Index Sort

Hari Krishna Gurram *harikrishna553@gmail.com*
M.Tech (CS),
Ucek, JNTU Kakinada.

Gera Jaideep *jaideep.gera@gmail.com*
M.Tech(CS),
Ucek, JNTU Kakinada.

Abstract

One of the fundamental issues in computer science is ordering a list of items. Although there is a number of sorting algorithms, sorting problem has attracted a great deal of research, because efficient sorting is important to optimize the use of other algorithms. This paper presents a new sorting algorithm (Index Sort) which runs based on the previously sorted elements.. This algorithm was analyzed, implemented and tested and the results are promising for a random data.

Keywords: Index Sort, Binary Search, Position.

1. INTRODUCTION

Today real world getting tremendous amounts of data from various sources like data warehouse, data marts etc. To search for particular information we need to arrange this data in a sensible order. Many years ago, it was estimated that more than half the time on commercial computers was spent in sorting. Fortunately variety of sorting algorithms came into existence with different techniques [1].

Many algorithms are well known for sorting the unordered lists. Most important of them are merge sort, heap sort, shell sort and quick sort etc. [2]. As stated in [3], sorting has been considered as a fundamental problem in the study of algorithms, that due to many reasons:

- The need to sort the information is inherent in many applications.
- Algorithms often use sorting as a key subroutine.
- Many engineering issues come to the fore when implementing sorting algorithms.
- In algorithm design, there are many essential techniques represented in the body of sorting algorithms.

Sorting algorithms plays a vital role in various indexing techniques used in data warehousing, and daily transactions in online Transactionalprocessing (OLTP). Efficient sorting is important to optimize the use of other sorting algorithms that require sorted lists correctly.
Sorting algorithms can be classified by:

1. Computational complexity (best, average and worst behavior) of element comparisons in terms list size n. For a typical sorting algorithm best case, average case and worst case is O(n log n), example merge sort.
2. Number of swaps
3. Stability : A sorting algorithm is stable if whenever there are two records X and Y, with the same key and X appearing before Y in original list, X will be appear before Y in the sorted list.
4. Usage of memory

In this paper, a new sorting algorithm (Index sort) is proposed; here the basic idea is first we are going to sort the 2 elements and these 2 elements acts as an index to next 4 elements to sort, so totally 6 elements sorted, next these 6 elements are used as an index to sort 12 elements so totally 18 elements sorted, this process continues until all elements are sorted.

Section 2 presents the concept of Index sorting algorithm and its pseudo code. Section 3 shows the implementation results for various sizes of input. Finally, the conclusion was presented in section 4.

2. INDEX SORT

2.1 Concept

The main logic presented here is initially we sort forst two elements in the given input. An index is created by using these two elements which is used to sort twice the elements in the index, i.e., if index has ‘n’ number of elements, then ‘2n’ elements are sorted by this index. This process repeated until all the elements sorted. To get better results we are using binary search to search for the position of the element.

2.2 Pseudocode

```
functionindexSort( input, noEle )
    1. initialSort(input)
    2. varnoEle := 2
    3. sort( input, noEle )
    4. end indexSort
```

Line 1 of indexSort(input, noEle), calls the method initialSort(input), which sort the first two elements

in the given input. The pseudo code for initialSort(input) is given below.

```
functioninitialSort(input)
        Sort the first two elements in the given input.
endinitialSort
```

Line 2 of indexSort(input, noEle), calls the sort method which sort the remaining all elements by using this index. The pseudo code for sort method is given below.

```
Global variables:
var length := input.length
var flag := 0
    var position
function sort( input, noEle)
    1.  Node index[noEle]
    2.  copyFromArrayToNode( input, index, noEle )
    3.  varsortPos := 3*noEle
    4.  var length1 := index.length
    5.  if( sortPos>len )
    6.  sortPos := len
    7.    flag := 1
    8.  end if
    9.  for i in noEle to sortPos
    10.   if ( input[i] < index[1].element )
    11.         position := 1
    12.   else if ( input[i] > index[len].element )
    13.         position := len
    14.   else
    15.         Position := binarySearch( index, 1, len, input[i] )
    16. once the position is found, then compare the element input[i] with the
        index[position] and place input[i] in proper place by making single linked list,
        since index[] is of type Node.
    17. end LOOP
    18. copyFromNodeToArray( index, input )
    19. If ( flag = 0 )
    20.   Sort( input, sortPos )
```

Line 1 of sort(input, noEle) creats an array index of type Node, Node is of structure type, the definition of Node is given below.

```
Struct Node
        int element
        Node next
end Node
```

Line 2 of sort(input, noEle) calls the method copyFromArrayToNode(input, index, noEle), this method creates an index which is used to sort the elements in the given input, the index is created by using first ‘noEle’ elements of the input array. The pseudo code for copyFromArrayToNode(input, index, noEle) is given below.

functioncopyFromArrayToNode(input, index, noEle)

```
1.  for i 1 to noEle
2.      in[i].element := input[i]
3.      In[i].next := NULL
4.  end LOOP
5.  end copyFromArrayToNode
```

Line 4 of sort method has below type of declaration:

```
var length1 := index.length
```

hereindex.length returns the length of the array index, then this value is assigned to length1, same case for global variable length.

Lines 5 -8 has a condition, which checks when to stop the sorting procedure, and limit the number of elements to the size of given input data.

Lines 9-17 sorts the given elements from noEle to 3*noEle, since all the elements uptonoEle in the input array are already sorted, so by using this as index, 2*noEle elements of input array are sorted. To make this algorithm efficient we are using binary search to find out the position of the given input element in the index.

The pseudocode for binarySearch is given below.

functionbinarySearch(Node array[], var low, var high, var element)

1. find the position such that element is greater than or equal to array[position] and less than array[position+1].
2. return position
3. endbinarySearch.

Line 18 of sort method calls copyFromNodeToArraymethod which copy the sorted elements in the index to the input array, the pseudo code for copyFromNodeToArray is given below.

Function copyFromNodeToArray(Node in[], var input[])

```
1.  var length := in.length
2.  var count := 1
3.  for I in 1 to length
4.     while ( in[i] != NULL )
5.         Input[count] := in[i].element
6.         count := count +1
7.         In[i] := in[i].next
8.     end LOOP
9.  end LOOP
10. end copyFromNodeToArray
```

Line 20 calls the sort method recursively unti all the elements in the given array are sorted.

The index creation and the number of elements sorted by this index are explained by using the below table.

S.No	Index created by number of elements	Number of elements sorted by using the index	Total sorted elements
1	2	4	6
2	6	12	18
3	18	36	54
4	54	108	162
5	162	324	486
6	486	972	1458
7	1458	2916	4374
8	4374	8748	13122
9	13122	26244	39366
10	39366	78732	118098
11	118098	236198	354294
12	354294	708588	1062882
13	1062882	2125764	3188646
14	3188646	6399292	9564938
15	9565938	10434062 (Actually the valu is 19131876 but I t exceeded the input size 2 chrore)	20000000

TABLE 1: Shows how the index is created and elements are sorting.

3. IMPLEMENTATION RESULTS

Below graphs shows the implementation results of Index Sort.

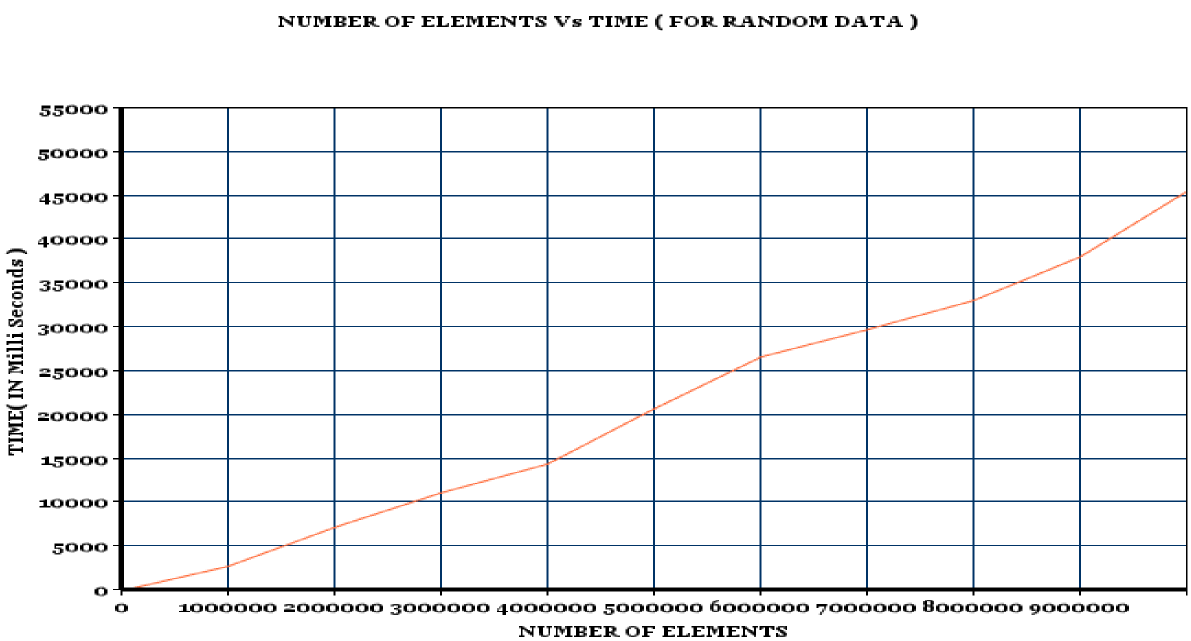

FIGURE 1 : Sorting the random elements (Number of elements Vs Time)

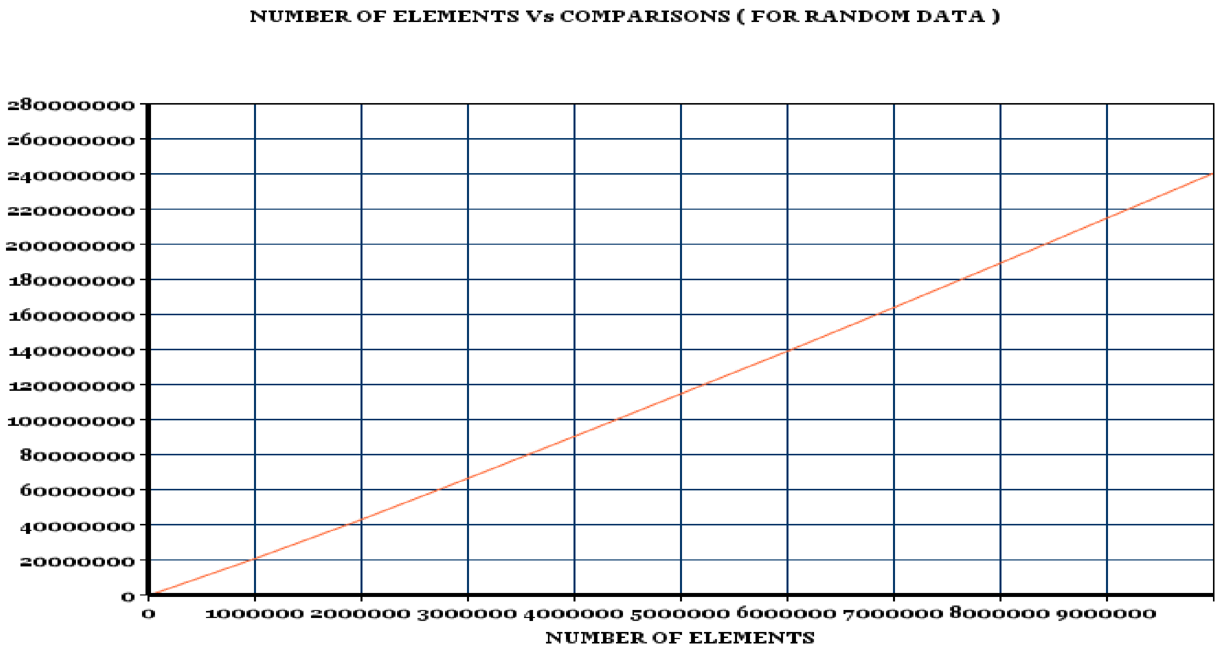

FIGURE 2 : Sorting the random elements (Number of elements Vs Comparisons)

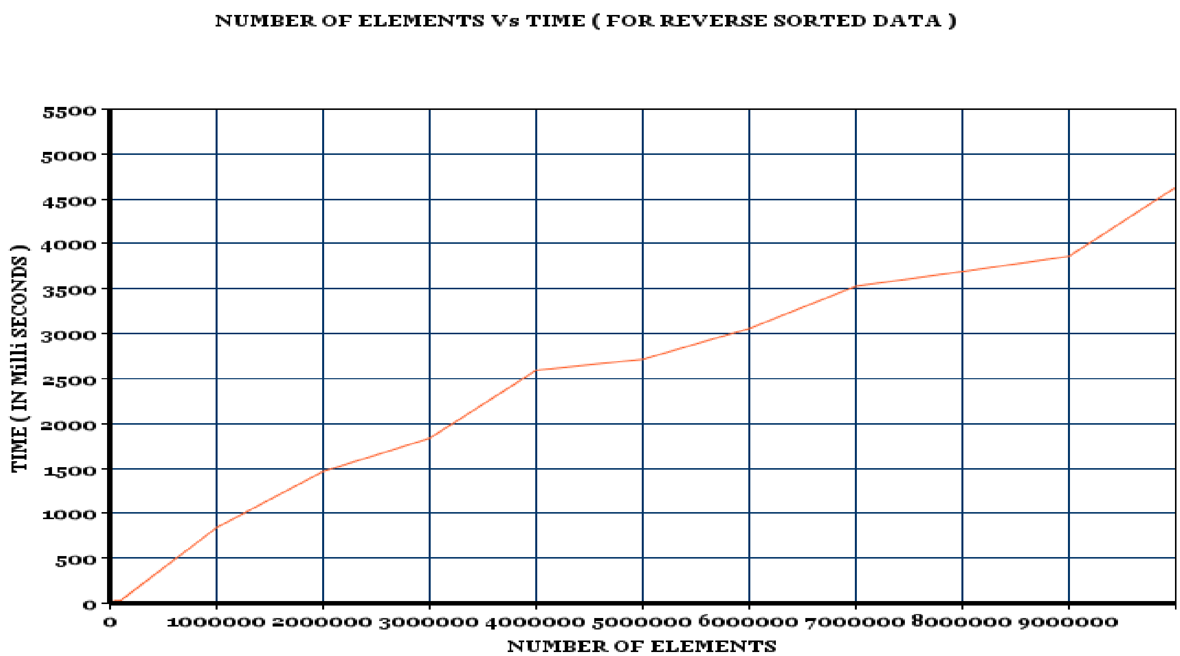

FIGURE 3: Sorting the reversely sorted elements (Number of elements VsTime)

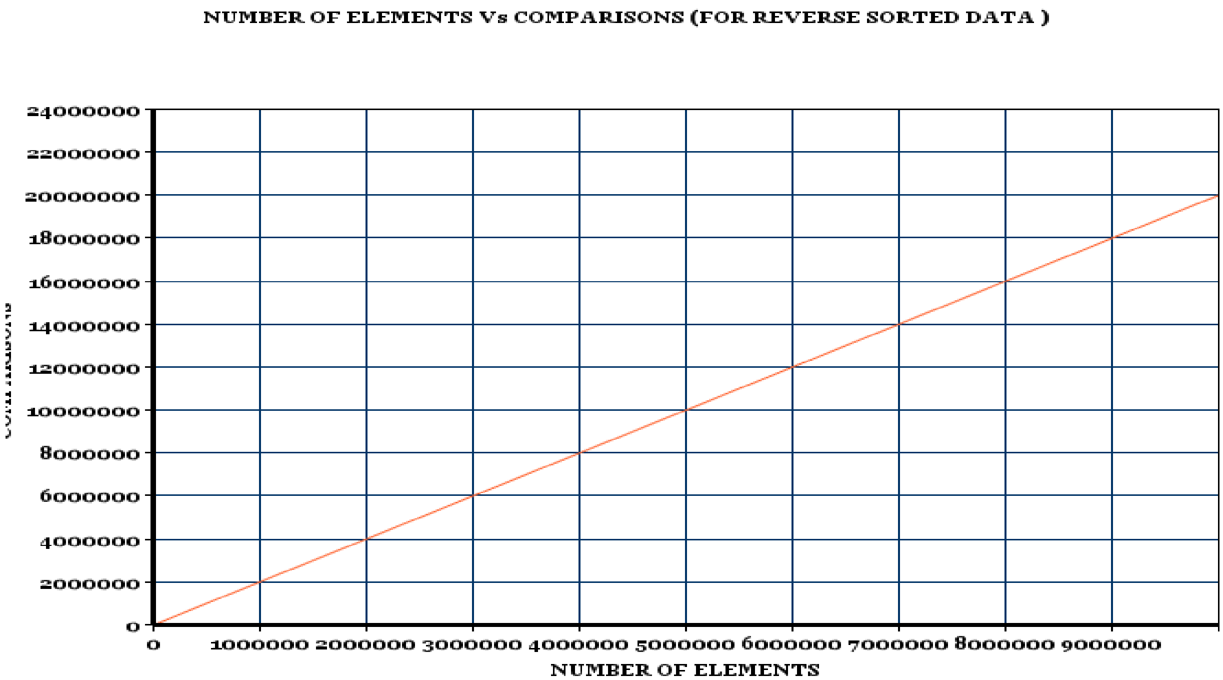

FIGURE 4: Sorting the reversely sorted elements (Number of elements VsComparisons)

When elements are in normal sorte order (i.e. not reverse sorted), then this algorithm has time complexity of O(n*n), when elements are in reverse sorted order then this algorithm works in best case.

4. CONCLUSION

The index sorting algorithm sort the elements by creating the index recursively and sort the elements which are in reverse order very efficiently and sort the random input also in an efficient manner. For all random inputs of data the implementation results shows promising results.

REFERENCES

[1] Kruse R., and Ryba A., *Data Structures and Program Design in C++*, Prentice Hall, 1999.

[2] Shahzad B. and Afzal M., “Enhanced ShellSorting Algorithm,” *Computer Journal of Enformatika*, vol. 21, no. 6, pp. 66-70, 2007.

[3] Cormen T., Leiserson C., Rivest R., and Stein C., *Introduction to Algorithms*, McGraw Hill, 2001.

[4] Aho A., Hopcroft J., and Ullman J., *The Design and Analysis of Computer Algorithms*, Addison Wesley, 1974.

[5] Astrachanm O., *Bubble Sort: An Archaeological Algorithmic Analysis*, Duk University, 2003.

[6] Bell D., “The Principles of Sorting,” *Computer Journal of the Association for Computing Machinery*, vol. 1, no. 2, pp. 71-77, 1958.

[7] Box R. and Lacey S., “A Fast Easy Sort,”*Computer Journal of Byte Magazine*, vol. 16,no. 4, pp. 315-315, 1991.

.

[8] Deitel H. and Deitel P., *C++ How to Program*, Prentice Hall, 2001.

[9] Friend E., “Sorting on Electronic ComputerSystems,” *Computer Journal of ACM*, vol. 3,

no. 2, pp. 134-168, 1956.

[10] Knuth D., *The Art of Computer Programming*,Addison Wesley, 1998.

[11] Ledley R., *Programming and Utilizing Digital Computers*, McGraw Hill, 1962.

[12] Levitin A., *Introduction to the Design andAnalysis of Algorithms*, Addison Wesley, 2007.

[13] Nyhoff L., *An Introduction to Data Structures*, Nyhoff Publishers, Amsterdam, 2005.

[14] Organick E., *A FORTRAN Primer*, AddisonWesley, 1963.

[15] Pratt V., *Shellsort and Sorting Networks*,Garland Publishers, 1979.

[16] Sedgewick R., "Analysis of Shellsort andRelated Algorithms," *in Proceedings of the 4th Annual European Symposium on Algorithms*,pp. 1-11, 1996.

[17] Seward H., "Information Sorting in theApplication of Electronic Digital Computers to Business Operations," *Masters Thesis*, 1954.

[18] Shell D., "A High Speed Sorting Procedure,"*Computer Journal of Communications of the ACM*, vol. 2, no. 7, pp. 30-32, 1959.

[19] Thorup M., "Randomized Sorting in O(n log logn) Time and Linear Space Using Addition,Shift, and Bit Wise Boolean Operations,"*Computer Journal of Algorithms*, vol. 42, no. 2,pp. 205-230, 2002.