# Linear Time Sorting Algorithm for Large Integer Data-set

Mr. Sudhir S. Venegurkar
Assistant Professor, Department of MCA,
SIBACA, Lonavala, Pune, India
(sudhir.venegurkar@gmail.com )

**Abstract—Dealing with the data is the basic task of almost every computer application. When we want to manipulate data sets then it is always easy to do that with sorted data sets. Today's computer applications are dealing with big data sets and sorting these big data sets is a tedious job. It will take more computing time even by using best comparison sorting techniques like Merge sort, Quick sort or Heap sort which takes O (n log n) computing time. In this paper, author would like to propose a special type of sorting algorithm 'SLinearSort' which sorts the large integer data sets in linear computing time.**

**Keywords-**algorithm; sorting; linear computing time; SLinearSort;

## I. INTRODUCTION

Sorting is the important and essential operation in the process of manipulating data. Different algorithms are available to sort the data as per the need of the application. A sorting algorithm is an algorithm that puts elements of a list in a certain order. The most-used orders are numerical order and lexicographical order. [4] Most of the efficient algorithms are comparison-based which will take time more than linear computing time. In the present business scenario, systems are automated so large amount of data gets generated. When we want to analyze and manipulate such a big data then efficient sorting method is required which takes less time to perform the sorting task. In this paper 'SLinearSort' method is suggested which sorts data in linear computing time and it is not comparison-based. It uses implicitly the 'MaxMin' algorithm which is based on Divide and Conquer technique and it finds maximum and minimum number of the given list. These Max and Min values of the given data sets are used for sorting.

## II. PREVIOUS ALGORITHMS

Usually when we want to sort elements in required order then we use appropriate sorting method. The proper selection of sorting method is mostly dependent on dataset which we want to sort. If we want to sort datasets having fewer numbers of elements then we can use conventional sorting methods like Bubble sort, Insertion sort and Selection sort. All above conventional methods are easy to implement but not so efficient for large datasets. Sorting has been considered as a fundamental problem in the study of algorithms due to many reasons:

- The need to sort information is inherent in many applications.
- Algorithms often use sorting as a key subroutine.
- In algorithm design there are many essential techniques represented in the body of sorting algorithms.
- Many engineering issues come to the fore when implementing sorting algorithms. [2].

Mostly sorting algorithms are problem-specific, meaning that those work well on some specific problem and do not work well for all the problems. All sorting algorithm are, therefore, appropriate for specific kinds of problems. [1] The efficiency of an algorithm is measured by two basic parameters, one is the total space required for an algorithm to execute, called as Space Complexity and other one is the total processor time required for an algorithm to produce result, called as Time Complexity. As per the present scenario, space is available with cheaper cost so the time complexity is of major concern for execution of an algorithm. To deal with Big Data we need to keep it in order which requires efficient sorting technique in terms of time complexity. Most of the comparison sorting methods like Bubble sort, Insertion sort, Selection sort take O ($n^2$) time to compute result and like Merge sort, Quick sort, Heap sort takes O (n log n) time to compute result. Here in this time complexity specification 'n' is input size i.e. total elements to be sorted. Algorithm having O ($n^2$) time complexity i.e. Quadratic time complexity will take more time as input size increases. Algorithm having O (n log n) time complexity are better than O ($n^2$) but also take much time when input size is large enough. So the algorithms with linear computing time for sorting given data sets are considered as the best one, especially when large datasets need to be sorted. In this paper author is intended to discuss 'SLinearSort' which sorts given integer large data set in linear computing time.

## III. PROPOSED ALGORITHM

The proposed algorithm is a non-comparison sorting algorithm 'SLinearSort', which sorts large integer data set in linear computing time. To sort the given data set it uses 'MaxMin' algorithm implicitly to find max value and min value from given data set. 'MaxMin' is the recursive algorithm which is designed by the use of divide and conquers technique. 'MaxMin' algorithm divides the given array at mid point, makes two identical sub-problems and solves them separately to find Max and Min value. Final Max and Min values are computed by comparing Max and Min values in the sub-problems. According to Min and Max values the Index [Min:Max] array is created which is used to sort the given data set. This Index [Min:Max] array is used to mark the values given in the array which is to be sorted .

The pseudo code of proposed algorithm is as given below:

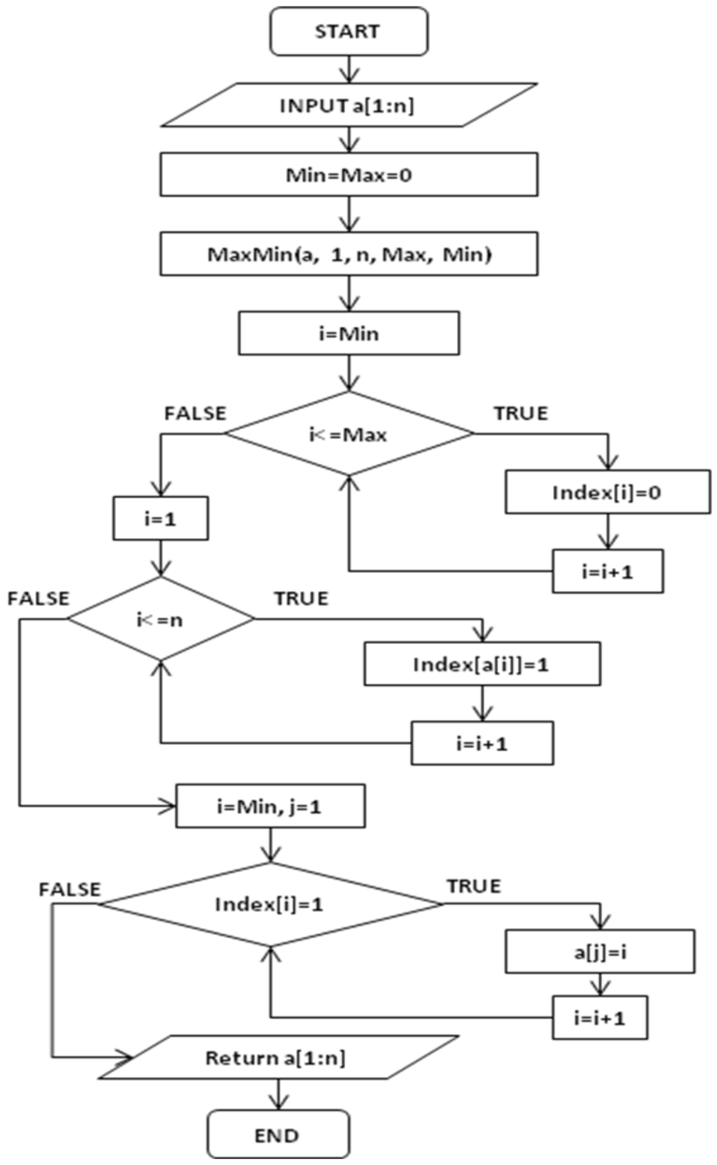
```
Algorithm SLinearSort(a, n)
// a[1:n] is the given array of integer elements having n elements. Variable Max and Min are used to store
// maximum and minimum value from given array. a [1:n]. Algorithm 'MaxMin' is used to find Max and Min.
//Index[Min:Max] is used to mark elements in given array for sorting.
{
    Min:=Max:=0;
    MaxMin(a, 1, n, Max, Min);
    Index[Min: Max];
    for i:=Min to Max do
        Index[i]:=0;
    for i:=1 to n do
        Index[a[i]]:=1;
    j=0;
    for i:=Min to Max do
    {
        if(Index[i]=1) then
        {
            a[j]:= i;
            j:=j+1;
        }
}
return a;
}
```

The above algorithm the unsorted elements in the given array a [1: n] itself. So this is also an in-place sorting method. The pseudo code of 'MaxMin' algorithm is given as above:

```
Algorithm MaxMin(a, i, j, Max, Min)
// a [1: n] is given array of n elements. i and j are counter variables which indicates start and end of list.
// Algorithm set Max and Min value to the largest and smallest value in a [1: n].
{
    if(i=j) then
        Max:=Min:=a[i];
    else if(i=j-1) then
    {
        if(a[i]<a[j]) then
        {
            Max:=a[j];
            Min:=a[i];
        }
        else
        {
            Max:=a[i];
            Min:=a[j];
        }
    }
    else
    {
        mid:= (i+j)/2;
        MaxMin(a, i, mid, Max, Min);
        MaxMin(a, mid+1, j, Max1, Min1);
        if(Max < Max1) then Max:=Max1;
        if(Min > Min1 ) then Min:=Min1;
    }
}  [3]
```

The algorithm ‘SLinearSort’ accepts the given unsorted array of n elements and first it finds maximum and minimum element from that array. To find maximum and minimum element it uses ‘MaxMin’ algorithm which is designed on the basis of divide and conquer technique. Using these Max and Min values Index [Max:Min] array will be created and every element of this array will be initialized to zero. All the elements of the given array are marked in the Index array by assigning that Index array position to one. Finally all the marked positions are transferred in the given array as an element that gives sorted list of given array.

Representation of ‘SLinearSort’ using flow chart:



## IV ANALYSIS OF PROPOSED ALGORITHM

The proposed algorithm ‘SLinearSort’ will sort the given unsorted list of integer elements in linear time. The time complexity of ‘SLinearSort principally depends on the maximum and minimum element of the given array. So for this algorithm the input parameter to compute the time complexity will be difference between Max value and Min value from the given set of elements. The time complexity of an algorithm can be computed by counting number of program steps in the algorithm. A *‘program step’* is loosely defined as a syntactically or semantically meaningful segment of a program that has an execution time that is independent of the instance characteristics. [3] The program steps having execution time will be considered as 1 step count and the program steps having considered as 0 step count. Total step count is calculated as per frequency i.e step per execution. The total program step count of ‘SLinearSort’ is as shown in the table below:

| Algorithm Steps | Step/ execution | Step count |
|---|---|---|
| 1. Algorithm SLinearSort(a, n) | 0 | 0 |
| 2. // a[1:n] is the given array of integer elements having | 0 | 0 |
| 3. // n elements. Variable Max and Min are used to store | 0 | 0 |
| 4. // maximum and minimum value from given array | 0 | 0 |
| 5. // a[1:n]. Algorithm 'MaxMin' is used to find Max | 0 | 0 |
| 6. // and Min. Index[Min:Max] is used to mark | 0 | 0 |
| 7. // elements in given array for sorting. | 0 | 0 |
| 8. { | 0 | 0 |
| 9. Min:=Max:=0 | 1 | 1 |
| 10. MaxMin(a, 1, n, Max, Min); | 1 | (3n/2)-2 [3] |
| 11. Index[Min: Max]; | 1 | 1 |
| 12. for i:=Min to Max do | 1 | k+2 |
| 13. Index[i]:=0; | 1 | k+1 |
| 14. for i:=1 to n do | 1 | n+1 |
| 15. Index[a[i]]:=1; | 1 | n |
| 16. j=0; | 1 | 1 |
| 17. for i:=Min to Max do | 1 | k+2 |
| 18. { | 0 | 0 |
| 19. if(Index[i]=1) then | 1 | k+1 |
| 20. a[j]:= i; | 1 | k+1 |
| 21. } | 0 | 0 |
| 22. return a; | 1 | 1 |
| 23. } | 0 | 0 |

Let the total time required for 'SlinearSort' be T (n) then,

$$T(n) = (7n/2) + 5k + 10$$

Ignoring the additive and multiplicative constants,

$$T(n) = O(n + k)$$

Where n = number of elements in the given array to be sorted

k = difference between maximum and minimum value in the given array

**Best Case Analysis:** In the given instance, difference between maximum and minimum value i.e k=n then the time required for the 'SLinearSort' will be,

$$T(n) = (7n/2) + 5n + 10$$
$$= (17n/2) + 10$$

If we ignore the additive and multiplicative constants then,

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

Thus, for the best case 'SLinearSort' sorts the given array of integer elements in linear time.

**Worst Case Analysis:** In the given instance, difference between maximum and minimum value i.e k>n then 'SLinearSort' may take large time to sort the instance. Suppose if $k=n^2$ then,

$$T(n) = (7n/2) + 5n^2 + 10$$

If we ignore the additive and multiplicative constants then,

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

So the computing time of 'SlinearSort' primarily reliant on value of k i.e difference between maximum and minimum value. When value of k is near to value of n i.e total number of elements then it will compute the answer in linear time.

**Example:** consider an array unsorted integer elements as

a [14]={15, 17, 9, 8, 7, 12, 11, 13, 19, 6, 20, 21, 16, 5}
n=14

Initially algorithm finds Max and Min element. In the given instance Max=21 and Min=5.

Then creates Index [5: 21] having size of 17 elements.

Index [17] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0}

All the elements in the array a [14] are considered as positions of Index [17] array and assigned by 1. So the Index [17] array becomes,

Index [17] = {1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1, 1}

Finally the positions of Index [17] array assigned by 1 are copied in the array a [14] to get final sorted list.

a [15]={5, 6, 7, 8, 9, 11, 12, 13, 15, 16, 17, 19, 20, 21}

For the above example, n=14 and k=16

$$\begin{aligned} T(n) &= (7n/2) + 5k + 10 \\ &= (7*14/2) + 5*16 + 10 \\ &= 139 \end{aligned}$$

Thus the above unsorted array a [14], sorted by 'SLinearSort' and takes around 139 program steps.

If we sort the same array by conventional comparison-based sorting method like bubble sort then it may take approximately O ($n^2$) program steps i.e around 196 program steps. When the numbers of elements are more than that will make this effect very significant.

## V CONCLUSION

Thus the algorithm 'SLinearSort' sorts the large integer data sets in linear computing time. This algorithm is very effective when the unsorted data set to be sorted is large in numbers and the difference between Max value and Min value in the data set is close to the number of elements. Whenever we have a data set which contains values from the specific range then algorithm 'SLinearSort' will be very effective. The limitation of this algorithm is that it will be applicable to the unsorted data sets which contain only distinct integer values.

## VI REFERENCES

[1] Sultanullah Jadoon , Salman Faiz Solehria, Prof. Dr. Salim ur Rehman, Prof. Hamid Jan, "Design and Analysis of Optimized Selection Sort Algorithm," in International Journal of Electric & Computer Sciences IJECS -IJENS Vol: 11 No: 01

[2] Jehad Alnihoud and Rami Mansi, "An Enhancement of Major Sorting Algorithms," in The International Arab Journal of Information Technology, Vol. 7, No. 1, January 2010.

[3] Ellis Horowitz, Sartaj Sahni, Sanguthevar Rajasekaran, "Book of Fundamentals of Computer Algorithms", University Press (India) Limited 2010 edition

[4] http://en.wikipedia.org/wiki/Sorting_algorithm

[5] Davide Pasetto and Albert Akhriev written "A Comparative Study of Parallel Sort Algorithms"

[6] Bressard, "Fundamental of Algorithm." PHI