Volume 4, Issue 3, March 2014 ISSN: 2277 128X

International Journal of Advanced Research in Computer Science and Software Engineering

Research Paper

Available online at: www.ijarcsse.com

Advance Index Sorting Algorithm

Pushpak Jaiswal
Department of Computer science
Manipal University Jaipur
Jaipur, India

Quashid Mahboob
Department of Computer science
Manipal University Jaipur
Jaipur, India

Abstract*— In this paper we have compared many sorting algorithms with new proposed Advance Index sorting Algorithm/AISA. Previously, Index Sorting Algorithm is capable of sorting non-repeating positive integers. In Advance Index sorting, we have come up with an idea of sorting positive repeating/non-repeating Integers having complexity, this algorithm is capable of searching element with O(1) complexity once sorting is complete. This algorithm can also be used to find the maximum repeating elements with complexity n. This algorithm was analysed, implemented and tested and the results are promising for a random data.

***Keywords*— Positive; sorting; integer; index; repeating;**

I. Introduction

Today real world getting tremendous amounts of data from various sources like data warehouse, data marts etc. To search for particular information we need to arrange this data in a sensible order. Many years ago, it was estimated that more than half the time on commercial computers was spent in sorting. Fortunately variety of sorting algorithms came into existence with different techniques. Search engine is basically using sorting algorithm. When you search some key word online, the feedback information is brought to you sorted by the importance of the web page. Bubble, Selection and Insertion Sort, they all have an O(n^2) time complexity that limits its usefulness to small number of element no more than a few thousand data points. The quadratic time complexity of existing algorithms such as Bubble, Selection and Insertion Sort limits their performance when array size increases.
In this paper we introduce Advance Index Sorting Algorithm /AISA which is able to rearrange elements of a list in ascending order. Advance Index Sorting sorts the element by putting that Element to its index position.
Our main contribution is the introduction of Advance Index Sorting Algorithm, an efficient algorithm can sort a list of array elements in O(n) time. We evaluate the O(n) time complexity theoretically and empirically.

II. THE TECHNIQUE FOR SORTING ELEMENTS

A. Bubble sort

The simplest sorting algorithm is bubble sort. The bubble sort works by iterating down an array to be sorted from the first element to the last, comparing each pair of elements and switching their positions if necessary. This process is repeated as many times as necessary, until the array is sorted. Since the worst case scenario is that the array is in reverse order, and that the first element in sorted array is the last element in the starting array, the most exchanges that will be necessary is equal to the length of the array.

B. Insertion sort

An insertion sort works by separating an array into two sections, a sorted section and an unsorted section. Initially the entire array is unsorted. The sorted section is then considered to be empty. The algorithm considers the elements in the unsorted portion one at a time, inserting each item in its suitable place among those already considered (keeping them sorted). To insert an item in a certain location all other items up to that location needs to be shifted to the right. However, memory writes are costlier than memory read and many write operations are required only to place a single item in its proper place. The proper location of a new item in the sorted portion can be searched either by a linear search or by a binary search. Certainly, the binary insertion sort, the version using binary search, has better performance than the other one.

C. Merge sort

Suppose an array A with n elements a1, a2, ..., an in memory. Merge sort algorithm works by the use of divide-and-conquer algorithm. It proceeds by taking pair of elements, sort them and merge it with another sorted pair and sorting this pair. After some pass K, the array will be partitioned into sorted sub-arrays where each sub-array, except possibly the last, will contain exactly 2k elements. Therefore the algorithm will require at most (worse-case) log n passes to sort an n element array A.

D Quicksort

The Quicksort algorithm developed by Hoare [9] is one of the most efficient internal sorting algorithms and is the method of choice for many applications. The algorithm is easy to implement, works very well for different types of input data, and is known to use fewer resources than any other sorting algorithm [1]. All these factors have made it very popular. Quicksort is a divide-and-conquer algorithm.

To sort an array A of elements, it partitions the array into two parts, placing small elements on the left and large elements on the right, and then recursively sorts the two sub arrays. Sedgwick studied Quicksort in his Ph.D. thesis [2] and it is widely described and studied in [3], [4], [5],[6] and [7].

III. INDEX SORT ALGORITHM

Elements in memory are stored as the index for the position of each number. Assuming the elements in memory are in array a[n], we store this elements into another array b[n] by assigning each element a[i] to variable say k, that is, k=a[i], i starting from 0, 1, ..., n . Write each element into their respective position in array B, that is b[k]= k.

Traverse array B and copy out the data leaving out indexes where the content is zero [8]. Specifically, the following program segment will sort any unsorted positive integer numbers in array A into array B:

```
for (j=0;j<=m;++j)
{
k=a[j];
b[k]=k;
}
```

IV. THE ADVANCED INDEX SORTING ALGORITHM /AISA

The main logic presented here is, Suppose we have some elements denoted by a1, a2,…an in array b[].This new algorithm "Advance Index sorting" works by storing these elements in another array say A, with each integer stored as their index. With this algorithm, an integer number say 1 will be stored in position 1 of the array, a number say 8 will be in position 8 of the array and so on. This will proceed until all the numbers have been fixed in their rightful position in the array and if the number repeats then there will be increment in the value of that index by 1. After that we will sort that number by using certain code.

We have only one index for a number so sorting two or more similar numbers on single index is really a difficult task. Sorting repeating number using index will increase one for loop in code but the result will be promising.

The working of Advanced Index Sorting is very similar to the working of Index Sort Algorithm. Advanced Index Sorting Algorithm/ AISA is capable of sorting repeating integers also which Index sorting lack.

Note: we are assuming that 0 will not be element of an array B[].

```
• Input: An unsorted array B[ ] of size n.
• Output: A sorted array B[ ] of size n.
1. for i=0 to n
2. begin
3.  l=B[i]
4.  if (A[l]==0) do  a[l]=l
5.  else do  A[l]=A[l]+1
6.  end
7.  k=0
8.  for i=1 to max+1
9.  begin
10.for j=0 to A[i]-i+1
11.begin
12. B[k]=i
13. k++
14. end
15. end
16. for i=0 to n+1
17. begin
18. cout<<B[i]<<" "<<"\n"
19. end
```

The working of the above algorithm can be understood by the following example. Consider the following input array:

n=10,

B[10]= {6,7,3,13,2,3,6,1,2,6}

max=13.

We can add an extra for loop to find maximum value in the given array. We will require an extra array say A[] of size max+1

Here, A[max+1]=A[14]={0}

Line 1-6 which is used to put the elements of array B[] to the proper index in array A[].

The first 5 non repeating element of array B[] will be placed to the proper index of array A[]. This process will be done by 4-5 line.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	2	3	0	0	6	7	0	0	0	0	0	13

Above 1st row indicates the index position and 2nd row indicates the placed value.

Now the 6th Element of array B[], 3 will be repeated to 3rd index so we will increment the value 3 by 1. This process will be done by line 5.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	2	4	0	0	6	7	0	0	0	0	0	13

similarly 7th Element of array B[], 6 will be repeated to 6th index so the value of 6th index will be incremented by 1.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	0	2	4	0	0	7	7	0	0	0	0	0	13

8th Element will be placed to its index position 1.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	2	4	0	0	7	7	0	0	0	0	0	13

9th Element (2) will be repeated so we will increment the value by 1 at 2nd index.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	3	4	0	0	7	7	0	0	0	0	0	13

10th Element (6) will be repeated so we will increment the value by 1 at 6th index.

0	1	2	3	4	5	6	7	8	9	10	11	12	13
0	1	3	4	0	0	8	7	0	0	0	0	0	13

1st row indicates the index i.
2nd row indicates the array A[].
A[i]-i= total repeated value for index i .
A[i]-i+1= total number of element i in array B[].
Line 8-15 will transfer all the values from array A[] to B[] and we will get a sorted array B[].

Program for above Example.

```
#include <stdio.h>
#include<stdlib.h>
#include<iostream.h>
using namespace std;
int main()
{
int b[10]={6,7,3,13,2,3,6,1,2,6}
int i,j,n=10,a[14]={0},l,k;
for(i=0;i<n;i++)
{
l=b[i];
if(a[l]==0)
{
a[l]=l;
}
else
{
a[l]=a[l]+1;
}
}
k=0;
for(i=1;i<14;i++)
{
for(j=0;j<a[i]-i+1;j++)
{
b[k]=i;
k++;
}
}
for(i=0;i<n;i++)
{
cout<<i<<" "<<b[i]<<" "<<"\n";
}
return 0;
}
```

Advantages

- Complexity : $n(n-m+1)$, Worst Case: $n(n-m+1)$, Average Case: $n(n-m+1)$, Best case: n
- Searching: we can directly search any number at its index position with complexity O(1).
- Repetition: We can also find the maximum repeating number with complexity O(n) in array a[].
- Two arrays for different purpose: Array b[] will store the sorted Elements and Array a[] will store the other information like repeating values in an array.
- No wastage of two memories. As both arrays will store some necessary information.
- Bucket sort stores its duplicate using link list but here we are using concept of increment at its index position.

Disadvantage

- It is capable of sorting only Integers.
- Space complexity (n^2).

IDEA FOR SORTING NEGATIVE NUMBERS

Sorting negative numbers is major problem using indices because indices are defined as positive integer. This idea can be used to sort negative number in any index sort algorithm. If there is negative element in array then firstly we need to find the minimum negative number let's say -k. Then we need to convert that negative number into positive number k. Add 1 to k. After that add (1+k) to entire array this will change every negative number into positive number ranging 1 to maximum value. Store all the numbers at its indices. This will sort the whole array. While transferring the element into second array subtract (k+1) from the previous array this will sort the entire integers.

Example

A[] ={-5,3,1,2,-3}

B[] ={0,0,0,0,0}

1.) k= -5 // minimum negative number
2.) k=5 // converting into +ve number.
3.) K=5+1 // adding 1 to k.
4.) K=6
5.) A[]={1,9,7,8,3}
6.) B[]=

0	1	0	3	0	0	0	7	8	9

Transfer whole element into another array at their index position. We can see that above elements are sorted.

7.) Now transfer the elements of array B[] neglecting 0 by subtracting k=6.
8.) A[]=

-5	-3	1	2	3

9.) Finally element will be sorted.
10.) For sorting repeating numbers, we can use the idea as discussed in above Algorithm.

We don't have negative index to sort negative integers so to do that we have to go through above idea. The idea of sorting negative numbers lacks in many sorting algorithm e.g. Array-indexed sorting algorithm for natural number, Index sort algorithm for positive integer, Insertion sort, Self-Indexed sort... etc. This idea will remove this disadvantage of above mentioned sorting algorithm. The disadvantage of this idea is it will increase the space complexity but there will not be any change in running cost.

V. RUNNING COST ANALYSIS

This algorithm is suitable for large arrays. The outer loop will run max+1 times and the inner loop will depend on the total number of repeating Elements in given array. By keen observing it the worst case running cost of algorithm is always less than O (n^2). The algorithm will attain complexity O (n^2) when the entire numbers in given array are similar. The behaviour of the algorithm in the best case will be O (n), depicting that all the element in array B[] are different. Similarly the average case of the running cost will be O (n) depending upon the total number of repeated elements in array B[].

VI. SPACE COMPLEXITY OF ALGORITHM

The space complexity of an algorithm is the number of elements which the program needs to store during its execution. This number is calculated by the size of input which is provided to a program. By keen observing complexity will be O(n^2) and it requires an extra array different size as input array in both cases.

VII. COMPARISON OF ADVANCE INDEX SORT WITH THE EXISTING SORTING ALGORITHM

To check the efficiency of algorithm, it was compared with other sorting techniques. By taking lists of various sizes, items were generated using a uniform random number generator with values ranging from 1 to 25000. Running times were noted down. The plots and table are given below.

Table: 7.1 Time comparison (in nano second)

No of Elements	**Time comparison for different Algorithm**		
	Advance index sorting Algorithm	**Quicksort**	**Bubble Sort**
5000	0	50000	140000
10000	10000	190000	520000
15000	10000	440000	1180000
20000	20000	780000	2080000
25000	20000	1190000	3230000

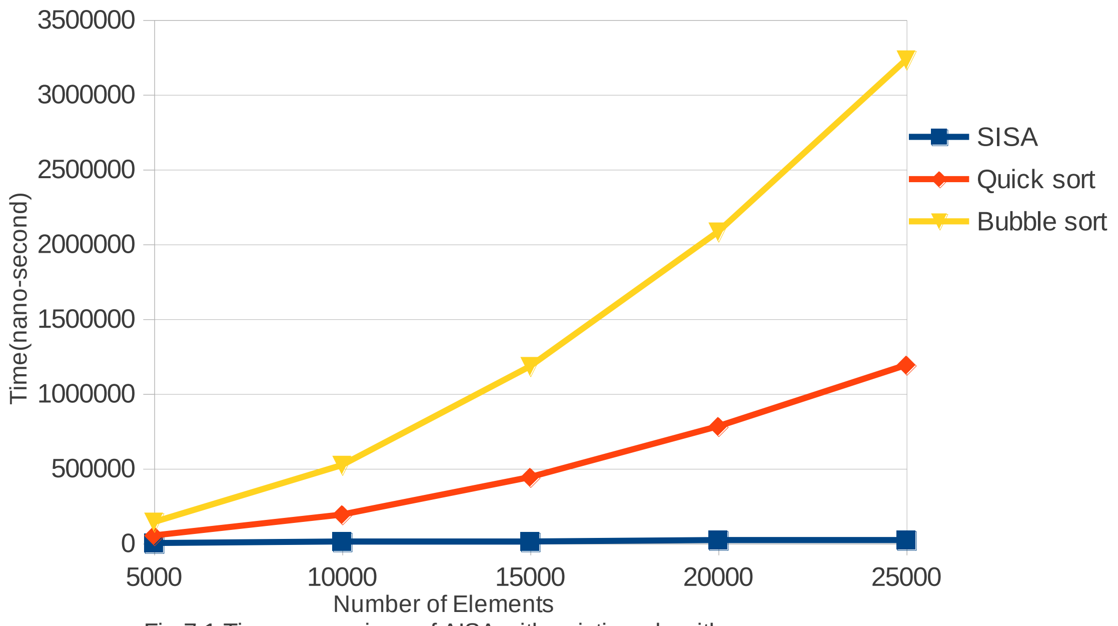

Fig:7.1 Time comparison of AISA with existing algorithm

VIII. CONCLUSION AND FUTURE WORK

In this paper we presented an algorithm Advanced Index Sorting, which give a better running time than the existing sorting algorithms of the different complexity class (e.g. Bubble sort, Quick sort, selection sort, merge sort, Insertion sort), for larger data, this algorithm is capable of searching element with O(1) complexity once sorting is complete. This algorithm can also be used to find the maximum repeating elements with complexity n. Overall this analysis shows that Advanced Index Sort/AISA is very efficient for large number items with same length of digits of elements in list. As we can see that the space complexity of above algorithm is so, the future work includes reducing this space complexity and sorting floating numbers.

ACKNOWLEDGMENT

I would like to express my gratitude and appreciation towards Dr. Shalini jain HOD, Department of mathematics , MUJ who was a constant guiding support throughout the process.

REFERENCES

[1] R. Sedgewick, Algorithms in C++, 3rd edition, Addison Wesley, 1998.
[2] R. Sedgewick, “Quicksort,” PhD dissertation, Stanford University, Stanford, CA, May 1975. Stanford Computer Science Report STAN-CS-75-492.
[3] R. Loeser, “Some performance tests of :quicksort: and descendants,” Comm. ACM 17, 3 , pp 143 – 152, Mar. 1974.
[4] J. L. Bentley and R. Sedgewick, “Fast algorithms for sorting and searching strings”, In Proc. 8th annual ACM-SIAM symposium on Discrete algorithms, New Orleans, Louisiana, USA, 1997, pp 360 - 369 .

[5] R. Chaudhuri and A. C. Dempster, "A note on slowing Quicksort", SIGCSE Vol . 25, No . 2, Jane 1993.
[6] R. Sedgewick, "The Analysis of Quicksort Programs," Act Informatica 7, pp 327 – 355,, 1977.
[7] R. Sedgewick, "Implementing Quicksort programs," Comm.of ACM, 21(10), pp 847 – 857, Oct. 1978.
[8] Index sort algorithm for positive integers S.E Adewumi science world journal vol 3 (No3) 2008.
[9] C.A.R. Hoare, "Algorithm 64: Quicksort," Communications of the ACM, Vol. 4 , pp. 321, 1961.
[10] D.E. Kunth, The Art of Computer Programming: Vol. 3,Sorting and Searching, 2nd printing, Addison-Wesley, Reading, MA, 1975.