Relative Error Streaming Quantiles

Graham Cormode
University of Warwick
Coventry, UK
G.Cormode@warwick.ac.uk

Zohar Karnin
Amazon
USA
zkarnin@gmail.com

Edo Liberty
Pinecone
San Mateo, CA, USA
edo@edoliberty.com

Justin Thaler
Georgetown University
Washington, D.C., USA
justin.thaler@georgetown.edu

Pavel Veselý
Computer Science Institute
Charles University
Prague, Czech Republic
vesely@iuuk.mff.cuni.cz

ABSTRACT

Approximating ranks, quantiles, and distributions over streaming data is a central task in data analysis and monitoring. Given a stream of n items from a data universe $\mathcal{U}$ equipped with a total order, the task is to compute a sketch (data structure) of size poly($\log(n), 1/\varepsilon$). Given the sketch and a query item $y \in \mathcal{U}$, one should be able to approximate its rank in the stream, i.e., the number of stream elements smaller than or equal to y.

Most works to date focused on additive εn error approximation, culminating in the KLL sketch that achieved optimal asymptotic behavior. This paper investigates *multiplicative* $(1 \pm \varepsilon)$-error approximations to the rank. Practical motivation for multiplicative error stems from demands to understand the tails of distributions, and hence for sketches to be more accurate near extreme values.

The most space-efficient algorithms due to prior work store either $O(\log(\varepsilon^2 n)/\varepsilon^2)$ or $O(\log^3(\varepsilon n)/\varepsilon)$ universe items. This paper presents a randomized algorithm storing $O(\log^{1.5}(\varepsilon n)/\varepsilon)$ items, which is within an $O(\sqrt{\log(\varepsilon n)})$ factor of optimal. The algorithm does not require prior knowledge of the stream length and is fully mergeable, rendering it suitable for parallel and distributed computing environments.

1 INTRODUCTION

Understanding the distribution of data is a fundamental task in data monitoring and analysis. The problem of streaming quantile approximation captures this task in the context of massive or distributed datasets.

The problem is as follows. Let $\sigma = \{x_1, \ldots, x_n\}$ be a stream of items, all drawn from a data universe $\mathcal{U}$ equipped with a total order. For any $y \in \mathcal{U}$, let $R(y; \sigma) = |\{x_i \mid x_i \leq y\}|$ be the rank of y in the stream. When σ is clear from context, we write $R(y)$. The objective is to process the stream while storing a small number of items, and then use those to approximate $R(y)$ for any $y \in \mathcal{U}$. A guarantee for an approximation $\hat{R}(y)$ is *additive* if $|\hat{R}(y) - R(y)| \leq \varepsilon n$, and *multiplicative* or *relative* if $|\hat{R}(y) - R(y)| \leq \varepsilon R(y)$.

A long line of work has focused on achieving additive error guarantees [2, 3, 8, 9, 12, 14, 18, 19]. However, additive error is not appropriate for many applications. Indeed, often the primary purpose of computing quantiles is to understand the tails of the data distribution. When $R(y) \ll n$, a multiplicative guarantee is much more accurate and thus harder to obtain. As pointed out by Cormode et al. [4], a solution to this problem would also yield high accuracy when $n - R(y) \ll n$, by running the same algorithm with the reversed total ordering (simply negating the comparator).

A quintessential application that demands relative error is monitoring network latencies. In practice, one often tracks response time percentiles 50, 90, 99, and 99.9. This is because latencies are heavily long-tailed. For example, Masson et al. [16] report that for web response times, the 98.5th percentile can be as small as 2 seconds while the 99.5th percentile can be as large as 20 seconds. These unusually long response times affect network dynamics [4] and are problematic for users. Furthermore, as argued by Tene in his talk about measuring latency [22], one needs to look at extreme percentiles such as 99.995 to determine the latency such that only about 1% of users experience a larger latency during a web session with several page loads. Hence, highly accurate rank approximations are required for items y whose rank is very large ($n - R(y) \ll n$); this is precisely the requirement captured by the multiplicative error guarantee.

Achieving multiplicative guarantees is known to be *strictly* harder than additive ones. There are comparison-based additive error algorithms that store just $\Theta(\varepsilon^{-1})$ items for constant failure probability [12], which is optimal. In particular, to achieve additive error, the number of items stored may be independent of the stream length n. In contrast, any algorithm achieving multiplicative error must use $\Omega\left(\varepsilon^{-1} \cdot \log(\varepsilon n) \cdot \log(\varepsilon|\mathcal{U}|)\right)$ bits of space, which holds

even for offline, non-comparison-based algorithms (see [4, Theorem 2] and Appendix A).[1]

The best known algorithms achieving multiplicative error guarantees are as follows. Zhang et al. [24] give a randomized algorithm storing $O(\varepsilon^{-2} \cdot \log(\varepsilon^2 n))$ universe items. This is essentially a ε^{-1} factor away from the aforementioned lower bound. There is also an algorithm of Cormode et al. [5] that stores $O(\varepsilon^{-1} \cdot \log(\varepsilon n) \cdot \log |\mathcal{U}|)$ items. However, this algorithm builds a binary tree over the data universe $\mathcal{U}$ *a priori* and is inapplicable when $\mathcal{U}$ is huge or even unbounded, e.g., if $\mathcal{U}$ consists of all strings of arbitrary length. Finally, Zhang and Wang [23] designed a deterministic algorithm requiring $O(\varepsilon^{-1} \cdot \log^3(\varepsilon n))$ space. Very recent work of Cormode and Veselý [6] proves an $\Omega(\varepsilon^{-1} \cdot \log^2(\varepsilon n))$ lower bound for the number of items stored by deterministic comparison-based algorithms, which is within a $\log(\varepsilon n)$ factor of Zhang and Wang's upper bound.

Despite both the practical and theoretical importance of multiplicative error (which is arguably an even more natural goal than additive error), there has been no progress on upper bounds, i.e., no new algorithms, since 2007.

In this work, we give a randomized algorithm that maintains the optimal linear dependence on $1/\varepsilon$ achieved by Zhang and Wang, with a significantly improved dependence on the stream length. Namely, we design a comparison-based, one-pass streaming algorithm that given $\varepsilon > 0$ and $\delta > 0$, computes a sketch consisting of $O\left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon n) \cdot \sqrt{\log(1/\delta)}\right)$ universe items (for a sufficiently small ε; see Theorem 1), and from which an estimate $\hat{R}(y)$ of $R(y)$ can be derived for every $y \in \mathcal{U}$. For any fixed $y \in \mathcal{U}$, with probability at least $1 - \delta$, the returned estimate satisfies the multiplicative error guarantee $|\hat{R}(y) - R(y)| \leq \varepsilon R(y)$. Ours is the first algorithm to be strictly more space efficient than *any* deterministic comparison-based algorithm (owing to the $\Omega(\varepsilon^{-1} \log^2(\varepsilon n))$ lower bound in [6]) and is within an $\tilde{O}(\sqrt{\log(\varepsilon n)})$ factor of the known lower bound for randomized algorithms achieving multiplicative error. (In this manuscript, the $\tilde{O}$ notation hides factors polynomial in $\log(1/\delta)$, $\log \log n$, and $\log(1/\varepsilon)$.)

We also show that the algorithm processes the input stream efficiently. In particular, the amortized update time of the algorithm is a logarithm of the space bound, which equals $O\left(\log(\varepsilon^{-1}) + \log \log(n) + \log \log(1/\delta)\right)$; see Section 4 for details.

Mergeability. The ability to merge sketches of different streams to get an accurate sketch for the concatenation of the streams is highly significant both in theory [1] and in practice [20]. Such mergeable summaries enable trivial, automatic parallelization and distribution of processing massive data sets, by arbitrarily splitting the data up into pieces, summarizing each piece separately, and then merging the results.

We show that our sketch is *fully mergeable*. This means that, if a data set is split into pieces and each piece is summarized separately, and the resulting summaries are combined via an arbitrary sequence

of merge operations, the algorithm maintains the same relative error guarantees while using essentially the same space as if the entire data set had been processed as a single stream (the details are deferred to the full version[2]).

The following theorem is the main result of this paper. We stress that our algorithm does *not* require any advance knowledge about n, the total size of input, which indeed may not be available in many applications.[3]

Theorem 1. *Let $0 < \delta \leq 0.5$ and $0 < \varepsilon \leq 1$ be parameters satisfying $\varepsilon \leq 4/\sqrt[4]{2 \log_2(n)}$. There is a randomized, comparison-based, one-pass streaming algorithm that, when processing a data stream consisting of n items, produces a summary S satisfying the following property. Given S, for any $y \in \mathcal{U}$ one can derive an estimate $\hat{R}(y)$ of $R(y)$ such that*

$$\Pr\left[|\hat{R}(y) - R(y)| \geq \varepsilon R(y)\right] < \delta,$$

where the probability is over the internal randomness of the streaming algorithm. If $\varepsilon \leq 4 \cdot \sqrt{\ln \frac{1}{\delta} / \log_2(\varepsilon n)}$, then the size of S is

$$O\left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon n) \cdot \sqrt{\log\left(\frac{1}{\delta}\right)}\right);$$

otherwise, storing S takes $O\left(\log^2(\varepsilon n)\right)$ memory words. Moreover, the summary produced is fully mergeable.

Note that the assumption $\varepsilon \leq 4/\sqrt[4]{2 \log_2(n)}$ is *very* weak as for any $n \leq 2^{128}$, it holds that $\sqrt[4]{2 \log_2(n)} \leq 4$, rendering the assumption vacuous in practical scenarios. Similarly, the space bound that holds in the case $\varepsilon \leq 4 \cdot \sqrt{\ln \frac{1}{\delta} / \log_2(\varepsilon n)}$ certainly applies for values of ε and n encountered in practice (e.g., for $n \leq 2^{64}$ and $\delta \leq 1/e$, this latter requirement is implied by $\varepsilon \leq 1/2$).

All-quantiles approximation. As a straightforward corollary of Theorem 1, we obtain a space-efficient algorithm whose estimates are simultaneously accurate for *all* $y \in \mathcal{U}$ with high probability. The proof is a standard use of the union bound combined with an epsilon-net argument; we include the proof in Appendix B.

Corollary 1 (All-Quantiles Approximation). *The error bound from Theorem 1 can be made to hold for all $y \in \mathcal{U}$ simultaneously with probability $1 - \delta$ while storing*

$$O\left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon n) \cdot \sqrt{\log\left(\frac{\log(\varepsilon n)}{\varepsilon \delta}\right)}\right)$$

stream items if $\varepsilon \leq O\left(\sqrt{\log \frac{1}{\varepsilon \delta} / \log(\varepsilon n)}\right)$ and $O\left(\log^2(\varepsilon n)\right)$ items otherwise.

---

[1]Intuitively, the reason additive-error sketches can achieve space independent of the stream length is because they can take a subsample of the stream of size about $\Theta(\varepsilon^{-2})$ and then sketch the subsample. For any fixed item, the additive error to its rank introduced by sampling is at most εn with high probability. When multiplicative error is required, one cannot subsample the input: for low-ranked items, the multiplicative error introduced by sampling will, with high probability, not be bounded by any constant.

---

[2]The full version of our paper is available at https://arxiv.org/abs/2004.01668.

[3]A proof-of-concept Python implementation of our algorithm is available at GitHub: https://github.com/edoliberty/streaming-quantiles/blob/master/relativeErrorSketch.py. A production-quality implementation in the Apache DataSketches library is available at https://datasketches.apache.org/.

Challenges and techniques. A starting point of the design of our algorithm is the KLL sketch [12] that achieves optimal accuracy-space trade-off for the additive error guarantee. The basic building block of the algorithm is a buffer, called a *compactor*, that receives an input stream of n items and outputs a stream of at most $n/2$ items, meant to "approximate" the input stream. The buffer simply stores items and once it is full, we sort the buffer, output all items stored at either odd or even indexes (with odd vs. even selected via an unbiased coin flip), and clear the contents of the buffer—this is called the *compaction operation*. Note that a randomly chosen half of items in the buffer is simply discarded, whereas the other half of items in the buffer is "output" by the compaction operation.

The overall KLL sketch is built as a sequence of at most $\log_2(n)$ such compactors, such that the output stream of a compactor is treated as the input stream of the next compactor. We thus think of the compactors as arranged into *levels*, with the first one at level 0. Similar compactors were already used, e.g., in [1, 13–15], and additional ideas are needed to get the optimal space bound for additive error, of $O(1/\epsilon)$ items stored across all compactors [12].

The compactor building block is not directly applicable to our setting for the following reasons. A first observation is that to achieve the relative error guarantee, we need to always store the $1/\epsilon$ smallest items. This is because the relative error guarantee demands that estimated ranks for the $1/\epsilon$ lowest-ranked items in the data stream are *exact*. If even a single one of these items is deleted from the summary, then these estimates will not be exact. Similarly, among the next $2/\epsilon$ smallest items, the summary must store essentially every other item to achieve multiplicative error. Among the next $4/\epsilon$ smallest items in the order, the sketch must store roughly every fourth item, and so on.

The following simple modification of the compactor from the KLL sketch indeed achieves the above. Each buffer of size B "protects" the $B/2$ smallest items stored inside, meaning that these items are not involved in any compaction (i.e., the compaction operation only removes the $B/2$ largest items from the buffer). Unfortunately, it turns out that this simple approach requires space $\Theta(\epsilon^{-2} \cdot \log(\epsilon^2 n))$, which merely matches the space bound achieved in [24], and in particular has a (quadratically) suboptimal dependence on $1/\epsilon$.

The key technical contribution of our work is to enhance this simple approach with a more sophisticated rule for selecting the number of protected items in each compaction. One solution that yields our upper bound is to choose this number in each compaction at random from an appropriate exponential distribution. However, to get a cleaner analysis and a better dependency on the failure probability δ, we in fact derandomize this distribution.

While the resulting algorithm is relatively simple, analyzing the error behavior brings new challenges that do not arise in the additive error setting. Roughly speaking, when analyzing the accuracy of the estimate for R(y) for any fixed item y, all error can be "attributed" to compaction operations. In the additive error setting, one may suppose that *every* compaction operation contributes to the error and still obtain a tight error analysis [12]. Unfortunately, this is not at all the case for relative error: as already indicated, to obtain our accuracy bounds it is essential to show that the estimate for any low-ranked item y is affected by very few compaction operations.

Thus, the first step of our analysis is to carefully bound the number of compactions on each level that affect the error for y, using a charging argument that relies on the derandomized exponential distribution to choose the number of protected items. To get a suitable bound on the variance of the error, we also need to show that the rank of y in the input stream to each compactor falls by about a factor of two at every level of the sketch. While this is intuitively true (since each compaction operation outputs a randomly chosen half of "unprotected" items stored in the compactor), it does not hold deterministically and hence requires a careful treatment in the analysis. Finally, we observe that the error in the estimate for y is a zero-mean sub-Gaussian variable with variance bounded as above, and thus applying a standard Chernoff tail bound yields our final accuracy guarantees for the estimated rank of y.

There are substantial additional technical difficulties to analyze the algorithm under an arbitrary sequence of merge operations, especially with no foreknowledge of the total size of the input. Most notably, when the input size is not known in advance, the parameters of the sketch must change as more inputs are processed. This makes obtaining a tight bound on the variance of the resulting estimates highly involved. In particular, as a sketch processes more and more inputs, it protects more and more items, which means that items appearing early in the stream may *not* be protected by the sketch, even though they *would have been protected* if they appeared later in the stream. Addressing this issue is reasonably simple in the streaming setting, because every time the sketch parameters need to change, one can afford to allocate an entirely new sketch with the updated parameters, without discarding the previous sketch(es); see Section 5 for details. Unfortunately, this simple approach does not work for a general sequence of merge operations, and we provide a much more intricate analysis to give a fully mergeable summary.

A second challenge when designing and analyzing merge operations arises from working with our *derandomized* exponential distribution, since this requires each compactor to maintain a "state" variable determining the current number of protected items, and these variables need to be "merged" appropriately. It turns out that the correct way to merge state variables is to take a bitwise OR of their binary representations. With this technique for merging state variables in hand, we extend the charging argument bounding the number of compactions affecting the error in any given estimate so as to handle an arbitrary sequence of merge operations.

Analysis with extremely small probability of failure. We close by giving an alternative analysis of our algorithm that achieves a space bound with an exponentially better (double logarithmic) dependence on $1/\delta$, compared to Theorem 1. However, this improved dependence on $1/\delta$ comes at the expense of the exponent of $\log(\epsilon n)$ increasing from 1.5 to 2. Formally, we obtain the following theorem, where we also show that it directly implies a deterministic space bound of $O(\epsilon^{-1} \cdot \log^3(\epsilon n))$, matching the state-of-the-art result in [23]. For simplicity, we only prove the theorem in the streaming setting, although we conjecture that an appropriately modified proof of Theorem 1 would yield the same result even when the sketch is built using merge operations. The formal proof is deferred to the full version of our paper.[2]

THEOREM 2. *For any parameters $0 < \delta \leq 0.5$ and $0 < \epsilon \leq 1$, there is a randomized, comparison-based, one-pass streaming algorithm that computes a sketch consisting of $O\left(\epsilon^{-1} \cdot \log^2(\epsilon n) \cdot \log\log(1/\delta)\right)$*

universe items, and from which an estimate $\hat{R}(y)$ of $R(y)$ can be derived for every $y \in \mathcal{U}$. For any fixed $y \in \mathcal{U}$, with probability at least $1 - \delta$, the returned estimate satisfies the multiplicative error guarantee $|\hat{R}(y) - R(y)| \le \varepsilon R(y)$.

We remark that this alternative analysis builds on an idea from [12] to analyze the top few levels of compactors deterministically rather than obtaining probabilistic guarantees on the errors introduced to estimates by these levels.

Organization of the paper. Since the proof of full mergeability in Theorem 1 is quite involved, we proceed in several steps of increasing complexity. We describe our sketch in the streaming setting in Section 2, where we also give a detailed but informal outline of the analysis. We then formally analyze the sketch in the streaming setting in Sections 3 and 4, also assuming that a polynomial upper bound on the stream length is known in advance. The space usage of the algorithm grows polynomially with the logarithm of this upper bound, so if this upper bound is at most n^c for some constant $c \ge 1$, then the space usage of the algorithm remains as stated in Theorem 1, with only the hidden constant factor changing. Then, in Section 5, we explain how to remove this assumption in the streaming setting, yielding an algorithm that works without *any* information about the final stream length.

The full description of the merge procedure and the analysis of the accuracy under an arbitrary sequence of merge operations is deferred to the full version of our paper[2] (for didactic purposes, we outline a simplified merge operation in Section 2.3).

1.1 Detailed Comparison to Prior Work

Some prior works on streaming quantiles consider queries to be *ranks* $r \in \{1, \ldots, n\}$, and the algorithm must identify an item $y \in \mathcal{U}$ such that $R(y)$ is close to r. In this work we focus on the dual problem, where we consider queries to be universe items $y \in \mathcal{U}$ and the algorithm must yield an accurate estimate for $R(y)$. Unless specified otherwise, algorithms described in this section directly solve both formulations (this holds for our algorithm as well). Algorithms are randomized unless stated otherwise. For simplicity, randomized algorithms are assumed to have constant failure probability δ. All reported space costs refer to the number of universe items stored.[4]

Additive Error. Manku, Rajagopalan, and Lindsay [14, 15] built on the work of Munro and Paterson [17] and gave a deterministic solution that stores at most $O(\varepsilon^{-1} \cdot \log^2(\varepsilon n))$ items, assuming prior knowledge of n. Greenwald and Khanna [10] created an intricate deterministic streaming algorithm that stores $O(\varepsilon^{-1} \cdot \log(\varepsilon n))$ items. This is the best known deterministic algorithm for this problem, with a matching lower bound for comparison-based streaming algorithms [6]. Agarwal, Cormode, Huang, Phillips, Wei, and Yi [1] provided a mergeable sketch of size $O(\varepsilon^{-1} \cdot \log^{1.5}(1/\varepsilon))$. This paper contains many ideas and observations that were used in later work. Felber and Ostrovsky [8] managed to reduce the space complexity to $O(\varepsilon^{-1} \cdot \log(1/\varepsilon))$ items by combining sampling with the Greenwald-Khanna sketches in non-trivial ways. Finally, Karnin, Lang, and

Liberty [12] resolved the problem by providing an $O(1/\varepsilon)$-space solution, which is optimal. For general (non-constant) failure probabilities δ, the space upper bound becomes $O(\varepsilon^{-1} \cdot \log \log(1/\delta))$, and they also prove a matching lower bound for comparison-based randomized algorithms, assuming δ is exponentially small in n.

Multiplicative Error. A large number of works sought to provide more accurate quantile estimates for low or high ranks. Only a handful offer solutions to the relative error quantiles problem (also sometimes called the biased quantiles problem) considered in this work. Gupta and Zane [11] gave an algorithm for relative error quantiles that stores $O(\varepsilon^{-3} \cdot \log^2(\varepsilon n))$ items, and use this to approximately count the number of inversions in a list; their algorithm requires prior knowledge of the stream length n. As previously mentioned, Zhang et al. [24] presented an algorithm storing $O(\varepsilon^{-2} \cdot \log(\varepsilon^2 n))$ universe items. Cormode et al. [5] designed a deterministic sketch storing $O(\varepsilon^{-1} \cdot \log(\varepsilon n) \cdot \log |\mathcal{U}|)$ items, which requires prior knowledge of the data universe $\mathcal{U}$. Their algorithm is inspired by the work of Shrivastava et al. [21] in the additive error setting and it is also mergeable (see [1, Section 3]). Zhang and Wang [23] gave a deterministic merge-and-prune algorithm storing $O(\varepsilon^{-1} \cdot \log^3(\varepsilon n))$ items, which can handle arbitrary merges with an upper bound on n, and streaming updates for unknown n. However, it does not tackle the most general case of merging without a prior bound on n. Cormode and Veselý [6] recently showed a space lower bound of $\Omega(\varepsilon^{-1} \cdot \log^2(\varepsilon n))$ items for any deterministic comparison-based algorithm.

Other related works that do not fully solve the relative error quantiles problem are as follows. Manku, Rajagopalan, and Lindsay [15] designed an algorithm that, for a specified number $\phi \in [0, 1]$, stores $O(\varepsilon^{-1} \cdot \log(1/\delta))$ items and can return an item y with $R(y)/n \in [(1 - \varepsilon)\phi, (1 + \varepsilon)\phi]$ (their algorithm requires prior knowledge of n). Cormode et al. [4] gave a deterministic algorithm that is meant to achieve error properties "in between" additive and relative error guarantees. That is, their algorithm aims to provide multiplicative guarantees only up to some minimum rank k; for items of rank below k, their solution only provides additive guarantees. Their algorithm does not solve the relative error quantiles problem: [24] observed that for adversarial item ordering, the algorithm of [4] requires linear space to achieve relative error for all ranks. Dunning and Ertl [7] describe a heuristic algorithm called t-digest that is intended to achieve relative error, but they provide no formal accuracy analysis.

Most recently, Masson, Rim, and Lee [16] introduced a new notion of error for quantile sketches (they also refer to their notion as "relative error", but it is quite distinct from the notion considered in this work). They require that for a query percentile $\phi \in [0, 1]$, if y denotes the item in the data stream satisfying $R(y) = \phi n$, then the algorithm should return an item $\hat{y} \in \mathcal{U}$ such that $|y - \hat{y}| \le \varepsilon \cdot |y|$. This definition only makes sense for data universes with a notion of magnitude and distance (e.g., numerical data), and the definition is not invariant to natural data transformations, such as incrementing every data item y by a large constant. It is also trivially achieved by maintaining a (mergeable) histogram with buckets $((1 + \epsilon)^i, (1 + \epsilon)^{i+1}]$. In contrast, the standard notion of relative error considered in this work does not refer to the data

---

[4]Apart from storing universe items, the algorithms may store, for example, bounds on ranks of stored items or some counters, but the number of such variables is proportional to the number of items stored or even smaller. Thus, the space bounds are in memory words, which can store any item or an integer with $O(\log(n + |\mathcal{U}|))$ bits.

items themselves, only to their ranks, and is arguably of more general applicability.

2 DESCRIPTION OF THE ALGORITHM

2.1 The Relative-Compactor Object

The crux of our algorithm is a building block that we call the relative-compactor. Roughly speaking, this object processes a stream of n items and outputs a stream of at most $n/2$ items (each "up-weighted" by a factor of 2), meant to "approximate" the input stream. It does so by maintaining a buffer of limited capacity.

Our complete sketch, described in Section 2.2 below, is composed of a sequence of relative-compactors, where the input of the $h+1$'th relative-compactor is the output of the h'th. With at most $\log_2(\varepsilon n)$ such relative-compactors, n being the length of the input stream, the output of the last relative-compactor is of size $O(1/\varepsilon)$, and hence can be stored in memory.

Compaction Operation. The basic subroutine used by our relative-compactor is a compaction operation. The input to a compaction operation is a list X of $2m$ items $x_1 \le x_2 \le \ldots \le x_{2m}$, and the output is a sequence Z of m items. This output is chosen to be one of the following two sequences, uniformly at random: Either $Z = \{x_{2i-1}\}_{i=1}^m$ or $Z = \{x_{2i}\}_{i=1}^m$. That is, the output sequence Z equals either the even or odd indexed items in the sorted order, with both outcomes equally probable.

Consider an item $y \in \mathcal{U}$ and recall that $R(y;X) = |\{x \in X | x \le y\}|$ is the number of items $x \in X$ satisfying $x \le y$. The following is a trivial observation regarding the error of the rank estimate of y with respect to the input X of a compaction operation when using Z. We view the output Z of a compaction operation (with all items up-weighted by a factor of 2) as an approximation to the input X; for any y, its weighted rank in Z should be close to its rank in X. Observation 3 below states that this approximation incurs *zero error* on items that have an even rank in X. Moreover, for items y that have an odd rank in X, the error for $y \in \mathcal{U}$ introduced by the compaction operation is $+1$ or -1 with equal probability.

OBSERVATION 3. *A universe item $y \in \mathcal{U}$ is said to be even (odd) w.r.t a compaction operation if $R(y;X)$ is even (odd), where X is the input sequence to the operation. If y is even w.r.t the compaction, then $R(y;X) - 2R(y;Z) = 0$. Otherwise $R(y;X) - 2R(y;Z)$ is a variable taking a value from $\{-1, 1\}$ uniformly at random.*

The observation that items of even rank (and in particular items of rank zero) suffer no error from a compaction operation plays an especially important role in the error analysis of our full sketch.

Full Description of the Relative-Compactor Object. The complete description of the relative-compactor object is given in Algorithm 1. The high-level idea is as follows. The relative-compactor maintains a buffer of size $B = 2 \cdot k \cdot \lceil \log_2(n/k) \rceil$ where k is an even integer parameter controlling the error and n is the upper bound on the stream length. (For now, we assume that such an upper bound is available; we remove this assumption in Section 5.) The incoming items are stored in the buffer until it is full. At this point, we perform a compaction operation, as described above.

The input to the compaction operation is not all items in the buffer, but rather the largest L items in the buffer for a parameter $L \le$
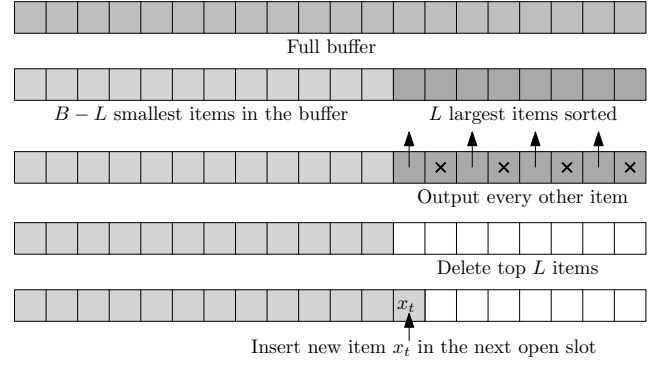
Figure 1: Illustration of the execution of a relative-compactor when inserting a new item x_t into a buffer that is full at time t. See Lines 5-14 of Algorithm 1.

$B/2$ such that L is even. These L largest items are then removed from the buffer, and the output of the compaction operation is sent to the output stream of the buffer. This intuitively lets low-ranked items stay in the buffer longer than high-ranked ones. Indeed, by design the lowest-ranked half of items in the buffer are *never* removed. We show later that this facilitates the multiplicative error guarantee.

The crucial part in the design of Algorithm 1 is to select the parameter L in a right way, as L controls the number of items compacted each time the buffer is full. If we were to set $L = B/2$ for all compaction operations, then analyzing the worst-case behavior reveals that we need $k \approx 1/\varepsilon^2$, resulting in a sketch with a quadratic dependency on $1/\varepsilon$. To achieve the linear dependency on $1/\varepsilon$, we choose the parameter L via a *derandomized* exponential distribution subject to the constraint that $L \le B/2$.[5]

In more detail, one can think of Algorithm 1 as choosing L as follows. During each compaction operation, the second half of the buffer (with $B/2$ largest items) is split into $\lceil \log_2(n/k) \rceil$ sections, each of size k and numbered from the right so that the first section contains the k largest items, the second one next k largest items, and so on; see Figure 2. The idea is that the first section is involved in every compaction (i.e., we always have $L \ge k$), the second section in every other compaction (i.e., $L \ge 2k$ every other time), the third section in every fourth compaction, and so on. This can be described concisely as follows: Let C be the number of compactions performed so far. During the next (i.e., the $C+1$-st) compaction of the relative-compactor, we set $L_C = (z(C) + 1) \cdot k$, where $z(C)$ is the number of trailing ones in the binary representation of C. We call the variable C the *state* of this "compaction schedule" (i.e., a particular way of choosing L). See Lines 6-7 of Algorithm 1, where we also define $S_C = B - L_C + 1$ as the first index in the compacted part of the buffer.

Observe that $L_C \le B/2$ always holds in Algorithm 1. Indeed, there are at most n/k compaction operations (as each discards at least k items), so the binary representation of C never has more

---

[5] A prior version of this manuscript used an actual exponential distribution; see https://arxiv.org/abs/2004.01668v1. The algorithm presented here uses randomness only to select which items to place in the output stream, not how many items to compact. This leads to a cleaner analysis and isolates the one component of the algorithm for which randomness is essential.

		7	6	5	4	3	2	1

$B/2$ slots (never compacted) $\qquad\qquad$ $\lceil \log_2(n/k) \rceil = 7$ sections with k slots each

Figure 2: Illustration of a relative-compactor and its sections, together with the indexes of the sections.

Algorithm 1 Relative-Compactor

Input: Parameters $k \in 2\mathbb{N}^+$ and $n \in \mathbb{N}^+$, and a stream of items $x_1, x_2, \ldots$ of length at most n

1: Set $B = 2 \cdot k \cdot \lceil \log_2(n/k) \rceil$
2: Initialize an empty buffer $\mathcal{B}$ of size B, indexed from 1
3: Set $C = 0$ $\qquad$ ▷ State of the compaction schedule
4: **for** $t = 1 \ldots$ **do**
5: $\quad$ **if** $\mathcal{B}$ is full **then** $\qquad$ ▷ Compaction operation
6: $\quad\quad$ Compute $z(C) =$ the number of trailing ones in the binary representation of C
7: $\quad\quad$ Set $L_C = (z(C) + 1) \cdot k$ and $S_C = B - L_C + 1$
8: $\quad\quad$ Pivot $\mathcal{B}$ s.t. the largest L_C items occupy $\mathcal{B}[S_C : B]$
9: $\qquad\qquad\qquad$ ▷ $\mathcal{B}[S_C : B]$ are the last L_C slots of $\mathcal{B}$
10: $\quad\quad$ Sort $\mathcal{B}[S_C : B]$ in non-descending order
11: $\quad\quad$ Output either even or odd indexed items in the range $\mathcal{B}[S_C : B]$ with equal probability
12: $\quad\quad$ Mark slots $\mathcal{B}[S_C : B]$ in the buffer as clear
13: $\quad\quad$ Increase C by 1
14: $\quad$ Store x_t to the next available slot in the buffer $\mathcal{B}$.

---

than $\lceil \log_2(n/k) \rceil$ bits, not even after the last compaction. Thus, $z(C)$, the number of trailing ones in the binary representation of C, is always less than $\lceil \log_2(n/k) \rceil$ and hence, $L_C \leq \lceil \log_2(n/k) \rceil \cdot k = B/2$. It also follows that there is at most one compaction operation that compacts all $\lceil \log_2(n/k) \rceil$ sections at once. Our deterministic compaction schedule has the following crucial property:

FACT 4. *Between any two compaction operations that involve exactly j sections (i.e., both have $L = j \cdot k$), there is at least one compaction operation that involves more than j sections.*

PROOF. Let $C < C'$ denote the states of the compaction schedule in two steps $t < t'$ with a compaction operation involving exactly j sections. Then we can express the binary representations of C and C' as $(\mathbf{x}, 0, \mathbf{1}^{j-1})$ and $(\mathbf{x'}, 0, \mathbf{1}^{j-1})$, respectively, where $\mathbf{1}^{j-1}$ denotes the all-1s vector of length $j - 1$, and $\mathbf{x}$ and $\mathbf{x'}$ are respectively the binary representations of two numbers y and z with $y < z$. Consider the binary vector $(\mathbf{x}, \mathbf{1}^j)$. This is the binary representation of a number $\hat{C} \in (C, C')$ with strictly more trailing ones than the binary representations of C and C'. The claim follows as there must be a step $\hat{t} \in (t, t')$ when the state equals $\hat{C}$ and a compaction operation is performed. $\qquad \square$

2.2 The Full Sketch

Following prior work [1, 12, 14], the full sketch uses a sequence of relative-compactors. At the very start of the stream, it consists of a single relative-compactor (at level 0) and opens a new one (at level 1) once items are fed to the output stream of the first relative-compactor (i.e., after the first compaction operation, which occurs on the first stream update during which the buffer is full).

In general, when the newest relative-compactor is at level h, the first time the buffer at level h performs a compaction operation (feeding items into its output stream for the first time), we open a new relative-compactor at level $h + 1$ and feed it these items. Algorithm 2 describes the logic of this sketch. To answer rank queries, we use the items in the buffers of the relative-compactors as a weighted coreset. That is, the union of these items is a weighted set C of items, where the weight of items in relative-compactor at level h is 2^h (h starts from 0), and the approximate rank of y is the sum of weights of items in C smaller than or equal to y.

The construction of layered exponentially-weighted compactors and the subsequent rank estimation is virtually identical to that explained in prior works [1, 12, 14]. Our essential departure from prior work is in the definition of the compaction operation, not in how compactors are plumbed together to form a complete sketch.

Algorithm 2 Relative-Error Quantiles sketch

Input: Parameters $k \in 2\mathbb{N}^+$ and $n \in \mathbb{N}^+$, and a stream of items $x_1, x_2, \ldots$ of length at most n
Output: A sketch answering rank queries

1: Let RelCompactors be a list of relative-compactors
2: Set $H = 0$, initialize relative-compactor at RelCompactors[0], with parameters k and n
3: **for** $t = 1 \ldots$ **do**
4: $\quad$ INSERT(x_t, 0)
5: **function** INSERT(x,h)
6: $\quad$ **if** $H < h$ **then**
7: $\quad\quad$ Set $H = h$
8: $\quad\quad$ Initialize relative-compactor at RelCompactors[h], with parameters k and n
9: $\quad$ Insert item x into RelCompactors[h]
10: $\quad$ **for** z in output stream of RelCompactors[h] **do**
11: $\quad\quad$ INSERT(z, $h + 1$)
12: **function** ESTIMATE-RANK(y)
13: $\quad$ Set $\hat{R}(y) = 0$
14: $\quad$ **for** $h = 0$ to H **do**
15: $\quad\quad$ **for** each item $y' \leq y$ stored in RelCompactors[h] **do**
16: $\quad\quad\quad$ Increment $\hat{R}(y)$ by 2^h
$\quad$ **return** $\hat{R}(y)$

2.3 Merge Operation

We describe a merge operation that takes as input two sketches S' and S'' which have processed two separate streams σ' and σ'', and that outputs a sketch S that summarizes the concatenated stream $\sigma = \sigma' \circ \sigma''$ (the order of σ' and σ'' does not matter here). For simplicity, we assume w.l.o.g. that sketch S' has at least as many levels as sketch S''. Then, the resulting sketch S inherits parameters k and n from sketch S'. We further assume that both S' and S''

have the same value of k and that n is still an upper bound on the combined input size. In the full version of our paper[2], we show how to remove the latter assumption about the knowledge of n and provide a tight analysis of the sketch created by an arbitrary sequence of merge operations without any advance knowledge about the total input size, thus proving Theorem 1.

The basic idea of the merge operation is straightforward: At each level, concatenate the buffers and if that causes the capacity of the compactor to be exceeded, perform the compaction operation, as in Algorithm 1. However, there is crucial subtlety: We need to combine the states C of the compaction schedule at each level in a manner that ensures that relative-error guarantees are satisfied for the merged sketch. Consider a level h and let C' and C'' be the states of the compaction schedule at level h in S' and S'', respectively. The new state C at level h will be the bitwise OR of C' and C''. The purpose of using bitwise OR is to make an extension of our charging scheme from Section 3 work in the general setting with merge operations. Note that while in the streaming setting, the state corresponds to the number of compaction operations already performed, after a merge operation this may not hold anymore. Still, if the state is zero, this indicates that the buffer has not yet been subject to any compactions. Algorithm 3 provides a pseudocode of the merge operation, where we use $S.H$ for the index of the highest level of sketch S and similarly, $S.k$ and $S.n$ for the parameters k and n of S, respectively.

Algorithm 3 Merge operation

Input: Sketches S' and S'' to be merged such that $S'.H \geq S''.H$
Output: A sketch answering rank queries for the combined inputs of S' and S''
1: **for** $h = 0, \ldots, S''.H$ **do** ▷ Merge S'' into S'
2:    $S'.\mathrm{RelCompactors}[h].C = S'.\mathrm{RelCompactors}[h].C$ OR $S''.\mathrm{RelCompactors}[h].C$
3:    Insert all items in $S''.\mathrm{RelCompactors}[h]$ into $S'.\mathrm{RelCompactors}[h]$
4: **for** $h = 0, \ldots, S'.H$ **do**
5:    **if** buffer $S'.\mathrm{RelCompactors}[h]$ exceeds its capacity **then**
6:       Perform compaction operation as in lines 6-13 of Algorithm 1 and insert output items into $S'.\mathrm{RelCompactors}[h+1]$
7: **return** S'

2.4 Informal Outline of the Analysis

To analyze the error of the full sketch, we focus on the error in the estimated rank of an arbitrary item $y \in \mathcal{U}$. For clarity in this informal overview, we consider the failure probability δ to be constant, and we assume that $\varepsilon^{-1} > \sqrt{\log_2(\varepsilon n)}$, or equivalently, $n < \varepsilon^{-1} \cdot 2^{\varepsilon^{-2}}$ (this assumption is loosened in the formal analysis in Sections 3 and 4). Recall that in our algorithm, all buffers have size $B = \Theta(k \log(n/k))$; we ultimately will set $k = \Theta\left(\varepsilon^{-1}/\sqrt{\log(\varepsilon n)}\right)$, in which case $B = O\left(\varepsilon^{-1}\sqrt{\log(\varepsilon n)}\right)$.

Let $R(y)$ be the rank of item y in the input stream, and $\mathrm{Err}(y) = \hat{R}(y) - R(y)$ the error of the estimated rank for y. Our analysis of $\mathrm{Err}(y)$ relies on just two properties.

(1) The level-h compactor only does at most $R(y)/(k \cdot 2^h)$ compactions that affect the error of y (up to a constant factor). Roughly speaking, this holds by the following reasoning. First, recall from Observation 3 that y needs to be odd w.r.t any compaction affecting the error of y, which implies that at least one item $x \leq y$ must be removed during that compaction. We show that as we move up one level at a time, y's rank with respect to the input stream fed to that level falls by about half (this is formally established in Lemma 9). This is the source of the 2^h factor in the denominator. Second, we show that each compaction operation that affects y can be "attributed" to k items smaller than or equal to y inserted into the buffer, which relies on using our particular compaction schedule (see Lemma 5). This is the source of the k factor in the denominator.

(2) Let H_y be the smallest positive integer such that $2^{H_y} \gtrsim R(y)/B$ (the approximate inequality $\gtrsim$ hides a universal constant). Then no compactions occurring at levels above H_y affect y, because y's rank relative to the input stream of any such buffer is less than $B/2$ and no relative-compactor ever compacts the lowest-ranked $B/2$ items that it stores.

   Again, this holds because as we move up one level at a time, y's rank w.r.t each level falls by about half (see Lemma 9).

Together, this means that the variance of the estimate for y is at most (up to constant factors):

$$\sum_{h=1}^{H_y} \frac{R(y)}{k \cdot 2^h} \cdot 2^{2h} = \sum_{h=1}^{H_y} \frac{R(y)}{k} \cdot 2^h, \tag{1}$$

where in the LHS, $R(y)/(k2^h)$ bounds the number of level-h compaction operations affecting the error (this exploits Property 1 above), and 2^{2h} is the variance contributed by each such compaction, due to Observation 3 and because items processed by relative-compactor at level h each represent 2^h items in the original stream.

The RHS of Equation (1) is dominated by the term for $h = H_y$, and the term for that value of h is at most (up to constant factors)

$$\frac{R(y)}{k} \cdot 2^{H_y} \lesssim \frac{R(y)}{k} \cdot \frac{R(y)}{B} = \frac{R(y)^2}{k \cdot B} \simeq \frac{R(y)^2 \cdot \log(\varepsilon n)}{B^2}. \tag{2}$$

The first inequality in Equation (2) exploits Property 2 above, while the last equality exploits the fact that $B = O(k \cdot \log(\varepsilon n))$.[6] We obtain the desired accuracy guarantees so long as this variance is at most $\varepsilon^2 R(y)^2$, as this will imply that the standard deviation is at most $\varepsilon R(y)$. This hoped-for variance bound holds so long as $B \gtrsim \varepsilon^{-1} \cdot \sqrt{\log_2(\varepsilon n)}$, or equivalently $k \gtrsim \varepsilon^{-1}/\sqrt{\log_2(\varepsilon n)}$.

---

[6]In the derivations within Equation (2), there is a couple of important subtleties. The first is that when we replace 2^{H_y} with $\Theta(R(y)/B)$, that substitution is only valid if $R(y)/B \geq \Omega(1)$. However, we can assume w.l.o.g. that $R(y) \geq B/2$, as otherwise the algorithm will make no error on y by virtue of storing the lowest-ranked $B/2$ items deterministically. The second subtlety is that the algorithm is only well-defined if $k \geq 2$, so when we replace k with $\Theta(B/\log(\varepsilon n))$, that is a valid substitution only if $B \geq \Omega(\log(\varepsilon n))$, which holds by the assumption that $\varepsilon^{-1} > \sqrt{\log_2(\varepsilon n)}$.

2.5 Roadmap for the Formal Analysis in the Streaming Setting

Section 3 establishes the necessary properties of a single relative-compactor (Algorithm 1), namely that, roughly speaking, each compaction operation that affects a designated item y can be charged to k items smaller than or equal to y added to the buffer. Section 4 then analyzes the full sketch (Algorithm 2), completing the proof of our result in the streaming setting when a polynomial upper bound on n is known in advance. Finally, we remove the assumption of having such an upper bound on n in Section 5.

3 ANALYSIS OF THE RELATIVE-COMPACTOR IN THE STREAMING SETTING

To analyze our algorithm, we keep track of the error associated with an arbitrary fixed item y. Throughout this section, we restrict our attention to any single relative-compactor at level h (Algorithm 1) maintained by our sketching algorithm (Algorithm 2), and we use "time t" to refer to the t'th insertion operation to this particular relative-compactor.

We analyze the error introduced by the relative-compactor for an item y. Specifically, at time t, let $X^t = \{x_1, \ldots, x_t\}$ be the input stream to the relative-compactor, Z^t be the output stream, and $\mathcal{B}^t$ be the items in the buffer after inserting item x_t. The error for the relative-compactor at time t with respect to item y is defined as

$$\text{Err}_h^t(y) = \text{R}(y; X^t) - 2\,\text{R}(y; Z^t) - \text{R}(y; \mathcal{B}^t). \tag{3}$$

Conceptually, $\text{Err}_h^t(y)$ tracks the difference between y's rank in the input stream X^t at time t versus its rank as estimated by the combination of the output stream and the remaining items in the buffer at time t (output items are upweighted by a factor of 2 while items remaining in the buffer are not). The overall error of the relative-compactor is $\text{Err}_h^n(y)$, where n is the length of its input stream. To bound $\text{Err}_h^n(y)$, we keep track of the error associated with y over time, and define the increment or decrement of it as

$$\Delta_h^t(y) = \text{Err}_h^t(y) - \text{Err}_h^{t-1}(y),$$

where $\text{Err}_h^0(y) = 0$.

Clearly, if the algorithm performs no compaction operation in a time step t, then $\Delta_h^t(y) = 0$. (Recall that a compaction is an execution of lines 6-13 of Algorithm 1.) Let us consider what happens in a step t in which a compaction operation occurs. Recall from Observation 3 that if y is even with respect to the compaction, then y suffers no error, meaning that $\Delta_h^t(y) = 0$. Otherwise, $\Delta_h^t(y)$ is uniform in $\{-1, 1\}$.

Our aim is to bound the number of steps t with $\Delta_h^t(y) \neq 0$, equal to $\sum_{t=1}^n |\Delta_h^t(y)|$, and use this in turn to help us bound $\text{Err}_h^n(y)$. We call a step t with $\Delta_h^t(y) \neq 0$ *important*. Likewise, call an item x with $x \leq y$ *important*. Let $\text{R}_h(y)$ be the rank of y in the input stream to level h; so there are $\text{R}_h(y)$ important items inserted to the buffer at level h (in the notation above, we have $\text{R}_h(y) = \text{R}(y; X^n)$). Recall that k denotes the parameter in Algorithm 1 controlling the size of the buffer of each relative-compactor and that B denotes the buffer's capacity.

Our main analytic result regarding relative-compactors is that there are at most $\text{R}_h(y)/k$ important steps. Its proof explains the intuition behind our compaction schedule, i.e., why we set L as described in Algorithm 1.

LEMMA 5. *Consider the relative-compactor at level h, fed an input stream of length at most n. For any fixed item $y \in \mathcal{U}$ with rank $\text{R}_h(y)$ in the input stream to level h, there are at most $\text{R}_h(y)/k$ important steps. In particular,*

$$\sum_{t=1}^n |\Delta_h^t(y)| \leq \frac{\text{R}_h(y)}{k} \quad \text{and} \quad \left|\text{Err}_h^n(y)\right| \leq \frac{\text{R}_h(y)}{k}.$$

PROOF. We focus on steps t in which the algorithm performs a level-h compaction operation (possibly not important), and call a step t a j-step for $j \geq 1$ if the compaction operation in step t (if any) involves exactly j sections (i.e., $L_C = j \cdot k$ in line 7 of Algorithm 1). Recall from Section 2.1 that sections are numbered from the right, so that the first section contains the k largest items in the buffer, the second section contains the next k largest items, and so on. Note that we think of the buffer as being sorted all the time.

For any $j \geq 1$, let s_j be the number of important j-steps. Further, let $\text{R}_{h,j}(y)$ be the number of important items that are either removed from the j-th section during a compaction, or remain in the j-th section at the end of execution, i.e., after the relative-compactor has processed its entire input stream. We also define $\text{R}_{h,j}(y)$ for $j = \lceil \log_2(n/k) \rceil + 1$. In this case, we define the j-th section to be the last k slots in the first half of the buffer (which contains $B/2$ smallest items); this special section is never involved in any compaction.

Observe that $\sum_{j \geq 1} s_j$ is the number of important steps and that $\sum_{j \geq 1} \text{R}_{h,j}(y) \leq \text{R}_h(y)$. We will show

$$s_j \cdot k \leq \text{R}_{h,j+1}(y). \tag{4}$$

Intuitively, our aim is to "charge" each important j-step to k important items that are either removed from section $j + 1$, or remain in section $j + 1$ at the end of execution, so that each such item is charged at most once.

Equation 4 implies the lemma as the number of important steps is

$$\sum_{t=1}^n |\Delta^t(y)| = \sum_{j=1}^{\lceil \log_2(n/k) \rceil} s_j \leq \sum_{j=1}^{\lceil \log_2(n/k) \rceil} \frac{\text{R}_{h,j+1}(y)}{k} \leq \frac{\text{R}_h(y)}{k}.$$

To show the lower bound on $\text{R}_{h,j+1}(y)$ in (4), consider an important j-step t. Since the algorithm compacts exactly j sections and $\Delta_h^t(y) \neq 0$, there is at least one important item in section j by Observation 3. As section $j + 1$ contains smaller-ranked (or equal-ranked) items than section j, section $j + 1$ contains important items only. We have two cases for charging the important j-step t:

Case A: There is a compaction operation after step t that involves at least $j+1$ buffer sections, i.e., a j'-step for $j' \geq j+1$. Let t' be the first such step. Note that just before the compaction in step t', the $(j+1)$-st section contains important items only as it contains important items only immediately after step t. We charge the important step t to the k important items that are in the $(j + 1)$-st section just before step t'. Thus, all of these charged items are removed from level h in step t'.

Case B: Otherwise, there is no compaction operation after step t that involves at least $j + 1$ buffer sections. Then, we charge step t

to the k important items that are in the $(j + 1)$-st section at the end of execution.

It remains to observe that each important item x accounted for in $R_{h,j+1}(y)$ is charged at most once. (Note that different compactions may be charged to the items which are consumed during the same later compaction, but our charging will ensure that these are assigned to different sections. For example, consider a sequence of three important compactions that compacts 2 sections, then 1 section, then 3. The first compaction will be charged to section 3 of the last compaction, and the second compaction is charged to section 2 of the last compaction.)

Formally, suppose that x is removed from section $j + 1$ during some compaction operation in a step t'. Item x may only be charged by some number of important j-steps before step t' (satisfying the condition of Case A). To show there is at most one such important step, we use the crucial property of our compaction schedule (Fact 4) that between every two compaction operations involving exactly j sections, there is at least one compaction that involves more than j sections. Since any important j-step is charged to the first subsequent compaction that involves more than j sections, item x is charged at most once.

Otherwise, x remains in section $j + 1$ of the level-h buffer at the end of processing. The proof in this case is similar to the previous case. Item x may only be charged by some number of important j-steps (that fall into Case B) such that there are no subsequent compaction operations involving at least $j + 1$ buffer sections. There is at most one such important step by Fact 4. This shows (4), which implies the lemma as noted above. □

4 ANALYSIS OF THE FULL SKETCH IN THE STREAMING SETTING

We denote by $\mathrm{Err}_h(y)$ the error for item y at the end of the stream when comparing the input stream to the compactor of level h and its output stream and buffer. That is, letting $\mathcal{B}_h$ be the items in the buffer of the level-h relative-compactor after Algorithm 2 has processed the input stream,

$$\mathrm{Err}_h(y) = R_h(y) - 2 R_{h+1}(y) - R(y; \mathcal{B}_h). \tag{5}$$

For the analysis, we first set the value of parameter k of Algorithm 2. Namely, given (an upper bound on) the stream length n, the desired accuracy $0 < \varepsilon \leq 1$ and desired upper bound $0 < \delta \leq 0.5$ on failure probability, we let

$$k = 2 \cdot \left\lceil \frac{4}{\varepsilon} \cdot \sqrt{\frac{\ln \frac{1}{\delta}}{\log_2(\varepsilon n)}} \right\rceil. \tag{6}$$

In the rest of this section, we suppose that parameters ε and δ satisfy $\delta > 1/\exp(\varepsilon n/64)$ (note that this a very weak assumption as for $\delta \leq 1/\exp(\varepsilon n/64)$ the accuracy guarantees hold nearly deterministically and furthermore, in the full version of our paper[2], we provide an analysis not requiring such an assumption). We start by showing a lower bound on $k \cdot B$.

CLAIM 6. *If parameter k is set according to Equation (6) and B is set as in Algorithm 1 (line 1), then the following inequality holds:*

$$k \cdot B \geq 2^6 \cdot \frac{1}{\varepsilon^2} \cdot \ln \frac{1}{\delta}. \tag{7}$$

PROOF. We first need to relate $\log_2(n/k)$ (used to define B, see Line 1 of Algorithm 1) and $\log_2(\varepsilon n)$ (that appears in the definition of k, see Equation (6)). Using the assumption $\delta > 1/\exp(\varepsilon n/64)$, we have $k \leq 8\varepsilon^{-1} \cdot \sqrt{\ln(1/\delta)} \leq 8\varepsilon^{-1} \cdot \sqrt{\varepsilon n/64} = \varepsilon^{-1} \cdot \sqrt{\varepsilon n}$, which gives us

$$\log_2\left(\frac{n}{k}\right) \geq \log_2\left(\frac{\varepsilon n}{\sqrt{\varepsilon n}}\right) = \frac{\log_2(\varepsilon n)}{2}.$$

Using this and the definition of k, we bound $k \cdot B$ as follows:

$$k \cdot B = 2 \cdot k^2 \cdot \left\lceil \log_2 \frac{n}{k} \right\rceil \geq 2 \cdot 2^6 \cdot \frac{1}{\varepsilon^2} \cdot \frac{\ln \frac{1}{\delta}}{\log_2(\varepsilon n)} \cdot \frac{\log_2(\varepsilon n)}{2} = 2^6 \cdot \frac{1}{\varepsilon^2} \cdot \ln \frac{1}{\delta}.$$
□

We now provide bounds on the rank of y on each level, starting with a simple one that will be useful for bounding the maximum level h with $R_h(y) > 0$.

OBSERVATION 7. $R_{h+1}(y) \leq \max\{0, R_h(y) - B/2\}$ *for any $h \geq 0$.*

PROOF. Since the lowest-ranked $B/2$ items in the input stream to the level-h relative-compactor are stored in the buffer $\mathcal{B}_h$ and never given to the output stream of the relative-compactor, it follows immediately that $R_{h+1}(y) \leq \max\{0, R_h(y) - B/2\}$. □

Next, we prove that $R_h(y)$ roughly halves with every level. This is easy to see in expectation and we show that it is true with high probability up to a certain crucial level $H(y)$. Here, we define $H(y)$ to be the minimal h for which $2^{2-h} R(y) \leq B/2$. For $h = H(y) - 1$ (assuming $H(y) > 0$), we particularly have $2^{3-H(y)} R(y) \geq B/2$, or equivalently

$$2^{H(y)} \leq 2^4 \cdot \frac{R(y)}{B}. \tag{8}$$

Below, in Lemma 9, we show that no important item (i.e., one smaller than or equal to y) can ever reach level $H(y)$. Recall that a zero-mean random variable X with variance σ^2 is sub-Gaussian if $\mathbb{E}[\exp(sX)] \leq \exp(-\frac{1}{2} \cdot s^2 \cdot \sigma^2)$ for any $s \in \mathbb{R}$; note that a (weighted) sum of independent zero-mean sub-Gaussian variables is a zero-mean sub-Gaussian random variable as well. We will use the standard (Chernoff) tail bound for sub-Gaussian variables:[7]

FACT 8. *Let X be a zero-mean sub-Gaussian variable with variance at most σ^2. Then for any $a > 0$, it holds that*

$$\Pr[X > a] \leq \exp\left(-\frac{a^2}{2\sigma^2}\right) \quad and \quad \Pr[X < -a] \leq \exp\left(-\frac{a^2}{2\sigma^2}\right).$$

LEMMA 9. *Assuming $H(y) > 0$, with probability at least $1 - \delta$ it holds that $R_h(y) \leq 2^{-h+1} R(y)$ for any $h < H(y)$.*

PROOF. We prove by induction on $0 \leq h < H(y)$ that, conditioned on $R_\ell(y) \leq 2^{-\ell+1} R(y)$ for any $\ell < h$, with probability at least $1 - \delta \cdot 2^{h-H(y)}$ it holds that $R_h(y) \leq 2^{-h+1} R(y)$. Taking the union bound over all $0 \leq h < H(y)$ implies the claim. As $R_0(y) = R(y)$, the base case follows immediately.

Next, consider $h > 0$ and condition on $R_\ell(y) \leq 2^{-\ell+1} R(y)$ for any $\ell < h$. Observe that any compaction operation at any level ℓ that involves a important items inserts $\frac{1}{2}a$ such items to the input stream at level $\ell + 1$ in expectation, no matter whether a is odd

---

[7]See, for example, Lemma 1.3 of https://ocw.mit.edu/courses/mathematics/18-s997-high-dimensional-statistics-spring-2015/lecture-notes/MIT18_S997S15_CourseNotes.pdf.

or even. Indeed, if a is odd, then the number of important items promoted is $\frac{1}{2}(a + X)$, where X is a zero-mean random variable uniform on $\{-1, 1\}$. For an even a, the number of important items that are promoted is $\frac{1}{2}a$ with probability 1.

Thus, random variable $R_\ell(y)$ for any level $\ell > 0$ is generated by the following random process: To get $R_\ell(y)$, start with $R_{\ell-1}(y)$ important items and remove those stored in the level-$(\ell-1)$ relative-compactor $\mathcal{B}_{\ell-1}$ at the end of execution; there are $R(y; \mathcal{B}_{\ell-1}) \leq B$ important items in $\mathcal{B}_{\ell-1}$. Then, as described above, each compaction operation at level $\ell-1$ involving $a > 0$ important items promotes to level ℓ either $\frac{1}{2}a$ important items if a is even, or $\frac{1}{2}(a+X)$ important items if a is odd. In total, $R_{\ell-1}(y) - R(y; \mathcal{B}_{\ell-1})$ important items are involved in compaction operations at level $\ell - 1$. Summarizing, we have

$$R_\ell(y) = \frac{1}{2} \cdot (R_{\ell-1}(y) - R(y; \mathcal{B}_{\ell-1}) + \text{Binomial}(m_{\ell-1})), \quad (9)$$

where $\text{Binomial}(n)$ represents the sum of n zero-mean i.i.d. random variables uniform on $\{-1, 1\}$ and $m_{\ell-1}$ is the number of important compaction operations at level $\ell - 1$ (which are those involving an odd number of important items).

To simplify (9), consider the following sequence of random variables $Y_0, \ldots, Y_h$: Start with $Y_0 = R(y)$ and for $0 < \ell < h$ let

$$Y_\ell = \frac{1}{2} \cdot (Y_{\ell-1} + \text{Binomial}(m_{\ell-1})). \quad (10)$$

Note that $\mathbb{E}[Y_\ell] = 2^{-\ell} R(y)$. Since variables Y_ℓ differ from $R_\ell(y)$ only by not subtracting $R(y; \mathcal{B}_{\ell-1})$ at every level $\ell > 0$, variable Y_h stochastically dominates variable $R_h(y)$, so in particular,

$$\Pr[R_h(y) > 2^{-h+1} R(y)] \leq \Pr[Y_h > 2^{-h+1} R(y)], \quad (11)$$

which implies that it is sufficient to bound $\Pr[Y_h > 2^{-h+1} R(y)]$. Unrolling the definition of Y_h in (10), we obtain

$$Y_h = 2^{-h} \cdot R(y) + \sum_{\ell=0}^{h-1} 2^{-h+\ell} \cdot \text{Binomial}(m_\ell). \quad (12)$$

Observe that Y_h equals a fixed amount $(2^{-h} \cdot R(y))$ plus a zero-mean sub-Gaussian variable

$$Z_h = \sum_{\ell=0}^{h-1} 2^{-h+\ell} \cdot \text{Binomial}(m_\ell), \quad (13)$$

since $\text{Binomial}(n)$ is a sum of n independent zero-mean sub-Gaussian variables (with variance 1).

To bound the variance of Z_h, first note that for any $\ell < h$, we have $m_\ell \leq R_\ell(y)/k \leq 2^{-\ell+1} R(y)/k$ by Lemma 5 and by conditioning on $R_\ell(y) \leq 2^{-\ell+1} R(y)$. As $\text{Var}[\text{Binomial}(n)] = n$, the variance of Z_h is

$$\text{Var}[Z_h] \leq \sum_{\ell=0}^{h-1} 2^{-2h+2\ell} \cdot m_\ell \leq \sum_{\ell=0}^{h-1} 2^{-2h+2\ell} \cdot \frac{2^{-\ell+1} R(y)}{k}$$
$$= \sum_{\ell=0}^{h-1} \frac{2^{-2h+\ell+1} R(y)}{k} \leq \frac{2^{-h+1} \cdot R(y)}{k}.$$

Note that $\Pr[Y_h > 2^{-h+1} R(y)] = \Pr[Z_h > 2^{-h} R(y)]$. To bound the latter probability, we apply the tail bound for sub-Gaussian

variables (Fact 8) to get

$$\Pr[Z_h > 2^{-h} R(y)] < \exp\left(-\frac{2^{-2h} \cdot R(y)^2}{2 \cdot (2^{-h+1} \cdot R(y)/k)}\right)$$
$$= \exp\left(-2^{-h-2} \cdot R(y) \cdot k\right)$$
$$= \exp\left(-2^{-h+H(y)-6} \cdot 2^{4-H(y)} R(y) \cdot k\right)$$
$$\leq \exp\left(-2^{-h+H(y)-6} \cdot B \cdot k\right)$$
$$\leq \exp\left(-2^{-h+H(y)-6} \cdot 2^6 \cdot \frac{1}{\varepsilon^2} \cdot \ln\frac{1}{\delta}\right)$$
$$\leq \exp\left(-2^{-h+H(y)} \cdot \ln\frac{1}{\delta}\right)$$
$$= \delta^{2^{H(y)-h}} \leq \delta \cdot 2^{-H(y)+h},$$

where the second inequality uses $2^{4-H(y)} R(y) \geq B$ (by the definition of $H(y)$, cf. Equation (8)), the third inequality follows from Claim 6, the fourth inequality uses $\varepsilon \leq 1$, and the last inequality uses $\delta \leq 0.5$. As explained above, this concludes the proof. $\quad\square$

In what follows, we condition on the bound on $R_h(y)$ in Lemma 9 for any $h < H(y)$.

LEMMA 10. *Conditioned on the bound on $R_{H(y)-1}(y)$ in Lemma 9, it holds that $R_{H(y)}(y) = 0$.*

PROOF. According to Lemma 9 and the definition of $H(y)$ as the minimal h for which $2^{2-h} R(y) \leq B/2$,

$$R_{H(y)-1}(y) \leq 2^{2-H(y)} R(y) \leq \frac{1}{2}B.$$

Invoking Observation 7, we get $R_{H(y)}(y) \leq \max\{0, R_{H(y)-1}(y) - B/2\} = 0$. $\quad\square$

We are now ready to bound the overall error of the sketch for item y, i.e., $\text{Err}(y) = \hat{R}(y) - R(y)$ where $\hat{R}(y)$ is the estimated rank of y. It is easy to see that

$$\text{Err}(y) = \sum_{h=0}^{H} 2^h \text{Err}_h(y),$$

where H is the highest level with a relative-compactor (that never produces any output). To bound this error we refine the guarantee of Lemma 5. Notice that for any particular relative-compactor, the bound $\sum_{t=1}^{n} |\Delta_h^t(y)|$ referred to in Lemma 5 applied to a level h is a potentially crude upper bound on $\text{Err}_h(y) = \sum_{t=1}^{n} \Delta_h^t(y)$: Each non-zero term $\Delta_h^t(y)$ is positive or negative with equal probability, so the terms are likely to involve a large amount of cancellation. To take advantage of this, we bound the variance of $\text{Err}(y)$.

LEMMA 11. *Conditioned on the bound on $R_h(y)$ in Lemma 9 for any $h < H(y)$, $\text{Err}(y)$ is a zero-mean sub-Gaussian random variable with $\text{Var}[\text{Err}(y)] \leq 2^5 \cdot R(y)^2/(k \cdot B)$.*

PROOF. Consider the relative-compactor at any level h. By Lemma 5, $\text{Err}_h(y)$ is a sum of at most $R_h(y)/k$ random variables, i.i.d. uniform in $\{-1, 1\}$. In particular, $\text{Err}_h(y)$ is a zero-mean sub-Gaussian random variable with $\text{Var}[\text{Err}_h(y)] \leq R_h(y)/k$. Thus, $\text{Err}(y)$ is a sum of independent zero-mean sub-Gaussian random

variables, and as such is itself a zero-mean sub-Gaussian random variable.

It remains to bound the variance of $\mathrm{Err}(y)$, for which we first bound $\mathrm{Var}[\mathrm{Err}_h(y)]$ for each h. If $\mathrm{R}_h(y) = 0$, then Observation 3 implies that $\mathrm{Err}_h(y) = 0$, and hence that $\mathrm{Var}[\mathrm{Err}_h(y)] = 0$. Thus, using Lemma 10, we have $\mathrm{Var}[\mathrm{Err}_h(y)] = 0$ for any $h \geq H(y)$. For $h < H(y)$, we use $\mathrm{Var}[\mathrm{Err}_h(y)] \leq \mathrm{R}_h(y)/k$ to obtain:

$$\mathrm{Var}[\mathrm{Err}(y)] = \sum_{h=0}^{H(y)-1} 2^{2h} \, \mathrm{Var}[\mathrm{Err}_h(y)] \leq \sum_{h=0}^{H(y)-1} 2^{2h} \cdot \frac{\mathrm{R}_h(y)}{k}$$

$$\leq \sum_{h=0}^{H(y)-1} 2^{h+1} \cdot \frac{\mathrm{R}(y)}{k} \leq 2^{H(y)+1} \cdot \frac{\mathrm{R}(y)}{k} \leq 2^5 \cdot \frac{\mathrm{R}(y)^2}{k \cdot B},$$

where the second inequality is due to Lemma 9 and the last inequality follows from (8). □

To show that the space bound in maintained, we also need to bound the number of relative-compactors.

OBSERVATION 12. *The number of relative-compactors ever created by the full algorithm (Algorithm 2) is at most* $\lceil \log_2(n/B) \rceil + 1$.

PROOF. Each item on level h has weight 2^h, so there are at most $n/2^h$ items inserted to the buffer at that level. Applying this observation to $h = \lceil \log_2(n/B) \rceil$, we get that on this level, there are fewer than B items inserted to the buffer, which is consequently not compacted, so the highest level has index at most $\lceil \log_2(n/B) \rceil$. The claim follows (recall that the lowest level has index 0). □

We are now ready to prove the main result of this section, namely, the accuracy guarantees in the streaming setting when the stream length is essentially known in advance.

THEOREM 13. *Assume that (a polynomial upper bound on) the stream length n is known in advance. For any parameters $0 < \delta \leq 0.5$ and $0 < \varepsilon \leq 1$ satisfying $\delta > 1/\exp(\varepsilon n/64)$, let k be set as in (6). Then, for any fixed item y, Algorithm 2 with parameters k and n computes an estimate $\hat{\mathrm{R}}(y)$ of $\mathrm{R}(y)$ with error $\mathrm{Err}(y) = \hat{\mathrm{R}}(y) - \mathrm{R}(y)$ such that $\Pr\left[|\mathrm{Err}(y)| \geq \varepsilon \mathrm{R}(y) \right] < 3\delta$. If $\varepsilon \leq 4 \cdot \sqrt{\ln(1/\delta)/\log_2(\varepsilon n)}$, then the memory used by the algorithm is $O\left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon n) \cdot \sqrt{\log(1/\delta)} \right)$; otherwise, the algorithm uses $O\left(\log^2(\varepsilon n) \right)$ memory words.*

PROOF. Note that k is an even positive integer as required by Algorithm 2. By Lemma 9, with probability at least $1 - \delta$, we have $\mathrm{R}_h(y) \leq 2^{-h+1} \mathrm{R}(y)$ for any $h < H(y)$ and we condition on this event happening.

We again apply the standard (Chernoff) tail bound for sub-Gaussian variables (Fact 8) together with Lemma 11 (for which we need the bound on $\mathrm{R}_h(y)$ for any $h < H(y)$) and obtain

$$\Pr\left[|\mathrm{Err}(y)| \geq \varepsilon \mathrm{R}(y) \right] < 2 \exp\left(-\frac{\varepsilon^2 \cdot \mathrm{R}(y)^2}{2 \cdot 2^5 \cdot \mathrm{R}(y)^2/(k \cdot B)} \right)$$

$$\leq 2 \exp\left(-\frac{\varepsilon^2 \cdot 2^6 \cdot \varepsilon^{-2} \cdot \ln \frac{1}{\delta}}{2^6} \right)$$

$$= 2 \exp\left(-\ln \frac{1}{\delta} \right) = 2\delta,$$

where we use Claim 6 in the second inequality. This concludes the calculation of the failure probability.

Regarding the memory usage, there are at most $\lceil \log_2(n/B) \rceil + 1 \leq \log_2(\varepsilon n)$ relative-compactors by Observation 12, and each requires $B = 2 \cdot k \cdot \lceil \log_2(n/k) \rceil$ memory words. Thus, the memory needed to run the algorithm is at most

$$\log_2(\varepsilon n) \cdot 2 \cdot k \cdot \left\lceil \log_2 \frac{n}{k} \right\rceil$$

$$\leq \log_2(\varepsilon n) \cdot 2 \cdot 2 \cdot \left\lceil \frac{4}{\varepsilon} \cdot \sqrt{\frac{\ln \frac{1}{\delta}}{\log_2(\varepsilon n)}} \right\rceil \cdot O\left(\log(\varepsilon n) \right), \quad (14)$$

where we use that $\lceil \log_2(n/k) \rceil \leq O(\log(\varepsilon n))$, which follows from $k \geq \varepsilon^{-1}/\sqrt{\log_2(\varepsilon n)}$. In the case $\varepsilon \leq 4 \cdot \sqrt{\ln(1/\delta)/\log_2(\varepsilon n)}$, we have $a := 4\varepsilon^{-1} \cdot \sqrt{\ln(1/\delta)/\log_2(\varepsilon n)} \geq 1$, so $\lceil a \rceil \leq 2a$ and it follows that (14) is bounded by $O\left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon n) \cdot \sqrt{\log(1/\delta)} \right)$. Otherwise, $a < 1$, thus (14) becomes at most $O\left(\log^2(\varepsilon n) \right)$. □

Update time. We now analyze the amortized update time of Algorithm 2 and show that it can be made $O(\log B) = O(\log(k) + \log \log(\varepsilon n))$, i.e., the algorithm processes n streaming updates in total time $O(n \cdot \log B)$. To see this, first observe that the time complexity is dominated, up to a constant factor, by running Algorithm 1 for the relative-compactor at level 0. Indeed, the running time can be decomposed into the operations done by Algorithm 2 itself, plus the running time of Algorithm 1 for each level of the sketch, and the former is bounded by the latter. Moreover, at level h there are at most $n/2^h$ items added to the buffer, implying that the running time of Algorithm 1 decreases exponentially with the level. At level 0, the update time is $O(1)$, except for performing compaction operations (line 6-13 of Algorithm 1). To make those faster, we maintain the buffer sorted after each insertion, which can be achieved by using an appropriate data structure in time $O(\log B)$ per update. Then the time to execute each compaction operation is linear in the number of items removed from the buffer, making it amortized constant. Hence, the amortized update time with such adjustments is $O(\log B)$.

5 HANDLING UNKNOWN STREAM LENGTHS

The algorithm of Section 2.2 and analysis in Sections 3-4 proved Theorem 13 in the streaming setting assuming that (an upper bound on) n is known, where n is the true stream length. The space usage of the algorithm grows polynomially with the logarithm of this upper bound, so if this upper bound is at most n^c for some constant $c \geq 1$, then the space usage of the algorithm will remain as stated in Theorem 13, with only the hidden constant factor changing.

In the case that such a polynomial upper bound on n is not known, we modify the algorithm slightly, and start with an initial estimate N_0 of n, such as $N_0 = O(\varepsilon^{-1})$. As soon as the stream length hits the current estimate N_i, the algorithm "closes out" the current data structure and continues to store it in "read only" mode, while initializing a new summary based on the estimated stream length of $N_{i+1} = N_i^2$.[8] This process occurs at most $\log_2 \log_2(\varepsilon n)$ many times,

---

[8]In a practical implementation, we suggest not to close out the current summary, but rather recompute the parameters k and B of every relative-compactor in the summary, according to the new estimate N_{i+1}, and continue with using the summary. The

before the guess is at least the true stream length n. At the end of the stream, the rank of any item y is estimated by summing the estimates returned by each of the at most $\log_2 \log_2(\varepsilon n)$ summaries stored by the algorithm.

To prove Theorem 13 for unknown stream lengths, we need to bound the space usage of the algorithm, and the probability of having a too large error for a fixed item y. We start with some notation. Let ℓ be the biggest index i of estimate N_i used by the algorithm; note that $\ell \leq \log_2 \log_2(\varepsilon n)$. Let σ_i denote the substream processed by the summary with the i'th guess for the stream length for $i = 0, \ldots \ell$. Let $\sigma' \circ \sigma''$ denote the concatenation of two streams σ' and σ''. Then the complete stream processed by the algorithm is $\sigma = \sigma_0 \circ \sigma_1 \circ \cdots \circ \sigma_\ell$. Let k_i and B_i be the values of parameters k and B computed for estimate N_i.

Space bound. We claim that the sizes of summaries for the substreams $\sigma_0, \sigma_1, \ldots, \sigma_\ell$ sum up to $O\left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon n) \cdot \sqrt{\log(1/\delta)}\right)$, as required. Here, we assume for simplicity that $\varepsilon \leq 4 \cdot \sqrt{\ln(1/\delta)/\log_2(\varepsilon n)}$; the other case can be handled similarly. By Theorem 13, the size of the summary for σ_i is $O\left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon N_i) \cdot \sqrt{\log(1/\delta)}\right)$. In the special case $\ell = 0$, the size of the summary for σ_0 satisfies the bound provided that $N_0 = O(\varepsilon^{-1})$. For $\ell \geq 1$, since $N_{\ell-1} < n$ and $N_\ell = N_{\ell-1}^2$, it holds that $N_\ell \leq n^2$ and thus, the size of the summary for σ_ℓ satisfies the claimed bound. As $N_{i+1} = N_i^2$, the $\log^{1.5}(\varepsilon N_i)$ factor in the size bound from Theorem 13 increases by a factor of $2^{1.5}$ when we increase i. It follows that the total space usage is dominated, up to a constant factor, by the size of the summary for σ_ℓ. □

Failure probability. We need to show that $|\operatorname{Err}(y)| = |\hat{R}(y) - R(y)| \leq \varepsilon R(y)$ with probability at least $1 - \delta$ for any fixed item y. Note that $R(y) = R(y; \sigma) = \sum_{i=0}^{\ell} R(y; \sigma_i)$.

We apply the analysis in Section 4 to all of the summaries at once. Observe that for the tail bound in the proof of Theorem 13, we need to show that $\operatorname{Err}(y)$ is a zero-mean sub-Gaussian random variable with a suitably bounded variance. Let $\operatorname{Err}^i(y)$ be the error introduced by the summary for σ_i. By Lemma 11, $\operatorname{Err}^i(y)$ is a zero-mean sub-Gaussian random variable with $\operatorname{Var}[\operatorname{Err}^i(y)] \leq 2^5 \cdot R(y; \sigma_i)^2/(k_i \cdot B_i)$. As $\operatorname{Err}(y) = \sum_i \operatorname{Err}^i(y)$ and as the summaries are created with independent randomness, variable $\operatorname{Err}(y)$ is also zero-mean sub-Gaussian and its variance is bounded by

$$\operatorname{Var}[\operatorname{Err}(y)] = \sum_{i=0}^{\ell} \operatorname{Var}[\operatorname{Err}^i(y)] \leq \sum_{i=0}^{\ell} 2^5 \cdot \frac{R(y; \sigma_i)^2}{k_i \cdot B_i} \leq \frac{\varepsilon^2 \cdot R(y)^2}{2 \cdot \ln(1/\delta)}$$

where the last inequality uses that $\sum_{i=0}^{\ell} R(y; \sigma_i)^2 \leq R(y)^2$, which follows from $R(y) = \sum_{i=0}^{\ell} R(y; \sigma_i)$, and that $k_i \cdot B_i = \Omega(\varepsilon^{-2} \cdot \ln(1/\delta))$, which holds by Claim 6. Applying the tail bound for sub-Gaussian variables similarly as in the proof of Theorem 13 concludes the proof of Theorem 13 for unknown stream lengths. □

---

analysis in full version of our paper (which applies in the more general mergeability setting) shows that the same accuracy guarantees as in Theorem 13 hold for this variant of the algorithm. Here, we choose to have one summary for each estimate of n because it is amenable to a much simpler analysis (it is not clear how to extend this simpler analysis from the streaming setting to the general mergeability setting).

6 DISCUSSION AND OPEN PROBLEMS

For constant failure probability δ, we have shown an $O(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon n))$ space upper bound for relative error quantile approximation over data streams. Our algorithm is provably more space-efficient than any deterministic comparison-based algorithm, and is within a $\tilde{O}\left(\sqrt{\log(\varepsilon n)}\right)$ factor of the known lower bound for randomized algorithms (even non-streaming algorithms, see Appendix A). Moreover, the sketch output by our algorithm is fully mergeable, with the same accuracy-space trade-off as in the streaming setting, rendering it suitable for a parallel or distributed environment. The main remaining question is to close this $\tilde{O}(\sqrt{\log(\varepsilon n)})$-factor gap.

REFERENCES

[1] Pankaj K Agarwal, Graham Cormode, Zengfeng Huang, Jeff M Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. *ACM Transactions on Database Systems (TODS)*, 38(4):26, 2013.

[2] Rakesh Agrawal and Arun Swami. A one-pass space-efficient algorithm for finding quantiles. In *Proc. 7th Intl. Conf. Management of Data (COMAD-95), Pune, India*, 1995.

[3] Arvind Arasu and Gurmeet Singh Manku. Approximate counts and quantiles over sliding windows. In *Proceedings of the 23rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '04, pages 286–296. ACM, 2004.

[4] Graham Cormode, Flip Korn, , S. Muthukrishnan, and Divesh Srivastava. Effective computation of biased quantiles over data streams. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 20–31, Washington, DC, USA, 2005. IEEE Computer Society.

[5] Graham Cormode, Flip Korn, S Muthukrishnan, and Divesh Srivastava. Space-and time-efficient deterministic algorithms for biased quantiles over data streams. In *Proceedings of the 25th ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '06, pages 263–272. ACM, 2006.

[6] Graham Cormode and Pavel Veselý. A tight lower bound for comparison-based quantile summaries. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*, PODS'20, page 81–93, New York, NY, USA, 2020. ACM.

[7] Ted Dunning and Otmar Ertl. Computing extremely accurate quantiles using t-digests. *CoRR*, abs/1902.04023, 2019.

[8] David Felber and Rafail Ostrovsky. A randomized online quantile summary in O(1/epsilon * log(1/epsilon)) words. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2015)*, volume 40 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 775–785, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

[9] Sumit Ganguly. A nearly optimal and deterministic summary structure for update data streams. *arXiv preprint cs/0701020*, 2007.

[10] Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *ACM SIGMOD Record*, volume 30, pages 58–66. ACM, 2001.

[11] Anupam Gupta and Francis X. Zane. Counting inversions in lists. In *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '03, pages 253–254, Philadelphia, PA, USA, 2003. Society for Industrial and Applied Mathematics.

[12] Zohar Karnin, Kevin Lang, and Edo Liberty. Optimal quantile approximation in streams. In *Proceedings of the 57th Annual Symposium on Foundations of Computer Science (FOCS '16)*, pages 71–78. IEEE, 2016.

[13] Ge Luo, Lu Wang, Ke Yi, and Graham Cormode. Quantiles over data streams: Experimental comparisons, new analyses, and further improvements. *The VLDB Journal*, 25(4):449–472, August 2016.

[14] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *ACM SIGMOD Record*, volume 27, pages 426–435. ACM, 1998.

[15] Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *ACM SIGMOD Record*, volume 28, pages 251–262. ACM, 1999.

[16] Charles Masson, Jee E. Rim, and Homin K. Lee. DDSketch: A fast and fully-mergeable quantile sketch with relative-error guarantees. *PVLDB*, 12(12):2195–2205, 2019.

[17] J Ian Munro and Michael S Paterson. Selection and sorting with limited storage. *Theoretical computer science*, 12(3):315–323, 1980.

[18] Ira Pohl. *A minimum storage algorithm for computing the median*. IBM TJ Watson Research Center, 1969.

[19] Viswanath Poosala, Venkatesh Ganti, and Yannis E. Ioannidis. Approximate query answering using histograms. *IEEE Data Eng. Bull.*, 22(4):5–14, 1999.

[20] Lee Rhodes, Kevin Lang, Alexander Saydakov, Edo Liberty, and Justin Thaler. DataSketches: A library of stochastic streaming algorithms. Open source software: https://datasketches.apache.org/, 2013.

[21] Nisheeth Shrivastava, Chiranjeeb Buragohain, Divyakant Agrawal, and Subhash Suri. Medians and beyond: new aggregation techniques for sensor networks. In *Proceedings of the 2nd international conference on Embedded networked sensor systems*, pages 239–249. ACM, 2004.

[22] Gil Tene. How NOT to measure latency. https://www.youtube.com/watch?v=lJ8ydIuPFeU, 2015.

[23] Qi Zhang and Wei Wang. An efficient algorithm for approximate biased quantile computation in data streams. In *Proceedings of the 16th ACM conference on Conference on information and knowledge management*, pages 1023–1026, 2007.

[24] Ying Zhang, Xuemin Lin, Jian Xu, Flip Korn, and Wei Wang. Space-efficient relative error order sketch over data streams. In *Proceedings of the 22nd International Conference on Data Engineering (ICDE'06)*, pages 51–51. IEEE, 2006.

A A LOWER BOUND FOR NON-COMPARISON BASED ALGORITHMS

Cormode and Veselý [6, Theorem 6.5] proved an $\Omega(\varepsilon^{-1} \cdot \log^2(\varepsilon n))$ lower bound on the number of items stored by any deterministic comparison-based streaming algorithm for the relative-error quantiles problem. Below, we provide a lower bound which also applies to offline, non-comparison-based randomized algorithms, but at the (necessary) cost of losing a $\log(\varepsilon n)$ factor in the resulting space bound. This result appears not to have been explicitly stated in the literature, though it follows from an argument similar to [4, Theorem 2]. We provide details in this appendix for completeness.

Theorem 14. *For any randomized algorithm that processes a data stream of items from universe $\mathcal{U}$ of size $|\mathcal{U}| \geq \Omega(\varepsilon^{-1} \cdot \log(\varepsilon n))$ and outputs a sketch that solves the all-quantiles approximation problem for multiplicative error ε with probability at least $2/3$ requires the sketch to have size $\Omega\left(\varepsilon^{-1} \cdot \log(\varepsilon n) \cdot \log(\varepsilon|\mathcal{U}|)\right)$ bits of space.*

Proof. We show that any multiplicative-error sketch for all-quantiles approximation can be used to losslessly encode an arbitrary subset S of the data universe $\mathcal{U}$ of size $|S| \geq \Omega\left(\varepsilon^{-1}\log(\varepsilon n)\right)$. This requires $\log_2 \binom{|\mathcal{U}|}{|S|} = \Theta\left(\log((|\mathcal{U}|/|S|)^{|S|})\right) = \Theta\left(|S| \log(\varepsilon|\mathcal{U}|)\right)$ bits of space. The theorem follows.

Let $\ell = 1/(8\varepsilon)$ and $k = \log_2(\varepsilon n)$; for simplicity, we assume that both ℓ and k are integers. Let S be a subset of $\mathcal{U}$ of size $s := \ell \cdot k$. We will construct a stream σ of length less than $\ell \cdot 2^k \leq n$ such that a sketch solving the all-quantiles approximation problem for σ enables reconstruction of S. To this end, let $\{y_1, \ldots, y_s\}$ denote the elements of S in increasing order. Consider the stream σ where items $y_1, \ldots, y_\ell$ each appear once, items $y_{\ell+1}, \ldots, y_{2\ell}$ appear twice, and in general items $y_{i\ell+1}, \ldots, y_{(i+1)\ell}$ appear 2^i times, for $i = 0, \ldots, k-1$. Let us refer to all universe items in the interval $[y_{i\ell+1}, y_{(i+1)\ell}]$ as "Phase i" items.

The construction of σ means that the multiplicative error ε in the estimated rank of any Phase i item is at most $2^{i+1}/8 < 2^{i-1}$. This means that for any phase $i \geq 0$ and integer $j \in [1, \ell]$, one can

identify item $y_{i\ell+j}$ by finding the smallest universe item whose estimated rank is strictly greater than $(2^i - 1) \cdot \ell + 2^i \cdot j - 2^{i-1}$. Here, $(2^i - 1) \cdot \ell$ is the number of stream updates corresponding to items in Phases $0, \ldots, i-1$, while 2^{i-1} is an upper bound on the error of the estimated rank of any Phase i item. Hence, from any sketch solving the all-quantiles approximation problem for σ one can obtain the subset S, which concludes the lower bound. $\square$

Theorem 14 is tight up to constant factors, as an optimal summary consisting of $O(\varepsilon^{-1} \cdot \log(\varepsilon n))$ items can be constructed offline. For $\ell = \varepsilon^{-1}$, this summary stores all items of rank $1, \ldots, 2\ell$ appearing in the stream and assigns them weight one, stores every other item of rank between $2\ell + 1$ and 4ℓ and assigns them weight 2, stores every fourth item of rank between $4\ell + 1$ and 8ℓ and assigns them weight 4, and so forth. This yields a weighted coreset S for the relative-error quantiles approximation, consisting of $|S| = \Theta(\ell \cdot \log(\varepsilon n))$ many items. Such a set S can be represented with $\log_2 \binom{|\mathcal{U}|}{|S|} = \Theta\left(\varepsilon^{-1} \cdot \log(\varepsilon n) \cdot \log(\varepsilon|\mathcal{U}|)\right)$ many bits.

B PROOF OF COROLLARY 1

Corollary 1 (All-Quantiles Approximation). *The error bound from Theorem 1 can be made to hold for all $y \in \mathcal{U}$ simultaneously with probability $1 - \delta$ while storing*

$$O\left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon n) \cdot \sqrt{\log\left(\frac{\log(\varepsilon n)}{\varepsilon \delta}\right)}\right)$$

stream items if $\varepsilon \leq O\left(\sqrt{\log \frac{1}{\varepsilon \delta}}/\log(\varepsilon n)\right)$ and $O\left(\log^2(\varepsilon n)\right)$ items otherwise.

Proof. Let S^* be the offline optimal summary of the stream with multiplicative error $\varepsilon/3$, i.e., a subset of items in the stream such that for any item x, there is $y \in S^*$ with $|\, R(y) - R(x)| \leq (\varepsilon/3) \cdot R(x)$. Here, y is simply the closest item to x in the total order that is an element of S^*. Observe that S^* has $O(\varepsilon^{-1} \cdot \log(\varepsilon n))$ items; see the remark below Theorem 14 in Appendix A for a construction of S^*.

Thus, if our sketch with parameter $\varepsilon' = \varepsilon/3$ is able to compute for any $y \in S^*$ a rank estimate $\hat{R}(y)$ such that $|\hat{R}(y) - R(y)| \leq (\varepsilon/3) \cdot R(y)$, then we can approximate $R(x)$ by $\hat{R}(y)$ using $y \in S^*$ with $|\, R(y) - R(x)| \leq (\varepsilon/3) \cdot R(x)$ and the multiplicative guarantee for x follows from

$$|\hat{R}(y) - R(x)| \leq |\hat{R}(y) - R(y)| + |\, R(y) - R(x)|$$
$$\leq \frac{\varepsilon}{3} \cdot R(y) + \frac{\varepsilon}{3} \cdot R(x)$$
$$\leq \left(\frac{\varepsilon}{3} \cdot (1 + \frac{\varepsilon}{3}) + \frac{\varepsilon}{3}\right) \cdot R(x)$$
$$\leq \varepsilon \cdot R(x)\,.$$

It remains to ensure that our algorithm provides a good-enough rank estimate for any $y \in S^*$. We apply Theorem 1 with error parameter $\varepsilon' = \varepsilon/3$ and with failure probability set to $\delta' = \delta/|S^*| = \Theta(\delta \cdot \varepsilon/\log(\varepsilon n))$. By the union bound, with probability at least $1 - \delta$, the resulting sketch satisfies the $(1 \pm \varepsilon/3)$-multiplicative error guarantee for any item in S^*. In this event, the previous paragraph implies that the $(1 \pm \varepsilon)$-multiplicative guarantee holds for all $x \in \mathcal{U}$. The space bound follows from Theorem 1 with ε' and δ' as above. $\square$