Published: 13 May 2024

RESEARCH-ARTICLE

Simple & Optimal Quantile Sketch: Combining Greenwald-Khanna with Khanna-Greenwald

ELENA GRIBELYUK, Princeton University, Princeton, NJ, United States

PACHARA SAWETTAMALYA, Princeton University, Princeton, NJ, United States

HONGXUN WU, University of California, Berkeley, Berkeley, CA, United States

HUACHENG YU, Princeton University, Princeton, NJ, United States

Proceedings of the ACM on Management of Data, Volume 2, Issue 2 (May 2024)
https://doi.org/10.1145/3651610
EISSN: 2836-6573

Simple & Optimal Quantile Sketch: Combining Greenwald-Khanna with Khanna-Greenwald

ELENA GRIBELYUK, Princeton University, USA
PACHARA SAWETTAMALYA, Princeton University, USA
HONGXUN WU, UC Berkeley, USA
HUACHENG YU, Princeton University, USA

Estimating the ϵ-approximate quantiles or ranks of a stream is a fundamental task in data monitoring. Given a stream $x_1, ..., x_n$ from a universe $\mathcal{U}$ with total order, an additive-error quantile sketch $\mathcal{M}$ allows us to approximate the rank of any query $y \in \mathcal{U}$ up to additive ϵn error.

In 2001, Greenwald and Khanna gave a deterministic algorithm (GK sketch) that solves the ϵ-approximate quantiles estimation problem using $O(\epsilon^{-1}\log(\epsilon n))$ space [18]; recently, this algorithm was shown to be optimal by Cormode and Veselý in 2020 [14]. However, due to the intricacy of the GK sketch and its analysis, over-simplified versions of the algorithm are implemented in practical applications, often without any known theoretical guarantees. In fact, it has remained an open question whether the GK sketch can be simplified while maintaining the optimal space bound. In this paper, we resolve this open question by giving a simplified deterministic algorithm that stores at most $(2 + o(1))\epsilon^{-1}\log(\epsilon n)$ elements and solves the additive-error quantile estimation problem; as a side benefit, our algorithm achieves a smaller constant factor than the $\frac{11}{2}\epsilon^{-1}\log(\epsilon n)$ space bound in the original GK sketch [18]. Our algorithm features an easier analysis and still achieves the same optimal asymptotic space complexity as the original GK sketch.

Lastly, our simplification enables an efficient data structure implementation, with a worst-case runtime of $O(\log(1/\epsilon) + \log\log(\epsilon n))$ per-element for the ordinary ϵ-approximate quantile estimation problem. Also, for the related "weighted" quantile estimation problem, we give efficient data structures for our simplified algorithm which guarantee a worst-case per-element runtime of $O(\log(1/\epsilon) + \log\log(\epsilon W_n/w_{\min}))$, achieving an improvement over the previous upper bound of [7].

CCS Concepts: • **Theory of computation → Sketching and sampling**.

Additional Key Words and Phrases: quantiles, sketching, streaming

ACM Reference Format:
Elena Gribelyuk, Pachara Sawettamalya, Hongxun Wu, and Huacheng Yu. 2024. Simple & Optimal Quantile Sketch: Combining Greenwald-Khanna with Khanna-Greenwald. *Proc. ACM Manag. Data* 2, 2 (PODS), Article 109 (May 2024), 25 pages. https://doi.org/10.1145/3651610

1 INTRODUCTION

Computing the approximate ranks or quantiles with respect to a stream of elements is central to our understanding of the distribution of massive datasets. In settings where the size of a dataset exceeds the feasible amount of storage, streaming algorithms help to store specific information

Authors' addresses: Elena Gribelyuk, Princeton University, NJ, USA, eg5539@princeton.edu; Pachara Sawettamalya, Princeton University, NJ, USA, pachara@princeton.edu; Hongxun Wu, UC Berkeley, CA, USA, wuhx@berkeley.edu; Huacheng Yu, Princeton University, NJ, USA, yuhch123@gmail.com.

ACM 2836-6573/2024/5-ART109
https://doi.org/10.1145/3651610

about the data, called the *sketch*, while not storing the entire data stream in memory. A natural computational model for this setting is the *streaming model* [5].

In this work, we study the quantile estimation problem in the streaming model, wherein elements $x_1, ..., x_n$ arrive one at a time, and we hope to estimate the rank of any query $y \in \mathcal{U}$ up to additive-error ϵn after processing the stream into some data structure $\mathcal{M}$. Note that the query may be presented at any point in the stream, in which case we hope to approximate the rank of the query element with respect to the stream seen so far. Furthermore, we focus our attention on *comparison-based* algorithms, which are only allowed to compare elements via the total-ordering on universe $\mathcal{U}$.

There are multiple related definitions for the ϵ-quantile-estimation problem in the streaming literature. Alternatively, we may pose a query $r \in [n]$ and require the streaming algorithm to return an element x such that $|\text{rank}(x) - r| \leq \epsilon n$ with respect to the stream observed so far. In other formulations, the algorithm may be given advance knowledge of the specific query $x^* \in \mathcal{U}$ that will be asked at the end. We note that all of these variants can be solved by solving the harder *all-quantiles* ϵ-approximate quantile estimation problem, which maintains an ϵ-approximate quantile estimate for every possible query $x \in \mathcal{U}$ simultaneously. Formally, we define the problem as follows:

Definition 1 (All-Quantiles Sketch)**.** *Given a stream of elements $x_1, ..., x_n$ arriving one at a time, compute a sketch $\mathcal{M}$ and an estimator $\widehat{\text{rank}}^{(t)}$ such that for any query $x \in \mathcal{U}$ presented at any time t in the stream, we have that $|\widehat{\text{rank}}^{(t)}(x) - \text{rank}^{(t)}(x)| \leq \epsilon n$, where $\text{rank}(x)$ represents the true rank of element x with respect to the stream elements observed so far.*

Over the past 50 years, this problem has been studied under various formulations. The celebrated median of medians algorithm by Blum et al. [9] finds the median of n elements from any universe $\mathcal{U}$ with total order deterministically using $5.43n$ time and n space. Naturally, one might have hoped for a sublinear-space algorithm for computing the exact median of n elements which appear in a streaming fashion. Unfortunately, this was shown to be impossible: Munro and Paterson [23] proved that any streaming algorithm that can make p passes over the input stream requires at least $\Omega(n^{1/p})$ space to find the exact median.

In many downstream applications, it suffices to find an approximate quantile, or more generally, the ϵ-approximate ϕ-quantile for any $\phi \in [0, 1]$. To this end, Manku, Rajagopalan, and Lindsay gave the first streaming algorithm for approximate quantiles with theoretical guarantee. The MRL algorithm takes $O(\epsilon^{-1} \log^2(\epsilon n))$ space. This is improved by Greenwald and Khanna [18]. They proposed the intricate GK algorithm that achieves $O(\epsilon^{-1} \log(\epsilon n))$ space. This bound was later proved to be optimal for deterministic algorithms (even for the approximate median problem) by Cormode and Vesleý [14].

Moreover, if we allow algorithms to be randomized, one approach is to sample $\Theta(\epsilon^{-1} \log(1/\delta))$ elements from the stream and maintain them with using a GK sketch; this already gives an $O(\epsilon^{-1} \cdot (\log(1/\epsilon) + \log\log(1/\delta)))$-space algorithm that correctly answers one ϵ-approximate quantile query with probability $1 - \delta$. But, observe that this simple algorithm would require that the stream length n is known in advance. A follow-up work by Felber and Ostrovsky [16] later showed that one can achieve the same space bound without prior knowledge of the stream length n. Finally, Karnin, Lang, and Liberty gave an improved algorithm (KLL sketch) that requires only $O(\epsilon^{-1} \log\log(1/\delta))$ space, and showed a matching lower bound [22]. Importantly, the deterministic GK sketch is actually an essential component of both of these algorithms.

Deterministic streaming algorithms are interesting objects of study for their own sake. First, when insisting that the algorithm must always succeed (meaning that $\delta < 1/n!$, as there are

$n!$ possible inputs), the lower bound in Karnin, Lang, and Liberty [22] shows that randomized streaming algorithms cannot beat deterministic algorithms for approximate quantile estimation. Additionally, there have been many recent works on the *adversarially robust streaming model*, wherein an adversary may make queries at each time t during the stream and selects future stream elements after observing the previous outputs of the algorithm [4, 8, 17, 20, 24]. So, by making repeated quantile queries to a randomized quantile sketch, an adaptive adversary may eventually learn which elements were sampled by the randomized algorithm (i.e. the internal randomness of the algorithm), and can select the rest of the input adversarially in order to break the error guarantee of the streaming algorithm. This is very concerning, as it is known that many natural streaming problems require $\Omega(n)$ space if the computation is carried out by a deterministic algorithm. However, luckily for us, the ϵ-approximate quantile estimation problem admits sublinear-space solutions, and deterministic algorithms are automatically adversarially robust due to their correctness guarantee for all possible inputs.

In practice, the GK algorithm is quite popular. It is implemented in Spark-SQL [6] and also in various third-party packages [1, 3, 21] for popular programming languages such as Java, R, and Rust. However, since both the GK algorithm and its analysis are very intricate, what is actually implemented is a (over)simplified variant. It is an open problem asked by Cormode and Yi in their book [15] whether this variant has any nontrivial theoretical guarantee. Motivated by this, it was asked in [14] if there is a simpler algorithm without intricate merging of tuples achieves the optimal space bound. Similar questions are also asked as Problem 2 of "List of Open Problems in Sublinear Algorithms" [2]. Prior to our work, Assadi et al. [7] took a step towards resolving this open problem by proposing a simplified version of GK algorithm that requires $O(\epsilon^{-1}\log^2(\epsilon n))$ space (note: this is the same space bound as the MRL algorithm). However, it was still open to construct a simplified algorithm that achieves the optimal space bound of $O(\epsilon^{-1}\log n)$.

Lastly, the recent work of [7] also gave a non-trivial generalization of the approximate quantile estimation problem for *weighted* streams. In this setting, each stream element x_i carries a weight $w(x_i)$, and the rank of any query $y \in \mathcal{U}$ is calculated with respect to the weights of stream elements; that is, $\text{rank}(y) = \sum_{x_i \le y} w(x_i)$. Specifically, the problem statement is as follows:

Definition 2 (All-Quantiles Sketch for Weighted Streams)**.** *Given a input stream of pairs $(x_1, w(x_1))$, $(x_2, w(x_w)), \ldots, (x_n, w(x_n))$ arriving one at a time, let $\sum_{i=1}^{t} w(x_i) = W_t$ denote the total weight for the first t elements. An all-quantile sketch approximates the rank of query $x \in \mathcal{U}$ with $\widehat{\text{rank}}^{(t)}(x)$ such that $|\widehat{\text{rank}}^{(t)}(x) - \text{rank}^{(t)}(x)| \le \epsilon W_t$.*

We note that by simply interpreting each $(x_i, w(x_i))$ as x_i appearing $w(x_i)$ times consecutively in the stream and inserting each copy of x_i into our sketch $(\mathcal{M}_t, \mathcal{I}_t)$ one-by-one, we would trivially obtain an ϵ-quantile summary using $O(\epsilon^{-1}\log(\epsilon W_n))$ space for the weighted stream setting. However, the major disadvantage of inserting $w(x_i)$ copies of x_i one-at-a-time is that the insertion-time for each element $(x_i, w(x_i))$ scales linearly with the weight of x_i. To this end, the algorithm of [7] achieved a worst-case runtime of $O(\log(1/\epsilon) + \log\log(\epsilon W_n) + \frac{\log^2(\epsilon W_n)}{\epsilon n})$ per-element. In **??**, we show that our algorithm can be extended to solve this general problem on weighted streams as well.

1.1 Our Results

In this work, we propose a variant of GK sketch that is both simple and optimal, storing at most $\frac{2+o(1)}{\epsilon}\log(\epsilon n)$ elements. In fact, this improves upon the constant factor of $\frac{11}{2}$ of the original GK sketch [18]. At a high-level, the original GK sketch maintains a set of representatives $\{e_i\}$, such that each element x_t that appears in the stream is represented by some e_{x_t} (x_t may itself be a

representative stored in sketch $\mathcal{M}$, or it may have been deleted earlier and is now represented by some other stream element e_{x_t}). Furthermore, at each time step, the GK sketch decides whether to *merge* two adjacent representatives in memory (e_i, e_{i+1}) such that the algorithm maintains an additive-error of ϵt at all times $t \in [n]$ – this inherently limits the number of elements x_i that one e_j can represent. Additionally, when merging (e_i, e_{i+1}), the GK algorithm always keeps the larger representative e_{i+1} of the two sets, and uses it to represent the set of elements that were.

There are two involved components in the original GK algorithm:

(1) First, the authors define the notion of a *band value* for each element $e_i \in \mathcal{M}_t$; by grouping elements into a small number of "bands," each of which contains elements with similar insertion times, the original GK algorithm keeps track of some measure of how *accurate* our current estimate for the rank of each $x \in \mathcal{U}$ would be.
(2) Second, the authors merge representatives e_i according to a particular *tree structure* over the sets of elements covered by each representative e_i. To achieve the optimal space bound, their algorithm must merge all the sets in an entire subtree when certain invariants are satisfied. This approach inherently complicates the implementation of the GK sketch.

Roughly speaking, the analysis of the original GK algorithm can be divided into handling an "easy case" and a "hard case." Both of the components described above are essential for resolving the hard case. In attempt to simplify previous approaches, in Section 3, we begin our journey with a (over)simplified version of the GK algorithm. This version achieves the optimal space bound for the easy case, but has no space guarantees for the hard case. However, in Section 4, we show that by combining this over-simplified algorithm with its "mirroring," we are able to avoid dealing with the hard case entirely. Here, the mirroring of a GK sketch is obtained by flipping the total ordering of the universe $\mathcal{U}$, and maintaining a second GK sketch over the flipped ordering. Additionally, while the original GK sketch maintains the invariant that each representative e_i is *larger* than the set of elements that it represents, our algorithm maintains the property that the representative element e_i is always the *oldest* element among those it represents. This change allows for a more intuitive algorithm, since, as we will see later, older elements give rise to smaller margins of error.

Lastly, we provide efficient data structures for our algorithms in both the unweighted and weighted streaming settings. In particular, since our algorithm only ever merges pairs of adjacent elements, we simplify the implementation of [7] and obtain a worst-case running-time per element of $O(\log(1/\epsilon) + \log\log(\epsilon t))$ for unweighted streams, and $O(\log(1/\epsilon) + \log\log(\epsilon W_n/w_{\min}))$ for weighted streams. Note that this improves upon the worst-case per-element running time of [7] in the weighted streaming setting; since their algorithm relied on merging long "segments" of representatives during each compression step, their algorithm obtained a worst-case per-element runtime of $O(\log(1/\epsilon) + \log\log(\epsilon W_n) + \frac{\log^2(\epsilon W_n)}{\epsilon n})$.

Overall, our algorithm achieves both simplicity and optimality. We also present a novel potential analysis which circumvents the technical difficulties introduced in the GK sketch.

1.2 Further Related Works

Although in this work we focused only on comparison-based algorithms, there are non-comparison-based algorithms [10, 13, 25] that depend on the universe $\mathcal{U}$. For example, the q-digest [25] algorithm by Shrivastava et al. uses $O(\epsilon^{-1}\log|\mathcal{U}|)$ words of memory. Note for non-comparison-based algorithms, one has to assume the prior knowledge of the universe $\mathcal{U}$, which limits their application to floating point numbers in practice. Another difference is that for comparison-based model, we are counting the memory usage by the number of elements stored. But in the non-comparison-based model, there is a difference between the number of words used and the number of bits. This makes the comparison-based model a more generic model in theory.

Another interesting problem is the biased quantile problem, which instead of asking for a $[\phi-\epsilon, \phi+\epsilon]$ quantile (absolute error), the problem asks for a $[(1-\epsilon)\phi, (1+\epsilon)\phi]$ quantile (relative error). There is a line of research on streaming biased quantiles [11, 12, 14, 19, 26, 27]. The state of the art is a $O\left(\epsilon^{-1}\log^{1.5}(\epsilon n)\cdot\sqrt{\log(\log(\epsilon n)/\epsilon\delta)}\right)$-space randomized algorithm [11] by Cormode et al. It worth mentioning that Cormode and Vesleý proved that any deterministic streaming algorithm for biased quantile requires $\Omega(\epsilon^{-1}\log^2(\epsilon n))$ space. Hence, this randomized algorithm provably outperforms all deterministic ones. On the other hand, the GK algorithm does not seem to straightforwardly extend to biased quantiles. The state of the art deterministic algorithm is the one by Zhang and Wang [26] that takes $O(\epsilon^{-1}\log^3(\epsilon n))$ space. It is an open problem to close this gap for deterministic quantiles or to extend the GK algorithm to this setting.

2 PRELIMINARIES

Let $\mathcal{U}$ be the universe of elements with total order $\prec$. For any two elements $x, y \in \mathcal{U}$, we write $x \prec y$ to denote that x has smaller rank than y in $\mathcal{U}$. For a set of elements $S \subseteq \mathcal{U}$, we say $x \prec S$ when for all $y \in S$, we have $x \prec y$. We also denote $\succ, \preceq, \succeq$ analogously. Without loss of generality, we assume that all the elements in the input stream are distinct elements in $\mathcal{U}$.

Comparison-based model. In the comparison-based model, at any time t, the memory of a streaming algorithm is a tuple $(\mathcal{M}_t, I_t)$ where $\mathcal{M}_t \subseteq \mathcal{U}$ is a subset of elements (i.e. *representatives*), and I_t contains arbitrary auxiliary information. At any time t, the algorithm may perform comparisons between any two elements in $\mathcal{M}_t$. We note that the memory usage of the algorithm is measured by the size of $|\mathcal{M}_t|$ only.

2.1 The Basic Setup

We first provide a general overview of the original construction for our GK-based algorithm. This closely mimics the setup in [18].

Representatives. At any time t, let $\mathcal{S}^{(t)} = \{x_1, ..., x_t\}$ be the set of elements that have appeared in the stream so far. At a high level, the GK sketch maintains a partition of $\mathcal{S}^{(t)}$ into a disjoint union of sets and stores the largest element of each set in $\mathcal{M}_t$ as the representative of that set. We use $e_i^{(t)}$ to denote the i^{th} smallest element in $\mathcal{M}_t$ at time t. Likewise, $S_i^{(t)}$ denotes the set of elements represented by $e_i^{(t)} \in \mathcal{M}_t$ at time t. We note that $e_i^{(t)}$ itself is also in $S_i^{(t)}$; later in our work, we will actually *exclude* $e_i^{(t)}$ from $S_i^{(t)}$ for convenience. Also, to simplify our notation, we omit the superscript (t) when it is clear from the context and simply write e_i and S_i.

By this construction, we see that

$$e_i \succeq S_i, \quad \forall e_i \in \mathcal{M}_t. \tag{A0}$$

This enforces a partition $\mathcal{S}^{(t)} = \bigcup_{e_i \in \mathcal{M}_t} S_i$ at each time-step t. See Figure 1 for visual illustration.

Tuples. To maintain the error guarantee of the algorithm, the GK sketch also stores the following additional information in $\mathcal{I}_t$.

Definition 3 (Tuples). *For every representative $e_i^{(t)} \in \mathcal{M}_t$, the GK sketch stores a tuple $(g_i^{(t)}, \Delta_i^{(t)})$ in the auxiliary memory $\mathcal{I}_t$. We define $g_i^{(t)}, \Delta_i^{(t)}$ as follows:*

- *$g_i^{(t)} \leftarrow |S_i^{(t)}|$ is the number of elements in the set $S_i^{(t)}$ which is represented by stream element $e_i^{(t)}$.*

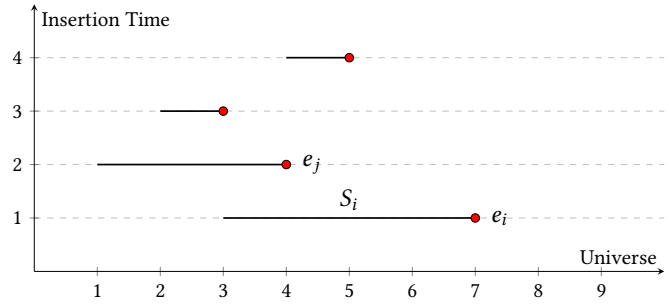


Fig. 1. GK sketch. Every node here corresponds to a representative $e_i \in \mathcal{M}_t$, while the line segment corresponds to the set of elements S_i it represents. We can see that we always have $e_i \succ S_i$. Some elements in S_i may even be smaller than some other representative $e_j \in \mathcal{M}_t$ that are smaller than e_i.

- $\Delta_i^{(t)}$ *is an upper bound on the number of elements e such that e is smaller than representative e_i, but belongs to set S_j for $e_j \succ e_i$ (informally, this serves as a measure of "uncertainty" for our rank estimate). More concretely,*

$$\sum_{j=i+1}^{|\mathcal{M}_t|} |\{e \in S_j \mid e \prec e_i\}| \le \Delta_i \tag{A1}$$

Jumping ahead, this will be one of the invariants maintained throughout the GK algorithm.

The following property is straightforward but crucial to GK sketch:

Observation 4. *For any element $e_i \in \mathcal{M}_t$, let $\mathrm{rank}^{(t)}(e_i)$ be the rank of e_i with respect to the stream inserted until time t. Then, we see that $r_{\min}^{(t)}(e_i) \le \mathrm{rank}^{(t)}(e_i) \le r_{\max}^{(t)}(e_i)$, where $r_{\min}^{(t)}(e_i) := \sum_{j=1}^{i} g_j^{(t)}$ and $r_{\max}^{(t)}(e_i) := r_{\min}^{(t)}(e_i) + \Delta_i^{(t)}$. (We omit the superscript (t) when it is clear from the context.)*

Proof. Since $\mathcal{S}^{(t)} = \cup_{e_i \in \mathcal{M}_t} S_i$ is a partition of the elements inserted so far, we know that

$$\mathrm{rank}^{(t)}(e_i) \;=\; \sum_{j=1}^{|\mathcal{M}_t|} |\{e \in S_j \mid e \preceq e_i\}| \;=\; \sum_{j=1}^{i} |S_j| + \sum_{j=i+1}^{|\mathcal{M}_t|} |\{e \in S_j \mid e \prec e_i\}|$$

where the last equality follows from the following case discussion: For all $j \le i$, we have $S_j \preceq e_j \preceq e_i$. So the entire sets S_j are counted. For all $j > i$, as i is a representative itself, it is not in S_j. We can then safely change $e \preceq e_i$ to $e \prec e_i$.

The first term here equals exactly $r_{\min}(e_i)$. By the invariant (A1), the second term is at least 0 and at most Δ_i. This finishes the proof. □

The main GK sketch invariant. To facilitate the additive-error guarantee, the GK sketch maintains the following invariant at each time-step $t \in [n]$:

$$g_i + \Delta_i \le \epsilon t, \quad \forall e_i \in \mathcal{M}_t \tag{A2}$$

Claim 5. *As long as the invariant* (A2) *holds, GK sketch can answer any $\epsilon/2$-approximate ϕ-quantile query correctly.*

Proof. Consider any two adjacent elements e_i and e_{i+1} in $\mathcal{M}_t$, by Observation 4, the rank of e_i is at least $r_{\min}(e_i) = \sum_{j=1}^{i} g_j$, and the rank of e_{i+1} is at most $r_{\max}(e_{i+1}) = r_{\min}(e_{i+1}) + \Delta_{i+1} = r_{\min}(e_i) + g_{i+1} + \Delta_{i+1} \le r_{\min}(e_i) + \epsilon t$ (by the invariant (A2)). As a result, any two adjacent elements in $\mathcal{M}_t$ can be at most ϵt rank apart.

Let i be the largest index such that $r_{\min}(e_i) \le (\phi - \epsilon/2)t$. We know that the rank of e_{i+1} must be between $(\phi - \epsilon/2)t$ and $(\phi + \epsilon/2)t$. Therefore, we can simply return e_{i+1} as the answer. (In the corner case where such i does not exist, we simply return the smallest element in the sketch.) □

Indeed, to approximate the rank of any query $x \in \mathcal{U}$, we can simply find the smallest element $e_i \in \mathcal{M}_t$ such that $e_i \preceq x \prec e_{i+1}$ and return $\widehat{\text{rank}}^{(t)}(x) = \sum_{j=1}^{i} g_j$. Thus, it remains for us to show how to maintain the invariant (A2). We provide an overview of this in the following sections. We note that the original GK sketch also relies on two other more-involved ingredients, such as "band-values" and maintaining an implicit tree structure over the representatives, which we discuss in Section 3.3. Readers may also refer to Appendix A for the procedure of the original GK sketch.

3 WARM UP: AN (OVER)SIMPLIFIED GK ALGORITHM

In this section, we present a toy algorithm that maintains the invariant (A2) but may use unbounded space. This serves as an important starting point for our work. Along the way, we will specify other invariants of the algorithm that help us achieve the additive-error guarantee and $O(\epsilon^{-1}\log(\epsilon n))$ space bound. For a summary of all of the invariants, the reader may refer to Table 1.

3.1 Our Motivation: The Toy Algorithm

Recall that at time t, the GK sketch maintains a tuple (g_i, Δ_i) for every representative $e_i \in \mathcal{M}_t$. In this toy algorithm, we also add attribute t_i to form a triple (g_i, Δ_i, t_i), where t_i is the insertion time of element e_i.

Insertion. First, we explain how to insert a new element into the GK sketch. Consider the time t where we insert the t^{th} stream element x_t. First, let $\mathcal{M}_t = \mathcal{M}_{t-1} \cup \{x_t\}$. Suppose that x_t is the i-th smallest element in $\mathcal{M}_t$; that is $x_t = e_i$. We create a new singleton set $S_i = \{x_t\}$. Moreover, for the triple (g_i, Δ_i, t_i), we define $g_i = |S_i| = 1$ and $t_i = t$. We also set $\Delta_i = g_{i+1} + \Delta_{i+1} - 1$ (which we will justify below). For the pseudocode, see Algorithm 1.

Algorithm 1: Inserting a new element x_t at time t (toy algorithm)

1 $\mathcal{M}_t \leftarrow \mathcal{M}_{t-1} \cup \{x_t\}$.
2 Let i be the rank of x_t in $\mathcal{M}_t$.
3 $(g_i, \Delta_i, t_i) \leftarrow (1, g_{i+1} + \Delta_{i+1} - 1, t)$.

The following claim justifies our choice to set $\Delta_i := g_{i+1} + \Delta_{i+1} - 1$ for the newly-inserted element e_i.

Claim 6. *Upon the insertion of element e_i, we have*

$$\sum_{j=i+1}^{|\mathcal{M}_t|} |\{e \in S_j \mid e \prec e_i\}| \le g_{i+1} + \Delta_{i+1} - 1.$$

As a corollary, the invariant (A1) *is maintained after insertion.*

Proof. Let e be any element such that $e \in S_j$ for $j > i$ and $e \prec e_i$. We do casework on j. If $j = i + 1$, then we must have $e \in S_{i+1} \setminus \{e_{i+1}\}$. On the other hand, if $j > i + 1$, as $e \prec e_i$, it must also

be the case that $e \prec e_{i+1}$. As a result, we have

$$\begin{aligned}\sum_{j=i+1}^{|\mathcal{M}_t|} |\{e \in S_j \mid e \prec e_i\}| &= |\{e \in S_{i+1} \mid e \prec e_i\}| + \sum_{j=i+2}^{|\mathcal{M}_t|} |\{e \in S_j \mid e \prec e_i\}| \\ &\le |S_{i+1} \setminus \{e_{i+1}\}| + \sum_{j=i+2}^{|\mathcal{M}_t|} |\{e \in S_j \mid e \prec e_{i+1}\}| \\ &\le g_{i+1} - 1 + \Delta_{i+1}.\end{aligned}$$

□

Compression. As insertion creates new singleton sets, the total number of representatives $|\mathcal{M}_t|$ increases by 1. After each insertion, we need an operation to again reduce $|\mathcal{M}_t|$ if it becomes too large. The idea is to check every pair (e_i, e_{i+1}) and merge consecutive elements when possible.

Definition 7 (Mergeable pairs)**.** *We say that two consecutive elements in memory (e_i, e_{i+1}) are mergeable if both of the following conditions hold:*

(1) $g_i + g_{i+1} + \Delta_{i+1} \le \epsilon t$.
This condition guarantees that after merging two adjacent elements (e_i, e_{i+1}) in memory $\mathcal{M}_t$, the invariant (A2) *is still maintained. This is crucial to obtain the additive-error guarantee.*

(2) $t_i > t_{i+1}$.
The second condition guarantees that the sets represented by e_i will only be represented by an elder representative e_{i+1} (i.e. a representative that was inserted earlier in the stream). Intuitively, as Claim 6 guarantees that $\Delta_i \le \epsilon t_i$, so the elder representative e_{i+1} will have smaller Δ_{i+1}. This is beneficial to us[1]*, since we will see in future sections that after merging any two elements (e_i, e_{i+1}), the Δ-value of the new set can only increase.*

To merge any pair of consecutive elements (e_i, e_{i+1}) stored in $\mathcal{M}_t$, we first merge S_i into S_{i+1}. Then, the larger representative e_{i+1} becomes the representative for the new set, and we update $g_{i+1} \leftarrow g_i + g_{i+1}$ accordingly. Since Δ_{i+1} is defined as an upper bound on $\sum_{j=i+2}^{|\mathcal{M}_t|} |\{e \in S_j \mid e \prec e_{i+1}\}|$, Δ_{i+1} is not affected by this merge and remains unchanged. Finally, we remove e_i from memory $\mathcal{M}_t$ and delete the tuple (g_i, Δ_i, t_i) from $\mathcal{I}_t$. The original e_{i+1} becomes the i-th smallest element, and we shift the indices of the tuples accordingly. See Algorithm 2 for the pseudocode of this operation.

Algorithm 2: Try to merge e_i and e_{i+1} (toy algorithm)

```
1 Let e_i, e_{i+1} ∈ M_t be two adjacent elements in memory.
2 if t_i > t_{i+1} then
3     if g_i + g_{i+1} + Δ_{i+1} ≤ εt then
4         g_{i+1} ← g_i + g_{i+1}. /* Merge S_i into S_{i+1}.          */
5         Δ_{i+1} and t_{i+1} remain unchanged.
6         Remove e_i from M_t and tuple (g_i, Δ_i) from I_t.
```

Recall that in the "if" condition (Line 2), we use the exact insertion-time t_i. We observe that this is different from the original GK algorithm, which uses $\text{band}_t(e_i)$ in place of t_i here. Now, we make the following observation:

[1]As asked in the book by Cormode and Yi [15], it is open whether the variant of GK algorithm without condition (b) has any space guarantee.

Observation 8 (The representative element is always the oldest.)**.** *Let t' denote the insertion time of element $x_{t'}$. Then, for any time t, we have that*

$$t' \geq t_i, \quad \forall e_i \in \mathcal{M}_t,\ x_{t'} \in S_i \tag{A3}$$

Proof. We use induction on t. The base case $t = 1$ is trivial since the stream only has one element; thus, there is only one element being stored in $\mathcal{M}_1$.

For the inductive step, assume that for all $e_i \in \mathcal{M}_t$ and $x_{t'} \in S_i$, we have that $t' \geq t_i$. Consider time-step $t + 1$. It suffices to show that any insert or merge operations that occurred between time t and time $t + 1$ will maintain this property. Note that if the operation at time t is an insertion, this holds trivially since if we insert an element i, then $S_i = \{i\}$ and no changes occur to other tuples stored in memory. In the case that a merge operation occurs at time t, suppose that we merge S_i into S_{i+1}. Since $t_i > t_{i+1}$ and the merged element uses $i + 1$ as a representative (which is the older of the two), any $x_{t'} \in S_i \cup S_{i+1}$ has $t' \geq \min\{t_i, t_{i+1}\} = t_{i+1}$. This means after merging, this property is maintained as desired. □

We observe that Observation 8 holds in our toy algorithm, but this is not true for the original GK algorithm. In summary, we have introduced the following invariants (Table 1). It is easy to verify that these invariants are all maintained throughout the toy algorithm.

(A0)	$e_i \geq S_i,$	$\forall e_i \in \mathcal{M}_t.$
(A1)	$\sum_{j=i+1}^{\lvert\mathcal{M}_t\rvert} \lvert\{e \in S_j \mid e \prec e_i\}\rvert \leq \Delta_i$	$\forall e_i \in \mathcal{M}_t$
(A2)	$g_i + \Delta_i \leq \epsilon t,$	$\forall e_i \in \mathcal{M}_t$
(A3)	$t' \geq t_i,$	$\forall e_i \in \mathcal{M}_t,\ x_{t'} \in S_i$

Table 1. Invariants of the toy algorithm.

Here (A0) holds by definition, while (A1), (A2), (A3) holds by Claim 6, Line 3 of Algorithm 2, and Observation 8 respectively.

We briefly recall the definition of *mergeable pairs* given in Definition 7: namely, if a pair $(e_i, e_{i+1}) \in \mathcal{M}_t$ cannot be merged, it must be that either

- **Case 1:** e_i is actually the elder representative among (e_i, e_{i+1}) and thus S_i cannot be merged into S_{i+1} ($t_i < t_{i+1}$), or
- **Case 2:** performing the merge operation would make us fail to satisfy the invariant (A1) ($g_i + g_{i+1} + \Delta_{i+1} > \epsilon t$).

In the next few subsections, we will refer to Case 1 as the **hard case**, and we consider Case 2 to be the **easy case** of the GK sketch analyses of both [18] and [7].

3.2 The Easy Case: a Novel Potential Analysis

In this subsection, we will prove that our toy algorithm "half works," i.e. it solves the easy case of the GK sketch analysis, but provides no space guarantees for the hard case. Specifically, we show that there can be only a bounded number of pairs (e_i, e_{i+1}) in $\mathcal{M}_t$ with $t_i > t_{i+1}$. However, for our toy algorithm, it is possible for it to maintain an unbounded number of such non-mergeable pairs (e_i, e_{i+1}) in the hard case. We prove this formally in the following lemma, which proceeds via a potential-based argument. We defer the proof of this lemma to Appendix B.

Lemma 9. *At any time t, the number of non-mergeable pairs (e_i, e_{i+1}) in $\mathcal{M}_t$ with $t_i > t_{i+1}$ is at most $\frac{(2+o(1))\log(\epsilon t)}{\epsilon}$.*

3.3 Handling the Hard Case: Some Heavy Lifting

Before proceeding to our algorithm, we provide some background on the previous tools that were used to upper bound the number of not mergeable pairs $(e_i, e_{i+1}) \in \mathcal{M}_t$ where $t_i < t_{i+1}$. As we discussed previously, our naive toy algorithm does not provide any guarantees for the hard case. Indeed, this is not surprising, as the original GK sketch [18] handles the hard case with its two most involved components, which we will discuss in greater detail below. Note that our algorithm completely avoids this hard case, so these components are not needed for our new algorithm. For the complete procedure of the original GK sketch, readers may refer to Appendix A.

Band-values. To start, the authors define the notion of a *band-value* for every stored element $e_i \in \mathcal{M}_t$, where $\text{band}_t(e_i) \approx \log_2(\epsilon(t - t_i + 1))$ (defined according to some rounding scheme). Notably, previous works used band-values in place of keeping track of exact insertion-times t_i directly for each element e_i: so, to be more precise, the "hard" case of [18] and [7] was actually the case that $\text{band}_t(e_i) > \text{band}_t(e_{i+1})$ (i.e. this corresponds to the case when e_i is older than e_{i+1}, just as we defined in our hard case). But, one might wonder: why keep track of a more-complicated quantity like $\text{band}_t(e_i)$ in lieu of a simple quantity like an insertion-time t_i? Intuitively, by introducing band-values, the authors prevent the worst-case that $\mathcal{M}_t$ consists of an arbitrary long chain of representatives with *strictly-increasing* insertion times (i.e. $t_i < t_{i+1}$ for every e_i in the chain), since any chain of representative elements with strictly-increasing band-values can have length at most $\log(\epsilon n)$.

In fact, this idea motivated a simplified algorithm in Assadi et al.[7], which requires $O(\epsilon^{-1}\log^2(\epsilon n))$ space. However, for the analysis to go through, their rounding scheme must guarantee a consistency condition: for any $e_i \neq e_j$, once the band values equal, they will always be equal. Namely, once $\text{band}_t(e_i) = \text{band}_t(e_j)$ for some t, for all $t' > t$, we must have $\text{band}_{t'}(i) = \text{band}_{t'}(j)$. This requirement adds extra complication to their algorithm.

Tree representation. To achieve the optimal space complexity, the GK algorithm constructs a tree structure over the elements in $\mathcal{M}_t$ [18]. Instead of simply trying to merge adjacent pairs $(e_i, e_{i+1}) \in \mathcal{M}_t$, the GK algorithm tries to merge all elements belonging to the same subtree. A priori, it is not clear that this is necessary, but it appears to be essential for their analysis. Alternatively, a variant of the GK algorithm presented in [7] made an attempt to simplify this tree-based merging pattern by introducing the concept of a *segment*. Here, for any element e_i stored in memory, the authors try to merge e_i along wtih its entire segment $\text{seg}(e_i) = \{e_j : \text{band}_t(e_j) < \text{band}_t(e_i)\}$, as long as appropriate additive-error guarantees are met. While this approach provides a conceptual simplification of the original algorithm, it heavily relies on merging long segments of elements to show the optimal space bound.

4 COMBINING GK SKETCH WITH ANOTHER "MIRRORED" GK SKETCH

In this section, we present our simplified algorithm and its partial analysis (some proofs are deferred to the Appendix). In our work, we eliminate the need for band-values, and provide an algorithm that does not require a tree-based merging pattern nor segment-based merging pattern. In fact, our algorithm only ever performs merges between *pairs* of adjacent elements stored in memory. This significantly simplifies our data structure implementation of the algorithm, and gives rise to a short and clean analysis. To our knowledge, our algorithm is the first ϵ-approximate quantile summary with this property that achieves the optimal $O(\epsilon^{-1}\log(\epsilon n))$ space.

Theorem 10. *For any $\epsilon > 0$ and a stream of length n, our algorithm maintains an ϵ-approximate quantile summary while keeping $(2 + o(1))\epsilon^{-1}\log(\epsilon n)$ elements in $\mathcal{M}_t$.*

Intuition. Our algorithm is based on the following simple intuition that has been overlooked for over 20 years. Recall that the easy case (Section 3.2) is to upper bound the number of pairs (e_i, e_{i+1}) in $\mathcal{M}_t$ with $t_i < t_{i+1}$. The hard case (Section 3.3) is to upper bound the number of pairs with $t_i > t_{i+1}$.

Now, consider the following thought experiment: suppose we reverse the total order $\prec$ of the universe $\mathcal{U}$ by defining $x \prec_{\text{rev}} y$ if and only if $x \succ y$. Then, we examine the result of running our toy algorithm on the input stream $x_1, ..., x_n$ equipped with this reversed total ordering on $\mathcal{U}$. By symmetry, it is clear that running GK sketch with the reversed total order will maintain a set of representatives $e_i \in \mathcal{M}_t$ corresponding tuples (g_i, Δ_i, t_i) such that representative e_i is the *smallest* element in S_i (recall: representatives e_i used to always be the *largest* element in S_i in the original GK sketch). Due to this swapped total ordering, we then see that the hard case (that $t_i < t_{i+1}$) will now become the easy case ($t_i > t_{i+1}$). So, by combining the original toy algorithm with the "mirrored" algorithm, one might hope to completely avoid dealing with the hard case.

Finally, as we later point out in Remark 14, there is an alternate interpretation of our algorithm. Instead of using the largest element of each set S_i as the representative like the GK sketch, our algorithm can be seen as using the *oldest element* (the earliest inserted element) of each set as representative.

4.1 Our Algorithm

In our final algorithm, we split our input stream into two disjoint sets, one maintained by the toy algorithm and one maintained by its mirroring. However, we will use the same set of representatives for these two algorithms, though one key difference (from the toy algorithm) is that we no longer store the representative e_i in S_i.

Representatives and Tuples. Formally, let $\mathcal{M}_t$ be the set of representatives stored in memory at time t. For every element $e_i \in \mathcal{M}_t$, e_i simultaneously represents both the set S_i (that satisfies $S_i \prec e_i$) and set S_i° (that satisfies $e_i \prec S_i^\circ$). Importantly, we note that e_i itself is not contained in either set S_i or S_i°. Additionally, we denote $\mathcal{S}^{(t)} = \cup_{i\in[|\mathcal{M}_t|]} S_i$ and $\mathcal{S}^{\circ(t)} = \cup_{i\in[|\mathcal{M}_t|]} S_i^\circ$. We may omit the superscript (t) whenever it is clear from context. Observe that the stream $\{x_1, ..., x_t\}$ inserted by time t can be partitioned as $\mathcal{S}^{(t)} \cup \mathcal{S}^{\circ(t)} \cup \mathcal{M}_t$.

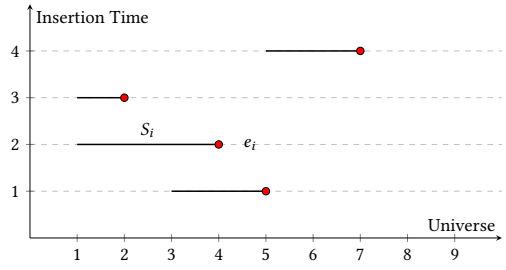
For each $e_i \in \mathcal{M}_t$, we define two triples (g_i, Δ_i, t_i) and $(g_i^\circ, \Delta_i^\circ, t_i)$, which we store in the auxiliary memory $\mathcal{I}_t$. As before, we consider $g_i = |S_i|$ and $g_i^\circ = |S_i^\circ|$. Similarly, we interpret Δ_i and Δ_i° to be a measure of uncertainty for our rank estimate; more formally, Δ_i and Δ_i° are defined to satisfy

$$\sum_{j=i+1}^{|\mathcal{M}_t|} |\{e \in S_j \mid e \prec e_i\}| \le \Delta_i \qquad \text{and} \qquad \sum_{j=1}^{i-1} |\{e \in S_j^\circ \mid e \succ e_i\}| \le \Delta_i^\circ$$

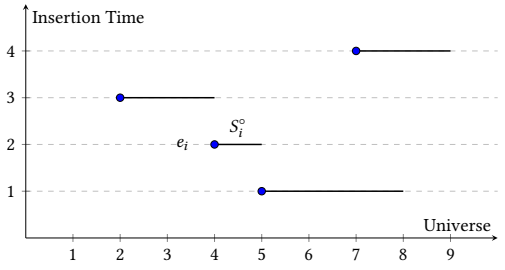
respectively. See Figure 2 for an example.

Insertion. To insert the t^{th} stream element x_t into sketch $\mathcal{M}_t$, we let $\mathcal{M}_t = \mathcal{M}_{t-1} \cup \{x_t\}$. Suppose x_t is the i^{th} smallest element in $\mathcal{M}_t$. This insertion introduces new sets S_i, S_i°, which will contain other stream elements that are *represented* by x_t. In particular, when x_t is first inserted into $\mathcal{M}_t$, we have that $S_i = S_i^\circ = \emptyset$. We then initialize the auxiliary information tuples (g_i, Δ_i, t_i) and $(g_i^\circ, \Delta_i^\circ, t_i)$ as follows:

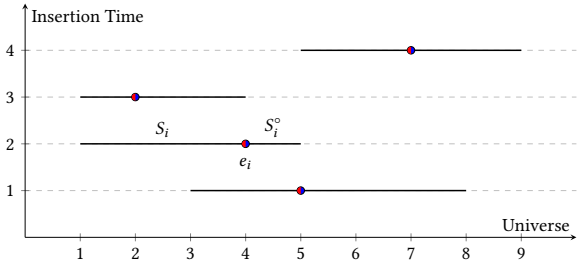
- **g-values**: Since $g_i \leftarrow |S_i|$ and $g_i^\circ \leftarrow |S_i^\circ|$, we initialize $g_i = 0$ and $g_i^\circ = 0$, as e_i does not represent any other stream elements upon its insertion.



(a) The toy algorithm.



(b) Its mirroring algorithm.



(c) The Final Algorithm.

Fig. 2. Note these two sketches share the same set of representatives $\mathcal{M}_t = \{2, 4, 5, 7\}$. Each element e_i is simultaneously the representative of two sets, S_i and S_i°.

- **Δ-value:** As in our toy algorithm, for Δ_i and Δ_i°, we set $\Delta_i = g_{i+1} + \Delta_{i+1}$. For Δ_i°, we let $\Delta_i^\circ = g_{i-1}^\circ + \Delta_{i-1}^\circ$; note that e_i inherits its value for Δ_i° from the $(i-1)$-th smallest element, due to the reversed total order on $\mathcal{U}$ in our mirrored sets S_i°.
- t_i: finally, we let $t_i \leftarrow t$ represent the time of insertion for $e_i = x_t$, as before.

The insertion operation for any new stream element x_t is also summarized in Algorithm 1, provided below.

Algorithm 3: Inserting a new element x_t at time t (Our algorithm)

$\mathcal{M}_t \leftarrow \mathcal{M}_{t-1} \cup \{x_t\}$.
Suppose x_t is the i^{th} smallest element in $\mathcal{M}_t$.
Set $(g_i, \Delta_i, t_i) \leftarrow (0, g_{i+1} + \Delta_{i+1}, t)$.
Set $(g_i^\circ, \Delta_i^\circ, t_i) \leftarrow (0, g_{i-1}^\circ + \Delta_{i-1}^\circ, t)$.

Compression. Next, we describe the procedure to merge any two adjacent elements (e_i, e_{i+1}). With this in mind, we provide the full algorithm for merging any pair (e_i, e_{i+1}) below.

Definition 11 (Mergeable pairs)**.** *At any time t, we say that that a pair of consecutive elements (e_i, e_{i+1}) is mergeable if one of the following is satisfied:*

(1) $t_i > t_{i+1}$ and $g_i + g_i^\circ + g_{i+1} + \Delta_{i+1} + 1 \le \epsilon t$, or
(2) $t_i < t_{i+1}$ and $g_i^\circ + g_i + g_{i+1}^\circ + \Delta_i^\circ + 1 \le \epsilon t$.

Otherwise, we say (e_i, e_{i+1}) is not mergeable.

Algorithm 4: Merging e_i, e_{i+1} (Our algorithm)

Let $e_i, e_{i+1} \in \mathcal{M}_t$ be two adjacent elements in memory.
if $t_i > t_{i+1}$ **then**
  if $g_i + g_i^\circ + g_{i+1} + \Delta_{i+1} + 1 \le \epsilon t$ **then**
    $g_{i+1} \leftarrow g_i + g_i^\circ + g_{i+1} + 1$. /* Merge $S_i \cup S_i^\circ \cup \{e_i\}$ into S_{i+1} */
    Remove e_i from $\mathcal{M}_t$ and tuples (g_i, Δ_i) and $(g_i^\circ, \Delta_i^\circ)$ from $\mathcal{I}_t$.
else
  if $g_{i+1}^\circ + g_{i+1} + g_i^\circ + \Delta_i^\circ + 1 \le \epsilon t$ **then**
    $g_i^\circ \leftarrow g_{i+1}^\circ + g_{i+1} + g_i^\circ + 1$. /* Merge $S_{i+1}^\circ \cup S_{i+1} \cup \{e_{i+1}\}$ into S_i° */
    Remove e_{i+1} from $\mathcal{M}_t$ and tuples (g_{i+1}, Δ_{i+1}) and $(g_{i+1}^\circ, \Delta_{i+1}^\circ)$ from $\mathcal{I}_t$.

Remark 12. *When we compute $g_{i+1} \leftarrow g_i + g_i^\circ + g_{i+1}$ (as in Algorithm 4 above), this can be interpreted as combining sets $S_i \cup S_i^\circ \cup \{e_i\}$ into S_{i+1}, and removing e_i (and its auxiliary information) from memory. However, it is important to remember that we never actually store the sets S_i throughout the algorithm, and we refer to them only for intuition and clarity of our analysis.*

Invariants. Just as we had in the toy algorithm (Table 1), we need to ensure that the following invariants are maintained throughout the algorithm.

(B0)	$S_i \prec e_i \prec S_i^\circ,$	$\forall e_i \in \mathcal{M}_t$
(B1)	$\sum_{j=i+1}^{\lvert\mathcal{M}_t\rvert} \lvert\{e \in S_j \mid e \prec e_i\}\rvert \le \Delta_i$	$\forall e_i \in \mathcal{M}_t$
	and $\sum_{j=1}^{i-1} \lvert\{e \in S_j^\circ \mid e \succ e_i\}\rvert \le \Delta_i^\circ,$	
(B2)	$g_i + \Delta_i \le \epsilon t \quad$ and $\quad g_i^\circ + \Delta_i^\circ \le \epsilon t,$	$\forall e_i \in \mathcal{M}_t$
(B3)	$t' \ge t_i,$	$\forall e_i \in \mathcal{M}_t,\ x_{t'} \in S_i \cup S_i^\circ$

Table 2. Invariants of our algorithm.

Lemma 13. *All invariants in Table 2 hold throughout our algorithm.*

In fact, all invariants (other than (B0)) follow easily from the construction of our algorithm. So, we defer the proof of this lemma to the Appendix C. The proof that (B0) holds is crucial, and is presented in Section 4.2.

Remark 14 (The oldest element represents the set.)**.** *We note that if $C_i = S_i \cup S_i^\circ \cup \{e_i\}$ is viewed as a single set of elements represented by e_i in both sketches, invariant* (B3) *says that e_i is always the* oldest *element in this set, i.e. e_i has the smallest insertion time $t_i < t_j$ for all $e_j \in C_i$. Thus, our algorithm can also be understood as using the oldest element of each set as the representative. In contrast, the original GK algorithm defines the representative of each set to be the largest element.*

4.2 Correctness

In this section, we will address the following two major components that certify the correctness of the algorithm.

- *Proof of invariant* (B0). In Algorithm 4, we move elements around these two sketches by, for example, merging $S_i \cup S_i^\circ$ into S_{i+1}. This may be worrying, as such an operation seems to be capable of breaking our invariant (B0).
- *Answering rank queries.* After processing the stream, our sketch will be presented with any element $x \in \mathcal{U}$, and must return $\widehat{\text{rank}}(x) \in [\text{rank}(x) - \epsilon n, \text{rank}(x) + \epsilon n]$. Currently, since our algorithm now uses two GK sketches, it might not be clear how we should define our estimator $\widehat{\text{rank}}$.

Proof of invariant (B0). To prove that (B0) holds, we first need the following claim:

Claim 15. *At any time t and representatives $e_i, e_j \in \mathcal{M}_t$ such that $e_i \prec e_j$, we have*

$$\text{if } t_i > t_j, \text{ then } S_i^\circ \prec e_j \quad \text{and} \quad \text{if } t_i < t_j, \text{ then } S_i \succ e_j.$$

Proof. Suppose $t_i > t_j$. Consider any stream element $x_{t'} \in S_i^\circ$. By invariant (B3), we know that $t' > t_i > t_j$.[2] At time t' (its insertion of $x_{t'}$), the representative e_j is already in $\mathcal{M}_t$.

Since we only merge adjacent representatives and now $x_{t'} \in S_i^\circ$ at time t, we must have $x_{t'} \prec e_j$. Otherwise, e_j should have been merged with e_i first. The other part follows from a symmetric proof. □

Lemma 16. *Invariant* (B0) *holds throughout our algorithm.*

Proof. We use induction and consider each insertion and merge. Suppose that the invariant (B0) holds before the operation. For an insertion, it is easy because upon inserting e_i, we shall have $S_i^\circ = S_i = \emptyset$.

Now suppose that we merge (e_i, e_{i+1}). Due to symmetry, we focus only on the case where $t_i > t_{i+1}$. In this case, we merge $S_i \cup S_i^\circ$ into S_{i+1}. By the inductive hypothesis, (B0) holds before this merge, and we have $S_i \prec e_i \prec e_{i+1}$ and $S_{i+1} \prec e_{i+1}$. From Claim 15, we have $S_i^\circ \prec e_{i+1}$. Therefore, $S_i \cup S_i^\circ \cup S_{i+1} \prec e_{i+1}$ as desired. □

Answering rank queries. Now, we will prove that our sketch can answer rank queries within the additive-error guarantee. Before we show this, we first need to show an analogous claim to Observation 4.

Observation 17. *For any element $e_i \in \mathcal{M}_t$, let $\text{rank}^{(t)}(e_i)$ be the rank of e_i with respect to the stream inserted until time t. Then, we see that $r_{\min}^{(t)}(e_i) \le \text{rank}^{(t)}(e_i) \le r_{\max}^{(t)}(e_i)$, where*

$$r_{\min}^{(t)}(e_i) \coloneqq i + \sum_{j=1}^{i} g_j + \sum_{j=1}^{i-1} g_j^\circ - \Delta_i^\circ \qquad \text{and} \qquad r_{\max}^{(t)}(e_i) \coloneqq i + \sum_{j=1}^{i} g_j + \Delta_i + \sum_{j=1}^{i-1} g_j^\circ.$$

Proof. We define $\text{rank}_{\mathcal{S}}^{(t)}(e_i)$ to be the number of elements in $\mathcal{S} = \bigcup_{e_j \in \mathcal{M}_t} S_j$ that are smaller than e_i, that is $\text{rank}_{\mathcal{S}}^{(t)}(e_i) = \{e \in \bigcup_{e_j \in \mathcal{M}_t} S_j \mid e \prec e_i\}$. Similarly, we define $\text{rank}_{\mathcal{S}^\circ}^{(t)}(e_i) = \{e \in \bigcup_{e_j \in \mathcal{M}_t} \mathcal{S}_j^\circ \mid e \prec e_i\}$. Then we have $\text{rank}^{(t)}(e_i) = \text{rank}_{\mathcal{S}}^{(t)}(e_i) + \text{rank}_{\mathcal{S}^\circ}^{(t)}(e_i) + i$. (Note $e_1, e_2, \ldots, e_{i-1}, e_i$ are not in these S_j's and S_j°'s.)

We know that

$$\text{rank}_{\mathcal{S}}^{(t)}(e_i) = \sum_{j=1}^{|\mathcal{M}_t|} |\{e \in S_j \mid e \prec e_i\}| = \sum_{j=1}^{i} |S_j| + \sum_{j=i+1}^{|\mathcal{M}_t|} |\{e \in S_j \mid e \prec e_i\}|.$$

[2] We remark that the proof of (B3) is starightforward and do not depend on (B0).

The second term is lower bounded by 0 and upper bounded by Δ_i. Similarly, due to the fact that for all $j \geq i$, $e_i \prec S_j^\circ$, we have

$$\operatorname{rank}_{S^\circ}^{(t)}(e_i) = \sum_{j=1}^{|\mathcal{M}_t|} |\{e \in S_j^\circ \mid e \prec e_i\}| = \sum_{j=1}^{i-1} |S_j^\circ| - |\{e \in S_j^\circ \mid e \prec e_i\}|.$$

The sum of the second term is upper bounded by 0 and lower bounded by $-\Delta_i^\circ$. Putting these together, we get

$$\sum_{j=1}^{i} g_j \leq \operatorname{rank}_{S}^{(t)}(e_i) \leq \Delta_i + \sum_{j=1}^{i} g_j \qquad \text{and} \qquad \sum_{j=1}^{i-1} g_j^\circ - \Delta_i^\circ \leq \operatorname{rank}_{S^\circ}^{(t)}(e_i) \leq \sum_{j=1}^{i-1} g_j^\circ.$$

Since $\operatorname{rank}^{(t)}(e_i) = \operatorname{rank}_{S}^{(t)}(e_i) + \operatorname{rank}_{S^\circ}^{(t)}(e_i) + i$, this concludes the proof. □

Lemma 18. *As long as invariant* (B2) *holds at time t, our sketch can answer any rank query $x \in \mathcal{U}$ with ϵt additive error.*

Proof. Let $i \in [|\mathcal{M}_t|]$ such that $e_i \preceq x \prec e_{i+1}$. We claim that estimator $\widehat{\operatorname{rank}}_t(x) = \frac{r_{\min}^{(t)}(e_i) + r_{\max}^{(t)}(e_{i+1}) - 1}{2}$ obtains an ϵ-approximation to $\operatorname{rank}^{(t)}(x)$. Since, $e_i \preceq x \prec e_{i+1}$, from Observation 17 we must have $r_{\min}^{(t)}(e_i) \leq \operatorname{rank}^{(t)}(e_i) \leq \operatorname{rank}^{(t)}(x) \leq \operatorname{rank}^{(t)}(e_{i+1}) - 1 \leq r_{\max}^{(t)}(e_{i+1}) - 1$. Hence,

$$|\widehat{\operatorname{rank}}^{(t)}(x) - \operatorname{rank}^{(t)}(x)| \;\leq\; \frac{r_{\max}^{(t)}(e_{i+1}) - 1 - r_{\min}^{(t)}(e_i)}{2} \;=\; \frac{(g_{i+1} + \Delta_{i+1}) + (g_i^\circ + \Delta_i^\circ)}{2} \leq \epsilon t$$

following the invariant (B2). □

4.3 Space Analysis

In this section, we adapt the potential argument of Section 3.2 and finish the proof of Theorem 10. Recall Definition 11 of mergeable pairs. The following lemma upper bounds the number of non-mergable pairs at any time t, thus proving the space bound of our algorithm.

Lemma 19. *At any time t, the total number of non-mergeable pairs (e_i, e_{i+1}) in $\mathcal{M}_t$ is at most $\frac{(2+o(1))\log(\epsilon t)}{\epsilon}$.*

Proof. For any stream element $x_{t'}$ inserted at time $t' \leq t$ (which might no longer reside in the memory), we assign it a *potential* given by

$$p_{x_{t'}}(t) = \frac{1}{1 + \epsilon(t - t')}.$$

On the one hand, the total potential of all elements is bounded. Since there is at most one $e_i \in \mathcal{M}_t$ with $t_i = t$, we only focus on elements inserted strictly before time t.

$$\sum_{t'=1}^{t-1} p_{x_{t'}}(t) = \sum_{t'=1}^{t-1} \frac{1}{1 + \epsilon(t - t')} = \frac{(1 + o(1))\log(\epsilon t)}{\epsilon}.$$

This step is exactly the same as that of Lemma 9. On the other hand, for any pair (e_i, e_{i+1}) that is non-mergeable. We want to prove that the total potential in $S_i \cup S_i^\circ \cup S_{i+1} \cup S_{i+1}^\circ \cup \{e_i, e_{i+1}\}$ is high. Let us focus on the case where $t_i > t_{i+1}$. In this case, we know that $g_i + g_i^\circ + g_{i+1} + \Delta_{i+1} + 1 > \epsilon t$

Efficient Data Structures for Our Algorithm

First of all, one can maintain $\mathcal{M}_t$ and tuples in $\mathcal{I}_t$ using a binary search tree (BST). This allows the insertion / deletion of $\mathcal{M}_t$ and $\mathcal{I}_t$ in $O(\log|\mathcal{M}_t|)$ time. Given e_i, it also supports access to predecessor / successor (e_{i-1} / e_{i+1}) in the same running time.

Then, let q be a priority queue that maintains items of form $(d, (e_j, e_{j+1}))$, where d is a number used as the key, and (e_j, e_{j+1}) are always two adjacent elements of $\mathcal{M}_t$. The item with smallest d is always at the top of q.

Initially, q is empty. Every time there is an insertion x_t, we run the following:

(1) Insert x_t using Algorithm 3. Suppose that x_t becomes the i^{th} smallest element in $\mathcal{M}_t$.
(2) Delete the old item in q that corresponds to (e_{i-1}, e_{i+1}). Then if $t_i > t_{i+1}$, insert a new item $(g_i + g_i^\circ + g_{i+1} + \Delta_{i+1} + 1, (e_i, e_{i+1}))$. Otherwise, $t_i < t_{i+1}$, and insert $(g_{i+1}^\circ + g_{i+1} + g_i^\circ + \Delta_i^\circ + 1, (e_i, e_{i+1}))$. We also insert a new item for (e_{i-1}, e_i) defined in a similar way.
(3) Suppose that the top item in q is $(d, (e_j, e_{j+1}))$. If $d \le \epsilon t$, we remove the top item from q and merge e_j and e_{j+1}. After the merge, we update q accordingly.[3](Note we only perform this merge once per insertion.)

Table 3. Our algorithm with efficient update time.

because (e_i, e_{i+1}) is non-mergeable. Then:

$$\begin{aligned}\sum_{x_{t'} \in S_i \cup S_i^\circ \cup S_{i+1} \cup S_{i+1}^\circ \cup \{e_i, e_{i+1}\}} p_{x_{t'}}(t) &\ge \sum_{x_{t'} \in S_i \cup S_i^\circ \cup S_{i+1} \cup \{e_i, e_{i+1}\}} \frac{1}{1+\epsilon(t-t')} \\ &\ge \frac{g_i + g_i^\circ + g_{i+1} + 2}{1+\epsilon(t-t_{i+1})} \qquad \text{(Invariant (B3) and } t_i > t_{i+1}) \\ &\ge 1.\end{aligned}$$

Here, the last step follows since $\Delta_{i+1} \le \epsilon t_{i+1}$ by invariant (B2) and $g_i + g_i^\circ + g_{i+1} + 1 \ge \epsilon t - \Delta_{i+1}$.

From symmetry, we know that in the case when $t_i < t_{i+1}$, the total potential in the non-mergable pair (e_i, e_{i+1}) is also at least 1. Each e_i, S_i, S_i° contributes to the potential of at most two non-mergeable pairs. As a result, there can be at most $\frac{(2+o(1))\log(\epsilon t)}{\epsilon}$ many non-mergeable pairs in $\mathcal{M}_t$. □

4.4 Efficient Data Structures for our Algorithm

In this section, we demonstrate a simple and efficient data structure implementation of our algorithm. See Table 3 for our implementation. Since our algorithm only checks whether adjacent pairs can be merged, it is much easier to implement compared to previous works. For weighted streams, this allows for an improvement in the time complexity. (See Lemma 24.)

Lemma 20. *The space complexity of our implementation is* $|\mathcal{M}_t| \le \frac{(2+o(1))\log(\epsilon t)}{\epsilon}$ *at any time* t.

Proof. Suppose $t' < t$ is the last time we did not perform a merge operation. Between t' and t, the size of the sketch does not increase ($|\mathcal{M}_t| \le |\mathcal{M}_{t'}|$). This is because each time we insert a new representative, there is always a merge operation performed. At time t', because the top item in the priority queue q is not a mergeable pair, there is no mergeable pair in $\mathcal{M}_{t'}$. Hence, by Lemma 19, there can be at most $\frac{(2+o(1))\log(\epsilon t)}{\epsilon}$ elements in $\mathcal{M}_{t'}$. □

Lemma 21. *Our algorithm has worst-case per-element running time* $O(\log(1/\epsilon) + \log\log(\epsilon t))$.

Proof. A priority queue supports insertion / deletion and update in $O(\log |q|)$ time, where $|q|$ is the number of elements in it. From the implementation, we know that $|q| \le |\mathcal{M}_t| - 1$. Anytime there are at most $\frac{(2+o(1))\log(\epsilon t)}{\epsilon}$ items in q, and after every insertion, we only perform the $O(1)$ priority queue operation. Hence, the running time is worst case $O(\log(1/\epsilon) + \log\log(\epsilon t))$ per element. Similarly, the running time of the BST is $O(\log |\mathcal{M}_t|)$ per element, which equals the same running time. □

5 GENERALIZATION FOR WEIGHTED STREAMS

Next, we consider a variant of the quantile estimation problem, where each stream element x_i appears together with a *weight* $w(x_i)$. Recently, the work of Assadi et al. [7] gave a nontrivial generalization of the GK sketch to weighted streams. Our approach also smoothly generalizes to weighted streams. Intuitively, we may interpret a stream element $(x_i, w(x_i))$ by considering x_i to appear $w(x_i)$ times consecutively in the stream. For notational convenience, we let $W_t = \sum_{t'=1}^{t} w(x_{t'})$ and $w(S) = \sum_{x \in S} w(x)$ for any set $S \subseteq \mathcal{U}$. Additionally, we note that if $e_i \in \mathcal{M}_t$ is an element such that $e_i \preceq x \prec e_{i+1}$, then the exact rank of query x with respect to the stream is given by $\mathrm{rank}^{(t)}(x) = \sum_{j=1}^{i} w(e_i) + 1$. Now, we consider the following generalized all-quantiles estimation problem:

Definition 22 (All-Quantiles Sketch for Weighted Streams)**.** *Given a input stream of pairs $(x_1, w(x_1))$, $(x_2, w(x_w)), \ldots, (x_n, w(x_n))$ arriving one at a time, let $\sum_{i=1}^{t} w(x_i) = W_t$ denote the total weight for the first t elements. An all-quantiles sketch approximates the rank of any query $x \in \mathcal{U}$ with $\widehat{\mathrm{rank}}^{(t)}(x)$ such that $|\widehat{\mathrm{rank}}^{(t)}(x) - \mathrm{rank}^{(t)}(x)| \le \epsilon W_t$.*

The following theorem extends our previous result to the weighted streaming setting.

Theorem 23. *Suppose that the minimum weight of the elements is $w_{\min}$. There is a deterministic comparison-based all-quantile sketch for weighted streams using $\frac{(2+o(1))\log(\epsilon \cdot W_t / w_{\min})}{\epsilon}$ space.*

5.1 Our algorithm for the weighted stream setting

In this section, we highlight essential changes to extend our algorithm to the weighted setting.

Representatives and Tuples. First of all, the representatives in $\mathcal{M}_t$ and the sets S_i, S_i° are defined in the same way as the unweighted case. Recall that e_i is the i^{th} smallest element stored in the memory $\mathcal{M}_t$. As before, we maintain the invariant $S_i \prec e_i \prec S_i^\circ$, and we store tuples of attributes for each representative e_i, defined as follows:

- G**-values**: In the weighted case, we let $G_i = w(S_i)$ and $G_i^\circ = w(S_i^\circ)$. Note that both G_i and G_i° exclude the weight contributed by element e_i itself.
- Δ**-values:** For any element $e_i \in \mathcal{M}_t$, we define Δ_i and Δ_i° to be upper bounds satisfying

$$\sum_{j \in \mathcal{M}_t, j > i} w(\{e \in S_j \mid e \prec i\}) \le \Delta_i \qquad \text{and} \qquad \sum_{j \in \mathcal{M}_t, j < i} w(\{e \in S_j^\circ \mid e \succ i\}) \le \Delta_i^\circ.$$

As before, we also maintain the insertion-time t_i for each representative e_i. We now describe the insertion and compression operations for our quantile summary in the weighted stream setting.

[3]To be more precise, suppose $t_j > t_{j+1}$ and e_j is the representative of the new set. Before merging, we have $e_{j-1}, e_j, e_{j+1}, e_{j+2} \in \mathcal{M}_t$. Afterwards, we have $e_{j-1}, e_j, e_{j+2} \in \mathcal{M}_t$. We already removed the top item $(d, (e_j, e_{j+1}))$ and need to further remove the items that corresponds to (e_{j+1}, e_{j+2}) from q. Then update the item that corresponds to (e_{j-1}, e_j) as the g value of e_j changes. Finally, we need to add to q a new item for (e_j, e_{j+2}).

Insertion. This part of the algorithm stays almost the same. When inserting x_t which is ranked the i^{th} in $\mathcal{M}_t$, we set $G_i = G_i^\circ = 0$ and $\Delta_i = G_{i+1} + \Delta_{i+1}$, $\Delta_i^\circ = G_{i-1}^\circ + \Delta_{i-1}^\circ$. Finally, we set the insertion time $t_i = t$.

Compression. The only change in compression is that we need to replace $g_i + g_i^\circ + g_{i+1} + 1$ with $G_i + G_i^\circ + G_{i+1} + w(e_i)$ (as well as a similar replacement for the other symmetric case).

Algorithm 5: Merging e_i, e_{i+1} (Simpler Algorithm for Weighted Streams)

Let $(e_i, e_{i+1}) \in \mathcal{M}_t$ be two adjacent elements in memory at time t.
if $t_i > t_{i+1}$ **then**
  if $G_i + G_i^\circ + G_{i+1} + w(e_i) + \Delta_{i+1} \le \epsilon W_t$ **then**
    $G_{i+1} \leftarrow G_i + G_i^\circ + w(e_i) + G_{i+1}$.
    Remove e_i from $\mathcal{M}_t$ and tuples $(G_i, \Delta_i), (G_i^\circ, \Delta_i^\circ)$ from $\mathcal{I}_t$.
else
  if $G_{i+1}^\circ + G_{i+1} + G_i^\circ + w(e_{i+1}) + \Delta_i^\circ \le \epsilon W_t$ **then**
    $G_i^\circ \leftarrow G_{i+1}^\circ + G_{i+1} + w(e_{i+1}) + G_i^\circ$.
    Remove e_{i+1} from $\mathcal{M}_t$ and tuples $(G_{i+1}, \Delta_{i+1}), (G_{i+1}^\circ, \Delta_{i+1}^\circ)$ from $\mathcal{I}_t$.

Invariants. The invariants are mostly unchanged compared to the unweighted setting. See Table 4.

Efficient and simplified implementation. Our algorithm for the weighted case can be implemented in the same way as in the unweighted setting, which is described in Section 4.4. Note that our result (stated formally below) improves upon the $O\left(\log(1/\epsilon) + \log\log(\epsilon W_n) + \frac{\log^2(\epsilon W_n)}{\epsilon n}\right)$ per-element running time of Assadi et al. [7], in the regime where the total weight $W_n \gg n$. This improvement directly results from the fact that our algorithm only merges *adjacent pairs* of elements.

Lemma 24. *Suppose that there are n elements in the stream and the minimum weight of the elements is $w_{\min}$. Our algorithm can be implemented in $O(\log(1/\epsilon) + \log\log(\epsilon W_n / w_{\min}))$ worst case update time per element.*

We defer the proofs of correctness and space analysis to the Appendix D, as they closely follow the proofs for the unweighted setting.

(C0)	$S_i \prec e_i \prec S_i^\circ,$	$\forall e_i \in \mathcal{M}_t$
(C1)	$\sum_{j=i+1}^{\lvert\mathcal{M}_t\rvert} w(\{e \in S_j \mid e \prec e_i\}) \le \Delta_i$ and $\sum_{j=1}^{i-1} w(\{e \in S_j^\circ \mid e \succ e_i\}) \le \Delta_i^\circ,$	$\forall e_i \in \mathcal{M}_t$
(C2)	$G_i + \Delta_i \le \epsilon W_t$ and $G_i^\circ + \Delta_i^\circ \le \epsilon W_t,$	$\forall e_i \in \mathcal{M}_t$
(C3)	$t' \ge t_i,$	$\forall e_i \in \mathcal{M}_t,\ x_{t'} \in S_i \cup S_i^\circ$

Table 4. Invariants of our algorithm for the weighted streaming setting.

REFERENCES

[1] [n. d.]. GKQuantiles Class - Micrometer Core 0.11.0.RELEASE Documentation. https://www.javadoc.io/doc/io.micrometer/micrometer-core/0.11.0.RELEASE/io/micrometer/core/instrument/stats/quantile/GKQuantiles.html. Accessed: 2023-11-15.

[2] [n. d.]. Problem 2: Quantiles - Open Problems in Sublinear Algorithms. https://sublinear.info/index.php?title=Open_Problems:2. Suggested by Graham Cormode, Source: Kanpur 2006, Accessed: 2023-11-15.

[3] [n. d.]. quantiles Crate Documentation - Rust. https://docs.rs/quantiles/latest/quantiles/. Accessed: 2023-11-15.

[4] Noga Alon, Omri Ben-Eliezer, Yuval Dagan, Shay Moran, Moni Naor, and Eylon Yogev. 2021. Adversarial laws of large numbers and optimal regret in online classification. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*. 447–455.

[5] Noga Alon, Yossi Matias, and Mario Szegedy. 1996. The space complexity of approximating the frequency moments. In *Proceedings of the twenty-eighth annual ACM symposium on Theory of computing*. 20–29.

[6] Michael Armbrust, Reynold S Xin, Cheng Lian, Yin Huai, Davies Liu, Joseph K Bradley, Xiangrui Meng, Tomer Kaftan, Michael J Franklin, Ali Ghodsi, et al. 2015. Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data*. 1383–1394.

[7] Sepehr Assadi, Nirmit Joshi, Milind Prabhu, and Vihan Shah. 2023. Generalizing Greenwald-Khanna Streaming Quantile Summaries for Weighted Inputs. *arXiv preprint arXiv:2303.06288* (2023).

[8] Omri Ben-Eliezer, Rajesh Jayaram, David P. Woodruff, and Eylon Yogev. 2020. A Framework for Adversarially Robust Streaming Algorithms. In *Proceedings on Database Systems*. arXiv:2003.14265 [cs.DS]

[9] Manuel Blum, Robert W. Floyd, Vaughan R. Pratt, Ronald L. Rivest, Robert Endre Tarjan, et al. 1973. Time bounds for selection. *J. Comput. Syst. Sci.* 7, 4 (1973), 448–461.

[10] Andrej Brodnik, Alejandro López-Ortiz, Venkatesh Raman, and Alfredo Viola. 2013. *Space-Efficient Data Structures, Streams, and Algorithms*. Springer.

[11] Graham Cormode, Zohar Karnin, Edo Liberty, Justin Thaler, and Pavel Veselỳ. 2021. Relative error streaming quantiles. In *Proceedings of the 40th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 96–108.

[12] Graham Cormode, Flip Korn, Shanmugavelayutham Muthukrishnan, and Divesh Srivastava. 2006. Space-and time-efficient deterministic algorithms for biased quantiles over data streams. In *Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. 263–272.

[13] Graham Cormode and Shan Muthukrishnan. 2005. An improved data stream summary: the count-min sketch and its applications. *Journal of Algorithms* 55, 1 (2005), 58–75.

[14] Graham Cormode and Pavel Veselỳ. 2020. A tight lower bound for comparison-based quantile summaries. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 81–93.

[15] Graham Cormode and Ke Yi. 2020. *Small summaries for big data*. Cambridge University Press.

[16] David Felber and Rafail Ostrovsky. 2017. A Randomized Online Quantile Summary in $O((1/\varepsilon)\log(1/\varepsilon))$ Words. *Theory of Computing* 13, 1 (2017), 1–17.

[17] Anna C Gilbert, Brett Hemenway, Atri Rudra, Martin J Strauss, and Mary Wootters. 2012. Recovering simple signals. In *2012 Information Theory and Applications Workshop*. IEEE, 382–391.

[18] Michael Greenwald and Sanjeev Khanna. 2001. Space-efficient online computation of quantile summaries. *ACM SIGMOD Record* 30, 2 (2001), 58–66.

[19] Anupam Gupta and Francis X. Zane. 2003. Counting Inversions in Lists. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms* (Baltimore, Maryland) *(SODA '03)*. Society for Industrial and Applied Mathematics, USA, 253–254.

[20] Moritz Hardt and David P Woodruff. 2013. How robust are linear sketches to adaptive inputs?. In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 121–130.

[21] Hemant Ishwaran, Udaya B Kogalur, and Maintainer Udaya B Kogalur. 2023. Package 'randomForestSRC'. *breast* 6, 1 (2023), 854.

[22] Zohar Karnin, Kevin Lang, and Edo Liberty. 2016. Optimal quantile approximation in streams. In *2016 ieee 57th annual symposium on foundations of computer science (focs)*. IEEE, 71–78.

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

[24] Moni Naor and Eylon Yogev. 2015. Bloom filters in adversarial environments. In *Annual Cryptology Conference*. Springer, 565–584.

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

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

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

A THE ORIGINAL GK SKETCH

For completeness, we present the procedure of the original GK sketch of [18]. In addition to the basic ingredients from Section 2.1, the original GK sketch also relies on the following more-involved components, which we briefly discussed in Section 3.3.

Bands. First, the GK sketch divides all tuples into "bands": in particular, at any time t and for any $e_i \in \mathcal{M}_t$, the authors define $\text{band}_t(e_i)$ to be an integer that satisfies the following two properties:

- Scale: $\text{band}_t(e_i) \approx \log_2(\epsilon t - \Delta_i)$;
- Consistency: for any two elements e_i and e_j, if $\text{band}_t(e_i) = \text{band}_t(e_j)$ at time t, for all future $t' > t$, we always have $\text{band}_{t'}(e_i) = \text{band}_{t'}(e_j)$.

To satisfy the two properties above, the band value is defined as

$$\text{band}_t(e_i) = \min\{\alpha \in \mathbb{Z} \mid \lfloor \epsilon t \rfloor - \Delta_i < 2^\alpha + \lfloor \epsilon t \rfloor \bmod 2^\alpha\}.$$

Observe that this definition clearly satisfies the first property. Intuitively, the second one holds, since we can equivalently write

$$\text{band}_t(e_i) = \min\{\alpha \in \mathbb{Z} \mid \lfloor \epsilon t \rfloor - 2^\alpha - \lfloor \epsilon t \rfloor \bmod 2^\alpha < \Delta_i\}.$$

So, we note that the α^{th}-band contains all elements i with $\Delta_i \in (\lfloor \epsilon t \rfloor - 2^\alpha - \lfloor \epsilon t \rfloor \bmod 2^\alpha, \lfloor \epsilon t \rfloor - 2^{\alpha-1} - \lfloor \epsilon t \rfloor \bmod 2^{\alpha-1}]$. Roughly speaking, the $\lfloor \epsilon t \rfloor \bmod 2^\alpha$ term cancels out the increase of $\lfloor \epsilon t \rfloor$ and keeps the boundary static. For a detailed proof that this definition satisfies the consistency property, readers may refer to the original paper [18].

Tree Structure and g^*-values. The idea of bands alone can only give a suboptimal space complexity of $O(\log(\epsilon n)^2/\epsilon)$. To achieve optimal space complexity, the original GK sketch and prior work need to arrange the elements in $\mathcal{M}_t$ into a tree structure.

For any element $e_i \in \mathcal{M}_t$, its parent on the tree is defined as

$$\text{parent}_t(e_i) = \min\{e_j \mid j > i \text{ and } \text{band}_t(e_j) > \text{band}_t(e_i)\}.$$

If no such element e_j exists, define the $\text{parent}_t(e_i)$ to be a special tree root R. Crucially, this tree structure maintains the invariant that the collection of all descendants of any node e_j forms a continuous interval of elements in $\mathcal{M}_t$, i.e. $e_i, e_{i+1}, \ldots, e_j$. Then, g_j^* is defined as the sum of the g-values of all these descendants, i.e. $g_j^* = g_i + g_{i+1} + \cdots + g_j$.

Algorithm 6: Compress the subtrees of e_i and e_{i+1} (GK sketch [18])

```
Let e_i, e_{i+1} ∈ M_t be two adjacent elements in memory.
if band_t(e_i) ≤ band_t(e_{i+1}) then
    if g*_i + g_{i+1} + Δ_{i+1} ≤ εt then
        g_{i+1} ← g*_i + g_{i+1}.
        Δ_{i+1} and t_{i+1} remain unchanged.
        Remove all descendants of e_i (including itself) from M_t and remove the
          corresponding tuples from I_t.
```

Procedure. Finally, we are ready to present the algorithm of the original GK sketch, which crucially relied on the two components defined above (in addition to $g-$ and Δ-values, as we described in Section 2.1). To insert a new stream element x_t, the insertion procedure is the same as Algorithm 1. But, the algorithm to merge pairs stored in $\mathcal{M}_t$ is more complex: after every $1/\epsilon$ insertions, the original GK sketch performs the compression procedure (see Algorithm 6) for each element $e_i \in \mathcal{M}_t$.

Importantly, we highlight that, unlike our simplified algorithm, it does not suffice to only consider merging adjacent elements e_i and e_{i+1} in the original GK sketch. Instead, the algorithm of [18] must check the entire subtree of e_i and e_{i+1}. However, it is not clear why merging entire subtrees should be necessary, and indeed, we give a simplified algorithm that avoids both the complex tree structure and the band-values entirely. We note that this tree structure makes implementing the original GK sketch quite difficult, and this is the main reason why the original GK sketch (with theoretical guarantees) is often not implemented in practice.

B PROOF OF LEMMA 9

To prove Lemma 9, we first need the following observation.

Observation 25. *At any time t in the algorithm, we always have $\Delta_{i+1} \le \epsilon t_{i+1} - 1$.*

Proof. When we insert e_{i+1} at time t_{i+1}, by Invariant (A1), we know that $g_{i+1} + \Delta_{i+1} \le \epsilon t$. As $g_{i+1} = 1$ since we just inserted e_{i+1}, we have $\Delta_{i+1} \le \epsilon t_{i+1} - 1$. Then since the Δ value is always unchanged during merging, the inequality continues to hold. □

Proof. (Proof of Lemma 9) We associate with each element $x_{t'}$ in the input stream (not necessarily a representative) a potential

$$p_{x_{t'}}(t) = \frac{1}{1 + \epsilon(t - t')}.$$

Note that this potential changes over time t. The recently inserted representatives weigh more than the old representatives.

On the one hand, we have an upper bound on the total potential of all elements inserted before t. (Note there is only one set S_i containing x_t. So it is without loss of generality to consider only elements inserted strictly before t.)

$$\sum_{t'=1}^{t-1} p_{x_{t'}}(t) = \sum_{t'=1}^{t-1} \frac{1}{1 + \epsilon(t - t')} = \frac{(1 + o(1))\log(\epsilon t)}{\epsilon}.$$

The last step holds because $\sum_{t'=1}^{t-1} \frac{1}{1+\epsilon(t-t')} = \sum_{t'=1}^{t-1} \frac{1}{1+\epsilon t'} \le \frac{1}{\epsilon} \cdot \sum_{x=1}^{1+\lceil \epsilon t \rceil} \frac{1}{x} = \frac{(1+o(1))\log(\epsilon t)}{\epsilon}$.

On the other hand, for any pair $(i, i+1)$ that is non-mergeable and $t_i > t_{i+1}$, we want to prove that the sets represented by them have a large total potential. For any such pair $(i, i+1)$, we must have $g_i + g_{i+1} + \Delta_{i+1} > \epsilon t$. The following observation upper bounds Δ_{i+1}.

Hence, we know that $g_i + g_{i+1} > 1 + \epsilon(t - t_{i+1})$. As a result,

$$\begin{aligned}
\sum_{x_{t'} \in S_i \cup S_{i+1}} p_{x_{t'}}(t) &= \sum_{x_{t'} \in S_i \cup S_{i+1}} \frac{1}{1 + \epsilon(t - t')} \\
&\ge \frac{|S_i \cup S_{i+1}|}{1 + \epsilon(t - \min(t_i, t_{i+1}))} && \text{(Invariant (A3))} \\
&\ge \frac{g_i + g_{i+1}}{1 + \epsilon(t - t_{i+1})} && (t_i > t_{i+1}) \\
&\ge 1 && \text{(Observation (25))}
\end{aligned}$$

To conclude the proof, the total potential is $\frac{(1+o(1))\log(\epsilon t)}{\epsilon}$ and each not-mergeable pair with $t_i > t_{i+1}$ has potential at least 1. Since each e_i may belong to two such pairs, there can be at most $\frac{(2+o(1))\log(\epsilon t)}{\epsilon}$ such pairs. □

C MISSING PROOFS FROM SECTION 4

We have shown, via Section 4.2, that invariant (B0) holds throughout the algorithm. Thus, it remains showing that invariants (B1), (B2), and (B3) are all maintained by our simplified algorithm for the unweighted stream setting.

Lemma 26. *Invariants* (B1), (B2), *and* (B3) *stated in Table 2 hold throughout the runtime of our algorithm.*

Proof. To show that these invariants are maintained throughout our algorithm, we will show that they are each maintained after insertion and compression operations.

(1) Invariant (B1): $\forall e_i \in \mathcal{M}_t$, $\sum_{j=i+1}^{|\mathcal{M}_t|} |\{e \in S_j | e \prec e_i\}| \le \Delta_i$ and $\sum_{j=1}^{i-1} |\{e \in S_j^\circ | e \succ e_i\}| \le \Delta_i^\circ$.
First, suppose we insert a new element x_t which is the i^{th}-smallest element in $\mathcal{M}_t$. Then, we set $\Delta_i = g_{i+1} + \Delta_{i+1}$. Since Δ_{i+1} was already a valid upper bound on the number of elements $e \in S_j$ for $j \ge i+2$ such that $e \prec e_i$ and g_{i+1} is the number of other elements that are represented by e_{i+1}, it follows directly that $\forall e_i \in \mathcal{M}_t$, $\sum_{j=i+1}^{|\mathcal{M}_t|} |\{e \in S_j | e \prec e_i\}| \le \Delta_i$ upon insertion (this is also shown for the toy algorithm in Claim 6). The analogous inequality for Δ_i° follows by a similar argument.
Next, suppose two adjacent elements $(e_i, e_{i+1}) \in \mathcal{M}_t$ are mergeable and we perform a compression operation. There are two cases here: if $t_i > t_{i+1}$, we set $g_{i+1} \leftarrow g_i + g_i^\circ + g_{i+1} + 1$ and leave Δ_i unchanged; moreover, we interpret this as merging sets $S_i \cup S_i^\circ \cup \{e_i\}$ into S_{i+1}. In fact, we can see that invariant (B1) directly carries over from before the merge operation, so the claim follows by induction. The case that $t_i < t_{i+1}$ can be shown similarly.

(2) Invariant (B2): $\forall e_i \in \mathcal{M}_t$, $g_i + \Delta_i \le \epsilon t$ and $g_i^\circ + \Delta_i^\circ \le \epsilon t$.
For any insertion x_t which is the i^{th}-smallest element in $\mathcal{M}_t$, we initialize $g_i = 0$, $S_i = \emptyset$, and set $\Delta_i = g_{i+1} + \Delta_{i+1}$. Then, it follows trivially that $g_i + \Delta_i = g_{i+1} + \Delta_{i+1} \le \epsilon t$. Likewise, by initializing the "mirrored" attributes as $g_i^\circ = 0$, $S_i^\circ = \emptyset$, and $\Delta_i^\circ = g_{i-1}^\circ + \Delta_{i-1}^\circ$, we also have that $g_i^\circ + \Delta_i^\circ = g_{i-1}^\circ + \Delta_{i-1}^\circ \le \epsilon t$.
Now, we show that this property is preserved after compressing a mergeable pair of adjacent elements $(e_i, e_{i+1}) \in \mathcal{M}_t$. For simplicity, we consider the case that $t_i > t_{i+1}$ and we set $g_{i+1} \leftarrow g_i + g_i^\circ + g_{i+1} + 1$ (the other case that $t_{i+1} > t_i$ will be symmetric). In fact, by the definition of our algorithm, we can merge (e_i, e_{i+1}) (with $t_i > t_{i+1}$) only if $g_i + g_i^\circ + g_{i+1} + \Delta_{i+1} \le \epsilon t$; this requirement enforces the property that, after merging, we maintain $g_{i+1} + \Delta_{i+1} \le \epsilon t$ (and the tuples for all other elements are unchanged).

(3) Invariant (B3): $\forall x_{t'} \in S_i \cup S_i^\circ$, $t' \ge t_i$, where t_i is the insertion time of representative e_i.
Suppose we insert an element x_t which is the i^{th}-smallest element in $\mathcal{M}_t$. Then, by definition, we set $g_i = g_i^\circ = 0$, and think of $S_i = S_i^\circ = \emptyset$; also, we initialize $t_i \leftarrow t$. So, at the time of insertion, this invariant is trivially satisfied.
Now, we check that this property is maintained after any merge operation. Suppose $(e_i, e_{i+1}) \in \mathcal{M}_t$ is a mergeable pair of adjacent elements, and we assume that the invariant holds at time $t-1$. Importantly, we recall that our algorithm works as follows: if $t_i > t_{i+1}$ and merging would not disrupt the additive-error guarantee, then we merge $S_i \cup S_i^\circ$ into S_{i+1}. Since the invariant held at $t-1$, we know that every $x \in S_i \cup S_i^\circ$ had $t_x > t_i$; thus, it follows that $t_x > t_i > t_{i+1}$ for every $x \in S_{i+1}$ after merging. Likewise, we can argue the case that $t_i < t_{i+1}$ via a very similar argument.

This completes the proof. □

D MISSING PROOFS FROM SECTION 5

In this section, we show that the generalization of our algorithm to weighted streams illustrated in Section 5 preserves correctness of approximation and desirable space usages. Our proofs follow closely from those in Section 4.3 and 4.2.

D.1 Correctness

We first discuss the correctness of our algorithm for weighted streams. With the same proofs as Lemma 26, we can show that the invariants (C0) , (C1), (C2), and (C3) are maintained. For the sake of succinctness, we shall omit these proofs.

Crucially, we need to justify answering rank query in the weighted setting. Note that they follow closely the proofs of Observation 17 and Lemma 18.

Observation 27. *For any element $e_i \in \mathcal{M}_t$, let $\mathrm{rank}^{(t)}(e_i)$ be the rank of e_i with respect to the stream inserted until time t. Then, we see that $r^{(t)}_{\min}(e_i) \le \mathrm{rank}^{(t)}(e_i) \le r^{(t)}_{\max}(e_i)$, where*

$$r^{(t)}_{\min}(e_i) \coloneqq \sum_{j=1}^{i} w(e_j) + \sum_{j=1}^{i} G_j + \sum_{j=1}^{i-1} G^\circ_j - \Delta^\circ_i \quad \text{and} \quad r^{(t)}_{\max}(e_i) \coloneqq \sum_{j=1}^{i} w(e_j) + \sum_{j=1}^{i} G_j + \Delta_i + \sum_{j=1}^{i-1} G^\circ_j.$$

Proof. We define $\mathrm{rank}^{(t)}_{\mathcal{S}}(e_i)$ to be the total weights of elements in $\bigcup_{e_j \in \mathcal{M}_t} S_j$ that are smaller than e_i, that is $\mathrm{rank}^{(t)}_{\mathcal{S}}(e_i) = w(\{e \in \bigcup_{e_j \in \mathcal{M}_t} S_j \mid e \prec e_i\})$. Similarly, we define $\mathrm{rank}^{(t)}_{\mathcal{S}^\circ}(e_i) = w(\{e \in \bigcup_{e_j \in \mathcal{M}_t} S^\circ_j \mid e \prec e_i\})$. Then we have $\mathrm{rank}^{(t)}(e_i) = \mathrm{rank}^{(t)}_{\mathcal{S}}(e_i) + \mathrm{rank}^{(t)}_{\mathcal{S}^\circ}(e_i) + \sum_{j'=1}^{i} w(e_{j'})$. (Note $e_1, e_2, \ldots, e_{i-1}, e_i$ are not in these S_j's and S°_j's.)

We know that

$$\begin{aligned}\mathrm{rank}^{(t)}_{\mathcal{S}}(e_i) &= \sum_{j=1}^{|\mathcal{M}_t|} w(\{e \in S_j \mid e \prec e_i\}) \\ &= \sum_{j=1}^{i} w(S_j) + \sum_{j=i+1}^{|\mathcal{M}_t|} w(\{e \in S_j \mid e \prec e_i\}).\end{aligned}$$

The second term is lower bounded by 0 and upper bounded by Δ_i. Similarly, due to the fact that for all $j \ge i$, $e_i \prec S^\circ_j$, we have

$$\mathrm{rank}^{(t)}_{\mathcal{S}^\circ}(e_i) = \sum_{j=1}^{|\mathcal{M}_t|} w(\{e \in S^\circ_j \mid e \prec e_i\}) = \sum_{j=1}^{i-1} w(S^\circ_j) - w(\{e \in S^\circ_j \mid e \prec e_i\})/$$

The sum of the second term is upper bounded by 0 and lower bounded by $-\Delta^\circ_i$.

Putting these together, we get

$$\sum_{j=1}^{i} G_j \le \mathrm{rank}^{(t)}_{\mathcal{S}}(e_i) \le \Delta_i + \sum_{j=1}^{i} G_j \quad \text{and} \quad \sum_{j=1}^{i-1} G^\circ_j - \Delta^\circ_i \le \mathrm{rank}^{(t)}_{\mathcal{S}^\circ}(e_i) \le \sum_{j=1}^{i-1} G^\circ_j.$$

Together with $\mathrm{rank}^{(t)}(e_i) = \mathrm{rank}^{(t)}_{\mathcal{S}}(e_i) + \mathrm{rank}^{(t)}_{\mathcal{S}^\circ}(e_i) + \sum_{j'=1}^{i} w(e_{j'})$, this finishes the proof. □

Lemma 28. *As long as the invariant* (C2) *holds at time t, our sketch can answer any rank query $x \in \mathcal{U}$ with ϵW_t additive error.*

Proof. Let $i \in [|\mathcal{M}_t|]$ such that $e_i \preceq x \prec e_{i+1}$. We claim that the estimator $\widehat{\text{rank}}_t(x) = \frac{r^{(t)}_{\min}(e_i)+r^{(t)}_{\max}(e_{i+1})-w(e_{i+1})}{2}$ obtains an ϵ-approximation to $\text{rank}^{(t)}(x)$.

Since, $e_i \preceq x \prec e_{i+1}$, from Observation 17 we must have $r^{(t)}_{\min}(e_i) \le \text{rank}^{(t)}(e_i) \le \text{rank}^{(t)}(x) \le \text{rank}^{(t)}(e_{i+1}) - w(e_{i+1}) \le r^{(t)}_{\max}(e_{i+1}) - w(e_{i+1})$. Hence,

$$\begin{aligned}|\widehat{\text{rank}}^{(t)}(x) - \text{rank}^{(t)}(x)| &\le \frac{r^{(t)}_{\max}(e_{i+1}) - w(e_{i+1}) - r^{(t)}_{\min}(e_i)}{2} \\ &= \frac{(G_{i+1} + \Delta_{i+1}) + (G^\circ_i + \Delta^\circ_i)}{2} \\ &\le \epsilon W_t\end{aligned}$$

following the invariant (B2). □

D.2 Space Analysis

In this section, we adapt the potential argument of Section 3.2 and finish the proof of Theorem 10. The most important change is the change in the potential function. Recall that $w_{\min}$ is the smallest weight of all elements.

Lemma 29. *At any time t, the total number of pairs (e_i, e_{i+1}) in $\mathcal{M}_t$ that cannot be merged is at most $\frac{(2+o(1))\log(\epsilon \cdot W_t/w_{\min})}{\epsilon}$.*

Proof. For any stream element $x_{t'}$ inserted at time $t' \le t$ (which might no longer reside in the memory), we assign it a *potential* given by

$$p_{x_{t'}}(t) = \frac{w(x_{t'})}{w_{\min} + \epsilon(W_t - W_{t'-1})}.$$

On the one hand, the total potential of all elements is bounded. Since there is at most one $e_i \in \mathcal{M}_t$ with $t_i = t$, we only focus on elements inserted strictly before time t.

$$\sum_{t'=1}^{t-1} p_{x_{t'}}(t) \;=\; \sum_{t'=1}^{t-1} \frac{w(x_{t'})}{w_{\min} + \epsilon(W_t - W_{t'-1})} \le \frac{(1+o(1))\log(\epsilon W_t/w_{\min})}{\epsilon}$$

where the final inequality follows from Fact 31 in Appendix E.

On the other hand, for any pair (e_i, e_{i+1}) that is not mergeable, we want to prove that the total potential in $S_i \cup S^\circ_i \cup S_{i+1} \cup S^\circ_{i+1} \cup \{e_i, e_{i+1}\}$ is high. Focusing on the case where $t_i > t_{i+1}$, we know that $g_i + g^\circ_i + g_{i+1} + \Delta_{i+1} + w(e_i) > \epsilon W_t$. Then, we have that

$$\begin{aligned}\sum_{x_{t'} \in S_i \cup S^\circ_i \cup S_{i+1} \cup S^\circ_{i+1} \cup \{e_i, e_{i+1}\}} p_{x_{t'}}(t) \;&\ge \sum_{x_{t'} \in S_i \cup S^\circ_i \cup S_{i+1} \cup \{e_i, e_{i+1}\}} \frac{w(x_{t'})}{w_{\min} + \epsilon(W_t - W_{t'-1})} \\ &\ge \frac{G_i + G^\circ_i + G_{i+1} + w(e_i) + w(e_{i+1})}{w_{\min} + \epsilon(W_t - W_{t_{i+1}-1})} \\ &\qquad \text{(Invariant (C3) and } t_i > t_{i+1}) \\ &\ge \frac{w(e_{i+1}) + \epsilon W_t - \Delta_{i+1}}{w_{\min} + \epsilon(W_t - W_{t_{i+1}-1})} \qquad \text{(Non-mergeability)} \\ &\ge \frac{w_{\min} + \epsilon W_t - \Delta_{i+1}}{w_{\min} + \epsilon(W_t - W_{t_{i+1}-1})} \qquad (w_{e_{i+1}} \ge w_{\min}) \\ &\ge 1\end{aligned}$$

Here, the last step follows from the fact that, when we insert e_{i+1}, by invariant (C2), we have $\Delta_{i+1} = G_i + \Delta_i \le \epsilon W_{t_{i+1}-1}$.

From symmetry, we see that in the case when $t_i < t_{i+1}$, the total potential for any not mergeable pair (e_i, e_{i+1}) is also at least 1. Also, e_i, S_i, S_i° contributes to the potential of at most two not mergeable pairs. As a result, there can be at most $\frac{(2+o(1))\log(\epsilon \cdot W_t/w_{\min})}{\epsilon}$ many not mergeable pairs in $\mathcal{M}_t$, as desired. □

Corollary 30. *When all weights $w_i \le poly(n)$, the space required for our ϵ-approximate quantile summary on weighted streams is at most $O(\epsilon^{-1}\log(\epsilon n))$.*

E MISCELLANEOUS

In this part, we provide the omitted proof for Fact 31.

Fact 31. *For any $a_1, ..., a_t \in \mathbb{R}^+$ and $a_0 \le \min\{a_1, ..., a_t\}$, we have*

$$\sum_{k=1}^{t} \frac{a_k}{a_0 + \epsilon(a_k + ... + a_t)} \le \frac{1+o(1)}{\epsilon} \log\left(\frac{\epsilon \cdot (a_1 + ... + a_t)}{a_0}\right).$$

Proof. First, we upper bound each term $\frac{a_k}{a_0+\epsilon(a_k+...+a_t)}$. Observe that

$$\begin{aligned}
\frac{a_k}{a_0 + \epsilon(a_k + ... + a_t)} &= \sum_{\ell=1}^{a_k} \frac{1}{a_0 + \epsilon(a_k + ... + a_t)} \\
&\le \sum_{v=1}^{\epsilon a_k} \frac{\frac{1}{\epsilon}}{a_0 + \epsilon(a_{k+1} + ... + a_t) + v} \\
&\le \frac{1+o(1)}{\epsilon} \log\left(\frac{a_0 + \epsilon(a_k + ... + a_t)}{a_0 + \epsilon(a_{k+1} + ... + a_t)}\right)
\end{aligned}$$

where the last inequality follows by making a change of variables and applying the fact that $\sum_{i=1}^{t} \frac{1}{t} = (1+o(1))\log(t)$. Summing over all $k \in [t]$ and applying the term-wise upper bound from above, we see that

$$\begin{aligned}
\sum_{k=1}^{t} \frac{a_k}{a_0 + \epsilon(a_k + ... + a_t)} &\le \frac{1+o(1)}{\epsilon} \sum_{k=1}^{t} \log\left(\frac{a_0 + \epsilon(a_k + ... + a_t)}{a_0 + \epsilon(a_{k+1} + ... + a_t)}\right) \\
&= \frac{1+o(1)}{\epsilon} \log\left(\frac{\epsilon(a_1 + ... + a_t)}{a_0}\right)
\end{aligned}$$

as desired. □

Received December 2023; revised February 2024; accepted March 2024