NOVEL DYNAMIC RADIX SORTING ALGORITHM FOR IMPROVING PERFORMANCE

[1] PARESH M TANK, [2] HITESH A PATEL

[1] Lecturer, Computer Engineering Department, B & B Institute of Engineering, VV Nagar-388120, India.
[2] Lecturer, Computer Engineering Department, B & B Institute of Engineering, VV Nagar-388120, India.

pareshtank@gmail.com, hiteshpatelbbit@gmail.com

ABSTRACT— *This Paper Present a new in-place improved radix sort algorithm. The proposed algorithm called improved dynamic radix sort is an efficient and improved algorithm compared to ordinary radix sort. Improved radix sort uses the maximum number available in input array to identify the unused bits of word. This algorithm sort input array using Least Significant Digit radix sort algorithm but instead of sorting with respect to every bit of word, proposed algorithm will sort only with respect to used bits. The proposed algorithm and its runtime and space complexity are discussed in detail. We used counting sort as an in-place sorting algorithm to sort array in each pass. We investigated by implementation the performance of our proposed scheme is far better then ordinary radix sort.*

Keywords— ***sorting, radix sort, counting sort, in-place sorting, LSD, MSD.***

I: INTRODUCTION

Sorting is computational procedure of arranging a collection of data into an ascending order or descending order as per requirement. Sorting is very time-consuming process and time complexity increases as number of data elements increases. Sorting activity is frequently performed in computers today. Over the period, numbers of sorting algorithms have been developed. All These algorithms have different way to sort data. User must select the appropriate algorithm as per need, depending on input and by considering advantage and disadvantage of each.

Sorting algorithm can be classified into sorting by data partitioning and sorting by comparison (Lau, 1992). The distributive portioning sorting algorithm executes in a linear time when the data distribution is uniform. (Dobosiewicz, 1978). The second important class of sorting algorithms is sorting by comparison. The basic operation in such a algorithm is a comparison operation between two data elements using all the bits in data element. Sorting by comparison average case is O(N log N), where N is the input size. An example of sorting by comparison is quicksort (Hoare, 1962).

II: BACKGROUNDS

Radix sorting algorithms have a linear running time. Radix sort uses a subsequence of the key bits, called a component or digit, to directly index the bucket where key belongs. The number of buckets is decided based on based on the digit-size by the algorithm

Radix sorting algorithm fall into two major categories, depending if they process the key forward, from left to right, or backward, from right to left. The forward scanning algorithm is called top-down radix sort, or left radix sort. The back word scanning algorithm is called bottom-up radix sort, or right radix sort.

MSD (Most Significant Digit) is left radix algorithm, which splits the keys into subgroups. The algorithm is applied recursively foe each subgroup separately, with the first digits removed from consideration. After the ith step of the algorithm, the input keys will be sorted according to their first I digit.

LSD (Least Significant Digit) is a right radix algorithm, which splits the keys into groups according to their last digits, with the last digits removed from consideration after ith iteration. LSD is non-recursive and must use a stable loop to rearrange keys. MSD needs only to scan the distinguishing prefixes, while the entire key is scanned in LSD. There is no way to inspect fewer digits in MSD and still be sure that the keys are correctly sorted.

Radix sort divides the word into digits for each data word. Then sort whole input with respect to digits by making as many pass as number of digits. Radix sort internally uses any in-place sorting algorithm that sorts the input data element with respect to given digit.

Radix sort requires d(digits) number of passes. Each pass takes O(n+k) if we use any linear time algorithm. So average case execution time of radix sort is O(d(n+k)) where n is number of input element and k is maximum number occurs in a input.

III: PROPOSED ALGORITHM

Algorithm of improved radix sort is given below. Algorithm uses the counting sort to sort the number in pass. First the maximum number available in input are found out. Depending on maximum number, how many digits of word would require to sort are found out.

So proposed algorithm utilizes the fact that those digit that contains all zero bits are not required to sort. By sorting according to that digit, we wasting valuable time and space.

In sorting, most circumstances have numbers of data to sort are quite larger than the maximum number available in them. So most of Most Significant bits of word are zero. It gives the advantage of not to sort that digits.

PSEUDO CODE FOR IMPROVED RADIX SORT

Step-1 find out the maximum number presents in the input.
For(all data elements)
If (data element >max)
Max=data element
Step-2 find out the number of bits would be require to sort by considering maximum number presents in input bits=$\log_2$(max).
Step-3 find out the number of digits that would be required to sort by considering digit size and bits
Digits=bits/digits-size
Step-4 Extract the least digit from all input data.
Step-5 call a in-place linear time sorting algorithm (Counting sort) to sort the input data with retrieved digits
Step-6 repeat Step-4 up to all the digits is considered.

Counting Sort (A,B,k)
Step-1 Determine the number of elements less than x, for each input x
for (i=0 to k)
do C[j]=0
for (j=1 to n)
do C[A[j]]=C[A[j]]+1
for (j=1 to k)
do C[j]=C[j]+C[j-1]
Step-2 Place x directly in its position.
for(j=n down to 1)
do B[C[A[j]]]=A[j]
C[A[j]]=C[A[j]]-1

IV: EXPERIMENTAL RESULTS

Algorithm tested by giving 10 million number as input. All the tests performed take the average of ten runs. The results display the run time of input size 10 million and different digits size. Machine used in testing is Pentium IV with 2GB of RAM having Fedora 10 platform.

Execution time was monitored using getitimeofday () function available in Linux OS. Function was called just before and after sorting procedure executed. Then time difference of both times has been considered

TABLE I

pass	Ordinary Radix sort (B=32, Max=255)		
	Execution Time	Input(n)	Extra space used
1	0 seconds,622569 µs	10000000	16 GB
2	1 seconds,179393 µs	10000000	256KB
4	1 seconds,542625 µs	10000000	1 KB
8	2 seconds,842635 µs	10000000	64 Bytes
16	5 seconds,701468 µs	10000000	16 Bytes
32	11 seconds,51109 µs	10000000	08 Bytes

TALBE II

pass	Dynamic Radix sort (B=32 Max=255)		
	Execution Time	Input(n)	Extra space used
1	0 seconds,422415 µs	10000000	1 KB
2	1 seconds,743225 µs	10000000	64 Bytes
4	1 seconds,402713 µs	10000000	16 Bytes
8	2 seconds,719055 µs	10000000	08 Bytes

TALBE III

pass	Dynamic Radix sort (B=32 Max=65535)		
	Execution Time	Input(n)	Extra space used
1	0 seconds,782559 µs	10000000	16 GB
2	1 seconds,801847 µs	10000000	256KB
4	1 seconds,416214 µs	10000000	1 KB
8	2 seconds,724348 µs	10000000	64 Bytes
16	5 seconds,384252 µs	10000000	16 Bytes

Here all tests are performed on input of constant size i.e. 10 million number.

First column indicates the different value of digits on which sort has been performed. Second column depict the execution time of the algorithm. Last column indicates the extra space would be required to sort by counting sort.

Table I shows the results when sorting has been performed using ordinary radix sort. In this case inputs have the maximum number 255.

Table II shows the results when sorting has been performed using dynamic radix sort. In this case inputs have the same maximum number of 255.

Table III shoes the results when sorting has been performed using dynamic radix sort. In this case inputs have the maximum number of 65535.

Results are quite visual from each table. If maximum number presents in input is a small then it would make very big difference. Even it is not small then also a marginal difference will be present.

So, it will be very astonishing to use this novel idea in sorting process. We can reduce the execution time as well as memory required by using the dynamic radix sort. Theoretically it has same complexity as ordinary radix sort.

IV: CONCLUSIONS

In this paper we present the new “dynamic radix sort” which reduces the overall execution time of sorting process and the space required. The algorithm was implemented and tested. Different cases were given to input algorithm. It can be clearly seen that algorithm works as desired.

It is also observed that digit size should not be too small or to big otherwise it will degrade the performance of the algorithm. when digit size is too small it will increase the passes requires to sort and so the execution time’

If digit size is too big it will reduce the number of passes the algorithm will take but at expense of the large space requirement.

REFERENCES

[1] Khreisat Laila, “QuickSort A Historical Perspective and Empirical Study”, IJCSNS International Journal of Computer Science and Network Security, VOL.7 No.12, December 2007

[2] Sharma Vandana, Singh Satwinder and Kahlon K. S.,“Performance Study of Improved Heap SortAlgorithm and Other Sorting Algorithms on

Different Platforms”, IJCSNS International Journalof Computer Science and Network Security, VOL.8 No.4, April 2008

[3] Amer Al-Badarneh, Fouad El-Aker, “Efficient Adaptive In-Place Radix Sorting”, Informatica, 2004, Vol. 15, No. 3, 295–302, 2004 Institute of Mathematics and Informatics, Vilnius

[4] Islam Tarique Mesbaul, Kaykobad M., “Worst-case analysis of generalized heapsort algorithm revisited”, International Journal of Computer Mathematics, Vol. 83, No. 1, January 2006, 59–67

[5] Bratbergsengen Kjell, “Hashing Methods And Relational Algebra Operations”

[6] Khurana Udayan, “Decision Sort and its Parallel Formulation”

[7] Maus Arne, “Sorting by generating the sorting permutation, and the effect of caching on sorting”

[8] Astrachan Owen, “Bubble Sort: An Archaeological Algorithmic Analysis”

[9] Shakhnarovich Gregory, Viola Paul, Darrell Trevor, “Fast Pose Estimation with Parameter Sensitive Hashing”

[10] Nevalainen Olli, Raita Timo, Thimbleby Harold, “An improved insert sort algorithm”

[11] Chen Jing-Chao, “Quick Sort on SCMPDS”, Journal Of Formalized Mathematics, Volume 12, Released 2000, Published 2003, Inst. of Computer Science, Univ. of Białystok

[12] Franceschini1Gianni, Muthukrishnan S., Patrascu Mihai, “Radix Sorting With No Extra Space” Aho, A., J. Hopcroft and J. Ullman (1974). The Design and Analysis of Computer Algorithms. Addison- Wesley (Reading, Massachusetts).

[13] Anderson, A., and S. Nilsson (1998). Implementing radixsort. ACM Journal of Experimental Algorithmics, 3(7).

[14] Dobosiewicz, W. (1978). Sorting by distributive partition. Information Processing Letters, 7(1), 1–6. Hoare, C. (1962). Quicksort. Computer Journal, 5(1), 10–15.

[15] Knuth, D.E. (1973). The Art of Computer Programming, vol. 3, 2nd. ed. Addison-Wesley (Reading, Massachusetts).

[16] Lau, K.K. (1992). Top-down synthesis of sorting algorithms. The Computer Journal, 35, A001–A007. Maus, A. (2002). ARL: a faster in-place, cache friendly sorting algorithm. In Norsk Informatik konferranse NIK’2002. pp. 85–95.

[17] McIlroy, P., K. Bostic and M. McIlroy (1993). Engineering radix sort. Computer Systems, 6(1), 5–27. Mehlhorn, K. (1984). Data Structures and Algorithms 1: Sorting and Searching. Springer-Verlag. Sedgewick, R. (2003). Algorithms in Java, Parts 1–4, 3rd. ed. Addison-Wesley (Reading, Massachusetts).

[18] Sedgewick, R., and P. Flajolet (1996). An Introduction to the Analysis of Algorithms. Addison- Wesley (Reading, Massachusetts).

[19] Dr Anupam Shukla and Rahul kala Predictive Sort. IJCSNS International journal of computer science