Research Journal of Applied Sciences, Engineering and Technology 8(2): 263-271, 2014
DOI:10.19026/rjaset.8.969
ISSN: 2040-7459; e-ISSN: 2040-7467
© 2014 Maxwell Scientific Publication Corp.
Submitted: April 15, 2014 Accepted: May 25, 2014 Published: July 10, 2014

Research Article

Enhancement of Selection, Bubble and Insertion Sorting Algorithm

Muhammad Farooq Umar, Ehsan Ullah Munir, Shafqat Ali Shad and Muhammad Wasif Nisar
Department of Computer Science, COMSATS Institute of Information Technology, Wah Cantt, Pakistan

Abstract: In everyday life there is a large amount of data to arrange because sorting removes any ambiguities and make the data analysis and data processing very easy, efficient and provides with cost less effort. In this study a set of improved sorting algorithms are proposed which gives better performance and design idea. In this study five new sorting algorithms (Bi-directional Selection Sort, Bi-directional bubble sort, MIDBiDirectional Selection Sort, MIDBidirectional bubble sort and linear insertion sort are presented. Bi-directional Selection Sort and MIDBiDirectional Selection Sort are the enhancement on basic selection sort while Bidirectional bubble sort and MIDBidirectional bubble sort are the enhancement on basic bubble sort by changing the selection and swapping mechanism of data for sorting. Enhanced sorting algorithms reduced the iteration by half and quarter times respectively. Asymptotically complexities of these algorithms are reduced to O ($n^2/2$) and O ($n^2/4$) from O (n^2). Linear insertion sort is the enhancement of insertion sort by changing the design of algorithm (convert two loops to one loop). So asymptotically this algorithm is converted to linear time complexity from quadratic complexity. These sorting algorithms are described using C. The proposed algorithms are analyzed using asymptotic analysis and also using machine-running time and compared with their basic sorting algorithms. In this study we also discuss how the performance and complexity can be improved by optimizing the code and design.

Keywords: 2-element insertion sort, 2-way bubble sort, 2-way mid bubble sort, 2-way mid selection sort, 2-way selection sort, asymptotic analysis, bubble sort, insertion sort, running time, selection sort

INTRODUCTION

In algorithmic engineering Sorting is very profound area for researchers and algorithmic engineers. Sorting also improves performance of logarithms and make suitable, efficient and convenient for usage data. There are many sorting algorithms are proposed for sorting data which have some pros and cons respectively in terms of its efficiency and algorithmic complexity by Friend (1956). Some sorting algorithms are considered basic or well known for sorting the unsorted data like Selection sort, Bubble sort by Astrachan (2003), Quick sort by Hoare (1962), Insertion sort by Cormen *et al*. (2004) and Shell (1959). But still many researchers are working for the optimization of these algorithms.

Optimization is much more important factor for writing or designing algorithms. The output produced by the algorithm is also most important factor, which should be reliable and convenient to use, read and understand by Deitel and Deitel (1998) and Friend (1956). The output of sorting algorithm should be in non-decreasing order by using permutation or reordering concepts by Mansi (2010).

The sorting algorithms can be categorized in two types, which are called internal sort and external sort. Internal sort uses the data residing in RAM while in external sort data is very large so that data may be resided on external memory by Xiaokai (1995), Weimin and Weimin (2000) and Min (2010). Results of sorting algorithm are abundant and are not absolute because these are specific to some factors. It can be used as traditional knowledge by Khamitkar *et al*. (2010).

Analysis of sorting algorithms also depends on the type of the data. Analysis of algorithms by considering all input is very difficult. The time complexity of algorithm can be analyzed by considering just one input with the help of Kolmogorov complexity by Vitányi (2007) and Xusong (1996).

Selection, bubble and insertion are characterizes as comparison sort because these algorithms sort the data by comparing the data element with each other and selects the element location and swap on that location. These all costs O (n^2) in worst case scenario which is the basic limitation of comparison sort techniques. Another limitation of these algorithms is that these also has n2 times iteration in worst case but it can be reduced by optimizing these algorithms and in this study the approach followed is same.

In this study five algorithms are presented two algorithms are the extension of selection sort and two algorithms are the extension of bubble sort and one algorithm is the enhancement of insertion sort. These

Corresponding Author: Muhammad Farooq Umar, Department of Computer Science, COMSATS Institute of Information Technology, Wah Cantt, Pakistan
This work is licensed under a Creative Commons Attribution 4.0 International License (URL: http://creativecommons.org/licenses/by/4.0/).

presented algorithms are optimized, efficient asymptotically as well as running time wise.

LITERATURE REVIEW

Selection sort: In the basic selection sort algorithm we sort the data by selecting (maximum or minimum) element in the selected list by decreasing the list size from n to 2 with the decrement of size by one in each iteration. This has the two loops outer and inner loop. Outer loop manages the size of list to be processed for finding maximum or minimum number and the inner loop is for finding the smallest or biggest number from list selected by outer loop.

The asymptotic complexity of basic selection sort in worst case is O (n^2) which is due to comparisons of each data element with each other and its number of iterations. So these algorithms can be improved or enhanced by reducing the number of iterations or comparisons. In this study two enhancement of selection sort algorithms have been discussed which have been discussed below.

Bi-directional selection sort algorithm:
Concept: In this enhancement of selection sort algorithm we sort the data by selecting (maximum and minimum) elements in the selected list by decreasing the list size from n to 2 with the decrement of two in the size. This has the two loops outer and inner loop. Outer loop manages the size of list to be processed for finding maximum and minimum number and the inner loop is for finding the smallest and biggest number from list selected by outer loop.

Pseudo code: Pseudo code for bi-selection sort is given below:

```
Function Bi Selection Sort (array, size)
1 var i, j
2 var min, max, temp
3 for i: = 0 to size/2 do
4 min: = max: = i
5 for j: = i+1 to size-i do
6 if array (j) <array (min)
7 min: = j
8 end if
9 if array (j) >array (max)
10 max: = j
11 end if
12 end for j
13 if array (i) <> array (min)
14 temp: = array (i)
15 array (i): = array (min)
16 array (min): = temp
17 if i <> max
18 temp: = array (i)
19 array (i): = array (max)
20 array (max): = temp
21 end for i
```

In this pseudo code we use array subscript form where Array () is the array to be sort, min is the variable for smallest number selected from the selected list from outer loop and temp is a temporary variable used for swapping.

Running time: The recurrence for Bi-selection sort algorithm is:

$$T(n) = 0 \qquad n = 0$$
$$= T(n-2) + n \qquad n>0$$
$$T(n) = n + T(n-2)$$
$$= n + n - 1 + T(n-4)$$
$$= n + n - 1 + n - 2 + T(n-6)$$
$$= n + n - 1 + n - 2 + n - 3 + T(n-8)$$
$$= \ldots$$
$$= n + n - 1 + n - 2 + n - 3 + \ldots + (n - k + 1) + T(n - 2k) \text{ for } n \geq k$$

To terminate the recursion, we should have n-2k = 0 => 2k = n.

By solving this recurrence with the help of any algorithm gives its order of growth as:

$$T(n) = O((n^2)/2$$
$$T(n) = O(n^2)$$

Bi-directional mid selection sort algorithm:
Concept: In this enhancement of selection sort algorithm we sort the data by selecting (maximum and minimum) elements by starting searching from middle to both sides in the selected list by decreasing the list size from n to 2 with the decrement of two in the size. This has the two loops outer and inner loop. Outer loop manages the size of list to be processed for finding maximum and minimum number and the inner loop is for finding the smallest and biggest number from list selected by outer loop.

Pseudo code: Pseudo code for basic selection sort is given below:

```
Function BiMID Selection Sortn (array, size)
1 var i, j, left, right
2 var min, max, temp
3 If limit <2
4 return
5 right: = left: = limit/2
6 if limit mod 2
7 right++
8 for j: = 0 to size/2 do
9 if array [left] > array [right]
10 max: = left
11 min: = right
12 if array [left] <= array [right]
13 min: = left
```

```
14 max: = right
15 for i: = j to (size-j)/2 do
16 if array (left-i) <array (min)
17 min: = left-i
18 end if
19 if array (left - i) >array (max)
20 max: = left-i
21 end if
22 if array (right + i) <array (min)
23 min: = right + ii
24 end if
25 if array (right + i) >array (max)
26 max: = right + i
27 end if
28 end for i
29 if array (j) <> array (min)
30 temp: = array (j)
31 array (j): = array (min)
32 array (min): = temp
33 if size-1-j <> max
34 temp: = array (size - 1 - j)
35 array (size - 1 - j): = array (max)
36 array (max): = temp
37 end for j
```

In this pseudo code we use array subscript form where Array() is the array to be sort, min is the variable for smallest number selected from the selected list from outer loop and temp is a temporary variable used for swapping.

Running time: The recurrence for Bi-MIDselection sort algorithm is:

$$
\begin{aligned}
T(n) &= 0 \qquad n = 0 \\
&= T(n-2) + n/2 \qquad n>0 \\
T(n) &= n/2 + T(n-2) \\
&= (n + n - 1)/2 + T(n-4) \\
&= (n + n - 1 + n - 2)/2 + T(n-6) \\
&= (n + n - 1 + n - 2 + n - 3)/2 + T(n-8) \\
&= \ldots \\
&= (n + n - 1 + n - 2 + n - 3)/2 + \ldots + \\
&(n - k + 1)/2 + T(n - 2k) \\
&=, \text{ for } n \geq k
\end{aligned}
$$

To terminate the recursion, we should have $n/2 - 2k = 0 \Rightarrow 4k = n$.

By solving this recurrence with the help of any algorithm gives its order of growth as:

$$T(n) = O((n^2)/4)$$
$$T(n) = O(n^2)$$

The bubble sort algorithm: In the basic bubble sort algorithm we sort the data by finding and replacing the (maximum or minimum) elements in the selected list by decreasing the list size from n to 2 with the decrement of size by one. This has the two loops outer and inner loop. Outer loop manages the size of list to be processed for finding and replacing maximum or minimum number and the inner loop is for finding and replacing the smallest or biggest number from list selected by outer loop.

The asymptotic complexity of basic bubble sort in worst case is $O(n^2)$ which is due to comparisons of each data element with each other and its number of iterations. So these algorithms can be improved or enhanced by reducing the number of iterations, comparisons or swapping. In this study two enhancement of bubble sort algorithms have been discussed which are given below.

Bi-directional bubble sort algorithm:

Concept: In this enhancement of bubble sort algorithm we sort the data by finding and replacing (maximum and minimum) elements in the selected list by decreasing the list size from n to 2 with the decrement of two in the size. This has the two loops outer and inner loop. Outer loop manages the size of list to be processed for finding maximum and minimum number and the inner loop is for finding and replacing the smallest and biggest number from list selected by outer loop.

Pseudo code: Pseudo code for basic bubble sort is given below:

```
Function Bi Bubble Sort (array, size)
1 var i, j
2 var temp
3 for i: = 0 to size/2 do
4 for j: = i+1 to size-i do
5 if array (j) < array (i)
6 temp: = array (j)
7 array (j): = array (i)
8 array (i): = temp
9 end if
10 if array (j) >array (limit-1-i)
11 temp: = array (j)
12 array (j): = array (limit-1-i)
13 array (limit-1-i): = temp
14 end if
15 end for j
16 end for i
```

In this pseudo code we use array subscript form where Array() is the array to be sort, min is the variable for smallest number selected from the selected list from outer loop and temp is a temporary variable used for swapping.

Running time: The recurrence for Bi-bubble sort algorithm is:

T(n) = 0 n = 0
= T (n - 2) +n n>0
T(n) = n + T (n - 2)
= n + n - 1 + T (n - 4)
= n + n - 1 + n - 2 + T (n - 6)
= n + n - 1 + n – 2 + n - 3 + T (n - 8)
= …
= n + n - 1 + n - 2 + n - 3 + … +
(n - k + 1) + T (n - 2k)
=, for n≥k

To terminate the recursion, we should have n-2k = 0 => 2k = n.

By solving this recurrence with the help of any algorithm gives its order of growth as:

T (n) = O ((n^2) /2)
T (n) = O (n^2)

Bi-directional mid bubble sort algorithm:

Concept: In this enhancement of bubble sort algorithm we sort the data by finding and replacing (maximum and minimum) elements by starting searching from middle to both sides in the selected list by decreasing the list size from n to 2 with the decrement of two in the size. This has the two loops outer and inner loop. Outer loop manages the size of list to be processed for finding maximum and minimum number and the inner loop is for finding and replacing the smallest and biggest number from list selected by outer loop.

Pseudo code: Pseudo code for Bidirectional mid bubble sort is given below:

```
Function BiMID Bubble Sort (array, size)
1 var i, j, left, right
2 var min, max, temp
3 If limit <2
4 return
5 End if
6 right: = left: = limit/2
7 if limit mod 2
8 right++
9 End if
10 for j: = 0 to size/2 do
11 if array [left] >array [right]
12 max: = left
13 min: = right
14 if array [left] <= array [right]
15 min: = left
16 max: = right
17 for i: = j to (size-j) /2 do
18 if array (left-i) <array (j)
19 temp: = array (j)
20 array (j): = array (left-i)
21 array (left-i): = temp
22 end if
23 if array (left-i) >array (size-1-j)
24 temp: = array (size-1-j)
25 array (size-1-j): = array (left-i)
26 array (left-i): = temp
27 end if
28 if array (right+i) <array (j)
29 temp: = array (j)
30 array (j): = array (right+i)
31 array (right+i): = temp
32 end if
33 if array (right+i) >array (size-1-j)
34 temp: = array (size-1-j)
35 array (size-1-j): = array (right+i)
36 array (right+i): = temp
37 end if
38 end for i
39 end for j
```

In this pseudo code we use array subscript form where Array() is the array to be sort, min is the variable for smallest number selected from the selected list from outer loop and temp is a temporary variable used for swapping.

Running time: The recurrence for Bi-MIDBubble sort algorithm is:

T (n) = 0 n = 0
= T (n - 2) + n/2 n>0
T (n) = n/2 + T (n - 2)
= (n + n - 1) /2 + T (n - 4)
= (n + n - 1 + n - 2) /2 + T (n - 6)
= (n + n - 1 + n - 2 + n - 3) /2 + T (n - 8)
= …
= (n + n - 1 + n - 2 + n - 3) /2 + … +
(n - k + 1) /2 + T (n - 2k)
=, for n≥k

To terminate the recursion, we should have n/2 - 2k = 0 => 4k = n.

By solving this recurrence with the help of any algorithm gives its order of growth as:

T (n) = O ((n^2) /4)
T (n) = O (n^2)

Insertion sort algorithm:

Concept: In basic insertion sort algorithm we sort the data by inserting each element one by one in sorted list to its location according to its order starting list size from one to n. This has the two loops outer and inner loop. Outer loop manages the size of sorted list which increases by one after each iteration in which new element has to be inserted and the inner loop is for finding and inserting at the location according to sorting.

The asymptotic complexity of basic insertion sort in worst case is O (n^2) which is due to inserting each data element to its specific location and two loops. So this algorithm can be improved or enhanced by reducing the number of iterations, comparisons or loop. In this study one enhancement of insertion sort algorithm have been discussed which are given below.

Linear order insertion sort algorithm:

Concept: In this enhancement of insertion sort algorithm we sort the data by inserting an element one by one in sorted list to its location according to its order starting list size from one to n. This has just one loop outer and inner loop. Outer loop manages the size of sorted list in which new element has to be inserted and the inner loop is for finding and inserting at the location according to sorting.

Pseudo code: Pseudo code for linear insertion sort is given below:

```
Function Linear Insertion Sort (array, size)
1. temp = true
2. for i = 1 to size
3.    if temp == true
4.          Y = array (i)
5.          index←i
6.    End if
7.    if array (i-1) >y
8.          array (i) = array (i-1)
9.          If i == 1
10.           array (i-1) = y
11.           temp = true
12.           i = index
13.         else
14.           i = i-2
15.           temp = false
16.         End if
17.   else if temp == false
18.          array (i) = y
19.          i←index
20.          temp←true
21.        End if
22.   End for
```

In this pseudo code we use array subscript form where Array() is the array to be sort, size is the variable for representing the array size to be sorted.

Running time: The recurrence for linear insertion sort algorithm is:

$$T(n) = \begin{cases} 0 & n = 0 \\ T(n-1) + k & n > 0 \end{cases}$$

$$\begin{aligned} T(n) &= k + T(n-1) \\ &= k + k + T(n-2) \\ &= k + k + k + T(n-3) \\ &= k + k + k + k + T(n-4) \\ &= \ldots \\ &= k + k + k + k + \ldots + (n-y+1) + T(n-y) \\ &=, \text{ for } n \geq y \text{ and } n \geq k \end{aligned}$$

To terminate the recursion, we should have n - k = 0 => y = n.

By solving this recurrence with the help of any algorithm gives its order of growth as:

$$T(n) = O(kn)$$
$$T(n) = O(n)$$

RESULTS AND DISCUSSION

Analysis: The behavior of algorithms can be analyzed by determining their order of growth in terms of big-O notation for west case scenario.

Basic selection sort has the order of growth O (n^2) but the presented two enhancements of selection sort have the order of growth as O ($n^2/2$) and O ($n^2/4$) which is shown in Fig. 1.

In Fig. 1 it is clearly is showing that the enhanced algorithms have the much better order of growth. This order of growth of basic and enhanced selection sort can be described as Fig. 2.

From Fig. 2 it can be easily noticed that number of iterations (order of growth) has been reduced to half in Fig. 2b and quarter in Fig. 2c.

Basic bubble sort has the order of growth O (n^2) but the presented two enhancements of bubble sort have the order of growth as O ($n^2/2$) and O ($n^2/4$) which can be shown using chart as below.

In Fig. 3 it is clearly is showing that the enhanced algorithms have the much better order of growth. This order of growth of basic and enhanced selection sort can be described as Fig. 4.

From Fig. 4 it can be easily noticed that number of iterations (order of growth) has been reduced to half in Fig. 4b and quarter in Fig. 4c.

Basic insertion sort has the order of growth O (n^2) and has two nested loops but the presented enhancement of insertion sort have the order of growth as O (n + k) which can be says as O (n) which can be shown using chart as below in Fig. 5.

In Fig. 5 it is clearly is showing that the enhanced algorithms have the much better order of growth. This order of growth of basic and enhanced selection sort can be described as Fig. 6.

From Fig. 6 it can be easily noticed that number of iterations (order of growth) has been reduced to linear from quadratic in Fig. 6b.

Running time: All mentioned basic sorting algorithms with their enhancements are implemented using c++ on windows 7. The pc used for this purpose

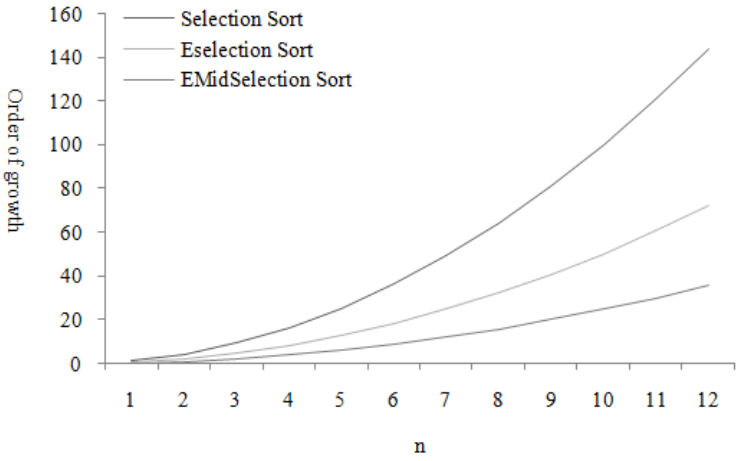


Fig. 1: Comparison of order of growth of basic selection sort and its enhancements

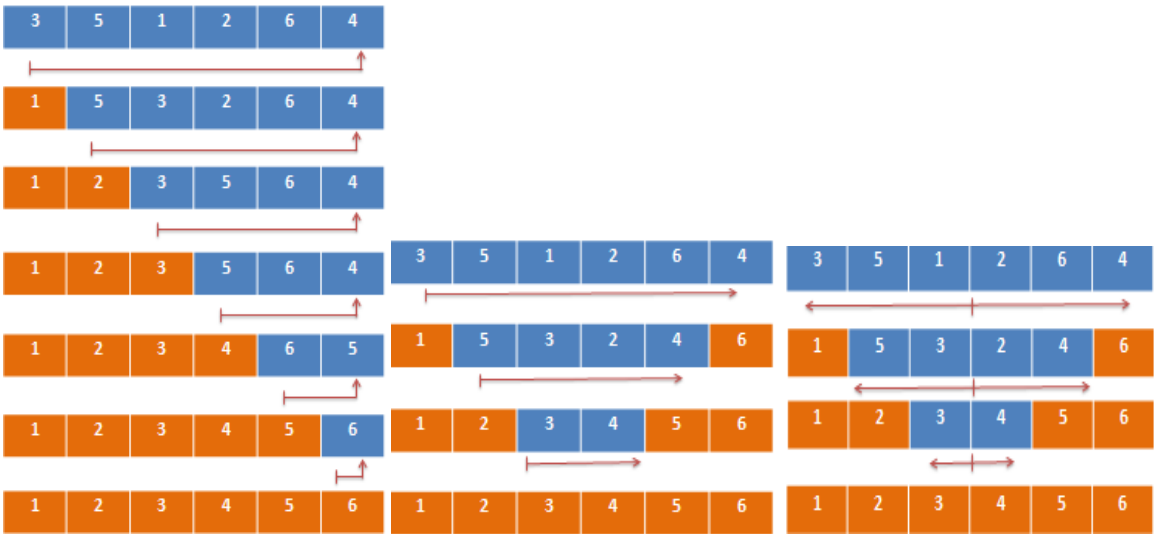


Fig. 2: Comparison of order of growth of basic selection sort and its enhancements, (a) standard selection sort, (b) bi-directional selection sort, (c) MIDBiDirectional selection sort

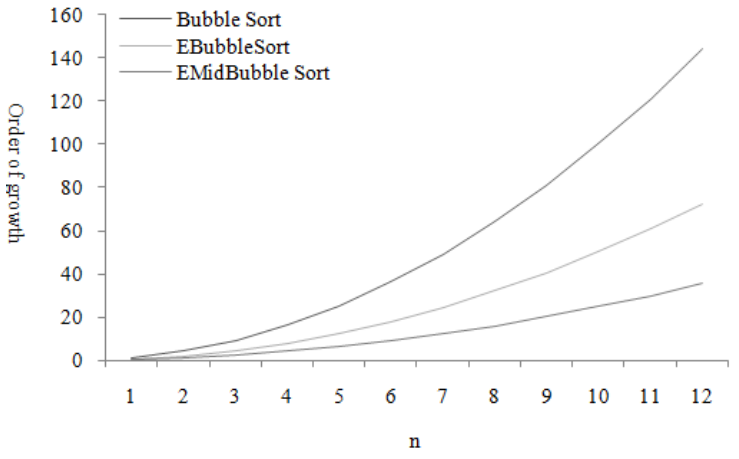
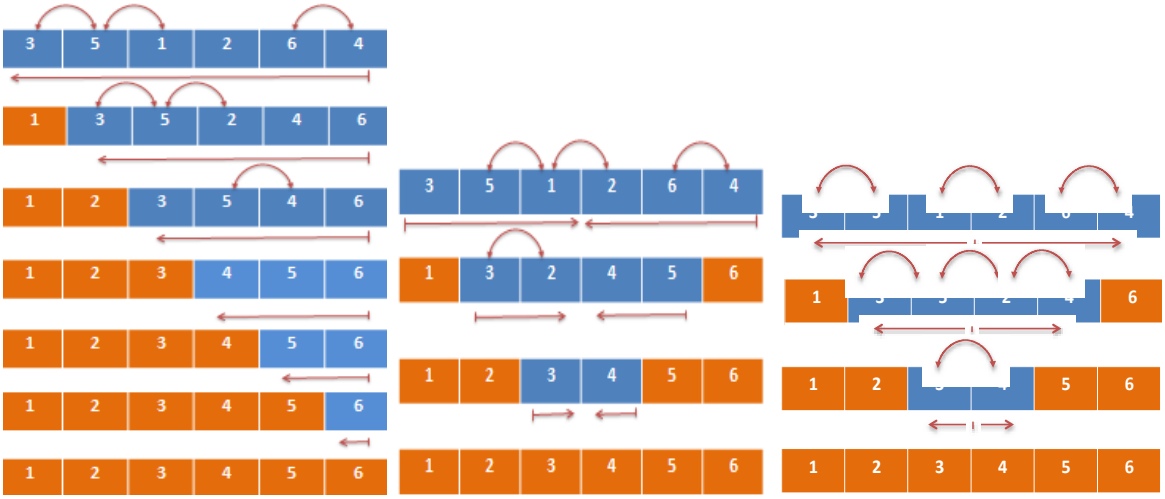


Fig. 3: Comparison of order of growth of basic bubble sort and its enhancements



(a) (b) (c)

Fig. 4: Comparison of order of growth of basic bubble sort and its enhancements, (a) standard bubble sort, (b) bi-directional bubble sort, (c) MIDBiDirectional bubble sort

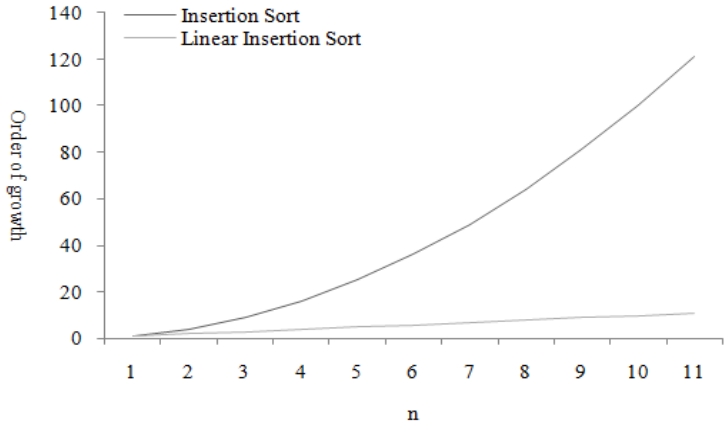


Fig. 5: Comparison of order of growth of basic insertion sort and its enhancements

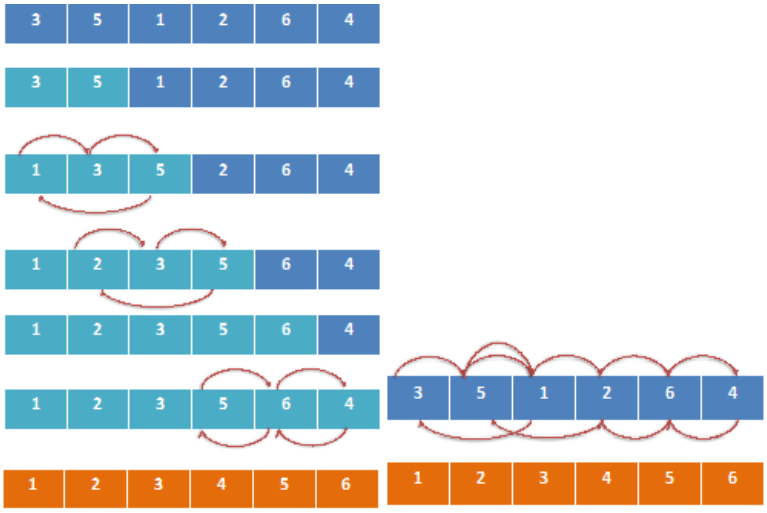


Fig. 6: Comparison of order of growth of basic insertion sort and its enhancements, (a) standard insertion sort, (b) linear insertion sort

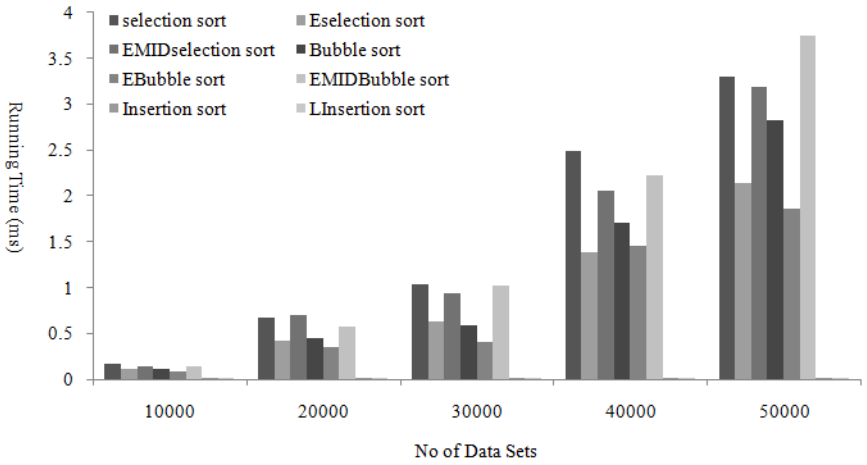


Fig. 7: Graph of machine running time of basic and their proposed sorting algorithms (best case)

Table 1: Machine running time of basic and their proposed sorting algorithms (best case)

Data set size	Selection sort	E selection sort	EMID selection sort	Bubble sort	E bubble sort	EMID bubble sort	Insertion sort	L insertion sort
10000	0.171	0.110	0.140	0.109	0.078	0.141	0.016	0.016
20000	0.671	0.421	0.702	0.453	0.343	0.577	0.016	0.016
30000	1.030	0.624	0.936	0.593	0.405	1.014	0.015	0.016
40000	2.496	1.389	2.059	1.700	1.451	2.230	0.016	0.016
50000	3.307	2.137	3.183	2.823	1.857	3.744	0.016	0.016

Table 2: Machine running time of basic and their proposed sorting algorithms (worst case)

Data set size	Selection sort	E selection sort	EMID selection sort	Bubble sort	E bubble sort	EMID bubble sort	Insertion sort	L insertion sort
10000	0.172	0.140	0.187	0.375	0.094	0.187	0.375	0.390
20000	0.484	0.374	0.484	0.889	0.234	0.406	0.983	0.889
30000	0.593	0.515	0.717	1.467	0.748	1.092	3.214	3.016
40000	1.856	1.685	1.716	2.886	1.373	2.481	6.287	5.054
50000	3.167	2.808	3.697	6.599	1.887	3.541	8.705	8.298

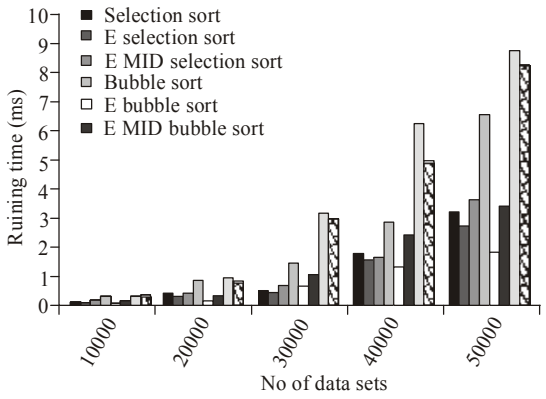


Fig. 8: Graph of machine running time of basic and their proposed sorting algorithms (worst case)

has the core i3 processor with 2 gb RAM. In this experiment, inputs are used in sorted order in best case while in reverse order in worst case. For both cases, results have been gotten for 10, 20, 30, 40 and 50 k number of data sets, respectively. The results of these algorithms are the machine running times which are given below in the form of two tables and charts as shown in Table 1 and Fig. 7 for each case.

Best time: As from above table and chart, it shows that in case of sorted datasets (best case) all enhancements of basic sorting algorithms performs better than their basic sorting algorithms for each set of data sets except EMID bubble sort which takes little more time.

Worst case: As shown in Table 2 and Fig. 8 it shows that in case of reversed sorted datasets (worst case) all enhancements of basic sorting algorithms performs better than their basic sorting algorithms for each set of data sets.

CONCLUSION

In this study five new sorting algorithms are presented by enhanced the selection, bubble and insertion sort algorithms. These are named as E selection Sort, E Mid Selection Sort, E Bubble Sort, E Mid Bubble Sort and linear insertion Sort. As from experiments by using integers as inputs it is showed that these enhanced algorithms are much better in terms of order of growth as well as machine running time by using Intel core i3 processor. These are implemented and executed on Intel core i3 processors. In these

algorithms the complexity of algorithm, order of growth and machine running time has been improved by improving the algorithm's design and execution.

REFERENCES

Astrachan, O., 2003. Bubble sort: An archaeological algorithmic analysis. Proceeding of the 34th SIGCSE Technical Symposium on Computer Science Education (SIGCSE '03), 35(1): 1-5.

Cormen, T., C. Leiserson, R. Rivest and C. Stein, 2004. Introduction to Algorithms. 1st Edn., McGraw Hill, USA.

Deitel, H.M. and P.J. Deitel, 1998. *C*++ How to Program. 1st Edn., Prentice Hall, Upper Saddle River, NJ.

Friend, E., 1956. Sorting on electronic computer systems. J. ACM (JACM), 3(3): 134-168.

Hoare, C., 1962. Quick sort. Comput. J., 5(1): 10-16.

Khamitkar, S., P. Bhalchra, S. Lokhe and N. Deshmukh, 2010. The folklore of sorting algorithms. Int. J. Comput. Sci. Issues, 7(4).

Mansi, R., 2010. A new algorithm for sorting small integers. Int. Arab J. Inf. Techn., 7(2).

Min, W., 2010. Analysis on bubble sort algorithm optimization. Proceeding of International Forum on Information Technology and Applications (IFITA, 2010), 1: 208-211.

Shell, D., 1959. A high-speed sorting procedure. Commun. ACM, 2(7): 30-32.

Vitányi, P., 2007. Analysis of Sorting Algorithms by Kolmogorov Complexity (a Survey). In: Csiszár, I., G.O.H. Katona and G. Tard´os (Eds.), Entropy Search Complexity. Number 16 in Bolyai Society Mathematical Studies, pp: 209-232.

Weimin, Y. and W. Weimin, 2000. Data Structures. 1st Edn., Tsinghua Universit Press, Beijing, pp: 263-268.

Xiaokai, X., 1995. Simple Data Structure Tutorial. 1st Edn., Tsinghua University Press, Beijing, pp: l93-196.

Xusong, X., 1996. Introduction to Data Structures and Algorithms. 1st Edn., Electronics Industry Press, Beijing, pp: 162-164.