Journal of Information and Operations Management
ISSN: 0976–7754 & E-ISSN: 0976–7762 , Volume 3, Issue 1, 2012, pp-319-321
Available online at http://www.bioinfo.in/contents.php?id=55

NOVEL SORTING TECHNIQUE FOR LARGE DATABASES

ANUJ KUMAR[1], RAMA SUSHIL[1]* AND SUSHIL KUMAR[2]

[1]Department of MCA SGRRITS, Dehradun, Uttarakhand, India
[2]Wadia Institute of Himalayan Geology, Dehradun, Uttarakhand, India
*Corresponding Author: Email- ramasushil@yahoo.co.in

Received: December 12, 2011; Accepted: January 15, 2012

Abstract- Sorting is frequently used in a large variety of important applications used by schools, hospitals, banks and in many other organizations. This paper presents a novel sorting technique named "Position Sort". This sorting technique provides the correct position to an element by only one swapping operation. It is an improved sorting algorithm with lesser running time and number of swapping operations in comparison to some other existing techniques like Bubble sort and Selection sort.
Keywords- Sorting, Bubble sort, Selection sort, Swapping, Correct position

Citation: Anuj kumar, Rama Sushil and Sushil Kumar. (2012) Novel Sorting Technique for Large Databases. Journal of Information and Operations Management ISSN: 0976–7754 & E-ISSN: 0976–7762, Volume 3, Issue 1, pp-319-321.

Introduction

An algorithm is a finite sequence and well-defined set of computational instructions that takes some value or set of values as input and produces some value as a result [1]. A good algorithm is that which gives satisfactory result for every range of data set. Sorting is a very basic concept and important for solving other problems like is prerequisite for Binary Search. Sorting is the fundamental problem of computer science and remained burning issue for research over the last several years due to time complexity [2]. Sorting is often used in a large variety of critical applications and is a fundamental task that is used by most computers. Sorting algorithm falls into two basic categories: comparison based and non-comparison based. The comparison based sorting algorithm works on the basis of comparing the elements. Comparison based important algorithms are: quick sort, merge sort, heap sort, bubble sort, and insertion sort [3]. A non-comparison based algorithm sorts an array without consideration of pair wise data elements. Radix sort is a non-comparison based algorithm [4].

Some existing algorithms are very fast but complex to implement, while some are not fast but easy to implement. Moreover some are better option for small size data while some for larger size data. Some sorting algorithms work on small data-size, some are suitable for floating point numbers, some are good for specific range, some are better for large dataset, and some are useful for data set having non-distinct values also.

There are two groups of sorting algorithms one having complexity $O(n^2)$ which include bubble, insertion, selection and other with complexity O(nlogn) which includes heap, merge and quick sort techniques.

In general we have two operations in comparison based sorting techniques, one is "comparison" and second is "swapping". But Comparison operation is considered as the key operation and complexity of a sorting technique is defined on the basis of total number of comparison operations while ignoring the "swapping" operations. Practically it is observed that swapping operation effects the running time and increases the CPU work load.

In this paper we are introducing a simple and efficient novel sorting technique named "Position Sort". This technique finds the correct

position of a particular element and places that element at that position. After placing the element at correct position that element does not remain involved in swapping operations further. It places the elements at their correct position after one swapping operation only. Practically it takes lesser running time than selection and bubble sort therefore is an efficient sorting technique for large data set.

Position Sort

This sorting technique is applicable on distinct and non-distinct data set. Suppose we have an array of size 10 and we select the i^{th} indexed element. Than we count all the smaller elements than the pivot element. Suppose total no. of smaller element is "count" then we swap the pivot (i^{th} indexed) element with the $[count+i]^{th}$ indexed element. That position will be the correct position of that pivot element. After this swapping, that pivot element will not be involved in another swapping operation.

In the second case we allow the repetition of elements in the list. The procedure remains same but if the element which is ready to swap with pivot element is equal to the pivot element then we will not swap but move on to the next element and check if that element is not equal to pivot element then perform swapping and so on.

Analysis of the position sort algorithm provides the following results: In the average case, the position sort performs the sorting by maximum (n-1) swapping operations only, where n is data size. In the worst case it performs maximum n/2 swapping operations only.

Position Sort Steps by an Example

Let us take an array named list[10] as following:

0	1	2	3	4	5	6	7	8	9
12	9	7	13	23	2	17	4	8	21

First we select 0^{th} index element (as pivot element). Total numbers of smaller elements are counted than the pivot element (we have a variable named count. Initially assigned by 0 and increment that when we find smaller element). After all comparisons we have the '5' smaller elements then we swap the pivot element with 5^{th} element from itself. It means 12 will be swapped with 2 and list[10] becomes as following:

0	1	2	3	4	5	6	7	8	9
2	9	7	13	23	12	17	4	8	21

We have completed the first iteration and placed the pivot element (12) at its correct and final position. Correct position is the position where it would be in the sorted list and 12 is placed at its appropriate position by just one swapping. Element 12 is shaded with grey color. It's an indication of correct position of the particular element. Now again we selected the 0^{th} index element as pivot element which is '2' and repeat the above procedure, no element is smaller than '2'. It means the pivot element is already its appropriate position. Now we will move at next element. So now two elements are at final correct position shown in below list:

0	1	2	3	4	5	6	7	8	9
2	23	7	13	9	12	17	4	8	21

Similar steps will be running till all elements get their appropriate position. Let's see a complete solution in a single glance.

Array: list

12	9	7	13	23	2	17	4	8	21
2	9	7	13	23	12	17	4	8	21
2	9	7	13	23	12	17	4	8	21
2	23	7	13	9	12	17	4	8	21
2	21	7	13	9	12	17	4	8	23
2	8	7	13	9	12	17	4	21	23
2	13	7	8	9	12	17	4	21	23
2	17	7	8	9	12	13	4	21	23
2	4	7	8	9	12	13	17	21	23
2	4	7	8	9	12	13	17	21	23
2	4	7	8	9	12	13	17	21	23

Pseudocode of the Position Sort

```
Algorithm: Position_sort (list, record, n-1)
        i=0
        while (i <= n-1)
        {   count = 0;  j = i+1 ;
                while (j < n)
                    { If (list[i] > list[j]) then
                            count++ ;
                            J++ ;
                    }
        k = 0;
        If (count > 0)  then
            {   while (k <= n-1)
                    If   (list[i] != list[i+count+k])
                    { swap   the   list[i]   and list[i+count+k]
                      break;
                    }
                Else
                  {
                    k++ ;
                    }
              }
              Else
              {
                    i++ ;
               }
        }
```

Implementation & Comparative Study

Running time of position sort is observed with bubble sort, selection sort and insertion sort using same data set using "C free" compiler. Comparative running time has taken at various sizes of data in worst-case and shown in figure 1

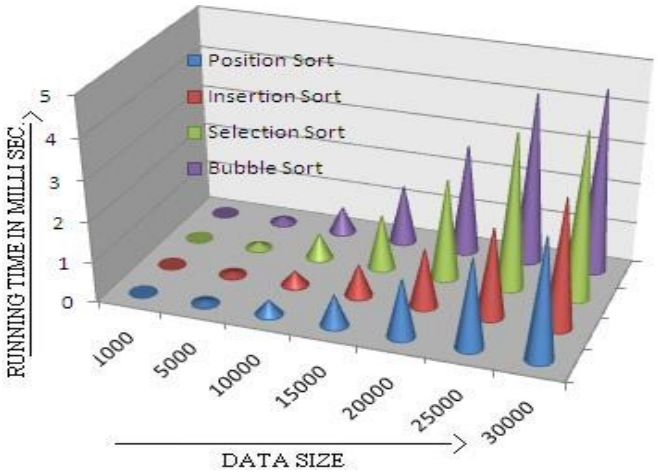

Fig. 1- Running time vs. Data size

Number of swapping operations are also observed for the above four techniques. In worst case the Position Sort has the lesser swapping operations in comparison to other sorting algorithms to perform the sorting operation shown in figure 2.

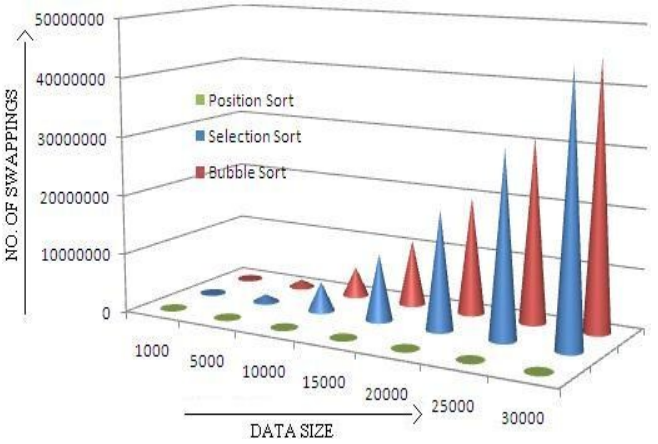

Fig. 2- Number of swapping operations vs. Data size

Conclusion and Future Work

Concept of Position sort is simple. It uses maximum (n-1) swapping operations to sort the given data of size 'n'. It places the element at its correct position, i.e. position where that element will be in sorted list, after one swapping operation only. Use of the second array named "record" for keeping the record of correct positioning elements helps to increase the efficiency of the technique by decreasing the comparison operations.

In future we intend to compare it with other existing techniques and computing the time complexity of it.

References

[1] Cormen T.H., Leiserson C.E., Rivest R.L. and Stein C. (2003) *Introduction to Algorithms MIT Press, Cambridge, MA, 2nd edition*.

[2] Alfred V., Aho J., Horroroft, Jeffrey D.U. (2002) *Data Structures and Algorithms*.

[3] Frank M.C. (2004) *Data Abstraction and Problem Solving with C++. US: Pearson Education, Inc*.

[4] Seymour Lipschutz (2009) *Data Structure with C, schaum Series, Tata McGraw-Hill Education*.