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
HyperCube
USA
edo@edoliberty.com

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

Pavel Veselý
University of Warwick
Coventry, UK
Pavel.Vesely@warwick.ac.uk

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.

ACM Reference Format:
Graham Cormode, Zohar Karnin, Edo Liberty, Justin Thaler, and Pavel Veselý. 2021. Relative Error Streaming Quantiles. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 26 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

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 $\mathrm{R}(y;\sigma) = |\{x_i \mid x_i \le y\}|$ be the rank of y in the stream. When σ is clear from context, we write $\mathrm{R}(y)$. The objective is to process the stream while storing a small number of items, and then use those to approximate $\mathrm{R}(y)$ for any $y \in \mathcal{U}$. A guarantee for an approximation $\hat{\mathrm{R}}(y)$ is *additive* if $|\hat{\mathrm{R}}(y) - \mathrm{R}(y)| \le \varepsilon n$, and *multiplicative* or *relative* if $|\hat{\mathrm{R}}(y) - \mathrm{R}(y)| \le \varepsilon\, \mathrm{R}(y)$.

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 $\mathrm{R}(y) \ll n$, 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 $n - \mathrm{R}(y) \ll n$, 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. Hence, highly accurate rank approximations are required for items y whose rank is very large ($n - \mathrm{R}(y) \ll n$); 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 store $\Omega(\varepsilon^{-1} \cdot \log(\varepsilon n))$ items (see [4, Theorem 2] and Appendix A).[1]

The best known algorithms achieving multiplicative error guarantees are as follows. Zhang et al. [23] 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

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[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.

algorithm of Cormode et al. [5] that stores $O(\varepsilon^{-1} \cdot \log(\varepsilon n) \cdot \log |\mathcal{U}|)$ items. However, this algorithm requires prior knowledge of the data universe $\mathcal{U}$ (since it builds a binary tree over $\mathcal{U}$), and is inapplicable when $\mathcal{U}$ is huge or even unbounded (e.g., if the data can take arbitrary real values). Finally, Zhang and Wang [22] 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 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, and from which an estimate $\hat{\mathrm{R}}(y)$ of $\mathrm{R}(y)$ 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 $|\hat{\mathrm{R}}(y) - \mathrm{R}(y)| \le \varepsilon\, \mathrm{R}(y)$. 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. Namely, the amortized update time of the algorithm is a logarithm of the space bound, that is, $O\left(\log(\varepsilon^{-1}) + \log\log(n) + \log\log(1/\delta)\right)$; 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 (see Appendix C for details).

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.[2]

[2] 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 in preparation and will be available at https://datasketches.apache.org/.

Theorem 1. *Let $0 < \delta \le 0.5$ and $0 < \varepsilon \le 1$ be parameters satisfying $\varepsilon \le 4/\sqrt[4]{2\log_2(n)}$. 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 $\hat{\mathrm{R}}(y)$ of $\mathrm{R}(y)$ such that*

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

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

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

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 $O(1/\varepsilon)$ 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 $1/\varepsilon$ 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 $\Theta(\varepsilon^{-2} \cdot \log(\varepsilon^2 n))$, which merely matches the space bound achieved in [23], and in particular has a (quadratically) suboptimal dependence on $1/\varepsilon$.

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 $\mathrm{R}(y)$ 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 only holds with high probability 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 bit-wise 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 $\log(\varepsilon n)$ increasing from 1.5 to 2. Formally, we prove the following theorem in Appendix D, where we also show that it directly implies a deterministic space bound of $O(\varepsilon^{-1} \cdot \log^3(\varepsilon n))$, matching the state-of-the-art result in [22]. 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.

Theorem 2. *For any parameters $0 < \delta \le 0.5$ and $0 < \varepsilon \le 1$, there is a randomized, comparison-based, one-pass streaming algorithm that computes a sketch consisting of $O\left(\varepsilon^{-1} \cdot \log^2(\varepsilon n) \cdot \log\log(1/\delta)\right)$ universe items, and from which an estimate $\hat{\mathrm{R}}(y)$ of $\mathrm{R}(y)$ 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 $|\hat{\mathrm{R}}(y) - \mathrm{R}(y)| \le \varepsilon\, \mathrm{R}(y)$.*

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 $c \geq 1$, 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.

Finally, we fully describe the merge procedure and analyze the accuracy of our sketch under an arbitrary sequence of merge operations in Appendix C (for didactic purposes, we outline a simplified merge operation in Section 2.3). As mentioned above, Appendix D contains an alternative analysis that yields better space bounds for extremely small failure probabilities δ.

1.1 Detailed Comparison to Prior Work

Some prior works on streaming quantiles consider queries to be *ranks* $r \in \{1, \dots, n\}$, and the algorithm must identify an item $y \in \mathcal{U}$ such that $\mathrm{R}(y)$ 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 $\mathrm{R}(y)$. 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.[3]

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. [23] 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 [22] 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 $\mathrm{R}(y)/n \in [(1-\varepsilon)\phi, (1+\varepsilon)\phi]$ (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: [23] 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 $\mathrm{R}(y) = \phi n$, then the algorithm should return an item $\hat{y} \in \mathcal{U}$ such that $|y - \hat{y}| \leq \varepsilon \cdot |y|$. 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 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,

[3] 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.

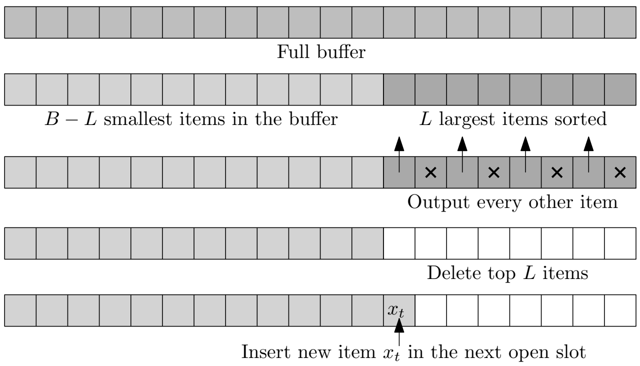

Figure 1: Illustration of the execution of a relative-compactor when inserting a new item x_t into a buffer that is full at time t. See Lines 5-14 of Algorithm 1.

the output of the last relative-compactor is of size $O(1/\varepsilon)$, and hence can be stored in memory.

Compaction Operations. 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 $\mathrm{R}(y; X) = |\{x \in X | x \le y\}|$ 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 $\mathrm{R}(y; X)$ is even (odd), where X is the input sequence to the operation. If y is even w.r.t the compaction, then $\mathrm{R}(y; X) - 2\,\mathrm{R}(y; Z) = 0$. Otherwise $\mathrm{R}(y; X) - 2\,\mathrm{R}(y; Z)$ 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 $L \le B/2$ 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 $L \le B/2$.[4]

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

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

[4]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.

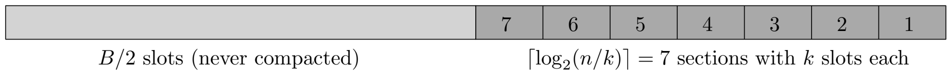

Figure 2: Illustration of a relative-compactor and its sections, together with the indexes of the sections.

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 $L_C \leq B/2$ 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 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. □

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

```
Let RelCompactors be a list of relative-compactors
Set H = 0, initialize relative-compactor at RelCompactors[0], with parameters k and n
for t = 1 ... do
    INSERT(x_t, 0)
function INSERT(x,h)
    if H < h then
        Set H = h
        Initialize relative-compactor at RelCompactors[h], with parameters k and n
    Insert item x into RelCompactors[h]
    for z in output stream of RelCompactors[h] do
        INSERT(z, h + 1)
function ESTIMATE-RANK(y)
    Set R̂(y) = 0
    for h = 0 to H do
        for each item y' ≤ y stored in RelCompactors[h] do
            Increment R̂(y) by 2^h
    return R̂(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. Later, in Appendix C, we show how to remove the latter assumption 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''. We explain the intuition behind using the bitwise OR in Appendix C, where we also prove an extension of Fact 4 for mergeability. 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''

for $h = 0, \ldots, S''.H$ **do** ▷ Merge S'' into S'
  S'.RelCompactors[h].C = S'.RelCompactors[h].C OR S''.RelCompactors[h].C
  Insert all items in S''.RelCompactors[h] into S'.RelCompactors[h]
for $h = 0, \ldots, S'.H$ **do**
  if buffer S'.RelCompactors[h] exceeds its capacity **then**
    Perform compaction operation as in lines 6-13 of Algorithm 1 and insert output items into S'.RelCompactors[$h+1$]
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, $n < \varepsilon^{-1} \cdot 2^{\varepsilon^{-2}}$. 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 $\mathrm{R}(y)$ be the rank of item y in the input stream, and $\mathrm{Err}(y) = \hat{\mathrm{R}}(y) - \mathrm{R}(y)$ 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 $\mathrm{R}(y)/(k \cdot 2^h)$ 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 $2^{H_y} \gtrsim \mathrm{R}(y)/B$ (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):

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

where in the LHS, $\mathrm{R}(y)/(k2^h)$ 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)

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

The first inequality in Equation (2) exploits Property 2 above, while the last equality exploits the fact that $B = O(k \cdot \log(\varepsilon n))$.[5] We obtain the desired accuracy guarantees so long as this variance is at most $\varepsilon^2\,\mathrm{R}(y)^2$, as this will imply that the standard deviation is at most $\varepsilon\,\mathrm{R}(y)$. This hoped-for variance bound holds so long as $B \gtrsim \varepsilon^{-1} \cdot \sqrt{\log_2(\varepsilon n)}$, or equivalently $k \gtrsim \varepsilon^{-1}/\sqrt{\log_2(\varepsilon n)}$.

2.5 Roadmap for the Formal Analysis

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.

For the analysis under an arbitrary sequence of merge operations (i.e., for the proof of full mergeability), we refer to Appendix C.

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

[5] In the derivations within Equation (2), there is a couple of important subtleties. The first is that when we replace 2^{H_y} with $\Theta(\mathrm{R}(y)/B)$, that substitution is only valid if $\mathrm{R}(y)/B \geq \Omega(1)$. However, we can assume w.l.o.g. that $\mathrm{R}(y) \geq B/2$, 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 $\varepsilon^{-1} > \sqrt{\log_2(\varepsilon n)}$.

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 $X^t = \{x_1, \dots, x_t\}$ 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

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

Conceptually, $\mathrm{Err}_h^t(y)$ 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 $\mathrm{Err}_h^n(y)$, where n is the length of its input stream. To bound $\mathrm{Err}_h^n(y)$, we keep track of the error associated with y over time, and define the increment or decrement of it as

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

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

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 $\mathrm{Err}_h^n(y)$. We call a step t with $\Delta_h^t(y) \neq 0$ *important*. Likewise, call an item x with $x \leq y$ *important*. Let $\mathrm{R}_h(y)$ be the rank of y in the input stream to level h; so there are $\mathrm{R}_h(y)$ important items inserted to the buffer at level h (in the notation above, we have $\mathrm{R}_h(y) = \mathrm{R}(y; X^n)$). 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 $\mathrm{R}_h(y)/k$ 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 $\mathrm{R}_h(y)$ in the input stream to level h, there are at most $\mathrm{R}_h(y)/k$ important steps. In particular,*

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

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 $\mathrm{R}_{h,j}(y)$ 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 $\mathrm{R}_{h,j}(y)$ 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 $\sum_{j \geq 1} \mathrm{R}_{h,j}(y) \leq \mathrm{R}_h(y)$. We will show

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

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

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

To show the lower bound on $\mathrm{R}_{h,j+1}(y)$ 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 $\mathrm{R}_{h,j+1}(y)$ 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,

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

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 $0 < \varepsilon \le 1$ and desired upper bound $0 < \delta \le 0.5$ 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 $\delta \le 1/\exp(\varepsilon n/64)$ the accuracy guarantees hold nearly deterministically and furthermore, the analysis in Appendix C does not require 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:*

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

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 $k \le 8\varepsilon^{-1} \cdot \sqrt{\ln(1/\delta)} \le 8\varepsilon^{-1} \cdot \sqrt{\varepsilon n/64} = \varepsilon^{-1} \cdot \sqrt{\varepsilon n}$, which gives us

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

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

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

□

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 $\mathrm{R}_h(y) > 0$.

Observation 7. $\mathrm{R}_{h+1}(y) \le \max\{0, \mathrm{R}_h(y) - B/2\}$ *for any* $h \ge 0$.

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 $\mathrm{R}_{h+1}(y) \le \max\{0, \mathrm{R}_h(y) - B/2\}$. □

Next, we prove that $\mathrm{R}_h(y)$ 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 $2^{2-h}\,\mathrm{R}(y) \le B/2$. For $h = H(y) - 1$ (assuming $H(y) > 0$), we particularly have $2^{3-H(y)}\,\mathrm{R}(y) \ge B/2$, or equivalently

$$2^{H(y)} \le 2^4 \cdot \mathrm{R}(y)/B. \tag{8}$$

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 $\mathbb{E}[\exp(sX)] \le \exp(-\frac{1}{2} \cdot s^2 \cdot \sigma^2)$ 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:[6]

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*

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

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

Proof. We prove by induction on $0 \le h < H(y)$ that, conditioned on $\mathrm{R}_\ell(y) \le 2^{-\ell+1}\,\mathrm{R}(y)$ for any $\ell < h$, with probability at least $1 - \delta \cdot 2^{h-H(y)}$ it holds that $\mathrm{R}_h(y) \le 2^{-h+1}\,\mathrm{R}(y)$. Taking the union bound over all $0 \le h < H(y)$ implies the claim. As $\mathrm{R}_0(y) = \mathrm{R}(y)$, the base case follows immediately.

Next, consider $h > 0$ and condition on $\mathrm{R}_\ell(y) \le 2^{-\ell+1}\,\mathrm{R}(y)$ 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 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 $\mathrm{R}_\ell(y)$ for any level $\ell > 0$ is generated by the following random process: To get $\mathrm{R}_\ell(y)$, start with $\mathrm{R}_{\ell-1}(y)$ important items and remove those stored in the level-$(\ell-1)$ relative-compactor $\mathcal{B}_{\ell-1}$ at the end of execution; there are $\mathrm{R}(y; \mathcal{B}_{\ell-1}) \le B$ 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, $\mathrm{R}_{\ell-1}(y) - \mathrm{R}(y; \mathcal{B}_{\ell-1})$ important items are involved in compaction operations at level $\ell - 1$. Summarizing, we have

$$\mathrm{R}_\ell(y) = \frac{1}{2} \cdot \left(\mathrm{R}_{\ell-1}(y) - \mathrm{R}(y; \mathcal{B}_{\ell-1}) + \mathrm{Binomial}(m_{\ell-1})\right) , \tag{9}$$

where $\mathrm{Binomial}(n)$ 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

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

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 $Y_0, \dots, Y_h$: Start with $Y_0 = \mathrm{R}(y)$ and for $0 < \ell < h$ let

$$Y_\ell = \frac{1}{2} \cdot (Y_{\ell-1} + \mathrm{Binomial}(m_{\ell-1})) \,. \tag{10}$$

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

$$\Pr[\mathrm{R}_h(y) > 2^{-h+1}\,\mathrm{R}(y)] \le \Pr[Y_h > 2^{-h+1}\,\mathrm{R}(y)] \,, \tag{11}$$

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

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

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

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

since $\mathrm{Binomial}(n)$ 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 $m_\ell \le \mathrm{R}_\ell(y)/k \le 2^{-\ell+1}\,\mathrm{R}(y)/k$ by Lemma 5 and by conditioning on $\mathrm{R}_\ell(y) \le 2^{-\ell+1}\,\mathrm{R}(y)$. As $\mathrm{Var}[\mathrm{Binomial}(n)] = n$, the variance of Z_h is

$$\begin{aligned}\mathrm{Var}[Z_h] &\le \sum_{\ell=0}^{h-1} 2^{-2h+2\ell} \cdot m_\ell \le \sum_{\ell=0}^{h-1} 2^{-2h+2\ell} \cdot \frac{2^{-\ell+1}\,\mathrm{R}(y)}{k} \\ &= \sum_{\ell=0}^{h-1} \frac{2^{-2h+\ell+1}\,\mathrm{R}(y)}{k} \le \frac{2^{-h+1} \cdot \mathrm{R}(y)}{k} \,.\end{aligned}$$

Note that $\Pr[Y_h > 2^{-h+1}\,\mathrm{R}(y)] = \Pr[Z_h > 2^{-h}\,\mathrm{R}(y)]$. To bound the latter probability, we apply the tail bound for sub-Gaussian variables (Fact 8) to get

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

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

In what follows, we condition on the bound on $\mathrm{R}_h(y)$ in Lemma 9 for any $h < H(y)$.

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

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

$$\mathrm{R}_{H(y)-1}(y) \le 2^{2-H(y)}\,\mathrm{R}(y) \le \frac{1}{2}B \,.$$

Invoking Observation 7, we get $\mathrm{R}_{H(y)}(y) \le \max\{0, \mathrm{R}_{H(y)-1}(y) - B/2\} = 0$. □

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

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

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 $\mathrm{Err}_h(y) = \sum_{t=1}^{n} \Delta_h^t(y)$: 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 $\mathrm{Err}(y)$.

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

Proof. Consider the relative-compactor at any level h. By Lemma 5, $\mathrm{Err}_h(y)$ is a sum of at most $\mathrm{R}_h(y)/k$ random variables, i.i.d. uniform in $\{-1, 1\}$. In particular, $\mathrm{Err}_h(y)$ is a zero-mean sub-Gaussian random variable with $\mathrm{Var}[\mathrm{Err}_h(y)] \le \mathrm{R}_h(y)/k$. Thus, $\mathrm{Err}(y)$ 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 $h \ge H(y)$. For $h < H(y)$, we use $\mathrm{Var}[\mathrm{Err}_h(y)] \le \mathrm{R}_h(y)/k$ to obtain:

$$\begin{aligned}\mathrm{Var}[\mathrm{Err}(y)] &= \sum_{h=0}^{H(y)-1} 2^{2h}\,\mathrm{Var}[\mathrm{Err}_h(y)] \le \sum_{h=0}^{H(y)-1} 2^{2h} \cdot \frac{\mathrm{R}_h(y)}{k} \\ &\le \sum_{h=0}^{H(y)-1} 2^{h+1} \cdot \frac{\mathrm{R}(y)}{k} \le 2^{H(y)+1} \cdot \frac{\mathrm{R}(y)}{k} \le 2^5 \cdot \frac{\mathrm{R}(y)^2}{k \cdot B} \,,\end{aligned}$$

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 $0 < \delta \le 0.5$ and $0 < \varepsilon \le 1$ 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 $\Pr[|\mathrm{Err}(y)| \ge \varepsilon \mathrm{R}(y)] < 3\delta$. If $\varepsilon \le 4 \cdot \sqrt{\ln(1/\delta)/\log_2(\varepsilon n)}$, 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 $\mathrm{R}_h(y) \le 2^{-h+1}\,\mathrm{R}(y)$ 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

$$\begin{aligned}
\Pr[|\mathrm{Err}(y)| \ge \varepsilon \mathrm{R}(y)] &< 2\exp\left(-\frac{\varepsilon^2 \cdot \mathrm{R}(y)^2}{2 \cdot 2^5 \cdot \mathrm{R}(y)^2/(k \cdot B)}\right) \\
&\le 2\exp\left(-\frac{\varepsilon^2 \cdot 2^6 \cdot \varepsilon^{-2} \cdot \ln\frac{1}{\delta}}{2^6}\right) \\
&= 2\exp\left(-\ln\frac{1}{\delta}\right) = 2\delta\,,
\end{aligned}$$

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 $\lceil \log_2(n/B) \rceil + 1 \le \log_2(\varepsilon n)$ 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

$$\begin{aligned}
&\log_2(\varepsilon n) \cdot 2 \cdot k \cdot \left\lceil \log_2 \frac{n}{k} \right\rceil \\
&\qquad \le \log_2(\varepsilon n) \cdot 2 \cdot 2 \cdot \left\lceil \frac{4}{\varepsilon} \cdot \sqrt{\frac{\ln\frac{1}{\delta}}{\log_2(\varepsilon n)}} \right\rceil \cdot O(\log(\varepsilon n))\,, \qquad (14)
\end{aligned}$$

where we use that $\lceil \log_2(n/k) \rceil \le O(\log(\varepsilon n))$, which follows from $k \ge \varepsilon^{-1}/\sqrt{\log_2(\varepsilon n)}$. In the case $\varepsilon \le 4 \cdot \sqrt{\ln(1/\delta)/\log_2(\varepsilon n)}$, we have $a := 4\varepsilon^{-1} \cdot \sqrt{\ln(1/\delta)/\log_2(\varepsilon n)} \ge 1$, so $\lceil a \rceil \le 2a$ 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 $c \ge 1$, 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 $N_{i+1} = N_i^2$.[7] This process occurs at most $\log_2\log_2(\varepsilon n)$ many times, 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 $\ell \le \log_2\log_2(\varepsilon n)$. 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.

[7] 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 analysis in Appendix C (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 of Appendix C).

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

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 $\operatorname{Var}[\operatorname{Err}^i(y)] \leq 2^5 \cdot \mathrm{R}(y; \sigma_i)^2/(k_i \cdot B_i)$. 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

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

where the last inequality uses that $\sum_{i=0}^{\ell} \mathrm{R}(y; \sigma_i)^2 \leq \mathrm{R}(y)^2$, which follows from $\mathrm{R}(y) = \sum_{i=0}^{\ell} \mathrm{R}(y; \sigma_i)$, 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. □

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.

Acknowledgments. The research is performed in close collaboration with DataSketches https://datasketches.apache.org/, the Apache open source project for streaming data analytics. G. Cormode and P. Veselý were supported by European Research Council grant ERC-2014-CoG 647557. J. Thaler was supported by NSF SPX award CCF-1918989, and NSF CAREER award CCF-1845125.

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, 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 Saidakov, 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] 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.

[23] 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 $\log_2\binom{|\mathcal{U}|}{|S|} = \Theta\left(\log((|\mathcal{U}|/|S|)^{|S|})\right) = \Theta\left(|S|\log\left(\varepsilon|\mathcal{U}|\right)\right)$ 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 $\{y_1, \dots, y_s\}$ denote the elements of S in increasing order. Consider the stream σ where items $y_1, \dots, y_\ell$ each appear once, items $y_{\ell+1}, \dots, y_{2\ell}$ appear twice, and in general items $y_{i\ell+1}, \dots, y_{(i+1)\ell}$ appear 2^i times, for $i = 0, \dots, k-1$. 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 $0, \dots, i-1$, 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. □

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 $1, \dots, 2\ell$ 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 $|S| = \Theta\left(\ell \cdot \log(\varepsilon n)\right)$ 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

Here we prove Corollary 1, restated for the reader's convenience.

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 $|\mathrm{R}(y) - \mathrm{R}(x)| \leq (\varepsilon/3) \cdot \mathrm{R}(x)$. 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 $\hat{\mathrm{R}}(y)$ such that $|\hat{\mathrm{R}}(y) - \mathrm{R}(y)| \leq (\varepsilon/3) \cdot \mathrm{R}(y)$, then we can approximate $\mathrm{R}(x)$ by $\hat{\mathrm{R}}(y)$ using $y \in S^*$ with $|\mathrm{R}(y) - \mathrm{R}(x)| \leq (\varepsilon/3) \cdot \mathrm{R}(x)$ and the multiplicative guarantee for x follows from

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

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 $\delta' = \delta/|S^*| = \Theta\left(\delta \cdot \varepsilon/\log(\varepsilon n)\right)$. 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. □

C FULL MERGEABILITY

We show that our sketch is fully mergeable, as formalized in Theorem 1 from Section 1. Fully-mergeable sketches allow the ability to sketch many different streams (or any inputs) and then merge the resulting sketches (via an arbitrary sequence of pairwise merge operations) to get an accurate summary of the concatenation of the streams. Mergeable sketches form an essential primitive for parallel and distributed processing of massive data sets.

The merge operation takes as input two sketches S' and S'' that processed two separate streams σ' and σ'' and outputs a sketch S that summarizes the concatenated stream $\sigma = \sigma' \circ \sigma''$ (the order of σ' and σ'' does not matter here). For full mergeability, S must satisfy the space and accuracy guarantees as if it was created by processing stream σ in one pass. Moreover, we do not assume that we built S' by processing stream σ' directly and similarly for S'', but we allow to create S' and S'' using merge operations. Thus, we may create the resulting summary from many summaries by merging them in an arbitrary way.

We stress that we do not assume any advance knowledge about n, the total size of all the inputs merged, which indeed may not be available in many applications.

C.1 Merge Operation

In this section, we describe a merge operation of our sketch, without assuming a foreknowledge of the total input size n. The description builds on Section 2.3, which outlines a simplified merge procedure under the assumption that a polynomial upper bound on n is available. To facilitate the merge operation, each sketch maintains a list RelCompactors of its relative-compactors and the following variables:

H = index of the highest level with a relative-compactor in the sketch.

n = size of the input currently summarized by the sketch.

N = an upper bound on n, based on which the subsequent parameters k and B (defined below) are calculated.

$\hat{k}$ = a parameter that depends on the desired accuracy ε and failure probability δ, namely, $\hat{k} = \varepsilon^{-1} \cdot \sqrt{\ln(1/\delta)}$—see (25) in Section C.5. Unlike N, the parameter $\hat{k}$ remains constant during the computation. The section size parameter k (defined below) depends on $\hat{k}$ in addition to N.

k = size of a buffer section.

B = size of the buffer at each level.

The parameter N is set similarly as in Section 5, that is, it is equal to N_i for some i, where $N_0 = \lceil 2^8 \cdot \hat{k} \rceil$ and $N_{i+1} = N_i^2$. We set the parameters k and B based on N similarly as in Section 4 (cf. Equation (6)), namely,

$$k(N) = 2^5 \cdot \left\lceil \frac{\hat{k}}{\sqrt{\log_2(N/\hat{k})}} \right\rceil \text{ and } B(N) = 2 \cdot k(N) \cdot \left\lceil \log_2\left(\frac{N}{k(N)}\right) + 1 \right\rceil . \tag{15}$$

Note that compared to (6), we increase the constant factor in front of the ceiling function in the definition of k, which ensures that $k \geq 2^5$, and we increase the number of buffer sections by one.

The merge operation that creates sketch S from S' and S'' goes as follows: Suppose that both S' and S'' are based on the same parameter $\hat{k}$ and that S' has at least as many levels as S'' (otherwise, we swap the sketches). Then, via the following procedure, we merge S'' into S', so S'' acts as a source sketch, while S' is a target sketch of the merge operation. First, we compute the parameters of the resulting sketch. For sketch S resulting from the merge operation, $S.n$ is just a sum of $S'.n$ and $S''.n$. If $S'.N \geq S.n$, then we keep parameters N, k, and B as they are set in S'. Otherwise, $S'.N < S.n = S'.n + S''.n$, so $S'.N$ would be too small after merging. In this case, we choose the next upper bound by setting $S.N = S'.N^2$ and also recompute k and B as described in Equation (15) above.

For technical reasons, before changing the parameters we perform a *special* compaction operation at each level of both S' and S''. More precisely, we perform these special compactions on each level h of S' if $S.N > S'.N$ and the level-h buffer of S' contains at least $(S'.B)/2$ items, and on level h of S'' if $S.N > S''.N$ and the level-h buffer of S'' contains at least $(S''.B)/2$ items. Note that from the viewpoint of S'', parameters may change (i.e., $S''.N < S.N$) even if $S.N = S'.N$. A special compaction on S' leaves at most $S'.B/2$ items at each level and similarly for S''. We call compaction operations that are not special *scheduled*.

Recall from Section 2.3 that the crucial part of the merge operation is to combine the states of the compaction schedules at each level in a manner that ensures that relative-error guarantees are satisfied for the merged sketch.[8] 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''; we explain the intuition behind using the bitwise OR below. 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.

Having set up the parameters and states at each level, we concatenate the level-h buffers of S' and of S'' at each level that appears in both of them. Then we perform a single compaction operation at each level that has at least $S.B$ items, in the bottom-up fashion. For such a compaction operation, all but the smallest $S.B$ items in the buffer are automatically included in the compaction, while the smallest B items are treated exactly as a full buffer is treated in the streaming setting to determine what suffix is compacted. That is, the state variable C of the compaction schedule determines how many sections amongst the smallest B items in the buffer are compacted, via the number of trailing 1s in the binary representation of C. If this number of trailing 1s is $j \geq 0$, then $j+1$ sections are compacted and we say that the compaction *involves exactly* $j+1$ *sections of the buffer*. Thus, there is at most one compaction per level during the merge operation.

Algorithm 4 provides a pseudocode describing the merge operation specified above. For simplicity, in the pseudocode we merge sketch S'' into S' so that after performing the procedure S' becomes the sketch S resulting from the merge operation . We remark that inserting a single item x can be viewed as a trivial merge with a summary consisting just of x (with weight 1).

Several remarks and observations are in order. First, the combined buffer contains at most $2 \cdot S.B$ items before the merge procedure begins performing compactions level-by-level, because each buffer of S' and each buffer of S'' stores at most $S.B$ items. Second, when we perform a compaction on a level-h buffer during a merge procedure, it contains no more than $\frac{7}{2} \cdot S.B$ items. To see this, observe that there are three sources of input to the buffer at level h during a merge operation: the at most $S.B$ items in S' at level h at the start of the merge operation, the at most $S.B$ items in S'' at level h at the start of the merge operation, and the output of the level-$(h-1)$ buffer during the merge procedure. An easy inductive argument shows that the third source of inputs consists of at most $\frac{3}{2} \cdot S.B$ item, as follows: Observe that if the level-$(h-1)$ buffer has size at most $\frac{7}{2}S.B$ when it is compacted, then the number of items compacted by that buffer is at most $\frac{7}{2}S.B - \frac{1}{2}S.B = 3S.B$, and hence the number of items output by the compaction is at most $\frac{3}{2} \cdot S.B$ (here, we also use that $S.B$ is divisible by four by (15), so $\frac{3}{2} \cdot S.B$ is

[8] By state of the compaction schedule, we mean the variable that determines how many sections of the buffer to include in a compaction operation if one is performed. In the streaming setting (Algorithm 1), we denoted this variable by C, and maintain this notation in the mergeability setting.

Algorithm 4 Merge operation

Input: Sketches S' and S'' to be merged such that $S'.\hat{k} = S''.\hat{k}$ and $S'.H \geq S''.H$

Output: A sketch answering rank queries for the combined inputs of S' and S'' ▷ We merge S'' into S'

1: Set $S'.n = S'.n + S''.n$ ▷ Combined input size
2: **if** $S'.N < S'.n$ **then** ▷ Upper bound on input size is too small
3: SpecialCompactions(S')
4: Set $S'.N = S'.N^2$ ▷ Square the upper bound
5: Set $S'.k$ and $S'.B$ according to (15)
6: **if** $S''.N < S'.N$ **then**
7: SpecialCompactions(S'')
8: **for** $h = 0, \ldots, S''.H$ **do**
9: ▷ Combine buffers and states of compaction schedules
10: Insert all items in S''.RelCompactors$[h]$ into S'.RelCompactors$[h]$
11: S'.RelCompactors$[h].C$ = S'.RelCompactors$[h].C$ OR S''.RelCompactors$[h].C$
12: **for** $h = 0, \ldots, S'.H$ **do**
13: **if** there are at least $S'.B$ items in S'.RelCompactors$[h]$ **then**
14: ▷ Scheduled compaction
15: Compute z = number of trailing 1s in binary representation of S'.RelCompactors$[h].C$
16: Set $s = S'.B - (z+1) \cdot S'.k + 1$ ▷ First slot of the buffer involved in the compaction
17: PerformCompaction(S', h, s)
18: **return** S'

19: **function** SpecialCompactions($\hat{S}$)
20: **for** $h = 0, \ldots, \hat{S}.H - 1$ **do**
21: PerformCompaction($\hat{S}$, h, $\hat{S}.B/2$) ▷ Does nothing if $\hat{S}[h]$ contains $\leq \hat{S}.B/2$ items

22: **function** PerformCompaction($\hat{S}$, h, s)
23: **if** $h = \hat{S}.H$ **then**
24: Increase $\hat{S}.H$ by one
25: Initialize relative-compactor at RelCompactors$[h+1]$
26: Set $\mathcal{B} = \hat{S}$.RelCompactors$[h]$ ▷ The level-h buffer of $\hat{S}$
27: Let $|\mathcal{B}|$ be the number of items stored in $\mathcal{B}$
28: Sort items in $\mathcal{B}$ in non-descending order
29: Set Z = equally likely either even or odd indexed items in the range $\mathcal{B}[s : |\mathcal{B}|]$
30: ▷ Note that the range $\mathcal{B}[s : |\mathcal{B}|]$ may be of an odd size, which does not cause any issues
31: Insert each item in Z to S.RelCompactors$[h+1]$
32: Mark slots $\mathcal{B}[s : |\mathcal{B}|]$ in the buffer as clear
33: Increase $\mathcal{B}.C$ by 1

even). This guarantees that at the time a level-h buffer is actually compacted during a merge procedure, it contains no more than $\frac{7}{2} \cdot S.B$ items.

Third, using the bitwise OR in line 11 to combine the states has two simple but important implications.

Fact 15. *When the j-th bit of C' or of C'' is set to 1, then the j-th bit of $C = C'$ OR C'' is also set to 1.*

We use this basic property of bitwise OR to show an analogue of Fact 4—informally that between every two compaction operations involving exactly j sections, there is one that involves more than j sections. See Fact 18 for details.

Fact 16. *The bitwise OR of C' and C'' (interpreted as bitstrings) is no larger than $C' + C''$ (interpreted as integers).*

Fact 16 will be used later to show that the state C never has more than $\lceil \log_2(S.N/S.k) \rceil$ bits, so we never compact more than $\lceil \log_2(S.N/S.k) \rceil$ buffer sections during a scheduled compaction (only the special compaction involves all $\lceil \log_2(S.N/S.k) \rceil + 1$ sections). See Observation 17 for details.

C.2 Preliminaries for the Analysis of the Merge Procedure

Consider a sketch S built using an arbitrary sequence of merge operations from an input of size n. As we will show, that the space bound holds for S follows from an argument similar to the one in the proof of Theorem 13, but the calculation of the failure probability needs to be modified compared to Section 4. The main challenge is that the parameters k and B change as more and more merge operations are performed. We introduced special compactions into the merge procedure specifically to handle these changes in parameters in the analysis.

To prove that the accuracy guarantees hold for S, consider the binary tree T in which each of n leaves corresponds to a single item of the input. Internal nodes correspond to merge operations (recall that inserting one item to the sketch can be seen as the merge of the sketch with a trivial sketch storing the item to be inserted), and hence each internal node t in T represents a sketch S_t resulting from the merge operation that corresponds to node t. Also, for a particular level h, t represents the level-h buffer of S_t. The root of T represents the final merge operation, which outputs S. The two children of each internal node are ordered so that the left child corresponds to the target sketch S' and the right child to the source sketch S''.

Recall that the merge operation captured by an internal node t performs at most one scheduled compaction operation at each level h, and we say that t represents the level-h compaction operation (if any). We further introduce additional nodes to represent special compactions. Suppose that at node t, sketch S' (represented by node t') is to be merged with sketch S'' (represented by node t'') to form sketch S. If special compactions are performed on S' during the merge operation, then we subdivide the edge between nodes t' and t by adding a new node t'_1 to represent the special compaction operation applied to S' and include the two edges (t', t'_1) and (t'_1, t). Similarly, if a special compaction is performed on S'', we subdivide the edge between nodes t'' and t by adding a new node t''_1 and including the two edges (t'', t''_1) and (t''_1, t).

Recall that we set the upper bounds N on the input size used by the sketches as $N_0 = \lceil 2^8 \cdot \hat{k} \rceil$ and $N_i = N_{i-1}^2$ for $0 \leq i \leq \ell \leq \lceil \log_2 \log_2(\varepsilon n) \rceil$ (as $N_0 \geq \hat{k} \geq 1/\varepsilon$). We may assume that $\ell > 0$, otherwise the whole input can be stored in space $O(\hat{k}) = O(\varepsilon^{-1} \cdot \sqrt{\log(1/\delta)})$. Let k_i and B_i denote the parameters k and B set via Equation (15) with $N = N_i$, i.e., $k_i = k(N_i)$ and $B_i = B(N_i)$.

We say that an (internal) node t in tree T is an i-*node* for $0 \leq i \leq \ell$ if the sketch S_t represented by t satisfies $S_t.N = N_i$, i.e., it uses the parameters k_i and B_i. Note that this means that if parameter N is updated from N_{i-1} to N_i during the merge operation represented by t, then t is considered an i-node. Moreover, we say that node t is a *topmost* i*-node* if the parent of t is a j-node for some $j > i$ or t is the root of T. Note that the subtrees of topmost i-nodes are disjoint and that topmost i-nodes for $i < \ell$ represent special compactions and have just one child.

To simplify the presentation of the analysis, we assume that a special compaction is also done at the root of T (which is the single topmost ℓ-node). The actual algorithm does not perform such a special compaction at the root of T, and our analysis in fact applies even in the absence of such a special compaction (namely, we would consider a special case of $i = \ell$ in the proof of Lemma 19 separately).

As in Sections 3 and 4, we consider a fixed item y and analyze the error of the estimated rank of y. Recall that $\mathrm{R}(y)$ denotes the rank of y in the input summarized by the sketch and that $\hat{\mathrm{R}}(y)$ is the estimated rank of y obtained from the final sketch S. Our aim is to show that $|\operatorname{Err}(y)| = |\hat{\mathrm{R}}(y) - \mathrm{R}(y)| \le \varepsilon\,\mathrm{R}(y)$ with probability at least $1 - \delta$.

C.3 Analysis of a Single Level

For the duration of this section, we consider a single level h. We start by showing that the binary representation of the state C at level h never has more than $\lceil \log_2(S.N/S.k) \rceil$ bits, or equivalently, $C \le S.N/S.k$. Consequently, C (viewed as a bitstring) never has $\lceil \log_2(S.N/S.k) \rceil$ trailing ones just before a compaction operation (as after the operation, it would have more than $\lceil \log_2(S.N/S.k) \rceil$ bits).

Observation 17. *Consider a node t of tree T and sketch S represented by t. Let C be the state of the level-h buffer of S. Then $C \le S.N/S.k$.*

Proof. Let r be the number of items removed from the level-h buffer of S during compactions represented by nodes in the subtree of t. We show that $C \le r/S.k$ by induction. This implies $C \le S.N/S.k$ as $r \le S.n \le S.N$.

The base case of a leaf node follows as $C = 0$ and $r = 0$. Let S be the sketch represented by an internal node and let S' and S'' be the sketches represented by its children. Let C' and C'' be the states of the level-h buffers of S' and S'', and let r' and r'' be the number of items removed from the level-h buffer during compactions represented by nodes in the subtrees of S' and S'', respectively. By the induction hypothesis, we have $C' \le r'/S'.k$ and $C'' \le r''/S''.k$. Note that r equals $r' + r''$ plus the number of items removed from the level-h buffer during a compaction represented by t if there is one. Let $b \in \{0, 1\}$ be the indicator variable with $b = 1$ iff there is a level-h compaction represented by t. Observe that $C = (C' \,\mathrm{OR}\, C'') + b$ and if $b = 1$, then the compaction removes at least $S.k$ items from the level-h buffer. We thus have $r \ge r' + r'' + b \cdot S.k$ and using this, we obtain

$$\begin{aligned} C = (C' \,\mathrm{OR}\, C'') + b &\le C' + C'' + b \\ &\le \frac{r'}{S'.k} + \frac{r''}{S''.k} + b \\ &\le \frac{r'}{S.k} + \frac{r''}{S.k} + \frac{b \cdot S.k}{S.k} \\ &\le \frac{r}{S.k}\,, \end{aligned}$$

where the penultimate inequality uses $S.k \le \min(S'.k, S''.k)$. □

Recall that the second half of a buffer of size B_i has $\lceil \log_2(N_i/k_i) + 1 \rceil$ sections of size k_i (see Equation (15)). The definition of the compaction operation and Observation 17 imply that section $\lceil \log_2(N_i/k_i) + 1 \rceil$ (i.e., the leftmost section of the second half of the buffer) is involved only in a special compaction (done when updating N_i to $N_{i+1} = N_i^2$).

Next, we prove an analogue of Fact 4, which for the streaming setting (where we can more easily impose a total ordering on the ocurrence of events) states that between every two compaction operations involving exactly j sections of a buffer, there is at least one compaction of section $j + 1$. Recall from Section C.1 that we say a compaction of sketch S at level h involves exactly j sections if parameter s of function PerformCompaction in Algorithm 4 equals $B - j \cdot k + 1$. In words, a compaction involves exactly j sections if j is the number of compacted sections amongst those that contain just the lowest-ranked $S.B$ items in the level-h buffer of S (higher-ranked items in the buffer are compacted regardless of the state variable $S.C$). For mergeability, we conceptually replace the notion of "how the summary evolves over time" with how the summary evolves as we traverse any leaf-to-root path in the merge tree T.

Fact 18. *Consider the relative-compactor at level h and any integer i satisfying $0 \le i \le \ell$. Suppose there are two compaction operations represented by i-nodes t and t' of tree T that involve exactly j sections such that node t' is a descendant of t. Then there exists a node $\hat{t} \notin \{t, t'\}$ on the t-t'-path in T such that there is a compaction operation in node $\hat{t}$ that involves more than j sections.*

Proof. Let C and C' denote the states of the compaction schedule just before the compaction operations represented by nodes t and t', respectively. Then we can express the binary representations of C and C' respectively as $(\mathbf{x}, 0, \mathbf{1}^{j-1})$ and $(\mathbf{x}', 0, \mathbf{1}^{j-1})$, 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 $y < z$; this relies on using the bitwise OR operation when combining the states during a merge operation (cf. Fact 15). Note that after the compaction operation in node t, the state of the compaction schedule is $(\mathbf{x}, 1, \mathbf{0}^{j-1})$, so the j-th bit from the right equals 1. Observe that this bit switches from 1 to 0 only after a compaction that involves more than j sections; this again relies on Fact 15.Since the j-th bit from the right is 0 in node t', there must be a node $\hat{t}$ on the t-t'-path in T such that there is a compaction operation in node $\hat{t}$ that involves more than j sections. □

As in Section 3, the key part of the analysis is bounding the number of compaction operations that introduce some error for the fixed item y; recall that we call such compactions important. Also, recall that we call items $x \le y$ important and that for $h > 0$, $\mathrm{R}_h(y)$ denotes the total number of important items promoted to level h during compaction operations at level $h - 1$ (represented by any node in T). For level 0, we have $\mathrm{R}_0(y) = \mathrm{R}(y)$. Note that a compaction is important (i.e., affects the error for y) if and only if it involves an odd number of important items, by Observation 3.

We start by bounding the number of important scheduled level-h compactions represented by i-nodes for a fixed i. Then we use this lemma to show a bound for multiple i's.

Lemma 19. *Consider level h and $0 \le i \le \ell$. Let $\overline{m}_h^i$ be the number of important scheduled compaction operations at level h represented* *by i-nodes. It holds that $\overline{m}^i_h \le \overline{R}^i_h(y)/k_i$, where $\overline{R}^i_h(y)$ is the number of important items that are removed from level h during a compaction represented by an i-node, including those removed by special compactions at level h represented by topmost i-nodes.*

Proof. For simplicity, when we refer to a buffer or a compaction operation we mean the one at level h. The proof is by an extension of the charging argument in Lemma 5. We aim to charge each important compaction (except for special compactions) represented by an i-node to k_i important items that are removed during a compaction represented by an i-node (possibly a special compaction represented by a topmost i-node). Moreover, we will show that each important item is charged at most once, which will imply that there are at most $\overline{R}^i_h(y)/k_i$ important scheduled compactions represented by i-nodes.

Call an i-node t *heavy* if both children of t have an important compaction represented by an i-node in their subtrees. This implies that both children of t are i-nodes and that both buffers (of the sketches) represented by t's children have at least $B_i/2$ important items.

Consider an i-node t representing an important scheduled compaction that involves exactly j buffer sections. Note here that $j \le \lceil \log_2(N_i/k_i) \rceil$ by Observation 17.

Let t' be the i-node on the path from t to the root in T such that either t' is heavy and t is in the subtree of the right child of t', or t' represents a compaction operation involving more than j sections; if there are more such nodes, t' is the lowest such node (i.e., the closest to t). Note that t' is well-defined, since the topmost i-node on the path from t to the root represents a special compaction that involves all $\lceil \log_2(N_i/k_i) \rceil + 1$ buffer sections (recall that we perform a special compaction only if the level-h buffer has more than $B_i/2$ items, which is satisfied for the topmost i-node above t as there is a scheduled compaction represented by an i-node in its subtree). Observe that after the compaction represented by t, section $j+1$ contains important items only and this property does not change until we compact section $j+1$. We consider the two cases in the definition of t' separately to define a charging scheme:

Case A: t' is heavy and t is in the subtree of the right child of t'. We claim that there are at least $(\frac{3}{2}B_i - j \cdot k_i)$ important items in the buffer just before the compaction operation represented by t'. Indeed, as t' is heavy, the sketch represented by the left child of t' has at least $B_i/2$ important items at the buffer and since after the compaction represented by t, section $j+1$ contains important items only, the sketch represented by the right child of t' has at least $B_i - j \cdot k_i$ important items; here we use that there is no compaction operation that involves section $j+1$ represented by a node on the path between t and t'. We charge the important compaction represented by t to the k_i important items at indexes in the interval $[(\frac{3}{2}B_i - (j+1) \cdot k_i + 1), (\frac{3}{2}B_i - j \cdot k_i)]$ that are all removed during the compaction represented by t'; we call this interval *extra section* $j+1$.[9]

Case B: Otherwise, t' represents a compaction operation involving more than j sections. Then we charge the important compaction represented by t to the k_i important items removed from section $j+1$ during the compaction represented by t'.

Having defined the charging scheme, it remains to observe that each of $\overline{R}^i_h(y)$ important items is charged at most once. The argument relies on the following claim:

Claim 20. *Let t_1 and t_2 be two i-nodes with $t_1 \neq t_2$ in the subtree of an i-node t' such that $t' \notin \{t_1, t_2\}$ and for both t_1 and t_2 there is an important scheduled compaction involving exactly j sections represented by the respective node. Then, the important compaction of at least one of t_1 and t_2 is charged to important items either removed by a compaction represented by a node strictly below t' (i.e., in the subtree of t' and different from t'), or removed from an extra section of t'.*

Proof. If t_2 lies on the t_1-t'-path, then Fact 18 implies that the compaction represented by t_1 is charged to items removed by a compaction represented by a node strictly below t' as there is a node $\bar{t}$ on the t_1-t_2-path representing a compaction that involves section $j+1$; a symmetric argument applies if t_1 lies on the t_2-t'-path.

Otherwise, let $\hat{t}$ be the lowest common node of the t_1-t'-path and of the t_2-t'-path; possibly $\hat{t} = t'$. Note that $\hat{t} \notin \{t_1, t_2\}$ since $t_1 \neq t_2$. W.l.o.g., suppose that t_1 is in the subtree of the left child of $\hat{t}$ and t_2 is in the subtree of the right child of $\hat{t}$. Since there are important compactions in both t_1 and t_2, node $\hat{t}$ is heavy. Hence, as t_2 is in the right subtree of $\hat{t}$, the compaction represented by t_2 is charged to important items either removed from an extra section of $\hat{t}$, or removed from some section of a node below $\hat{t}$ on the t_2-$\hat{t}$-path. ◻

To show the full claim that each important item is charged at most once for the fixed level h, consider any important item x that is removed from the level-h buffer during some compaction operation represented by an i-node t. We have two cases that correspond to the two cases of the charging scheme described above:

Case I: x is not in an extra section of the buffer just before it is removed from the buffer (i.e., x is at index at most B_i in that buffer) . Let $j+1$ be the index of the section that contains x just before it is removed (thus, the compaction involves at least $j+1$ sections); note that $j \ge 0$ as items in section 1 are not charged. Item x may only be charged by some number of important compactions represented by nodes in the subtree of t that involve exactly j sections according to Case B of the above charging scheme. Claim 20 then implies that x is charged at most once.

Case II: Otherwise, for item x to be charged to, x must be removed from an extra section during a compaction represented by a heavy node t^*. Let $j+1$ be the index of this extra section. Then x may be charged by some number of important compactions that involve exactly j sections and are represented by nodes in the subtree of the right child of t^* (according to Case A of the charging scheme). Using Claim 20 (with t' in the statement of Claim 20 equal to the right child of t^*) again implies that x is charged at most once. ◻

The following lemma combines the bounds for different i's and takes important special compactions into account. We first give a few definitions. We say that a compaction *involves important items* iff it removes at least one important item from the buffer. Recall

[9]The "extra section" terminology references the fact that $\frac{3}{2}B_i - (j+1) \cdot k_i + 1 \ge B_i + 1$, so it falls outside of our usual section numbering scheme. As buffers of i-nodes store at most B_i items at the start and end of a merge operation, any items other than the smallest B_i items in the buffer are thought of as "extra" items. All extra items in a buffer are automatically involved in a compaction at the time of the merge operation.

that we only consider a compaction to be important if it affects an odd number of important items, so these compactions involving important items are a superset of important compactions. Let Q_h be the set of nodes t such that (i) t represents a level-h compaction that involves important items, and (ii) there is no node on the t-to-root path (except for t) that represents a level-h compaction involving important items. Intuitively, Q_h captures "maximal" nodes that represent a level-h compaction removing one or more important items from level h. Observe then that an important item that remains in the level-h buffer represented by a node $t \in Q_h$ (after performing the compaction operation represented by t) is never removed from the level-h buffer, by the definition of Q_h. For $0 \le i \le \ell$, let Q^i_h be the set of i-nodes in Q_h.

Finally, for some $0 \le a \le \ell$, let $\mathrm{R}^{\ge a}_h(y)$ be the number of important items that are either removed from level h during a compaction represented by an i-node for $i \ge a$, or remain at the level-h buffer (of the sketch) represented by a node $t \in Q^i_h$ for $i \ge a$ (after the compaction operation represented by t is done).

Lemma 21. *Consider level h. For $0 \le i \le \ell$, let m^i_h be the number of important compaction operations (both scheduled and special) at level h represented by i-nodes in the merge tree T. Then for any $0 \le a \le \ell$, it holds that*

$$\sum_{i=a}^{\ell} m^i_h \cdot k_i \le 2\,\mathrm{R}^{\ge a}_h(y)\,. \tag{16}$$

Proof. By Lemma 19, we have a bound on the number of important scheduled compactions, namely, $\overline{m}^i_h \cdot k_i \le \overline{\mathrm{R}}^i_h(y)$ for any $a \le i \le \ell$. Observe that $\sum_{i=a}^{\ell} \overline{\mathrm{R}}^i_h(y)$ is the number of important items removed by compactions represented by i-nodes for $a \le i \le \ell$ (including special compactions) and that $\sum_{i=a}^{\ell} \overline{\mathrm{R}}^i_h(y) \le \mathrm{R}^{\ge a}_h(y)$, so we get

$$\sum_{i=a}^{\ell} \overline{m}^i_h \cdot k_i \le \mathrm{R}^{\ge a}_h(y)\,. \tag{17}$$

We now turn our attention to special compactions. Let $\hat{m}^i_h$ be the number of important special compactions represented by topmost i-nodes; note that $m^i_h = \overline{m}^i_h + \hat{m}^i_h$. Observe that $\hat{m}^i_h \cdot B_i/2 \le \mathrm{R}^{\ge a}_h(y)$ since the level-h buffer represented by a topmost i-node t contains at least $B_i/2$ important items if t represents an important special compaction, since these sets of important items are disjoint, and since these important items are either removed from the level-h buffer by a compaction represented by an i'-node for $i' > i \ge a$, or remain at the level-h buffer represented by a node $t' \in Q^{i'}_h$ for some $i' \ge i$ (possibly $t' = t$).

We thus have

$$\sum_{i=a}^{\ell} \hat{m}^i_h \le \sum_{i=a}^{\ell} \frac{2\,\mathrm{R}^{\ge a}_h(y)}{B_i} \le \frac{\mathrm{R}^{\ge a}_h(y)}{k_0}\,, \tag{18}$$

where we use that[10] $B_i \gtrsim \left(\sqrt{2}\right)^i \cdot B_0$ and $B_0 \ge 8 \cdot k_0$; the latter holds if $N_0 \ge 2^3 \cdot k_0$, which is implied by $N_0 \ge 2^8 \cdot \hat{k}$.

[10]This inequality holds up to rounding issues, however, as $B_0 \ge 2^5$, the error introduced by rounding is small enough to ensure the validity of the inequalities.

Since $k_0 \ge k_1 \ge \cdots \ge k_\ell$, inequality (18) implies

$$\sum_{i=a}^{\ell} \hat{m}^i_h \cdot k_i \le \mathrm{R}^{\ge a}_h(y)\,. \tag{19}$$

Combining (17) and (19) and using $m^i_h = \overline{m}^i_h + \hat{m}^i_h$ implies (16). ◻

As $k_i \ge k_\ell$ for any $i \le \ell$ and $\mathrm{R}^{\ge 0}_h(y) \le \mathrm{R}_h(y)$, Lemma 21 with $a = 0$ has a simple corollary.

Corollary 2. *Consider level h and let $m_h = \sum_{i=0}^{\ell} m^i_h$ be the total number of important compaction operations at level h across all merge operations captured by the merge tree T. Then $m_h \le 2\,\mathrm{R}_h(y)/k_\ell$.*

C.4 Analysis of the Full Sketch with an Additional Factor of $\log\log(n)$

As a warmup, in this section, we complete the proof of full mergeability, but with an additional factor of $\log_2\log_2(\varepsilon n)$ appearing in the final space bound relative to our result in the streaming setting (Theorem 13). The analysis in this section is less delicate than our analysis that avoids this $\log_2\log_2(\varepsilon n)$ factor, thereby establishing Theorem 1. We nevertheless do not assume any advance knowledge about the final input size n. (We remark that special compactions are actually not needed to achieve the result of this section, however, the analysis needs some adjustments.)

We first set the value of parameter $\hat{k}$. Namely, given the desired accuracy $0 < \varepsilon \le 1$ and desired upper bound $0 < \delta \le 0.5$ on failure probability, we let

$$\hat{k} = \frac{8}{\varepsilon} \cdot \sqrt{\ln\frac{1}{\delta}}\,. \tag{20}$$

We also modify the definitions of k_i and B_i for $i \ge 0$ compared to Equation (15), as follows:

$$k_i = 2 \cdot \left\lceil \frac{\max(i,1) \cdot \hat{k}}{\sqrt{\log_2(N_i/\hat{k})}} \right\rceil \quad \text{and} \quad B_i = 2 \cdot k_i \cdot \left\lceil \log_2 \frac{N_i}{k_i} \right\rceil. \tag{21}$$

In particular, relative to Equation (15), note the extra factor of i in the definition of k_i. Including this extra factor considerably simplifies the analysis, but it is responsible for the additional $\log_2\log_2(\varepsilon n)$ term in the space bound we obtain in this section.

Observe that for $i \le \ell$, it holds that $k_i \ge k_\ell$, since $\sqrt{\log_2(N_i/\hat{k})}$ grows faster than i. We assume that ε satisfies $\varepsilon \le 1/\sqrt[4]{2\log_2(n)}$; we remark that this is a very weak restriction put on ε as for all practical values of $n \le 2^{128}$ we have $\sqrt[4]{2\log_2(n)} \le 4$, in which case the assumption becomes $\varepsilon \le 1/4$.

We need two lower bounds on the products of k_i and B_i for $0 \le i \le \ell$. First, note that

$$\begin{aligned} k_\ell \cdot B_0 &\ge k_\ell \cdot 2 \cdot k_0 \cdot \log_2(N_0/\hat{k}) \\ &\ge \frac{8 \cdot \ell \cdot \hat{k}^2}{\sqrt{\log_2(N_\ell/\hat{k})}} \\ &\ge 2^9 \cdot \ell \cdot \frac{1}{\varepsilon^2} \cdot \ln\frac{1}{\delta} \cdot \frac{1}{\sqrt{\log_2(N_\ell/\hat{k})}} \\ &\ge 2^9 \cdot \ell \cdot \ln\frac{1}{\delta}\,, \end{aligned} \tag{22}$$

where we apply the assumption $\varepsilon \le 1/\sqrt[4]{2\log_2(n)}$ to get

$$\varepsilon^{-2}/\sqrt{\log_2(N_\ell/\hat{k})} \ge \varepsilon^{-2}/\sqrt{\log_2(n^2)} \ge 1,$$

where the first inequality uses $N_\ell/\hat{k} \le n^2$ (because $\sqrt{N_\ell} = N_{\ell-1} < n$).

Second, using a similar calculation as in Claim 6, we show a lower bound on $k_i \cdot B_i$.

Claim 22. *Parameters k_i and B_i set according to (21) and based on $\hat{k}$ as in (20) satisfy*

$$k_i \cdot B_i \ge 2^6 \cdot \frac{i^2}{\varepsilon^2} \cdot \ln\frac{1}{\delta}\,. \tag{23}$$

Proof. We first need to relate $\log_2(N_i/k_i)$ (used to define B_i) and $\log_2(N_i/\hat{k})$ (that appears in the definition of k_i). Since $k_i \le 2\cdot\hat{k}$ (this inequality is tightest for k_0), it holds that

$$\log_2(N_i/k_i) \ge \log_2(N_i/\hat{k}) - 1 \ge \log_2(N_i/\hat{k})/2,$$

where we use that $N_i \ge N_0 \ge 4\cdot\hat{k}$, so $\log_2(N_i/\hat{k}) \ge 2$. Using this, we bound $k_i \cdot B_i$ as follows:

$$\begin{aligned} k_i \cdot B_i &= 2\cdot k_i^2 \cdot \left\lceil \log_2\frac{N_i}{k_i}\right\rceil \\ &\ge 2\cdot 2^8 \cdot \frac{i^2}{\varepsilon^2}\cdot\frac{\ln\frac{1}{\delta}}{\log_2(N_i/\hat{k})}\cdot\frac{\log_2(N_i/\hat{k})}{2} \\ &= 2^8\cdot\frac{i^2}{\varepsilon^2}\cdot\ln\frac{1}{\delta}\,. \end{aligned}$$

□

For any $0 \le i \le \ell$, we define $H_i(y)$ to be the minimal h for which $2^{-h+1}\,\mathrm{R}(y) \le B_i/2$. As y is fixed, we use just H_i for brevity. For $h = H_i - 1$ (assuming $H_i > 0$), we have in particular that $2^{2-H_i}\,\mathrm{R}(y) \ge B_i/2$, or equivalently

$$2^{H_i} \le 2^3 \cdot \mathrm{R}(y)/B_i\,. \tag{24}$$

As increasing i by one increases B_i, we have $H_0 \ge H_1 \ge \cdots \ge H_\ell$. We show below that no important item (i.e., one smaller than or equal to y) can ever reach level $H_0 + 1$.

Lemma 23. *Assuming $H_0 > 0$, with probability at least $1-\delta$ it holds that $\mathrm{R}_h(y) \le 2^{-h+1}\,\mathrm{R}(y)$ for any $h \le H_0$.*

Proof. The proof is similar to that of Lemma 9, except that we need to deal with parameters k and B changing over time. We show by induction on $0 \le h \le H_0$ that $\mathrm{R}_h(y) \le 2^{-h+1}\,\mathrm{R}(y)$ with probability at least $1-\delta\cdot 2^{h-H_0-1}$, conditioned on $\mathrm{R}_{h'}(y) \le 2^{-h'+1}\,\mathrm{R}(y)$ for any $h' < h$. The base case holds by $\mathrm{R}_0(y) = \mathrm{R}(y)$.

Consider $0 < h \le H_0$, and recall that $m_{h'}$ denotes the number of important compactions at level h' over all merge operations represented in the merge tree T. As in the proof of Lemma 9,

$$\Pr[\mathrm{R}_h(y) > 2^{-h+1}\,\mathrm{R}(y)] \le \Pr[Z_h > 2^{-h}\,\mathrm{R}(y)],$$

where $Z_h = \sum_{h'=0}^{h-1} 2^{-h+h'}\cdot\mathrm{Binomial}(m_{h'})$ is a zero-mean sub-Gaussian random variable. To bound the variance of Z_h, first note that for any $h' < h$, we have that

$$m_{h'} \le \frac{2\,\mathrm{R}_{h'}(y)}{k_\ell} \le \ell\cdot\frac{2^{-h'+2}\,\mathrm{R}(y)}{k_\ell}\,,$$

using Corollary 2 for each $i \in [1,\ell]$, the fact that $k_i \ge k_\ell$, and the assumption that $\mathrm{R}_{h'}(y) \le 2^{-h'+1}\,\mathrm{R}(y)$.

As $\mathrm{Var}[\mathrm{Binomial}(n)] = n$, the variance of Z_h is

$$\begin{aligned} \mathrm{Var}[Z_h] &\le \sum_{h'=0}^{h-1} 2^{-2h+2h'}\cdot m_{h'} \\ &\le \sum_{h'=0}^{h-1} 2^{-2h+2h'}\cdot\frac{2^{-h'+2}\,\mathrm{R}(y)}{k_\ell} \\ &= \sum_{h'=0}^{h-1}\frac{2^{-2h+h'+2}\,\mathrm{R}(y)}{k_\ell} \\ &\le \frac{2^{-h+2}\cdot\mathrm{R}(y)}{k_\ell}\,. \end{aligned}$$

To bound $\Pr[Z_h > 2^{-h}\,\mathrm{R}(y)]$, we apply the Chernoff tail bound for sub-Gaussian variables (Fact 8) to get

$$\begin{aligned} \Pr[Z_h > 2^{-h}\,\mathrm{R}(y)] &< \exp\left(-\frac{2^{-2h}\cdot\mathrm{R}(y)^2}{2\cdot(2^{-h+2}\cdot\mathrm{R}(y)/k_\ell)}\right) \\ &= \exp\left(-2^{-h-3}\cdot\mathrm{R}(y)\cdot k_\ell\right) \\ &= \exp\left(-2^{-h+H_0-6}\cdot 2^{3-H_0}\,\mathrm{R}(y)\cdot k_\ell\right) \\ &\le \exp\left(-2^{-h+H_0-6}\cdot B_0\cdot k_\ell\right) \\ &\le \exp\left(-2^{-h+H_0+1}\cdot\ln\frac{1}{\delta}\right) \\ &= \delta^{2^{-h+H_0+1}} \le \delta\cdot 2^{-H_0+h-1}\,, \end{aligned}$$

where the second inequality uses $2^{3-H_0}\,\mathrm{R}(y) \ge B_0$ (by Equation 24), the third inequality follows from (22), and the last inequality uses $\delta \le 0.5$. Hence, taking the union bound over levels $h \le H_0$, with probability at least $1-\delta$ it holds that $\mathrm{R}_h(y) \le 2^{-h+1}\,\mathrm{R}(y)$ for any $h \le H_0$. □

Lemma 24. *Conditioned on the bounds in Lemma 23 holding, for any $0 \le i \le \ell$, there is no important compaction at level $h \ge H_i$ represented by an i-node.*

Proof. By Lemma 23, $\mathrm{R}_{H_i}(y) \le 2^{-H_i+1}\,\mathrm{R}(y) \le B_i/2$, where the second inequality follows from the definition of H_i. Hence, no important item is ever compacted during merge operations represented by i-nodes when the buffer size is B_i. □

We are now ready to state the main theorem of this section.

Theorem 25. *Let $0 < \delta \le 0.5$ and $0 < \varepsilon \le 1$ be parameters satisfying $\varepsilon \le 4/\sqrt[4]{2\log_2(n)}$. 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 $\hat{\mathrm{R}}(y)$ of $\mathrm{R}(y)$ such that*

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

where the probability is over the internal randomness of the streaming algorithm. If $\varepsilon \le O\left(\log\log(\varepsilon n)\cdot\sqrt{\log(1/\delta)/\log(\varepsilon n)}\right)$, then the size *of S is*

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

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

Proof. We condition on the bounds in Lemma 23 that all hold with probability at least $1-\delta$. Let $\mathrm{Err}^i_h(y)$ be the error introduced by compaction operations at level h represented by i-nodes. By Lemma 24, $\mathrm{Err}^i_h(y)=0$ for any $h\ge H_i$. For $h<H_i$, by Lemmas 21 and 23,

$$\mathrm{Var}[\mathrm{Err}^i_h(y)]\le\frac{2\,\mathrm{R}_h(y)}{k_i}\le\frac{2^{-h+2}\cdot\mathrm{R}(y)}{k_i}.$$

We thus have

$$\begin{aligned}\mathrm{Var}[\mathrm{Err}(y)]&=\sum_{i=0}^{\ell}\sum_{h=0}^{H_i-1}2^{2h}\cdot\mathrm{Var}[\mathrm{Err}^i_h(y)]\\&\le\sum_{i=0}^{\ell}\sum_{h=0}^{H_i-1}2^{h+2}\cdot\frac{\mathrm{R}(y)}{k_i}\\&\le\sum_{i=0}^{\ell}2^{H_i+2}\cdot\frac{\mathrm{R}(y)}{k_i}\\&\le\sum_{i=0}^{\ell}2^{5}\cdot\frac{\mathrm{R}(y)^2}{k_i\cdot B_i}\\&\le\frac{\varepsilon^2\cdot\mathrm{R}(y)^2}{4\cdot\ln(1/\delta)}\cdot\sum_{i=0}^{\ell}\frac{1}{i^2}\\&\le\frac{\varepsilon^2\cdot\mathrm{R}(y)^2}{2\cdot\ln(1/\delta)}\end{aligned}$$

where the third inequality follows from (24), the penultimate inequality uses (23), and the last inequality holds as $\sum_{i=0}^{\ell}\frac{1}{i^2}<\pi^2/6<2$. Plugging this into the tail bound for sub-Gaussian variables (Fact 8) we get

$$\begin{aligned}\Pr[|\mathrm{Err}(y)|\ge\varepsilon\,\mathrm{R}(y)]&<2\exp\left(-\frac{\varepsilon^2\cdot\mathrm{R}(y)^2}{2\cdot\frac{\varepsilon^2\cdot\mathrm{R}(y)^2}{2\cdot\ln(1/\delta)}}\right)\\&=2\exp\left(-\ln\frac{1}{\delta}\right)=2\delta.\end{aligned}$$

This concludes the analysis of the failure probability.

Finally, we bound the size of the final sketch S. Let H be the index of the highest level in S. Observe that $H\le\lceil\log_2(n/B_0)\rceil$, since each item at level $h=\lceil\log_2(n/B_0)\rceil$ has weight 2^h, so there are fewer than B_0 items inserted to level h. Consequently, level H is never compacted (here, we also use that $B_0\le B_1\le\cdots\le B_\ell$). Hence, as $B_0\ge 1/\varepsilon$, there are $O(\log(\varepsilon n))$ levels in S. Each level has capacity $B_\ell=2\cdot k_\ell\cdot\lceil\log_2(N_\ell/k_\ell)\rceil$, so the total memory requirement of S is

$$O\left(\log(\varepsilon n)\cdot k_\ell\cdot\log\frac{N_\ell}{k_\ell}\right)=O\left(\log(\varepsilon n)\cdot\left\lceil\frac{\ell\cdot\hat{k}}{\sqrt{\log(N_\ell/\hat{k})}}\right\rceil\cdot\log\left(\frac{N_\ell}{k_\ell}\right)\right).$$

If $\varepsilon\le O\left(\log\log(\varepsilon n)\cdot\sqrt{\log\frac{1}{\delta}/\log(\varepsilon n)}\right)$, or equivalently $\ell\cdot\hat{k}\ge\Omega\left(\sqrt{\log(N_\ell/\hat{k})}\right)$, then the space bound is

$$\begin{aligned}&O\left(\log(\varepsilon n)\cdot\frac{\ell\cdot\hat{k}}{\sqrt{\log(N_\ell/\hat{k})}}\cdot\log\left(\frac{N_\ell}{k_\ell}\right)\right)\\&\le O\left(\log(\varepsilon n)\cdot\frac{\log\log(\varepsilon n)}{\varepsilon}\cdot\sqrt{\log\frac{1}{\delta}}\cdot\sqrt{\log(\varepsilon n)}\right),\end{aligned}$$

where we use that $\ell\le\log_2\log_2(\varepsilon n)$, $\log_2(N_\ell/k_\ell)\le O(\log_2(N_\ell/\hat{k}))$ (as $k_\ell\ge\hat{k}/\sqrt{\log_2(N_\ell/\hat{k})}$) and $\hat{k}\ge 1/\varepsilon$.

Otherwise, $\ell\cdot\hat{k}\le O\left(\sqrt{\log(N_\ell/\hat{k})}\right)$ and since $N_\ell\le n^2$, the size is bounded by $O(\log(\varepsilon n)\cdot\log(n))=O\left(\log^2(\varepsilon n)\right)$, also using $\log(n)\gg\log(1/\varepsilon)$ when $1/\varepsilon\le\hat{k}\le O\left(\sqrt{\log(N_\ell/\hat{k})}\right)$. □

C.5 A Tight Analysis of the Full Sketch

In this section, we complete the proof of full mergeability that matches our result in the streaming setting (Theorem 13). We stress that we assume no advance knowledge of n, the total size of the input. We now set the value of parameter $\hat{k}$. Namely, given the desired accuracy $0<\varepsilon\le 1$ and desired upper bound $0<\delta\le 0.5$ on failure probability, we let

$$\hat{k}=\frac{1}{\varepsilon}\cdot\sqrt{\ln\frac{1}{\delta}}. \tag{25}$$

Recall that by (15), $k_i=2^5\cdot\left\lceil\hat{k}/\sqrt{\log_2(N_i/\hat{k})}\right\rceil$ and $B_i=2\cdot k_i\cdot\lceil\log_2(N_i/k_i)+1\rceil$.

We assume that ε satisfies $\varepsilon\le 4/\sqrt[4]{2\log_2(n)}$; we remark that this is a very weak restriction put on ε as for all practical values of $n\le 2^{128}$ we have $\sqrt[4]{2\log_2(n)}\le 4$, in which case the assumption is implied by $\varepsilon\le 1$.

We need two lower bounds on the products of k_i and B_i for $0\le i\le\ell$. First, note that

$$\begin{aligned}k_\ell\cdot B_0&\ge k_\ell\cdot 2\cdot k_0\cdot\log_2(N_0/\hat{k})\\&\ge\frac{2^{11}\cdot\hat{k}^2}{\sqrt{\log_2(N_\ell/\hat{k})}}\\&\ge 2^{11}\cdot\frac{1}{\varepsilon^2}\cdot\ln\frac{1}{\delta}\cdot\frac{1}{\sqrt{\log_2(N_\ell/\hat{k})}}\\&\ge 2^7\cdot\ln\frac{1}{\delta},\end{aligned} \tag{26}$$

where we apply the assumption $\varepsilon\le 4/\sqrt[4]{2\log_2(n)}$ to get

$$2^4\cdot\varepsilon^{-2}/\sqrt{\log_2(N_\ell/\hat{k})}\ge 2^4\cdot\varepsilon^{-2}/\sqrt{\log_2(n^2)}\ge 1,$$

where the first inequality uses $N_\ell/\hat{k}\le n^2$ (because $\sqrt{N_\ell}=N_{\ell-1}<n$).

Second, using a similar calculation as in Claim 6, we show a lower bound on $k_i\cdot B_i$.

Claim 26. *Parameters k_i and B_i set according to* (15) *and based on $\hat{k}$ as in* (25) *satisfy*

$$k_i \cdot B_i \geq 2^{10} \cdot \frac{1}{\varepsilon^2} \cdot \ln \frac{1}{\delta} . \quad (27)$$

Proof. We first need to relate $\log_2(N_i/k_i)+1$ (used to define B_i) and $\log_2(N_i/\hat{k})$ (that appears in the definition of k_i). As $k_i \leq 2^5 \cdot \hat{k}$, it holds that $\log_2(N_i/k_i) + 1 \geq \log_2(N_i/\hat{k}) - 4 \geq \log_2(N_i/\hat{k})/2$, where we use that $N_i \geq N_0 \geq 2^8 \cdot \hat{k}$, so $\log_2(N_i/\hat{k}) \geq 8$. Using this, we bound $k_i \cdot B_i$ as follows:

$$\begin{aligned} k_i \cdot B_i &= 2 \cdot k_i^2 \cdot \left\lceil \log_2 \frac{N_i}{k_i} + 1 \right\rceil \\ &\geq 2 \cdot 2^{10} \cdot \frac{1}{\varepsilon^2} \cdot \frac{\ln \frac{1}{\delta}}{\log_2(N_i/\hat{k})} \cdot \frac{\log_2(N_i/\hat{k})}{2} \\ &= 2^{10} \cdot \frac{1}{\varepsilon^2} \cdot \ln \frac{1}{\delta} . \end{aligned}$$

□

For any $0 \leq i \leq \ell$, we define $H_i(y)$ be the minimal h for which $2^{-h+1} \mathrm{R}(y) \leq B_i/2$. As y is fixed, we write H_i rather than $H_i(y)$ for brevity. In particular, by considering $h = H_i - 1$ (assuming $H_i > 0$), it can be seen that $2^{2-H_i} \mathrm{R}(y) \geq B_i/2$, or equivalently

$$2^{H_i} \leq 2^3 \cdot \mathrm{R}(y)/B_i . \quad (28)$$

As increasing i by one increases B_i, we have $H_0 \geq H_1 \geq \cdots \geq H_\ell$. We show below that no important item (i.e., one smaller than or equal to y) can ever reach level $H_0 + 1$.

Lemma 27. *Assuming $H_0 > 0$, with probability at least $1 - \delta$ it holds that $\mathrm{R}_h(y) \leq 2^{-h+1} \mathrm{R}(y)$ for any $h \leq H_0$.*

Proof. The proof is similar to that of Lemma 9, except that we need to deal with parameters k and B changing over time. We show by induction on $0 \leq h \leq H_0$ that $\mathrm{R}_h(y) \leq 2^{-h+1} \mathrm{R}(y)$ with probability at least $1 - \delta \cdot 2^{h-H_0-1}$, conditioned on $\mathrm{R}_{h'}(y) \leq 2^{-h'+1} \mathrm{R}(y)$ for any $h' < h$. The base case holds by $\mathrm{R}_0(y) = \mathrm{R}(y)$.

Consider $0 < h \leq H_0$, and recall that $m_{h'}$ denotes the number of important compactions at level h' over all merge operations represented in the merge tree T. As in the proof of Lemma 9,

$$\Pr[\mathrm{R}_h(y) > 2^{-h+1} \mathrm{R}(y)] \leq \Pr[Z_h > 2^{-h} \mathrm{R}(y)],$$

where $Z_h = \sum_{h'=0}^{h-1} 2^{-h+h'} \cdot \mathrm{Binomial}(m_{h'})$ is a zero-mean sub-Gaussian random variable. To bound the variance of Z_h, first note that for any $h' < h$, we have that $m_{h'} \leq 2\,\mathrm{R}_{h'}(y)/k_\ell \leq 2^{-h'+2} \mathrm{R}(y)/k_\ell$ using Corollary 2 and the assumption that $\mathrm{R}_{h'}(y) \leq 2^{-h'+1} \mathrm{R}(y)$.

As $\mathrm{Var}[\mathrm{Binomial}(n)] = n$, the variance of Z_h is

$$\begin{aligned} \mathrm{Var}[Z_h] &\leq \sum_{h'=0}^{h-1} 2^{-2h+2h'} \cdot m_{h'} \\ &\leq \sum_{h'=0}^{h-1} 2^{-2h+2h'} \cdot \frac{2^{-h'+2} \mathrm{R}(y)}{k_\ell} \\ &= \sum_{h'=0}^{h-1} \frac{2^{-2h+h'+2} \mathrm{R}(y)}{k_\ell} \\ &\leq \frac{2^{-h+2} \cdot \mathrm{R}(y)}{k_\ell} . \end{aligned}$$

To bound $\Pr[Z_h > 2^{-h} \mathrm{R}(y)]$, we apply the tail bound for sub-Gaussian variables (Fact 8) to get

$$\begin{aligned} \Pr[Z_h > 2^{-h} \mathrm{R}(y)] &< \exp\left(-\frac{2^{-2h} \cdot \mathrm{R}(y)^2}{2 \cdot (2^{-h+2} \cdot \mathrm{R}(y)/k_\ell)}\right) \\ &= \exp\left(-2^{-h-3} \cdot \mathrm{R}(y) \cdot k_\ell\right) \\ &= \exp\left(-2^{-h+H_0-6} \cdot 2^{3-H_0} \mathrm{R}(y) \cdot k_\ell\right) \\ &\leq \exp\left(-2^{-h+H_0-6} \cdot B_0 \cdot k_\ell\right) \\ &\leq \exp\left(-2^{-h+H_0+1} \cdot \ln \frac{1}{\delta}\right) \\ &= \delta^{2^{-h+H_0+1}} \leq \delta \cdot 2^{-H_0+h-1} , \end{aligned}$$

where the second inequality uses $2^{3-H_0} \mathrm{R}(y) \geq B_0$ by (28), the third inequality follows from (26), and the last inequality uses $\delta \leq 0.5$. Hence, taking the union bound over levels $h \leq H_0$, with probability at least $1 - \delta$ it holds that $\mathrm{R}_h(y) \leq 2^{-h+1} \mathrm{R}(y)$ for any $h \leq H_0$. □

As a corollary, we obtain a bound on the highest level with a compaction removing important items from the level-h buffer (no matter whether such a compaction is important or not).

Lemma 28. *Conditioned on the bounds in Lemma 27 holding, for any $0 \leq i \leq \ell$, no compaction involving important items occurs at level H_i or above during any merge procedure represented by any i-node in the merge tree T.*

Proof. By Lemma 27, $\mathrm{R}_{H_i}(y) \leq 2^{-H_i+1} \mathrm{R}(y) \leq B_i/2$, where the second inequality follows from the definition of H_i. Hence, no important item is ever compacted during merge operations represented by i-nodes when the buffer size is B_i. □

Next, we prove an initial bound on the estimated rank of y, namely, that it is at most $2\,\mathrm{R}(y)$ with high probability (we do not need a lower bound). Such an initial bound will in turn be used within the proof of the final, more refined bound on the variance of $\mathrm{Err}(y)$.

Lemma 29. *Conditioned on the bounds in Lemma 27 holding, with probability at least $1 - \delta$ it holds that $\hat{\mathrm{R}}(y) \leq 2\,\mathrm{R}(y)$, or equivalently that $\mathrm{Err}(y) \leq \mathrm{R}(y)$.*

Proof. Recall that $\mathrm{Err}(y)$ is a zero-mean sub-Gaussian random variable. Lemma 28 implies that there is no important compaction at level H_0 or above, so $\mathrm{Err}_h(y) = 0$ for any $h \geq H_0$. We bound the variance for levels $h < H_0$ as follows:

$$\begin{aligned} \mathrm{Var}[\mathrm{Err}(y)] &= \sum_{h=0}^{H_0-1} 2^{2h} \,\mathrm{Var}[\mathrm{Err}_h(y)] \\ &\leq \sum_{h=0}^{H_0-1} 2^{2h} \cdot \frac{2\,\mathrm{R}_h(y)}{k_\ell} \\ &\leq \sum_{h=0}^{H_0-1} 2^{h+2} \cdot \frac{\mathrm{R}(y)}{k_\ell} \end{aligned}$$

$$\leq 2^{H_0+2} \cdot \frac{\mathrm{R}(y)}{k_\ell}$$
$$\leq 2^5 \cdot \frac{\mathrm{R}(y)^2}{k_\ell \cdot B_0} \leq \frac{\mathrm{R}(y)^2}{2\ln(1/\delta)}\,,$$

where the first inequality is by Corollary 2, the second inequality is due to Lemma 27, the penultimate inequality is by (28), and the last inequality uses (26).

It remains to apply the tail bound for sub-Gaussian variables (Fact 8) to obtain

$$\Pr\left[\mathrm{Err}(y) > \mathrm{R}(y)\right] < \exp\left(-\frac{\mathrm{R}(y)^2}{2\cdot \mathrm{R}(y)^2/(2\ln(1/\delta))}\right) = \exp\left(-\ln\frac{1}{\delta}\right) = \delta\,.$$

□

Consider level h. Recall from Section C.3 that Q_h is the set of nodes t such that (i) t represents a level-h compaction that involves important items (this compaction may or may not be important), and (ii) there is no node on the t-to-root path (except for t) that represents a level-h compaction involving important items. Note that an important item that remains in the level-h buffer represented by a node in Q_h (after performing the compaction operation) is never removed from the level-h buffer, by the definition of Q_h. For $0 \leq i \leq \ell$, let Q^i_h be the set of i-nodes in Q_h and let $q^i_h = |Q^i_h|$.

Note that $q^i_h = 0$ for $h \geq H_0$ by Lemma 28. Now we observe that values q^i_h for $i = 0, \dots, \ell$ give upper bounds on the number of important items at level h. This follows from the fact that the level-h buffer represented by a node in Q^i_h contains at most B_i items.

Observation 30. *For any $h \geq 0$ and $0 \leq g \leq \ell$, the level-h buffers of the sketches represented by nodes in Q^i_h for some $i \geq g$ contain at most $\sum_{i=g}^{\ell} q^i_h \cdot B_i$ important items in total (after performing compaction operations represented by these nodes).*

Next, we show that values q^i_h can as well be used to lower bound the number of important items at level h in the final sketch. Combined with Lemma 29, this will give us a useful bound on $\sum_{h\geq 0}\sum_{i=0}^{\ell} 2^h \cdot q^i_h \cdot B_i$ at the very end of the analysis. Intuitively, the observation also implies that the q^i_h values cannot be too big, namely, $q^i_h \leq 2B_\ell/B_i = O(\log \varepsilon n)$ as there are at most B_ℓ items in the level-h buffer of the final sketch.

In the observation, we also take into account items added to level h from compactions (at level $h-1$ if $h > 0$) that are *not* represented by a node in the subtree of a node in Q_h. Namely, for $h > 0$ and any $0 \leq i \leq \ell$, let z^i_h be the number of items added to level h during merge operations represented by i-nodes that are not in the subtree of a node in Q_h. For $h = 0$, we define $z^i_0 = 0$ for any i.

Observation 31. *For any level h, the final sketch represented by the root of T contains at least $\sum_{i=0}^{\ell} q^i_h \cdot B_i/2 + z^i_h$ important items at level h.*

Proof. Consider an i-node $t \in Q^i_h$ and the level-h buffer represented by t. As the level-h compaction represented by t removes one or more important items and as t is an i-node, there must be at least $B_i/2$ important items in the level-h buffer that remain there after the compaction operation is done. Furthermore, by condition (ii) in the definition of Q_h, these $B_i/2$ important items are not removed from the level-h buffer and the sets of these $B_i/2$ important items for two nodes $t, t' \in Q_h$ are disjoint. Finally, the z^i_h items added to level h during merge operations represented by i-nodes that are not in the subtree of a node in Q_h are disjoint (w.r.t index i) and distinct from items in the buffers of nodes in Q_h, which shows the claim. □

The following technical lemma bounds the variance on each level in a somewhat different way than in the streaming setting (Section 4). The idea is to bound the variance in terms of the q^i_h values so that we can then use Observation 31. To this end, we first use Observation 30 to bound $\mathrm{R}_h(y)$ in terms of the q^i_h values, using the following observation: For each important item at level $h+1$, there are roughly two important items removed from level h. Here, "roughly" refers to the fact that each compaction operation that promotes b important items removes at most $2b+1$ important items from the level-h buffer. To mitigate the $+1$ for each compaction operation, we use factor 3 in the formal proof. Applying this observation together with Observation 30, we show by an induction on h that $\mathrm{R}^{\geq 0}_h(y) \leq \sum_{i=0}^{\ell}\sum_{h'\geq h} 2\cdot 3^{h'-h}\cdot (q^i_{h'}\cdot B_i + z^i_{h'})$. Recall that $\mathrm{R}^{\geq a}_h(y)$ is the number of important items that are either removed from level h during a compaction represented by an i-node for $a \leq i \leq \ell$, or remain at the level-h buffer represented by a node $t \in Q^i_h$ for $a \leq i \leq \ell$ (after the compaction operation represented by t is done).

Then we apply Lemma 21 to get our variance bound, which however brings additional technical difficulties. To overcome them, we use a careful proof by induction over $g \in [0, \ell]$.

Lemma 32. *Conditioned on the bounds in Lemma 27 holding, for any $h \geq 0$, it holds that*

$$\mathrm{Var}[\mathrm{Err}_h(y)] \leq \sum_{i=0}^{\ell}\sum_{h'\geq h} \frac{4\cdot 3^{h'-h}\cdot (q^i_{h'}\cdot B_i + z^i_{h'})}{k_i}\,. \tag{29}$$

Proof. As outlined above, we first bound $\mathrm{R}^{\geq g}_h(y)$ for any $0 \leq g \leq \ell$ and in particular, we prove by a "backward" induction on $h = H, H-1, \dots, 0$ that the following inequality holds for any $0 \leq g \leq \ell$:

$$\mathrm{R}^{\geq g}_h(y) \leq \sum_{i=g}^{\ell}\left(\sum_{h'\geq h+1}\left(2\cdot 3^{h'-h}\cdot (q^i_{h'}\cdot B_i + z^i_{h'})\right) + 2\cdot q^i_h \cdot B_i\right)\,. \tag{30}$$

For any $h \geq H_0$, by Lemma 28 we have that $q^i_h = 0$ and that no important item is removed from level h, thus $\mathrm{R}^{\geq g}_h(y) = 0$ for any g. Consequently, inequalities (30) holds trivially for $h \geq H_0$ and any g, which establishes the base case.

Consider $h < H_0$ and suppose that (30) holds for $h+1$, i.e., we have that

$$\mathrm{R}^{\geq g}_{h+1}(y) \leq \sum_{i=g}^{\ell}\left(\sum_{h'\geq h+2}\left(2\cdot 3^{h'-h-1}\cdot (q^i_{h'}\cdot B_i + z^i_{h'})\right) + 2\cdot q^i_{h+1} \cdot B_i\right)\,. \tag{31}$$

To show (30), we first bound the number of important items removed from level h in terms of $\mathrm{R}^{\geq g}_{h+1}(y)$. For brevity, let $z^{\geq g}_{h+1} = \sum_{i=g}^{\ell} z^i_{h+1}$. Note that there are at most $\mathrm{R}^{\geq g}_{h+1}(y) + z^{\geq g}_{h+1}$ important items added to level $h+1$ during compactions represented by i-nodes for some $i \in [g, \ell]$, since each such important item either gets removed from level $h+1$ or remains in the level-$(h+1)$ buffer represented by a node in Q^i_{h+1} for some $i \in [g, \ell]$ or is added to level $h+1$ during a merge operation represented by an i-node t for $i \in [g, \ell]$ such that t is not in the subtree of a node in Q_h. Further, observe that each compaction that adds b important items to level $h+1$ removes $2b+1$ items from from the level-h buffer if the compaction is important, and if the compaction is not important, then it removes $2b$ items from the level-h buffer. The number of important compactions represented by i-nodes for some $i \in [g, \ell]$ is at most $\mathrm{R}^{\geq g}_h(y)/5$ by Lemma 21 with $a = g$ (where we use $k_i \geq 10$ for any i). Thus the number of important items removed from level h during compactions represented by i-nodes for $i \in [g, \ell]$ is upper bounded by $2\,\mathrm{R}^{\geq g}_{h+1}(y) + 2z^{\geq g}_{h+1} + (\mathrm{R}^{\geq g}_h(y)/5)$.

By Observation 30, at most $\sum_{i=g}^{\ell} q^i_h \cdot B_i$ important items remain at the level-h buffers of the sketches represented by nodes in Q^i_h for some $i \geq g$. We thus have that

$$\mathrm{R}^{\geq g}_h(y) \leq 2\,\mathrm{R}^{\geq g}_{h+1}(y) + 2z^{\geq g}_{h+1} + (\mathrm{R}^{\geq g}_h(y)/5) + \sum_{i=g}^{\ell} q^i_h \cdot B_i .$$

After subtracting $\mathrm{R}^{\geq g}_h(y)/5$ from both sides of this inequality, and then multiplying both sides of the inequality by $5/4$, we get

$$\begin{aligned}
\mathrm{R}^{\geq g}_h(y) &\leq \frac{5}{2} \cdot \mathrm{R}^{\geq g}_{h+1}(y) + \frac{5}{2} \cdot z^{\geq g}_{h+1} + \frac{5}{4} \cdot \sum_{i=g}^{\ell} q^i_h \cdot B_i \\
&\leq \frac{5}{2} \cdot \left(\sum_{i=g}^{\ell} \left(\sum_{h' \geq h+2} \left(2 \cdot 3^{h'-h-1} \cdot (q^i_{h'} \cdot B_i + z^i_{h'}) \right) \right. \right. \\
&\qquad \left. \left. + 2 \cdot q^i_{h+1} \cdot B_i \right) \right) + \frac{5}{2} \cdot z^{\geq g}_{h+1} + \frac{5}{4} \cdot \sum_{i=g}^{\ell} q^i_h \cdot B_i \\
&\leq \sum_{i=g}^{\ell} \left(\sum_{h' \geq h+1} \left(2 \cdot 3^{h'-h} \cdot (q^i_{h'} \cdot B_i + z^i_{h'}) \right) + 2 \cdot q^i_h \cdot B_i \right),
\end{aligned}$$

where the second inequality uses the induction hypothesis (31). Thus, (30) holds.

Using $z^i_h \geq 0$, we simplify (30) and get

$$\mathrm{R}^{\geq g}_h(y) \leq \sum_{i=g}^{\ell} \sum_{h' \geq h} \left(2 \cdot 3^{h'-h} \cdot (q^i_{h'} \cdot B_i + z^i_{h'}) \right). \tag{32}$$

Finally, we bound the variance. Recall from Section C.3 that m^i_h is the number of important compaction operations at level h represented by i-nodes. Note that $\mathrm{Var}[\mathrm{Err}_h(y)] = \sum_{i=0}^{\ell} m^i_h$. We prove by a "backward" induction on $g = \ell, \ell - 1, \ldots, 0$ that the following inequality holds for any $h \geq 0$:

$$\sum_{i=g}^{\ell} m^i_h \leq \sum_{i=g}^{\ell} \sum_{h' \geq h} \frac{4 \cdot 3^{h'-h} \cdot (q^i_{h'} \cdot B_i + z^i_{h'})}{k_i}. \tag{33}$$

Note that (33) for $g = 0$ gives (29) and that we may suppose that $h < H_0$ as $\sum_{i=g}^{\ell} m^i_h = 0$ for $h \geq H_0$ and any g by Lemma 28. Consider $0 \leq g \leq \ell$ and suppose that for any $g' > g$ (in the case $g < \ell$), we have that

$$\sum_{i=g'}^{\ell} m^i_h \leq \sum_{i=g'}^{\ell} \sum_{h' \geq h} \frac{4 \cdot 3^{h'-h} \cdot (q^i_{h'} \cdot B_i + z^i_{h'})}{k_i}. \tag{34}$$

To show (33), we use Lemma 21 with $a = g$ to get $\sum_{i=g}^{\ell} m^i_h \cdot k_i \leq 2\,\mathrm{R}^{\geq g}_h(y)$. We divide this inequality by k_g and use (32) to get

$$\sum_{i=g}^{\ell} \frac{k_i}{k_g} \cdot m^i_h \leq \sum_{i=g}^{\ell} \sum_{h' \geq h} \frac{4 \cdot 3^{h'-h} \cdot (q^i_{h'} \cdot B_i + z^i_{h'})}{k_g}.$$

For every $g' > g$, we add inequality (34) (that holds by the induction hypothesis) multiplied by $(k_{g'-1} - k_{g'})/k_g$ (which is non-negative as $k_{g'-1} \geq k_{g'}$) to obtain

$$\begin{aligned}
&\sum_{i=g}^{\ell} \left(\frac{k_i}{k_g} + \sum_{g'=g+1}^{i} \frac{k_{g'-1} - k_{g'}}{k_g} \right) \cdot m^i_h \\
&\quad \leq \sum_{i=g}^{\ell} \left(\frac{k_i}{k_g \cdot k_i} + \sum_{g'=g+1}^{i} \frac{k_{g'-1} - k_{g'}}{k_g \cdot k_i} \right) \\
&\qquad \cdot \sum_{h' \geq h} 4 \cdot 3^{h_j - h} \cdot (q^i_{h'} \cdot B_i + z^i_{h'}).
\end{aligned} \tag{35}$$

Note that the sum of fractions of k_i's on the RHS of (35) equals $1/k_i$ for any i, and similarly the sum of fractions of k_i's on the LHS of (35) equals 1 for any i, so the LHS equals $\sum_{i=g}^{\ell} m^i_h$. This shows (33). □

Finally, we have all ingredients needed to show that we can match the streaming result of Theorem 13 even when creating the sketch using an arbitrary sequence of merge operations without any advance knowledge about the total size of the input. That is, we now prove the full mergeability claim of Theorem 1, which we restate for convenience.

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 $\hat{\mathrm{R}}(y)$ of $\mathrm{R}(y)$ such that*

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

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.

Proof. We condition on the bounds from Lemmas 27 and 29, which together hold with probability at least $1 - 2\delta$. We bound the variance using Lemma 32 as follows:

$$\mathrm{Var}[\mathrm{Err}(y)] = \sum_{h \geq 0} 2^{2h} \cdot \mathrm{Var}[\mathrm{Err}_h(y)]$$

$$
\begin{aligned}
&\le \sum_{h\ge 0} 2^{2h} \cdot \sum_{i=0}^{\ell} \sum_{h'\ge h} \frac{4\cdot 3^{h'-h}\cdot (q^i_{h'}\cdot B_i + z^i_{h'})}{k_i} \\
&= \sum_{i=0}^{\ell} \sum_{h'\ge 0} \sum_{h=0}^{h'} 2^{2h} \frac{4\cdot 3^{h'-h}\cdot (q^i_{h'}\cdot B_i + z^i_{h'})}{k_i} \\
&\le \sum_{i=0}^{\ell} \sum_{h'\ge 0} \frac{2^{2h'+4}\cdot (q^i_{h'}\cdot B_i + z^i_{h'})}{k_i} \\
&\le \sum_{i=0}^{\ell} \sum_{h'=0}^{H_i} \frac{2^{h'+7}\cdot (q^i_{h'}\cdot B_i + z^i_{h'})\cdot \mathrm{R}(y)}{k_i\cdot B_i} \\
&\le \frac{\varepsilon^2\cdot \mathrm{R}(y)}{8\ln(1/\delta)} \cdot \sum_{i=0}^{\ell} \sum_{h'=0}^{H_i} 2^{h'}\cdot (q^i_{h'}\cdot B_i + z^i_{h'}),
\end{aligned}
$$

where the second inequality follows from

$$
\begin{aligned}
\sum_{h=0}^{h'} 2^{2h}\cdot 4\cdot 3^{h'-h} &= 4\cdot 3^{h'}\cdot \sum_{h=0}^{h'} \left(\frac{4}{3}\right)^h \\
&\le 4\cdot 3^{h'}\cdot 3\cdot \left(\frac{4}{3}\right)^{h'+1} = 4^{h'+2} = 2^{2h'+4},
\end{aligned}
$$

the third inequality uses $q^i_{h'} = 0$ and $z^i_{h'} = 0$ for $h' > H_i$ by Lemma 28 and $2^{H_i} \le 2^3 \cdot \mathrm{R}(y)/B_i$ by (28), and the fourth inequality follows from the bound on $k_i \cdot B_i$ in (27).

By Observation 31, there are at least $\sum_{i=0}^{\ell} q^i_h \cdot B_i/2 + z^i_h$ important items stored in the level-h buffer of the final sketch, thus the estimated rank of y satisfies $\hat{\mathrm{R}}(y) \ge \sum_{h=0}^{H_0} \sum_{i=0}^{\ell} 2^h \cdot q^i_h \cdot B_i/2 + z^i_h$. On the other hand, conditioned on Lemma 29, the estimated rank of y is at most $2\,\mathrm{R}(y)$, which gives us

$$
\sum_{i=0}^{\ell} \sum_{h'=0}^{H_i} 2^{h'}\cdot (q^i_{h'}\cdot B_i + z^i_{h'}) \le 2\hat{\mathrm{R}}(y) \le 4\,\mathrm{R}(y)\,.
$$

Using this inequality, we obtain our final variance bound:

$$
\mathrm{Var}[\mathrm{Err}(y)] \le \frac{\varepsilon^2\cdot \mathrm{R}(y)^2}{2\ln(1/\delta)}\,.
$$

Plugging this into the tail bound for sub-Gaussian variables (Fact 8) we finally conclude that

$$
\begin{aligned}
\Pr\left[|\,\mathrm{Err}(y)| > \varepsilon\,\mathrm{R}(y)\right] &< 2\exp\left(-\frac{\varepsilon^2\cdot \mathrm{R}(y)^2}{2\cdot\varepsilon^2\cdot \mathrm{R}(y)^2/(2\ln(1/\delta))}\right) \\
&= 2\exp\left(-\ln\frac{1}{\delta}\right) = 2\delta\,.
\end{aligned}
$$

This completes the calculation of the failure probability.

Finally, we bound the size of the final sketch S. Let H be the index of the highest level in S. Observe that $H \le \lceil \log_2(n/B_0)\rceil$, since each item at level $h = \lceil \log_2(n/B_0)\rceil$ has weight 2^h, so there are fewer than B_0 items inserted to level h. Consequently, level H is never compacted (here, we also use that $B_0 \le B_1 \le \cdots \le B_\ell$). Hence, as $B_0 \ge 1/\varepsilon$, there are $O(\log(\varepsilon n))$ levels in S. Each level has capacity $B_\ell = 2\cdot k_\ell \cdot \lceil \log_2(N_\ell/k_\ell) + 1\rceil$, so the total memory requirement of S is

$$
O\left(\log(\varepsilon n)\cdot k_\ell \cdot \log\left(\frac{N_\ell}{k_\ell}\right)\right) = O\left(\log(\varepsilon n)\cdot \left\lceil \frac{\hat{k}}{\sqrt{\log(N_\ell/\hat{k})}} \right\rceil \cdot \log\left(\frac{N_\ell}{k_\ell}\right)\right).
$$

If $\varepsilon \le 4\cdot \sqrt{\ln\frac{1}{\delta}/\log_2(\varepsilon n)}$, or equivalently $\hat{k} \ge \Omega\left(\sqrt{\log(N_\ell/\hat{k})}\right)$, then the space bound is

$$
\begin{aligned}
&O\left(\log(\varepsilon n)\cdot \frac{\hat{k}}{\sqrt{\log(N_\ell/\hat{k})}}\cdot \log\left(\frac{N_\ell}{k_\ell}\right)\right) \\
&\le O\left(\log(\varepsilon n)\cdot \frac{1}{\varepsilon}\cdot \sqrt{\log\frac{1}{\delta}}\cdot \sqrt{\log(\varepsilon n)}\right),
\end{aligned}
$$

where we use that $\log_2(N_\ell/k_\ell) \le O(\log(N_\ell/\hat{k}))$ (as $k_\ell \ge \hat{k}/\sqrt{\log_2(N_\ell/\hat{k})}$) and $\hat{k} \ge 1/\varepsilon$.

Otherwise, $\hat{k} \le O\left(\sqrt{\log(N_\ell/\hat{k})}\right)$ and since $N_\ell \le n^2$, the size is bounded by $O(\log(\varepsilon n)\cdot \log(n)) = O\left(\log^2(\varepsilon n)\right)$, also using $\log(n) \gg \log(1/\varepsilon)$ when $1/\varepsilon \le \hat{k} \le O\left(\sqrt{\log(N_\ell/\hat{k})}\right)$. □

D ANALYSIS WITH EXTREMELY SMALL FAILURE PROBABILITY

In this section, we provide a somewhat different analysis of our algorithm, which yields an improved space bound for extremely small values of δ, at the cost of a worse dependency on n. In particular, we show a space upper bound of $O(\varepsilon^{-1}\cdot \log^2(\varepsilon n)\cdot \log\log(1/\delta))$ for any $\delta > 0$. For simplicity, we only give the subsequent analysis in the streaming setting, although we conjecture that an appropriately adjusted analysis in Appendix C would yield the same bound under arbitrary merge operations. We further assume foreknowledge of (a polynomial bound on) n, the stream length; this assumption can be removed in a similar fashion to Section 5. As a byproduct, we show at the end of this appendix that this result implies a deterministic space upper bound of $O(\varepsilon^{-1}\cdot \log^3(\varepsilon n))$ for answering rank queries with multiplicative error ε, thus matching the state-of-the-art result of Zhang and Wang [22].

To this end, we use Algorithm 2 with a different setting of k, namely,

$$
k = 2^4 \cdot \left\lceil \frac{1}{\varepsilon}\cdot \log_2 \ln\frac{1}{\delta} \right\rceil. \tag{36}
$$

We remark that, unlike in Section 4, the value of k does not depend on n directly (only possibly indirectly if δ or ε is set based on n). Note that the analysis of a single relative-compactor in Section 3 still applies and in particular, there are at most $\mathrm{R}_h(y)/k$ important steps at each level h by Lemma 5.

We enhance the analysis for a fixed item y of Section 4. The crucial trick to improve the dependency on δ from $\sqrt{\ln(1/\delta)}$ to $\log_2\ln(1/\delta)$ is to analyze the sketch using Chernoff bounds only below a certain level $H'(y)$ and provide deterministic bounds for levels $H'(y) \le h < H(y)$. This idea was first used by Karnin et al. [12] to get their optimal result for the additive error guarantee. We define

$$
H'(y) = \max\left(0, H(y) - \lceil \log_2\ln(1/\delta)\rceil\right);
$$

here $H(y)$ is defined as in Section 4 as the minimal h for which $2^{2-h}\,\mathrm{R}(y) \le B/2$. Next, we provide modified rank bounds.

Lemma 33. *Assuming $H(y) > 0$, for any $h < H(y)$ it holds that $\mathrm{R}_h(y) \le 2^{-h+2}\,\mathrm{R}(y)$ with probability at least $1-\delta$.*

Proof. We first show by induction on $0 \le h < H'(y)$ that $\mathrm{R}_h(y) \le 2^{-h+1}\,\mathrm{R}(y)$ with probability at least $1-\delta\cdot 2^{h-H'(y)}$, conditioned on $\mathrm{R}_\ell(y) \le 2^{-\ell+1}\,\mathrm{R}(y)$ for any $\ell < h$. This part of the proof is similar to that of Lemma 9. The base case holds by $\mathrm{R}_0(y) = \mathrm{R}(y)$.

Consider $0 < h < H'(y)$. As in Lemma 9,

$$\Pr[\mathrm{R}_h(y) > 2^{-h+1}\,\mathrm{R}(y)] \le \Pr[Z_h > 2^{-h}\,\mathrm{R}(y)],$$

where Z_h is a zero-mean sub-Gaussian variable with variance at most $\mathrm{Var}[Z_h] \le 2^{-h+1}\cdot \mathrm{R}(y)/k$. We apply the tail bound for sub-Gaussian variables (Fact 8) on Z_h to get

$$\begin{aligned}
\Pr[Z_h > 2^{-h}\,\mathrm{R}(y)] &< \exp\left(-\frac{2^{-2h}\cdot \mathrm{R}(y)^2}{2\cdot(2^{-h+1}\cdot \mathrm{R}(y)/k)}\right)\\
&= \exp\left(-2^{-h-2}\cdot \mathrm{R}(y)\cdot k\right)\\
&= \exp\left(-2^{-h+H'(y)-6}\cdot 2^{H(y)-H'(y)}\cdot 2^{4-H(y)}\,\mathrm{R}(y)\cdot k\right)\\
&\le \exp\left(-2^{-h+H'(y)-6}\cdot 2^{H(y)-H'(y)}\cdot B\cdot k\right)\\
&\le \exp\left(-2^{-h+H'(y)}\cdot \ln\frac{1}{\delta}\right) = \delta^{2^{-H'(y)+h}} \le \delta\cdot 2^{-H'(y)+h},
\end{aligned}$$

where the second inequality uses $2^{4-H(y)}\,\mathrm{R}(y) \ge B$ (by the definition of $H(y)$) and the third inequality follows from $2^{H(y)-H'(y)} \ge \ln\frac{1}{\delta}$ and $B\cdot k \ge k^2 \ge 2^6$. This concludes the proof by induction. Taking the union bound over levels $h < H'(y)$, it holds that $\mathrm{R}_h(y) \le 2^{-h+1}\,\mathrm{R}(y)$ for any $h < H'(y)$ with probability at least $1-\delta$.

Finally, consider level $h \ge H'(y)$ and condition on $\mathrm{R}_{H'(y)-1}(y) \le 2^{-H'(y)+2}\,\mathrm{R}(y)$. (In the case $H'(y) = 0$, we have $\mathrm{R}_0(y) = \mathrm{R}(y)$.) Note that for any $\ell > 0$, it holds that $\mathrm{R}_\ell(y) \le \frac{1}{2}\cdot(1+1/k)\cdot \mathrm{R}_{\ell-1}(y)$. Indeed, $\mathrm{R}_\ell(y) \le \frac{1}{2}\cdot(\mathrm{R}_{\ell-1}(y) + \mathrm{Binomial}(m_{\ell-1}))$ (see Equation 9) and $\mathrm{Binomial}(m_{\ell-1}) \le m_{\ell-1} \le \mathrm{R}_{\ell-1}(y)/k$ by Lemma 5. That is, regardless of the outcome of the random choices, we always obtain this weaker bound on the rank of an item.

By using this deterministic bound for levels $H'(y) \le \ell \le h$, we get

$$\begin{aligned}
\mathrm{R}_h(y) &\le 2^{-h+H'(y)-1}\cdot\left(1+\frac{1}{k}\right)^{h-H'(y)+1}\cdot \mathrm{R}_{H'(y)-1}(y)\\
&\le 2^{-h+H'(y)-1}\cdot\left(1+\frac{1}{k}\right)^{0.5\cdot k}\cdot 2^{-H'(y)+2}\cdot \mathrm{R}(y)\\
&\le 2^{-h+2}\cdot \mathrm{R}(y),
\end{aligned}$$

where in the second inequality, we use $h-H'(y)+1 \le 0.5\cdot k$ (which follows from $h < H(y)$ and $H(y)-H'(y) \le \log_2 \ln\frac{1}{\delta} \le 0.5\cdot k$) together with the bound on $\mathrm{R}_{H'(y)-1}(y)$, and the last inequality uses the fact that $(1+1/k)^{0.5\cdot k} \le \sqrt{e} < 2$. □

We now state the main result of this section, which proves Theorem 2 assuming an advance knowledge of (a polynomial upper bound on) the stream length n. This assumption can be removed using the technique described in Section 5.

Theorem 34. *Assume that (a polynomial upper bound on) the stream length n is known in advance. For any parameters $0 < \delta \le 0.5$ and $0 < \varepsilon \le 1$, let k be set as in (36). 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*

$$\Pr[|\mathrm{Err}(y)| \ge \varepsilon\,\mathrm{R}(y)] < 3\delta\,.$$

The overall memory used by the algorithm is $O\left(\varepsilon^{-1}\cdot\log^2(\varepsilon n)\cdot\log\log(1/\delta)\right)$.

Proof. We condition on the bounds in Lemma 33, which together hold with probability at least $1-\delta$. We split $\mathrm{Err}(y)$, the error of the rank estimate for y, into two parts:

$$\mathrm{Err}'(y) = \sum_{h=0}^{H'(y)-1} 2^h\cdot \mathrm{Err}_h(y) \quad\text{and}\quad \mathrm{Err}''(y) = \sum_{h=H'(y)}^{H} 2^h\cdot \mathrm{Err}_h(y)\,.$$

Note that $\mathrm{Err}(y) = \mathrm{Err}'(y) + \mathrm{Err}''(y)$; we bound both these parts by $\frac{1}{2}\varepsilon\,\mathrm{R}(y)$ w.h.p., starting with $\mathrm{Err}'(y)$. If $H'(y) = 0$, then clearly $\mathrm{Err}'(y) = 0$. Otherwise, we analyze the variance of the zero-mean sub-Gaussian variable $\mathrm{Err}'(y)$

$$\begin{aligned}
\mathrm{Var}[\mathrm{Err}'(y)] &= \sum_{h=0}^{H'(y)-1} 2^{2h}\cdot \mathrm{Var}[\mathrm{Err}_h(y)]\\
&\le \sum_{h=0}^{H'(y)-1} 2^{2h}\cdot\frac{\mathrm{R}_h(y)}{k}\\
&\le \sum_{h=0}^{H'(y)-1} 2^{2h}\cdot\frac{2^{-h+2}\,\mathrm{R}(y)}{k}\\
&\le 2^{H'(y)+2}\cdot\frac{\mathrm{R}(y)}{k}\\
&= 2^{H'(y)-H(y)+2}\cdot 2^{H(y)}\cdot\frac{\mathrm{R}(y)}{k}\\
&\le 2^{H'(y)-H(y)+6}\cdot\frac{\mathrm{R}(y)^2}{k\cdot B}
\end{aligned}$$

where the first inequality is by Lemma 5, the second by Lemma 33, and the last inequality uses $2^{H(y)} \le 2^4\cdot \mathrm{R}(y)/B$, which follows from the definition of $H(y)$.

We again apply Fact 8 to obtain

$$\begin{aligned}
\Pr\left[\,|\mathrm{Err}'(y)| \ge \frac{\varepsilon\,\mathrm{R}(y)}{2}\right] &< 2\exp\left(-\frac{\varepsilon^2\cdot \mathrm{R}(y)^2}{4\cdot 2\cdot 2^{H'(y)-H(y)+6}\cdot \mathrm{R}(y)^2/(k\cdot B)}\right)\\
&= 2\exp\left(-\varepsilon^2\cdot k\cdot B\cdot 2^{-H'(y)+H(y)-9}\right)\\
&\le 2\exp\left(-2^{-H'(y)+H(y)}\right) = 2\exp\left(-\ln\frac{1}{\delta}\right) = 2\delta\,,
\end{aligned}$$

where the second inequality uses $k\cdot B \ge 2\cdot k^2 \ge \varepsilon^{-2}\cdot 2^9$.

Finally, we use deterministic bounds to analyze $\mathrm{Err}''(y)$. Note that

$$\mathrm{R}_{H(y)}(y) \le 2^{-H(y)+2}\,\mathrm{R}(y) \le B/2,$$

where the first inequality holds because we have conditioned on the bounds of Lemma 33 holding, and the second inequality holds by definition of $H(y)$. It follows that there is no important step at level $H(y)$, and hence no error introduced at any level $h \geq H(y)$, i.e., $\mathrm{Err}_h(y) = 0$ for $h \geq H(y)$. We thus have

$$\begin{aligned}
\mathrm{Err}''(y) &= \sum_{h=H'(y)}^{H(y)-1} 2^h \cdot \mathrm{Err}_h(y) \\
&\leq \sum_{h=H'(y)}^{H(y)-1} 2^h \cdot \frac{\mathrm{R}_h(y)}{k} \\
&\leq \sum_{h=H'(y)}^{H(y)-1} 2^h \cdot \frac{2^{-h+2}\,\mathrm{R}(y)}{k} \\
&\leq \sum_{h=H'(y)}^{H(y)-1} \frac{\varepsilon\,\mathrm{R}(y)}{2 \cdot \lceil \log_2 \ln \frac{1}{\delta} \rceil} \leq \frac{\varepsilon\,\mathrm{R}(y)}{2}\,,
\end{aligned}$$

where the first inequality is by Lemma 5, the second by Lemma 33, the third inequality follows from the definition of k in (36), and the last step uses that the sum is over $H(y) - H'(y) \leq \lceil \log_2 \ln \frac{1}{\delta} \rceil$ levels. This concludes the analysis of $\mathrm{Err}(y)$ and the calculation of the failure probability.

Regarding the space bound, there are at most $H \leq \lceil \log_2(n/B) \rceil + 1 \leq \log_2(\varepsilon n)$ relative-compactors by Observation 12, and each requires $B = 2 \cdot k \cdot \lceil \log_2(n/k) \rceil = O\left(\varepsilon^{-1} \cdot \log\log(1/\delta) \cdot \log(\varepsilon n)\right)$ memory words. □

The proof of Theorem 34 implies a deterministic sketch of size $O(\varepsilon^{-1} \cdot \log^3(\varepsilon n))$, which matches the state-of-the-art result by Zhang and Wang [22]. Indeed, when $\log_2 \ln(1/\delta) \geq \log_2(\varepsilon n) \geq H$ (i.e., $\delta < \exp(-\varepsilon n)$), we have $H'(y) = 0$, and in this case it is easily seen by inspecting the proofs of Lemma 33 and Theorem 34 that the entire analysis holds with probability 1. In more detail, when $H'(y) = 0$, the bounds in Lemma 33 hold with probability 1, and the quantity $\mathrm{Err}'(y)$ in the proof of Theorem 34 is deterministically 0, while the bound on $\mathrm{Err}''(y)$ in the proof of Theorem 34 holds with probability 1 as well. This is sufficient to conclude that the error guarantee holds for *any* choice of the algorithm's internal randomness. The resulting algorithm is reminiscent of deterministic algorithms for the uniform quantiles problem [14].

We remark that a deterministic algorithm achieving space $O(\varepsilon^{-1} \cdot \log^2(\varepsilon n) \cdot \log(n))$ (nearly matching the $O(\varepsilon^{-1} \cdot \log^3(\varepsilon n))$ bound derived above) also follows in a black-box manner from the statement of Theorem 34 by setting $\delta = 1/n^n$. This setting of δ is so small that one can union bound over all possible orderings of the input to conclude that there is some fixed setting of the algorithm's randomness that guarantees that for any possible input and any possible query y, $|\mathrm{Err}(y)| \leq \varepsilon\,\mathrm{R}(y)$.