



Enhancing the Efficiency of Radix Sort by Using Clustering Mechanism

¹Manju Devi, ²Saurabh Charaya

¹P.G. Student, Computer Science & Engineering Department, OM Institute of Technology & Management, Hisar, Haryana, (India)

²Assistant Professor, Computer Science & Engineering Department, OM Institute of Technology & Management, Hisar, Haryana, (India)

¹manjusingla75@gmail.com , ²saurabh.charaya@gmail.com

Abstract: Sorting is a technique that arranges data in a specific order, whether it be ascending or descending or lexicographical. Various sorting algorithms have been proposed till the time. Radix Sort is one of them. Time taken for sorting is the major issue for all sorting algorithms. Less the time taken for sorting, more efficient the algorithm is considered. Our aim is to enhance the efficiency of Radix Sort by using clustering mechanism without altering the existing Radix Sort algorithm. Elements to be sorted are grouped into various clusters on the basis of their distance from each other. For this purpose K-means clustering algorithm is used. Then these clusters are sorted separately using radix sort. By using this approach, time taken for sorting is reduced substantially and hence efficiency of radix sort has been increased.

Keywords: Radix sort, LSD, MSD, Clustering, K-means, Unstable sorting

I. INTRODUCTION

Sorting refers to arranging data within a particular fashion. Sorting algorithms specify way to arrange data within a particular order. Most commonly used orders are numerical or lexicographical order. Today, in the era of computer science, efficient sorting is the key requirement.

Importance of sorting lies within the fact that data searching could be optimized to a very high level if data is stored in the database within a sorted manner. Also after sorting data become more readable. Many sorting algorithms are available which can be used to sort any type of data and of any size.

1.1 In-place Sorting and Not-in-place Sorting: Sorting algorithms may require some extra space for comparisons and temporary storage of few data elements. The sorting algorithms which do not require any extra space for sorting and sorting is said to be happened in-place, or in other words, within array itself, is called in-place sorting algorithm. Examples of in-place sorting include bubble sort.

But within some sorting algorithms, extra space is required for sorting which may be more than or equal to the space required for the elements being sorted. This type of sorting is called **not-in-place sorting**. Examples of not in-place sorting include merge sort.

1.2 Stable and Not Stable Sorting: If a sorting algorithm, after sorting elements, does not change the sequence of similar elements within which they appear, called stable sorting algorithm.

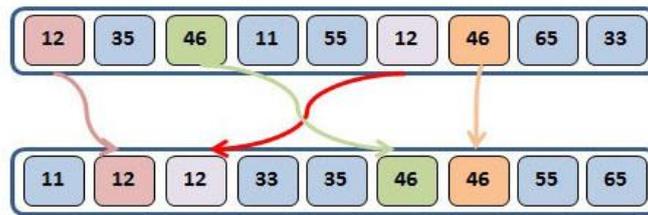


Fig. 1 Stable Sorting

If a sorting algorithm, after sorting contents, changes the sequence of similar elements within which they appear, is called **unstable sorting algorithm**.

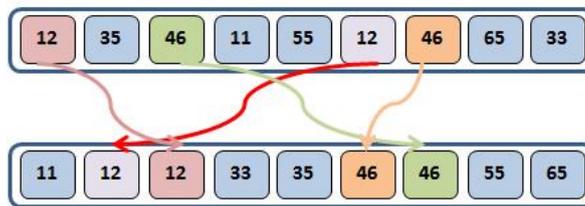


Fig. 2 Unstable Sorting

1.3 Adaptive & Non-Adaptive Sorting Algorithm: A sorting algorithm is said to be adaptive, if it takes advantage of already 'sorted' elements within the list that is to be sorted. In other words, while sorting if source list has some elements which are already sorted, adaptive algorithm will take this into account and will try not to re-order them.

A non-adaptive sorting algorithm is one which doesn't take into account elements which are already sorted. It tries to force every single element to be re-ordered to confirm its sorting.

Radix Sort: It is a non-comparison based integer sorting algorithm that sorts data with integer keys by grouping keys by individual digits which share same significant position and value. Because integers could represent strings of characters (e.g., names or dates) and specially formatted floating point numbers, radix sort is not limited to integers. Database containing integers as well as strings and floats is called heterogeneous database and MRS is a radix sort algorithm developed by Avinash Shukla and Anil Kishore Saxena for sorting heterogeneous data set [6]. Least significant digit (LSD) radix sort and most significant digit (MSD) radix sort are two classifications of radix sort.

1.4 LSD Radix Sort: Least Significant Digit (LSD) radix sort or Right Radix Sort processes integers starting from least significant digit (or right most digit) and move towards most significant digit. This is also called Right Radix Sort.

1.5 MSD Radix Sort: Most Significant Digit (MSD) Radix Sort or Left Radix Sort processes integers starting from most significant digit(or left most digit) and move towards least significant digit. This is also called Left Radix Sort.

II. RELATED WORK

2.1 ARL, a faster in-place, cache friendly sorting algorithm: This paper presented ARL which is fast, in-place, unstable, adaptive, recursive left radix (MSD) sorting algorithm. Among these features adaptive and in-place are new features added by its author [Maus]. Its time complexity is $O(N \cdot \log M)$ and space requirements are $O(N + \log M)$ – where N is the number of

integers to be sorted and M is the maximum value sorted. It is coded in pseudo-Java. ARL is almost twice as fast as Quicksort if number of integers to be sorted is greater than 100[Maus]. ARL is also faster than Right Radix algorithm when number of integers to be sorted is large ($n > 5 \cdot 10^6$). ARL uses half as much memory as Right Radix algorithm [3].

2.2 A full parallel radix sorting algorithm for multicore processors: This paper presented PARL which is a parallel left radix sorting algorithm which is to be used on ordinary shared memory with multi core machines, that has just one simple statement in its sequential part. It can be seen as a full parallel version of the ARL (Adaptive Left Radix) sequential sorting algorithm. It uses multithreading in its implementation considering the fact that to start threads in Java costs approximately 2 to 3 milliseconds but its substantiality reduces the overhead of sequential execution of small problems. For implementing it, a special design pattern is used for generating a thread pool of k threads (an old idea), the same number of threads as processor cores. PARL is 10-30 faster than standard Java Sort for sufficiently large values of n (n is number of integers to be sorted)[5].

2.3 Review of Radix Sort & Proposed Modified Radix Sort for Heterogeneous Data Set within Distributed Computing Environment: This paper discussed problems of radix sort, brief study of previous works of radix sort & presented new modified pure radix sort algorithm for large heterogeneous data set. This algorithm is implemented on principal of divide & conquers. It was observed that no single method is optimal to all available data sets with varying complexity of size, number of fields, length etc. Thus an attempt was made to select a set of data set & optimize the implementation by modifying the basic algorithm. This algorithm is dependent on the distributed Computing Environment. It was implemented on many core machines. The given heterogeneous list is divided into two main processes one is numeric and other is string. Strings were provided some numeric value and then proposed algorithm was applied on these numeric values for sorting the given string. According to the authors, the given algorithm could do much better job over existing sorting algorithms. Both time & space complexities are optimized with their algorithm. Their results had shown an improvement of 10:20% within computational complexity compound with MRS sort & GPU Quick sort [6].

2.4 Comparison of Bucket Sort and RADIX Sort: In this paper, time usage and memory consumption of Bucket Sort and Radix Sort for different kinds of input sequences has been measured. RADIX sort was of least significant digit version and used counting sort as underlying sorting algorithm. Bucket sort used the insertion sort as its underlying sorting algorithm. The sorting algorithms were compared using six different use cases and three different input sizes. Bucket sort was found faster than radix sort in all cases, but bucket sort uses more memory in most cases. RADIX sort is as quick for unsorted inputs as it is for sorted inputs but bucket sort is quicker for already sorted inputs. Both radix sort and bucket sort were found to be slow for large input ranges [7].

III. OBJECTIVES

- To conduct a brief literature review to study various implementations of Radix Sort and analyze their effect on time and space complexity of Radix Sort
- To apply clustering mechanism on the data to be sorted
- To reduce the time taken for sorting by Radix Sort using clustered data
- To reduce the space requirements by removing empty clusters if any

IV. RESEARCH METHODOLOGY

The sources used for getting information and knowledge for this technical paper are literature readings such like books, articles, research papers & journals, internet pages, blogs etc. Some Research Methodologies on which this Research will be focused are:

- **K-means Clustering:** For clustering of data K-means clustering algorithm is used
- **Empty Cluster Removal Code:** To reduce space requirements, empty cluster removal code is used
- **JDK 1.7:** For compiling and executing the K-mean clustering program and empty cluster removal program, java tool JDK 1.7 is used
- **MATLAB R2010a:** For applying `radixsort(x,r)` function on clusters, taking readings of time taken for sorting various clusters and plotting graphs MATLAB R2010a tool is used
- **MATLAB Implementation of Radix Sort:** For sorting purpose we have taken the MATLAB implementation of Radix sort function which is available at

<https://www.mathworks.com/matlabcentral/fileexchange/45125-sorting-methods/content/Sorting%20Methods/radixsort.m>

V. EXPERIMENTAL RESULTS

We have taken a database having the fields ID, MHEAD, HEAD, ITEM, QTY and UNITS with about 2380 records. We have to sort this database according to field QTY (which denotes the value of quantity of each item) in increasing order. So our dataset to be sorted is the values of the field QTY. We have applied clustering process on the dataset. **Cluster analysis** or **Clustering** is the process of grouping a set of objects in more than one groups in such a way that objects in the same group (called a **cluster**) are more similar (in some sense or another) to each other than to those in other groups (clusters).

For clustering, K-Means clustering algorithm is used. K-means algorithm uses distance of objects in the dataset as grouping function. We have classified the dataset into three clusters. In the implementation of K-Means clustering [9], first three values of the dataset are taken as initial mean value of clusters. So we have placed the record with lowest value of quantity at the top most position so that small values of quantity get grouped into first cluster. Similarly, the record with highest value of quantity is placed at third position and average of highest and lowest value is placed at second position so that large values of quantity get grouped into third cluster and medium values get grouped into second cluster. Now, items in the quantity dataset are assigned to different clusters depending upon the distance from mean value of clusters. The item is assigned to the cluster from which the distance of the item is lowest. After assigning items one time, mean of each cluster is recalculated and again items are assigned to the clusters according to their distance from new mean. This process is repeated until the grouping of items is not similar to the previous step's grouping. Empty clusters (if formed) are removed using empty cluster removal code.

After applying K-Means clustering on dataset, all the three clusters get stored into three text files. Now these three text files are opened in read mode in MATLAB and every value in the text files is converted into integers. Integer values from each text file are then stored in different vectors. Then these vectors belonging to three different clusters are sorted using MATLAB implementation of `radixsort(x,r)` function. As the first cluster contains small values, second cluster contains medium values and third cluster contains large values of the quantities, so there is no need to sort the sorted clusters again for getting the complete sorted dataset. The sorted clusters are simply appended serial wise one after another and we get the complete sorted dataset. Following table shows the time taken for sorting in three different cases. In the first case the vector clusters are sorted in parallel on multicore processors (we have simulated the behaviour of multi core processors using MATLAB), in the second case all the vector clusters are sorted sequentially on a single processor. In the third case no clustering is used, rather sorting is applied on the whole dataset.

Table 1 Time Taken for Sorting

Case	Time taken for sorting (in seconds)		
	Speciman 1	Speciman 2	Speciman 3
Clusters are sorted in parallel	0.0077	0.0105	0.0094
Clusters are sorted sequentially	0.0117	0.0168	0.0122
No clustering is used	0.0760	0.0710	0.0825

Every time we apply sorting in MATLAB, different timings are shown. That is why; we have taken three spcimans of readings.

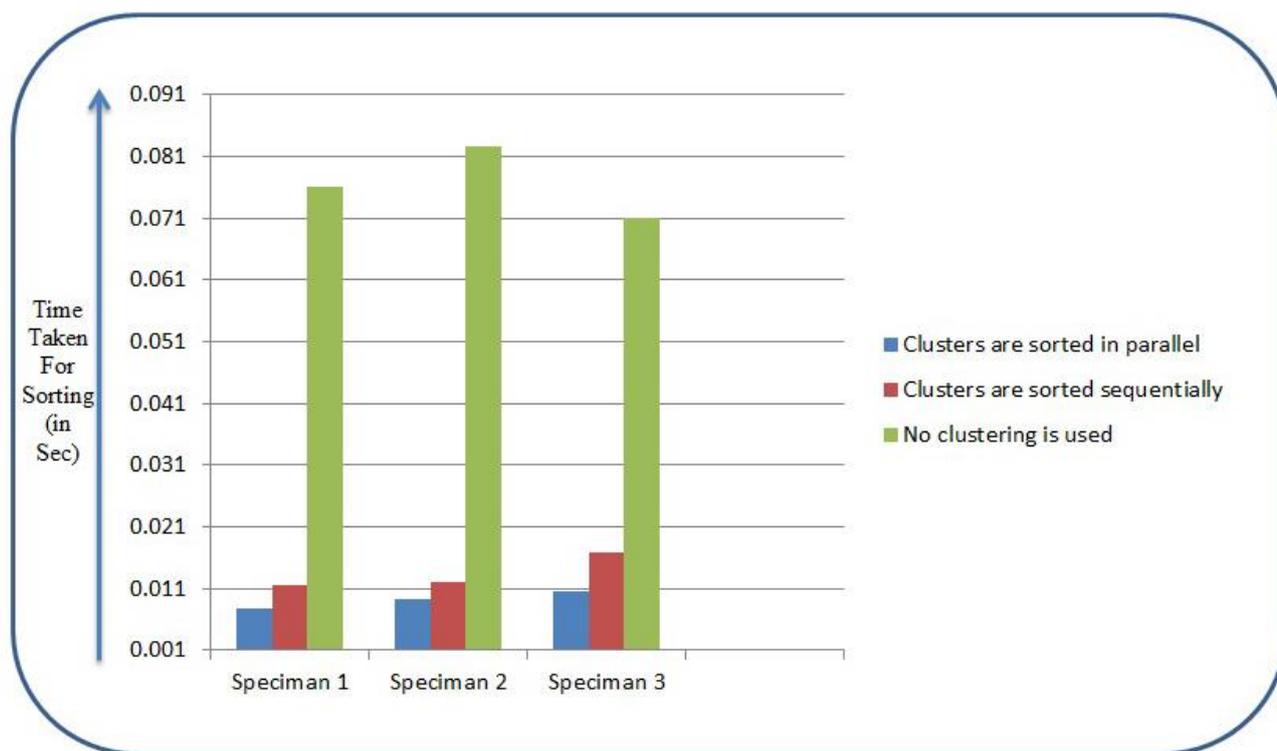


Fig. 3 Experimental Results

VI. CONCLUSION AND FUTURE SCOPE

It can be clearly seen that time taken for sorting is substantially reduced when clustering is used. Radix sort becomes 7-9 times faster when clustering is used and different clusters are sorted in parallel. It becomes 4-6 times faster when clustering is used and clusters are sorted sequentially. Space complexity is also optimized because empty clusters are also removed.

In future, studies could be done by integrating clustering approach with comparison based sorting algorithms like bubble sort, insertion sort, selection sort etc. Also a more large set with heterogeneous values could be taken for sorting in place of our homogeneous data set.

REFERENCES

- [1] https://en.wikipedia.org/wiki/Radix_sort
- [2] Nilsson, Stefan (1 April 2000) "The fastest sorting algorithm" Dr. Dobb's journal 311: 38-45.
- [3] Arne Maus, "ARL, a faster in-place, cache friendly sorting algorithm", in NIK'2002, Norwegian Informatics Conf, Kongsberg, Norway, 2002 (ISBN 82-91116-45-8)
- [4] C.A.R Hoare : Quicksort, Computer Journal vol. 5(1962), 10-15
- [5] Arne Maus," A full parallel radix sorting algorithm for multicore processors", Dept. of Informatics, University of Oslo
- [6] Avinash Shukla , Anil Kishore Saxena, "Modified Pure Radix Sort for Large Heterogeneous Data Set" IOSR Journal of Computer Engineering (IOSRJCE) ISSN 2278-0661 Volume 3, Issue 1 (July-Aug.2012), PP 20-23
- [7] Panu Horsmalahi, "Comparison of Bucket Sort and RADIX Sort" arXiv:1206.3511v1 [cs.DS] 15 Jun 2012
- [8] <https://www.mathworks.com/matlabcentral/fileexchange/45125-sorting-methods/content/Sorting%20Methods/radixsort.m>
- [9] Navjot Kour, "Efficient K-mean Clustering Algorithm Using Ranking method in Data Mining", International Journal of Advance Research in Computer Engineering and Technology
- [10] https://en.wikipedia.org/wiki/k-means_clustering
- [11] https://en.wikipedia.org/wiki/Cluster_analysis