Increasing Time Efficiency of Insertion Sort for the Worst Case Scenario

Surabhi Patel
Department of Information Technology,
Christ University Faculty of Engineering

Moirangthem Dennis Singh
Department of Computer Science and Engineering,
Christ University Faculty of Engineering

Chethan Sharma
Department of Computer Science and Engineering,
Christ University Faculty of Engineering

ABSTRACT
Insertion sort gives a time complexity of O(n) for the best case. In the worst case where the input is in the descending order fashion, the time complexity is $O(n^2)$. In the case of arrays, shifting takes $O(n^2)$ while in the case of linked lists comparison comes to $O(n^2)$. Here a new way of sorting for the worst case problem is proposed by using arrays as data structure and taking more space. 2n spaces is taken where n is the number of elements and starts the insertion from $(n-1)^{th}$ location of the array. In this proposed technique the time complexity is O(nlogn) as compared to $O(n^2)$ in the worst case.

Keywords
Insertion Sort, Time Complexity, Space Complexity.

1. INTRODUCTION
Insertion sort is a comparison sort algorithm [1]in which the sorted array is built one entry at a time. The efficiency of this sorting technique is comparatively very less on large set of elements than more advanced algorithms such as heapsort, quicksort, or merge sort. In each iteration of insertion sort, an element is removed from the input data and it is inserted into the right position in the already-sorted list. This is continued until no input elements are remaining. The best case scenario is when the input is an already sorted array. In this case insertion sort the time complexity is O(n) which is linear. During each repetition, in the sorted subsection of the array, the right-most element is compared with the remaining element of the input. The worst case scenario is when the input is an already sorted array but in reverse order. In this case, before inserting the next elementeach iteration of the inner loop will search and shift the subsection of the array which is already sorted. For this case insertion sort has a quadratic running time which is $O(n^2)$ [2].

The average case also has a quadratic running time of$O(n^2)$.

2. LITERATURE
In an insertion sort algorithm, there are always two constraints in time complexity [3]. One is shifting the elements and the other one is comparison of the elements. The time complexity is also dependent on the data structure [4] which is used while sorting. If array is used as the data structure then shifting takes $O(n^2)$ in the worst case. While using linked list data structure, searching takes more time, viz. $O(n^2)$.
Take the following examples:

Sort 50, 40, 30, 20, 10 using arrays.

0	1	2	3	4
50				

Shifting = 0, Comparison = 0

0	1	2	3	4
50	40			
40	50			

Shifting = 1, Comparison = log1

0	1	2	3	4
40	50	30		
40	30	50		
30	40	50		

Shifting = 2, Comparison = log2

0	1	2	3	4
30	40	50	20	
30	40	20	50	
30	20	40	50	
20	30	40	50	

Shifting = 3, Comparison = log3

0	1	2	3	4
20	30	40	50	10
20	30	40	10	50
20	30	10	40	50
20	10	40	40	50
10	20	30	40	50

Shifting = 4, Comparison = log4

Time Complexity in Shifting: $O(n^2)$

Time Complexity in Comparison: O(nlogn)

Total time complexity: $O(n^2)$

Here as the array is sorted, binary search can be used for comparison which will lead to a time complexity of O(nlogn) but shifting takes $O(n^2)$. Therefore the total time complexity becomes $O(n^2)$.

To solve the problem of shifting, linked list can be used as illustrated in the following example.

Now, sort 50, 40, 30, 20, 10 using linked list. In a linked list shifting takes O(1) as new elements can be inserted at their right positions without shifting.

50	

Comparison = 0

50	→	40	
40	→	50	

Comparison = 1

40	→	50	→	30	
30	→	40	→	50	

Comparison = 2

30	→	40	→	50	→	20	
20	→	30	→	40	→	50	

Comparison = 3

20	→	30	→	40	→	50	→	10	
10	→	20	→	30	→	40	→	50	

Comparison = 4

Time Complexity in Shifting: O(1)
Time Complexity in Comparison: $O(n^2)$
Total Time Complexity: $O(n^2)$
Here as binary search cannot be used for comparison which will lead to a time complexity $O(n^2)$ even though shifting takes a constant amount of time.
As observed in the examples illustrated above, in both the cases the Time complexity is not getting reduced. Hence an improvised insertion sort taking additional space to sort the elements is proposed in this paper. As space complexity is less important than time complexity [5][6], this paper concentrates more on the time taken instead of space.

3. PROPOSED WORK

In the insertion sort technique proposed here, 2n spaces is taken in an array data structure, where n is the total number of elements. The insertion of elements will start from $(n-1)^{th}$ position of the array. The same procedure of a standard insertion sort is followed in this technique. Finding the suitable positions of the elements to be inserted will be done using binary search. In the following cases the details of this technique has been discussed.

3.1 Case 1

For comparing with the best case scenario of a standard Insertion Sort, the following input elements are sorted using proposed technique.

e.g. 10, 20, 30, 40, 50

0	1	2	3	4	5	6	7	8	9
				10					

Shifting =0, Comparison = 0

0	1	2	3	4	5	6	7	8	9
				10	20				

Shifting =0, Comparison = 1

0	1	2	3	4	5	6	7	8	9
				10	20	30			

Shifting =0, Comparison = 1

0	1	2	3	4	5	6	7	8	9
				10	20	30	40		

Shifting =0, Comparison = 1

0	1	2	3	4	5	6	7	8	9
				10	20	30	40	50	

Shifting =0, Comparison = 1

Total Shifting =0, Total Comparison = n-1

Therefore time complexity is O(1)+O(n) = O(n)

3.2 Case 2

For comparing with the worst case scenario of a standard Insertion Sort, the following input elements are sorted using proposed technique.

e.g. 50, 40, 30, 20, 10

0	1	2	3	4	5	6	7	8	9
				50					

Shifting =0, Comparison = 0

0	1	2	3	4	5	6	7	8	9
				50	40				
			40	50					

Shifting =1, Comparison = log1

0	1	2	3	4	5	6	7	8	9
			40	50	30				
		30	40	50					

Shifting =1, Comparison = log2

0	1	2	3	4	5	6	7	8	9
		30	40	50	20				
	20	30	40	50					

Shifting =1, Comparison = log3

0	1	2	3	4	5	6	7	8	9
	20	30	40	50	10				
10	20	30	40	50					

Shifting =1, Comparison = log4

Total Shifting =n-1,

Total Comparison =log(1*2*3*4)

=log((n-1)!)

=log((n-1) (n-1))

=(n-1)log(n-1)

=nlog(n-1) - log(n-1)

Therefore time complexity is O(n)+O(nlogn) = O(nlogn)

3.3 Case 3

For the average case scenario in a standard Insertion Sort, the input elements are in random order. Although the same procedure is followed in the proposed technique, comparison is done via binary search algorithm. Hence it takes O(nlogn) for comparison. For shifting the elements, the time taken tends to $O(n^2)$ but is not equal to $O(n^2)$. As there are more spaces, possibilities are there that shifting of some elements may be reduced because elements may be inserted both at the end as well as in the beginning

4. RESULTS

The time complexity of the proposed sorting technique and the standard Insertion sort is compared in Table 1.

Table 1. Comparison of time complexities

Input Elements	Standard Insertion Sort	Proposed Sorting Technique
Best Case (Ascending Order)	O(n)	O(n)
Worst Case (Descending Order)	$O(n^2)$	O(nlogn)
Average Case (Random Order)	$O(n^2)$	Tends to $O(n^2)$

The graphical representation of the comparison between the proposed technique and the standard insertion sort for the worst case scenario is shown in Fig 1. The graph shows the time complexity of both the algorithms for an input ranging from 1 to 10 in number.

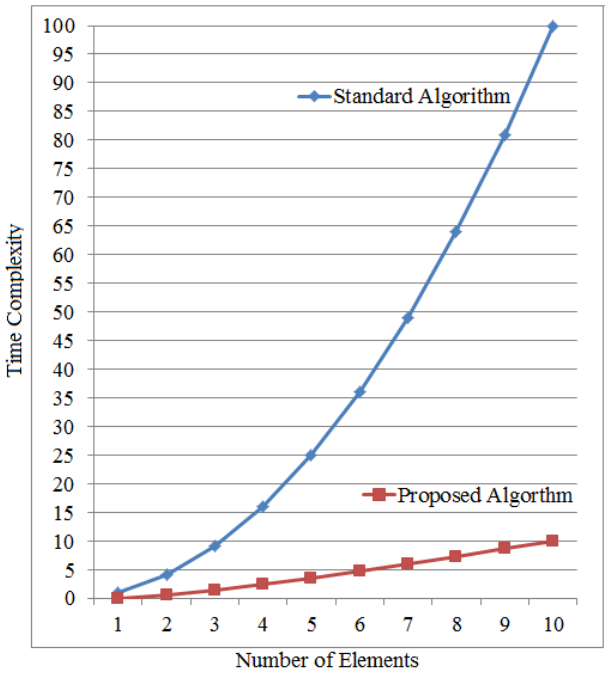

Fig 1: Comparison of proposed and standard technique

5. CONCLUSION

Here, the time complexity of worst case scenario in Insertion sort algorithm is decreased by increasing the space complexity. Future scope of work includes decreasing time complexity of the average case which is$O(n^2)$ currently. There are promising results shown in the average case scenario where the time complexity may be reduce from $O(n^2)$,if the probability of the input elements is a combination of increasing and decreasing order.

6. ACKNOWLEDGEMENT

We would like to thank Prof Anirban Roy, Department of Basic Sciences, Christ University Faculty of Engineering, Bangalore for helpful discussions and support.

7. REFERENCES

[1] Insertion Sort,http://www.princeton.edu/~achaney/tmve/wiki100k/docs/Insertion_sort.html

[2] Wang Min, "Analysis on 2-Element Insertion Sort Algorithm", 2010 International Conference On Computer Design And Applications, IEEE Conference Publications, Pages 143-146 , 2010

[3] Thomas H.Cormen, Charles.E.Leiserson, Ronald L.Rivest, and Clifford Stein, Introduction to Algorithms:Printice-Hall, Inc., 2009

[4] Mark Allen Weiss, Data Structures and Algorithm Analysis in C++: Pearson Addison-Wesley, 2006

[5] Michael A. Bender, "Insertion Sort is O(nlogn)," Third International Conference on Fun With Algorithms(FUN), Pages 16-23, 2004

[6] H. W. Thimbleby, "Using Sentinels in Insert Sort," SoftwarePractice and Experience, Volume 19(3), Pages 303–307,1989.