Global Journal of Computer Science and Technology: E
Network, Web & Security
Volume 16 Issue 1 Version 1.0 Year 2016
Type: Double Blind Peer Reviewed International Research Journal
Publisher: Global Journals Inc. (USA)
Online ISSN: 0975-4172 & Print ISSN: 0975-4350

An Optimized Input Sorting Algorithm

By Anshu Mishra & Garima Goyal
Jyothy Institute of Technology, India

Abstract- One of the fundamental issues in compute science is ordering a list of items. Although there is a huge 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. Sorting involves rearranging information into either ascending or descending order. This paper presents a new sorting algorithm called Input Sort. This new algorithm is analyzed, implemented, tested and compared and results were promising.

Keywords: algorithm, sorting, input sort, insert.

GJCST-H Classification: F.2.2

An Optimized Input Sorting Algorithm

Anshu Mishra α & Garima Goyal σ

Abstract- One of the fundamental issues in compute science is ordering a list of items. Although there is a huge 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. Sorting involves rearranging information into either ascending or descending order. This paper presents a new sorting algorithm called Input Sort. This new algorithm is analyzed, implemented, tested and compared and results were promising.

Keywords: algorithm, sorting, input sort, insert.

I. Introduction

Information growth rapidly in our world and to search for this information, it should be ordered in some sensible order. Many years ago, it was estimated that more than half the time on many commercial computers was spent in sorting. Fortunately this is no longer true organizing data, methods which do not require that the data be kept in any special order [1].

Many algorithms are very well known for sorting the unordered lists. Most important of them are Bubble, Heap, Merge, Selection[2]. As stated in [3], sorting has been considered as a fundamental problem in the study of algorithms, that due to many reasons:

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

Efficient sorting algorithms is important to optimize the use of other algorithms that require sorted lists to work correctly; it is also often in producing human readable output. Formally, the output should satisfy two major conditions:

1. The output is in non-decreasing order.
2. The output is a permutation or reordering of the input.

Since the early beginning of computing, the sorting problem has attracted many researchers, perhaps due to the complexity of solving it efficiently. Bubble sort was analyzed as early as 1956[6].

Many researchers considered sorting as a solved problem. Even so, useful new sorting algorithms are still being invented. For example, library sort was first published in 2004. Sorting Algorithms are prevalent in introductory computer science classes, where the abundance of algorithm for the problem provides a gentle introduction of core algorithm concepts[4,5].

In [4], they classified sorting algorithms by:

1. Computational complexity (worst, average and best)of element comparison in terms of list size(n).For typical sorting algorithms good behavior is O(nlogn) and bad behavior is $\Omega(n^2)$. Ideal behavior for a sort is O(n). Sort algorithms which only use an abstract key comparison operation always need Ω(nlogn) comparison in worst case.
2. Use of memory and computer resources. Some sorting algorithms are "in-place", such that only O(1)or O(log n)memory is needed beyond the items being stored, while others need to create auxiliary locations for data to temporally stored.
3. Recursion some algorithms are either recursive or non recursive, while others may be both (e.g merge sort).
4. Whether or not they are a comparison sort. A comparison sort examines the data only by comparing two elements with a comparison operator.

This paper presents a new sorting algorithm called input sort. Its typical use is when sorting the elements of a stream from file.

II. Input Sort

a) Concept

A simple sorting algorithm which sort the data whenever it is input from any input source e.g. keyboard or data from a stream of file. when new item comes then it is inserted at its specific position through a recursive function if there are n elements then n items 1 at a time is inserted in array which increase array size automatically and take its appropriate position.

b) Steps

The procedure of the algorithm can be described as follows:

1. Input one element one at a time.
2. Call the INPUT-SORT function to insert the item.
3. Recursive function determines where is to be the new item inserted in existing array by comparing from the middle element and place it at its specific position
4. After inserting n elements we have new array which is finally sorted.
5. End.

Author α σ : Assistant Professor, Department of Information Science & Engineering Jyothy Institute of Technology, Bangalore.
e-mails: anshu.garg13@gmail.com, goyal.garima18@gmail.com

c) Pseudo Code

```
Size:=0,array c[1…max]  //Global variables
INPUT_SORT(ele , p,r)
q:=(p+r)/2//calculate mid position
if p == r  //base criteria for recursion
{
INSERT(ele,p)
return;
}
 if(ele<c[q]
INPUT_SORT(ele,p,q)
else
INPUT_SORT(ele,q+1,r)
end.

INSERT(ele,pos)
for   j= size down to pos
c[j+1]:=c[j]
c[pos]:=ele
size:=size+1
end.

GET_INPUT( )
for  i = 1  to  n
{
scan(c[i])
call INPUT_SORT(c[i],1,size+1)
}
end
```

III. Working

Suppose we have array c[1…5] of five elements as follows:

First we call GET_INPUT() function and read input from input source

5	4	3	2	1

During 1st Pass: insert 5 call INPUT_SORT(1,1)
Content of array c[size] i.e c[1]

5

P=1,q=1,r=1,size=1

During 2nd Pass: insert 4 call INPUT_SORT(1,2)
Content of array c[size] i.e c[2]

4	5

P=1,q=1,r=2

P=1,q=1,r=1

Similarly ,

During 5th Pass: insert 4 call INPUT_SORT(1,5)
Content of array c[size] i.e c[1…5]

1	2	3	4	5

P=1,q=3,r=5

P=1,q=2,r=3

P=1,q=1,r=2

P=1,q=1,r=1

Now finally: Content of Array c[1…5]

1	2	3	4	5

Which is Sorted Array .

IV. Complexity

Generally the complexity of an algorithm is measured in two phases. When one measures the complexity of an algorithm by pen and paper, he/she can only predict the complexity which give an idea how much time and space this algorithm takes to finish in its execution. This phase is called priory analysis. After implementing the algorithm in computer, we get the actual time and space. This phase of analyzing the algorithm is called the posterior analysis. complexity of an algorithm can be of two types:

1. *Time Complexity:* The analysis of algorithm for the prediction of computation time for execution of each and every instruction in the algorithm is called the time complexity.
2. *Space Complexity:* The analysis of algorithm for prediction of memory requirement of the algorithm is known as space complexity.

The complexity of the algorithms are as follows:

1. Our algorithm run in O(n log n) time
2. Better in average and worst case of bubble and Insertionsort.
3. Better in all cases of selection sort.
4. Better from worst case of quick sort.
5. Easy to implement.
6. It required less memory space than Heap and Merge Sort

Sorting	**Best Case**	**Average**	**Worst**
Bubble	$\theta(n)$	$\Theta(n^2)$	$O(n^2)$
Insertion	$\theta(n)$	$\Theta(n^2)$	$O(n^2)$
Selection	$\theta(n^2)$	$\theta(n^2)$	$O(n^2)$
Merge	θ (nlogn)	θ (nlogn)	O(nlogn)
Quick	θ (nlogn)	θ (nlogn)	$O(n^2)$
Heap	θ (nlogn)	θ (nlogn)	O(nlogn)
Input	**θ (nlogn)**	**θ (nlogn)**	**O(nlogn)**

V. Recurrence Equation of Input Sort

Using the standard recurrence equation $T(n)=aT[n/b]+f(n)$ get this equation:

$T(n)=2T[n/2]+n$ $a=2$ $b=2$ $f(n)=n$

$n^{\log_b a} = n^{\log_2 2} = n$

using master method's 2nd case apply

if $f(n)=\theta(n^{\log_b a})$, then

$T(n)=\theta(n^{\log_b a} . \log n)$

Time complexity of Input Sort is

$T(n)=\theta(n \log n)$

VI. Comparison with Heap and Merge Sort

Now if we talk about heap merge sort than our algorithm is better from two in the sense that In merge sort we need two extra temporary array which increase its space complexity but no need of extra memory in our algorithm. our algorithm has order of O(nlog n) but it execute fast because of less comparisons than Merge heap and Quick sort.

VII. Conclusion

In this paper new sorting algorithm is presented INPUT-SORT has O(nlog n) complexity but it is faster than existing sort mentioned in section 4 in detail. INPUT-SORT is definitely faster than other sort to sort n elements. Furthermore, the proposed algorithms are compared with some recent sorting algorithms; selection sort and bubble sort, heap, merge, insertion, quick sort. These algorithm can be applied on a real world application. any sorting algorithm might be a subroutine of another algorithms which affects its complexity.

References Références Referencias

1. Knuth D., The Art Of Computer Programming, Addison Wesley,1998.
2. Shahzad B. and Afzal M.,"Enhanched Shell Sorting Algorithm,"computer general of Enformatika, vol.21, no.6, pp.66-70,2007.
3. Cormen T., Leisersion 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. Thoroup M.,"Randomized Sorting in O(n log log n) Time And Linear Space Addition, Shift, and Bit Wise Boolean Operations,"Computer Journal of Algo rithms, vol:42, no.2, pp.205-230, 2002.
6. Astrachanm O., Bubble Sort: An Archaeological Algorithmic Analysis, Duk University, 2003

Global Journal of Computer Science and Technology (H) Volume XVI Issue I Version I Year 2016

This page is intentionally left blank