INFORMATICA, 2004, Vol. 15, No. 3, 295–302
© 2004 *Institute of Mathematics and Informatics, Vilnius*

Efficient Adaptive In-Place Radix Sorting

Amer AL-BADARNEH
Computer Information Systems Department, Jordan University of Science and Technology
P.O. Box 3030, Irbid 22111, Jordan
e-mail: amerb@just.edu.jo

Fouad EL-AKER
Computer Science Department, New York Institute of Technology
P.O. Box 940650, Amman 11194, Jordan
e-mail: elaker_fouad@yahoo.ca

Received: January 2004

Abstract. This paper presents a new in-place pseudo linear radix sorting algorithm. The proposed algorithm, called MSL (Map Shuffle Loop) is an improvement over ARL (Maus, 2002). The ARL algorithm uses an in-place permutation loop of linear complexity in terms of input size. MSL uses a faster permutation loop searching for the next element to permute group by group, instead of element by element. The algorithm and its runtime behavior are discussed in detail. The performance of MSL is compared with quicksort and the fastest variant of radix sorting algorithms, which is the Least Significant Digit (LSD) radix sorting algorithm (Sedgewick, 2003).

Key words: sorting, radix sort, quicksort, straight radix, in-place sorting, LSD, MSD, ARL, MSL.

1. Introduction

Sorting is a computational process of arranging a collection of data items into a predetermine order and it is the most heavily activity performed frequently in computers. This subject has been investigated and studied for many years and several books are written on this subject (Aho *et al.*, 1974; Knuth, 1973; Mehlhorn, 1984; Sedgewick and Flajolet, 1996). Sedgewick in his book "Algorithms in Java" (Sedgewick, 2003) presents a thorough, up-to-date treatment of the entire topic of sorting algorithms.

Sorting algorithms can be classified into sorting by data partitioning and sorting by comparison algorithms (Lau, 1992). Data partitioning sorting algorithms basic operation is mapping a key value to the subset it belongs to. The distributive partitioning sorting algorithm (Dobosiewicz, 1978) is the first data partitioning sort algorithm. The distributive partitioning sorting algorithm executes in linear runtime when the data distribution is uniform.

Radix sorting algorithms (McIlroy *et al.*, 1993) have a pseudo linear running time. They use a subsequence of the key bits, called a component or a digit, to directly index the bucket where the key belongs. The number of buckets is decided based on the *digit-size* by the algorithm. The number of buckets is the same as the number of integers representable by the bits making up the digit.

The second important class of sorting algorithms is sorting by comparison. The basic operation in such algorithms is a comparison operation between two different keys, using all the bits in the two keys. 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).

Quicksort is said to sort adaptively because it selects the pivot from the input data array. By comparison, adaptive radix sorting algorithms set the *digit-size* according to the input array size or a subgroup size, at least once. Adaptive radix sorting algorithms may use the same *digit-size* throughout the sorting process, or vary the *digit-size* per subgroup.

The proposed algorithm MSL (Map Shuffle Loop) is an adaptive radix sorting algorithm, which uses a possibly different *digit-size* for different calls. The rest of the paper is organized as follows. Section 2 gives a background of radix sorting algorithms. Section 3 describes MSL algorithm. In Section 4, we present the analytical comparison of MSL and its competitive algorithms. Section 5 presents experimental comparison results. Finally, Section 6 gives conclusion and future work.

2. Background

Radix sorting algorithms fall into two major categories, depending if they process the keys 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 backward scanning algorithm is called bottom up radix sort, or right radix sort.

MSD (Most Significant Digit) is a left radix algorithm, which splits the keys into subgroups. The algorithm is applied recursively for 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 digits.

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 the ith iteration. LSD is non-recursive and must use a stable loop to rearrange the 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.

ARL (Maus, 2002) is a new in-place left radix sorting algorithm. It removes the extra space requirement of MSD and uses a different loop than MSD to rearrange the keys. This loop is called the permutation loop in ARL. ARL also uses an adaptive *digit-size*.

LSD does not partition the input array and therefore it cannot call a different sorting algorithm for small subgroups. Switching to insertion sorting is performed to achieve speed up. Insertion sorting is faster than radix sorting algorithms for small size arrays. We use the size of 25.

3. The MSL Algorithm

MSL is a modification of the ARL algorithm. ARL permutation loop inserts each key into its destination group. MSL permutation loop searches for the next element to insert group by group, instead of one element at the time, as ARL.

ARL uses a loop of N steps to find the next element to permute, while MSL uses a loop of K steps to find the next element to permute, where N is the input array size and K is the number of groups.

The *digit-size* passed to MSL is used by an initial step that computes the shift value and the mask value. The shift value is used for right shifting keys. The mask value is used to extract the group (bucket) number from the key. Masking is done after shifting. On recursive calls, the mask value changes, since we use an adaptive *digit-size*, while the shift value is always decreased in recursive calls, in order to get the next digit.

MSL permutation loop places keys into their respective subgroups, where the start and the end addresses of each subgroup are already calculated. During the permutation loop, the left most subgroup that does not have all its key elements inserted is called the *origin-group*.

The first element in the *origin-group* is selected by MSL to root a permutation cycle (a cycle of exchanges). Every permutation cycle begins and ends at a *current-root-key*, which is the first element in the *origin-group* not in its correct position. At each step of the permutation cycle, an exchange takes place. The current key is exchanged with the element that is in the correct position of the current key. Initially the current key is the same as the *current-root-key*.

A condition tests if the current key destination address is in the *origin-group*. The array element at the root key initial address is not valid once the permutation cycle has started. Therefore, the current key and the element at the root key initial address cannot be exchanged, when the above condition is true.

At this time, a permutation cycle is done, and the algorithm attempts to find a new *origin-group*, to restart a new permutation cycle. If the search for a new *origin-group* fails, the permutation loop terminates (Program 1).

4. Analytical Study

MSL uses extra space for groups (buckets). For each group, MSL computes and stores the boundary or the limit addresses. Each group has a lower limit address, and an upper limit address. The group's sizes are computed beforehand and are used to compute group boundary addresses. No extra space is used for the group's sizes, where the upper limit address array is used temporarily instead for this purpose. The formula used for computing the number of groups is $2^{digitsize}$. The *digit-size* in bits parameter is set prior to any call to MSL.

The ARL and the MSL algorithms contribute to radix sorting algorithms, by adding a new category, which we call in-place radix sorting. LSD and MSD are not in-place.

Program 1: Pseudo code for MSL algorithm

```
Initial Step:  Compute shift and mask values

Step 1:        Initialize boundary addresses to Zero
               for (all used buckets k){
               group_lower_bound[k] = group_upper_bound[k] = 0;}

Step 2:        Compute group sizes
               for (all input array elements, i){
               // Use the upper address memory cell for each bucket to hold the size
               // information. This step also keeps track of the smallest used bucket
               // number (minGroup) and the largest used bucket number (maxGroup)
               // for speed up reasons.
                      int k = (InputArray[i] »> bkt_shiftValue) & bkt_maskValue);
                      group_upper_bound[k]++;
                      if (k > maxGroup) maxGroup = k;
                      if (k < minGroup) minGroup = k;}

Step 3:        Compute group lower and upper bounds
               int i_start = bkt_left ;
               for (group k, in range of groups : minGroup to maxGroup){
               // group_lower_bound[k] point to group k start address
                      group_lower_bound[k] = i_start;
               $//group\_upper\_bound[k]pointtogroupk + 1startaddress$
                      i_start += group_lower_bound[k];
                      group_upper_bound[k] = i_start;}

Step 4.1:      Set originGroup initially to minGroup

Step 4.2:      Update origin-group
               while(originGroup < maxGroup && group_lower_bound[originGroup] ==
               group_upper_bound[originGroup])
                      originGroup++;                    //advance originGroup

Step 4.3:      if(originGroup == maxGroup) go to Step 5;

Step 4.4:      Set information for initial key
               current_key=initial_key=InputArray[group_upper_bound[originGroup] – 1];
               // Compute address of initial key
               initial_key = group_upper_bound[originGroup] – 1;
               // Compute destination address of current key
               current_key = group_upper_bound[dest_group(initial_key)] – 1;

Step 4.5:      Permutation/rearrangement loop
               exchange(current_key, InputArray[dest_address(current_key)];
               Compute destination group of current_key
               // Compute destination address of current_key
               current_key = –group_upper_bound[dest_group(current_key)];
               if(dest_address(current_key) == dest_address(initial_key))
                      then go to Step 4.2               // to find a new origin-group
               else repeat Step 4.5;

Step 5:        Process subgroups
               For all subgroups with size less than 25, use insertion sort,
               otherwise call MSL recursively. Use the lower bounds array to
               get group’s left and right addresses (bound addresses).
```

Table 1

A general comparison

Algorithm	**In-place**	**Stable**	**Recursive**	**Adaptive**
LSD	No	Yes	No	No
MSD	No	No	Yes	No
ARL	Yes	No	Yes	Yes
MSL	Yes	No	Yes	Yes

A general comparison of MSL with the three competitive radix sorting algorithms LSD, MSD, and ARL is shown in Table 1.

MSD and LSD are using N extra space. ARL and MSL solve the extra space requirement of LSD and MSD. ARL and MSL use extra space for holding groups information and do not need extra space for holding any keys. MSL is an in-place sorting algorithm, like ARL, however it uses more extra space, as the initial *digit-size* used in MSL is larger than *digit-size* s used by ARL. This extra space is small compared to the input array size. ARL and MSL set the *digit-size* adaptively to save time.

Out of the four algorithms, only the LSD algorithm is stable. Otherwise, the LSD algorithm will not sort the input array correctly. Unlike LSD, the algorithms MSD, ARL, and MSL recursively sort subgroups and may use simpler methods for this task when a subgroup size is small. Using simpler methods to sort small subgroups is used often to save time.

Regarding the time analysis, MSL is expected to be faster than ARL because MSL uses a faster permutation loop. The pointer to the next array element to use when restarting the permutation loop is advanced group by group instead of element by element. Sorting an array with N keys and K groups, ARL uses a loop of N steps to locate the next element to permute, while MSL uses a loop of K steps. The worst case for K is N, and the best case is 1. The number of groups is usually much smaller than the input size therefore the time saved by MSL is an order of N.

5. Experimental Results

All the tests performed take the average of five runs. The results display the run time for input sizes 500 000 to 4 000 000. We have also performed initial tests that show that the same algorithm gives the same results, within a small percentage, when ran against many arrays with the same characteristic distributions. The machine used in the tests is Pentium III, with 128 MB RAM.

The *digit-size* value used by MSL is set before every call to MSL. The *digit-size* setting depends initially on the input size N and on group sizes after the initial call. The initial *digit-size* is set differently in the algorithm. The *digit-size* value used on the initial call is 15 for MSL. The data sorted is 31-bits Java positive integers. The following is the code used for setting the *digit-size* in the MSL algorithm.

```
if (group-size $\leqslant 25$)            call insertion sort;
else if (group-size $\leqslant 500$)      DIGITSIZE = 4;
else if (group-size $\leqslant 5000$)     DIGITSIZE = 8;
else                                      DIGITSIZE = 12.
```

Table 2 compares LSD, MSL, as well as quicksort. Quicksort is included because it is a popular fast sorting method used commonly in computer applications. The java *Arrays.sort*() method, which is a tuned quicksort, is used in the tests. We note from these results that MSL is better than the two algorithms, LSD and tuned Java quicksort. U/x means that the uniform distribution is on the range U[0...Max_Java_Int/x]. We used different uniform distributions with varying density, since the run time of MSL is affected by the range of values. We use these distributions described because any distribution in general can be expressed as a mix of distributions U/x. For example, the input array can be divided into z sub-lists, each filled with data from one of the distributions U/x. In

Table 2

A comparison shows MSL outperforms LSD & Java tuned quicksort

Size$\times 10^6$	Distr.	0.5	1	1.5	2	2.5	3	3.5	4
LSD		372	770	1156	1550	1946	2340	2734	3130
QSort	U/1	614	1352	2102	2868	3670	4460	5266	6064
MSL		328	724	1064	1428	1758	2144	2526	2880
LSD		372	746	1110	1480	1844	2240	2616	2988
QSort	U/2	658	1352	2118	2876	3634	4450	5250	6054
MSL		320	628	948	1300	1658	2020	2418	2822
LSD		330	660	1002	1338	1704	2034	2384	2702
QSort	U/10	670	1360	2112	2866	3626	4448	5248	6046
MSL		272	646	958	1078	1360	1628	1900	2176
LSD		304	600	912	1210	1526	1856	2164	2482
QSort	U/100	604	1364	2144	2892	3646	4406	5206	6208
MSL		230	502	836	1118	1382	1648	1900	2166
LSD		306	604	902	1204	1494	1810	2088	2426
QSort	U/1000	636	1330	2064	2778	3538	4272	4998	5734
MSL		276	520	826	1154	1506	1814	2110	2372
LSD		300	590	890	1196	1486	1804	2086	2410
QSort	U/10000	592	1220	1832	2408	3024	3614	4208	4822
MSL		296	582	868	1110	1428	1702	2000	2294
LSD		252	518	746	988	1264	1504	1792	2022
QSort	U/100000	452	978	1428	1924	2438	2968	3440	3924
MSL		200	416	646	880	1100	1318	1562	1770
LSD		198	406	614	808	988	1240	1482	1670
QSort	U/1000000	372	750	1152	1528	1934	2362	2780	3196
MSL		230	374	550	750	936	1098	1296	1486

general, the fact that MSL performs better than other algorithms on these distributions is quite valuable.

6. Conclusion and Future Work

Well known radix sorting algorithms, MSD and LSD are not in-place. A new radix sorting category is added, this category is in-place radix sorting algorithms, which includes the ARL and the MSL algorithms. The run time of the in-place radix sorting algorithms is as fast as other radix sorting algorithms. From the results, MSL is faster than all the algorithms compared against, for the test cases, and array sizes used.

In addition, the followings are two important tradeoffs to remember. When setting the *digit-size* as a function of the size of subgroups, it is good to select the *digit-size* as small as possible, since the larger the *digit-size* is, the more time MSL takes. On the other hand, setting the *digit-size* to a higher value is also desirable. This is because the partitioning may not be enough, and generated subgroups may still require more partitioning. In this case, MSL body execution is repeated once per group. The adaptive *digit-size* setting has to be set accurately compromising the MSL iteration speed and amount of partitioning.

Future work on in-place radix sorting algorithms includes: (1) Designing and testing new in-place radix sorting strategies. In-place radix sorting algorithms are important since very fast sorting is done with no extra space requirements. (2) A comparison of MSL against ARL should be done. ARL is a new algorithm and its code is not available at the time when this paper is published. Later work will compare ARL and MSL.

References

Aho, A., J. Hopcroft and J. Ullman (1974). *The Design and Analysis of Computer Algorithms*. Addison-Wesley (Reading, Massachusetts).

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

Dobosiewicz, W. (1978). Sorting by distributive partition. *Information Processing Letters*, **7**(1), 1–6.

Hoare, C. (1962). Quicksort. *Computer Journal*, **5**(1), 10–15.

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

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.

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).

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

A. Al-Badarneh received his BSc degree (1987) in computer science from Yarmouk University (Jordan), MSc (1995) and PhD (1999) degrees in computer science from Wayne State University (USA). He worked in the University of Jordan (1990–1992) as a teaching assistant at the Department of Computer Science. In May 2000, he joined the Department of Computer Science and Information Systems at Jordan University of Science and Technology, Jordan, where he is currently an assistant professor and a chairman of the Department of Computer Information Systems. His research interests include: data mining, sorting and graph algorithms, and parallel heuristic search

F. El-Aker received his BSc degree (1986) in computer science from Kuwait University (Kuwait), MSc (2003) honors degree in computer science from N.Y.I.T. (USA). He worked at several software engineering and internet companies in development and design of software. His current research interests are sorting, divide and conquer algorithms, building fast efficient software, graph algorithms, TSP, and parallel algorithms.

Adaptyvus skiltinio rikiavimo algoritmas

Amer AL-BADARNEH, Fouad EL-AKER

Pateikiamas naujas skiltinio rūšiavimo (rikiavimo) algoritmas, pavadintas MSL, kurį taikant daugelį kartų surikiuojamas duomenų rinkinys. Jis yra pseudotiesinis. Tai žinomo skiltinio rikiavimo algoritmo, vadinamo ARL, kuriame pradedama rikiuoti nuo vyriausiosios skaičiaus skilties, modifikacija, gauta jungiant perkeliamus duomenų elementus į grupes. Dėl to MSL algoritmas spartesnis už kitus žinomus algoritmus. Pateikiamas efektyvumo palyginimas su kitais panašiais algoritmais esant įvairiems rikiuojamo duomenų rinkinio dydžiams.