A New Approach of Sorting Using Recursive Partition

Mr. Maulik K. Patel [1], Ms. Shruti Yagnik [2]

***Abstract*— To address how to make basic insertion algorithm efficient, we present a research paper. In this paper, we describe one approach for improving performance of insertion sort algorithms. The goal of this research is to perform an extensive empirical analysis of insertion sort to reduce time complexity and compares them on the basis of various parameters to reach out conclusion. To prove the effectiveness of the algorithm, the new sorting algorithm is analyzed, implemented and tested. Significant improvement over insertion sort is achieved for various cases.**

***Index Terms*—Time complexity, Space Complexity, Best case, Average case, Worst case, Tick**

I. INTRODUCTION

Basic insertion sort is used for sorting small dataset. We inherited a basic insertion sort algorithm for sorting sequences and partition technique for reducing processing time. Our analysis shows that Adaptive insertion sort is better than simple insertion sort. In Adaptive Insertion sort comparison of element done both side of pivot element that reduce swap operation. Adaptive insertion sort execute sorting in recursive fashion. It use divide and conquer technique having complexity nlogn for all cases. In Section III concept of Adaptive insertion sort and pseudo code is described. We also performed analysis and comparison with other sorting algorithm.

II. INSERTION SORT

Insertion sort is widely used for sorting small number of data set. This is one of the most common algorithms that is used by many sorting algorithm to sort their sub data set. For example: Shell sort use insertion sort [6]. Insertion sort is a simple sorting algorithm that builds the final sorted array by picking one item at a time. But average case and worst case time complexity is $O(n^2)$ [1].

III. ADAPTIVE INSERTION SORT

A. *Concept of Adaptive Insertion Sort*

It is hard to insert a new element at desired place in already descending ordered element for insertion sort. Adaptive insertion sorting algorithm is enhancement of insertion sort. It is based on Divide-and-Conquer paradigm. In this paradigm complexity of sorting a set is reduced to the problem of sorting smaller sets. The three basic main steps of divide and conquer strategy for sorting a typical sub array A[s….e] is as follows:

1) Divide: The array A[s….e] is partitioned (rearranged) into two (possibly empty) sub arrays A[s….p-1] and A[p+1....e].These sub array generated by inserting elements such that each element of A[s….p-1] is less than or equal to A[p],which is, in turn ,less than and equal to each element of A[p+1….e]. The index of p is adjusted according to partition procedure.

2) Conquer: The two sub arrays A[s….p-1] and A [p+1….e] are sorted adaptive insertion sort in recursive calls to procedure using single buffer.

3) Combine: After completion of conquer step, the sub arrays are already sorted. To combine them no procedure is needed, the entire array A[s....e] is now sorted.

The algorithm is divided into two procedures. One procedure called adaptive insertion sort, which executes other procedure which perform sorting and also called itself to partition the entire list. Here s is starting index and e is ending index in respective array.

B. *Algorithm*

Input: An unsorted array A[] of size n
Output: A sorted array A[] of size n
Adaptiveinsertionsort(A,s,e)

```
1.  If s<e
2.      p=(s+e)/2
3.      buffer=a[p]
4.      i=p-1
5.      j=p+1
6.      while(a[i]<a[p])
7.              i=i-1
8.      while(a[j]>a[p])
9.              j=j+1
10.     exchange A[i] with A[j]
11.     while(j<=e)
12.             if A[j]<buffer
13.               Exchange A[j] with A[p]
14.               p=p+1
15.             j=j+1
16.     while(i>=s)
17.             if A[i]>buffer
18.               Exchange A[i] with A[p]
19.               p=p-1
20.             i=i-1
21.     If( a[p]>=buffer)
```


22. p=p-1
23. Adaptiveinsertionsort(A,s,p)
24. Adaptiveinsertionsort(A,p+1,e)

C. Analysis:

Time Complexity of Adaptive insertion Sort

1) Best case analysis::

The best case of adaptive insertion sort occurs when the pivot divides the array into two exactly equal parts, in every step and no of swap operations are also nothing or minimal, in every step. In this case adaptive insertion sort run faster.

Thus we have after division $k = n/2$ and $n - k = n/2$ for the original array of size n.

Consider, the recurrence:

$T(n) = 2T(n/2) + \alpha n$

$= 2(2T(n/4) + \alpha n/2) + \alpha n$

(Here $T(n/2) = 2T(n/4) + \alpha n/2$ by substituting n/2 for n)

$= 2^2T(n/4) + 2\alpha n$

$=2^2(2T(n/8) + \alpha n/4) + 2\alpha n$

$=2^3T(n/8) + 3\alpha n$

$=2^kT(n/2^k) + k\alpha n$ (Continuing likewise till the k^{th} step)

This recurrence will continue only until $n = 2^k$(otherwise we have $n/2^k < 1$) or until $k = \log n$. Thus, by putting $k = \log n$, we have the following equation:

$T(n) = nT(1) + \alpha n \log n$, which is $O(n \log n)$.

This is the best case complexity for adaptive insertion sort.

2) Worst case analysis:

The worst case of adaptive insertion sort occurs when the pivot we pick happens to divide the array into two exactly equal parts, in every step and no of swap operations are n/2for every step. In this case adaptive insertion sort run slower because swap operation at each step increase time complexity. Memory write operation take more time than memory read operation. In our adaptive insertion sort pivot is taken at middle and division is generated from middle.

Thus we have after division $k = n/2$ and $n - k = n/2$ for the original array of size n.

Consider the recurrence equation:

$T(n) = 2T(n/2) + \alpha n$

$= 2(2T(n/4) + \alpha n/2) + \alpha n$

(Here $T(n/2) = 2T(n/4) + \alpha n/2$ by just substituting n/2 for n)

$= 2^2T(n/4) + 2\alpha n$

$=2^2(2T(n/8) + \alpha n/4) + 2\alpha n$

$=2^3T(n/8) + 3\alpha n$

$=2^kT(n/2^k) + k\alpha n$ (Continuing likewise till the k^{th} step)

This recurrence will continue only until $n = 2^k$

Thus, by putting $k = \log n$, we have the following equation:

$T(n) = nT(1) + \alpha n \log n$, which is $O(n \log n)$.

3) Average Case Analysis:

When we run adaptive insertion sort on random input array, the partition is highly unlikely to happen in the same way at every level. We expect some of the partitions will be reasonably well balanced and some will be fairly unbalanced.

At root of the tree, the cost is n/2 for partitioning because we have pivot at middle of array. Sub arrays produced having size n/2 and n/2.Now sub array that is produced is combination of good split and bad split. Some sub array will take more time for swapping and some will take less. Thus the running time of adaptive insertion sort when level alternate between good and bad splits, is like the running time for good split alone still O(n log n).

Space Complexity of Adaptive insertion Sort

Worst case auxiliary space complexity is O(n).

D. Environment Setup

1) Microsoft Visual Studio:

Microsoft Visual Studio is used for experiment analysis of sorting algorithm. It is a complete set of development tools for building ASP.NET Web applications, desktop applications. To get better understanding the actual performance of proposed algorithm is conducted on Microsoft Visual Studio2005 .c# is used as programming language.

2) Hardware Configuration

OS: Windows 7 Ultimate
Processor: Intel Core(TM) 2 Duo 1.80 GHz
RAM: 2 GB
System Type: 32 bit Operating System

3) Performance Factor for Time Complexity

Ticks
Namespace: System. Diagnostics (Microsoft Visual Studio)
Assembly: System (in System.dll)

This property represents the number of elapsed ticks in the underlying timer mechanism. A tick is the smallest unit of time that the Stopwatch timer can measure. Use the Frequency field to convert the Elapsed Ticks value into a number of seconds.

Elapsed Milliseconds
Namespace: System. Diagnostics (Microsoft Visual Studio)
Assembly: System (in System.dll)

E. Experiment

TABLE I: ANALYSIS OF PROPOSED ALGORITHM FOR SWAP OPERATION

Insertion sort	**No of Swaps**	**Adaptive Insertion**	**No of Swaps**

		sort	
10 9 8 7 6 5 4 3	28	10 9 8 7 6 5 4 3	10
1 2 3 4 10 9 8 7	6	1 2 3 4 10 9 8 7	4
10 9 8 7 1 2 3 4	22	10 9 8 7 1 2 3 4	4

Above swap operation calculation for each algorithm depict that insertion sort takes more no of swap operation for N that is N(N-1)/2 than adaptive insertion sort ,where N is in descending order. For other two cases adaptive insertion sort has less no of swap operations.

Comparison of proposed algorithm with other sorting algorithm

In order to verify the efficiency of proposed algorithm we do some experiments. We use array to store original record .Data record are taken in descending and ascending order for analysis. We pick 100, 500, 1000, 5000,10000,20000,40000 and 50000 elements to carry out comparison experiments. In order to measure CPU time, ticks and elapsed time is counted for each sorting algorithm.

TABLE II: CPU TIME TAKEN BY THE FOUR ALGORITHMS TO SORT ELEMENTS

Number of Elements in Descending order	Insertion Sort		Adaptive Insertion Sort		Merge Sort		Heap Sort	
	Ticks	Elapsed Time	Ticks	Elapsed Time	Ticks	Elapsed Time	Ticks	Elapsed Time
500	624	00:03	44	00:00	89	00:00	68	00:00
1000	2521	00:14	91	00:00	195	00:01	150	00:00
5000	61976	00.333	490	00:02	1111	00:06	960	00:05
10000	172559	01.207	1074	00:06	2413	00.13	2147	00.12
20000	5360 58	03.012	2600	00:14	5350	00.30	4522	00.26
40000	1675949	9.123	4828	00.27	11465	00.65	9756	00.59
50000	2538109	19.256	6020	00.34	13679	00.83	12653	00.71

In tables ticks are calculated per millisecond and elapsed time represented in second: millisecond (00:00) format. In Table II proposed algorithms are compared with basic insertion sort, merge sort and heap sort.

TABLE III: CPU TIME TAKEN BY THE FOUR ALGORITHMS TO SORT RANDOM ELEMENTS

Random Number of Elements	Insertion Sort		Adaptive Insertion Sort		Merge Sort		Heap Sort	
	Ticks	Elapsed Time	Ticks	Elapsed Time	Ticks	Elapsed Time	Ticks	Elapsed Time
500	313	00:01	66	00:00	104	00:00	76	00:00
1000	1595	00.08	156	00:00	312	00.01	167	00:00
5000	31691	00.173	847	00.05	1286	00.07	1022	00.06
10000	1199902	00.657	1805	00.11	3157	00.20	2228	00.13
20000	3188821	01.130	3910	00.22	6393	00.36	5065	00.28
40000	8778 40	06.02	8299	00.47	13616	00.77	10697	00.59
50000	13156 59	08.620	11019	00.60	19161	00.99	13392	00.79

For getting better result we have also taken array element randomly using random function and compared proposed

algorithm with other sorting algorithms. To get actual behavior of algorithm we computed more than five times elapsed time in every sort algorithm over random data set.

F. Result Analysis

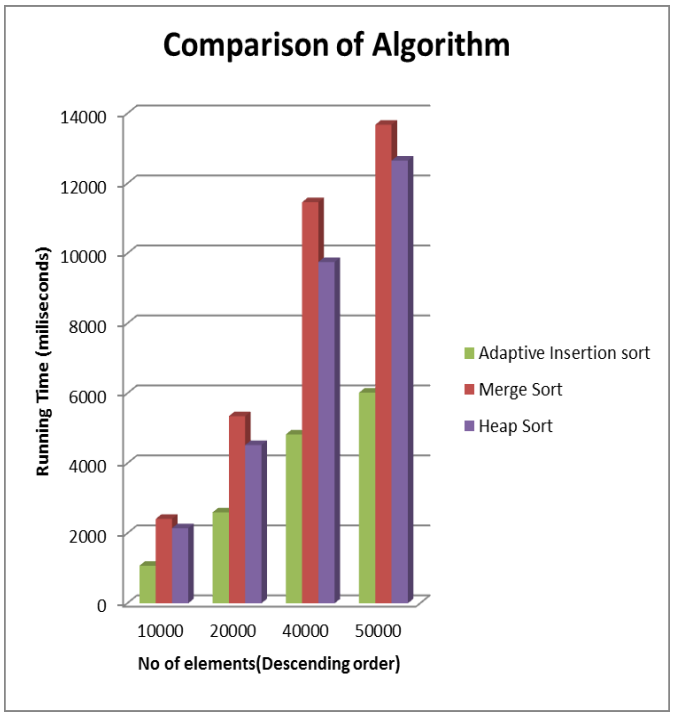

Fig. 1 A graph comparing all three algorithms for descending order elements

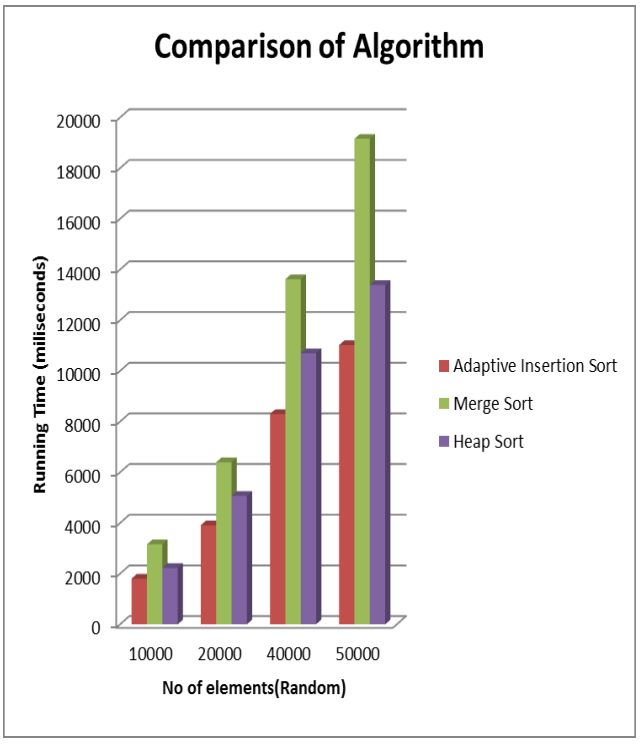

Fig. 2 A graph comparing all three algorithms for random elements

In Fig. 1 and Fig. 2 x axis represent array size of random elements and y axis is the execution time.when we have random no of elements than prformance difference started.From graph it can be said that Adaptive insertion sort takes less time as comapere to other merge sort and heap sort.

IV. CONCLUSION

Basic sorting algorithm can be adapted in variety of ways. Selection of proper technique for sorting given elements depends upon time complexity and space availability. Our proposed adaptive insertion sort is easy to understand and easy to implement. By partitioning from middle, we can reduce number of comparisons and actual running time of insertion sort in optimal way. It does not require scanning all elements, because of partition method. In theoretically Average case and worst case running time is reduced from O (n^2) to O (nlogn).By analyzing graph, it can be easily examined that Adaptive insertion sort is better option for sorting when we have to deal with random input elements. Adaptive insertion sort takes less time for sorting large number of data items as compare to other sorting algorithm like merge sort and heap sort.

ACKNOWLEDGMENT

I would like to take this opportunity to express my gratitude towards all those people who have, in various ways, helped in the successful completion of this research paper. This work is the result of the inspiration, support, guidance, co-operation and facilities that were provided to me by persons at various levels and i am obliged by all of them.

REFERENCES

[1] Thomas H .Cormen, Charles E.leiserson, Ronald L.Rivest, Clifford Stein, "Introduction to Algorithm", 3rd Edition.
[2] A.A. Puntambekar, "Analysis And Design Of Algorithms".
[3] Insertion Sort,http://en.wikipedia.org/wiki/Insertion_sort.
[4] Ms. Nidhi Chhajed, Mr. Imran Uddin, Mr. Simarjeet Singh Bhatia, "A Comparison Based Analysis of Four Different Types of Sorting Algorithms in Data Structures with Their Performances", International Journal of Advanced Research in Computer Science and Software Engineering, ISSN: 2277 128X ,Volume 3, Issue 2 ,2013.
[5] Michael A. Bender,MartinFarach-Colton,MiguelA.Mosteriro, "Insertion Sort is o(nlogn)",Theory of Computing Systems, Volume 39, Issue 3,pp 391-397.
[6] D.L.Shell, "A high-speed Sorting procedure", Magazine, Communications of the ACM,Volume 2 Issue 7,July 1959,pp.30-32.
[7] Tarundeep Singh Sodhi, SurmeetKaur,Snehdeepkaur,"Enhanced insertion Sort Algorithm", International Journal of Computer Applications ISSN:0975 – 8887, Volume 64– No.21, February 2013.
[8] ParthaSarathiDutta, "An Approach to Improve the Performance of Insertion Sort Algorithm", International Journal of Computer Science & Engineering Technology, ISSN : 2229-3345 Vol. 4 No. 05 May 2013.
[9] Wang Min, "Analysis on 2-Element Insertion Sort Algorithm", IEEE, pp. V1 143- V1 146,2010.
[10] R.Srinivas, A.RagaDeepthi, "Novel Sorting Algorithm", International Journal on Computer Science and Engineering, ISSN: 0975-3397 Vol. 5 No. 01 Jan 2013.
[11] Muhammad AnjumQureshi,"Qureshi Sort: A new Sorting Algorithm",AERO (PVT) LTD.IEEE,2009 .

Maulik Patel received B.E Degree in Computer Engineering from Kalol Institute of Technology and Research centre. He is pursuing M.E in Computer Engineering from L.J Institute of Engineering and Technology, Gujarat. His research area includes Data Structure and Cryptography.

Shruti B. Yagnik completed Bachelors in Information Technology from L.J Institute of Engineering and Technology from Gujarat University and Masters in Computer Engineering specialization in IT Systems and Network Security from Gujarat Technological University. She is currently working as Assistant Professor at L.J Institute of Engineering and Technology carrying out research in Cyber forensics and Network Security and Artificial Intelligence.

