SSGM Sort Algorithm to Breach Lower Bounds of Comparison Based Sorting

Sunil Gupta[1] , Vinita Gupta[2]
[1]Associate Professor, Dept. of Computer Science and Engineering, Poornima College of Engineering and Research Scholar, School of Computer Science, Jaipur National University, Jaipur, Rajasthan, India
[2]Research Scholar, Dept. of Statistics, University of Rajasthan, Jaipur, Rajasthan, India

***Abstract* —** this work proposes a totally new sorting algorithm named as SSGM Sort. This sorting algorithm not only breaches lower bounds of time complexity but minimizes other requirements also like Space Complexity, Adaptive, Structural Complexity, Processor complexity etc.. SSGM Sorting Algorithm is in-place sequential algorithm which exhibits O(n log n) comparisons in worst case, O(n) comparisons in best case, O(n log √n) in average case which is ultimately breaching the limits of Ω(n log n) explicitly. Other requirements are also minimized as space complexity is O(1) in all the cases. The time complexity of O(n log n) with space complexity of O(1) together is not achieved by any other algorithm. Here SSGM sorting algorithm uses simple array movements with itself, so we need not to maintain any other complicated data structure. SSGM Sorting rather sets an upper bound of O(n log n).

***Keywords—SSGM* Sort, Lower Bound Theorem, Comparison based Sorting, Time Complexity, Space Complexity, Adaptive Sort, Structural Complexity, Bandwith Complexity, Interface Complexity, Stable Sorting.**

I. INTRODUCTION

Since the development of computers, rather since the inception of mathematics sorting has been a major topic of discussion and development. Efforts have been made to reduce resources required to sort a sequence, due to its importance in other data processing components. Sorting is still constitutes major part of complexity of Operating System Routines, Database Management System Procedures, Transactions and Processes, Networking Routines and Processes, Machine Learning Artificial Intelligence. Due to the importance of Sorting Algorithms here is an effort to further reduce the complexities of the Sorting with the introduction of SSGM Sorting method. Here not only proposed a method but presented a comparative analysis of SSGM method with other sorting algorithms which were considered optimized so far. This work further extended efforts to implement SSGM Sort in C++ as part of this article.

II. METHODOLOGY AND EXPERIMENTAL SETUP

The Proposed Sorting Algorithm SSGM_Sort is proved using asymptotic complexity analysis. Thus the complexity achieved for SSGM_Sort is then tested against Modified Lower Bound theorem. Thus SSGM_Sort has been found the most optimized one. It has even been found better than Merge and Heap Sort.

III. A COMPARATIVE ANALYTICAL REVIEW OF SEQUENTIAL SORTING ALGORITHMS AND SSGM SORT

Lower Bound theorem states that "Any Comparison based Sorting algorithm will take at least Ω (n log n) Comparisons". Merge Sort and Heap Sort are found to be optimized one based on this theorem[1][2][3][4][5] but here proposed SSGM Sort which is even more efficient then lower bounds. Following table shows a comparative analysis of sorting algorithms [6]. This table clearly reflects that SSGM Sort is the optimized most. It achieves the lower bounds without any trade off at the cost of memory, processor, structure on anything else. SSGM_Sort even found the most optimized one which is fit for even Modified Lower Bound Theorem [1].

Algorithm	Worst Case Time Complexity	Worst Case Space Complexity	Adaptive	Structural Complexity	Processor Complexity	Tradeoff
Bubble Sort	$O(n^2)$	O(1)	No	Array	O(1)	Already Poor time complexity
Insert Sort[	$O(n^2)$	O(n)	Yes	Array	O(1)	Already Poor time complexity
Selection Sort	$O(n^2)$	O(1)	No	Link List	O(1)	Already Poor time complexity
Quick Sort	$O(n^2)$	O(n)	No	Array	O(1)	Already Poor time complexity
Merge Sort[7][8]	Θ (n log n)	O(n)	No	Array	O(1)	Trade off is Space Complexity
Heap	Ω (n log n)	O(1)	No	Heap	O(1)	Trade off is

Sort[9]						Structural Complexity
Counting Sort[2]	O(n+k) If k>>n then O(n+k)>O(n^2) or even higher.	O(k)	No	Array	O(1)	For k<=n and for integers only. Not Generalized for others.
SSGM Sort (Proposed in this Paper)	O(n log n) Breaching lower bound of Ω (n log n)	O(1)	Yes	Array	O(1)	Achieved Lower Bound Without and Tradeoff
Remarks	SSGM is best in this segment.	SSGM is best in this segment.	SSGM is best in this segment.	SSGM is best in this segment.	SSGM is best in this segment.	SSGM is a new breakthrough in comparison based sorting.

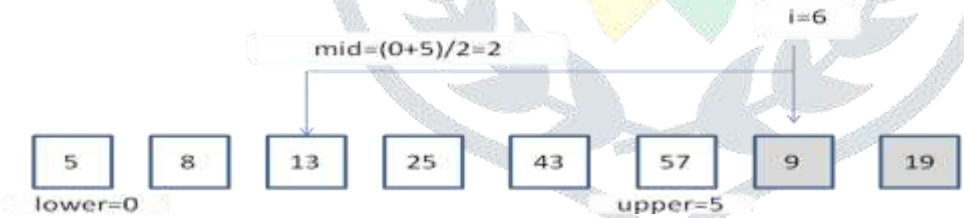
Table 1: Comparative Analysis of SSGM Sort with other Sorting methods

Although there are GPU, Co-Processor and multi processor based algorithms [10][11] but these are out of scope of this work, as here in this work sequential comparisons are analyzed. We always have opportunity to make an parallel version of these algorithms more time saving and use it implement it even to solve larger problems such as image analysis and medical diagnosis[12][13]. SSGM Sort rather adheres to Modified Lower Bound theorem which also addresses other key issues of sorting [1]

IV. THE SSGM SORT: A VISUAL DEMO OF SSGM SORT

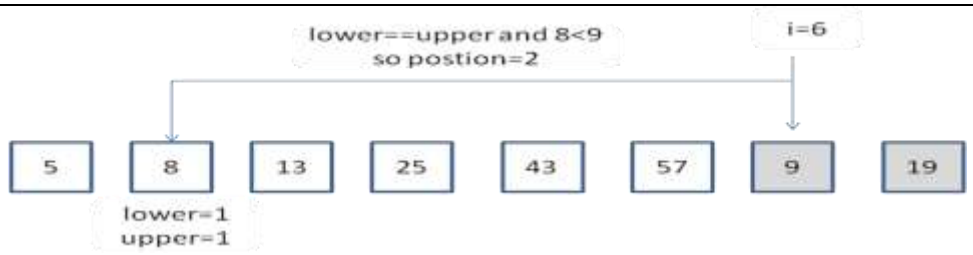
SSGM Sort is based on the principle that an array which has n elements is partly sorted starting from first element to (i-1)th element. So to sort ith element FindPosition() will take only O(log i-1) time to discover new position of ith element using modified binary search. Here binary search is modified in such a way so that it returns the expected position of the key which is being searched. If binary search founds equal value then it returns same position immediately. Thus in worst case where all the values are unique this modified binary search will take O(log n) time and in best case where large number of duplicate values exists it takes O(1) time. Thus overall in place SSGM Sort will take at most O(n log n) time in worst case and O(n) time in best case. Following table demonstrates how the SSGM Sort, Sorts the sequence. To demonstrate it is trying to show that how the index position 6th is sorted once the previous positions are sorted.

1. First Comparison 0 to i-1 (That is 0 to 5) initially lower=0 and upper=5

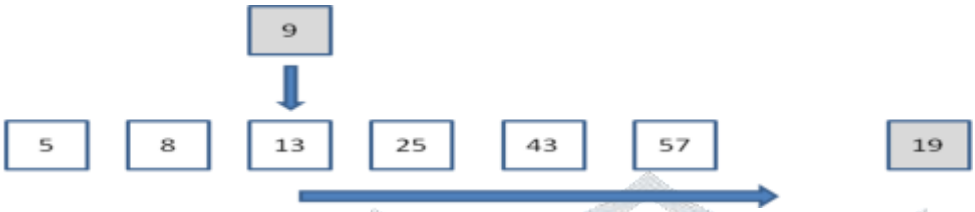


2. Second Comparison from lower (that is 0) to upper (that is 1). upper shifted to mid-1 (that is 2-1=1)

3. Third Comparison lower to upper (that 0 to 0) as upper shifted further to mid-1. So upper=1- 1=0.



4. Now value 9 will be inserted at index position 2.



5. Values after index position will be shifted using a series of swaps till ith position.

6. Finally 7th element (i = 6) is sorted.

Table 2: Demo of SSGM Sort to sort element at 6th index position (7th element)

V. THE SSGM SORT: AN IMPLEMENTATION OF SSGM SORT IN C++ PROGRAMMING LANGUAGE

Following is the code in C++ to implement SSGM Sort. Here the main() is also given to demonstrate the call to SSGM Sort. The way here SSGM Sort implemented is given such that it can be used as library function.

Algorithm 1: C++ Code to implement SSGM Sort

Input: An Array of size n which has unsorted values.
Output: Array has sorted sequence

```
1. #include<iostream.h>
2. #include<conio.h>
3. int FindPosition(int key, int *a, int lower, int upper); //finds position of key in array a[] from a[lower]
                                                            //to a[upper] using Modified Binary Search
4. void placement(int i, int position, int *a);            //simply inserts ith element at position in array a[]
5. void SSGMSort(int *a, int size);                        //Sorting module
6. void main(){                                            //To demonstrate main() calls SSGMSort()
7.      clrscr();
8.      int a[]={13,200,135,4,350,260,170,580,18,27,437,2000,910,31000,111,5298};
9.      int i,size=16;
```

```
10.     SSGMSort(a,16);
11.     for(i=0;i<size;i++){
12.             cout<<a[i]<<endl;
13.     }
14.     cout<<count;
15. }
16. void SSGMSort(int *a, int size){
17.     int position,i;
18.     for(i=1;i<size;i++){
19.         position=FindPosition(a[i],a,0,i-1);      // find position of ith element in array which is
                                                      //already sorted from 0 to i-1
20.                                                   //takes O(log i-1) Comparisons and O(1) space.
21.         placement(i,position,a);                  //place ith element at positionth position in array a[ ]
22.     }
23. }
24. int FindPosition(int key, int *a, int lower, int upper){
25.         int mid, position;
26.         while(lower!=upper){
27.             mid=(lower+upper)/2;
28.             if(a[mid]==key)
29.                 return mid;
30.             else if(a[mid]>key)
31.                 upper= mid-1;
32.             else
33.                 lower=mid+1;
34.         }
35.         if(a[lower]<key) return lower+1;
36.         else return lower;
37. }
38. void placement(int i, int position, int *a){
39.         int j,temp;
40.         for(j=position;j<i;j++){
41.             temp=a[i];
42.             a[i]=a[j];
43.             a[j]=temp;
44.         }
45. }
```

VI. RESULTS: SSGM SORT PERFORMANCE ANALYSIS

Here SSGM Sort is analyzed based on the standard Mathematical Proof for sorting algorithm analysis based on various parameters as given here:

(A) Space Complexity: As shown in the C++ implementation it is clear that SSGM Sort is in place sorting method which takes O(1) space. It is best among all those sorting methods which are considered optimized one and achieved lower bounds for example Merge Sort and Heap Sort. Merge Sort and Heap Sort does give lower bounds but at the cost of space and structural complexity.

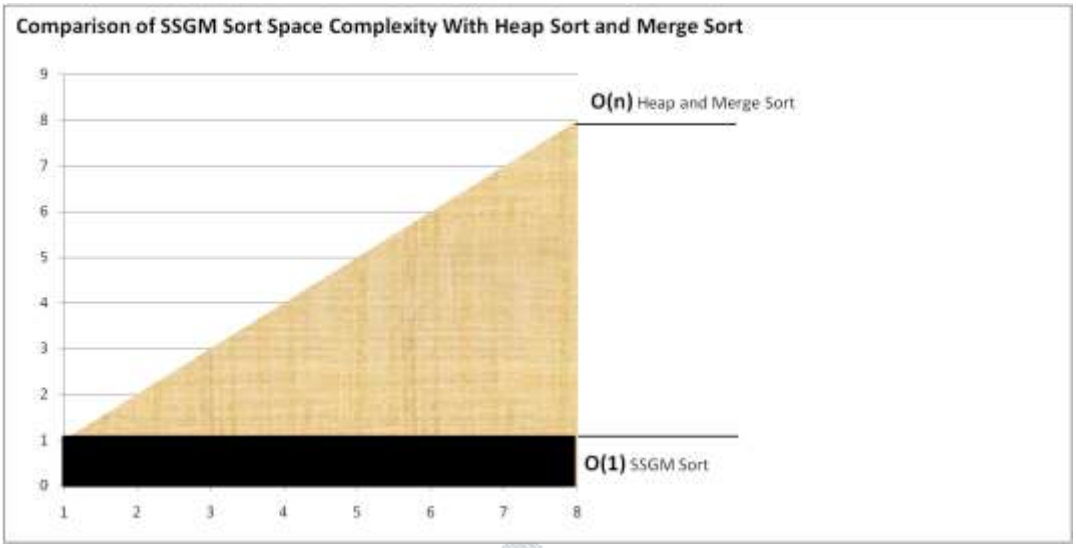


Figure 1: Comparison of Space complexity of SSGM Sort with Heap Sort and Merge Sort.

(B) Time Complexity: As mathematical analysis of C++ implementation of SSGM Sort states that there are two parts in analysis. First Binary Search based position search that is already proved to take O(log n) time [1] and this process is repeated i number of times. Son following are the observations:

a. Worst case complexity is O(n log n) which is better than lower bound of comparison based sorting algorithms that is Ω (n log n). Here worst case is a sequence with unique values only.

b. Best case time Complexity of SSGM Sort is O(n) which lower than even Merge Sort and Heap Sort. The Best Case for SSGM Sort is a sequence which has large number of duplicate values as it happens in large data sets. Here Best case is a sequence with sparsely speeded duplicate values.

c. Average Case complexity of SSGM Sort is O(n log √n) which is far better than Merge Sort and Heap Sort.

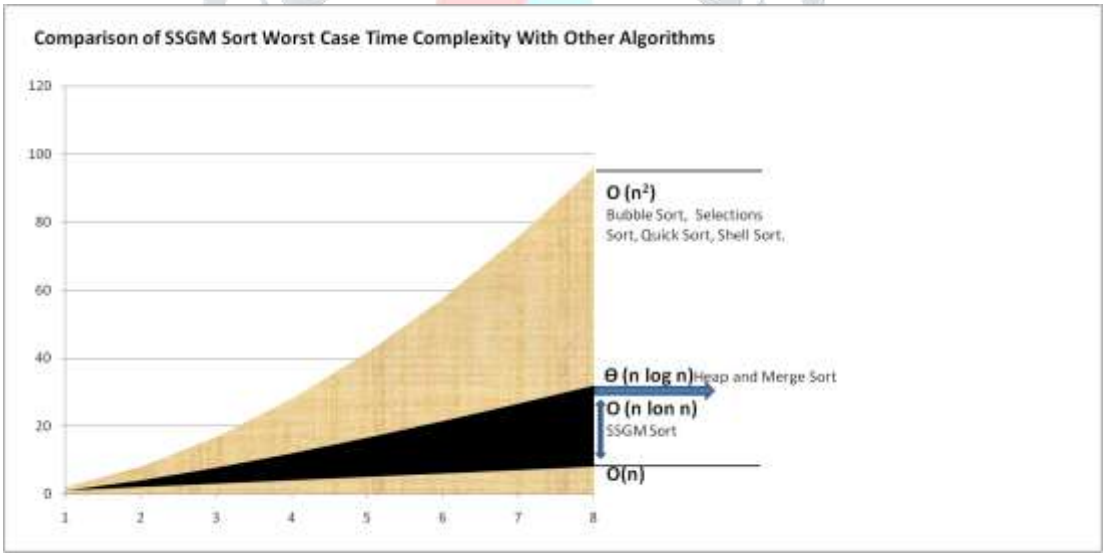


Figure 2: Comparison of Time Complexity of SSGM Sort with Other Algorithms

(C) Adaptive: SSGM Sort is adaptive as the new element can be inserted without reinitializing the sorting process from beginning. The new element just need to find out its position and get inserted.

(D) Structural Complexity: it is an in place array based sorting method. It does not involve any other complicated data structure maintenance.

VII. CONCLUSION

The SSGM Sort is fastest and Space efficient most sorting algorithm. Its time complexity is O(n log n) in worst case and O(1) space. This much efficacy is not even achieved by Merge Sort or Heap Sort. It is very simple and straight also and does not has any hidden complexity due to underlying data structure. SSGM sort has been proved to be adaptive also. Thus SSGM Sort is the optimized most sequential comparison based sorting algorithm so far.

VIII. FUTURE SCOPE

To implement SSGM Sort in many other languages so that advantages of SSGM Sort can be utilized in other paradigms and applications also. Further a dynamic SSGM Sort version will also be worked upon so that it will be easy to apply in all the cases. SSGM Sort can also be extended for nonnumeric object oriented comparisons.

References

[1] Sunil Gupta, Prof. (Dr.) Prashant Sahai Saxena, "Modified Lower Bound Theorem For Comparison Based Sorting Algotithms", IJRAR - International Journal of Research and Analytical Reviews (IJRAR), E-ISSN 2348-1269, P- ISSN 2349-5138, Volume.5, Issue 4, Page No pp.740-743, October 2018, Available at : http://www.ijrar.org/papers/IJRAR1904213.pdf

[2] "Introduction to Algorithms" Thomas H. Cormen, Charles E. Leiserson, Ronald L. Reivest, Clifford Stein, PHI, 3rd Edition ISBN-978-81-203-4007-7.

[3] "Design and Analysis of Computer Algorithms" Alfred V.Aho, John E.Hopcroft, Jeffrey D. Ullman, Pearson, ISBN-8178081032 (ISBN13: 9788178081038)

[4] "Fundamentals of Computer Algorithms" Sanguthevar Rajasekaran, Sartaj Sahni, Ellis Horowitz, Orient BlackSwan, 2ND Edition, ISBN: 9788173716126, 8173716129

[5] Ben-Or M. "Lower Bounds For Algebraic Computation Trees", Preliminary Report 1983 ACM 0-89791-099-0/83/004/0080 Pg(80-86)

[6] Ali K. "A Comparative Study of Well Known Sorting Algorithms" International Journal of Advanced Research in Computer Science ISSN No. 0976-5697 Volume 8, No. 1, Jan-Feb 2017 Pg.277-280

[7] Marszałek Z. "Parallelization of Modified Merge Sort Algorithm" Symmetry 2017, 9, 176; doi:10.3390/sym9090176

[8] Marszałek Z. "Performance Tests On Merge Sort and Recursive Merge Sort For Big Data Processing" UWM Technical Sciences 2018, 21(1), 19–35, ISSN 1505-4675, e-ISSN 2083-4527

[9] Marszałek Z. "Performance Test On Triple Heap Sort" UWM Technical Sciences 2017, 20(1), 49–61, ISSN 1505-4675, e-ISSN 2083-4527

[10] Ye1 Y., Du1 Z., Bader D.A., Yang Q. and Huo W. "GPUMemSort: A High Performance Graphics Co-processors Sorting Algorithm for Large Scale In-Memory Data" GSTF IJOC,Vol.1,No.2, Feb2011 DOI: 10.5176_2010-2283_1.2.34

[11] Satish N., Harris M., Garland M., "Designing Efficient Sorting Algorithms for Manycore GPUs" IEEE:978-1-4244-3750-4/09/2009

[12] Gupta S., Kumar D., "Pattern Classification of Breast Cancer Patients for Personalized Medical Diagnosis" IJLTET – International Journal of Latest Trends in Engineering and Technology" Volume 11 Issue 3 – September 2018 e-ISSN : 2278-621X p-ISSN : 2319-3778 DOI: 10.21172

[13] Sharma V., Gupta S., " Advanced Imaging for Personalized Medical Diagnosis" IJSRD - International Journal for Scientific Research & Development| Vol. 3, Issue 01, 2015 | ISSN (online): 2321-0613