INTERNATIONAL JOURNAL OF CIRCUITS, SYSTEMS AND SIGNAL PROCESSING
DOI: 10.46300/9106.2020.14.39 Volume 14, 2020

Bounds on Complexity when Sorting Reals

Marcel Jiřina,
Institute of Computer Science, Academy of Sciences of the Czech Republic,
Pod Vodárenskou věží, Prague 8, 18207
Czech Republic

Received: March 13, 2020. Revised: May 28, 2020. Accepted: June 30, 2020. Published: July 2, 2020.

Abstract- We derive the upper bounds on the complexity of the counting sort algorithm applied to reals. We show that the algorithm has a time complexity O(n) for n data items distributed uniformly or exponentially. The proof is based on the fact that the use of comparison-type sorting for small portion of a given data set is bounded by a linear function of n. Some numerical demonstrations are discussed.

Keywords- Linear time, Sorting reals, Time complexity

I. Introduction

MANY sorting algorithms have existed since 1954 [8] and among them the bucket sort and counting sort work with linear time complexity but for integer sorting keys. Usual cost for linear time complexity is linear space complexity, i.e. some array, ev. arrays are needed.

Problem of fast sorting with noninteger (real, float) sorting keys is dominated by the quicksort algorithm [3] that exists in countless versions. The big advantage of the quicksort is a small and constant space complexity. There is also the proof that for rational numbers generally the best mean time complexity of the sorting based on comparison is given by $n \log_2 n$.

However, a fast algorithm for sorting reals in a linear time is not known. Here we prove linear time complexity for the counting sort algorithm modified for sorting reals.

When ranking reals in the counting sort, it often happens that several items sorted fall into one cell. These cases form groups. The correct sorting of items in a group is solved by a comparison sort, especially the quicksort.

We prove here the linear time complexity of this algorithm for reals with uniform and exponential distributions. The proof is based on the fact of very fast convergence of the number of groups to zero with their size. That is why it is possible to bound from above the time complexity of sorting all the groups by a constant multiplied by the total number of items sorted n.

At the same time, we will show how to use this algorithm for heavy-tailed distributions including the Cauchy distribution of items sorted.

II. Background

A. Problem formulation

Let there be a vector random variable $\Psi = \{\psi_1, \psi_2, ..., \psi_n\}$, where ψ_i are independent and indentically distributed random variables with a given distribution.

Let $x = \{x_1, x_2, ..., x_n\}$ be a fixed sample (realization) of Ψ.

The question is if x, considered a sequence, can be sorted in a time proportional to n.

B. Sorting algorithms

B..1 Comparison sort

Definition 1 *The comparison sort is a procedure that uses the only operation*

$$if \quad x_i > x_j \quad and \quad i < j \quad then \quad t = x_i, \quad x_i = x_j, \quad x_j = t, \tag{1}$$

where t is a temporary variable.

Theorem 1 *The best mean sorting time of the comparison sort is proportional to*

$$n \ln_2 n \tag{2}$$

Proof see [1].

B..2 Counting sort

The counting sort is based on the idea to use integer keys directly as ranks of items sorted. It works by creating an integer array of size S and using individual bins to count the occurrences of items in the input. Each item is then counted by incrementing the value of its corresponding bin in S. Afterwards, the counting array is looped through to arrange all of the inputs in order.

The algorithm uses only simple loops of fixed length, without recursion or subroutine calls.

This sorting algorithm is extremely fast and has a very good asymptotic behavior as n increases. It also can be modified to provide stable behavior [1].

B..3 Assignment sort

The simplest version of the counting sort uses assumption that the minimal difference between any two items is nonzero and known as Δ.

Algorithm 1 *(The assignment sort, ev. the direct counting sort.) Let for sample x:*

1) *there exist Δ such that for any two x_i, $x_j \in x$, $|x_i - x_j| < \Delta$, and let integer $S \geq (x_M - x_m)/\Delta$, where $x_M = \max(x_{1,2}, ..., x_n)$, $x_m = \min(x_{1,2}, ..., x_n)$.*
2) *there be S integer variables (bins) $c_1, c_2, ..., c_S$, initially $c_i = 0$.*
3) *for $x_i, i = 1, 2, ..., n$*

$$r_i = \lfloor 1 + (S-1)\frac{x_i - x_m}{x_M - x_m} \rfloor, \tag{3}$$

and c_{r_i} be incremented by one.

4) *for $c_i, i = 2, 3, ..., S$ let $c_i = c_i + c_{i-1}$, and $c_1 = 1$.*
5) *for $i = 1, 2, ..., n$ $y_{c_{r_i}} = x_i$.*

Then $y = \{y_1, y_2, ..., y_n\}$ is the sorted sequence x.

This algorithm is very fast with only n data moves, i.e. one data move per item. Supposing Δ is known, then each data item is read only twice, once to give the maximum and minimum, and then to get rank r_i. Unfortunately, it is difficult to state the Δ, and, moreover, in case of reals it can be extremely small.

B..4 The counting sort for reals

This algorithm allows that a single bin is assigned to several items. These items are unsorted and appear in the same order as they are in the original sequence x. These groups of items have to be sorted. Because individual items are reals, a comparison sort has to be used for these groups.

Algorithm 2 *(The counting sort for reals.) Let for sample x:*

1) *there be n integer variables (bins) $c_1, c_2, ..., c_n$, initially for each i $c_i = 0$.*
2) *for $i = 1, 2, ..., n$ rank r_i is computed according to*

$$r_i = \lfloor 1 + (n-1)\frac{x_i - x_m}{x_M - x_m} \rfloor, \tag{4}$$

where $x_M = \max(x_{1,2}, ..., x_n)$,
$x_m = \min(x_{1,2}, ..., x_n)$, and c_{r_i} be incremented by one.

3) *for $c_i, i = 2, 3, ..., n$ let $c_i = c_i + c_{i-1}$, and $c_1 = 1$.*
4) *for $i = 1, 2, ..., n$, $y_{c_{r_i}} = x_i$ and c_{r_i} be incremented by one.*
5) *for $i = 1, 2, ..., n-1$ and $c_{i+1} - c_i > 1$ sort $y_{c_i}, y_{c_i+1}, ..., y_{c_{i+1}}$ using any comparison sort.*

Then $y = \{y_1, y_2, ..., y_n\}$ is the sorted sequence x.

Note 1 *We call a multiplicity the number of items that have the same value of r_i according to step 4, i.e. they are assigned to the same bin. These items (a group) are sorted by a comparison sort according to step 5). We denote the multiplicity by k.*

The Algorithm breaks down data to small and easy to handle groups. Thus, it may remind the replicated subtree problem, or the scaling down data [5], eventually the k shortest paths problem [6].

III. Results

A. The computational complexity

In the Algorithm 2 there are several loops, all of them of length n.

Step 5) is important. The loop here contains an inner comparison sort that runs in cases of multiple items having the same r_i.

A..1 The uniform distribution

Supposing uniform distribution of $x_i, i = 1, 2, ..., n$, we can estimate the probability of zero, 1, 2, ... items in a bin. This is given by binomial distribution and it decreases rapidly with the size k of the group, i.e. with the number of items assigned to a bin.

Theorem 2 *Let $x_i, i = 1, 2, ..., n$ be independent and uniformly distributed. Then, Algorithm 2 has time complexity $O(n)$.*

Proof. In step 5), there are groups of length k equal to two or more (multiplicity). These groups appear randomly. Under assumption of the uniform distribution of random variables x_i this is the Bernoulli trial. Then the probability of appearance of a group of multiplicity k (0, 1, 2, ... n) is given by

$$P_k = \binom{n}{k} p^k (1-p)^{n-k} \tag{5}$$

The mean total number of items in bins with multiplicity k is

$$n_k = n\binom{n}{k} p^k (1-p)^{n-k} \ , \tag{6}$$

where

$$p = \frac{1}{n} \tag{7}$$

Ratio n_k/k gives the number of groups with multiplicity k. The worst case complexity for sorting them is proportional to k^2 (e.g. when using the bubble sort). The time needed to sort all groups of size k is

$$\frac{n_k}{k} k^2 = k n_k \ . \tag{8}$$

The time needed to process all groups of all sizes is proportional to

$$C_e = \sum_{k>0} k n_k \tag{9}$$

(There is $\sum_{k\geq 0} k = n$.) Empty groups ($k = 0$) mean no effort, in groups with multiplicity $k = 1$ the time complexity is proportional to their number.

We show that cases of k equal to 2 and more represent together time complexity proportional to n. There is

$$C_{e2} = \sum_{k=2}^{n} k n_k.$$

After substitution of (6)

$$C_{e2} = \sum_{k=2}^{n} k n P_k \ ,$$

$$C_{e2} = \sum_{k=2}^{n} k n \binom{n}{k} p^k (1-p)^{n-k} \ .$$

Then

$$C_{e2} = \sum_{k=2}^{n} k n \binom{n}{k} \frac{1}{n^k} \left(1 - \frac{1}{n}\right)^{n-k} \ .$$

It holds

1.
$$\binom{n}{k} \leq \frac{n^k}{k!} < (\frac{n.e}{k})^k$$
[9] (e=2.71828...)

2.
$$(1 - \frac{1}{n})^{n-k} \leq 1 \tag{10}$$
for $k = 1, 2, ..., n$

We have

$$C_{e2} \leq n \sum_{k=2}^{n} k (\frac{n.e}{k})^k \frac{1}{n^k} (1 - \frac{1}{n})^{n-k}$$

$$= n \sum_{k=2}^{n} k^{1-k} e^k (1 - \frac{1}{n})^{n-k}$$

$$\leq n \sum_{k=2}^{n} k^{1-k} e^k = nW \ .$$

Then we have to show that $W = \sum_{k=2}^{n} k^{1-k} e^k$ is finite.We can see the series

$$\{k^{1-k} e^k\} \tag{11}$$

as a geometric series with variable quotient $q = \frac{e}{k} (\frac{k}{k+1})^k \leq \frac{e}{2k}$. There exists $k_0 > e/2$ such that $q_0 = \frac{e}{2k_0} < 1$. Then

$$R = \sum_{k=2}^{k_0 - 1} k^{2-k} e^k = const$$

and

$$k^{1-k} e^k < k_0^{1-k_0} e^{k_0}$$

for $k > k_0$. Then the series

$$\{k_0 (\frac{e}{k_0})^{k_0} q_0^{k-k_0}\} \tag{12}$$

with constant quotient q_0 and the first term

$$a_1 = k_0 (\frac{e}{k_0})^{k_0}$$

is convergent and maximizes series (11). The maximizing series

$$\{a_1 q_0^k\}$$

where $k = 1, 2, ...$, has the sum equal to

$$S = k_0 {q_0}^{k_0} \frac{1}{1 - q_0} \tag{13}$$

and there is

$$C_{e2} < (R + S)n \ . \tag{14}$$

for $k = 1, 2, ..., n$ This finishes the proof.

Note 2 In the case of more effective sorting algorithm for multiplicities with time complexity $k \log_2 k$ in (9) the k changes to $\log_2 k$ and the same appears in numerator of (13) in the form

$$S = \log_2 k_0 {q_0}^{k_0} \frac{1}{1 - q_0} \tag{15}$$

A..2 Exponential distribution of items sorted

For the exponential distribution of $x_i, i = 1, 2, ..., n$ we must use the Poisson binomial distribution [2], [4].

Theorem 3 *Let sorting keys be exponentially distributed. Then, Algorithm 2 has time complexity $O(n)$.*

Proof. As in the previous proof the step 5) is essential. Under assuption of the exponential distribution of items sorted this is the Bernoulli trial but for each k with diffrerent probability [10]. Let p_i be the probability of success in the ith trial. Then, there is a formula for upper bound for probability mass function P_k of groups of size k. The Chernoff bound of the probability that a Poisson binomial distribution gets large is given by [2]

$$Pr(S > k) \leq \exp\left(-kt + \sum_{i=1}^{n} \ln(p_i(e^t - 1) + 1)\right), \tag{16}$$

where $t = \ln(k/\sum_{i=1}^{n} p_i)$. Substituting, we get

$$Pr(S > k) \leq \exp\left(-k \ln \frac{k}{v} \sum_{i=1}^{n} \ln(p_i(\frac{k}{v} - 1) + 1)\right) \tag{17}$$

and

$$Pr(S > k) \leq \exp\left(k - v - k \ln \frac{k}{v}\right) \tag{18}$$

with $v = \sum_{i=1}^{n} p_i$. One can find that the last form the Chernoff bound (18) is independent of n. Under the assumption that $\sum_{i=1}^{n} p_i = 1$ there exists function of k

$$C_\infty(k) = \exp(k(1 - \ln k))/e \tag{19}$$

maximizing (17) independently of n.

The maximal number of groups of a given size k are groups with $k = 2$. Items in groups of this size are either left as they are or exchanged. This represents the time complexity proportional to the total number of samples in groups of two samples.

Groups with $k \geq 3$ must be sorted. The complexity of sorting a group of size k is proportional to $k \log_2 k$. The probability that there are k items in a group is given by difference of (17), eventually (18) for k and $k + 1$ items. The upper bound for the number of groups of size k can be estimated by this difference and by the total number of items $nC_\infty(3)$. Then, the complexity to sort all items in all groups of size k is proportional to

$$nC_\infty(3)[C_\infty(k) - C_\infty(k + 1)]k \log_2 k \ . \tag{20}$$

For all groups of size $k = 3$ and more there is

$$nC_\infty(3)\sum_{k\geq 3}^{n-1}[C_\infty(k) - C_\infty(k+1)]k\log_2 k\ . \qquad (21)$$

From now on we omit n. The ratio of (20) for $k+1$ and k diminishes fast for $k > 9$. Then we split (21) into the finite term

$$C_\infty(3)\sum_{k=3}^{9}[C_\infty(k) - C_\infty(k+1)]k\log_2 k \qquad (22)$$

and a series

$$\{s(k)\} = \{C_\infty(3)[C_\infty(k) - C_\infty(k+1)]k\log_2 k\} \qquad (23)$$

that can be bounded by geometric series

$$\left\{C_\infty(3)[C_\infty(9) - C_\infty(10)]9\log_2 9q^{k-9}\right\}\ , \qquad (24)$$

where we denote the constant term

$$v_9 = C_\infty(3)[C_\infty(9) - C_\infty(10)]9\log_2 9$$

so that the geometric series has the form

$$\left\{v_9 q^{k-9}\right\}\ . \qquad (25)$$

We have to compare (23) and (25)

$$C_\infty(3)[C_\infty(k) - C_\infty(k+1)]k\log_2 k < v_9 q^{k-9}\ . \qquad (26)$$

It holds $C_\infty(k) - C_\infty(k+1) < C_\infty(k)$ and then

$$C_\infty(3)[C_\infty(k) - C_\infty(k+1)]k\log_2 k < C_\infty(3)C_\infty(k)k\log_2 k$$

and then

$$C_\infty(3)C_\infty(k)k\log_2 k < v_9 q^{k-9}\ .$$

It is

$$C_\infty(3)/(e.v_9).exp(k(1-\ln k)k\log_2 k < q^{k-9} \qquad (27)$$

Using the logarithm there is

$$\ln(C_\infty(3)/(e.v_9.\ln 2)) + k - k\ln k + \ln k + \ln\ln k < (k-9)\ln q$$

and we denote the left-hand side as L_h. For large k the dominating term is $-k\ln k$. On the right-hand side, the dominating term is $k\ln q$. For $q < 1$ both sides diverge to $-\infty$, the L_h as fast as $k\ln k$, the right-hand side linearly with k. From it we decide that in (27) the left-hand side converges to zero as fast as $\exp(-k\ln k)$ and then faster than the right-hand side that converges to zero as fast as $\exp(-k)$.

We can conclude that $\{q^{k-9}\}$ is the series with a finite sum, which bounds from upper the sum of series $\{s(k)\}$, see (23). Therefore, the complexity (21) is proportional to the total number n of items sorted. As all the loops of the algorithm have time complexity $O(n)$ and follow one after another, the algorithm has the time complexity $O(n)$. This finishes the proof.

IV. Numerical demonstrations

A. The maximal multiplicity

Table 1 shows estimated maximal multiplicities k_m, i.e. sizes of largest groups of items in dependence on the total number of items n. To state numbers in the table we use Chernoff bound in the form (19). The last column in Table 1 shows a simple formula roughly approximating k_m till $n = 10^{15}$.

Table 1: The size of the largest bin.

n	k_m	$\lfloor 2+\log n\rfloor$
10	3	3
100	4	4
1000	5	5
10^4	5	6
10^5	6	7
10^6	8	8
10^7	11	9
10^{10}	14	12
10^{15}	16	17
10^{30}	27	32

B. Structure of the computational complexity

Table 2 shows time complexity for individual bin sizes k from 1 to 10, and for sets of items according to the extend in which sorting relates to them. There are three sets, a set of items directly moved to their final positions, a set where items are in pairs and approximately in half of cases their mutual positions must be exchanged, and finally, sets of multiplicity three and more that each must be sorted with the use of comparison sort. We suppose sorting complexity $k\log_2 k$ for bin of size k.

Table 2: The computational complexity per one item sorted.

k	Complexity per one item and k given	Complexity per one item for the set	Set type
1	0.3204	0.3204	single items
2	0.4059	0.4059	pairs
3	0.3094	0.4729	multiplicity 3 and more
4	0.1219		
5	0.03318		
6	0.00695		
7	0.00119		
8	0.00017		
9	2.18E-5		
10	2.44E-6		
Total	1.9923	1.9923	per one item

Note 3 In practice it can be faster to sort all groups of two or more samples with the comparison sort.

C. Space complexity

The space complexity of Algorithm 2 is $2n$, not counting input data x and output y. It is also supposed that the algorithm used for comparison sort of bins with multiplicities has negligible space demand as the bubble sort or the quicksort.

D. Data alocation

Table 3 allows to quantify data allocation with the use of the Chernoff bound. After initial assignment of items to bins in step 2), there are 41.3% empty bins, 32% of items have their own bin, 40.6% items share bin in pairs. For this 20.3% bins are needed. Remaining 27.4% items share each a bin

with at least two other items. The number of items having no bin (last column) equals to the number of empty bins.

Table 3: The use of bins, i.e. bin occupacy.

samples per bin*	Cernoff bound	Percenrage of samples	Percenrage of bins
empty		0	0.41279
1	1.0000	**0.3204**	0.3204
2	0.6796	**0.4059**	0.2030
4	**0.2737**	0.1952	0.0488
5	0.0785	0.0610	0.0122
6	0.0175	0.0143	0.0024
7	3.18E-03	2.69E-03	3.84E-04
8	4.90E-04	4.25E-04	5.31E-05
9	6.54E-05	5.77E-05	6.41E-06
10	7.69E-06	6.88E-06	6.88E-07
11	8.10E-07	8.10E-07	7.37E-08

E. Data moves

The first move for an item happens in step 4) of Algorithm 2. This is exactly n moves.

The other moves appear in sorting the groups of items. There is a formula for the mean number of moves $\bar{\mu}(k)$ in dependence on the number k of items sorted when using the quicksort, see [7] Corollary 5.

$$\bar{\mu}(k) = \frac{2}{3}(k+1)H_k - \frac{4k+1}{18}, \tag{28}$$

where H_k is the kth harmonic number. In Table 4 the frequency of appearance of bins of a particular size are taken into account. Values in the Table are derived with the use of the Chernoff bound, and then hold exactly for the exponential distribution of items sorted. The sum of the last column is the number of moves $0.898n$. This gives total $1.898n$ moves when sorting n items with the use of Algorithm 2.

Table 4: The mean number of data moves in the quicksort for various data set size per one item sorted.

k	Quicksort moves	Items in a bin	Number of bins	Moves per item
1	0.00	0.320	0.320	0
2	2.50	0.406	0.203	0.507
3	4.17	0.195	0.065	0.271
4	6.00	6.10E-02	1.52E-02	9.15E-02
5	7.97	1.43E-02	2.86E-03	2.28E-02
6	10.04	2.69E-03	4.49E-04	4.51E-03
7	12.22	4.25E-04	6.06E-05	7.41E-04
8	14.47	5.77E-05	7.21E-06	1.04E-04
9	16.80	6.88E-06	7.65E-07	1.29E-05
10	19.20	7.33E-07	7.33E-08	1.41E-06

F. Other distributions

We proved the linear sorting time complexity for uniform and exponential distribution of sorting key. We use the finding for uniform and exponential distributions that even there are groups of items to be sorted with a comparison sort, the computational complexity per one item can be bounded from above. It can be supposed that a similar behavior appears also for other light-tailed distributions. With some licence we can rely on that the linear time complexity holds for many light-tailed distributions.

For heavy-tailed distributions we use the fact that data can be transformed with the arctg function. The arctg function applied to data with Cauchy distribution transforms them into having the uniform distribution. For other heavy-tailed distributions the resulting distribution may be rather strange. Despite this fact, characteristics of the light-tailed distributions may be kept and Algorithm 2 can be used with success.

V. Conclusion

The essence of the algorithm studied here is start as a counting sort, ev. as a bucket sort with the number of bins or buckets equal to the number of items sorted. Items in buckets with two or more items sort with a comparison sort.

We found that the number of items in the largest bin grows very slowly with the total number of items sorted n. At the same time, the mean number of such bins lessens rapidly with their size. Overall computational complexity of sorting all bins with two and more items can be bounded by a constant multiplied by the total number of items sorted n. From it follows that the total time complexity of the algorithm is proportional to the total number of samples sorted n. There is also $1.898n$ data moves. At the same time, the space needed is equal to $2n$, ev. $4n$ including input and output arrays. Reduction of the space complexity of the algorithm is subject of futher research.

Acknowledgment

The work was supported (partly) by the long-term strategic development financing of the Institute of Computer Science (RVO:67985807) and in part by the Czech Ministry of Education, Youth and Sports in project No. LM2015068 Cooperation on experiments at the Fermi National Laboratory, USA.

References

[1] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein, *"Introduction to Algorithms"* (2nd ed.), MIT Press and McGraw-Hill, 2001, Chap. 8.2 Counting Sort, pp. 168-170, ISBN 0-262-03293-7.

[2] M. Fernandez, S. Williams, "Closed-Form Expression for the Poisson-Binomial Probability Density Function". IEEE Transactions on Aerospace and Electronic Systems. vol. 46 (2010), No. 2, pp. 803-817. doi:10.1109/TAES.2010.5461658. Eventually available at *https : //en.wikipedia.org/wiki /Poisson_binomial _distribution*

[3] C. A. R. Hoare, "Quicksort" *The Computer Journal* Vol. 5 1962, No. 1, pp. 10-16. Available at *https : //doi.org/*10.1093/*comjnl/*5.1.10

[4] L. Le Cam, "An approximation theorem for the Poisson binomial distribution." *Pacific J. Math.* Vol. 10, 1960, No. 4, pp. 1181-1197.

[5] H. Liu, A. Gegov, F. Stahl, "J-measure Based Hybrid Pruning for Complexity Reduction in Classification Rules", WSEAS Transactions on Systems, pp.433-446 Issue 9, Volume 12, September 2013.

[6] G. Liu, Z. Qiu, W. Chen, "An Iterative Algorithm for Single-pair K Shortest Paths Computation" WSEAS Transactions on Information Science and Applications, pp.305-314, Volume 12, 2015.

[7] C. Martnez, H. Prodinger, "Moves and displacements of particular elements in Quicksort" *Theoretical Computer Science* Volume 410, Issues 2123, 17 May 2009, Pages 2279-2284, available at https://doi.org/10.1016/j.tcs.2009.01.006

[8] H. H. Seward, "Information sorting in the application of electronic digital computers to business operations." *Report No. r-232*, Digital Computer Laboratory, MIT, Cambridge, Mass., USA, 1954.

[9] P. Stanica, "Good lower and upper bounds on binomial coefficients." *Journal of Inequalities in Pure and Applied Mathematics*, Vol. 2 (2001), No. 3, 5 pp. Eventually available at $https : /en.wikipedia.org/wiki/Binomial\_coefficient$.

[10] T. H. Wang, "On the number of successes in independent trials." *Statistica Sinica*, Vo. 3 (1993), pp. 295-312.

Appendix 1: The Algorithm 2

```
void LS(double* arr, int* rank, long samples, double
      keymin, double keymax) {
      //uses IquickSort and double computation
      //of position
      long k,j,i,ix,kx;
      double logdd;
      int* rank1  = new int [samples+1];
      logdd=1.0/(keymax-keymin);
      for(k=0;k<=samples;k++) rank1[k]=0;
      //FIRST loop to build histogram
      for(k=0;k<samples;k++){
            j=(long)fabs((double)(samples-1)*
             (arr[k]-keymin)*logdd);
            rank1[j+1]++;
      }
      //SECOND loop to build sums
      for(k=1;k<=samples;k++) rank1[k]+=
    rank1[k-1];
      //THIRD loop fill-in rank
      for(k=0;k<samples;k++) {
            ix=(long)(fabs((double)(samples-1)*
             (arr[k]-keymin)*logdd);
            kx=rank1[ix];
            rank[kx]=k;
            rank1[ix]++;//bin for the next time
      }
      ix=0;kx=rank1[0];
      //FOURTH loop sort groups by a comparison sort
      for(k=0;k<samples-1;k++){
            if(k>0) kx=rank1[k]+(rank1[k-1]==ix?0:1);
            i=kx-ix;
            if(i>1) IquickSort(arr,rank,ix,kx-1);
            ix=kx;
      }
} // There are indexes in the rank array such that
//arr[rank[i]] is arranged in ascending order.
//In the same way works the IquickSort
//(double* arr, long* rank, long from, long to).
```