Average Sort

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

GovardhanaBabuKolli *kgovardhanababu@yahoo.co.in*
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 which runs faster.. This algorithm was analyzed, implemented and tested and the results are promising for a random data

Keywords: Average Sort

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 (Average sort) is proposed; here the basic idea is first we are going to calculate the average of total elements, then we devide the elements into two parts

one part of elements less than the average, and other part contain greater than or equal to the average. This process reapeated until all elements are sorted.

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

2. AVERAGE SORT

2.1 Concept

The main logic presented here is by calculating the average, by comparing the average with each and every element in the array, we are able to arrange the elements like following. All the elements less than or equal to the average are at one (left of the array, i.eleftsubarray) side , and other elements are at other (right of the array, i.erightsubarry) side. Recursively do this for the sub arrays , finally all the elements are in sorted order.

2.2 Pseudocode

```
Function begin(input,size)
1. varavg:=average(input,size)
2.sort(0,size-1,avg)
3. end begin
Function sort(start,end,average)
1.var s:=start,e:=end
2.var average1:=0,average2:=0,sum1:=0,sum2:=0
3.for i:=start to end do
4.If input1[i]>average
5.input2[e--]:=input1[i]
6.sum1:=sum1+input1[i]
7.end if
8.else
9.input2[s++]:=input1[i]
10.sum2:=sum2+input1[i]
11.end else
12.end for
13.if end<=s+1 or e+1<=start
14.return
15.average1:=getaverage(sum1,end-s+1)
16.average2:=getaverage(sum2,e-start+1)
17.copyfrom2to1(start,end)
18.sort(s,end,average1)
19.sort(start,e,average2)
20.return
21.end sort

Function copyfrom2to1(start,end)
1.for i:=start to end do
2.input1[i]:=input2[i]
3.end for
```

4. end copyfrom2to1

Function getaverage(sum,no_of_elements)
1.var average:=sum/no_of_elements
2.returnMath.floor(average)
3.endgetaverage
In function begin,
Line 1, declares a variable avg, calling the function average which calculates the average of the given input array of given size. And the pseudo code for that function is given below.
Function getaverage(sum,no_of_elements)
1.var average:=sum/no_of_elements
2.returnMath.floor(average)
3.endgetaverage
In line2, we called the function, *sort*, thepseudocode for that function is given below.
Function sort(start,end,average)
1.var s:=start,e:=end
2.var average1:=0,average2:=0,sum1:=0,sum2:=0
3.for i:=start to end do
4.If input1[i]>average
5.input2[e--]:=input1[i]
6.sum1:=sum1+input1[i]
7.end if
8.else
9.input2[s++]:=input1[i]
10.sum2:=sum2+input1[i]
11.end else
12.end for
13.if end<=s+1 or e+1<=start
14.return
15.average1:=getaverage(sum1,end-s+1)
16.average2:=getaverage(sum2,e-start+1)
17.copyfrom2to1(start,end)
18.sort(s,end,average1)
19.sort(start,e,average2)
20.return
21.end sort
In the function sort,
This function used to arrange the elements of the given array from the index **start** to index **end** based on the average
In line1 , declaring variables s and e and initialize them to start and end respectively.
In line 2, declaring and initializing the variables average1,average2 to compute average of subarrys , sum1 and sum2 to store the sum of the elements for the subarrays (i.e left subarry and right subarray).

Line 3 to 12 , for comparing the array of elements with the average,and store them in the array input2 (Observe, we are computing sum1,sum2 for the subarrays while doing).
Line 13 to 14 , return call based on the condition end<=s+1 or e+1<=start
Line 15 to 16 , computing average of the two subarrays.
Line 17 , copying the elements that rearranged from array input2 to input1. And the pseudo code is as follows.

```
Function copyfrom2to1(start,end)
1.for i:=start to end do
2.input1[i]:=input2[i]
3.end for
4. end copyfrom2to1
```

Line 18 to 19 , recursively calling the function sort for the array from index **s** to **end** on average **average1** and index **start** to **e** on average **average2**.

```
Function copyfrom2to1(start,end)
1.for i:=start to end do
2.input1[i]:=input2[i]
3.end for
4. end copyfrom2to1
```

3. IMPLEMENTATION RESULTS

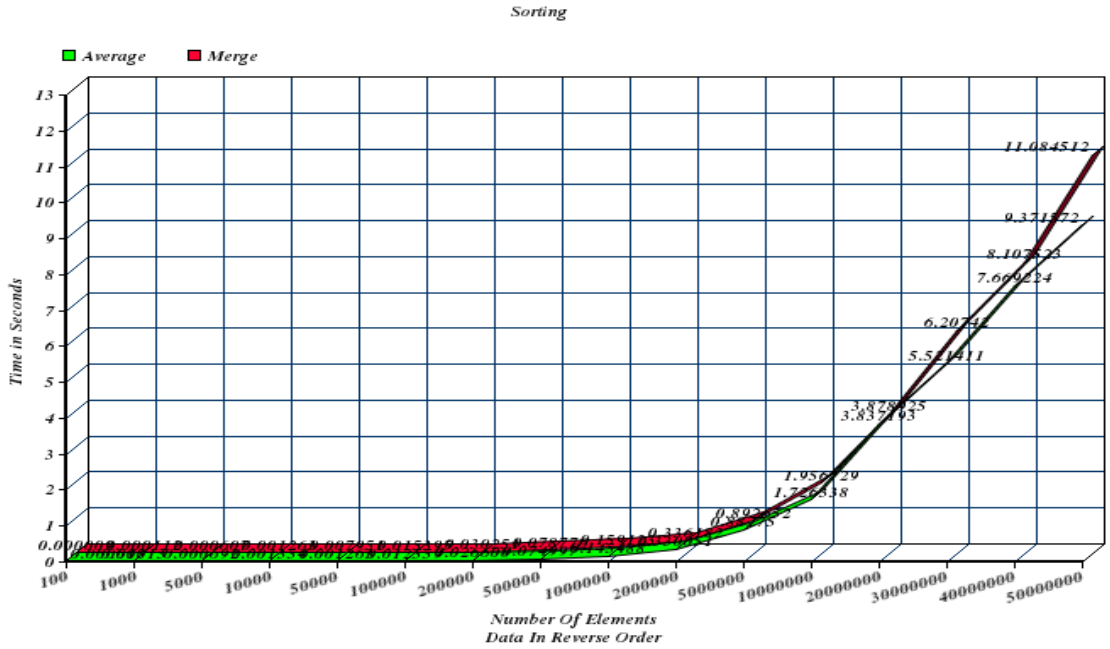

FIGURE 1 : Sorting the elements in reverse order

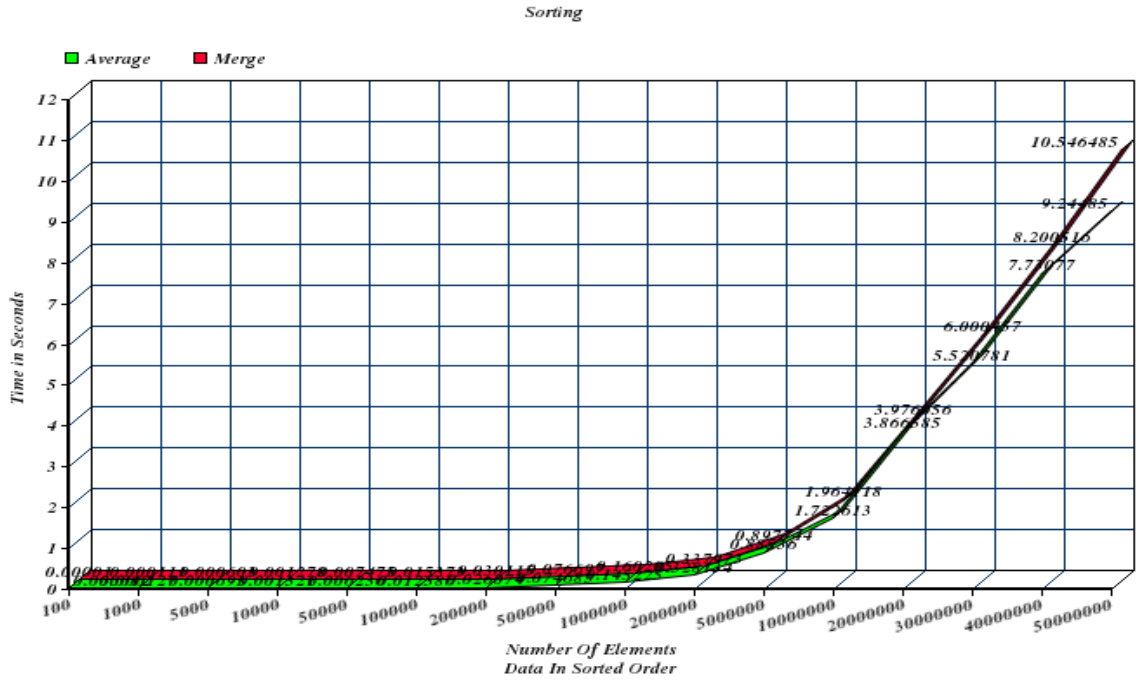

FIGURE 2: Sorting already sorted elements

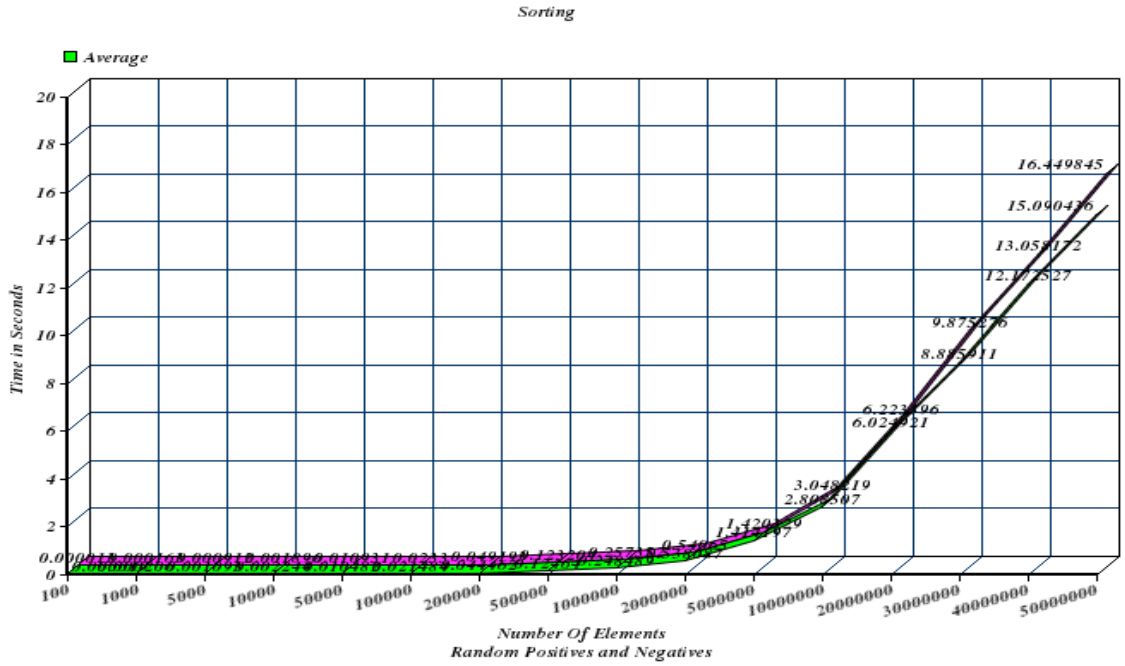

FIGURE 3: Sorting the random integers

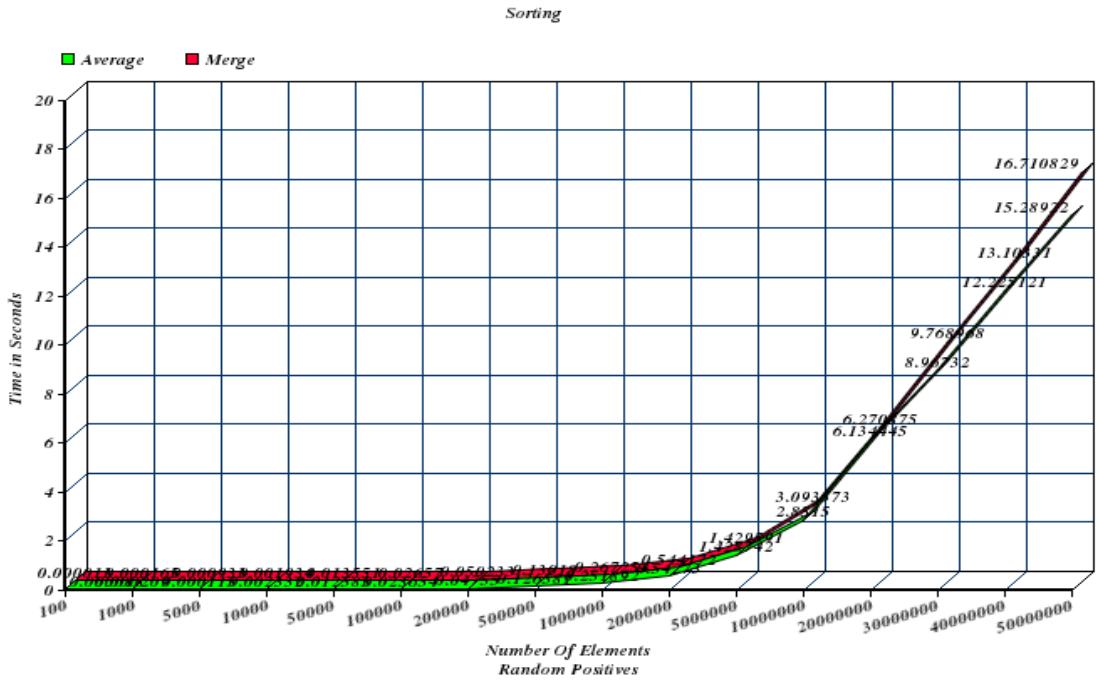

FIGURE 4: Sorting random positives

4. CONCLUSION

This average based sorting algorithm works based on the average of the numbers in the given input. Recursively dealing with the sub arrays for a particular index range and copying the results from the sub arrays to the main array. Finally, the given array will be sorted, with O(nlogn) time.

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.