245,246,247,248,249,250,750,751,752, 753,754,755,756,757,758,759,950	7	8
i =21	21	241	50,241,242,243,244,245,246,247,248,249,250,750,751,	8	9

			752,753,754,755,756,757,758,759,950		
i =22	22	760	50,241,242,243,244,245,246,247,248,249,250,750,751, 752,753,754,755,756,757,758,759,760,950	8	9
i =23	23	240	50,240,241,242,243,244,245,246,247,248,249,250,750, 751,752,753,754,755,756,757,758,759,760,950	8	9
i =24	24	761	50,240,241,242,243,243,245,246,247,249,250,750,751, 752,753,754,755,756,757,758,759,760,761,950	8	9

And so on….So to arrange any nth element in the list, number of comparisons required are $\lceil n/4 \rceil$ +12.
So as we can notice, after n=4 we have 4 times the same number of comparisons starting from 5,6,7……so on. So talking about n=48, no of comparisons will be
15+15+15+15+14+14+14+14+13+13+13……….=449

The number of comparisons as calculated are always less than $n^{1.585}$. So the complexity in the worst scenario of average case will be $O(n^{1.585})$ for constant 1.
The complexity may vary from O(n) to $O(n^{1.585})$
For example….for n=48,the number of comparisons calculated manually are 449 which is less than $48^{1.585}$ i.e 462.

3.4.2.3 Worst Case

In the worst case of Insertion Sort , when the array is in decreasing order, one must compare each element A[i] with each elements in the entire sorted sub array A[1],A[2],…….A[i-1] and thus the time can be expressed as a quadratic equation i.e $O(n^2)$ But in case of enhanced insertion technique, it is O(n) as we have just 1 comparisons for the first two elements, rest n-2 have just 2 comparisons. For example:

For the average case like this:-

100	80	70	50	40	20
80	100	70	50	40	20
70	80	100	50	40	20
50	70	80	100	40	20
40	50	70	80	100	20
20	40	50	70	80	100

We get 1+2*4=9comparisons
In general 1+2(n-2) comparisons=O (n)

3.4.3.Example

Consider the same example as below

45	30	60	25	70	20	80	75	15	10
30	45	60	25	70	20	80	75	15	10
30	45	60	25	70	20	80	75	15	10
25	30	45	60	70	20	80	75	15	10
25	30	45	60	70	20	80	75	15	10
20	25	30	45	60	70	80	75	15	10
20	25	30	45	60	70	80	75	15	10
20	25	30	45	60	70	75	80	15	10
15	20	25	30	45	60	70	75	80	10
10	15	20	25	30	45	60	70	75	80

Here as compared to insertion sort there are only 16

So there is always reduction in no of comparisons.

3.4.4 Pseudo code

Insertion sort works by removing an element from the input data for every repetition of insertion sort, inserting it into the correct position in the already sorted list, until no input elements remain. The insertion sort has a complexity of $O(n^2)$. In simple pseudo code, insertion sort algorithm might be expressed as:

```
1. for j ←1 to length (A)-1
2. key ← A [ j ]
3. > A[j] is added in the sorted sequence A [1... j-1]
4. i ← j - 1
5. while i >= 0 and A [i] > key
6. A [ i +1 ] ← A[ i ]
7. i ← i -1
8. A [i +1] ← key
```

3.4.5 Stability

As insertion sort is a stable algorithm, enhanced insertion sort is also stable as a sorting algorithm is stable if whenever there are two records R and S with the same key and with R appearing before S in the original list, R will appear before S in the sorted list.

Table showing comparisons:

Table2: Comparison with recently used algorithm

	Enhanced Insertion Sort	**Bubble Sort**	**Selection Sort**	**Insertion Sort**
Best Case Complexity	O(n)	$O(n^2)$	$O(n^2)$	O(n)
Average Case Complexity	Less than $O(n^{1.585})$	$O(n^2)$	$O(n^2)$	$O(n^2)$
Worst Case	Less than $O(n^{1.585})$	$O(n^2)$	$O(n^2)$	$O(n^2)$

Table3:Comparison with various enhanced algorithms

	Enhanced Insertion Sort	**Enhanced Selection Sort[3]**	**SMS Algorithm[8]**	**Enhanced Bubble Sort[3]**
Best Case Complexity	O(n)	$O(n^2)$	O(nlgn)	O(nlgn)
Average Case Complexity	Less than $O(n^{1.585})$	$O(n^2)$	O(nlgn)	O(nlgn)
Worst Case	Less than $O(n^{1.585})$	$O(n^2)$	O(nlgn)	O(nlgn)

4. CONCLUSION

This work focuses to provide an enhancement in insertion sort and making enhanced insertion sort more efficient for bigger list as it gives less than $O(n^{1.585})$ complexity in worst case and reduces near about half comparison. It does not requires scanning all elements, because of its hit method it provides a boost to sorting, also reduces the number of comparisons while sorting an array as compared to $O(n^2)$ complexity of insertion sort, in fact it is O(n) in best as well as sometimes in average case. Furthermore the proposed algorithm is compared with some recent used algorithms like bubble sort, selection sort etc. Basically its complexity varies from O(n) to $O(n^{1.585})$.

5. ACKNOWLEDGEMENT

I would like to express my gratitude to all those who gave me the possibility to complete this work. I would like to thank my GOD for giving me strength to complete this task and to my mentor, Mr. Parveen Kumar for providing support and material related to the area of this research.

6. REFERENCES

[1] Cormen T, Leiserson C, Rivest R and Stein C. 2001. Introduction to Algorithms, Tata Mc Graw Hill.

[2] Surmeet Kaur, Tarundeep Singh Sodhi and Parveen Kumar, May 2012. Freezing Sort, International Journal of Applied Information Systems VOL2.

[3] Jehad Alnihoud and Rami Mansi, 2010. An Enhancement of Major Sorting Algorithms, The International Journal of Information Technology VOL7.

[4] Rupesh Srivastava, Tarun Tiwari Sweetesh Singh, 2009. Bidirectional Expansion–Insertion Algorithm for Sorting. Second International Conference on Emerging Trend in Engineering and Technology, ICETET-09.

[5] Muhammad Anjum Qureshi 2010. Qureshi Sort: A new sorting Algorithm.

[6] Seymour Lipschutz, 2011. Data Structures with C, Shaum Series.

[7] Wang Min, 2010. Analysis on 2-Element Insertion Sort Algorithm, International Conference on Computer Design And Appliations (ICCDA).

[8] Rami Mansi,2010. Enhanced Quick Sort Algorithm International Arab Journal of Information Technology.

[9] Basit Shahzad and Muhammad Tanvir Afzal, 2007.Enhanced Shell Sort Algorithm, World Academy of Science,Engineering and Technology.

[10] Sultanullah Jadoon, Salman Faiz Solehria, MubashirQayum. Optimized Selection Sort Algorithm is faster than Insertion Sort Algorithm: a Comparative Study, International Journal of Electrical and Computer Sciences.