Relative Error Streaming Quantiles[*]

GRAHAM CORMODE, University of Warwick, UK
ZOHAR KARNIN, Amazon, USA
EDO LIBERTY, Pinecone, USA
JUSTIN THALER, Georgetown University, USA
PAVEL VESELÝ, Charles University, Czech Republic

Estimating 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 equipped with a total order, the task is to compute a sketch (data structure) of size polylogarithmic in n. Given the sketch and a query item y, 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. We present a randomized sketch storing $O(\log^{1.5}(\varepsilon n)/\varepsilon)$ items that can $(1 \pm \varepsilon)$-approximate the rank of each universe item with high constant probability; this space bound is within an $O(\sqrt{\log(\varepsilon n)})$ factor of optimal. Our algorithm does not require prior knowledge of the stream length and is fully mergeable, rendering it suitable for parallel and distributed computing environments.

CCS Concepts: • **Theory of computation** → **Sketching and sampling**; *Streaming models*; *Data structures and algorithms for data management*.

Additional Key Words and Phrases: streaming algorithms, quantiles

1 INTRODUCTION

Understanding the distribution of data is a fundamental task in data monitoring and analysis. In many settings, we want to understand the cumulative distribution function (CDF) of a large number of observations, for instance, to identify anomalies. In other words, we would like to track the median, percentiles, and more generally quantiles of a massive input in a small space, without storing all the observations. Although memory constraints make an exact computation of such order statistics impossible [20], most applications can be satisfied with approximating the quantiles, which also yields a compact function with a bounded distance from the true CDF.

Authors' addresses: Graham Cormode, University of Warwick, Coventry, UK, G.Cormode@warwick.ac.uk; Zohar Karnin, Amazon, USA, zkarnin@gmail.com; Edo Liberty, Pinecone, San Mateo, CA, USA, edo@edoliberty.com; Justin Thaler, Georgetown University, Washington, D.C., USA, justin.thaler@georgetown.edu; Pavel Veselý, Charles University, Prague, Czech Republic, vesely@iuuk.mff.cuni.cz.

The problem of streaming quantile approximation captures this task in the context of massive or distributed datasets. 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 $\mathrm{R}(y; \sigma) = \left| \left\{ i \in \{1, \ldots, n\} \mid x_i \leq y \right\} \right|$ 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 in one pass while storing a small number of universe items and $O(\log n)$-bit variables (e.g., counters), 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 $|\hat{\mathrm{R}}(y) - \mathrm{R}(y)| \leq \varepsilon n$, and *multiplicative* or *relative* if $|\hat{\mathrm{R}}(y) - \mathrm{R}(y)| \leq \varepsilon \mathrm{R}(y)$.

If the algorithm is randomized, the desired error guarantee holds for each item y with high probability that can be specified upfront and that affects the space bound. (On the other hand, the space bounds for all known algorithms hold in the worst case over the inputs and random bits.) We remark that estimating ranks immediately yields approximate quantiles, and vice versa, with a similar error guarantee. More precisely, for $\phi \in [0, 1]$, we define a ϕ-quantile as the $\lfloor \phi n \rfloor$-th smallest item in σ. On quantile query ϕ, the algorithm should return a ϕ'-quantile such that $|\phi' - \phi| \leq \varepsilon$ for the additive error and $|\phi' - \phi| \leq \varepsilon \cdot \phi$ for the multiplicative error.

We stress that we do not assume any particular data distribution or that the stream is randomly-ordered, that is, we focus on worst-case inputs. However, we assume the input is independent of the random bits used by the algorithm, i.e., we do not aim to achieve adversarial robustness of randomized algorithms; cf. [4].

A long line of work has focused on achieving additive error guarantees [1–3, 11–13, 15, 17, 21, 22]. 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. [5], 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, 99.9, etc. This is because latencies are heavily long-tailed. For example, Masson et al. [19] 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 [5] and are problematic for users. Furthermore, as argued by Tene in his talk about measuring latency [26], one needs to look at extreme percentiles such as 99.995 to determine the latency such that only about 1% of users experience a larger latency during a web session with several page loads. Hence, highly accurate rank approximations are required for items y whose rank is very large ($n - \mathrm{R}(y) \ll n$); 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 randomized comparison-based additive error algorithms that store just $\Theta(\varepsilon^{-1})$ items for constant failure probability [15], 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 [5, Theorem 2] and Appendix A).[1]

The study of the relative-error (rank) guarantee was initiated by Gupta and Zane [14], and the best known algorithms achieving this guarantee are as follows. Zhang et al. [28] 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 a deterministic algorithm of Cormode et al. [6] 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

---

[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.

and Wang [27] designed a deterministic algorithm requiring $O(\varepsilon^{-1} \cdot \log^3(\varepsilon n))$ space. Recent work of Cormode and Veselý [8] 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.

Our streaming result. 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 one-pass streaming algorithm that given $\varepsilon > 0$, computes a sketch consisting of $O\left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon n)\right)$ universe items, from which one can derive rank or quantile estimates satisfying the relative error guarantee with constant probability (see Theorem 1 for a more precise statement). 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 [8]) and is within an $O(\sqrt{\log(\varepsilon n)})$ factor of the known lower bound for randomized algorithms achieving multiplicative error. Furthermore, it only accesses items through comparisons, i.e., is comparison-based, rendering it suitable, e.g., for floating-point numbers or strings ordered lexicographically. Finally, our algorithm processes the input stream efficiently, namely, its amortized update time is a logarithm of the space bound, i.e., $O\left(\log(\varepsilon^{-1}) + \log\log(n)\right)$; 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 [23]. Such mergeable summaries enable trivial, automatic parallelization and distribution of processing massive data sets, by splitting the data up into pieces, summarizing each piece separately, and then merging the results in an arbitrary way. We say that a sketch is *fully mergeable* if building it using any sequence of merge operations (executed on singleton items) leads to the same guarantees as if the entire data set had been processed as a single stream (i.e., always merging the sketch with one item). We show that our sketch satisfies this strong definition of mergeability.

The following theorem is the main result of this paper. We stress that our algorithm, which we call ReqSketch, does *not* require any advance knowledge about n, the total size of the input, which indeed may not be available in many applications.[2]

THEOREM 1. *For any parameters $0 < \delta \le 0.5$ and $0 < \varepsilon \le 1$, there is a randomized, comparison-based, one-pass streaming algorithm that, when processing a data stream consisting of n items from a totally-ordered universe $\mathcal{U}$, produces a summary S satisfying the following property. Given S, for any $y \in \mathcal{U}$ one can derive an estimate $\hat{R}(y)$ of $R(y)$ such that*

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

where the probability is over the internal randomness of the streaming algorithm. The size of S in memory words[3] is

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

Moreover, the summary produced is fully mergeable.

---

[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 of ReqSketch is available in the Apache DataSketches library at https://datasketches.apache.org/.

[3]A memory word can store any universe item or an integer with $O(\log(n + |\mathcal{U}|))$ bits. We express all the space bounds in memory words.

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 holds for all $y \in \mathcal{U}$ simultaneously with probability $1 - \delta$ when the size of the sketch in memory words is*

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

Technical overview. A starting point of the design of our algorithm is the KLL sketch [15] that achieves optimal accuracy-space trade-off (of randomized algorithms) 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, 16–18], and additional ideas are needed to get the optimal space bound for additive error, of $O(1/\varepsilon)$ items stored across all compactors [15].

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 $1/\varepsilon$ 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 $2/\varepsilon$ smallest items, the summary must store essentially every other item to achieve multiplicative error. Among the next $4/\varepsilon$ 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 [28], 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 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 [15]. 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 bitwise OR of their binary representations. With this technique for merging state variables in hand, we extend the charging argument bounding the number of compactions affecting the error in any given estimate so as to handle an arbitrary sequence of merge operations.

Analysis with extremely small probability of failure. We close by giving an alternative analysis of our algorithm that achieves a space bound with an exponentially better (double logarithmic) dependence on $1/\delta$, compared to Theorem 1. However, this improved dependence on $1/\delta$ comes at the expense of the exponent of $\log(\varepsilon n)$ increasing from 1.5 to 2. Formally, we prove the following theorem in Section 7, 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 [27]. 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 $0 < \delta \le 0.5$ 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 $\hat{R}(y)$ of $R(y)$ can be derived for every $y \in \mathcal{U}$. For any fixed $y \in \mathcal{U}$, with probability at least $1 - \delta$, the returned estimate satisfies the multiplicative error guarantee $|\hat{R}(y) - R(y)| \le \varepsilon R(y)$.*

We remark that this alternative analysis builds on an idea from [15] 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 n is known in advance. The space usage of the algorithm grows polynomially with the logarithm of this upper bound, so if this upper bound is at most n^c for some constant $c \ge 1$, then the space usage remains as stated in Theorem 1, with only the hidden constant factor changing. Then, in Section 5, we explain how to remove the assumption of a

foreknowledge of n 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 Section 6 (for didactic purposes, we outline a simplified merge operation in Section 2.3). As mentioned above, Section 7 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 $R(y)$ is close to r; this is called the *quantile query*. In this work, we focus on the dual problem of *rank queries*, where we consider queries to be universe items $y \in \mathcal{U}$ and the algorithm must yield an accurate estimate for $R(y)$. Unless specified otherwise, algorithms described in this section directly solve both formulations (this holds for our algorithm as well). Algorithms are randomized unless stated otherwise. For simplicity, randomized algorithms are assumed to have constant failure probability δ. All reported space costs refer to the number of universe items stored. (Apart from storing items, the algorithms may store, for example, bounds on ranks of stored items or some counters, but the number of such $O(\log n)$-bit variables is proportional to the number of items stored or even smaller.)

Additive Error. Manku, Rajagopalan, and Lindsay [17, 18] built on the work of Munro and Paterson [20] 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 [13] 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 [8]. Agarwal et al. [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 [11] 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 [15] 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 $\delta \leq 1/n!$ (i.e., 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 considered in this work (sometimes also called the biased quantiles problem). Gupta and Zane [14] gave an algorithm for relative error quantiles that stores $O(\varepsilon^{-3} \cdot \log^2(\varepsilon n))$ items, and used 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. [28] presented an algorithm storing $O(\varepsilon^{-2} \cdot \log(\varepsilon^2 n))$ universe items. Cormode et al. [6] 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. [25] in the additive error setting and it is also mergeable (see [1, Section 3]). Zhang and Wang [27] 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 [8] 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 [18] designed an algorithm that, for a specified number $\phi \in [0, 1]$, stores $O(\varepsilon^{-1} \cdot \log(1/\delta))$ items and can return an item y with $R(y)/n \in [(1 - \varepsilon)\phi, (1 + \varepsilon)\phi]$ (their algorithm requires prior knowledge

of n). Cormode et al. [5] 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: [28] observed that for adversarial item ordering, the algorithm of [5] requires linear space to achieve relative error for all ranks.

Dunning and Ertl [9, 10] describe a heuristic algorithm called t-digest that is intended to achieve relative error, but they provide no formal accuracy analysis (note that t-digest is deterministic but *not* comparison-based). Indeed, Cormode et al. [7] show that the error of t-digest may be arbitrarily large on adversarially generated inputs. This latter paper also compares t-digest and ReqSketch (i.e., the algorithm of Theorem 1) on randomly generated inputs and proposes implementation improvements for ReqSketch that make it process an input stream faster than t-digest.

Most recently, Masson, Rim, and Lee [19] considered a notion of relative *value* error for quantile sketches, which is distinct from the notion of relative *rank* error considered in this work. They require that for a query percentile $\phi \in [0, 1]$, if y denotes the item in the data stream satisfying $R(y) = \phi n$, then the algorithm should return an item $\hat{y} \in \mathcal{U}$ such that $|y - \hat{y}| \leq \varepsilon \cdot |y|$. 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 $((1+\varepsilon)^i, (1+\varepsilon)^{i+1}]$. 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 of size $\Omega(\varepsilon^{-1})$ (where n is the length of the input stream), the output of the last relative-compactor is of size $O(1/\varepsilon)$, and hence can be stored in memory.

Compaction 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 $x_1 \leq x_2 \leq \ldots \leq x_{2m}$, 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 of X, with both outcomes equally probable.

Consider an item $y \in \mathcal{U}$ and recall that $R(y; X) = |\{x \in X \mid x \leq y\}|$ is the number of items $x \in X$ satisfying $x \leq y$ (we remark that both X and $\{x \in X \mid x \leq y\}$ are multisets of universe items). 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 2.1 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. Note that the ranks are only w.r.t. to the input X of the operation, which will contain a number of the largest items in a relative-compactor. However, the observation holds for any universe item that may not be present in X.
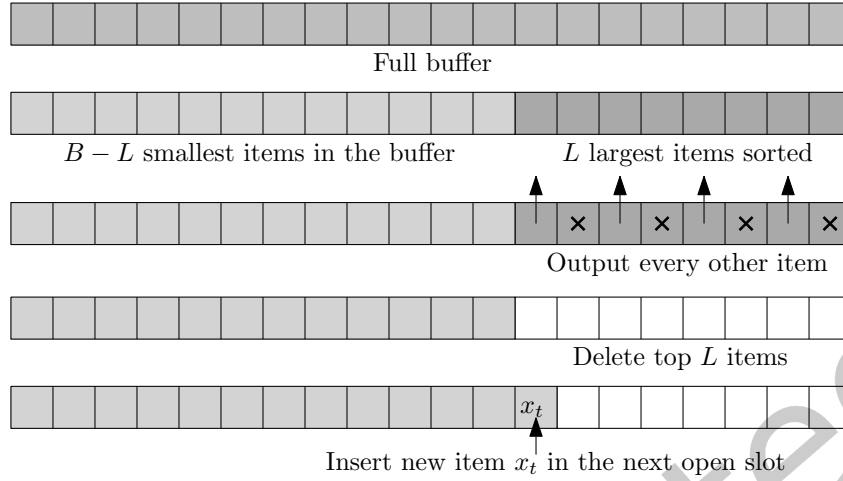
Fig. 1. Illustration of the execution of a relative-compactor when inserting a new item x_t into a buffer that is full at time t. See lines 5-14 of Algorithm 1.

OBSERVATION 2.1. *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 $\mathrm{R}(y; X) - 2\,\mathrm{R}(y; Z) = 0$, where Z is the output sequence of the operation. 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 $L \leq B/2$ 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 correctly, 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 $B \approx 1/\varepsilon^2$, 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 $L \leq B/2$.[4]

---

[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.

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 $\mathcal{B}$.

	7	6	5	4	3	2	1

$B/2$ slots (never compacted) $\lceil \log_2(n/k) \rceil = 7$ sections with k slots each
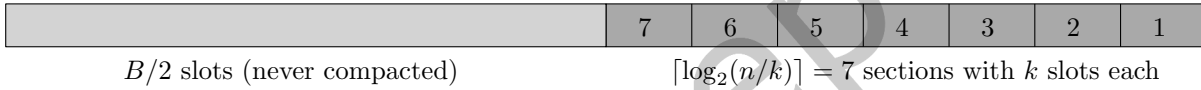
Fig. 2. Illustration of a relative-compactor and its sections, together with the indexes of the sections.

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 $L \geq k$), the second section in every other compaction (i.e., $L \geq 2k$ every other time), the third section in every fourth compaction, and so on. This can be described concisely as follows: Let C be the number of compactions performed so far. During the next (i.e., the $C + 1$-st) compaction of the relative-compactor, we set $L_C = (z(C) + 1) \cdot k$, where $z(C)$ is the number of trailing ones in the binary representation of C (that is, if C viewed as a bitstring can be written as $(\mathbf{x}, 0, \mathbf{1}^j)$, for some $\mathbf{x}$, then $z(C) = j$). 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:

OBSERVATION 2.2. *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 $(\mathbf{x}, 0, \mathbf{1}^{j-1})$

Algorithm 2 ReqSketch (Relative-Error Quantiles sketch)

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

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

---

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, 15, 17], 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 (recall that h starts from 0), and the approximate rank of y, denoted $\hat{R}(y)$, is the sum of weights of items in C smaller than or equal to y. Similarly, ReqSketch can answer quantile queries, i.e., for a given rank $r \in \{1, \ldots, n\}$, return an item $y \in \mathcal{U}$ with R(y) close to r; the algorithm just returns an item y stored in one of the relative-compactors with $\hat{R}(y)$ closest to the query rank r among all items in the sketch.

Algorithm 3 Merge operation

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

The construction of layered exponentially-weighted compactors and the subsequent rank estimation is virtually identical to that explained in prior works [1, 15, 17]. 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.

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 summarizing 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', and in fact, we will merge sketch S'' into S'. We further assume that both S' and S'' have the same value of k (otherwise, it would not be meaningful to analyze the error of S) and that n is still an upper bound on the combined input size. Later, in Section 6, 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 a 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 Section 6. 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, $S.k$ for the parameter k of 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 $R(y)$ be the rank of item y in the input stream, and $\mathrm{Err}(y) = \hat{R}(y) - R(y)$ the error of the estimated rank for y. Our analysis of $\mathrm{Err}(y)$ relies on just two properties.

(1) The level-h compactor only does at most $R(y)/(k \cdot 2^h)$ compactions that affect the error of y (up to a constant factor).

Roughly speaking, this holds by the following reasoning. First, recall from Observation 2.1 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 4.4). This is the source of the 2^h factor in the denominator. Second, we show that each compaction operation that affects $\mathrm{Err}(y)$ 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 3.1). This is the source of the k factor in the denominator.

(2) Let H_y be the smallest positive integer such that $2^{H_y} \gtrsim R(y)/B$ (the approximate inequality $\gtrsim$ hides a universal constant). Then no compactions occurring at levels above H_y affect $\mathrm{Err}(y)$, 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 4.4).

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

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

where in the LHS, $R(y)/(k2^h)$ bounds the number of level-h compaction operations affecting the error (this exploits Property 1 above), and 2^{2h} is the variance contributed by each such compaction, due to Observation 2.1 and because each item processed by the relative-compactor at level h represents 2^h items in the original stream.

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

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

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

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. In Section 5,

---

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

we provide a simple argument that the assumption of having such an upper bound on n is not needed in the streaming setting.

For the most general analysis under an arbitrary sequence of merge operations (i.e., for the proof of full mergeability) and without assuming a foreknowledge of n, we refer to Section 6.

3 ANALYSIS OF THE RELATIVE-COMPACTOR IN THE STREAMING SETTING

To analyze our algorithm, we keep track of the error associated with an arbitrary fixed item y. Throughout this section, we restrict our attention to any single relative-compactor at some level h (Algorithm 1) maintained by our sketching algorithm (Algorithm 2), and we use "time t" to refer to the t-th insertion operation to this particular relative-compactor.

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

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

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

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

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

Clearly, if the algorithm performs no compaction operation in a time step t, then $\Delta_h^t(y) = 0$. (Recall that a compaction is an execution of lines 6-13 of Algorithm 1.) Let us consider what happens in a step t in which a compaction operation occurs. Recall from Observation 2.1 that if y is even with respect to the compaction (i.e., y has even rank w.r.t. the L largest items in the relative-compactor), then y suffers no error, meaning that $\Delta_h^t(y) = 0$. Otherwise, $\Delta_h^t(y)$ is uniform in $\{-1, 1\}$.

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

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

LEMMA 3.1. *Consider the relative-compactor at level h, fed an input stream of length at most n. For any fixed item $y \in \mathcal{U}$ with rank $\text{R}_h(y)$ in the input stream to level h, there are at most $\text{R}_h(y)/k$ important steps. In particular,*

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

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

For any $j \geq 1$, let s_j be the number of important j-steps. Further, let $R_{h,j}(y)$ 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 $R_{h,j}(y)$ for $j = \lceil \log_2(n/k) \rceil + 1$; for this j, 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 $\sum_{j \geq 1} R_{h,j}(y) \leq R_h(y)$. We will show

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

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

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

To show the lower bound on $R_{h,j+1}(y)$ 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 2.1. As section $j + 1$ contains smaller-ranked (or equal-ranked) items than section j, section $j + 1$ contains important items only. We have two cases for charging the important j-step t:

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

Case B: Otherwise, there is no compaction operation after step t that involves at least $j + 1$ buffer sections. Then, we charge step t to the k important items that are in the $(j + 1)$-st section at the end of execution.

It remains to observe that each important item x accounted for in $R_{h,j+1}(y)$ is charged at most once. (Note that different compactions may be charged to the items which are consumed during the same later compaction, but our charging will ensure that these are assigned to different sections. For example, consider a sequence of three consecutive important steps (there is no compaction in other steps in between them) such that in the first one the algorithm compacts 2 sections, then 1 section, and 3 sections in the third important step. 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 (Observation 2.2) 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, and there is at most one such important step by Observation 2.2. This shows (4), which implies the lemma as noted above. □

4 ANALYSIS OF THE FULL SKETCH IN THE STREAMING SETTING

We denote by $\text{Err}_h(y)$ 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,

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

For the analysis, we first set the value of parameter k of Algorithm 2. Namely, given (an upper bound on) the stream length n, the desired accuracy $0 < \varepsilon \le 1$, and the 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, the space cost of $\sqrt{\ln(1/\delta)}$ becomes $\Omega(\sqrt{\varepsilon n})$, and furthermore, the analyses in Sections 6 and 7 do not require such an assumption). We start by showing a lower bound on $k \cdot B$.

CLAIM 4.1. *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 $R_h(y) > 0$.

OBSERVATION 4.2. $R_{h+1}(y) \le \max\{0, R_h(y) - B/2\}$ *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 $R_{h+1}(y) \le \max\{0, R_h(y) - B/2\}$. □

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

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

Below, in Lemma 4.4, we show that no important item (i.e., one smaller than or equal to y) can ever reach level $H(y)$ with high probability, unless $H(y) = 0$. (For $H(y) = 0$, all important items fit into the level-0 buffer, so the estimated rank $\hat{R}(y)$ equals $R(y)$.) Recall that a zero-mean random variable X with variance σ^2 is sub-Gaussian if $\mathbb{E}[\exp(sX)] \leq \exp(-\frac{1}{2} \cdot s^2 \cdot \sigma^2)$ for any $s \in \mathbb{R}$; note that a (weighted) sum of independent zero-mean sub-Gaussian variables is a zero-mean sub-Gaussian random variable as well. We will use the following standard (Chernoff) tail bound for sub-Gaussian variables (see, e.g., Lemma 1.3 in [24]):

FACT 4.3. *Let X be a zero-mean sub-Gaussian variable with variance at most σ^2. Then for any $a > 0$, it holds that*

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

LEMMA 4.4. *Assuming $H(y) > 0$, with probability at least $1 - \delta$ it holds that $R_h(y) \leq 2^{-h+1} R(y)$ for any $h < H(y)$.*

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

Next, consider $h > 0$ and condition on $R_\ell(y) \leq 2^{-\ell+1} R(y)$ for any $\ell < h$. Observe that any compaction operation at any level ℓ that involves a important items inserts $\frac{1}{2}a$ such items to the input stream at level $\ell + 1$ in expectation, no matter whether a is odd or even. Indeed, if a is odd, then the number of important items promoted is $\frac{1}{2}(a + X)$, where X is a zero-mean random variable uniform on $\{-1, 1\}$. For an even a, the number of important items that are promoted is $\frac{1}{2}a$ with probability 1.

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

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

where $\text{Binomial}(n)$ represents the sum of n zero-mean i.i.d. random variables uniform on $\{-1, 1\}$.

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

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

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

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

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

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

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

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

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

To bound the variance of Z_h, first note that for any $\ell < h$, we have $m_\ell \leq R_\ell(y)/k \leq 2^{-\ell+1} R(y)/k$ by Lemma 3.1 and by conditioning on $R_\ell(y) \leq 2^{-\ell+1} R(y)$. As $\text{Var}[\text{Binomial}(n)] = n$, the variance of Z_h is

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

Note that $\Pr[Y_h > 2^{-h+1} R(y)] = \Pr[Z_h > 2^{-h} R(y)]$. To bound the latter probability, we apply the tail bound for sub-Gaussian variables (Fact 4.3) to get

$$\Pr[Z_h > 2^{-h} R(y)] < \exp\left(-\frac{2^{-2h} \cdot R(y)^2}{2 \cdot (2^{-h+1} \cdot R(y)/k)}\right)$$

$$= \exp\left(-2^{-h-2} \cdot R(y) \cdot k\right)$$

$$= \exp\left(-2^{-h+H(y)-6} \cdot 2^{4-H(y)} R(y) \cdot k\right)$$

$$\leq \exp\left(-2^{-h+H(y)-6} \cdot B \cdot k\right) \tag{14}$$

$$\leq \exp\left(-2^{-h+H(y)-6} \cdot 2^6 \cdot \frac{1}{\varepsilon^2} \cdot \ln\frac{1}{\delta}\right) \tag{15}$$

$$\leq \exp\left(-2^{-h+H(y)} \cdot \ln\frac{1}{\delta}\right) = \delta^{2^{H(y)-h}} \leq \delta \cdot 2^{-H(y)+h} \, , \tag{16}$$

where inequality (14) uses $2^{4-H(y)} R(y) \geq B$ (by the definition of $H(y)$, cf. Equation (8)), inequality (15) follows from Claim 4.1, inequality (16) uses $\varepsilon \leq 1$, and the last inequality uses $\delta \leq 0.5$. As explained above, this concludes the proof. □

In what follows, we condition on the bound on $R_h(y)$ in Lemma 4.4 for any $h < H(y)$.

LEMMA 4.5. *Assume that $H(y) > 0$. Conditioned on the bound on $R_{H(y)-1}(y)$ in Lemma 4.4, it holds that* $R_{H(y)}(y) = 0$.

PROOF. According to Lemma 4.4 and the definition of $H(y)$ as the minimal h for which $2^{2-h} R(y) \leq B/2$,

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

Invoking Observation 4.2, we get $R_{H(y)}(y) \leq \max\{0, R_{H(y)-1}(y) - B/2\} = 0$. □

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

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

where H is the highest level with a relative-compactor (that never produces any output). To bound this error, we refine the guarantee of Lemma 3.1. Notice that for any particular relative-compactor, the bound $\sum_{t=1}^{n} |\Delta_h^t(y)|$ referred to in Lemma 3.1 applied to a level h is a potentially crude upper bound on $\text{Err}_h(y) = \sum_{t=1}^{n} \Delta_h^t(y)$: Each non-zero term $\Delta_h^t(y)$ is positive or negative with equal probability, so the terms are likely to involve a large amount of cancellation. To take advantage of this, we bound the variance of $\text{Err}(y)$.

LEMMA 4.6. *Conditioned on the bound on* $R_h(y)$ *in Lemma 4.4 for any* $h < H(y)$, $\text{Err}(y)$ *is a zero-mean sub-Gaussian random variable with* $\text{Var}[\text{Err}(y)] \leq 2^5 \cdot R(y)^2/(k \cdot B)$.

PROOF. Consider the relative-compactor at any level h. By Lemma 3.1, $\text{Err}_h(y)$ is a sum of at most $R_h(y)/k$ random variables, i.i.d. uniform in $\{-1, 1\}$. In particular, $\text{Err}_h(y)$ is a zero-mean sub-Gaussian random variable with $\text{Var}[\text{Err}_h(y)] \leq R_h(y)/k$. Thus, $\text{Err}(y)$ is a sum of independent zero-mean sub-Gaussian random variables, and as such is itself a zero-mean sub-Gaussian random variable.

It remains to bound the variance of $\text{Err}(y)$, for which we first bound $\text{Var}[\text{Err}_h(y)]$ for each h. If $R_h(y) = 0$, then Observation 2.1 implies that $\text{Err}_h(y) = 0$, and hence that $\text{Var}[\text{Err}_h(y)] = 0$. Thus, using Lemma 4.5, we have $\text{Var}[\text{Err}_h(y)] = 0$ for any $h \geq H(y)$. For $h < H(y)$, we use $\text{Var}[\text{Err}_h(y)] \leq R_h(y)/k$ to obtain:

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

where the second inequality is due to Lemma 4.4 and the last inequality follows from (8). □

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

OBSERVATION 4.7. *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 3. *Assume that (a polynomial upper bound on) the stream length* n *is known in advance. For any parameters* $0 < \delta \leq 0.5$ *and* $0 < \varepsilon \leq 1$ *satisfying* $\delta > 1/\exp(\varepsilon n/64)$, *there is a randomized, comparison-based, one-pass streaming algorithm that, when processing a data stream consisting of* n *items from a totally-ordered universe* $\mathcal{U}$, *produces a summary* S *satisfying the following property. Given* S, *for any* $y \in \mathcal{U}$ *one can derive an estimate* $\hat{R}(y)$ *of* $R(y)$ *such that*

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

where the probability is over the internal randomness of the streaming algorithm. The size of S *in memory words is*

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

PROOF. First, suppose that $\varepsilon \leq 4 \cdot \sqrt{\ln(1/\delta)/\log_2(\varepsilon n)}$. Then we use Algorithm 2 with parameters k and n, where k is set as in (6). Note that k is an even positive integer as required by Algorithm 2. By Lemma 4.4, with probability at least $1 - \delta$, we have $R_h(y) \leq 2^{-h+1} R(y)$ 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 4.3) together with Lemma 4.6 (for which we need the bound on $\mathrm{R}_h(y)$ for any $h < H(y)$) and obtain

$$
\Pr\left[\,|\operatorname{Err}(y)| \geq \varepsilon\, \mathrm{R}(y)\right] < 2\exp\left(-\frac{\varepsilon^2 \cdot \mathrm{R}(y)^2}{2 \cdot 2^5 \cdot \mathrm{R}(y)^2/(k \cdot B)}\right)
$$
$$
\leq 2\exp\left(-\frac{\varepsilon^2 \cdot 2^6 \cdot \varepsilon^{-2} \cdot \ln\frac{1}{\delta}}{2^6}\right) = 2\exp\left(-\ln\frac{1}{\delta}\right) = 2\delta,
$$

where we use Claim 4.1 in the second inequality. This concludes the calculation of the failure probability (up to scaling δ by a factor of $1/3$).

Regarding the memory usage, there are at most $\lceil\log_2(n/B)\rceil + 1 \leq \log_2(\varepsilon n)$ relative-compactors by Observation 4.7, and each requires $B = 2 \cdot k \cdot \lceil\log_2(n/k)\rceil$ memory words. Thus, the memory needed to run the algorithm is at most

$$
\log_2(\varepsilon n) \cdot 2 \cdot k \cdot \left\lceil\log_2\frac{n}{k}\right\rceil \leq \log_2(\varepsilon n) \cdot 2 \cdot 2 \cdot \left\lceil\frac{4}{\varepsilon} \cdot \sqrt{\frac{\ln\frac{1}{\delta}}{\log_2(\varepsilon n)}}\right\rceil \cdot O\left(\log(\varepsilon n)\right), \tag{17}
$$

where we use that $\lceil\log_2(n/k)\rceil \leq O\left(\log(\varepsilon n)\right)$, which follows from $k \geq \varepsilon^{-1}/\sqrt{\log_2(\varepsilon n)}$. Using $\varepsilon \leq 4\cdot\sqrt{\ln(1/\delta)/\log_2(\varepsilon n)}$, we have $a := 4\varepsilon^{-1}\cdot\sqrt{\ln(1/\delta)/\log_2(\varepsilon n)} \geq 1$, so $\lceil a\rceil \leq 2a$ and it follows that (17) is bounded by $O\left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon n) \cdot \sqrt{\log(1/\delta)}\right)$.

For $\varepsilon > 4 \cdot \sqrt{\ln(1/\delta)/\log_2(\varepsilon n)}$, we use the comparison-based streaming algorithm by Zhang et al. [28] that requires space $O\left(\varepsilon^{-2} \cdot \log(\varepsilon^2 n) \cdot \log(1/\delta)\right)$ and otherwise satisfies the same error guarantee as our algorithm. To get the desired space bound, we observe that the case condition implies $\sqrt{\log_2(\varepsilon n)} > 4 \cdot \sqrt{\ln(1/\delta)} \cdot \varepsilon^{-1}$ and thus, $O\left(\varepsilon^{-2} \cdot \log(\varepsilon^2 n) \cdot \log(1/\delta)\right) \leq O\left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon n) \cdot \sqrt{\log(1/\delta)}\right)$.[6] We remark that the algorithm from [28] does not require any foreknowledge of the total input length n. □

Update time. We now analyze the amortized update time of Algorithm 2 and show that it can be made $O(\log B) = O(\log(k) + \log\log(n/k))$, 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 (lines 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 3 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

---

[6]In fact, as we show in Section 6, one may use a variant of our algorithm also for the case of large ε, that is, when $\varepsilon > 4 \cdot \sqrt{\ln(1/\delta)/\log_2(\varepsilon n)}$. Namely, we compute the largest value of $\overline{n}$ such that $1 < k = 2 \cdot \left\lceil(4/\varepsilon) \cdot \sqrt{\ln(1/\delta)/\log_2(\varepsilon\overline{n})}\right\rceil$ (for given ε and δ); cf. (18) in Section 6. If $n > \overline{n}$, then using buffers of size $\Theta(\log\varepsilon\overline{n})$ is sufficient and we do not need to use the compaction schedule (intuitively, the section size k is too small to be useful). In this section, we omit these details for brevity and focus just on the main case of relatively small ε.

polynomially with the logarithm of this upper bound, so if this upper bound is at most n^c for some constant $c \geq 1$, then the space usage of the algorithm will remain as stated in Theorem 3, 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, namely, $N_0 = \Theta(\varepsilon^{-1})$. That is, we begin by running Algorithm 2 with parameters k and N_0. 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 $N_{i+1} = N_i^2$ (i.e., we execute Algorithm 2 with parameters k and N_{i+1}; only if $\varepsilon > 4 \cdot \sqrt{\ln(1/\delta)/\log_2(\varepsilon N_{i+1})}$ we switch to the algorithm from [28] as in the proof of Theorem 3).[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 a variant of Theorem 3 for unknown stream lengths, we need to bound the space usage of the algorithm, and the probability of having a too large error for a fixed item y. We start with some notation. Let ℓ be the biggest index i of estimate N_i used by the algorithm; note that $\ell \leq \log_2 \log_2(\varepsilon n)$. Let σ_i denote the substream processed by the summary with the i-th guess for the stream length for $i = 0, \ldots \ell$. Let $\sigma' \circ \sigma''$ denote the concatenation of two streams σ' and σ''. Then the complete stream processed by the algorithm is $\sigma = \sigma_0 \circ \sigma_1 \circ \cdots \circ \sigma_\ell$. Let k_i and B_i be the values of parameters k and B computed for estimate N_i.

Space bound. We claim that the sizes of summaries for the substreams $\sigma_0, \sigma_1, \ldots, \sigma_\ell$ sum up to $O\left(\varepsilon^{-1} \cdot \log^{1.5}(\varepsilon n) \cdot \sqrt{\log(1/\delta)}\right)$, as required. By Theorem 3, 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 3 increases by a factor of $2^{1.5}$ when we increase i. It follows that the total space usage is dominated, up to a constant factor, by the size of the summary for σ_ℓ. □

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

We apply the analysis in Section 4 to all of the summaries at once. Observe that for the tail bound in the proof of Theorem 3, 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 4.6, $\operatorname{Err}^i(y)$ is a zero-mean sub-Gaussian random variable with $\operatorname{Var}[\operatorname{Err}^i(y)] \leq 2^5 \cdot R(y; \sigma_i)^2/(k_i \cdot B_i)$. As $\operatorname{Err}(y) = \sum_i \operatorname{Err}^i(y)$ and as the summaries are created with independent randomness, variable $\operatorname{Err}(y)$ is also zero-mean sub-Gaussian and its variance is bounded by

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

where the last inequality uses that $\sum_{i=0}^{\ell} R(y; \sigma_i)^2 \leq R(y)^2$, which follows from $R(y) = \sum_{i=0}^{\ell} R(y; \sigma_i)$, and that $k_i \cdot B_i = \Omega(\varepsilon^{-2} \cdot \ln(1/\delta))$, which holds by Claim 4.1. Applying the tail bound for sub-Gaussian variables similarly as in the proof of Theorem 3 concludes the proof of (a variant of) Theorem 3 for unknown stream lengths. □

---

[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 Section 6 (which applies in the more general mergeability setting) shows that the same accuracy guarantees as in Theorem 3 hold for this variant of our 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 Section 6).

6 FULL MERGEABILITY

Fully-mergeable sketches allow us 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. We show that our sketch maintains its accuracy guarantees even in these settings, and therefore, it is fully mergeable.

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 same 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 (i.e., using an arbitrary merge tree).

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.

6.1 Merge Operation

In this section, we describe the 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 list RELCOMPACTORS of its relative-compactors and the following parameters:

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, $\hat{k} = 4\varepsilon^{-1} \cdot \sqrt{\ln(1/\delta)}$. 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.

Parameters N, k, and B. The parameter N is set similarly as in Section 5, that is, it is equal to N_i for some i, where $N_0 = \lceil 2^{10} \cdot \hat{k} \rceil$ 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)) so that k decreases and B increases as we increase N. Importantly, we no longer change k and B once $\hat{k} \leq \sqrt{\log_2(N_i/\hat{k})}$. To facilitate this, we define $\lambda \geq 0$ as the smallest integer i such that

$$\frac{\hat{k}}{\sqrt{\log_2(N_i/\hat{k})}} \leq 1, \tag{18}$$

and then for $i \geq 0$ we set

$$k_i := 2^5 \cdot \left\lceil \frac{\hat{k}}{\sqrt{\log_2(\overline{N_i}/\hat{k})}} \right\rceil \quad \text{and} \quad B_i := 2 \cdot k_i \cdot \left\lceil \log_2\left(\frac{\overline{N_i}}{k_i}\right) \right\rceil \quad \text{where } \overline{N_i} = \min\{N_i, N_\lambda\}. \tag{19}$$

From a practical point of view, since N_λ is about $\hat{k} \cdot 2^{\hat{k}^2}$, we have that $\overline{N_i} = N_i$ unless N_i is extremely large or $\hat{k} = 4\varepsilon^{-1} \cdot \sqrt{\ln(1/\delta)}$ is small (say, even for $\varepsilon = 0.2$ we have $N_\lambda \gg 2^{400}$). We use this truncation of N_i to guarantee the space bound when $n > N_\lambda$. Furthermore, observe that once we reach $n \geq N_\lambda$, the values of k_i and B_i do not

change; this is because, intuitively, the section size k_i becomes too small to help in the analysis and our algorithm can in fact be simplified by involving all sections in every compaction, without violating the error guarantees (i.e., when $n \geq N_\lambda$, the compaction schedule is no longer relevant). The most challenging part of the analysis is bounding the error for $i \leq \lambda$.

Description of the merge operation. 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 the 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 (19) above.

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 without violating the relative-error guarantees even when many merge operations are executed.[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 level-h 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 among 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.* Note that there is at most one compaction per level during the merge operation. Finally, when $N_i > N_\lambda$, we do not use the compaction schedule as the section size becomes too small, i.e., we compact all buffer sections.

Algorithm 4 provides pseudocode describing the merge operation specified above. We note 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 the 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 $\frac{3}{2} \cdot S.B$ items, 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 $S.B$ as defined in (19) is divisible by

---

[8]By the 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 of ReqSketch

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:     Set $S'.N = S'.N^2$ ▷ Square the upper bound
4:     Set $S'.k$ and $S'.B$ according to (19)
5: **for** $h = 0, \ldots, S''.H$ **do**   ▷ Combine buffers and states of compaction schedules
6:     Insert all items in $S''.\textsc{RelCompactors}[h]$ into $S'.\textsc{RelCompactors}[h]$
7:     $S'.\textsc{RelCompactors}[h].C = S'.\textsc{RelCompactors}[h].C$ OR $S''.\textsc{RelCompactors}[h].C$
8: **for** $h = 0, \ldots, S'.H$ **do**
9:     **if** there are at least $S'.B$ items in $S'.\textsc{RelCompactors}[h]$ **then**
10:        $\textsc{PerformCompaction}(S', h)$
11: **return** S'
12: **function** $\textsc{PerformCompaction}(S', h, s)$
13:     **if** $h = S'.H$ **then**
14:        Increase $S'.H$ by one
15:        Initialize relative-compactor at $\textsc{RelCompactors}[h + 1]$
16:     Set $\mathcal{B} = S'.\textsc{RelCompactors}[h]$ ▷ The level-h buffer of S'
17:     Sort items in $\mathcal{B}$ in non-descending order
18:     **if** $S'.N \leq N_\lambda$ **then**   ▷ λ is defined in (18)
19:        Compute z = number of trailing 1s in binary representation of $\mathcal{B}.C$
20:        Set $s = S'.B - (z + 1) \cdot S'.k + 1$ ▷ First slot of the buffer involved in the compaction
21:     **else**   ▷ Then $S'.k = \Theta(1)$
22:        Set $s = S'.B/2$ ▷ Compaction schedule not used when $S'.k$ is small
23:     Let $|\mathcal{B}|$ be the number of items stored in $\mathcal{B}$ ▷ $|\mathcal{B}|$ may be larger than $S'.B$
24:     Set Z = equally likely either even or odd indexed items in the range $\mathcal{B}[s : |\mathcal{B}|]$
25:              ▷ Note that the range $\mathcal{B}[s : |\mathcal{B}|]$ may be of an odd size, which does not cause any issues
26:     Insert each item in Z to $S'.\textsc{RelCompactors}[h + 1]$
27:     Mark slots $\mathcal{B}[s : |\mathcal{B}|]$ in the buffer as clear
28:     Increase $\mathcal{B}.C$ by 1

---

four, so $\frac{3}{2} \cdot S.B$ is 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 7 to combine the states has two simple but important implications.

FACT 6.1. *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.*

FACT 6.2. *The bitwise OR of C' and C'' (interpreted as bitstrings) is no larger than $C' + C''$ (interpreted as integers).*

Fact 6.2 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 compaction. See Observation 6.3 for details. (Note that this is only relevant for $S.N \leq N_\lambda$.)

6.2 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. We will show that the space bound holds for S using an argument similar to the one in the proof of Theorem 3, 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.

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, node t represents the level-h buffer of S_t. Finally, we say that t represents the level-h compaction operation (if any); recall that the merge operation captured by an internal node t performs at most one compaction operation at each level h. The root of T represents the final merge operation, which outputs the *final sketch*.

Recall that we set the upper bounds N on the input size used by the sketches as $N_0 = \lceil 2^{10} \cdot \hat{k} \rceil$ and $N_i = N_{i-1}^2$ for $1 \le i \le \ell \le \lceil \log_2 \log_2(\varepsilon n) \rceil$ (as $N_0 \ge \hat{k} \ge 1/\varepsilon$). We assume that $\ell > 0$, otherwise the whole input can be stored in space $O(\hat{k}) = O(\varepsilon^{-1} \cdot \sqrt{\log(1/\delta)})$.

We say that an (internal) node t in tree T is an *i-node* for $0 \le i \le \ell$ 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 for any i, the subtrees of topmost i-nodes are disjoint.

As in Sections 3 and 4, we consider a fixed item y and analyze the error of the estimated rank of y. Let $R(y)$ denote the rank of y in the input summarized by the sketch, and let $\hat{R}(y)$ be the estimated rank of y obtained from the final sketch S; recall that we get this estimate by summing over all levels $h \ge 0$ the number of items $x \le y$ in the level-h buffer of the final sketch, multiplied by 2^h. Our aim is to show that $|\operatorname{Err}(y)| = |\hat{R}(y) - R(y)| \le \varepsilon R(y)$ with probability at least $1 - \delta$.

6.3 Analysis of a Single Level for Mergeability

For the duration of this section, we consider a single level h and solely focus on i-nodes for $i \le \lambda$; recall that the compaction schedule helps to decrease the error from compactions and that we do not use the schedule during compactions represented by i-nodes for $i > \lambda$ (since the buffer section size is too small to make a difference). For convenience, λ refers to $\min\{\lambda, \ell\}$, i.e., if $\lambda > \ell$ we decrease λ to ℓ compared to (18). This is to ensure that, e.g., topmost λ-nodes are well-defined. Note that when $\lambda = \ell$, then the only topmost λ-node is the root of the merge tree T.

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 6.3. *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 all 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 $C' \leq r'/S'.k$ and $C'' \leq r''/S''.k$. Note that r equals $r' + r''$ plus the number of items removed from the level-h buffer during the 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 $C = (C' \text{ OR } C'') + b$ and if $b = 1$, then the compaction removes at least $S.k$ items from the level-h buffer. We thus have $r \geq r' + r'' + b \cdot S.k$ and using this, we obtain

$$C = (C' \text{ OR } C'') + b \leq C' + C'' + b \leq \frac{r'}{S'.k} + \frac{r''}{S''.k} + b \leq \frac{r'}{S.k} + \frac{r''}{S.k} + \frac{b \cdot S.k}{S.k} \leq \frac{r}{S.k},$$

where the penultimate inequality uses $S.k \leq \min\{S'.k, S''.k\}$, which follows from $k_0 \geq k_1 \geq \cdots \geq k_\lambda$. □

For $i \leq \lambda$, we recall that the second half of the buffer of size B_i has $\lceil \log_2(N_i/k_i) \rceil$ sections of size k_i (see Equation (19)) and that these sections are indexed from 1 such that the rightmost section (with slots $B_i - k_i + 1, \ldots, B_i$) is section 1 and section j consists of slots $B_i - j \cdot k_i + 1, \ldots, B_i - (j - 1) \cdot k_i$. The definition of the compaction operation and Observation 6.3 imply that section $\lceil \log_2(N_i/k_i) \rceil$ (i.e., the leftmost section of the second half of the buffer) is involved only in one compaction represented by an i-node on any leaf-to-root path in T.

Bounding the number of important compaction operations. As in Section 3, the key part of the analysis is bounding the number of level-h compaction operations that introduce some error for the fixed item y; recall that we call such compactions important and that by Observation 2.1, a compaction is important if and only if it removes an odd number of important items from the buffer. Also, recall that we call items $x \leq y$ important and that for $h > 0$, $R_h(y)$ 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 $R_0(y) = R(y)$.

The bound on the number of important level-h compactions in Lemma 6.4 below is more involved than in the streaming setting (Section 3), but this complexity allows for the tightest and most general analysis, presented in Section 6.4. In particular, for any $0 \leq a \leq \lambda$, we will need a bound on the number of important level-h compactions represented by i-nodes for $i \in [a, \lambda]$.

To state the bound, 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; note that compactions involving important items are a superset of important compactions. Let Q_h be the set of nodes t such that (i) t is an i-node for $i \leq \lambda$ that represents a level-h compaction involving important items (this compaction may or may not be important), and (ii) there is no node t' on the path from the parent of t to the topmost λ-node containing t in its subtree such that t' represents a level-h compaction involving important items. Intuitively, Q_h captures "maximal" nodes (disregarding i-nodes for $i > \lambda$, if any) that represent a level-h compaction removing one or more important items from level h. Note 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 during compactions represented by i-nodes for $i \leq \lambda$, by the definition of Q_h. For $i \in [0, \lambda]$, let Q_h^i be the set of i-nodes in Q_h.

For some $0 \leq a \leq \lambda$, let $R_h^{[a,\lambda]}(y)$ be the number of important items that are either (i) removed from level h during a compaction represented by an i-node for $i \in [a, \lambda]$, or (ii) remain at the level-h buffer of the sketch represented by a node $t \in Q_h^i$ for $i \in [a, \lambda]$ (after the compaction operation represented by t is performed). Note that important items in (ii) also belong to the level-h buffer represented by a topmost λ-node since the level-h buffer is not subject to a compaction that removes an important item and is represented by a node on the path from $t \in Q_h^i$ to its corresponding topmost λ-node, by the definition of Q_h^i. We remark that the level-h buffers represented by topmost λ-nodes may contain important items not present in the level-h buffers represented by nodes in Q_h (these are items promoted from level $h - 1$ to level h during merge operations represented by nodes on the path from a node $t \in Q_h$ to a topmost λ-node).

We now state the bound on the number of important level-h compactions represented by i-nodes for $i \leq \lambda$. Let m_h^i be the number of important compaction operations at level h represented by i-nodes.

LEMMA 6.4. *For any level h and any $0 \leq a \leq \lambda$, it holds that*

$$\sum_{i=a}^{\lambda} m_h^i \cdot k_i \leq 4 \, R_h^{[a,\lambda]}(y) \,. \tag{20}$$

Proof overview. The proof is an extension of the charging argument in Lemma 3.1 to the mergeability setting. In a nutshell, we will again charge each important compaction represented by an i-node for some $a \leq i \leq \lambda$ to k_i important items that are removed from the level-h buffer (during a compaction represented by an i'-node for some $i \leq i' \leq \lambda$) or that remain in the level-h buffer represented by a node in $Q_h^{i'}$ for $i \leq i' \leq \lambda$. However, unlike in the streaming setting, we will not identify specific important items to which we charge an important compaction.

Instead, for each node t in the subtree of a node in Q_h, we will maintain the overall charge from t's subtree to the (level-h) buffer represented by t. Intuitively, when two buffers are merged during the merge procedure represented by an i-node t for $a \leq i \leq \lambda$, the charge to the resulting buffer is the sum of the charges to the two buffers increased or decreased by the following:

- when the level-h compaction represented by i-node t (if any) is important, we increase the charge to the buffer by k_i,
- removing r important items during the compaction operation (not necessarily important) decreases the charge to the buffer by $3r$, and
- if a child t' of t is a topmost i'-node for $i' < i$ such that there is an important compaction represented by an i'-node in the subtree of t', we decrease by $2k_i$ the charge in the buffer represented by t (not by t').

The latter decrease helps us to deal with merge operations in which parameters k and B of the level-h buffer change (in particular, k_i decreases and therefore, we need to create a slack in the analysis). We prove below that (i) the charge to any buffer is always bounded by the number of important items in the buffer and that (ii) these properties imply (20), proving the lemma. Showing (ii) is not difficult given (i); the only non-trivial part is bounding the total decrease of the charge from the third bullet above, which is done in the parents of topmost i-nodes.

Proving (i) relies on the compaction schedule. We in particular show that for each i-node t either there is slack at t, i.e., the charge to t is smaller by at least k_i than the number of important items in the level-h buffer represented by t, or the schedule state C guarantees that at least k_i important items would be removed if a compaction is executed.[9]

PROOF OF LEMMA 6.4. For simplicity, when we refer to a buffer or a compaction operation represented by a node we implicitly mean the one at level h. For any node t in the subtree of a node in Q_h, we define its *charge* $\chi(t)$ (implicitly w.r.t. item y and level h) recursively as follows:

- If t is a leaf node or an i-node for $i < a$, we set $\chi(t) = 0$.
- Otherwise, let t' and t'' be the children of t and let $i \in [a, \lambda]$ be such that t is an i-node. To define $\chi(t)$, we need a few quantities and indicators:

  $r(t) = $ the number of important items removed from the buffer during the compaction represented by t (we use $r(t) = 0$ if there is no compaction operation represented by t);

  $I(t)$ is the indicator whether the compaction represented by t (if any) is important, i.e., $I(t) = 1$ if there is an important compaction represented by t, and $I(t) = 0$ otherwise; and

  $J(t)$ is the indicator whether for a child $\hat{t} \in \{t', t''\}$ of t, it holds that $\hat{t}$ is a topmost i'-node for some $a \leq i' < i$ and there is an important level-h compaction represented by an i'-node in the subtree of $\hat{t}$.

---

[9] A somewhat simpler but weaker proof of the lemma appears in the previous version of this manuscript; see https://arxiv.org/abs/2004.01668v3. However, this earlier analysis required a modified (and slightly more involved) merge procedure.

Then, we define

$$\chi(t) = \max\{\chi(t') + \chi(t'') - 3r(t) + I(t) \cdot k_i - J(t) \cdot 2 \cdot k_i, 0\}. \tag{21}$$

(We do not define $\chi(t)$ for nodes that are not in the subtree of a node in Q_h.) This recursive definition implies that $\chi(t) > 0$ only if there is an important compaction represented by an i'-node (for $a \le i' \le i$ with $i' \le \lambda$) in the subtree of t, including t (the converse may not be true). The key part is to prove that $\chi(t)$ as defined above is always bounded by the number of important items in the buffer represented by t.

CLAIM 6.5. *For any node t in the subtree of a node in Q_h, it holds that $\chi(t) \le R(y; \mathcal{B}_h(t))$, where $\mathcal{B}_h(t)$ is the level-h buffer represented by t and $R(y; \mathcal{B}_h(t))$ is the number of important items in that buffer.*

PROOF. We start with some notation. Let $C_h(t)$ be the state of the compaction schedule of the level-h buffer represented by a node t, and for a state C and $j \ge 1$, let $C[j]$ be the j-th bit from the right in the binary representation of C.

We prove by an induction over the tree T a stronger claim: *If t is an i-node for $a \le i \le \lambda$ in the subtree of a node in Q_h, then one of the following holds:*

(i) $\chi(t) \le \max\{R(y; \mathcal{B}_h(t)) - k_i, 0\}$, *or*

(ii) *there is an important level-h compaction represented by an i-node in the subtree of t and moreover, letting $j \ge 1$ be the smallest index of a section which contains important items only, $\chi(t) \le B_i - (j-1) \cdot k_i$ and, provided that $j > 1$, $C_h(t)[j-1] = 1$.*

Note that both (i) and (ii) are stronger requirements than $\chi(t) \le R(y; \mathcal{B}_h(t))$; specifically, in (ii), it holds that $R(y; \mathcal{B}_h(t)) \ge B_i - (j-1) \cdot k_i$ by the definition of j (recall that sections are indexed from the right).

The claim in (i) clearly holds if $\chi(t) = 0$ and thus, (i) holds for any leaf node or for any i-node t for $i < a$ as we define $\chi(t) = 0$ in both of these cases.

Consider a non-leaf i-node t with $a \le i \le \lambda$ and $\chi(t) > 0$, and let t' and t'' be the children of t. Note that $R(y; \mathcal{B}_h(t)) \ge R(y; \mathcal{B}_h(t')) + R(y; \mathcal{B}_h(t'')) - r(t)$; on the RHS of this inequality, we do not take into account important items added from level $h - 1$ during a compaction represented by t, if any. We consider several cases, using the first case that applies:

Case A: $r(t) \ge k_i$, i.e., the compaction operation represented by t removes at least k_i important items from the level-h buffer. Then, from (21), we obtain

$$\begin{aligned}
\chi(t) &= \max\{\chi(t') + \chi(t'') - 3r(t) + I(t) \cdot k_i - J(t) \cdot 2 \cdot k_i, 0\} \\
&\le \max\{\chi(t') + \chi(t'') - r(t) - k_i, 0\} \\
&\le \max\{R(y; \mathcal{B}_h(t')) + R(y; \mathcal{B}_h(t'')) - r(t) - k_i, 0\} \le \max\{R(y; \mathcal{B}_h(t)) - k_i, 0\},
\end{aligned}$$

where the first inequality follows from the case condition and $I(t) \le 1$ (that is, we use that $2r(t) \ge I(t) \cdot k_i + k_i$), and the second inequality uses the induction hypothesis. This shows that (i) holds for t.

Case B: $\chi(t') = 0$ and $\chi(t'') = 0$. If the compaction operation represented by t (if any) is not important, then $\chi(t) = 0$ and (i) holds for t. Otherwise, there is an important compaction represented by t, which may happen if many important items are added to level h during the level-$(h-1)$ compaction. Then, (21) and $\chi(t') = \chi(t'') = 0$ imply that

$$\chi(t) \le k_i \le B_i/2 - k_i \le R(y; \mathcal{B}_h(t)) - k_i,$$

where the second inequality uses $B_i \ge 2 \cdot k_i \cdot \log_2(N_i/k_i)$ and $N_i \ge 4 \cdot k_i$, and the last inequality follows from that there must be at least $B_i/2$ important items remaining in the buffer after the important compaction represented by t. Hence, (i) holds for t.

Case C: (i) holds for t' and $\chi(t') > 0$, or (i) holds for t'' and $\chi(t'') > 0$, or both. This condition implies that

$$\chi(t') + \chi(t'') \le \mathrm{R}(y; \mathcal{B}_h(t')) + \mathrm{R}(y; \mathcal{B}_h(t'')) - k_i; \tag{22}$$

note that t' or t'' may be an i'-node for some $i' < i$, but this inequality still holds as $k_i \le k_{i'}$ for $i' < i$. We consider two subcases:

Case C.1: $I(t) = 0$, i.e., there is no important compaction represented by t. Then, (i) holds for t as

$$\chi(t) \le \max\{\chi(t') + \chi(t'') - 3r(t), 0\} \le \max\{\mathrm{R}(y; \mathcal{B}_h(t')) + \mathrm{R}(y; \mathcal{B}_h(t'')) - k_i - r(t), 0\}$$
$$\le \max\{\mathrm{R}(y; \mathcal{B}_h(t)) - k_i, 0\}.$$

Case C.2: $I(t) = 1$, i.e., there is an important compaction represented by t. In this case, we show that (ii) holds for t. Since the (level-h) compaction represented by t is important, it removes $0 < r(t) < k_i$ important items from the buffer (for $r(t) \ge k_i$, case A applies). Let j be the smallest index of a section that contains important items only; it must be the same before and after the compaction as $r(t) < k_i$ and as only the whole sections are compacted. Note that we must have $j > 1$ as section 1 is involved in any compaction. Since the compaction does not involve section j, we have $C'_h(t)[j-1] = 0$ for the state $C'_h(t)$ before the compaction (recall that $C'_h(t)[j-1]$ is the $(j-1)$-st bit from the right in $C'_h(t)$). Moreover, $C'_h(t)[j'] = 1$ for all $0 < j' < j - 1$ as the compaction involves section $j - 1$. Thus, after the compaction, it holds that $C_h(t)[j-1] = 1$. Next, observe that $\mathrm{R}(y; \mathcal{B}_h(t)) = B_i - (j-1) \cdot k_i$ since the compaction involves the first $j - 1$ sections and it is important. It thus remains to obtain a suitable upper bound on $\chi(t)$:

$$\chi(t) \le \max\{\chi(t') + \chi(t'') - 3r(t) + k_i, 0\} \le \max\{\mathrm{R}(y; \mathcal{B}_h(t')) + \mathrm{R}(y; \mathcal{B}_h(t'')) - k_i - r(t) + k_i, 0\}$$
$$\le \mathrm{R}(y; \mathcal{B}_h(t)) = B_i - (j-1) \cdot k_i,$$

where the second inequality uses (22). Hence, (ii) holds for t.

Case D: None of the previous cases applies. Since we have that $\chi(t') > 0$ or $\chi(t'') > 0$ (as case B does not apply) and since case C does not apply, property (ii) holds for t' or for t'' or for both of them. For simplicity, we assume that (ii) holds for t' as the other case follows by symmetric arguments. Let $i' \le i$ be such that t' is an i'-node and let j' be the index from property (ii) for t'. To recall, $j' \ge 1$ is the smallest index of a section which contains important items only in $\mathcal{B}_h(t')$, and it holds that $\chi(t') \le B_{i'} - (j'-1) \cdot k_{i'}$ and, provided that $j' > 1$, $C_h(t')[j'-1] = 1$. Let $C'_h(t)$ be the state of the compaction schedule just before the compaction represented by t. Since we use the bitwise OR when merging states of the compaction schedule, we also have that $C'_h(t)[j'-1] = 1$ if $j' > 1$; see Fact 6.1.

We consider a few further subcases:

Case D.1: $i' < i$. Thus, t' is a topmost i'-node, which together with property (ii) for t' implies that $J(t) = 1$ (here, we also use that t is in the subtree of a node in Q_h). Then, (21) becomes

$$\chi(t) = \max\{\chi(t') + \chi(t'') - 3r(t) + I(t) \cdot k_i - 2 \cdot k_i, 0\}$$
$$\le \max\{\chi(t') + \chi(t'') - r(t) - k_i, 0\}$$
$$\le \max\{\mathrm{R}(y; \mathcal{B}_h(t')) + \mathrm{R}(y; \mathcal{B}_h(t'')) - r(t) - k_i, 0\} \le \max\{\mathrm{R}(y; \mathcal{B}_h(t)) - k_i, 0\},$$

where the second inequality uses the induction hypothesis for t' and t'', namely, that $\chi(t') \le \mathrm{R}(y; \mathcal{B}_h(t'))$ and $\chi(t'') \le \mathrm{R}(y; \mathcal{B}_h(t''))$. This shows (i).

Case D.2: $i' = i$. We show that $\chi(t'') = 0$ in such a case. Indeed, for a contradiction suppose that $\chi(t'') > 0$, which implies that property (ii) holds for t'' since otherwise, case C applies. Then, if t'' is an i'-node for $i' < i$, we use case D.1 with t'' acting as t'. Thus, t'' is an i-node and (ii) holds for both t' and t'', from which we obtain $\mathrm{R}(y; \mathcal{B}_h(t')) \ge B_i/2$ and $\mathrm{R}(y; \mathcal{B}_h(t'')) \ge B_i/2$ (as there is an important compaction represented by an i-node in the subtree of each of t' and t''). It follows that $\mathrm{R}(y; \mathcal{B}_h(t')) + \mathrm{R}(y; \mathcal{B}_h(t'')) \ge B_i$ and since all items with position

at least $B_i - k_i + 1$ in the sorted buffer when merging (cf. line 20 in Algorithm 4) are always involved in the compaction, we must have that $r(t) \geq k_i$ — a contradiction with the assumption that case A does not apply. This shows $\chi(t'') = 0$.

Let $a(t)$ be the number of important items added to the level-h buffer from level $h - 1$ during the level-$(h - 1)$compaction represented by t, if any. Let $\mathcal{B}'_h(t)$ be the sorted buffer obtained from merging the (level-h) buffers of t' and t'' and adding $a(t)$ important items from level $h-1$, but before performing the level-h compaction represented by t, if any; thus $\mathcal{B}'_h(t)$ may contain more than B_i items.

Note that section j' is not involved in the level-h compaction represented by t (if any), otherwise we would have $r(t) \geq k_i$ as section j' contains important items only in $\mathcal{B}_h(t')$ and thus also in $\mathcal{B}'_h(t)$. This implies that section $j' - 1$ is not involved in the compaction either, which follows from $C'_h(t)[j' - 1] = 1$ (here, we refer to the state just before the compaction). We consider a few further subcases:

Case D.2.a: $R(y; \mathcal{B}_h(t')) + R(y; \mathcal{B}_h(t'')) + a(t) < B_i - (j' - 2) \cdot k_i$. This means that in $\mathcal{B}'_h(t)$, section $j' - 2$ (for $j' > 2$) contains no important items and moreover, section $j' - 1$ contains an item $z > y$ (we suppose buffer $\mathcal{B}'_h(t)$ is sorted). Since section $j' - 1$ is not involved in the compaction, it follows that $r(t) = 0$, so the compaction represented by t (if any) is not important, i.e., $I(t) = 0$. As $\chi(t'') = 0$, we get

$$\chi(t) \leq \chi(t') \leq B_i - (j' - 1) \cdot k_i.$$

We show (ii) holds for t. Indeed, section $j' - 1$ of $\mathcal{B}_h(t)$ contains a non-important item, so j' is still the smallest index of a section with important items only. Furthermore, $C_h(t)[j' - 1] = 1$ and there is an important level-h compaction represented by an i-node in the subtree of t' as (ii) holds for t', concluding that (ii) holds for t.

Case D.2.b: $R(y; \mathcal{B}_h(t')) + R(y; \mathcal{B}_h(t'')) + a(t) \geq B_i - (j' - 2) \cdot k_i$ and $I(t) = 0$, i.e., the compaction represented by t (if any) is not important. As section $j' - 1$ is not involved in this compaction, we have that $R(y; \mathcal{B}_h(t)) \geq B_i - (j' - 2) \cdot k_i$. Then, (i) holds for t as

$$\chi(t) \leq \chi(t') \leq B_i - (j' - 1) \cdot k_i \leq R(y; \mathcal{B}_h(t)) - k_i.$$

Case D.2.c: $R(y; \mathcal{B}_h(t')) + R(y; \mathcal{B}_h(t'')) + a(t) \geq B_i - (j' - 2) \cdot k_i$ and $I(t) = 1$, i.e., the compaction represented by t is important. We show that (ii) holds. Let $j > 1$ be the smallest index of a section that contains important items only in $\mathcal{B}_h(t)$, i.e., after the compaction. By the case condition and since section $j' - 1$ is not involved in the compaction, we have $j \leq j' - 1$. Observe that $R(y; \mathcal{B}_h(t)) = B_i - (j - 1) \cdot k_i$ as the compaction removes important items from the buffer and thus, it involves the first $j - 1$ sections by the definition of j. After the compaction, the $(j - 1)$-st bit of the state is set to 1, i.e., $C_h(t)[j - 1] = 1$, by the definition of the compaction. Finally, we upper bound $\chi(t)$ as follows:

$$\chi(t) \leq \chi(t') + k_i \leq B_i - (j' - 1) \cdot k_i + k_i = B_i - (j' - 2) \cdot k_i \leq B_i - (j - 1) \cdot k_i,$$

where the last inequality uses $j \leq j' - 1$. Hence, (ii) holds. □

It remains to show that Claim 6.5 together with the definition of $\chi(t)$ in (21) implies Lemma 6.4, i.e., that (20) holds. To this end, first note that the definition of $\chi(t)$ for a non-leaf i-node t with $a \leq i \leq \lambda$ implies

$$\chi(t) \geq \chi(t') + \chi(t'') - 3r(t) + I(t) \cdot k_i - J(t) \cdot 2 \cdot k_i, \tag{23}$$

where t' and t'' are the children of t. For a node $q \in Q_h^j$ with $a \leq j \leq \lambda$, consider the sum of (23) over all non-leaf i-nodes t for $a \leq i \leq j$ such that t is in the subtree of q, and observe that $\chi(t)$ either appears exactly once on both sides of the resulting inequality, or t appears only on the right-hand side and $\chi(t) = 0$, or $t = q$ and q appears

only on the left-hand side. Letting T_q denote the subtree of q and T_q^i be the set of i-nodes in T_q, we obtain

$$\chi(q) \geq \sum_{i=a}^{j} \sum_{t \in T_q^i} -3r(t) + I(t) \cdot k_i - J(t) \cdot 2 \cdot k_i \,. \tag{24}$$

Next, consider the sum of (24) over all nodes $q \in Q_h^j$ for $a \leq j \leq \lambda$. Observe that if an i-node t for $a \leq i \leq \lambda$ represents a compaction removing at least one important item, then t must be in the subtree T_q of a node $q \in Q_h^j$ for $a \leq j \leq \lambda$. Furthermore, subtrees T_q are disjoint by the definition of Q_h. Letting $r(T^{[a,\lambda]})$ be the total number of important items removed from level h during a compaction represented by an i-node for $a \leq i \leq \lambda$ that is in the subtree of a node in Q_h, we thus have the following two equalities:

$$\sum_{j=a}^{\lambda} \sum_{q \in Q_h^j} \sum_{i=a}^{j} \sum_{t \in T_q^i} r(t) = r(T^{[a,\lambda]})$$

$$\sum_{j=i}^{\lambda} \sum_{q \in Q_h^j} \sum_{t \in T_q^i} I(t) = m_h^i \quad \text{for any } i \in [a,\lambda] \,.$$

Hence, summing (24) over all nodes $q \in Q_h^j$ for $a \leq j \leq \lambda$, we get

$$\sum_{j=a}^{\lambda} \sum_{q \in Q_h^j} \chi(q) \geq -3 \cdot r(T^{[a,\lambda]}) + \sum_{i=a}^{\lambda} m_h^i \cdot k_i - \sum_{j=a}^{\lambda} \sum_{q \in Q_h^j} \sum_{i=a}^{j} \sum_{t \in T_q^j} J(t) \cdot 2 \cdot k_i \,. \tag{25}$$

We now upper bound the last term on the RHS of (25). Let $\tau_{i'}$ be the number of topmost i'-nodes t' for $i' \leq \lambda$ satisfying that there is an important level-h compaction represented by an i'-node in the subtree of t' and that t' is in the subtree of a node $q \in Q_h^j$ for $a \leq j \leq \lambda$. Recall that if $J(t) = 1$ for an i-node t, then at least one of the children of t is a topmost i'-node for $a \leq i' < i$ accounted for in $\tau_{i'}$. Using $k_0 \geq k_1 \geq \cdots \geq k_\lambda$, we thus have

$$\sum_{j=a}^{\lambda} \sum_{q \in Q_h^j} \sum_{i=a}^{j} \sum_{t \in T_q^j} J(t) \cdot 2 \cdot k_i \leq \sum_{i'=a}^{\lambda} \tau_{i'} \cdot 2 \cdot k_{i'} \,. \tag{26}$$

For any $i' \in [a,\lambda]$, we claim that

$$\tau_{i'} \cdot \frac{B_{i'}}{2} \leq \mathrm{R}_h^{[a,\lambda]}(y) \,. \tag{27}$$

Indeed, any topmost i'-node t' accounted for in $\tau_{i'}$ has an important level-h compaction represented by an i'-node in the subtree of t'. At the time of this compaction operation, the buffer needs to have more than $B_{i'}/2$ important items (otherwise, the compaction would not be important). Since the lowest-ranked $B_{i'}/2$ important items are never removed from the buffer (when its capacity is $B_{i'}$), the buffer represented by t' has at least $B_{i'}/2$ important items. Furthermore, these sets of at least $B_{i'}/2$ important items are disjoint for any two topmost i'-nodes $t' \neq t''$ accounted for in $\tau_{i'}$. Finally, all these important items are accounted for in $\mathrm{R}_h^{[a,\lambda]}(y)$ as they are either removed from the level-h buffer by a compaction represented by an i''-node for $i' \leq i'' \leq \lambda$, or remain at the level-h buffer represented by a node $q \in Q_h^j$ for $i' \leq j \leq \lambda$. This shows (27).

Thus, the last term on the RHS of (25) is bounded by

$$\sum_{j=a}^{\lambda} \sum_{q \in Q_h^j} \sum_{i=a}^{j} \sum_{t \in T_q^j} J(t) \cdot 2 \cdot k_i \le \sum_{i'=a}^{\lambda} \tau_{i'} \cdot 2 \cdot k_{i'}$$

$$\le 2 \cdot \sum_{i'=a}^{\lambda} \frac{\tau_{i'} \cdot \frac{1}{2} \cdot B_{i'}}{\log_2(N_{i'}/k_{i'})} \le 2 \cdot \sum_{i'=a}^{\lambda} \frac{R_h^{[a,\lambda]}(y)}{\log_2(N_{i'}/k_{i'})} \le R_h^{[a,\lambda]}(y), \qquad (28)$$

where the first inequality is (26), the second inequality uses the definition of $B_{i'}$ in (19), the third inequality follows from (27), and the last inequality holds as $\sum_{i'=a}^{\lambda} 1/\log_2(N_{i'}/k_{i'}) \le 2\log_2(N_0/k_0) \le \frac{1}{2}$ (in more detail, here we use that $\log_2(N_{i'}/k_{i'})$ increases with i' by a factor of at least 2 for $i' \le \lambda$ and that $\log_2(N_0/k_0) \ge 4$, which holds by the definition of N_0).

To upper bound the LHS of (25), we use Claim 6.5 for each $q \in Q_h^j$ with $a \le j \le \lambda$ to get that $\chi(q) \le R(y; \mathcal{B}_h(q))$ for any such q. Plugging this together with (28) into (25), we obtain

$$\sum_{j=a}^{\lambda} \sum_{q \in Q_h^j} R(y; \mathcal{B}_h(q)) \ge -3 \cdot r(T^{[a,\lambda]}) + \sum_{i=a}^{\lambda} m_h^i \cdot k_i - R_h^{[a,\lambda]}(y), \qquad (29)$$

which implies (20) by rearranging and using $R_h^{[a,\lambda]}(y) = r(T^{[a,\lambda]}) + \sum_{j=a}^{\lambda} \sum_{q \in Q_h^j} R(y; \mathcal{B}_h(q))$ (the second term equals the total number of important items in the buffers represented by nodes in Q_h^j for $a \le j \le \lambda$). □

Lemma 6.4 with $a = 0$ has a simple corollary.

COROLLARY 2. *Consider level h and let $m_h^{\le \lambda} = \sum_{i=0}^{\lambda} m_h^i$ be the total number of important compactions at level h represented by i-nodes for $i \le \lambda$. Suppose that $R_h(y) \le 2^{-h+2} R(y)$ and let $i(h) \ge 0$ be the largest integer $0 \le i \le \lambda$ satisfying $2^{-h+2} R(y) > B_i/2$. Then $m_h^{\le \lambda} \le 4 R_h(y)/k_{i(h)}$.*

PROOF. This follows from Lemma 6.4 by observing that $m_h^i = 0$ for $i(h) < i \le \lambda$ and by using $k_i \ge k_{i(h)}$ for any $i \le i(h)$ and $R_h^{[0,\lambda]}(y) \le R_h(y)$ □

6.4 Analysis of the Full Sketch for Mergeability

In this section, we complete the proof of full mergeability that matches our result in the streaming setting (Theorem 3). The crucial part of analyzing the full sketch, similarly as in the streaming setting (Section 4), is bounding the variance of $\mathrm{Err}(y)$, using the bounds on the number of important level-h compactions from the previous section. The bound of this section is, however, substantially more involved than in the streaming setting, mainly because parameters k and B of the sketches change as merge operations are processed. Here, we again stress that we assume no advance knowledge of n, the total size of the input.

Before presenting the most general and tight analysis, we will however describe that a simple extension of the arguments used in the streaming setting readily gives the result with an additional factor of $\min\{\log\log(\varepsilon n), \log(\varepsilon^{-1}) + \log\log(\delta^{-1})\}$ in the asymptotic space complexity, relative to our result in the streaming setting (Theorem 3).[10] This simpler, non-tight analysis of the full sketch is less delicate than our analysis that avoids the additional factor, thereby establishing Theorem 1. We nevertheless do not assume any advance knowledge about the final input size n.

---

[10]We provide a detailed description of a simpler analysis with an additional $\log\log(\varepsilon n)$ factor in a prior version of this manuscript; see https://arxiv.org/abs/2004.01668v3.

6.4.1 A Sketch of a Simpler Analysis with an Additional Double Logarithmic Factor. The key trick that allows to apply similar arguments as in Section 4 is to modify the definition of k_i for $i \geq 0$ compared to (19), as follows:

$$k_i = \Theta(1) \cdot \left\lceil \frac{\min\{i+1, \lambda\} \cdot \hat{k}}{\sqrt{\log_2(\overline{N_i}/\hat{k})}} \right\rceil, \tag{30}$$

where λ and $\overline{N_i}$ are defined similarly as in (19) and the multiplicative constant is set appropriately. In particular, relative to Equation (19), note the extra factor of $\min\{i+1, \lambda\}$; including it considerably simplifies the analysis, but it is responsible for an additional $\min\{\log\log(\varepsilon n), \lambda\}$ term in the space bound, where $\lambda = O(\log(\varepsilon^{-1}) + \log\log(\delta^{-1}))$. Recall that $B_i = 2 \cdot k_i \cdot \left\lceil \log_2(\overline{N_i}/k_i) \right\rceil$.

We omit the detailed analysis and only highlight where we use the modified definition of the parameter k_i. As in the subsequent tight analysis, the error from compactions represented by i-nodes for $i > \lambda$ (if any) will be analyzed separately (and is much simpler to deal with). In particular, a similar calculation as in (7) gives us that for $0 \leq i \leq \lambda$,

$$k_i \cdot B_i \geq \Theta(1) \cdot \frac{(i+1)^2}{\varepsilon^2} \cdot \ln \frac{1}{\delta}, \tag{31}$$

so we have an extra factor of $(i+1)^2$ compared to (7). Using Corollary 2, one can show that

$$\mathrm{Var}[\mathrm{Err}(y)] \leq \sum_{i=0}^{\lambda} \Theta(1) \cdot \frac{\mathrm{R}(y)^2}{k_i \cdot B_i} \leq \frac{\varepsilon^2 \cdot \mathrm{R}(y)^2}{4 \cdot \ln(1/\delta)} \cdot \sum_{i=0}^{\lambda} \frac{1}{(i+1)^2} \leq \frac{\varepsilon^2 \cdot \mathrm{R}(y)^2}{2 \cdot \ln(1/\delta)},$$

where the second inequality uses (31) and the last step holds as $\sum_{i=0}^{\lambda} 1/(i+1)^2 < \pi^2/6 < 2$; the fact that this sum is bounded allows us to deal with the challenge of changing parameters k and B in a simple way. The application of the tail bound for sub-Gaussian variables and the derivation of the space bound is otherwise the same as in Theorem 3.

6.4.2 A Tight Analysis. Recall that $\hat{k} = 4\varepsilon^{-1} \cdot \sqrt{\ln(1/\delta)}$ and that by (19), $k_i = 2^5 \cdot \left\lceil \hat{k}/\sqrt{\log_2(\overline{N_i}/\hat{k})} \right\rceil$ and $B_i = 2 \cdot k_i \cdot \left\lceil \log_2(\overline{N_i}/k_i) \right\rceil$, where $\overline{N_i} = \min\{N_i, N_\lambda\}$ by (19). Here, $\lambda \geq 0$ is the smallest integer i such that $\hat{k}/\sqrt{\log_2(N_i/\hat{k})} \leq 1$. If $\lambda > \ell$, we decrease λ to ℓ for convenience. Using a similar calculation as in Claim 4.1, we show a lower bound on $k_i \cdot B_i$.

CLAIM 6.6. *Parameters k_i and B_i set according to* (19) *satisfy*

$$k_i \cdot B_i \geq 2^{14} \cdot \frac{1}{\varepsilon^2} \cdot \ln \frac{1}{\delta}. \tag{32}$$

PROOF. We first need to relate $\log_2(\overline{N_i}/k_i)$ (used to define B_i) and $\log_2(\overline{N_i}/\hat{k})$ (that appears in the definition of k_i). As $k_i \leq 2^5 \cdot \hat{k}$, it holds that $\log_2(\overline{N_i}/k_i) \geq \log_2(\overline{N_i}/\hat{k}) - 5 \geq \log_2(\overline{N_i}/\hat{k})/2$, where we use that $\overline{N_i} \geq N_0 \geq 2^{10} \cdot \hat{k}$, so $\log_2(\overline{N_i}/\hat{k}) \geq 10$. Using this, we bound $k_i \cdot B_i$ as follows:

$$k_i \cdot B_i = 2 \cdot k_i^2 \cdot \left\lceil \log_2 \frac{\overline{N_i}}{k_i} \right\rceil \geq 2 \cdot 2^{10} \cdot \frac{16}{\varepsilon^2} \cdot \frac{\ln \frac{1}{\delta}}{\log_2(\overline{N_i}/\hat{k})} \cdot \frac{\log_2(\overline{N_i}/\hat{k})}{2} = 2^{14} \cdot \frac{1}{\varepsilon^2} \cdot \ln \frac{1}{\delta}.$$

□

For analyzing the case $n > N_\lambda$, the following bound will be useful:

$$B_\lambda \geq 2^5 \cdot \hat{k}^2 \tag{33}$$

This is because the definition of λ implies that $\sqrt{\log_2(N_\lambda/\hat{k})} \geq \hat{k}$ while $k_\lambda \geq 2^5$, thus $B_\lambda \geq 2^6 \cdot \log_2(N_\lambda/k_\lambda) \geq 2^6 \cdot \log_2\left(N_\lambda/\hat{k}\right)/2 \geq 2^5 \cdot \hat{k}^2$, where the second inequality follows from the same argument as in Claim 6.6.

For any $0 \leq i \leq \ell$, let $H_i(y)$ be the minimal h for which $2^{-h+2} R(y) \leq B_i/2$. 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 $2^{3-H_i} R(y) \geq B_i/2$, or equivalently

$$2^{H_i} \leq 2^4 \cdot R(y)/B_i. \tag{34}$$

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

PROOF. The proof is similar to that of Lemma 4.4, except that we need to deal with parameters k and B changing over time. To this end, we use an idea from the KLL paper [15] to analyze the top $\log\log 1/\delta$ levels deterministically. We define

$$H_0' = \max\left(0, H_0 - \left\lceil \log_2 \sqrt{\ln \frac{1}{\delta}} \right\rceil\right).$$

(Note that for $\delta \leq 0.5$, we have $\left\lceil \log_2 \sqrt{\ln \frac{1}{\delta}} \right\rceil \geq 0$.)

We first show by induction on $0 \leq h \leq H_0'$ that $R_h(y) \leq 2^{-h+1} R(y)$ with probability at least $1 - \delta \cdot 2^{h-H_0'-1}$, conditioned on $R_{h'}(y) \leq 2^{-h'+1} R(y)$ for any $h' < h$. The base case holds by $R_0(y) = R(y)$.

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 4.4,

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

where $Z_h = \sum_{h'=0}^{h-1} 2^{-h+h'} \cdot \text{Binomial}(m_{h'})$ is a zero-mean sub-Gaussian random variable. To bound the variance of Z_h, first note that for any $h' < h$, since each important compaction needs to remove at least one important item from the buffer, we have that $m_{h'} \leq R_{h'}(y) \leq 2^{-h'+1} \cdot R(y)$, using the assumption that $R_{h'}(y) \leq 2^{-h'+1} \cdot R(y)$. (While this may seem like a very crude bound compared to Lemma 6.4, it is sufficient due to analyzing top levels deterministically and furthermore, it can be used for compactions represented by i-nodes for $i > \lambda$, where we do not use the deterministic compaction schedule.)

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

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

To bound $\Pr[Z_h > 2^{-h} \cdot R(y)]$, we apply the tail bound for sub-Gaussian variables (Fact 4.3) to get

$$
\begin{aligned}
\Pr[Z_h > 2^{-h} \cdot R(y)] &< \exp\left(-\frac{2^{-2h} \cdot R(y)^2}{2 \cdot (2^{-h+1} \cdot R(y))}\right) \\
&= \exp\left(-2^{-h-2} \cdot R(y)\right) \\
&= \exp\left(-2^{H_0-H_0'} \cdot 2^{-h+H_0'-6} \cdot 2^{4-H_0} R(y)\right) \\
&\leq \exp\left(-\sqrt{\ln\frac{1}{\delta}} \cdot 2^{-h+H_0'-6} \cdot B_0\right) \\
&\leq \exp\left(-\ln\frac{1}{\delta} \cdot 2^{-h+H_0'+1}\right) = \delta^{2^{-h+H_0'+1}} \leq \delta \cdot 2^{-H_0'+h-1},
\end{aligned}
$$

where the second inequality uses the definition of H_0' and $2^{4-H_0} R(y) \geq B_0$ by (34), the third inequality follows from $B_0 \geq 2 \cdot k_0 \cdot \log_2(N_0/k_0) \geq 2^7 \cdot \sqrt{\ln(1/\delta)}$, 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 $R_h(y) \leq 2^{-h+1} R(y)$ for any $h \leq H_0'$.

Finally, consider level h with $H_0' < h \leq H_0$ and condition on $R_{H_0'}(y) \leq 2^{-H_0'+1} R(y)$. (In the case $H'(y) = 0$, we have $R_0(y) = R(y)$.) We again proceed by induction and assume that $R_{h'}(y) \leq 2^{-h'+2} \cdot R(y)$ for any $h' < h$. First, we argue that for any h with $H_0' < h \leq H_0$ it holds that $\sum_{i>\lambda} m_{h'}^i = 0$, so we can use Corollary 2. Indeed, it is sufficient to show $R_{H_0'}(y) \leq B_\lambda/2$ as follows:

$$
R_{H_0'}(y) \leq 2^{-H_0'+1} R(y) = 2^{H_0-H_0'} \cdot 2^{-2} \cdot 2^{-H_0+3} R(y) \leq 2\sqrt{\ln\frac{1}{\delta}} \cdot 2^{-2} \cdot B_0 \leq 2^4 \cdot \hat{k}^2 \leq \frac{1}{2} B_\lambda, \tag{35}
$$

where the penultimate inequality uses the definitions of $\hat{k}$ and B_0 in (19) and the last inequality is by (33).

We now observe that for any $H_0' < h' \leq h$, it holds that $R_{h'}(y) \leq \frac{1}{2} \cdot (1+4/k_{i(h')}) \cdot R_{h'-1}(y)$, where $i(h') \leq \lambda$ is the largest integer i satisfying $2^{-h'+3} R(y) > B_i/2$. Indeed, $R_{h'}(y) \leq \frac{1}{2} \cdot (R_{h'-1}(y) + \text{Binomial}(m_{h'-1}))$ (see Equation 9) and $\text{Binomial}(m_{h'-1}) \leq m_{h'-1} \leq 4 R_{h'-1}(y)/k_{i(h')}$ by Corollary 2, using the definition of $i(h')$ and the induction hypothesis for level $h'-1$, i.e., $R_{h'-1}(y) \leq 2^{-h'+3} \cdot R(y)$. That is, regardless of the outcome of the random choices, we always obtain this bound on the rank of an item. By using this deterministic bound for levels $H_0' < h' \leq h$, we get

$$
R_h(y) \leq 2^{-h+H_0'} \cdot R_{H_0'}(y) \cdot \prod_{h'=H_0'+1}^{h} \left(1 + \frac{4}{k_{i(h')}}\right) \leq 2^{-h+H_0'} \cdot 2^{-H_0'+1} \cdot R(y) \cdot \prod_{h'=H_0'+1}^{h} \left(1 + \frac{4}{k_{i(h')}}\right). \tag{36}
$$

It remains to show that the product $\prod_{h'=H_0'+1}^{h} \left(1 + \frac{4}{k_{i(h')}}\right)$ is bounded by 2, which implies $R_h(y) \leq 2^{-h+2} \cdot R(y)$. We first observe that $k_{i(H_0'+1)} \geq k_\lambda \geq 2^5$, since $i(H_0' + 1) \leq \lambda$. Next, recall that the sequence of k_i's decreases exponentially with a factor of $\sqrt{2}$ (up to rounding) with increasing i. Thus, it is sufficient to show that the sequence $i(h')$ decreases for $h' = H_0' + 1, \ldots, h$. More precisely, we show that $i(h' + 1) \leq i(h') - 1$ for $h' = H_0' + 1, \ldots, h-1$ This latter inequality holds as increasing h' by one in $2^{-h'+3} R(y) > B_i/2$ implies that the largest i satisfying the inequality should decrease by at least one (recall that the sequence of B_i's increases by a factor of $\sqrt{2}$ (up to rounding) with increasing i). Note that we always have $2^{-h'+3} R(y) > B_0/2$ as $h' \leq h \leq H_0$. Summing up, we get

$$
\prod_{h'=H_0'+1}^{h} \left(1 + \frac{4}{k_{i(h')}}\right) \leq \prod_{j \geq 0}\left(1 + \frac{1}{2^3 \cdot \sqrt{2}^j}\right) \leq \exp\left(\sum_{j \geq 0} \log\left(1 + \frac{1}{2^3 \cdot \sqrt{2}^j}\right)\right) \leq \exp\left(\sum_{j \geq 0} \frac{1}{2^3 \cdot \sqrt{2}^j}\right) \leq 2.
$$

We remark that the last inequality has a slack, which is sufficient to deal with the rounding issues mentioned above. □

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). Recall from Section 6.3 that a compaction *involves important items* iff it removes at least one important item from the buffer. Recall 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 the important compactions.

LEMMA 6.8. *Conditioned on the bounds in Lemma 6.7 holding, for any $0 \le i \le \ell$, 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 6.7, $R_{H_i}(y) \le 2^{-H_i+2} R(y) \le B_i/2$, where the second inequality follows from the definition of H_i. Hence, no important item is ever removed from level H_i during merge operations represented by i-nodes when the buffer size is B_i. The same argument also works for any level $h > H_i$. □

Consider level h. Recall from Section 6.3 that Q_h is the set of nodes t such that (i) t is an i-node for $i \le \lambda$ that represents a level-h compaction involving important items (this compaction may or may not be important), and (ii) there is no node t' on the path from the parent of t to the topmost λ-node containing t in its subtree such that t' represents a level-h compaction involving important items. Intuitively, Q_h captures "maximal" nodes (with index $i \le \lambda$) that represent a level-h compaction removing one or more important items from level h. Note 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 $0 \le i \le \lambda$, let Q_h^i be the set of i-nodes in Q_h and let $q_h^i = |Q_h^i|$.

Note that $q_h^i = 0$ for $h \ge H_0$ by Lemma 6.8 (conditioned on the bounds in Lemma 6.7 holding). Now we observe that values q_h^i for $i = 0, \ldots, \lambda$ 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_h^i contains at most B_i items.

OBSERVATION 6.9. *For any $h \ge 0$ and $0 \le g \le \lambda$, the level-h buffers of the sketches represented by nodes in Q_h^i for some $i \ge g$ contain at most $\sum_{i=g}^{\lambda} q_h^i \cdot B_i$ important items in total (after performing compaction operations represented by these nodes).*

Next, in Observation 6.10, we show that the q_h^i values can as well be used to lower bound the total number of important items at level h in topmost λ-nodes. Combined with Lemma 6.11, this will give us a useful bound on $\sum_{h \ge 0} \sum_{i=0}^{\lambda} 2^h \cdot q_h^i \cdot B_i$ at the very end of the analysis.

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 $0 \le i \le \lambda$, let z_h^i 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 $z_0^i = 0$ for any i.

OBSERVATION 6.10. *For any level h, the level-h buffers of topmost λ-nodes contain at least $\sum_{i=0}^{\lambda} q_h^i \cdot B_i/2 + z_h^i$ important items.*

PROOF. Consider an i-node $t \in Q_h^i$ 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_h^i 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. □

Note that using Observation 6.10, the values of $\sum_{i=0}^{\lambda} q_h^i \cdot B_i/2 + z_h^i$ give a lower bound on the rank of y estimated by the topmost λ-nodes (if $\ell = \lambda$, then the only topmost λ-node is the root of the merge tree T). We now complement it with an upper bound showing that the rank of y estimated by the topmost λ-nodes cannot be too far from $R(y)$. This can be seen as an initial bound on the error which will be used within the proof of the final, more refined bound on the variance of $\text{Err}(y)$.

LEMMA 6.11. *Conditioned on the bounds in Lemma 6.7 holding, with probability at least $1 - \delta$ it holds that*

$$\sum_{i=0}^{\lambda} \sum_{h'=0}^{H_i} 2^{h'} \cdot \left(q_{h'}^i \cdot \frac{B_i}{2} + z_{h'}^i \right) \leq 2\,R(y) \tag{37}$$

PROOF. Note that $q_h^i = 0$ for $h \geq H_i$ and that there is no important compaction represented by an i-node at any level $h \geq H_i$ by Lemma 6.8. Let $\text{Err}^{\leq \lambda}(y)$ be the error introduced by compactions represented by i-nodes for $i \leq \lambda$. By Observation 6.10, it is sufficient to show that $\text{Err}^{\leq \lambda}(y) \leq R(y)$. Recall that $\text{Err}^{\leq \lambda}(y)$ is a zero-mean sub-Gaussian random variable. Similarly as in Lemma 6.7, we define

$$H_0' = \max\left(0, H_0 - \left\lceil \log_2 \sqrt{\ln \frac{1}{\delta}} \right\rceil \right).$$

We split $\text{Err}^{\leq \lambda}(y)$, the error of the rank estimate for y, into two parts (we drop the superscript $\leq \lambda$ for simplicity):

$$\text{Err}'(y) = \sum_{h=0}^{H_0'-1} 2^h \cdot \text{Err}_h(y) \quad \text{and} \quad \text{Err}''(y) = \sum_{h=H_0'}^{H_0-1} 2^h \cdot \text{Err}_h(y).$$

Note that $\text{Err}^{\leq \lambda}(y) = \text{Err}'(y) + \text{Err}''(y)$; we bound both these parts by $\frac{1}{2} R(y)$ w.h.p., starting with $\text{Err}'(y)$. If $H_0' = 0$, then clearly $\text{Err}'(y) = 0$. Otherwise, we analyze the variance of the zero-mean sub-Gaussian variable $\text{Err}'(y)$ as follows:

$$\begin{aligned}
\text{Var}[\text{Err}'(y)] &= \sum_{h=0}^{H_0'-1} 2^{2h} \cdot \text{Var}[\text{Err}_h(y)] \\
&\leq \sum_{h=0}^{H_0'-1} 2^{2h} \cdot R_h(y) \\
&\leq \sum_{h=0}^{H_0'-1} 2^{2h} \cdot 2^{-h+2} R(y) \\
&\leq 2^{H_0'+2} \cdot R(y) = 2^{H_0'-H_0+2} \cdot 2^{H_0} \cdot R(y) \leq 2^{H_0'-H_0+6} \cdot \frac{R(y)^2}{B_0} \leq \frac{R(y)^2}{8 \ln \frac{1}{\delta}}
\end{aligned}$$

where the first inequality is using a simple bound of $\text{Var}[\text{Err}_h(y)] \leq R_h(y)$, the second follows from Lemma 6.7, and the fourth inequality uses $2^{H_0} \leq 2^4 \cdot R(y)/B_0$ by (34), and the last inequality follows from the definition of H_0' and $B_0 \geq 2^9 \cdot \sqrt{\ln(1/\delta)}$ by (19). We again apply Fact 4.3 to obtain

$$\Pr\left[\text{Err}'(y) > \frac{1}{2} R(y) \right] < \exp\left(-\frac{R(y)^2 \cdot \ln \frac{1}{\delta} \cdot}{4 \cdot 2 \cdot \frac{1}{8} R(y)^2} \right) = \exp\left(-\ln \frac{1}{\delta} \right) = \delta.$$

Finally, we use deterministic bounds to analyze $\mathrm{Err}''(y)$, using that we only care about i-nodes for $i \leq \lambda$ As in Lemma 6.7, let $i(h) \leq \lambda$ be the largest integer i satisfying $2^{-h+2} \mathrm{R}(y) > B_i/2$. Then

$$\mathrm{Err}''(y) \leq \sum_{h=H_0'}^{H_0-1} 2^h \cdot m_h^{\leq \lambda}$$

$$\leq \sum_{h=H_0'}^{H_0-1} 2^h \cdot \frac{4\,\mathrm{R}_h(y)}{k_{i(h)}} \leq \sum_{h=H_0'}^{H_0-1} 2^h \cdot \frac{2^{-h+4}\,\mathrm{R}(y)}{k_{i(h)}} = \sum_{h=H_0'}^{H_0-1} \frac{2^4 \cdot \mathrm{R}(y)}{k_{i(h)}} \leq \frac{\mathrm{R}(y)}{2},$$

where the second inequality is by Corollary 2, the third by Lemma 6.7, and the last inequality uses that $k_{i(H_0')} \geq 2^5$ and that the values of $k_{i(h)}$ for $h \in [H_0', H_0 - 1]$ increase exponentially with increasing h (by a factor of $\sqrt{2}$), which follows from similar arguments as in the paragraph below (36) in Lemma 6.7. □

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_h^i values so that we can then use Observation 6.10. To this end, we first use Observation 6.9 to bound $\mathrm{R}_h(y)$ in terms of the q_h^i 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 level-h compaction operation that promotes $b \geq 1$ important items removes at most $2b + 1 \leq 3b$ important items from the level-h buffer. Applying this observation together with Observation 6.9, we show by an induction on h that $\mathrm{R}_h^{[0,\lambda]}(y) \leq \sum_{i=0}^{\lambda} \sum_{h' \geq h} 2 \cdot 3^{h'-h} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i)$. Recall that $\mathrm{R}_h^{[a,\lambda]}(y)$ is the number of important items that are either removed from level h during a compaction represented by an i-node for $a \leq i \leq \lambda$, or remain at the level-h buffer represented by a node $t \in Q_h^i$ for $a \leq i \leq \lambda$ (after the compaction operation represented by t is done). Note that this provides alternative rank bounds to Lemma 6.7.

Then we apply Lemma 6.4 to get our variance bound, which however brings additional technical difficulties. To overcome them, we use a careful proof by induction over $g \in [0, \lambda]$. We will only focus on i-nodes with $i \leq \lambda$ and on levels $h \geq H_{\lambda+1}$; the error from remaining nodes and levels will be analyzed later.

LEMMA 6.12. *Conditioned on the bounds in Lemma 6.7 holding, for any $h \geq H_{\lambda+1}$, it holds that*

$$\mathrm{Var}[\mathrm{Err}_h(y)] \leq \sum_{i=0}^{\lambda} \sum_{h' \geq h} \frac{8 \cdot 3^{h'-h} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i)}{k_i} \,. \tag{38}$$

PROOF. We first note that by Lemma 6.8 (conditioned on Lemma 6.7), there is no important compaction at any level $h \geq H_{\lambda+1}$ represented by an i-node for $i > \lambda$. Therefore, our focus will again be solely on i-nodes for $i \leq \lambda$. As outlined above, we first bound $\mathrm{R}_h^{[g,\lambda]}(y)$ for any $0 \leq g \leq \lambda$ and in particular, we prove by a "backward" induction on $h = H, H - 1, \ldots, H_{\lambda+1}$ that the following inequality holds for any fixed $0 \leq g \leq \lambda$:

$$\mathrm{R}_h^{[g,\lambda]}(y) \leq \sum_{i=g}^{\lambda} \left(\sum_{h' \geq h+1} \left(2 \cdot 3^{h'-h} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i) \right) + 2 \cdot q_h^i \cdot B_i \right). \tag{39}$$

At level $h = H$, there is no important compaction, implying that $\mathrm{R}_H^{[g,\lambda]}(y) = 0$ and $q_H^i = 0$ for any i, which establishes the base case.

Consider $h < H$ and suppose that (39) holds for $h + 1$, i.e., we have that

$$\mathrm{R}_{h+1}^{[g,\lambda]}(y) \leq \sum_{i=g}^{\lambda} \left(\sum_{h' \geq h+2} \left(2 \cdot 3^{h'-h-1} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i) \right) + 2 \cdot q_{h+1}^i \cdot B_i \right). \tag{40}$$

To show (39), we first bound the number of important items removed from level h in terms of $\mathrm{R}_{h+1}^{[g,\lambda]}(y)$. For brevity, let $z_{h+1}^{[g,\lambda]} = \sum_{i=g}^{\lambda} z_{h+1}^i$. Note that there are at most $\mathrm{R}_{h+1}^{[g,\lambda]}(y) + z_{h+1}^{[g,\lambda]}$ important items added to level $h + 1$ during compactions represented by i-nodes for some $i \in [g, \lambda]$, 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_{h+1}^i for some $i \in [g, \lambda]$ or is added to level $h + 1$ during a merge operation represented by an i-node t for $i \in [g, \lambda]$ such that t is not in the subtree of a node in Q_{h+1}. Further, observe that each compaction which adds b important items to level $h + 1$ removes at most $2b + 1$ important items from the level-h buffer — more precisely, it removes $2b$ important items if it is not important, and otherwise, it removes either $2b - 1$, or $2b + 1$ important items. The number of important compactions represented by i-nodes for some $i \in [g, \lambda]$ is at most $\mathrm{R}_h^{[g,\lambda]}(y)/5$ by Lemma 6.4 with $a = g$ and by $k_i \geq 20$ for any $i \leq \lambda$. Thus, the number of important items removed from level h during compactions represented by i-nodes for $i \in [g, \lambda]$ is upper bounded by $2\,\mathrm{R}_{h+1}^{[g,\lambda]}(y) + 2z_{h+1}^{[g,\lambda]} + (\mathrm{R}_h^{[g,\lambda]}(y)/5)$.

By Observation 6.9, at most $\sum_{i=g}^{\lambda} q_h^i \cdot B_i$ important items remain at the level-h buffers of the sketches represented by nodes in Q_h^i for some $i \geq g$. We thus have that

$$\mathrm{R}_h^{[g,\lambda]}(y) \leq 2\,\mathrm{R}_{h+1}^{[g,\lambda]}(y) + 2z_{h+1}^{[g,\lambda]} + \frac{1}{5} \cdot \mathrm{R}_h^{[g,\lambda]}(y) + \sum_{i=g}^{\lambda} q_h^i \cdot B_i.$$

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

$$
\begin{aligned}
\mathrm{R}_h^{[g,\lambda]}(y) &\leq \frac{5}{2} \cdot \mathrm{R}_{h+1}^{[g,\lambda]}(y) + \frac{5}{2} \cdot z_{h+1}^{[g,\lambda]} + \frac{5}{4} \cdot \sum_{i=g}^{\lambda} q_h^i \cdot B_i \\
&\leq \frac{5}{2} \cdot \left(\sum_{i=g}^{\lambda} \left(\sum_{h' \geq h+2} \left(2 \cdot 3^{h'-h-1} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i) \right) + 2 \cdot q_{h+1}^i \cdot B_i \right) \right) + \frac{5}{2} \cdot z_{h+1}^{[g,\lambda]} + \frac{5}{4} \cdot \sum_{i=g}^{\lambda} q_h^i \cdot B_i \\
&\leq \sum_{i=g}^{\lambda} \left(\sum_{h' \geq h+1} \left(2 \cdot 3^{h'-h} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i) \right) + 2 \cdot q_h^i \cdot B_i \right),
\end{aligned}
$$

where the second inequality uses the induction hypothesis (40). Thus, (39) holds.

Using $z_h^i \geq 0$, we simplify (39) and get

$$\mathrm{R}_h^{[g,\lambda]}(y) \leq \sum_{i=g}^{\lambda} \sum_{h' \geq h} \left(2 \cdot 3^{h'-h} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i) \right). \tag{41}$$

Finally, we bound the variance $\mathrm{Var}[\mathrm{Err}_h(y)]$, which is at most $\sum_{i=0}^{\lambda} m_h^i$ as $m_h^i = 0$ for $i > \lambda$ and $h \geq H_{\lambda+1}$, by Lemma 6.8. Recall from Section 6.3 that m_h^i is the number of important compaction operations at level h represented by i-nodes. We prove by a "backward" induction on $g = \lambda, \lambda - 1, \ldots, 0$ that the following inequality holds for any $h \geq H_{\lambda+1}$:

$$\sum_{i=g}^{\lambda} m_h^i \leq \sum_{i=g}^{\lambda} \frac{1}{k_i} \cdot \sum_{h' \geq h} 8 \cdot 3^{h'-h} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i). \tag{42}$$

Note that (42) for $g = 0$ gives (38) and that for $h = H$, there is no (important) compaction, thus we have that $h < H$. Consider $0 \leq g \leq \lambda$ and suppose that for any $g' > g$ (in the case $g < \lambda$), we have that

$$\sum_{i=g'}^{\lambda} m_h^i \leq \sum_{i=g'}^{\lambda} \frac{1}{k_i} \cdot \sum_{h' \geq h} 8 \cdot 3^{h'-h} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i). \tag{43}$$

To show (42), we use Lemma 6.4 with $a = g$ to get $\sum_{i=g}^{\lambda} m_h^i \cdot k_i \leq 4 R_h^{[g,\lambda]}(y)$. Dividing this inequality by k_g and using (41) gives

$$\sum_{i=g}^{\lambda} \frac{k_i}{k_g} \cdot m_h^i \leq \sum_{i=g}^{\lambda} \frac{1}{k_g} \cdot \sum_{h' \geq h} 8 \cdot 3^{h'-h} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i).$$

If $g = \lambda$, this proves the base case of the induction. Otherwise, for every $g' > g$, we add inequality (43) (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

$$\sum_{i=g}^{\lambda} \left(\frac{k_i}{k_g} + \sum_{g'=g+1}^{i} \frac{k_{g'-1} - k_{g'}}{k_g} \right) \cdot m_h^i$$

$$\leq \sum_{i=g}^{\lambda} \left(\frac{k_i}{k_g \cdot k_i} + \sum_{g'=g+1}^{i} \frac{k_{g'-1} - k_{g'}}{k_g \cdot k_i} \right) \cdot \sum_{h' \geq h} 8 \cdot 3^{h'-h} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i). \tag{44}$$

Note that the sum of fractions of k_i's on the RHS of (44) equals $1/k_i$ for any i, since the numerators in $\sum_{g'=g+1}^{i}(k_{g'-1} - k_{g'})/(k_g \cdot k_i)$ form a telescoping sum, which equals $k_g - k_i$. Similarly, the sum of fractions of k_i's on the LHS of (44) equals 1 for any i, so the LHS equals $\sum_{i=g}^{\lambda} m_h^i$. This shows (42). □

Finally, we have all ingredients needed to show that we can match the streaming result of Theorem 3 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. *For any parameters $0 < \delta \leq 0.5$ and $0 < \varepsilon \leq 1$, there is a randomized, comparison-based, one-pass streaming algorithm that, when processing a data stream consisting of n items from a totally-ordered universe $\mathcal{U}$, produces a summary S satisfying the following property. Given S, for any $y \in \mathcal{U}$ one can derive an estimate $\hat{R}(y)$ of $R(y)$ such that*

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

where the probability is over the internal randomness of the streaming algorithm. The size of S in memory words[11] is

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

Moreover, the summary produced is fully mergeable.

PROOF. We condition on the bounds from Lemmas 6.7 and 6.11, which together hold with probability at least $1 - 2\delta$. Using Lemma 6.12, we first bound the error on levels $h \geq H_{\lambda+1}$, for which we have that $m_h^i = 0$ for $i > \lambda$

---

[11]A memory word can store any universe item or an integer with $O(\log(n + |\mathcal{U}|))$ bits. We express all the space bounds in memory words.

by Lemma 6.8:

$$
\begin{aligned}
\sum_{h \geq H_{\lambda+1}} 2^{2h} \cdot \mathrm{Var}[\mathrm{Err}_h(y)] &\leq \sum_{h \geq H_{\lambda+1}} 2^{2h} \cdot \sum_{i=0}^{\lambda} \sum_{h' \geq h} \frac{8 \cdot 3^{h'-h} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i)}{k_i} \\
&= \sum_{i=0}^{\lambda} \sum_{h' \geq H_{\lambda+1}} \sum_{h=H_{\lambda+1}}^{h'} 2^{2h} \frac{8 \cdot 3^{h'-h} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i)}{k_i} \\
&\leq \sum_{i=0}^{\lambda} \sum_{h' \geq H_{\lambda+1}} \frac{2^{2h'+5} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i)}{k_i} &(45) \\
&\leq \sum_{i=0}^{\lambda} \sum_{h'=H_{\lambda+1}}^{H_i} \frac{2^{h'+9} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i) \cdot \mathrm{R}(y)}{k_i \cdot B_i} &(46) \\
&\leq \frac{\varepsilon^2 \cdot \mathrm{R}(y)}{2^5 \ln(1/\delta)} \cdot \sum_{i=0}^{\lambda} \sum_{h'=H_{\lambda+1}}^{H_i} 2^{h'} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i), &(47)
\end{aligned}
$$

where inequality (45) follows from

$$
\sum_{h=H_{\lambda+1}}^{h'} 2^{2h} \cdot 8 \cdot 3^{h'-h} = 8 \cdot 3^{h'} \cdot \sum_{h=H_{\lambda+1}}^{h'} \left(\frac{4}{3}\right)^h \leq 8 \cdot 3^{h'} \cdot 3 \cdot \left(\frac{4}{3}\right)^{h'+1} = 8 \cdot 4^{h'+1} = 2^{2h'+5},
$$

inequality (46) uses that $q_{h'}^i = 0$ and $z_{h'}^i = 0$ for $h' > H_i$ by Lemma 6.8 and that $2^{H_i} \leq 2^4 \cdot \mathrm{R}(y)/B_i$ by (34), and inequality (47) follows from the bound on $k_i \cdot B_i$ in (32).

By Lemma 6.11, $\sum_{i=0}^{\lambda} \sum_{h'=H_{\lambda+1}}^{H_i} 2^{h'} \cdot (q_{h'}^i \cdot B_i + z_{h'}^i) \leq 4\mathrm{R}(y)$, which implies our final variance bound for levels $h \geq H_{\lambda+1}$:

$$
\sum_{h \geq H_{\lambda+1}} 2^{2h} \cdot \mathrm{Var}[\mathrm{Err}_h(y)] \leq \frac{\varepsilon^2 \cdot \mathrm{R}(y)^2}{8 \ln(1/\delta)}.
$$

Let $\mathrm{Err}_{\geq H_{\lambda+1}}(y)$ be the error in the estimate of y from compactions at levels $h \geq H_{\lambda+1}$. Plugging the variance bound into the tail bound for sub-Gaussian variables (Fact 4.3) we conclude that

$$
\Pr\left[|\mathrm{Err}_{\geq H_{\lambda+1}}(y)| > \varepsilon \mathrm{R}(y)/2\right] < 2\exp\left(-\frac{\varepsilon^2 \cdot \mathrm{R}(y)^2}{8 \cdot \varepsilon^2 \cdot \mathrm{R}(y)^2/(8\ln(1/\delta))}\right) = 2\exp\left(-\ln\frac{1}{\delta}\right) = 2\delta.
$$

Next, we bound the error from compactions on levels below $H_{\lambda+1}$, denoted $\mathrm{Err}_{<H_{\lambda+1}}(y)$. The variance of this error is

$$
\begin{aligned}
\sum_{h=0}^{H_{\lambda+1}-1} 2^{2h} \cdot \mathrm{Var}[\mathrm{Err}_h(y)] &\leq \sum_{h=0}^{H_{\lambda+1}-1} 2^{2h} \cdot \mathrm{R}_h(y) \\
&\leq \sum_{h=0}^{H_{\lambda+1}-1} 2^{2h} \cdot 2^{-h+2} \cdot \mathrm{R}(y) &(48) \\
&\leq 2^{H_{\lambda+1}+2} \cdot \mathrm{R}(y) \leq 2^6 \cdot \frac{\mathrm{R}(y)^2}{B_{\lambda+1}} \leq \frac{\varepsilon^2 \cdot \mathrm{R}(y)^2}{8\ln(1/\delta)}
\end{aligned}
$$

where (48) uses Lemma 6.7 and the last two steps use (34) and (33), respectively (note that $B_{\lambda+1} \geq B_\lambda$). Using Fact 4.3 as above we get that $\Pr\left[|\mathrm{Err}_{<H_{\lambda+1}}(y)| > \varepsilon \mathrm{R}(y)/2\right] < 2\delta$. Rescaling δ, this completes the calculation of the failure probability.

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

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

$$= O\left(\log(\varepsilon n) \cdot \hat{k} \cdot \sqrt{\log\left(\frac{\overline{N_\ell}}{\hat{k}}\right)}\right)$$

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

where we use that $\log_2(\overline{N_\ell}/k_\ell) = O(\log(\overline{N_\ell}/\hat{k})) = O(\log(\varepsilon \overline{N_\ell})) = O(\log(\varepsilon n))$ (as $k_\ell \geq \hat{k}/\sqrt{\log_2(\overline{N_\ell}/\hat{k})}$, $\hat{k} \geq 1/\varepsilon$, and $\overline{N_\ell} \leq n^2$). □

7 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 Section 6 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 section 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 [27].

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

$$k = 2^4 \cdot \left\lceil \frac{1}{\varepsilon} \cdot \log_2 \ln \frac{1}{\delta} \right\rceil. \tag{49}$$

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 $R_h(y)/k$ important steps at each level h by Lemma 3.1.

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 $H'(y) \leq h < H(y)$, where $H(y)$ is defined as in Section 4 as the minimal h for which $2^{2-h} R(y) \leq B/2$. The idea to analyze a few top levels deterministically was first used by Karnin et al. [15] and we apply it also in Section 6.4. We define

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

Next, we provide modified rank bounds.

LEMMA 7.1. *Assuming $H(y) > 0$, for any $h < H(y)$ it holds that $R_h(y) \leq 2^{-h+2} R(y)$ with probability at least $1 - \delta$.*

PROOF. We first show by induction on $0 \leq h < H'(y)$ that $R_h(y) \leq 2^{-h+1} R(y)$ with probability at least $1 - \delta \cdot 2^{h-H'(y)}$, conditioned on $R_\ell(y) \leq 2^{-\ell+1} R(y)$ for any $\ell < h$. This part of the proof is similar to that of Lemma 4.4. The base case holds by $R_0(y) = R(y)$.

Consider $0 < h < H'(y)$. As in Lemma 4.4,

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

where Z_h is a zero-mean sub-Gaussian variable with variance at most $\mathrm{Var}[Z_h] \leq 2^{-h+1} \cdot R(y)/k$. We apply the tail bound for sub-Gaussian variables (Fact 4.3) on Z_h to get

$$\Pr[Z_h > 2^{-h} R(y)] < \exp\left(-\frac{2^{-2h} \cdot R(y)^2}{2 \cdot (2^{-h+1} \cdot R(y)/k)}\right)$$

$$= \exp\left(-2^{-h-2} \cdot R(y) \cdot k\right)$$

$$= \exp\left(-2^{-h+H'(y)-6} \cdot 2^{H(y)-H'(y)} \cdot 2^{4-H(y)} R(y) \cdot k\right)$$

$$\leq \exp\left(-2^{-h+H'(y)-6} \cdot 2^{H(y)-H'(y)} \cdot B \cdot k\right)$$

$$\leq \exp\left(-2^{-h+H'(y)} \cdot \ln\frac{1}{\delta}\right) = \delta^{2^{-H'(y)+h}} \leq \delta \cdot 2^{-H'(y)+h},$$

where the second inequality uses $2^{4-H(y)} R(y) \geq B$ (by the definition of $H(y)$), the third inequality follows from $2^{H(y)-H'(y)} \geq \ln\frac{1}{\delta}$ and $B \cdot k \geq k^2 \geq 2^6$, and the last inequality uses $\delta \leq 0.5$. This concludes the proof by induction. Taking the union bound over levels $h < H'(y)$, it holds that $R_h(y) \leq 2^{-h+1} R(y)$ for any $h < H'(y)$ with probability at least $1 - \delta$.

Finally, consider level $h \geq H'(y)$ and condition on $R_{H'(y)-1}(y) \leq 2^{-H'(y)+2} R(y)$. (In the case $H'(y) = 0$, we have $R_0(y) = R(y)$.) Note that for any $\ell > 0$, it holds that $R_\ell(y) \leq \frac{1}{2} \cdot (1 + 1/k) \cdot R_{\ell-1}(y)$. Indeed, $R_\ell(y) \leq \frac{1}{2} \cdot (R_{\ell-1}(y) + \mathrm{Binomial}(m_{\ell-1}))$ (see Equation 9) and $\mathrm{Binomial}(m_{\ell-1}) \leq m_{\ell-1} \leq R_{\ell-1}(y)/k$ by Lemma 3.1. 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 $H'(y) \leq \ell \leq h$, we get

$$R_h(y) \leq 2^{-h+H'(y)-1} \cdot \left(1 + \frac{1}{k}\right)^{h-H'(y)+1} \cdot R_{H'(y)-1}(y)$$

$$\leq 2^{-h+H'(y)-1} \cdot \left(1 + \frac{1}{k}\right)^{0.5 \cdot k} \cdot 2^{-H'(y)+2} \cdot R(y) \leq 2^{-h+2} \cdot R(y),$$

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

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 4. *Assume that (a polynomial upper bound on) the stream length n is known in advance. For any parameters $0 < \delta \leq 0.5$ and $0 < \varepsilon \leq 1$, let k be set as in (49). Then, for any fixed item y, Algorithm 2 with parameters*

k and n computes an estimate $\hat{R}(y)$ of $R(y)$ with error $\text{Err}(y) = \hat{R}(y) - R(y)$ such that

$$\Pr\left[\,|\text{Err}(y)| > \varepsilon\,R(y)\right] < 3\delta\,.$$

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 7.1, which together hold with probability at least $1 - \delta$. We split $\text{Err}(y)$, the error of the rank estimate for y, into two parts:

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

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

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

where the first inequality is by Lemma 3.1, the second by Lemma 7.1, and the last inequality uses $2^{H(y)} \leq 2^4 \cdot R(y)/B$, which follows from the definition of $H(y)$.

We again apply Fact 4.3 to obtain

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

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

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

$$R_{H(y)}(y) \leq 2^{-H(y)+2}\,R(y) \leq B/2,$$

where the first inequality holds because we have conditioned on the bounds of Lemma 7.1 holding, and the second inequality holds by the 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

$$\mathrm{Err}''(y) = \sum_{h=H'(y)}^{H(y)-1} 2^h \cdot \mathrm{Err}_h(y)$$

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

where the first inequality is by Lemma 3.1, the second by Lemma 7.1, the third inequality follows from the definition of k in (49), 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 4.7, 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 4 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 [27]. 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, inspecting the proofs of Lemma 7.1 and Theorem 4 yields that the entire analysis holds with probability 1. In more detail, when $H'(y) = 0$, the bounds in Lemma 7.1 hold with probability 1, and the quantity $\mathrm{Err}'(y)$ in the proof of Theorem 4 is deterministically 0, while the bound on $\mathrm{Err}''(y)$ in the proof of Theorem 4 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 [17].

8 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 [8], and is within an $\widetilde{O}\left(\sqrt{\log(\varepsilon n)}\right)$ 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 open question is to close the aforementioned $\widetilde{O}(\sqrt{\log(\varepsilon n)})$-factor gap.

REFERENCES

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

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

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

[4] Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. A framework for adversarially robust streaming algorithms. *J. ACM*, 69(2):17:1–17:33, 2022.

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

[6] 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.

[7] Graham Cormode, Abhinav Mishra, Joseph Ross, and Pavel Veselý. Theory meets practice at the median: A worst case comparison of relative error quantile algorithms. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, KDD '21, page 2722–2731, New York, NY, USA, 2021. Association for Computing Machinery.

[8] 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.

[9] Ted Dunning. The t-digest: Efficient estimates of distributions. *Software Impacts*, 7:100049, 2021.

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

[11] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

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

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

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

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

[24] Philippe Rigollet. 18.s997: High dimensional statistics: Lecture notes. https://ocw.mit.edu/courses/18-s997-high-dimensional-statistics-spring-2015/pages/lecture-notes/, 2015.

[25] 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.

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

[27] 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.

[28] 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 [8, 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 [5, Theorem 2]. We provide details in this appendix for completeness.

THEOREM 5. *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 $|S| = \Theta\left(\varepsilon^{-1} \log(\varepsilon n)\right)$. 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 $\{y_1, \ldots, y_s\}$ denote the elements of S in increasing order. Consider the stream σ where items $y_1, \ldots, y_\ell$ each appear once, items $y_{\ell+1}, \ldots, y_{2\ell}$ appear twice, and in general items $y_{i\ell+1}, \ldots, y_{(i+1)\ell}$ appear 2^i times, for $i = 0, \ldots, k-1$. Let us refer to all universe items in the interval $[y_{i\ell+1}, y_{(i+1)\ell}]$ as "phase-i" items.

The construction of σ means that the multiplicative error ε in the estimated rank of any phase-i item is at most $2^{i+1}/8 < 2^{i-1}$. This means that for any phase $i \geq 0$ and integer $j \in [1, \ell]$, one can identify item $y_{i\ell+j}$ by finding the smallest universe item whose estimated rank is strictly greater than $(2^i - 1) \cdot \ell + 2^i \cdot j - 2^{i-1}$. Here, $(2^i - 1) \cdot \ell$ is the number of stream updates corresponding to items in phases $0, \ldots, i-1$, while 2^{i-1} is an upper bound on the error of the estimated rank of any phase-i item. Hence, from any sketch solving the all-quantiles approximation problem for σ one can obtain the subset S, which concludes the lower bound. □

Theorem 5 is tight up to constant factors as an optimal summary consisting of $O(\varepsilon^{-1} \cdot \log(\varepsilon n))$ items can be constructed offline. For $\ell = \varepsilon^{-1}$, this summary stores all items of rank $1, \ldots, 2\ell$ appearing in the stream and assigns them weight one, stores every other item of rank between $2\ell + 1$ and 4ℓ and assigns them weight 2, stores every fourth item of rank between $4\ell + 1$ and 8ℓ and assigns them weight 4, and so forth. This yields a weighted coreset S for the relative-error quantiles approximation, consisting of $|S| = \Theta\left(\ell \cdot \log(\varepsilon n)\right)$ 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 holds for all $y \in \mathcal{U}$ simultaneously with probability $1 - \delta$ when the size of the sketch in memory words is*

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

PROOF. Let S^* be the offline optimal summary of the stream with multiplicative error $\varepsilon/3$, i.e., a subset of items in the stream such that for any item x, there is $y \in S^*$ with $|R(y) - R(x)| \leq (\varepsilon/3) \cdot R(x)$. Here, y is simply the closest item to x in the total order that is an element of S^*. Observe that S^* has $O(\varepsilon^{-1} \cdot \log(\varepsilon n))$ items; see the remark below Theorem 5 in Appendix A for a construction of S^*.

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

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

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 $\delta' = \delta/|S^*| = \Theta(\delta \cdot \varepsilon/\log(\varepsilon n))$. By the union bound, with probability at least $1 - \delta$, the resulting sketch satisfies the $(1 \pm \varepsilon/3)$-multiplicative error guarantee for any item in S^*. In this event, the previous paragraph implies that the $(1 \pm \varepsilon)$-multiplicative guarantee holds for all $x \in \mathcal{U}$. The space bound follows from Theorem 1 with ε' and δ' as above. □