Space-efficient Relative Error Order Sketch over Data Streams

Ying Zhang[†] Xuemin Lin[†] Jian Xu[†] Flip Korn[§] Wei Wang[†]

[†]University of New South Wales & NICTA
{yingz,lxue,xujian,weiw}@cse.unsw.edu.au

[§]AT&T Labs-Research
flip@research.att.com

Abstract

We consider the problem of continuously maintaining order sketches over data streams with a relative rank error guarantee ϵ. Novel space-efficient and one-scan randomised techniques are developed. Our first randomised algorithm can guarantee such a relative error precision ϵ with confidence $1-\delta$ using $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\log\epsilon^2 N)$ space, where N is the number of data elements seen so far in a data stream. Then, a new one-scan space compression technique is developed. Combined with the first randomised algorithm, the one-scan space compression technique yields another one-scan randomised algorithm that guarantees the space requirement is $O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon}\log\frac{1}{\delta})\frac{\log^{2+\alpha}\epsilon N}{1-1/2^{\alpha}})$ (for $\alpha > 0$) on average while the worst case space remains $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\log\epsilon^2 N)$. These results are immediately applicable to approximately computing quantiles over data streams with a relative error guarantee ϵ and significantly improve the previous best space bound $O(\frac{1}{\epsilon^3}\log\frac{1}{\delta}\log N)$. Our extensive experiment results demonstrate that both techniques can support an on-line computation against high speed data streams.

1 Introduction

Statistics computation in data streams is often required by many applications, including processing of relational type queries, [3, 8, 9, 10, 16, 17], data mining [18, 24, 28], and high speed network management [4, 6]. Among various statistics, order statistics computation is one of the most challenging, and is employed in many real applications, such as web ranking aggregation and log mining [1, 12], sensor data analysis [14], trends and fleeting opportunities detection in stock markets [3, 23], and load balanced data partitioning for distributed computation [25, 27]. Most order statistics computation problems require memory size linearly proportional to the size of a data stream for exact answers by one-scan techniques [1, 20]; this may be impractical in data stream applications where streams are massive in size and fast in arrival speed. Consequently, approximate computation is a good alternative.

In this paper, we consider the problem of continuously maintaining order sketches over a data stream with a guaranteed relative rank error. This problem arises in many applications where rank-oriented queries are involved [1, 12, 25], including *computing quantiles*. A ϕ-*quantile* ($\phi \in (0,1]$) of an ordered set of N data elements is the element with rank $\lceil\phi N\rceil$. Quantile computation is a key component in online decision support (e.g. portfolio risk measurement in the stock market [23]). It is also useful for the applications including data summarization via *equal-depth histograms* [26], *data mining* [21], and *data partitioning* [27].

It has been shown in [2, 15, 25, 22] that a space-efficient ϵ-approximation quantile summary can be maintained so that, for a quantile ϕ, it is always possible to find an element at rank r' with the precision guarantee $|\lceil\phi N\rceil - r'| \le \epsilon N$. These approximate quantile algorithms are immediately applicable to the problem of maintaining order sketches; however, they only guarantee the rank errors within *absolute* precision ϵN. If N is known in advance, then *relative* error guarantees of $\epsilon\phi N$ can be (over)satisfied by these algorithms using uniform error $\epsilon' = \epsilon\phi_0$, which is the relative error at the ϕ_0-quantile with the lowest rank (that is, $\phi_0 = 1/N$), though this is an undesirable solution since it wastes space by obtaining finer error than needed at most ranks. Moreover, if N is not known (e.g., an "infinite" stream), then no such minimum error bound will suffice. Therefore, the existing quantile techniques as cited above are not applicable to the relative error metric.

Using the relative error metric to measure approximation is not only of theoretical interest but is also very useful in many applications. For instance, as pointed out in [7], finer error guarantees at higher ranks are often desired in network management.[1] This is because IP traffic data often exhibits skew towards the tail and it is exactly in the most skewed region where one wants finer rank error guarantees, to get more precise information about changes in values. Relative error is also motivated by the problem of approximately *counting inversions* on a stream [19].

The problem of finding approximate quantiles with relative error guarantees was first studied by Gupta and Zane [19], who developed a one-scan randomized technique with $O(\frac{1}{\epsilon^3}\log^2 N)$ space requirement for approximately counting inversions, by maintaining an order sketch

The work of the first three authors was partially supported by ARC Discovery Grant (DP0346004).

[1] Note that the form of our relative error metric is biased towards the *head* (i.e., finer error guarantees towards lower ranks). Clearly, finer error guarantees towards the *tail* may be obtained if the data elements are ordered in reverse.

with the relative rank error guarantee ϵ. However, the technique requires advance knowledge of (an upper bound on) N to do one-scan sampling. This potentially limits its applications. Cormode *et al.* [7] studied the related problem of computing *biased quantiles*, that is, the set of quantiles $\Phi = \{\phi_i = \phi_0^i : 1 \leq i \leq k\}$, for a fixed k and some ϕ_0, which are estimated with precision $\epsilon\phi_i N$. [7] gives an algorithm to approximate such biased quantiles with deterministic error guarantees which performs very well against many real data sets. While the problem of computing biased quantiles focuses on the relative rank error guarantee bounded by a minimum quantile $\phi_0^k N$, our problem addresses relative error guarantees at *all* ranks, no matter how small ϕ is. As shown in Section 2, the application of their technique to our problem leads to a linear space requirement $\Omega(N)$ in the worst case; this can render the deterministic technique impracticable in applications where small space usage is imperative.

Given this motivation, in this paper we present novel, space-efficient algorithms to continuously maintain order statistics over data streams without any advance knowledge of N. Our techniques guarantee sub-linear space [2] bounds, and they are based on a one-scan multi-layer randomization. Our contributions may be summarized as follows:

1. We develop a novel, one-scan randomized algorithm ("MR") which guarantees the precision ϵ of relative rank errors with confidence $1-\delta$ and requires $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\log\epsilon^2 N)$ space.
2. We also develop an effective one-scan space compression technique. Combined with the above one-scan randomized technique, it leads to a more space-efficient one-scan randomized algorithm ("MRC") which guarantees the average space requirement $O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon}\log\frac{1}{\delta})\frac{\log^{2+\alpha}\epsilon N}{1-1/2^\alpha})$ (for $\alpha > 0$), while the worst case space requirement remains $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\log\epsilon^2 N)$.
3. These algorithms are the first that can compute approximate quantiles with relative error guarantees at all ranks in sub-linear space, for applications where N is unknown. Our results also significantly improve upon the best existing space bound of $O(\frac{1}{\epsilon^3}\log\frac{1}{\delta}\log N)$ for applications where (an upper-bound of) N is given *a priori*.[3]
4. To complement our theoretical analysis, we present experimental results demonstrating that our techniques can efficiently compute approximate quantiles over high speed data streams using very small space.

The rest of the paper is organized as follows. Section 2 gives the background knowledge information, problem definitions, as well as a brief overview of some closely related work. Section 3 presents our one-scan multi-layer randomization techniques. Section 4 presents a new one-scan space compression technique. In Section 5, we report our experiment results. Section 6 concludes the paper.

[2]Note that the space requirement in this paper refers to the maximum temporary storage during the computation.

[3]This is immediately obtained from the original space bound $O(\frac{1}{\epsilon^3}\log^2 N)$ in [19], where the confidence is $\frac{1}{N^2}$ instead of δ.

2 Background Information

In this section, we first present the problem statement, followed by a brief introduction of some most closely related techniques. Finally, we present preliminaries. Table 1 summarizes the math notations used throughout the paper.

Table 1. Math Notation

Notation	Definition
D	a whole data stream
$D_{i,j}$	a data stream from ith to jth issued elements
N	number of data stream elements seen so far
s (S)	set of sample elements
Γ ($\mathcal{D}$, $\mathcal{G}$)	set of data elements
X (Y)	random variables
ϵ $(1-\delta)$	precision (confidence) requirement
r (τ)	rank
r^-, r^+	lower and upper bounds of r
e $(r(e), v(e))$	a data element (rank and value of e)
p $(p(e))$	probability (of e)
w $(w(e))$	weight (of e)
$\sigma(e)$	summation of element weights upto e

2.1 Problem Statement

In this paper, we study the following rank query over a data stream with a total order on elements' values. For discussion simplification, we assume that a data element has a single value and the total order means an increasing order of data values.

Rank-Element (RE) Query: Given a rank r, find the element with rank r.

It was shown [20] that any algorithm for computing exact ϕ-quantiles of an ordered set of N data elements requires $\Omega(N^{1/\kappa})$ space if κ scans of the data set are allowed. Consider that quantiles computation may be immediately transformed to RE queries. This makes it impractical to compute exact results of RE queries for very large data sets since multiple scans are very costly, even infeasible (e.g. data streams). On the other hand, in many data stream applications an approximate processing of rank queries suffices. In this paper we investigate the problem of approximate processing of RE queries regarding a rank approximation.

Suppose that r is the given rank in a RE query, and r' is the rank of an approximate solution. We could use the constant-based absolute error metric; that is, enforce $|r - r'| \leq \epsilon$ for a given ϵ. Clearly, such an absolute error precision guarantee leads to the space requirement $\Omega(N/\epsilon)$ even for an off-line environment. In this paper, we use the relative error metric: $Err(r', r) = \frac{|r'-r|}{r}$. An answer to RE regarding r is a *relative ϵ-approximate* if $Err(r', r) \leq \epsilon$.

Example 1. *A data stream {15, 8, 10, 9, 1, 8, 10, 9, 6, 7, 8, 13, 5, 4, 2, 3} consists of 16 data elements. The sorted order of the sequence is { 1, 2, 3, 4, 5, 6, 7, 8, 8, 8, 9, 9, 10, 10, 13, 15 }. Let* $\epsilon = 0.2$. *For a rank 5, the RE query returns the element 5; its relative ϵ-approximate answer is either the element* 4, *or* 5, *or* 6.

Quantile Computation and RE Query. Without loss of generality, we may assume that a ϕ-quantile is simply an element with rank ϕN in a data stream with N elements. Clearly, a relative ϵ-approximate answer r' to ϕN of the RE query leads to a ϕ' ($\phi' = r'/N$) such that $\frac{|\phi-\phi'|}{\phi} \leq \epsilon$. Therefore, our results in this paper can be immediately applied to computing quantiles with relative error guarantees.

Problem Description. In the remaining of the paper, we investigate the problem of continuously maintaining a subset S of elements over a data stream D such that at any time, S can be used to return a relative ϵ-approximate answer to the RE rank queries. The aim is to minimize the maximum memory space required in such a continuous computation.

2.2 Related Work

Greenwald and Khanna [15] developed a one-scan technique with $O(\frac{1}{\epsilon}\log(\epsilon N))$ space bound and deterministic error guarantee $|r - r'| \leq \epsilon N$, while Manku *et al* [25] provided a space efficient randomized algorithm, with space bound $O(\frac{1}{\epsilon}\log^2(\frac{1}{\epsilon}\log\frac{1}{\epsilon\delta}))$, to achieve such an error guarantee with confidence $1-\delta$. Gilbert *et al* [13] developed a one-scan randomized technique to handle the applications where elements may be deleted. Arasu *et al* [2] and Lin *et al* [22] recently developed space-efficient ϵ-approximation techniques for sliding windows. Distributed algorithms over sensor networks have also been recently developed by Greenwald *et al* [14] and Cormode *et al.* [5].

Below, we discuss some closely related techniques. Some of them will be applied to our one-scan space compression techniques. Specifically, we will apply the approximate algorithm in [15], named GK-algorithm thereafter, as well as the merge technique in [2, 14, 22], named Algorithm-MERGE thereafter. We will apply these as black-boxes in our algorithm.

GK-algorithm. Using λ as a parameter, it maintains a set (sketch) of tuples over a data stream with N elements, $\xi = \{(e_i, r_i^-, r_i^+) : 1 \leq i \leq m\}$, with the following properties with respect to λ and N:

GK1: each $e_i \in D$;

GK2: for $1 \leq i \leq m-1$, $v(e_i) \leq v(e_{i+1})$, $r_i^- < r_{i+1}^-$; [4]

GK3: for $1 \leq i \leq m$, $r_i^- \leq r(e_i) \leq r_i^+$,

GK4: $r_1^+ \leq \lambda N + 1$,

GK5: $r_m^- \geq (1-\lambda)N$,

GK6: for $2 \leq i \leq m$, $r_i^+ \leq r_{i-1}^- + 2\lambda N$.

For each tuple (e_i, r_i^-, r_i^+) in the above quantile sketch, e_i is one of the data elements in the data stream seen so far. Here, e_1 and e_n have the smallest value and the largest value, respectively. To be efficient, the algorithm uses two parameters g_i and Δ_i to control r_i^- and r_i^+, where $g_i = r_i^- - r_{i-1}^-$, $\Delta_i = r_i^+ - r_i^-$, and $r_i^- = \sum_{j \leq i} g_j$. The following theorem has been proven in [15].

Theorem 1. *A set ξ of tuples with the properties GK1-GK6 with respect to λ and N has the precision guarantee λN for absolute rank errors; that is, for each r ($1 \leq r \leq N$), there is a (e_i, r_i^-, r_i^+) in ξ such that $r - \lambda N \leq r_i^- \leq r_i^+ \leq r + \lambda N$. (This implies $r - \lambda N \leq r(e_i) \leq r + \lambda N$.)*

[4] In the actual implementation of GK-algorithm, $r_i^+ \leq r_{i+1}^+$ for each i. However, this does not affect the precision of GK-algorithm.

Algorithm-MERGE. Suppose that a data stream D is divided into q disjoint sets of elements, $\{\mathcal{D}_i : 1 \leq i \leq q\}$. Let GK-algorithm run on each $\mathcal{D}_i$ to generate a sketch ξ_i with precision guarantee $\lambda_i N_i$ for absolute rank errors where $N_i = |\mathcal{D}_i|$. Then, Algorithm-MERGE ($\cup_{i=1}^q \xi_i$) in [2, 14, 22] can generate a global sketch ξ on D that satisfies GK1-GK6 with respect to the following λ and $\sum_{i=1}^q N_i$, where

$$\lambda = \frac{\sum_{i=1}^q \lambda_i N_i}{\sum_{i=1}^q N_i}. \tag{1}$$

According to Theorem 1, ξ has the absolute rank error precision guarantee $\sum_{i=1}^q \lambda_i N_i$.

CKMS-algorithm. In the very recent work [7], the problem of estimating biased quantiles has been studied. As defined in the last section, a set of biased quantiles $\Phi = \{\phi_i = \phi_0^i : 1 \leq i \leq k\}$ for a pre-fixed k and some ϕ_0 are to be estimated with precision $\epsilon\phi_i N$. The CKMS-algorithm is a variation of GK-algorithm; it modifies GK6 to

$$r_i^+ - r_{i-1}^- \leq f(r_i, N), \tag{2}$$

where $f(r_i, N)$ is defined as $2\epsilon\max(r_{i-1}^-, \phi_k N, 1/2\epsilon)$. It was shown [7] that this algorithm can guarantee the relative ϵ-approximation for a RE query with the ranks in $[\phi_k N, N]$ with a poly-logarithmic space requirement. However, to apply CKMS-algorithm to RE queries across all ranks in $[1, N]$, $\phi_k N$ has to be enforced to 1; that is,

$$f(r_i, N) = 2\epsilon\max\{r_{i-1}^-, \frac{1}{2\epsilon}\}. \tag{3}$$

This will immediately lead to a linear space requirement $\Omega(N)$, by CKMS-algorithm, in the worst case. Below is such an example.

Example 2. *Suppose that a data stream is divided into a number of batches B_i such that each batch consists of n consecutively arrived elements. In each batch B_i, a later arrived element has larger value. Moreover, the values of data elements of each batch B_i are always assigned between the value of the second last tuple and the value of the last tuple in the sketch created by CKMS-algorithm for the first $i-1$ batches.*

As shown in [7], there must be at least $\frac{1}{\epsilon}\log\epsilon n$ tuples in an order sketch to ensure the relative ϵ-approximation of ranks. We choose n such that $\frac{1}{\epsilon}\log\epsilon n \geq 3$. Suppose that after applying CKMS algorithm to B_1, there are m_1 tuples generated in the sketch ξ_1. Clearly, $m_1 \geq 3$.

According to the insertion and merge rules in CKMS-algorithm, it can be immediately verified that:

- The new tuples are between the second last and the last tuples in ξ_1.
- Any new tuples cannot merge with any tuples before the second last tuple (inclusive) in ξ_1 due to (2) and (3) to maintain the relative ϵ-approximation.

- Due to the sortedness of B_1 and B_2, the last tuple in ξ_1 already reaches its maximum merging capacity. Consequently, the last tuple in the new sketch can merge with another $\frac{2\epsilon}{1+2\epsilon}n$ elements in B_2 to reach the new maximum capacity.

These immediately imply that after adding B_2, the number of sketch tuples is increased by at least 1. These properties also hold for every batch insertion; consequently the number of tuples generated by CKMS-algorithm is at least $\frac{N}{n} = \Omega(N)$. Our experiments also demonstrate this in section 6.

Randomized Algorithms. In [19], Gupta and Zane proposed a one-scan randomized algorithm with $O(\frac{1}{\epsilon^3}\log^2 N)$ space requirement. The main idea is to appropriately select $\frac{1}{\epsilon}\log N$ ranks to do approximation such that for each selected rank, a uniform sampling method with a same sampling rate is developed. The lower the rank is, the higher the sampling rate is. Since this algorithm always has to sample data elements from the first element of a data stream for each selected rank and the sampling rates for different ranks are different, it is not applicable to the applications when the upper bond of N is not known a priori and only one scan of a dataset is allowed.

Manku *et al* in [25] proposed a very effective space compression technique based on an adaptive sampling technique. However, the effectiveness of this space compression technique mainly relies on an exponential reduction of sampling rates along with the increment of N. Therefore, that space compression technique can only guarantee the precision ϵN of absolute rank errors but cannot guarantee the relative ϵ-approximation of rank errors. Consequently, it is infeasible to combine the techniques in [25] and [19] to reduce the space requirement in [19].

2.3 Preliminaries

In this paper, we propose a novel randomized algorithm based on a multi-layer sampling technique. By avoiding sampling data elements from the first data stream element at each level/layer, our one-scan technique can work for the applications where N is not known a priori.

In our algorithm, we maintain a global sketch (i.e. set of tuples) with the following form $\xi = \{(e_j, \tau_j^-, \tau_j^+) : 1 \leq j \leq m\}$ over a data stream D, such that:

SK1: for $1 \leq j \leq m-1$, $v(e_j) \leq v(e_{j+1})$, $\tau_j^- < \tau_{j+1}^-$,

SK2: for $1 \leq j \leq m$, $e_j \in D$ and $\tau_j^- \leq \tau_j^+$.

Note that in our algorithms it is impossible to impose the same property, $\tau_j^- \leq r(e) \leq \tau_j^+$, as that in GK-algorithm; this is because of the nature of a randomization. To resolve this, we enforce the following properties.

Theorem 2. *Suppose that ξ is a set of tuples, over a data stream D with N elements, which satisfies the constraints in SK1 and SK2. Suppose that ξ also has the following properties regarding an interval $R = [r_{min}, r_{max}]$ of ranks:*

SK3: *$\forall r \in R$, $\exists (e_j, \tau_j^-, \tau_j^+) \in \xi$ such that $[\tau_j^-, \tau_j^+] \subseteq [r(1-\epsilon/2), r(1+\epsilon/2)]$;*

SK4: *for a $r \in R$, if $[\tau_j^-, \tau_j^+] \subseteq [r(1-\epsilon/2), r(1+\epsilon/2)]$, then $r(e_j) \in [r(1-\epsilon), r(1+\epsilon)]$.*

Then, ξ can always provide a relative ϵ-approximate answer to a RE query r with $r \in R$.

Proof. SK3 and SK4 immediate imply that such an ξ can always provide a relative ϵ-approximate answer to a RE query. □

3 Multi-Layer Randomization

In our algorithm, we divide the whole rank range into disjoint intervals with exponentially increased lengths, so that a local sketch is maintained for each interval of ranks with properties SK1-SK4 where SK4 is ensured with confidence $1-\delta$. We can show that the space required in our algorithm is $O(\frac{1}{\epsilon^2}(\log\frac{1}{\delta})\log \epsilon^2 N)$. The section is organized as follows. We first present the randomization technique; then show the quality of our randomization. This is followed by our query algorithms and space complexity analysis.

3.1 The Randomization

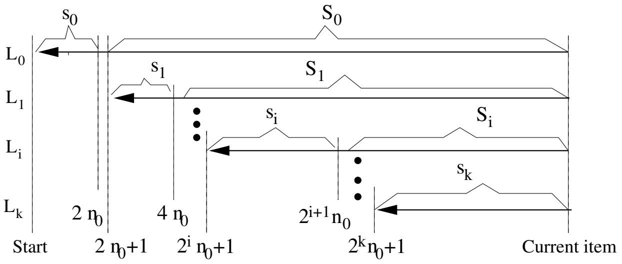


Figure 1. Illustration of our algorithm

The basic idea of the algorithm is as depicted in Figure 1. For a given n_0 we divide the data stream D into $\lceil \log\frac{N}{n_0}\rceil$ non-disjoint substreams: $\{L_i : 0 \leq i \leq \lceil \log\frac{N}{n_0}\rceil - 1\}$, where

- L_0 represents the data stream starting from the 1st issued element.
- for $1 \leq i \leq \lceil \log\frac{N}{n_0}\rceil - 1$, L_i represents the data stream starting from the $(2^i n_0 + 1)$th issued element.

Let $R_i = [2^i n_0, 2^{i+1}n_0]$ if $i \geq 1$ and $R_0 = [1, 2n_0]$. For each substream L_i (except the last substream) of D, two sets, s_i and S_i, of sample points are sampled by a uniform sampling method, while the last substream L_k we sample only s_k. Clearly, there will be $2\lceil \log\frac{N}{n_0}\rceil - 1$ sub-sketches maintained: two (s_i and S_i) per "level" L_i except the last level (only one s_k). Then, a sketch obtained from $\mathcal{F}_i = (\cup_{j=0}^{i} s_j) \cup S_i$ is for approximating the ranks in R_i. Note that a sample set s_i may serve globally for the ranges R_j ($j \geq i$), while S_i is used locally for R_i.

At each level L_i, the uniform sampling method randomly choose an element from each batch of consecutively arrived 2^i elements; we divide L_i into such disjoint batches. The set of sampled elements over the first $2^i n_0$ arrived points in L_i are kept in s_i, while the remaining are kept in S_i as depicted in Figure 1. In the next subsection, we will show that an appropriate selection of n_0 leads to a very good sampling quality. The algorithm is described in Algorithm 1.

Algorithm 1 : Multi-layer Randomization

Input: Data stream D and n_0 (derived from ϵ and δ).
Output: sample sets s_i and S_i for $0 \leq i \leq \lceil \log \frac{N}{n_0} \rceil - 1$
Description:

$k := 0$; $N := 0$; $s_0 := \emptyset$; $S_0 := \emptyset$;
while e arrives **do**
  $N := N + 1$;
  if $\lceil \log \frac{N}{n_0} \rceil - 1 > k$ **then**
    $k := k + 1$; $s_k := \emptyset$; $S_k := \emptyset$;
  end if
  for $j = 0$ to k **do**
    if $ram_j(e) = 1$ and $j \neq k$ **then**
      $S_i := S_i + \{e\}$;
    else if $ram_k(e) = 1$ **then**
      $s_k := s_k + \{e\}$;
    end if
  end for
end while

In the algorithm, ram_i () is used to randomly select one element from a batch of 2^i elements; $ram_i(e) = 1$ means that the element e should be selected.

Example 3. *For the data stream in Example 1 where $N = 16$, let $n_0 = 2$. There are 3 levels, L_0 (the whole stream), L_1 consisting of the most recent 12 elements, and L_2 consisting of the most recent 8 elements.*

At level L_0, the algorithm takes every element and put the first 4 arrived elements in s_0 and the remaining in S_0.

At level L_1, the algorithm randomly selects one element from each of $\{1, 8\}$, $\{10, 9\}$, $\{6, 7\}$, $\{8, 13\}$, $\{5, 4\}$, and $\{2, 3\}$, respectively, where $ram_1()$ is used for this purpose. The two sample elements obtained from $\{1, 8, 10, 9\}$ are put in s_1, and the remaining are put in S_2.

At level L_2, the algorithm randomly selects one element (controlled by $ram_2()$ from each of $\{6, 7, 8, 13\}$ and $\{5, 4, 2, 3\}$, respectively. The two sample points are put in s_2, while $S_2 = \emptyset$.

It is immediate that if we keep all sample elements then the space required is $O(N \log N)$. Note that $|s_i| = n_0$ for $i \geq 1$ and $|s_0| = 2n_0$. We will show later that at each level, we need only to keep $O(n_0)$ elements in S_i since we need only to approximate the ranks in R_i. Before proving this, we will first show the quality of our randomization.

3.2 Quality of our Randomization

In this subsection, we will show that for each i ($1 \leq i \leq \lceil \log \frac{N}{n_0} \rceil - 1$), the sketch constructed from sorting the elements in $\mathcal{F}_i = (\cup_{j=0}^{i} s_j) \cup S_i$, as follows, ensures SK1-SK3, while a violation of SK4 has a small probability δ if n_0 is chosen appropriately.

As a sample element e in $s_i \cup S_i$ is a representative of a batch of 2^i data elements, $w_e = 2^i$ represents its weight. Suppose that for each i, the elements in $\mathcal{F}_i$ are sorted according to the element values increasingly; that is, for $e_q, e_l \in \mathcal{F}_i$, $v(e_q) \geq v(e_l)$ if $q > l$. For every element $e_q \in \mathcal{F}_i$, we use $\sigma_{\mathcal{F}_i}(e_q)$ to denote the accumulative weights $\sum_{j=1}^{q} w_{e_j}$. Note that the weights of elements in $\mathcal{F}_i$ are from 1 to 2^i as the weight for an $e \in s_j$ ($j < i$) is 2^j.

For each element $e_j \in \mathcal{F}_i$, we make $\tau_j^- = \tau_j^+ = \sigma_{\mathcal{F}_i}(e_q)$. Therefore, for each i, the sketch created is $\xi_i = \{(e_j, \tau_j^-, \tau_j^+) : 1 \leq j \leq |\mathcal{F}_i|\}$.

Since ξ_0 takes all the data stream elements, it is immediate that ξ_0 satisfies SK1-SK4. Below we show the quality of ξ_i for $i \geq 1$ regarding SK4 and SK3 as it is trivial that SK1 and SK2 are enforced.

For a given rank $r \in R_i$ and an ϵ, let

$$\begin{aligned} \Gamma_{i,r,\epsilon} &= \{e : \ e \in \xi_i \ \& \ \sigma_{\mathcal{F}_i}(e) \in [r(1-\epsilon), r(1+\epsilon)]\}, \\ Q_{r,\epsilon} &= \{e : \ e \in D \ \& \ r(e) \in [r(1-\epsilon), r(1+\epsilon)]\}. \end{aligned}$$

Theorem 3. *If $n_0 \geq \frac{16}{\epsilon^2} \log(2/\delta)$, then for any $r \in R_i$ ($1 \leq i \leq \lceil \log \frac{N}{n_0} \rceil - 1$), $p(\Gamma_{i,r,\frac{\epsilon}{2}} \not\subseteq Q_{r,\epsilon}) \leq \delta$.*

Theorem 3 implies that an appropriate selection of n_0 will make the violation of SK4 have a small probability δ. The proof is based on a stronger version of Chernoff-Hoeffding bounds. It is well-known that Chernoff-Hoeffding bounds also hold for independent bounded random variables where the variables can take any values in an interval $[0, 1]$. Below are the Chernoff-Hoeffding bounds for independent bounded random variables [5].

Lemma 1. ***Chernoff-Hoeffding Bounds [11]:*** *Let $X_1, X_2, X_3, ..., X_n$ be independent random variables with values in $[0, b]$, $X = \sum_{i=1} X_i$, and $\epsilon \in (0, 1]$. Then,*

$$p(X > (1+\epsilon)E(X)) < exp(-E(X)\epsilon^2/3b), \ \ and \tag{4}$$

$$p(X < (1-\epsilon)E(X)) < exp(-E(X)\epsilon^2/2b) \tag{5}$$

To prove Theorem 3, the following two sets of random variables are constructed.

Left-Bound Trials for ξ_i: For each $(e_j, \tau_j^-, \tau_j^+) \in \xi_i$, let

$$X_{r,\epsilon,j} = \begin{cases} w_{e_j}, & \text{if } r(e_j) \leq r(1-\epsilon) \\ 0, & \text{otherwise} \end{cases} \tag{6}$$

Right-Bound Trials for ξ_i: For each $(e_j, \tau_j^-, \tau_j^+) \in \xi_i$, let

$$Y_{r,\epsilon,j} = \begin{cases} w_{e_j}, & \text{if } r(e_j) \leq r(1+\epsilon) \\ 0, & \text{otherwise} \end{cases} \tag{7}$$

Let $X_{r,\epsilon} = \sum_{j=1}^{|\xi_i|} X_{r,\epsilon,j}$ and $Y_{r,\epsilon} = \sum_{j=1}^{|\xi_i|} Y_{r,\epsilon,j}$.

It can be immediately verified that $p(X_{r,\epsilon,j} = w(e_j)) = \frac{q_j}{w_{e_j}}$ where q_j denotes the number of elements whose ranks are not great than $r(1-\epsilon)$ and which fall into the stream segment where e_j is selected. Consequently, $E(X_{r,\epsilon}) = r(1-\epsilon)$. Similarly, we can show that $E(Y_{r,\epsilon}) = r(1+\epsilon)$. Below we prove Theorem 3.

[5] In the original form of Chernoff-Hoeffding bound, those bounded variable values are in $[0, 1]$. It can be immediately extended to $[0, b]$ for a $b > 1$.

Proof. Without loss of generality, we prove the theorem for $\epsilon \leq 1/4$. If $\epsilon > 1/4$, then we can use this theorem to achieve the precision $1/4$. For a similar reason, we assume $\delta \leq 1/2$. Note that for those independent bounded random variables constructed above, $b = 2^i$.

Since $r \in [2^i n_0, 2^{i+1} n_0]$, we have

$$\frac{r}{b} > n_0. \tag{8}$$

To prove the theorem, we need only to prove

$$p(\exists e \in \Gamma_{i,r,\frac{\epsilon}{2}}, e \notin Q_{r,\epsilon}) \leq \delta. \tag{9}$$

To prove (9), we prove the following instead:

$$p(\exists e \in \Gamma_{i,r,\frac{\epsilon}{2}},\ r(e) < r(1-\epsilon)) \leq \frac{\delta}{2}, \tag{10}$$

$$p(\exists e \in \Gamma_{i,r,\frac{\epsilon}{2}}, r(e) > r(1+\epsilon)) \leq \frac{\delta}{2} \tag{11}$$

Clearly, an event "$\exists e \in \Gamma_{i,r,\frac{\epsilon}{2}},\ r(e) < r(1-\epsilon)$" must imply an event "$X_{r,\epsilon} \geq r(1-\frac{\epsilon}{2})$". Therefore,

$$p(\exists e \in \Gamma_{i,r,\frac{\epsilon}{2}},\ r(e) < r(1-\epsilon)) \leq p(X_{r,\epsilon} \geq r(1-\frac{\epsilon}{2}))$$

Similarly, an event "$\exists e \in \Gamma_{i,r,\frac{\epsilon}{2}},\ r(e) > r(1+\epsilon)$" must imply an event "$Y_{r,\epsilon} \leq (1+\frac{\epsilon}{2})r$. Thus,

$$p(\exists e \in \Gamma_{i,r,\frac{\epsilon}{2}},\ r(e) > r(1+\epsilon)) \leq p(Y_{r,\epsilon} \leq (1+\frac{\epsilon}{2})r) \tag{12}$$

Below we first prove the inequality (10). Immediately, $(1-\frac{\epsilon}{2})r \geq (1+\frac{\epsilon}{2})(1-\epsilon)r$. Consequently,

$$p(X_{r,\epsilon} \geq r(1-\frac{\epsilon}{2})) \leq p(X_{r,\epsilon} > (1+\frac{\epsilon}{2})(1-\epsilon)r) \tag{13}$$

From the upper-tail of Chernoff-Hoeffding Bound and $\epsilon \leq 0.25$ and (8), we have the following:

$$p(X_{r,\epsilon} > (1+\frac{\epsilon}{2})(1-\epsilon)r) = p(X_{r,\epsilon} > (1+\frac{\epsilon}{2})E(X_{r,\epsilon}))$$

$$< exp(-(1-\epsilon)\epsilon^2 r/3 \times 4 \times b) \leq exp(-\frac{\epsilon^2 n_0}{16})$$

Thus, when $n_0 \geq \frac{16}{\epsilon^2}\log 2/\delta$, $p(X_{r,\epsilon} > (1+\frac{\epsilon}{2})(1-\epsilon)r) < \frac{\delta}{2}$; consequently (10) holds.

It can be verified that $(1+\frac{\epsilon}{2})r \leq (1-0.4\epsilon)(1+\epsilon)r$ when $\epsilon \leq \frac{1}{4}$. Consequently,

$$p(Y_{r,\epsilon} \leq (1+\frac{\epsilon}{2})r) \leq p(Y_{r,\epsilon} \leq (1-0.4\epsilon)E(Y_{r,\epsilon})). \tag{14}$$

By the lower-tail of Chernoff-Hoeffding bound, the choice of n_0 as above, (12), and (14), (11) is immediate. Thus Theorem 3 holds. □

Due to Theorem 3, we set $n_0 = \frac{16}{\epsilon^2}\log(2/\delta)$ in Algorithm 1. It can be immediately verified that SK3 will be always met regarding each ξ_i and R_i. Below is the query algorithm (Algorithm 2) to return such a tuple from ξ_i for a $r \in R_i$ such that SK3 is satisfied.

Based on the above results, Theorem 2 immediately implies that $\xi = \cup_{\forall i}\xi_i$ is able to return an ϵ-approximate answer to a RE query (for a given r) with confidence $1-\delta$.

Algorithm 2 : RE query

Input: $1 \leq r \leq N$ and $\{\xi_i : 0 \leq i \leq \lceil \log \frac{N}{n_0} \rceil - 1\}$.
Output: a data element e such that $r(e) \in [r(1-\epsilon), r(1+\epsilon)]$ with $1-\delta$ confidence.
Step 1: Find i such that $2^i n_0 \leq r \leq 2^{i+1} n_0$.
Step 2: Return e_j in ξ_i if $\sigma_{\mathcal{F}_i}(e_j)$ is the first not smaller than r.

3.3 Space and Time Complexities

Let $n_0 = \frac{16}{\epsilon^2}\log(2/\delta)$, in this section we discuss the space complexities of Algorithm 1, and the time complexities for Algorithms 1 and 2, respectively.

Space Complexity. For each tuple $(e_j, \tau_j^-, \tau_j^+) \in \xi_i$, $\tau_j^- = \tau_j^+ = \sigma_{\mathcal{F}_i}(e_q)$. As the value of $\sigma_{\mathcal{F}_i}(e_q)$ will change when new elements are sampled, we do not materialize ξ_i for efficiency reasons. Instead, we keep every s_i, and maintain only the *smallest* n_0 *tuples* (with respect to the values) in S_i. This is because S_i is only used to query the ranks in $R_i = [2^i n_0, 2^{i+1} n_0]$ by Algorithms 2. Note $|s_0| = 2n_0$ and $|s_i| = n_0$ when $i \geq 1$. There are totally $O(\log \frac{N}{n_0})$ layers. Therefore, the space requirement of Algorithm 1 is $O(\frac{1}{\epsilon^2}\log(\frac{1}{\delta})\log(\frac{\epsilon^2 N}{\log 1/\delta}))$ $(= O(n_0 \log \frac{N}{n_0}))$.

Time Complexity. In our implementation, we maintain binary search trees (such as AVL trees) on the data element values in each s_i and S_i, respectively. Therefore, Algorithm 1 runs in $O(\log(\frac{\epsilon^2 N}{log 1/\delta})\log(\epsilon^{-2}\log\frac{1}{\delta}))$ time for processing each element. We claim, without proof, due to space limits, that Algorithm 2 can run in average time $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\log\log(\frac{\epsilon^2 N}{log 1/\delta}))$ by these data structures when ξ_i is not materialized.

4 Space Reduction Algorithm

In this section, we present an effective one-scan space reduction algorithm on each s_i and S_i generated by Algorithm 1. Although the worst case space requirement remains the same - $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta}\log\epsilon^2 N)$, this one-scan space reduction technique combining with Algorithm 1 leads to $O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon}\log\frac{1}{\delta})\frac{\log^{2+\alpha}\epsilon N}{1-1/2^\alpha})$ (for $\alpha > 0$) space on average. The compress technique is based on GK-algorithm.

Overview of the Algorithm. In order to meet the precision guarantee requirement as shown in the next several sections, for each s_i we use GK-algorithm with the parameter $\lambda = \frac{\epsilon}{8}$.

For each S_i, we want to have a sketch with the properties GK1-GK6 in order to have a fixed absolute "rank" error guarantee 2×2^i (i.e., $\lambda|S_i| = 2 \times 2^i$). As the size of S_i is not known because of no advance knowledge of N, it is impossible to figure out the parameter (λ), in advance, in GK-algorithm. Moreover, modifying GK-algorithm with the absolute rank error guarantee 2×2^i does not yield a similar space bound to that in the original GK-algorithm. To resolve this, in our compression technique we divide S_i into disjoint substreams with sizes exponentially increased. Then, for each sub-stream we apply GK-algorithm with an appropriately chosen λ.

Finally, we merge the GK-sketches on s_j (for $1 \leq j \leq$

i) with S_i using Algorithm-MERGE. If we use a similar space truncation condition as that in section 3.3, it can be guaranteed that the worst case space requirement remains the same as that of Algorithm 1.

Below are the details of our algorithms.

4.1 Key Observations

Suppose that we can represent $\mathcal{F}_i$, generated by Algorithm 1, by another sketch $\{(e_j, \tau_j^-, \tau_j^+) : 1 \leq j \leq m'\}$, such that the new sketch satisfies GK1-GK6 if we replace $r(e_j)$ by $\sigma_{\mathcal{F}_i}(e_j)$ in GK3, and replace r_j^- and r_j^+ by τ_j^- and τ_j^+, respectively. Then by Theorem 1, this sketch can satisfy SK1-SK3 for R_i if the precision guarantees in GK1-GK6 are chosen appropriately. Immediately, GK3 and Theorem 3 imply such a sketch violates SK4 with a small probability δ.

Suppose that the rank of an element $e \in s_i$ is r according to the values' order in s_i. It is immediate that $\sigma_{s_i}(e) = 2^i r$. Now, GK-algorithm (s_i, λ_i) (λ_i is the precision required by GK-algorithm) returns a "GK-sketch" $\varrho' = \{(e_j, r_j^-, r_j^+) : 1 \leq j \leq m\}$ satisfying GK1-GK6. Here, GK-algorithm runs on s_i by discounting element weights. Let:

$$w_i \otimes \varrho' := \{(e_j, \tau_j^- = w_i r_j^-, \tau_j^+ = w_i r_j^+) : 1 \leq j \leq m\} \quad (15)$$

It is immediate that $w_i \otimes \varrho'$ satisfies GK1-GK6 regarding $\lambda = \lambda_j$ and $N = w_i|s_i|$ where in GK3, we replace $r(e)$ by $\sigma_{s_i}(e)$. Similar results hold for S_i.

Suppose that there are q sample sets, $\{T_j : 1 \leq j \leq q\}$ generated by Algorithm 1, where a T_j is either in the form of a s_l or S_l. Consequently, elements in a T_j have the same weight, denoted by w'_j. Let ϱ_j = GK-algorithm (T_j, λ_j). We have the following theorem.

Theorem 4. *Let ϱ = Algorithm-MERGE $(\cup_{j=1}^q w'_j \otimes \varrho_j)$. Then, ϱ satisfy GK1-GK6 regarding $\lambda = \frac{\sum_{j=1}^q \lambda_j w'_j |T_j|}{\sum_{j=1}^q w'_j |T_j|}$ and $N = \sum_{j=1}^q w'_j |T_j|$ if $r(e)$ is replaced by $\sigma_{\cup_{j=1}^q T_j}(e)$ in GK3.*

Proof. This theorem is immediate based on the proof of (1) in [2, 14, 22]. □

By Theorem 4 and Theorem 1, it is immediate that for each $\kappa \in [1, \sum_{j=1}^q w'_j |T_j|]$, an element $(e, \tau^-, \tau^+) \in \varrho$ can always be obtained such that $[\tau^-, \tau^+] \in [r - \sum_{j=1}^q \lambda_j w'_j |T_j|, r + \sum_{j=1}^q \lambda_j w'_j |T_j|]$. Together with the fact that $\sigma_{\cup_{j=1}^q T_j}(e) \in [\tau^-, \tau^+]$ (GK3), this implies that we can apply Theorem 3 if $\sum_{j=1}^q \lambda_j w'_j |T_j|$ can be controlled appropriately. This is what we will do in our space reduction algorithm.

4.2 The Algorithm

In this subsection, we will compress each $\mathcal{F}_i$, generated by Algorithm 1, such that the absolute errors over $R_i = [r_{i,min}, r_{i,max}]$ are within $\frac{\epsilon}{2} r_{i,min}$ in order to meet SK1-SK4 by using GK1-GK6.

Consider that in Algorithm 1, the number of sample elements for $R_0 = [1, 2n_0]$ is large and cannot be effectively reduced by the absolute error guarantee $\frac{\epsilon}{2} \times 1$. In our space reduction algorithm, we further divide R_0 into several rank intervals. Specifically, we make $R'_0 = [1, 2\frac{8}{\epsilon}]$, and $R'_i = [2^i \frac{8}{\epsilon}, 2^{i+1}\frac{8}{\epsilon}]$ for $i \geq 1$. Note that N may not in the form of a $2^k \frac{8}{\epsilon}$ but this will not affect our results. To simplify our presentation, we also assume that $2^{i_0}\frac{8}{\epsilon} = 2n_0$ $(= O(\frac{1}{\epsilon^2} \log \frac{2}{\delta}))$; we can always increase n_0 a bit to make the equality hold.

To generate compressed sketches, which serve for the rank queries corresponding to the rank intervals $\{R'_i : 0 \leq i \leq i_0 - 1\}$, we feed our compression algorithm with s_0 and S_0. As s_0 and S_0 together contain every data element in a stream, different parts of the whole stream are fed to our algorithm with different compression rates. Algorithm 3 gives a brief description of our space reduction algorithm, where $D_{i,j}$ denotes the set of elements issued from ith time to jth time.

Algorithm 3 : Compression

Input: ϵ, δ, $n_0 = O(\frac{1}{\epsilon^2} \log \frac{2}{\delta})$, $\{s_i, S_i : i \geq 0\}$ continuously generated by Algorithm 1 on a stream D.

Output: Sketches $\{\varrho'_i, \varrho_i : i \geq 0\}$.

Phase 1: ϱ'_0: the smallest $2\frac{8}{\epsilon}$ elements dynamically maintained from D.

Phase 2: for $i = 1$ to $i_0 - 1$,

- ϱ'_i := GK-algorithm $(D_{2^i \frac{8}{\epsilon}+1, 2^{i+1}\frac{8}{\epsilon}}, \lambda_i = \frac{\epsilon}{8})$; and
- ϱ_i := Approximate $(D_{2^{i+1}\frac{8}{\epsilon}+1, \infty}, \epsilon)$; (described in Algorithm 4)

Phase 3: for $i = i_0$ to k,

- $\varrho'_i := 2^{i-i_0+1} \otimes$GK-algorithm $(s_{i-i_0+1}, \lambda_i = \frac{\epsilon}{8})$; and
- ϱ_i := Approximate (S_{i-i_0+1}, ϵ); (described in Algorithm 4)

Below, we describe the algorithm - Approximate ( ).

4.3 Approximate ( )

Let $\overline{S_i} = D_{2^{i+1}\frac{8}{\epsilon}+1, \infty}$ for $1 \leq i \leq i_0 - 1$ and $\overline{S_i} = S_{i-i_0+1}$ for $i_0 \leq i \leq k$. In this subsection, we combine GK-algorithm and Algorithm-Merge in a novel way to approximately represent an $\overline{S_i}$ $(i \geq 1)$.

Consider that $\overline{S_i}$ will be queried against the rank interval $R'_i = [r'_{i,min}, r'_{i,max}]$ where $r'_{i,min} = 2^i \frac{8}{\epsilon}$ and $r'_{i,max} = 2^{i+1}\frac{8}{\epsilon}$. To make an appropriate use of Theorem 3 we retain the absolute rank error to be at most $\frac{\epsilon}{4} \times (2^i \frac{8}{\epsilon})$(considering a merge step later). This can make our sketch to be built meet SK3-SK4. Since such a sketch responses only to the rank queries in R'_i, we do not need to consider an element e in S_i such that $\sigma_{S_i}(e) > r'_{i,max}(1 + \frac{\epsilon}{2}) = r'_{i,max} + \epsilon r'_{i,min}$ (noting SK3), where $r'_{i,max} = 2^{i+1}\frac{8}{\epsilon} = 2r'_{i,min}$.

Note that in the sketch maintained by Approximate (), every tuple is in form (e, τ^-, τ^+) with $\tau^- \leq \sigma_{S_i}(e) \leq \tau^+$. Thus, we **filter out** a new element e_{new} if $v(e_{new}) \geq v(e_c)$ where e_c is the lowest ranked element with its τ^- value has the following property:

$$\tau^- \geq r'_{i,max} + \epsilon r'_{i,min} (= 2^{i+1}\frac{8}{\epsilon} + \epsilon(2^i \frac{8}{\epsilon})). \quad (16)$$

This is the *filter* to be used in our algorithm Approximate (). e_c is called *cut-off* element.

Let $\epsilon_1 = \frac{\epsilon(1-\frac{1}{2^\alpha})}{4}$ for an $\alpha > 0$. The basic ideas of the algorithm Approximate () are as follows:

- run GK-algorithm on the first issued $\frac{8}{\epsilon} \times \frac{2^i}{w_i}$ elements in $\overline{S_i}$ regarding ϵ_1.
- For a data element e' issued thereafter, feed GK-algorithm with e' only if it passes the above filter regarding the current S_i; that is, $v(e') < v(e_c)$. Here e_c is the current cut-off element.
- For the elements passing the filter, we divide them exponentially into $q-1$ sets $\{\mathcal{G}_j : 2 \leq i \leq q\}$ such that $\mathcal{G}_j$ consists of $2^j \frac{8}{\epsilon} \frac{2^i}{w_i}$ elements passing the filter between $(2^j \frac{8}{\epsilon} \frac{2^i}{w_i} + 1)$th time and $2^{j+1} \frac{8}{\epsilon} \frac{2^i}{w_i}$th time. GK-algorithm is run on $\mathcal{G}_j$ regarding $\epsilon_1(\frac{1}{2 \times 2^\alpha})^{j-1}$.

Algorithm 4 gives a description of the algorithm Approximate().

Algorithm 4 : Approximate

Input: $\epsilon, \alpha > 0, \overline{S_i}$.
Output: ϱ_i.
Description:

```
if i ≤ i_0 − 1 then
    w_i := 1
else
    w_i := 2^{i−i_0+1};
end if
j := 1; K := 0; ε_1 = ε(1 − 1/2^α)/4;
while New element e' from S̄_i do
    get e_c from Algorithm-MERGE (∪_{l=1}^{j} ρ_l);
    if v(e') < v(e_c) then
        K := K + 1;
        feed GK-algorithm with e' for continuously building ρ_j
        with precision guarantee ε_1(1/(2×2^α))^{j−1};
        if K = 2^j (8/ε) × (2^i/w_i) then
            j := j + 1; K := 0;
        end if
    end if
end while
Return Algorithm-MERGE (∪_{l=1}^{j} w_i ⊗ ρ_l) (as ϱ_i);
```

Remark 1. *To further reduce the space, we also use a truncation mechanism similar to that in section 3.3 to filter out the tuples (if possible) in the existing sketch every time when a new element is inserted into it. However, to maintain the space guarantee in GK-algorithm, we do not filter out the tuples from current ρ_l where GK-algorithm is still running.*

To control the worst case space requirement in Algorithm 4, we keep n_0 as a threshold. Once the current total number of tuples kept exceeds n_0, the algorithm changes the mode and runs the filter-out (truncation) only mode for new coming elements as well as the existing tuples in the current sketch. This includes all ρ_l.

4.4 Precision and Space Complexity

We first show that the sketches produced by the algorithm Compression, Algorithm 3, follow SK1-SK3.

Let $\chi_0 = \varrho'_0$. Clearly, χ_0 meets SK1-SK4 regarding R'_0 and ϵ.

For $1 \leq i \leq \lceil \log \frac{\epsilon N}{8} \rceil - 1$, let $\chi_i =$ Algorithm-MERGE $(\cup_{j=0}^{i}(\varrho'_j) \cup \varrho_i)$.

Theorem 5. *For $1 \leq i \leq \lceil \log \frac{\epsilon N}{8} \rceil - 1$, χ_i meets SK1-SK3 regarding ϵ and $R'_i = [2^i \frac{8}{\epsilon}, 2^{i+1} \frac{8}{\epsilon}]$.*

Proof. Suppose that $\varrho_i :=$ Algorithm-MERGE $(\cup_{j=1}^{q} w_i \otimes \rho_l)$ is output by Algorithm 4. By Theorem 4, ϱ_j follows GK1-GK6 regarding $N'_i = \frac{2^i}{w_i} \sum_{j=1}^{q} 2^{j-1} \frac{8}{\epsilon}$ and

$$\lambda'_i = \frac{\epsilon_1 w_i \frac{2^i}{w_i} \frac{8}{\epsilon} \sum_{j=1}^{q} (\frac{1}{2^\alpha})^{j-1}}{N'_i} \leq \frac{\frac{\epsilon}{4} \times (2^i \frac{8}{\epsilon})}{N'_i}$$

Applying Theorem 4 to χ_i, it is immediate that χ_i satisfies GK1-GK6 regarding N_i and

$$\lambda''_i = \frac{\frac{\epsilon}{8} \sum_{j=1}^{i} (2^j \frac{8}{\epsilon}) + \lambda'_i N'_i}{N_i} \leq \frac{\frac{\epsilon}{2} \times 2^i \frac{8}{\epsilon}}{N_i}$$

It is immediate that χ_i meet SK1-3 regarding ϵ and $R'_i = [2^i \frac{8}{\epsilon}, 2^{i+1} \frac{8}{\epsilon}]$. □

Note that in Theorem 5, "a χ_i follows GK3" implies that we already replace $r(e)$ by $\sigma_{F_i}(e)$ where F_i denotes the set of sample elements sent to Algorithm 3 for generating χ_i; here F_i is slightly different than $\mathcal{F}_i$ in section 3 due to repartitioning R_0. Immediately, GK3 and Theorem 3 imply that χ_i violates SK4 with the probability less than δ for R'_i.

Clearly, for a RE query Algorithm 2 can be modified by checking SK3 using the first "hit" paradigm [15]. The theorem below is immediate.

Theorem 6. *The sketches $\{\chi_i : 0 \leq i \leq \lceil \log \frac{\epsilon N}{8} \rceil - 1\}$ can give an ϵ-approximate answer to a RE query with confidence at least $1 - \delta$.*

Next we show the space requirement in Algorithm 4. At each level i, we first estimate the total number of elements passing the filter. Note that in S_i $(i \geq 1)$, an element e passes the filter means that $v(e) < v(e_l)$ where e_l is the current cut-off element; that is, e_l is the lowest ranked element whose $\tau_l^- \geq 2^{i+1} \frac{8}{\epsilon} + \epsilon(2^i \frac{8}{\epsilon})$ in the current χ_i. As χ_i follows SK4, it is immediate that $\tau_j^- \leq \tau_j^+ \leq (2+3\epsilon/2)(2^i \frac{8}{\epsilon})$.

Theorem 7. *Let $a_i = (2 + 3\epsilon/2)(2^i \frac{8}{\epsilon})$. Suppose that each element's value distribution is the same and independent with each other. Then, the expected number N_s of elements e in S_i passed the filter during Algorithm 4 is not greater than*

$$\frac{a_i}{w_i}(\ln \frac{N}{a_i} + C), \tag{17}$$

where C is an constant and w_i is the weight in S_i.

Proof. According to the data distributions' assumption, each new element has an equal opportunity to take any rank regarding the existing sample elements. Suppose that at the time when an element is sampled from a jth segment, I_j, with 2^i elements; and suppose that there are l sample elements that are before the cut-off element e_l, as specified above, including e_l. Then each element e' in I_j has the probability $\frac{l}{j}$ to pass the filter.

Clearly, $w_i l \leq a_i$ as the difference of two adjacent tuples' τ^- values in the sketch is at least w_i according to GK-algorithm and Algorithm-MERGE [2, 15, 22]. Thus, a sampled element in I_j has the probability at most $\frac{a_i/w_i}{j}$ to be kept. Consequently, the expected number N_s of the total elements pass the filter has the property: $N_s \leq \frac{a_i}{w_i}(\sum_{j=a_i/w_i}^{N_i/w_i} \frac{1}{j} + 1)$. Thus, the theorem holds. □

Theorem 8. *Suppose that the elements' distributions satisfy the conditions in Theorem 7. Then, in Algorithm 4 at layer i* $\sum_{\forall j} |\rho_j|$ *is expected to be* $O(\frac{1}{\epsilon} \log \frac{a_i}{w_i} \frac{\ln^{1+\alpha} \frac{N}{a_i}}{1-1/2^\alpha})$, *where* $\alpha > 0$.

Proof. By the space result in GK-algorithm [15], we have $|\rho_j| \leq C\frac{4(2\times 2^\alpha)^{j-1}}{\epsilon(1-1/2^\alpha)} \log(\frac{(1-1/2^\alpha)\epsilon/4}{(2\times 2^\alpha)^{j-1}} 2^{j-1}(\frac{8}{\epsilon}\frac{2^i}{w_i}))$. This implies that $|\rho_j| = O(\frac{(2\times 2^\alpha)^{j-1}}{\epsilon(1-1/2^\alpha)} \log \frac{a_i}{w_i})$.

Assume there are k such ρ_j. According to Theorem 7, $\sum_{j=1}^{k-1} 2^{j-1} \frac{8}{\epsilon}\frac{2^i}{w_i} \leq \frac{a_i}{w_i}(\ln \frac{N}{a_i} + c)$.

By an elementary calculation, it is immediate that $k \leq C' \log_2(\ln N/a_i + c)$. Again, by elementary calculation it is immediate that $\sum_{j=1}^{k} |\rho_j| = O(\frac{1}{\epsilon} \log \frac{a_i}{w_i} \frac{\ln^{1+\alpha} \frac{N}{a_i}}{1-1/2^\alpha})$. □

In fact, we can show that Algorithm 3 has the following space guarantee combining with Algorithm 1.

Corollary 1. *Suppose that the elements' distributions satisfy the conditions in Theorem 7. Then, the space (i.e., the maximum number of elements in the sketches) in Algorithm 3 is expected to be* $O(\frac{1}{\epsilon} \log(\frac{1}{\epsilon} \log \frac{1}{\delta}) \frac{\ln^{2+\alpha} \epsilon N}{1-1/2^\alpha})$ *for a* $\alpha > 0$.

Proof. Note that $a_i/w_i \leq n_0$ and the space requirement by Algorithm 4 are dominant factors. There are totally $O(\log \epsilon N)$ layers. □

As described in Remark 1, once the total number of tuples in ϱ_i exceeds n_0 the algorithm just stops any further call of GK-algorithm. Instead, we filter out both new coming element and the existing sketch tuples whenever necessary. It is immediate that the number of elements kept is less than $\frac{a_i}{w_i}$ which is bounded by $O(n_0)$ for $i \geq i_0$, and $O(\frac{2^i}{\epsilon})$ for $i \leq i_0 - 1$. Consequently, the worst case space requirement in Algorithm 3 is $O(n_0 \log \epsilon^2 N + n_0) = O(\frac{1}{\epsilon^2} \log \frac{1}{\delta} \log \epsilon^2 N)$.

4.5 Final Notes

As with the algorithms in section 3, we do not materialize the results by Algorithm-MERGE for queries and filters. Instead, for each subsketch we maintain a tail and a head. A tail gives the latest tuple that has been added to the global sketch. To filter-out the existing tuples in the local sketches, we need only to examine those tails iteratively. Should any existing tuples be filtered out appropriately as described in Remark 1, the cut-off element must be one of those tails. So, we maintain a max-heap on the tails for quickly finding the cut-off element without running Algorithm-MERGE. Note that once GK-algorithm compresses or inserts a new element, we may have to update the heap once. Therefore, the complexity for processing each element at one level is $O(C_{gk} + C_{heap})$ if we also maintain a min-heap on heads. here, C_{gk} is the running costs of GK-algorithm per element and C_{heap} is the costs of one update to a heap.

It is clear that queries can be processed in a similar way as Algorithm-MERGE.

5 Performance Evaluation

In this section, we present results of a performance evaluation of our algorithms. We use the Algorithm 1 in [7] and the algorithm in [19] as benchmark algorithms to evaluate our techniques in this paper. The following algorithms are implemented and evaluated:

MR: The multi-layer randomization technique (Algorithm 1) combining with a simple space reduction technique presented in section 3.3.

MRC: The combination of Algorithm 1 and Algorithm 3, as well as the sub-algorithms in section 4.

CKMS: The deterministic approximate algorithm (Algorithm 1) in [7], using $\phi^k N \equiv 1$, as described in section 2.2.

GZ: The randomization algorithm proposed in [19] with an advance knowledge of N.

All experiments have been carried out on a PC with Intel P4 2.8GHz CPU and 1G memory. The operating system is Debian Linux. We implement the algorithms in C++ and compile them with gnu Gcc 3.3.4. We implement CKMS using *TREE* and *BATCH* structures as suggested in [7] for a speed-up. For GZ algorithm, given N, δ, and ϵ the sample space is fixed and could be extremely large due to the factor $O(\frac{1}{\epsilon^3})$. In case that the space required exceeds stream size N, we just keep all data elements.

Notation	Definition (Default Values)
d	Stream Model (Real data DP.)
N	Stream size (10M)
ϵ	Guaranteed precision (0.02)
$1-\delta$	Confidence (0.99)
$\mathcal{R}$	set of ranks ($\{\, i \times 20K\colon 1 \leq i \leq 500\}$)
α	Compress factor (1)

Table 2. System Parameters.

We have done the performance evaluations against the parameters that may affect performance studies. They are listed in Table 2.

We used five types of building blocks to generate data to evaluate the different algorithms. In the *Uni* (*Nor*) model, each element value follows uniform (normal) distribution. In the *Sort* (*Rev*) model, elements values are sorted by a non-decreasing (non-increasing) order. A stream of data elements in the *Semi* model is partitioned into groups with the average group size $2K$ such that values in later groups are larger than those in earlier groups, while data values within each group are in a random order.

In our experiments, to avoid favouring one particular stream order we generate a "heterogeneous" data distribution model - *Htr*. In the Htr model, a stream is divided into blocks with average size 2K. Note that different sizes of value domains usually do not affect space requirements if the domain sizes are large enough to allow each element to take a distinct value. In our Htr data, for each block we use a value domain with 5000 distinct values. For each pair of consecutive blocks, the intersection of the two value domains is randomly selected from length 0 (no intersection) to 5000 (using the same value domain). Moreover, for the elements in each block we randomly choose a stream model from one of Uni, Nor, Sort, and Rev.

We also evaluate the performance of the algorithms against a real data set from NYSE (New York Stock Exchange). The data set contains stock transaction records of DELL from Dec 3rd 2001 to Oct 31st 2002 where for each transaction, the average price per volume is recorded up to 8 decimal digits. We use the average prices as data values and the data set is called *DP* with about 11 million data elements (transaction records). The arrival order of elements is based on transactions' time stamps.

Finally , we test the performance of the algorithms over a data stream with very large volume. As there is no real single stock data with more than 11 million elements at hand, we concatenate 100 million transaction records from some 18 stocks (DELL, CSCO, SUN, etc) of NYSE during the trade period Nov/2000 - Oct/2002. Again, we use average price of each transaction as an element value. The whole data set size is about 2G and is named GIGR.

5.1 Space Evaluation

In this subsection, we do a performance evaluation on the space efficiency of our techniques. We normalized by the maximum number of tuples in a sketch during a continuous processing. We also record the ratio of such maximum number of tuples in a sketch over the number of data elements in the current stream, which is called *space ratio*.

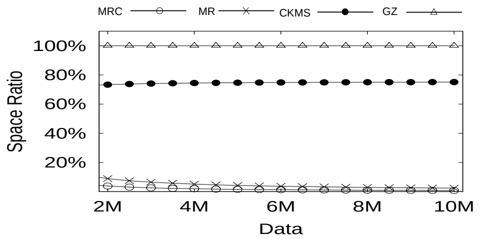


Figure 2. Linear Space by CKMS-algorithm

In our first experiment, we run algorithms GZ, CKMS, MR, MRC against a data stream constructed in the way as described in Example 2 (Section 2). We set the size of each batch to $2K$ and the whole data stream size to 10M, while other parameters are set to their default values. For each algorithm, we record the space ratios at a set of certain moments, as depicted in Figure 2. The space ratios in these 4 algorithms are reported in Figure 2, respectively. Figure 2 clearly demonstrates a linear space requirement($0.7N$) by CKMS for this data. Note that in this experiment, the arrival order of element values is specially designed in an "anti-CKMS" way to show its worst case. GZ requires the space N because the small value $\epsilon = 0.02$ cause the sample size larger than N.

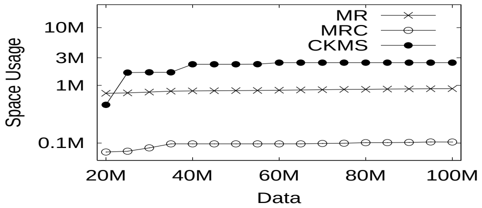


Figure 3. 2G Real Data: GIGR

In our second experiment, we evaluate the algorithms, GZ, CKMS, MR, and MRC, against the Gigabytes data (GIGR) using the system default values. This time, we record the space requirement in terms of number of elements. Figure 3 reports the results. Here we observe that CKMS, when used to obtain relative error at all ranks, can suffer on real data, not just on the synthetic data observed above. Note that we did not show the space requirement of GZ as again, the sample size exceeds the total number of the elements.

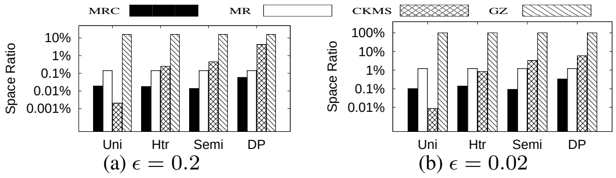


Figure 4. Space Efficiency over Diff. Stream Models

In the third experiment, we evaluate the space efficiency of MR, MRC, CKMS, and GZ against several different data streams (in particular, Uni, Semi, Htr, DP) with the same size 10M (millions). Note that DP is a subset of GIGR. We set $\epsilon = 0.2$ and 0.02, respectively, while the other parameters use default values. Figure 4 reports the results. It is interesting to note that CKMS performs very well against the Uni model. However, our MRC algorithm has the best performance against other data sets including the real data set; it outperforms CKMS and GZ by at least about an order of magnitude. Moreover, MRC has a quite stable good performance against all these data, while GZ does not perform very well against all these data, again, due to the cubic order of $\frac{1}{\epsilon}$.

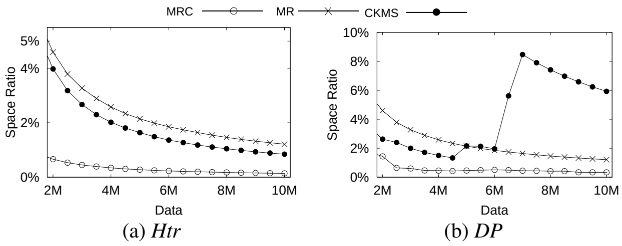
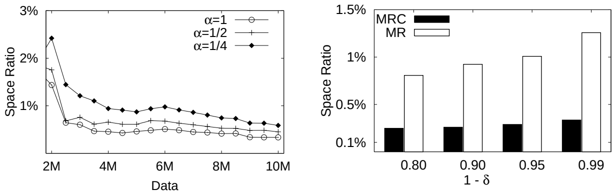


Figure 5. Space Efficiency with $\epsilon = 0.02$

Now, we evaluate the space efficiency as a function of data stream length. We run this set of experiments against two data streams – Htr and DP (real data) – and let other parameters use system default values. Since the space requirement of GZ exceeds N, the space ratio for GZ is always 100%. *Therefore, we exclude the space ratio from GZ in our graphs.* The results are reported in Figure 5. This experiment further validates the space efficiency of MRC. We also note that the performance of CKMS is not very stable.

Remark 2. *To prevent any possible confusion, we want to point out that the space ratio drops as the stream size increases does not mean that the overall space usage drops. In fact, the required space in this paper refers to maximum temporary storage during the computation; consequently it never drops as a stream size continuously increases.*

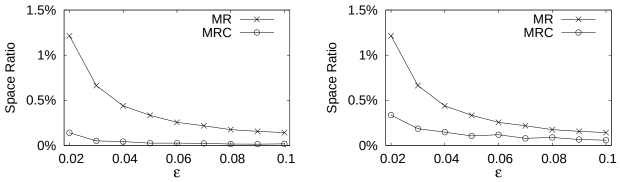
Remark 3. *Based on space usage demonstrated so far against MR and MRC, we do not further evaluate CKMS for the rest of our performance studies since time cost is directly related to space usage (MRC and CKMS apply a similar space reduction technique, which is linear in the number of samples). We also do not present results from GZ since, in our experiments below, the space required by GZ exceeds N, when $\epsilon \leq 0.05$, due to the cubic order of $\frac{1}{\epsilon}$.*



(a) DP on Diff. α (b) DP on Diff. $1 - \delta$

Figure 6. Space Efficiency for Diff. α and $1 - \delta$

In the fifth set of experiments, we evaluate the impact of α (in Algorithm 4). We choose α=$\frac{1}{4}$,$\frac{1}{2}$,1 and run MRC against 10M real data *DP* with other parameters using default values. Our experiment results (Figure 6(a)) demonstrate that the space requirements are close though $\alpha = 1$ gain the edge. We also evaluate the impact of confidence $1-\delta$ on MR and MRC. The experiment is conducted against different $1 - \delta$ values: 0.8, 0.9, 0.95, 0.99, while other parameters adopt default values. The results are reported in Figure 6(b). As expected, the memory space needed by MR and MRC increases when confidence increases.



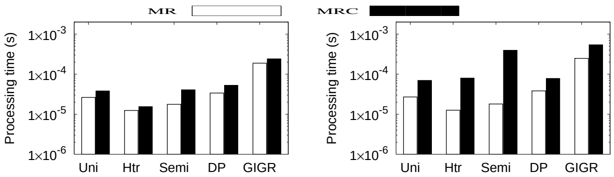
(a) *Htr* (b) *DP*

Figure 7. Space Consumption against Diff. ϵ

In the sixth set of experiments, we examine the effect of different ϵ values. Htr and DP data streams (data sets) are used against different ϵ values, while other parameters adopt default values. The results are reported in Figure 7. The experiment further demonstrates the space efficiency of MRC.

5.2 Evaluating Processing Costs

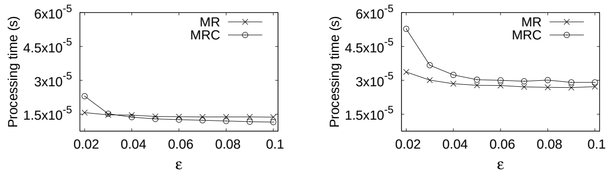
In this subsection, we evaluate the sketch maintenance costs of MR and MRC. We want to measure the processing time per element. We record the average time needed for processing every batch of 20K elements; this estimate is used as the processing delay "per element". In addition, we estimate the *maximum* delay per element using the maximum average delay over all batches.



(a) Avg. Processing Delay (b) Max. Processing Delay

Figure 8. Delay per Element with $\epsilon = 0.02$

In the first set of experiments, we evaluate the average delay and maximum delay per element with respect to $\epsilon = 0.02$ and the five streams (Uni, Htr, Semi, DP, GIGR), while other parameters adopt system default values. Figure 8 reports the experiment results. ¿From our results on maximum and average delay per data element, we can draw a very positive conclusion below. MRC can support a real time computation for a stream with arrival rate from about 7K elements (GIGR) per second to 60K (Htr) elements per second on average for a small precision $\epsilon = 0.02$ and high confidence 0.99, while in the worst case it can still support a stream with the arrival rate from about 1.3K elements/second (GIGR) to 10K elements (Uni). MR is even faster since it doesn't have to continually prune the space using a compression procedure.



(a) *Htr* (b) *DP*

Figure 9. Avg. Delay per Element vs Diff. ϵ

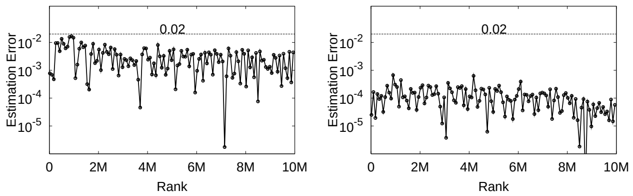
In the second set of evaluations, we examine the effect of different relative error guarantees - ϵ. We conduct experiments for two streams Htr and DP, while other parameters use default values. The experiment results are reported in Figure 9.

5.3 Evaluating Accuracy

We use the real data stream DP to evaluate the accuracy of MR and MRC. It is immediate that if we want to guarantee the precision by the confidence $1 - \delta$ for processing t queries, we need to enlarge the space requirement a bit by replacing δ by $\frac{\delta}{t}$ in MR and MRC. However, in our evaluation we discount such a requirement.

We evaluate the actual relative errors by MR and MRC for RE queries at different ranks spanning 1 to N. We issue the query set $\mathcal{R}$ (500 queries) with the default value after the sketches are created. We run the two algorithms against DP data stream with default settings for other parameters.

Figure 10(a) reports the relative errors obtained from the sketch generated by MRC, while Figure 10(b) reports the relative errors from the sketches by MR. It demonstrates that no relative errors exceeds the error guarantee ($\epsilon = 0.02$) in MRC and MR, respectively. As expected that MR pro-



(a) MRC (b) MR

Figure 10. Actual Relative Errors for $\epsilon = 0.02$

vides higher accuracy than that by MRC. This is because that MR generates the sample set with a larger size.

5.4 Summary

As a short summary, our comprehensive performance study clearly demonstrates that the proposed algorithms are both effective and efficient. MR and MRC can both support real time processing of rapid streams. Note that MRC requires much less space than MR does, while MR provides better accuracy. Although the deterministic algorithm CKMS [7] requires a linear space $\Omega(N)$ in the worst case, it still performs reasonably well in our performance study, especially for the Uni stream data. GZ algorithm generally requires large space for a small ϵ value; consequently, it is not applicable to situations where ϵ is small. Overall, the algorithm MRC could be significantly more space-efficient and more cost-efficient than CKMS, by an order of magnitude, in our experiments. Therefore, in applications where small space guarantee is crucial, MRC is the best choice.

6 Conclusions

In this paper, we developed novel techniques for maintaining online order sketches so that rank queries can be answered with a relative error guarantee. This work provides the first space-efficient streaming techniques that process approximate rank queries with relative error guarantees, without advance knowledge of N. Besides proven accuracy and space guarantees, our algorithms are also efficient enough to support on-line computation of very high speed data streams with element arrival rate up to 60K/second. In fact, our technique may be immediately extended to the problem of counting inversions with similar space requirements. For space limits, we omit the discussions in this paper.

Future work includes investigating if tighter space bounds can be found for our algorithms, as well as studying this problem over distributed data streams.

References

[1] M. Ajtai, I. S. Jayram, R. Kumar, and D. Sivakumar. Approximate counting of inversions in a data stream. In *STOC 2002*.

[2] A. Arasu and G. S. Manku. Approximate counts and quantiles over sliding windows. In *PODS04*, 2004.

[3] B. Babcock, S. Babu, M. Datar, R. Motwani, and J. Widom. Models and issues in data stream systems. In *PODS'02*.

[4] E. Cohen and H. Kaplan. Spatially-decaying aggregation over a network: model and algorithms. In *SIGMOD'04*.

[5] C. Cormode, M. Garofalakis, S. Muthukrishnan, and R. Rastogi. Holistic aggregates in a networked world: Distributed tracking of approximate quantiles. In *SIGMOD2005*, 2005.

[6] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Finding hierarchical heavy hitters in data streams. In *VLDB 2003*.

[7] G. Cormode, F. Korn, S. Muthukrishnan, and D. Srivastava. Effective computation of biased quantiles over data streams. In *ICDE'05*, 2005.

[8] A. Das, J. Gehrke, and M. Riedewald. Approximate join processing over data streams. In *SIGMOD'03*, 2003.

[9] M. Datar, A. Gionis, P. Indyk, and R. Motwani. Maintaining stream statistics over sliding windows: (extended abstract). In *SODA03*, 2002.

[10] A. Dobra, M. Garofalakis, J. Gehrke, and R. Rastogi. Processing complex aggregate queries over data streams. In *SIGMOD*, 2002.

[11] D. Dubhashi and A. Panconesi. *Concentration of Measure for Computer Scientists: Chapter 1, pp 12*. www.cs.unibo.it/˜ pancones/papers.html, 1998.

[12] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank aggregation methods for the web. In *WWW01*, 2001.

[13] A. Gilbert, Y. Kotidis, S. Muthukrishnan, and M. Strauss. How to summarize the universe: Dynamic maintenance of quantiles. In *VLDB2002*, 2002.

[14] M. Greenwald and S. Khanna. Power-conserving computation of order-statistics over sensor networks. In *PODS'04*.

[15] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *SIGMOD'01*, 2001.

[16] S. Guha and N. Koudas. Approximating a data stream for querying and estimation: Algorithms and performance evaluation. In *ICDE 2002*, 2002.

[17] S. Guha, N. Koudas, and K. Shim. Data-streams and histograms. In *STOC 2001*, 2001.

[18] S. Guha, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams. In *FOCS'00*, 2000.

[19] A. Gupta and F. Zane. Counting inversions in lists. In *SODA'03*, 2003.

[20] J.I.Munro and M.S.Paterson. Selection and sorting wiith limited storage. In *TCS12*, 1980.

[21] T. Johnson, S. Muthukrishnan, T. Dasu, , and V. Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *SIGMOD*, 2002.

[22] X. Lin, H. Lu, J. Xu, and J. X. Yu. Continuously maintaining quantile summaries of the most recent n elements over a data stream. In *ICDE*, 2004.

[23] S. Manganelli and R. Engle. Value at risk models in finance. *European Central Bank Working Paper Series No. 75*, 2001.

[24] G. Manku and R. Motwani. Approximate frequency counts over data streams. In *VLDB 2002*, 2002.

[25] G. S. Manku, S. Rajagopalan, and B. G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *SIGMOD*, 1999.

[26] V. Poosala, P. Haas, Y. Ioannidis, and E. Shekita. Improved histograms for selectivity estimation of range predicates. In *VLDB'96*, 1996.

[27] V. Poosala and Y. Ioannidis. Estimation of query-result distribution and its application in parallel-join load balancing. In *The VLDB Journal*, 1996.

[28] J. Yu, Z. Chong, H. Lu, and A. Zhou. False positive or false negative: Mining frequent itemsets from high speed transactional data streams. In *VLDB'04*, 2004.