IJCSN - International Journal of Computer Science and Network, Volume 3, Issue 6, December 2014
ISSN (Online): 2277-5420 **www.IJCSN.org**
Impact Factor: 0.274

An Unique Sorting Algorithm With Linear Time Complexity

[1] Sanjib Palui, [2] Somsubhra Gupta

[1] Computer Vision & Pattern Recognition (CVPR),
Indian Statistical Institute, Kolkata, India,

[2] Department of Information Technology,
JIS College of Engineering, Kalyani, India

Abstract - As volume of information is growing up day by day in the world around us and prerequisite of some optimized operations is sorted list, the efficient and cost effective sorting algorithm is required. There are several number of sorting algorithms but still now this field has attracted a great deal of research, perhaps due to the complexity of solving it efficiently despite of its simple and familiar statements. An algorithm is chosen according to one's need with respect to space complexity and time complexity. Now days, space is available comparatively in cheap cost. So, time complexity is a major issue for an algorithm. Here, the presented approach is to achieve linear time complexity using divide and conquer rule by partitioning a problem into n (input size) sub problem, then these sub problems are solved recursively. So, asymptotic efficiency of this algorithm is very high with respect to time.

Keywords - *sorting, searching, divide and conquer, algorithm, asymptotic efficiency, space complexity, time complexity, recursion.*

1. Introduction

An algorithm [2] [6] [7] is a finite set of instruction, that if followed, accomplish a particular task. Algorithm must satisfy some characteristic like having input, output, definiteness, finiteness, effectiveness. Sorting [1] [5] [10] means a particular permutation of a given sequence in which order the elements of the sequence are in increasing/decreasing order. Sorting algorithms used in computer science are often classified by:

- Computational complexity [5] [8] [9] (worst, average and best behavior) of element comparisons in terms of the size of the list.
- Computational complexity of swaps (for "in place" algorithms) are sometimes characterized in terms of the performances that the algorithms yield and the amount of time that the algorithms take.
- Usage of memory and other computer resources.
- Recursion [11]. Some algorithms are either recursive or non-recursive, while others may be both (e.g., merge sort).
- Stability: stable sorting algorithms maintain the relative order of records with equal keys (i.e., values).
- Whether or not they are a comparison sort. A comparison sort examines the data only by comparing two elements with a comparison operator.
- General method: insertion, exchange, selection, merging, *etc*. Exchange sorts include bubble sort and quick sort. Selection sorts include shaker sort and heap sort.
- Adaptability: Whether or not the pre-sortedness of the input affects the running time. Algorithms that take this into account are known to be adaptive.

Now a day there are many effective sorting algorithms. Although lots of researcher is working on this, but unlike other field of research, number of proposed new, innovative and cost effective work is very few in the field of sorting algorithm.

We have designed and applied one sorting algorithm to achieve linear time complexity. In this paper, this new algorithm is proposed. As compared to existing algorithm, it gives better result also it has linear time complexity, we named this algorithm as --"**A unique Sorting Algorithm with Linear Time Complexity**"
Here, the presented work is organized as follows: Previous Related Works is given in section 2, Algorithm of Propose Work is in section 3, Analysis of the Algorithm is in section 4, and finally Conclusion is stated on section 5 and then REFERENCES.

2. Previous Related Works

Some well known sorting algorithm are bubble sort, selection sort, insertion sort ,merge sort, quick sort ,heap sort ,radix sort , cocktail sort ,shell sort etc. All these algorithms can be classified according to their average case and worst case time complexity (asymptotic complexity-big theta notation-θ and big oh notation – O respectively). Here time complexity of some algorithms are given along with they are stable or not.

Table 1: Complexities and stability of some sorting algorithms

Name of the algorithm	***Average case time complexity***	***Worst case time complexity***	***Stable?***
Bubble sort	$\Theta(n^2)$	$O(n^2)$	Yes
Selection sort	$\Theta(n^2)$	$O(n^2)$	No
Insertion sort	$\Theta(n^2)$	$O(n^2)$	Yes
Merge sort	$\Theta(n \log_2 n)$	$O(n \log_2 n)$	Yes
Quick sort	$\Theta(n \log_2 n)$	$O(n^2)$	No
Bucket sort	$\Theta(d(n+k))$	$O(n^2)$	Yes
Heap sort	$\Theta(n \log_2 n)$	$O(n \log_2 n)$	No

Here *n* is the input size of the list to be sorted, *d* is number of digit in the largest number among inputs and *k* is all possible digit/word (for example- *k*=10 as decimal). Often, some of work have done in this field by comparative study [13] [14] of the above mentioned (in the table 1) algorithms and depend on application or data structure, which algorithm among these works better or modified [15] [16] existing algorithm which has less complexity than original one. Depending on inputs and some conditions, there are some new proposed algorithms [16] [17] [18] to achieve better complexity. Some algorithm is designed for linear complexity [19] [20] but they can be executed with some characteristic of the system platform as these demand for that. Poll sort, simple pancake sort, bead sort are the example of this type. Pancake sort is not stable but has linear time complexity. Bead sort requires special hardware design. Poll sort takes linear-time to execute but it is an analog algorithm for sorting a sequence of items, requiring O(*n*) stack space, and the sort is stable.

This requires *n (*number of elements in input list*)* parallel processors. There is an algorithm called Randomized Select [20] [21] algorithm for sorting. The expected running time of this algorithm is Θ(*n*), a linear asymptotic running time where *n is* the input size of the problem. This algorithm works like Randomized Quick-sort [22] (where *pivot* element is select randomly from the list).Two main constraint of this algorithm

- All the elements in the input sub-problem are distinct.
- Partition is done based on random selection of an element.

3. Algorithm for Proposed Work

The pseudocode of our work, --“ **A unique Sorting Algorithm with Linear Time Complexity**” is given below –

Sort (A, lb, ub)

Here *A* is a 1-D input list of decimal integer with *n* elements in memory where *n*=*ub*-*lb*+1 and *ub*=upper bound, *lb*=lower bound of the list. *B*[*n*] [*n*], 2-D array of decimal integer is used for calculation.

```
Begin
  1.  if (lb<ub) then,
  2.     find min and max
  3.     if (min!=max) then,
  4.        set n = ub-lb+1
  5.        set div =min+ (max - min) /n
  6.        for i=0 to n -1 by 1do,
  7.                set B[i][0]=0
  8.        end for
  9.        for i=0 to n -1 by 1do,
  10.               set j= A[i]/div
  11.               set k= ++ B[j][0]
  12.               set B[j][k]= A[i]
  13.       end for
  14.       set k=lb
  15.       for i=0 to n -1 by 1do,
  16.               set j = B[i][0]
  17.               set l=k
  18.               while j>0, do
  19.                  A(k++)= B[i][j--]
  20.               end while
  21.               if ((l<k-1) && (!(lb==l&&k==ub)))
  22.                  Sort (A, l, k-1)
  23.            end if
  24.        end for
  25.       end if
  26.  end if
End
```

Algorithm 1: proposed algorithm

min and *max* in line number2 are minimum and maximum number from the list respectively. We have designed here one variable for every “*sort(x , y, z)*” function call, called *div* which *plays major role* in the above mentioned pseudo code of our new sorting

algorithm. Every element of the input list forms its own index as it will be in the output list with the help of *div* variable[line number 9].Then all the elements of the input list with same indexes (as it is generated in line number 9) are solved as sub problems[12]. Elements with same index are treated here as one sub problem. For increasing order output----

- Every sub problem (except left most) as single unit has all smaller elements from the input list in its left.
- Every sub problem (except right most) as single unit has all larger elements from the input list in its right.
- Every sub problem can have 0 to n-1 elements.
- If number of elements in one sub problem is one it takes unit time cost to sort.
- Ideally list with n elements are partitioned in to n sub problems each of which have one element.
- It takes linear time complexity, O(n).
- Relative positions of two elements having same value are not changed, so this algorithm is stable.

All other strategies that are followed to execute this algorithm properly and efficiently-

- If the input list is combination of positive and negative integer, at list are formed one for negative numbers and one for positive number. Then all the elements of the list of negative numbers are made positive. After performing operations to make them as sorted, sign of elements are changed again and the list is reversed. Then operations are performed on positive list. At last, two lists are added.
- If the numbers are real, this algorithm can be applied easily. Problem is divided into sub problems according to their absolute value and if absolute values are same for all elements then partition is done based on their precision value.

4. Analysis of the Algorithm

From the number statistics if the numbers are in uniform distribution then almost no recursions are happened, otherwise after first partition of the array, it will make greater than or equal to two uniform distributed array where complexity is linearly dependant on n. Here, after partition of the array some variables are taken place in the array like in quick sort, one element is fixed in exact position. Here number of fixed elements is 1 to n where n is the number of elements to be sorted. Asymptotic time complexities [3] [4] of this algorithm in different case are described here-

4.1 Best Case

If the elements are uniform distributed, then ideally problems are divided in to n sub problems. So no recursive call is executed because almost each sub problem has one element. So, the required time is

$T(n) = c + T(1)+T(1)+T(1)+T(1)$……n times

$\equiv c + O(n)$ where c=constant time

$\equiv O(n)$ as n is very large

4.2 Average Case

If number of elements, n become very high ,in first function call divided sub-problems will be almost uniform distributed .Let , here recursive call is happened m times where ($m<n$) and sub problems have $n_1, n_2, \ldots\ldots n_m$ elements respectively.

So, $n_1, n_2, \ldots\ldots n_m + p = n$ ---- Eq. (1)

where p is the number of elements that are already taken place in the input list as sorted elements. So, time complexity is

$T(n)=c\ .n+ T(n_1)+T(n_2)+\ldots\ldots+T(n_m)$ where c=constant time

$=c.\ n+ c.\ n_1+c.\ n_2+ \ldots\ldots+c.\ n_m$

[From the best case as the sub problems are uniformly distributed into $n_1, n_2, \ldots\ldots n_m$ elements, there is rare chance for every sub problem to give average case time complexity]

$=c.\ n+ c.\ (n_1+n_2+\ldots\ldots+n_m)$

$=c.\ n+ c.\ n$ [from, Eq. (1)]

$=2cn$

$\equiv O(n)$

4.3 Worst Case

The algorithm gives worst case time complexity when,

- Inputs elements are randomly distributed.
- All elements except the largest one of the input list have values less than the value of *div* (as it is generated in line number 5 in algorithm1 and it is happen again and again for every sub problem.
- The input series is one of all possible permutation of the elements of the series-----

  $a, i*(i-1)^{th}$ term $+c$ $\forall$ $i=2$ to n ---Eq.(2)

  where a=starting element of this series and having a positive value and c > = 0.

At that situation, time complexity is $T(n)=c\ .n+ T(n-1)$

$\equiv O(n^2)$

which is possible theoretically but not in real life because we are talking about asymptotic time complexity. So, number of input elements is very very high. Every real life database collects and stores similar kinds of data and it is

a very rare chance that at least some of elements in the sorted output series satisfy Eq.(2). So, for every real life problem this algorithm gives average case time complexity. Details of the presented algorithm are in Table2.

Table 2: Details of the Presented Algorithm

Algorithm	***Time complexity***		***Space complexity***	***Stable?***
Algorithm of proposed work	Best case	O(n)	O(n^2)	Yes
	Average case	O(n)		
	Worst case	• O(n^2) Only for the series as in Eq.(2) • O(n)		

5. Conclusion

The main advantage of the presented algorithm is its speed. Selection of sorting algorithm is application and situation dependent but this algorithm works well in every field. For real life sorting problem, asymptotic time complexity of this algorithm is linear and space complexity is O(n^2) for the usage of two dimensional memory which makes it faster.

References

[1] D.E. Kunth, Sorting and Searching, *The Art of Computer Programmin*g: Vol. 3, Addison-Wesley, 1973. Second Edition, 1998.

[2] D.E. Kunth, Fundamental Algorithms, *The Art of Computer Programmin*g: Vol.1, Addison-Wesley, 1968. Third Edition, 1997.

[3] D. E. Kunth, Seminumerical Algorithms, *The Art of Computer Programmin*g: Vol.2, Addison-Wesley,1969. Third Edition 1997.

[4] D. E. Kunth, Big omicron and big omega and big theta. SIGACT News, 8(2)18-23,1976.

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

[6] A. V. Aho , J E. Hopcroft and J. D. Ullman *The Design and Analysis of Computer Algorithms*. Addison-Wesley,1974.

[7] A.V. Aho , J. E. Hopcroft and J. D. Ullman *Data Structures and Algorithms*.Addison-Wesley,1983

[8] R. Lavore, Arrays, *Big O Notation and Simple Sorting. Data Structures and Algorithms in Java* (2nd_Edition). Sams, 978-0-672-32453-6..

[9] S. Baase and A. V. Gelder. *Computer Algorithm: Introduction to Design and Analysis*. Addison-Wesley, Third edition, 2000.

[10] A. Aggarwal and J. S. Vitter.*The input/output Complexity of sorting and related problems*.Communication of the ACM,31(9):1116-1127,1998.

[11] M. Akra and L. Bazzi.*On the solution of linear recurrence equation.Computational Optimization and Application*,10(2):195-210,1998.

[12] J. L. Bentley, D. Haken and J. B. Saxe. *A general method for solving divides and conquers recurrence*. SIGACT News, 12(3):36-44,1980.

[13] S. Jadoon, S. F. Solehria, M. Qayum, "Optimized Selection Sort Algorithm is faster than Insertion Sort Algorithm: a Comparative Study" International Journal of Electrical & Computer Sciences IJECS-IJENS, Vol: 11 No: 02, 2011.

[14] Y. Yang, P. Yu, Y. Gan, (2011) "Experimental Study on the Five Sort Algorithms",International Conference on Mechanic Automation and Control Engineering (MACE).

[15] W. Min, "Analysis on 2-Element Insertion Sort Algorithm", International Conference on Computer Design And Appliations (ICCDA), 2010.

[16] E. Kapur, P. Kumar and S. Gupta-"Proposal Of A Two Way Sorting Algorithm And Performance Comparison With Existing Algorithms"- International Journal of Computer Science, Engineering and Applications (IJCSEA) Vol.2, No.3, June 2012

[17] S. Z. Iqbal, H. Gull, A. W. Muzaffar, "A New Friends Sort Algorithm",IEEE Second International Conference on Computer Science and Information Technology, 2009.

[18] J.L. Bentley and R. Sedgewick, Fast Algorithms for Sorting and Searching Strings, ACM-SIAM SODA" 97, 360-369, 1997.

[19] B. R. and L. S., A Fast Sort, Computer Journal of Byte Magazine, vol. 16, no. 4, pp. 315-315, 1991.

[20] C.A.Hoare. Algorithm 63 (PARTITION) and algorithm (FIND). Communications of the ACM,4(7),321-322,1961.

[21] R.W.Floyd and R.L.Rivest.Expected time bounds for selection. Communications of the ACM ,18(3):165-172, 1975.

[22] M.D.McIlroy. A killer adversary for quicksort. Software -Practice and Experience, 29(4):341-344,1999.

Sanjib Palui has completed his bachelor degree in Information Technology and master degree in Software Engineering in the year of 2012 and 2014 respectively. Then he has joined Indian Statistical Institute as Project Linked Person in the unit of Computer Vision & Pattern Recognition (CVPR). He is working on Data Structure & algorithm, Image Processing and Optical Character Recognition fields.

Dr. Somsubhra Gupta is presently the Assistant Professor of the Department of Information Technology, JIS College of Engineering (An Autonomous Institution). He is graduated from the University of Calcutta and completed his Masters from Indian Institute of Technology, Kharagpur. He received Ph. D. award entitled "Multiobjective Decision Making in Inexact Environment using Genetic Algorithm: Theory and Applications" form University of Kalyani. His area of teaching is Algorithm and allied domains and research area is Machine Intelligence. In research, he has around 56

papers including Book Chapters so far in National / International Journal / Proceedings and over 40 citations. He is Principal Investigator / Project Coordinator to some Research projects (viz. RPS scheme AICTE). He was the Convener of International Conference on Computation and Communication Advancement (IC3A-2013). He is invited in the Technical Programme Committee of number of Conferences, delivered as an invited speaker, an INS (Elsevier Science) reviewer and attended NAFSA-2013 conference of international educators at St. Louis, Missouri, USA