IJITKM 0973-4414 Volume-9, Number-1 Jun-Dec 2015 pp. 103-106 Impact Factor 2.8 available online at www.csjournalss.com

A Novel Technique to Improve Sorting Procedure

Karamjeet Kaur
University College, Kurukshetra UniversityKurushtra, India 136119

Abstract: Most of the present day applications are database dependent. Some of the examples are telephone directory, list of students, list of employees in an organization, database in doctor's clinic etc. The users of these applications want the data in some specified order that may be numerical or lexicographical. Sorting algorithms in computer science sort the data in some order (increasing or decreasing). These algorithms sort the data stored in database based on its primary key or any other key value. The order may be numerical or lexicographical. This paper presents a new sorting algorithm that is easy to implement and well efficient. Here, a new algorithm is proposed which is analyzed, implemented and , then, compared with bubble sort, insertion sort and the outcome is positive.
Index Terms: Sorting, Average Case, Swap, Comparison, Stability, Complexity.

1. INTRODUCTION

Information growth rapidly in our world and to search for this information, it should be ordered in some sensible order[9]. Information is stored in form of data . Symmetrical form of data is always better than asymmetrical one. We always want to arrange information in some order so that manipulation and searching of this information is easy and efficient. But what to do if the data is not in any specified order. One answer is sorting the data. Sorting is very common problems handled by computers. Computers performs sorting(arranging data in a specific order) of data by using sorting algorithms.. Since computers can compare a large number of items quickly, they are quite good at sorting.

There are many sorting algorithms available that puts the input data in a specified order. Most used orders are numerical order and lexicographical order. Efficient sorting is important for optimizing the use of other algorithms (such as search and merge algorithms) that require sorted lists to work correctly. It is also often useful for canonicalizing data and for producing human-readable output. More formally, the output must satisfy two conditions[9]:
The output is in no decreasing order (each element is no smaller than the previous element according to the desired total order);
The output is a permutation, or reordering, of the input.
since the dawn of computing, the sorting problem has attracted a great deal of research, perhaps due to the complexity of solving it efficiently despite its simple, familiar statement. For example, bubble sort was analyzed as early as 1956[2].
Sorting algorithms have become center of researcher's attractions because:-
Most database based applications use sorting algorithms to provide information to their users in a pattern in which searching & manipulation of information is easy.
Many algorithms use sorting algorithms that require sorted lists to work correctly.
There are many sorting algorithms available to solve a problem and one has to choose the efficient one based on some criteria. In [9,3] they are classified by computational complexity (worst, average and best behavior Computational complexity of element comparisons in terms of the size of the list (n). For typical sorting algorithms good behavior is o(n log n) and bad behavior is $o(n^2)$.Ideal behavior for a sort is o(n), but this is not possible in the average case. Comparison-based sorting algorithms, which evaluate the elements of the list via an abstract key comparison operation, need at least o(n log n) comparisons for most inputs.
Computational complexity of swaps(for "in place" algorithms).
Memory usage (and use of other computer resources):-. In particular, some sorting algorithms are "in place". Strictly, an in place sort needs only o(1) or o(log(n)) memory beyond the items being sorted; sometimes o(log(n)) additional memory is considered "in place".
Recursion:. Some algorithms are either recursive or non-recursive, while others may be both
Stability:-stable sorting algorithms maintain the relative order of records with equal keys
Enhanced Insertion Sort.:a Enhanced Insertion Sort examines the data only by comparing two elements with a comparison operator.
General method:insertion, exchange, selection, merging, etc.. Exchange sorts include bubble sort and quicksort. Selection sorts include shaker sort and heap-sort.
Adaptability: whether or not the pre-sortedness of the input affects the running time. Algorithms that take this into account are known to be adaptive.
In this paper a new sorting algorithm is presented. It is very easy to implement & efficient. It is compared to bubble sort, insertion sort, selection sort & results shows that it is easier to implement than insertion sort, selection sort and efficient than bubble sort and insertion sort. The concept of the algorithm is explained in section 2. The 3rd Section represents the steps of algorithm. This paper also presents a derivation for running time in section 4. An example implementation is given in section 5 .a comparison of compare algorithm with other sorting algorithms(insertion sort, selection sort, bubble sort) is presented in section 6. Section 7 concludes the study followed by references.

2.CONCEPT

In this algorithm two adjacent elements are compared and then swapped on if second one is smaller than first one(for sorting in increasing order). Elements are inserted and then sorted in the same array without using any extra space(except element temp which is used in swapping).
This concept makes sorting easier than selection sort and insertion sort algorithms and reduces number of swaps, comparisons than selection sort and bubble sort algorithm.

3.ALGORITHM

Enhanced Insertion Sort(a,n)
this algorithm sorts n elements of array a.
step 1:- set a[0]= -∞
step2:-repeat steps 3 to 4 for k=2,3….n
step3:-ptr=k-1;
step4:-repeat while (a[k]<a[ptr])
(a):- swap a[k] & a[ptr]
(b):-k=ptr, ptr=ptr-1
step5:-exit.

4.PSEODOCODE AND RUNNING TIME ANALYSIS OF ALGORITHM

The efficiency of an algorithm depends on its use of resources, such as:
The time it takes the algorithm to execute;
The memory it uses for its variables;
The network traffic it generates;
The number of disk accesses it makesetc.
The proposed work is going to focus pretty much exclusively on time.
Pseudocode and corresponding cost factor of each line with the no of times a line executes is given below

CODE	COST	TIME
1. FOR J-> 2TO INDEX DO	C1	n
{BEGINE FOR}		
2.PTR->N-1	C2	n-1
3.WHILE(A(PTR)>A(J)	C3	$\sum t_J$
{BEGIN WHILE}		
4.TEMP=A(J)	C4	t_j-1
5.A(J)=A(PTR)	C5	t_j-1
6.A(PTR)=TEMP	C6	t_j-1
7.J=PTR		t_j-1
8.PTR=PTR-1		t_j-1
{END WHILE}		
{END FOR}		

HERE t_j=Number of times while loop executes for j
$T(N)=C_1N+(C_2 – C_4 – C_5 – C_6 - C_7 - C_8)(n-1) +(C_3+ C_4 + C_5 + C_6 + C_7 + C_8)(\sum t_J)$……eq.(1)
For calculating running time following should be done:
First consider only leading terms and ignore lower order terms because they become insignificant for large N.
Secondly ignore constant factor cost because it is also less significant than rate.
Running time for average case
If the input array is unsorted then
If j=2 then 2 comparisons takes place,
if j=3 then 3 comparisons takes place.
So, for each j the number of comparisons is equal to

$$\sum \frac{j+1}{2}$$

Total number of expected comparison:-

$$\sum_{j=2}^{n} \frac{j+1}{2} = \frac{1}{2}\sum_{j=2}^{n}(j+1)$$

$$=(\sum_{j=2}^{j=n} j + \sum_{j=2}^{j=n} 1)/2$$

$=((n^2+n-2)/2 +(n-1))/2$
$=(((n^2+n-2)/4)+((n-1)/2))$
$=(n^2+3n-4)/2$
By using above expression in equation (1), the running time, $T(n)=o(n^2)$
Running time for average case
In this case the input array is already sorted. So only one key comparison is needed for each j. In this case, –

$$\sum_{j=2}^{j=n} tj = \sum_{j=2}^{j=n} 1 = n-1$$

By using above expression in equation (1) , running time, T(n)=o(n)
Running time for average case
In this case the input array is sorted in reverse order. So j number of key comparisons are needed for each j. In this case

$$\sum_{j=2}^{j=n} tj = \sum_{j=2}^{j=n} j$$

$=(n^2+n-2)/2$

By using above expression in equation, running time, $T(n)=o(n^2)$.

The following time table shows the run-time summary of Enhanced Insertion Sort algorithm

Criteria	Run-time
Best case	O(n).
Average case	$O(n^2)$.
Worst case	$O(n^2)$.

5. EXAMPLE IMPLEMENTATION OF COMPARE SORT

In following example the two adjacent underlined elements show a comparison and there will be a swap if later element is smaller than former one.

INPUT ARRAY IS	**30,20,40,41,19,18**
STEP-1(I)	- ∞, **30, 20**, 40, 41, 19, 18
STEP-1(II)	- ∞, **20**, 30, 40, 41, 19, 18
STEP-2	- ∞, 20, **30, 40**, 41, 19, 18
STEP-3	- ∞, 20, 30, **40, 41**, 19, 18
STEP-4(i)	- ∞, 20, 30, 40, **41, 19**, 18
STEP-4(ii)	- ∞, 20, 30, **40, 19**, 41, 18
STEP-4(iii)	- ∞, 20, **30, 19**, 40, 41, 18
STEP-4(iv)	- ∞, **20, 19**, 30, 40, 41, 18
STEP-4(v)	- ∞, **19**, 20, 30, 40, 41, 18
STEP-5 (i)	- ∞, 19, 20, 30, 40, **41, 18**
STEP-5(ii)	- ∞, 19, 20, 30, **40, 18**, 41
STEP-5(iii)	- ∞, 19, 20, **30, 18**, 40, 41
STEP-5(ii)	- ∞, 19, **20, 18**, 30, 40, 41
STEP-5(iv)	- ∞, **19, 18**, 20, 30, 40, 41
STEP-5(v)	- ∞, **18**, 19, 20, 30, 40, 41
SHORTED ARRAY	**18, 19, 20, 30, 40, 41**

At the end of step 5 the output is a sorted array (in increasing order)
18, 19, 20, 30, 40, 41

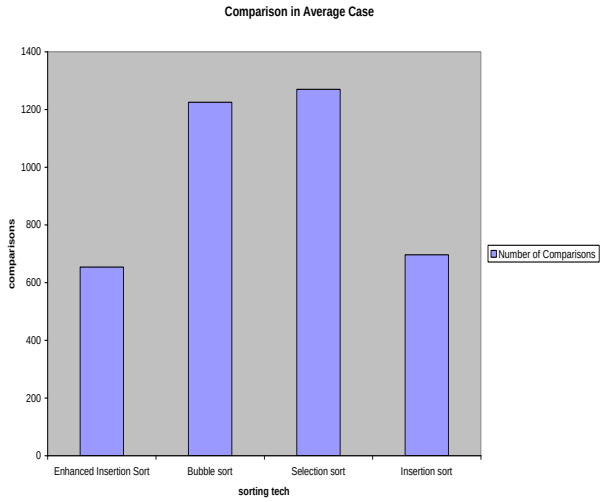
6. COMPARISON WITH SOME SORTING ALGORITHMS

Following table shows the comparison of all three cases of Enhanced Insertion Sort, selection sort and bubble sort

Name of algo.	Average case	Best case	Worst case	Memory
Enhanced Insertion Sort	$O(n^2)$	O(n)	$O(n^2)$	O(1)
Bubble sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)
Insertion sort	$O(n^2)$	O(n)	$O(n^2)$	O(1)

Although the running time of Enhanced Insertion Sort is same as that of insertion sort but implementation of Enhanced Insertion Sort is easier then insertion sort as because:
In insertion sort, the elements are to be shifted for making the right place of element j in j^{th}iteration but in the new purposed sorting algorithm, just two elements have to be compared and then swap them if necessary.
Swapping after comparison is easier than making the track of element at proper place by shifting elements(forward or backward according to condition)
Following table shows the comparison with bubble sort and selection sort in terms of swaps and comparisons on following array
20,30,29,28,50,14,13,12,21,11,22,9,33,32,34,36,35,38,39 ,37,31,40,41,49,48,47,2,3,46,4,42,43,45,44,8,7,10,15,6,1 6,17,15,18,19,24,23,25,27,26

Name	Criteria	Elements	Comparison	Swaps
Enhanced Insertion Sort	Average	50	654	631
	Best	50	49	0
	Worst	50	1274	1225
Bubble sort	Average	50	1225	629
	Best	50	1225	0
	Worst	50	1225	1225
Selection sort	Average	50	1270	45
	Best	50	1250	0
	Worst	50	1250	25
Insertion sort	Average	50	696	647
	Best	50	49	0
	Worst	50	1274	1225



Graph 1

As shown in graph 1 the enhanced insertion sort consumes less number of comparisons when applied on an unsorted array.

7. CONCLUSION

In this paper a new sorting algorithm has been presented. Enhanced Insertion Sort has complexity $o(n^2)$ but it requires less number of comparisons than bubble sort, selection sort. Enhanced insertion sort is easier in implemention terms than insertion sort. Mainly it bridges the gap between easiness and complexity, and improves number of comparisons and swaps. As number of swaps and comparisons needed are lesser then some other existing sorting algorithm(like insertion sort, selection sort, bubble sort) so enhanced insertion sort improves searching by taking lesser amount of time.

8. References

[1]. Weiss M., Data Structures and Problem Solving Using Java, Addison Wesley, 2002.
[2]. Demuth, H. Electronic Data Sorting. PhD thesis, Stanford University, 1956
[3]. Aho A., Hopcroft J., and Ullman J., The Design and Analysis of Computer Algorithms, Addison Wesley, 1974.
[4]. Box R. and Lacey S., "A Fast Easy Sort," Computer Journal of Byte Magazine, vol. 16, no. 4, pp. 315-315, 1991.
[5]. Cormen T., Leiserson C., Rivest R., and Stein C., Introduction to Algorithms, McGraw Hill, 2001.
[6]. Deitel H. and Deitel P., C++ How to Program, Prentice Hall, 2001.
[7]. Astrachanm O., Bubble Sort: An Archaeological Algorithmic Analysis, Duk University, 2003.
[8]. Friend E., "Sorting on Electronic Computer Systems," Computer Journal of ACM, vol. 3, no. 2, pp. 134-168, 1956.
[9]. The International Arab Journal of Information Technology, Vol. 7, No. 1, January 2010 55 An Enhancement of Major Sorting Algorithms JehadAlnihoud and Rami Mansi Department of Computer Science, Al al-Bayt University, Jordan
[10]. Bell D., "The Principles of Sorting," Computer Journal of the Association for Computing Machinery, vol. 1, no. 2, pp. 71-77, 1958.
[11]. Knuth D., The Art of Computer Programming, Addison Wesley, 1998.
[12]. Ledley R., Programming and Utilizing Digital Computers, McGraw Hill, 1962.
[13]. Levitin A., Introduction to the Design and Analysis of Algorithms, Addison Wesley, 2007.
[14]. Nyhoff L., An Introduction to Data Structures, Nyhoff Publishers, Amsterdam, 2005.
[15]. Kruse R., and Ryba A., Data Structures and Program Design in C++, Prentice Hall, 1999.
[16]. Organick E., A FORTRAN Primer, Addison Wesley, 1963.
[17]. Pratt V., Shellsort and Sorting Networks, Garland Publishers, 1979.

Authors' Profiles

Karamjeet Kaur is working as assistant professor in computer science in university college, KUK, India for last Two years. She has completed her post graduation from Kurukshetra University. She is NET qualified and her area of interest are Algorithm development , Networks and Data structure.