Mark-Set{} Sort

An Eidetic Sorting Technique

Shagun Bhardwaj
Amity University
Noida, India
E-mail: shagun008@gmail.com

***Abstract*— Data arranged in a particular fashion creates information. To arrange data several techniques are used amongst which sorting is a popular one. In this research study, I am proposing a new sorting algorithm. Complexity analysis and the results obtained after implementation are described in a graphical form with an objective to compare the efficiency of the proposed algorithm with standard algorithm technique.**

Keywords- Sorting, Bi-directional sort, complexity analysis.

I. INTRODUCTION

Computation has become a major power in today's environment from medicine to space technology everything relies on computation of data and extracting result from it that can benefit in several ways. Algorithm is a sequence of steps to solve computational problems [1].

Sorting is ordering of data in ascending and descending order. Data can be in numerical or character form.

The study in hand proposes a new sorting technique that is tested and analysed against previous sorting techniques to prove its efficiency. To search the information efficiently, arrangement of data is very important. Sorting is the rearrangement of the given items on the basis of some well-defined ordering rules [2].

The ultimate intention of sorting techniques is reduction in cost and complexity of the algorithms. In this research paper, I propose a new sorting technique; derive its algorithm and compare it with well-known standard techniques.

This sort is way more efficient than bubble sort and it also proves to be more efficient than insertion sort. All the analysis and graphs have been provided for the researchers and scientists to check its efficiency.

II. A BRIEF REVIEW OF EXISTING SORTING ALGORITHMS

This section will briefly discuss some standard sorting techniques. These are following:

A. Bubble Sort

This is one of the simplest and popular algorithms. It compares two elements from the list at a time and swaps them if required; this process is repeated till the list is fully sorted. The name of this algorithm comes from the way the smaller elements bubbles to the top of the list. For the list of 100 items, bubble sort makes 10000 comparisons to sort, this makes it inefficient as compared to today's faster sorting algorithms.

B. Insertion Sort

Another renowned technique of sorting is insertion sort. Insertion sort, an efficient sorting technique, selects one element in every pass and inserts it to its correct location in the list. It is highly efficient on small lists and is very simple and easy to implement. The insertion sort algorithm is a very slow algorithm when list is very large [3].

III MARK-SET{} SORT

Mark-Set{} sort is based on the bidirectional sorting technique. Assuming, we are sorting the list in the ascending order.

It takes benefit of the fact that after first forward iteration of the list, the largest element comes to its correct position at the end of the list.

Similarly, the first reverse iteration of the list sorts the smallest element to its correct position in the beginning of the list.

Another point to be noted here is that during forward and reverse iterations more than one element may come to their correct positions in the list.

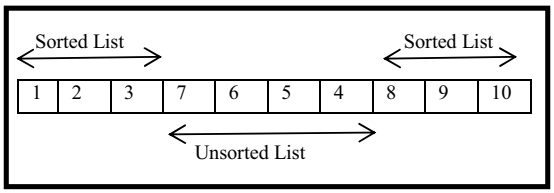
So next time when the list is iterated, it should be able skip the already sorted list and only iterate over the unsorted elements in order to sort them.

This approach is the basis of Mark-Set{} sort. This algorithm remembers the position where last swap of element took place in the list (both forward and reverse iterations respectively).This is useful because during forward iteration if we remember the last element where the swap took place, then we know that after point, the list is sorted. Similarly, in case of reverse sort if we remember the last element where the swap took place, then we know that after point, the list is sorted.

All this routine will save time in the next iterations of the list. Now, the whole list need not be iterated over and over again. Only the unsorted elements can be iterated and sorted as displayed in the Fig 1.

978-0-7695-4941-5/13 $26.00 © 2013 IEEE
DOI 10.1109/ACCT.2013.40

FIG 1. MARK-SET{} SORT



IV ALGORITHUM: PSEUDO CODE

```
MARK_SET_SORT (LIST)
FIRST= 0;
LAST= [LENGTH (LIST) – 1];
SWAPPED=FALSE;
do
  // Forward sort.
   For each i in FIRST to (LAST-1)
      If LIST [i] > LIST [i+1]
           SWAP (LIST [i], LIST [i+1])
           SWAPPED=TRUE;
           LAST=i;
       End if
   End for
   If SWAPPED==FALSE
      BREAK;
   End if

   // Reverse Sort.
   For each i in (LAST-1) to FIRST
      If LIST [i] > LIST [i+1]
           SWAP (LIST [i], LIST [i+1]);
           FIRST=j;
     End if;
   End for;
while SWAPPED==TRUE;
```

V. COMPLEXITY ANALYSIS

The algorithm depicts that there is an outer main loop within which there are two parallel inner oops. So the worst case running cost of algorithm is calculated to be $O(n^2)$.

The behaviour of the algorithm in the best case will be $O(n)$, depicting that the elements in the list are in sorted form.

Similarly the average case of the running cost will be $O(n)$ depending upon the elements in the list.

There lies a significant difference between the average case running cost of Mark-Set{}Sort and the other prevalent algorithms that cannot be overlooked. The results are depicted in Table 1.

TABLE 1. COMPARISON OF COMPLEXITY FOR DIFFERENT SORTING TECHNIQUES

Algorithm	Best Case	Average Case	Best Case
Bubble Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Insertion Sort	$O(n)$	$O(n^2)$	$O(n^2)$
Mark-Set{} Sort	$O(n)$	$O(n)$	$O(n^2)$

VI. COMPARISON OF MARK-SET{} SORT WITH EXISTING SORTING ALGORITHM

To estimate efficiency of Mark-Set{} sort performance, it was implemented along with existing famous sorting algorithms. By taking lists of various sizes, items were generated using a uniform random number generator with values ranging from 1 to 9, 00,000. Running time in nanoseconds was noted. The results are given below:-

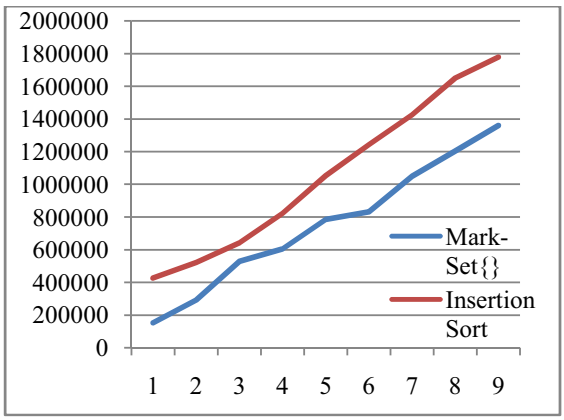
TABLE 2. COMPARISON OF RUNNING TIME FOR DIFFERENT SORTING TECHNIQUES

Sample	Bubble Sort	Insertion Sort	Mark-Set{} Sort
100000	2.66E+10	426120	263139
200000	1.06354E+11	521198	293703
300000	2.3369E+11	641682	529666
400000	4.15776E+11	822215	605113
500000	6.68018E+11	1053559	784491
600000	9.15257E+11	1243715	831068
700000	1.27187E+12	1425404	1049710
800000	1.72534E+12	1651359	1202912
900000	2.07673E+12	1779156	1359196

(Results calculated on Intel Core i5-480M having 2 Cores, 4 logical processors, 2.66 GHz, 4GB RAM on x-64 based system)

Based on data in Table 2, the Graph 1 demonstrates a significant improvement of Mark-Set{} over Insertion sort algorithm.

GRAPH 1. MARK-SET{} VS INSERTIOIN SORT



VII. CONCLUSIONS

By analysing the table and graph above, it can be easily deduces that Mark-Set{} Sort is clearly efficient than well-known standard sorts like Bubble sort and Insertion sort. It can also be deduced from the table (Table 1.0) that the rate of growth of Mark-Set{} sort over large input sizes is very steady as compared to Bubble Sort and Insertion Sort.

One more important point to note is that for partially sorted list this algorithm proves to be even more efficient.
In the future, I will try to come up with new sorting techniques, which hopefully will be more efficient.
Overall this analysis shows that Mark-Set{} sort is very efficient for large number of items in a list, so it will show high efficiency for large list of greater than to 1,00,000 elements.

ACKNOWLEDGMENT

I acknowledge the support and contribution from my family and friends in this research, which encourages and motivates me to continue my efforts for the benefit and ease of mankind. Thank you all.

REFERENCES

[1] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. Introduction to Algorithms, Second Edition. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7. Section 1.1: Algorithms, pp.5–6. Programmer, 2nd ed. McGraw-Hill, 2003.

[2] R. Sedgewick, Algorithms in C++. Reading, Massachusetts: Addison- Wesley, 1992.

[3] Seymour Lipschutz. Theory and Problems of Data Structures, Schaum's Outline Series: International Edition, McGraw- Hill, 1986. ISBN 0-07-099130-8., pp. 322–323, of Section 9.3: Insertion Sort.