



Generalizing Greenwald-Khanna Streaming Quantile Summaries for Weighted Inputs*

Sepehr Assadi  

Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Nirmit Joshi  

Department of Computer Science, Northwestern University, Evanston, IL, USA

Milind Prabhu  

Department of Computer Science and Engineering, University of Michigan, Ann Arbor, MI, USA

Vihan Shah  

Department of Computer Science, Rutgers University, Piscataway, NJ, USA

Abstract

Estimating quantiles, like the median or percentiles, is a fundamental task in data mining and data science. A (streaming) quantile summary is a data structure that can process a set S of n elements in a streaming fashion and at the end, for any $\phi \in (0, 1]$, return a ϕ -quantile of S up to an ε error, i.e., return a ϕ' -quantile with $\phi' = \phi \pm \varepsilon$. We are particularly interested in comparison-based summaries that only compare elements of the universe under a total ordering and are otherwise completely oblivious of the universe. The best known deterministic quantile summary is the 20-year old Greenwald-Khanna (GK) summary that uses $O((1/\varepsilon) \log(\varepsilon n))$ space [SIGMOD'01]. This bound was recently proved to be optimal for all deterministic comparison-based summaries by Cormode and Vesl y [PODS'20].

In this paper, we study weighted quantiles, a generalization of the quantiles problem, where each element arrives with a positive integer weight which denotes the number of copies of that element being inserted. The only known method of handling weighted inputs via GK summaries is the naive approach of breaking each weighted element into multiple unweighted items, and feeding them one by one to the summary, which results in a prohibitively large update time (proportional to the maximum weight of input elements).

We give the first non-trivial extension of GK summaries for weighted inputs and show that it takes $O((1/\varepsilon) \log(\varepsilon n))$ space and $O(\log(1/\varepsilon) + \log \log(\varepsilon n))$ update time per element to process a stream of length n (under some quite mild assumptions on the range of weights and ε). En route to this, we also simplify the original GK summaries for unweighted quantiles.

2012 ACM Subject Classification Theory of computation \rightarrow Streaming, sublinear and near linear time algorithms; Theory of computation \rightarrow Approximation algorithms analysis; Theory of computation \rightarrow Data structures design and analysis

Keywords and phrases Streaming algorithms, Quantile summaries, Rank estimation

Digital Object Identifier 10.4230/LIPIcs.ICDT.2023.19

Funding *Sepehr Assadi*: Research supported in part by the NSF CAREER Grant CCF-2047061, and gift from Google Research.

Vihan Shah: Research supported in part by the NSF CAREER Grant CCF-2047061. Part of this work was done when the author was an undergraduate student at Rutgers University-Camden and was supported in part by the NSF grant CCF-1910565.

Acknowledgements We would like to thank Rajiv Gandhi for making the collaboration between the authors possible and for his support throughout this project.

* A full version of the paper with the same title appears on arXiv.



1 Introduction

Given a set S of elements x_1, \dots, x_n from a totally ordered universe, the rank of an element x in this universe, denoted by $\text{rank}(x)$, is the number of elements x_j in S with $x_j \leq x$. Similarly, the ϕ -quantile of S , for any $\phi \in (0, 1]$, is the element $x_i \in S$ with $\text{rank}(x_i) = \lceil \phi \cdot n \rceil$. Computing quantiles is a fundamental problem with a wide range of applications considering they provide a concise representation of the distribution of the input elements. Throughout this paper, we solely focus on comparison-based algorithms for this problem that can only compare two elements of the universe according to their ordering and are otherwise completely oblivious of the universe.

We are interested in the quantile estimation problem in the *streaming* model, introduced in the seminal work of Alon, Matias, and Szegedy [3]. In this model, the elements of S are arriving one by one in an arbitrary order and the streaming algorithm can make just one pass over this data and use a limited memory and thus cannot simply store S entirely. Already more than four decades ago, Munro and Paterson proved that one cannot solve this problem *exactly* in the streaming model [17] and thus the focus has been on finding *approximation* algorithms: Given $\varepsilon > 0$, the algorithm is allowed to return an ε -approximate ϕ -quantile, i.e., a $(\phi \pm \varepsilon)$ -quantile. More formally, we are interested in the following data structure:

► **Definition 1** (Quantile Summary). *An ε -approximate quantile summary processes any set of elements in a streaming fashion and at the end finds an ε -approximate ϕ -quantile for any given quantile $\phi \in (0, 1]$, defined as any ϕ' -quantile for $\phi' \in [\phi - \varepsilon, \phi + \varepsilon]$.*

In the absence of the streaming aspect of the problem, one can always compute an ε -approximate quantile summary in $O(1/\varepsilon)$ space; simply store the ε -quantile, 3ε -quantile, 5ε -quantile and so on from S . It is easy to see that given any ϕ , returning the closest stored quantile results in an ε -approximate ϕ -quantile. It is also easy to see that this space is information-theoretically optimal for the problem. However, this approach cannot be directly implemented in the streaming model as a-priori it is not clear how to compute the needed quantiles of S in the first place.

The first (streaming) ε -approximate quantile summary was proposed by Manku, Rajagopalan, and Lindsay [14]. The MRL summary uses $O((1/\varepsilon) \log^2(\varepsilon n))$ space and requires the prior knowledge of the length of the stream. This summary was soon after improved by Greenwald and Khanna [9] who proposed the GK summary that uses $O((1/\varepsilon) \log(\varepsilon n))$ space and no longer requires knowing the length of the stream. This is the state-of-the-art for deterministic comparison-based summaries. By allowing for randomization one can further improve upon the space requirement of these algorithms and achieve bounds with no dependence on the length of the stream. The state-of-the-art result for randomized summaries is an algorithm due to Karnin, Lang and Liberty [12] which uses $O((1/\varepsilon) \log \log(1/\varepsilon \delta))$ space to construct an ε -approximate quantile summary with probability at least $(1 - \delta)$. We provide a more detailed discussion of the literature on randomized summaries and non-comparison based summaries in the full version of the paper. While using randomization gives streaming algorithms which are more space-efficient, a major drawback of most of these algorithms is that their analysis crucially depends on the assumption that the input stream is independent of the randomness used by the algorithm. This assumption is unrealistic in several settings; for instance, when the future input to the algorithm depends on its previous outputs. Recently, this has invoked an interest in *adverserially robust* algorithms that work even when an adversary is allowed to choose the stream adaptively [16, 7, 10, 2, 8, 18]. Deterministic algorithms are inherently adverserially robust and therefore understanding them is an interesting goal in itself.

In this paper, we focus on deterministic summaries; specifically on furthering our understanding of GK summaries. Over the years, two important questions have been raised about them: Is it possible to *improve* the space of GK summaries, perhaps even all the way down to the information-theoretic optimal bound of $O(1/\varepsilon)$? And, is it possible to *simplify* GK summaries and their intricate analysis in a way that allow for generalizations of these summaries to be more easily proposed and studied? (see Problem 2 of “List of Open Problems in Sublinear Algorithms” [19] posed by Cormode or [12, 6, 13, 1] for similar variations of this question, for example, when the input items are weighted).

The first question was addressed initially by Hung and Ting [11] who proved an $\Omega((1/\varepsilon) \log(1/\varepsilon))$ space lower bound for ε -approximate quantile summaries, improving over the information-theoretic bound. Very recently, this question was fully settled by Cormode and Vesléy [6] who proved that in fact GK summaries are asymptotically optimal: $\Omega((1/\varepsilon) \log(\varepsilon n))$ space is needed for any deterministic (comparison-based) summary. The second question above however is still left without a satisfying resolution. In this paper, we make progress towards answering this question by showing that the GK summary can be generalized to handle weighted inputs. Formally, we present algorithms to construct the following data-structure.

► **Definition 2 (Weighted Quantile Summary).** *Consider a weighted stream S_w of n updates $(x_i, w(x_i))$ for $1 \leq i \leq n$. The i -th update denotes the insertion of $w(x_i)$ copies of the element x_i (the weight $w(x_i)$ is guaranteed to be a positive integer). We define $W_k = \sum_{i=1}^k w(x_i)$ to be the sum of the weights of the first k elements of S_w . An ε -approximate weighted quantile summary is a data-structure that makes a single pass over S_w and at the end, for any $\phi \in [0, 1)$, finds an x_j such that,*

$$\left(\sum_{x_i < x_j} w(x_i), w(x_j) + \sum_{x_i < x_j} w(x_i) \right) \cap [(\phi - \varepsilon)W_n, (\phi + \varepsilon)W_n] \neq \emptyset. \quad (1)$$

A notable application of the weighted quantiles problem is in the very popular XGBoost library [5] which contains an efficient implementation of the gradient-boosted trees algorithm. To solve the weighted quantiles problem, XGBoost uses a merge and prune summary [4] via an extension of the ideas in [15]. However, they do not give an upper bound on the space achieved by this summary. Our result addresses this issue by proposing a new and efficient weighted quantile summary with formal space guarantees.

Our Contributions

One approach to construct a weighted quantile summary is to break each weighted item into multiple unweighted items and feed them to an unweighted summary such as the GK summary. However, such algorithms are slow since the time required to process an element is proportional to its weight. As such, it has been asked if faster algorithms exist. We answer this in the affirmative by proposing a fast algorithm for this problem in Section 3. In particular, this algorithm uses $O((1/\varepsilon) \log(\varepsilon n))$ space and $O(\log(1/\varepsilon) + \log \log(\varepsilon n))$ update time per element to process a stream of length n , when the weights are $\text{poly}(n)$ and $\varepsilon \geq \frac{1}{n^{1-\delta}}$ for any $\delta \in (0, 1)$ (Theorem 11). This matches the space and time complexity of the GK summary when it is used to summarize a stream of n unweighted items [9, 13]. To our knowledge, this constitutes the first (non-trivial) extension of the GK algorithm for weighted streams.

En route to this, we also present a new description of the GK summaries by simplifying or entirely bypassing several of their more intricate components in [9] such as their so-called “tree representation” and their complex “compress” operations in Section 4.2. As a warm-up to this, we also present a simple and greedy algorithm for unweighted quantiles which uses $O((1/\varepsilon) \log^2(\varepsilon n))$ space in Section 4.1. This algorithm, although has a suboptimal space bound, will be useful in motivating and providing intuition for GK summaries. Interestingly, this summary is quite similar (albeit *not* identical) to the so-called GKAdaptive summary [13] that was already proposed by [9] as a more practical variant of their GK summaries (Luo *et al.* [13] further confirmed this by showing that GKAdaptive outperforms GK summaries experimentally). While no theoretical guarantees are known for GKAdaptive, we prove that this slight modification of this algorithm submits to a simple analysis of an $O((1/\varepsilon) \log^2(\varepsilon n))$ space upper bound (Theorem 25).

We also emphasize that, similar to the original GK summaries, our weighted extension does not need foreknowledge of the stream length. This guarantee implies that we can track the quantiles throughout the stream, with error proportional to the current weight of the stream, and not only at the end.

2 Preliminaries

We now present the basic setup of our quantile summary and preliminary definitions. We start with an alternate equivalent formulation of the problem defined in Definition 2 in terms of the unweighted quantiles problem for which we first define the notion of *unfolding* streams.

Unfolding Streams

For the weighted stream S_w , we define its corresponding unfolded stream $\text{Unfold}(S_w)$ to be the stream which contains $w(x_i)$ copies of x_i for $1 \leq i \leq n$. More explicitly,

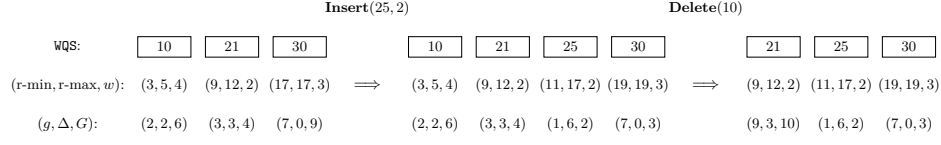
$$\text{Unfold}(S_w) := \langle x_1^{(1)}, x_1^{(2)}, \dots, x_1^{(w(x_1))}, \dots, x_n^{(1)}, x_n^{(2)}, \dots, x_n^{(w(x_n))} \rangle$$

where $x_i^{(j)}$ is the j -th copy of element x_i .

It is easy to verify that the goal of the problem, as stated in Definition 2, is equivalent to creating an ε -approximate quantile summary of $\text{Unfold}(S_w)$. Note that to break ties while assigning ranks to equal elements, we will assume that elements that appeared earlier in $\text{Unfold}(S_w)$ have lower ranks. As a side note we would like to point out here that although the algorithm we present does not “unfold” the stream, we will continue working with $\text{Unfold}(S_w)$ to present the analysis of the algorithm.

We use WQS to denote the summary of S_w that our algorithm creates. WQS will consist of a subset of the elements of the stream along with some auxiliary metadata about the stored elements. We use e_i to denote the i -th largest element of the stream stored in WQS . We use $e_i^{(j)}$ to refer to the the j -th copy of e_i in $\text{Unfold}(S_w)$, for $1 \leq j \leq w(e_i)$. We also use e to refer to an arbitrary element of the summary (when the rank is not relevant). The number of elements of the stream stored in WQS shall be denoted by s . For each element e , WQS stores $w(e)$. The other main information we store for each element e are its r-min and r-max values, which we now define:

- r-min(e) and r-max(e): are lower and upper bounds maintained by WQS on the rank of $e^{(1)}$ (the first copy of e to appear in $\text{Unfold}(S_w)$). Since we are not storing all elements, we cannot determine the exact rank of a stored element, and thus focus on maintaining proper lower and upper bounds.



■ **Figure 1** An illustration of the update operations in the summary starting from some arbitrary state (the parameters (g, Δ, G) in this figure are defined in Section 2.2).

To handle corner cases that arise later, we assume that WQS contains a sentinel element e_0 and define $\text{r-min}(e_0) = \text{r-max}(e_0) = 0$ and $w(e_0) = 1$. Also, we insert a $+\infty$ element at the start of the stream which is considered larger than any other element and store it in WQS as e_s . The r-min and r-max of this element is also always equal to the weight of inserted elements (including itself). Since $+\infty$ is the largest element, inserting it in S_w does not affect the rank of any other element.

► **Observation 3.** $(\text{r-min}(e) + j - 1)$ and $(\text{r-max}(e) + j - 1)$ are lower and upper bounds on the rank of $e^{(j)}$.

During the stream, we insert and delete elements from the summary. This changes the rank of the elements so we have to update WQS to reflect the changes. The procedure used to update the r-min and r-max values of elements is describe below:

Insert($x, w(x)$). Inserts a given element x with weight $w(x)$ into WQS.

- (i) Store the element x along with its weight $w(x)$ in WQS.
- (ii) Find the smallest element e_i in WQS such that $e_i > x$;
- (iii) Set $\text{r-min}(x) = \text{r-min}(e_{i-1}) + w(e_{i-1})$ and $\text{r-max}(x) = \text{r-max}(e_i)$; moreover, increase $\text{r-min}(e_j)$ and $\text{r-max}(e_j)$ by $w(x)$ for all $j \geq i$.

Delete(e_i). Deletes the element e_i from WQS.

- (i) Remove element e_i from WQS; keep all remaining r-min, r-max values unchanged.

We now justify that after the above operations are performed, for each element e in the summary, its r-min and r-max values are valid lower and upper bounds on the rank of $e^{(1)}$. Suppose that a new element x satisfying $e_{i-1} < x < e_i$ is inserted into WQS. The rank of x is at least one more than the rank of the last copy of e_{i-1} . Therefore, $\text{r-min}(x)$, which is set to $(\text{r-min}(e_{i-1}) + w(e_{i-1}) - 1) + 1 = \text{r-min}(e_{i-1}) + w(e_{i-1})$, is a valid lower bound on the rank of $e^{(1)}$. The rank of x is at most equal to the rank of the first copy of e_i . Therefore, setting $\text{r-max}(x)$ equal to $\text{r-max}(e_i)$ makes it a valid upper bound. After the insertion of x , the ranks of all elements in the summary larger than x increase by $w(x)$ and hence their r-min and r-max values need to be updated. The ranks of elements smaller than x do not change. Also, deleting an element from the summary does not change the bounds on the ranks of other elements in the summary.

The following claim shows that if a certain condition on r-min and r-max values of the elements in WQS is maintained, we can guarantee that WQS will be an ε -approximate summary of $\text{Unfold}(S_w)$.

▷ **Claim 4.** Suppose in WQS over a length n stream, $\text{r-max}(e_i) - (\text{r-min}(e_{i-1}) + w(e_{i-1}) - 1) \leq \lfloor \varepsilon W_n \rfloor$; then WQS is an ε -approximate quantile summary of $\text{Unfold}(S_w)$.

The proof of this claim is presented in the full version.

2.1 Time Steps and Bands

Another important notion is that of *time steps* and *bands*. For simplicity, we define this for the unweighted setting, and then we build upon that for to define them for the weighted setting.

Unweighted Setting. We measure the time as the number of elements appeared in the stream so far in multiples of $\Theta(1/\varepsilon)$. Formally,

► **Definition 5 (Time Steps).** Let $\ell := \frac{1}{\varepsilon}$ which we assume is an integer. We partition the stream into consecutive **chunks** of size ℓ ; the **time step** t then refers to the t -th chunk of elements denoted by $(x_1^{(t)}, \dots, x_\ell^{(t)})$ (we assume that the length of the stream is a multiple of ℓ)¹. We define $t_0(x)$ as the time step in which x appears in the stream.

The next important definitions are *band-values* and *bands* borrowed from [9]. Roughly speaking, we would like to be able to partition elements of the stream into a “small” number of groups (bands) so that elements within a group have “almost the same” time of insertion (as a proxy on how accurate our estimate of their r-min, r-max is). Formally,

► **Definition 6 (Band Values and Bands).** For any element x of the unweighted stream S , we assign an integer called a **band-value**, denoted by $\text{b-value}(x)$, as follows:

- (i) At the time step $t = t_0(x)$, we set $\text{b-value}(x) = 0$;
- (ii) At any time step $t > t_0(x)$, if t is a multiple of $2^{\text{b-value}(x)}$, then we increase $\text{b-value}(x)$ by one.

Weighted Setting. We define an equivalent notion of time steps for weighted streams. We say that $t_k = \lfloor \varepsilon W_k \rfloor$ time steps have elapsed after the arrival of k elements in S_w . Intuitively, a chunk of total weight $\ell := \frac{1}{\varepsilon}$ arrives in the stream in a single time step. For each element x_k in S_w , we define its insertion time step $t_0(x_k) = \lfloor \varepsilon(W_{k-1} + 1) \rfloor$. The band value of an element x of S_w is the band value assigned to the first copy of x in $\text{Unfold}(S_w)$ by Definition 6.

We would formally also have an equivalent definition of bands for the weighted setting that will allow us to compute them in $O(1)$ time. Their equivalence is proved in the full version of the paper.

► **Definition 7 (Band-Values and Bands).** When k elements of S_w have been inserted, for any element x of the stream, $\text{b-value}(x)$ is α if and only if the following inequality is satisfied,

$$2^{\alpha-1} + (t_k \bmod 2^{\alpha-1}) \leq t_k - t_0(x) < 2^\alpha + (t_k \bmod 2^\alpha).$$

For any integer $\alpha \geq 0$, we refer to the set of all elements x with $\text{b-value}(x) = \alpha$ as the **band** α , denoted by Band_α ; we also use $\text{Band}_{\leq \alpha}$ to denote the union of bands 0 to α .

A corollary of Definition 7 is that *number of band-values* after seeing k elements is $B^{(k)} = O(\log t_k) = O(\log(\varepsilon W_k))$. We also note that at any point, the sum of weights of all the elements belonging to bands 0 to α is at most $O(\ell \cdot 2^{\alpha+1})$ because all the copies of all these elements belong to $\text{Band}_{\leq \alpha}$ for $\text{Unfold}(S_w)$. We note these facts below:

$$\# \text{ of b-values } B^{(k)} = O(\log \varepsilon W_k) \quad \text{and} \quad \sum_{x \in \text{Band}_{\leq \alpha}} w(x) \leq O(\ell \cdot 2^{\alpha+1}) \text{ for all } \alpha \geq 0. \quad (2)$$

¹ Both assumptions in this definition are without loss of generality: we can change the value of ε by an $O(1)$ factor to guarantee the first one and add $O(1/\varepsilon)$ dummy elements at the end to guarantee the second one.

We now make the following observation:

► **Observation 8.** *At any point in time, if $\text{b-value}(x) \leq \text{b-value}(y)$ for elements x and y , then at any point after this, $\text{b-value}(x) \leq \text{b-value}(y)$.*

This is simply because, in the unweighted setting, band-values of elements are updated *simultaneously* based on the value of the current time step (Definition 6). Thus, the b-value of the first copies of each element in a band is also updated simultaneously in $\text{Unfold}(S_w)$. Thus, Observation 8 is true.

2.2 Indirect handling of r-min and r-max values

To describe our algorithm, it is better to store the r-min and r-max values indirectly as g and Δ values which we define for the weighted algorithm as follows. For any element e_i in WQS,

$$g_i = \text{r-min}(e_i) - (\text{r-min}(e_{i-1}) + w(e_{i-1}) - 1), \quad \Delta_i = \text{r-max}(e_i) - \text{r-min}(e_i); \quad (3)$$

The g value can be interpreted to be the difference between the minimum possible rank of e_i and the minimum possible rank of the last copy of e_{i-1} . The Δ value is the difference between the r-max and r-min values of the first copy of e_i . The r-min and r-max values can be recovered given the g -values, Δ -values and the weights of all elements in WQS as follows:

$$\text{r-min}(e_i) = g_i + \sum_{j=1}^{i-1} (g_j + w(e_j) - 1), \quad \text{r-max}(e_i) = \Delta_i + g_i + \sum_{j=1}^{i-1} (g_j + w(e_j) - 1).$$

This motivates the definition of the quantity G_i , for each element e_i in WQS:

$$G_i = g_i + w(e_i) - 1. \quad (4)$$

We will soon see that the G -value has a nice property that will prove useful in the analysis of the algorithms that we propose. We now use the g and Δ values defined to state the invariant that we maintain to ensure that WQS is an ε -approximate quantile summary.

► **Invariant 1.** *After seeing k elements of S_w , each element $e_i \in \text{WQS}$ satisfies $g_i + \Delta_i \leq t_k$.*

From Equation (3) we note that $g_i + \Delta_i = \text{r-max}(e_i) - (\text{r-min}(e_{i-1}) + w(e_{i-1}) - 1)$. Also, $t_k = \lfloor \varepsilon W_k \rfloor$ by definition. Therefore, if WQS maintains Invariant 1, Claim 4 implies that it is an ε -approximate quantile summary of S_w .

The following observation now describes how g and Δ values of elements are updated during **Insert** and **Delete** operations (see Section 2).

► **Observation 9.** *In the summary WQS:*

- **Insert($x, w(x)$):** Sets $g(x) = 1$ and $\Delta(x) = g_i + \Delta_i - 1$ and keeps the remaining (g, Δ) values unchanged.
- **Delete(e_i):** Sets g_{i+1} to equal $g_{i+1} + G_i = g_{i+1} + (g_i + w(e_i) - 1)$, and keeps the remaining (g, Δ) values unchanged.

The correctness of Observation 9 follows from Equation (3) and the way in which r-min and r-max values change when these operations are performed. As promised, we present useful properties of G and Δ values.

19:8 Generalizing GK Summaries for Weighted Inputs

G-value. To understand this, we define the notion of *coverage* of any element in WQS. We say that e_i *covers* e_{i-1} whenever e_{i-1} is deleted from the summary, in which case, e_i also covers all elements that e_{i-1} was covering so far (every element only covers itself upon insertion). We define:

- $C(e_i)$: the set of elements covered by e_i . By definition, at any point of time,

$$G_i = \sum_{x \in C(e_i)} w(x) \text{ and } C(e_i) \cap C(e_j) = \emptyset \quad (5)$$

for any e_i, e_j currently stored in WQS.

We claim that G_i equals the sum of weights of the elements in the coverage of e_i . This is easy to verify by induction. When we insert an element, we set its g value to be 1 and the element only covers itself, thus its G value is equal to its weight by Equation (4). In the way G -value is updated upon deletions, according to Observation 9, this continues to be the case throughout the algorithm.

Δ -value. The Δ -value of an element is a measure of the error with which we know its rank. We can use Invariant 1 to deduce the following upper bound on the Δ value of an element x in terms of its insertion time $t_0(x)$.

$$\Delta(x) \leq t_0(x). \quad (6)$$

The intuition here is that after seeing k elements of the stream, the maximum possible difference in possible ranks is bounded by t_k if Invariant 1 is maintained. Hence, the error in the rank of a newly inserted element is also upper bounded by t_k . Formally, suppose that x is the j -th element of the stream and satisfies $e_{i-1} < x < e_i$ at the time of insertion into WQS. When x is inserted into WQS, we set $\Delta(x) = g_i + \Delta_i - 1$. Invariant 1 implies that $\Delta(x) \leq \lfloor \varepsilon W_{j-1} \rfloor - 1 \leq \lfloor \varepsilon (W_{j-1} + 1) \rfloor = t_0(x)$.

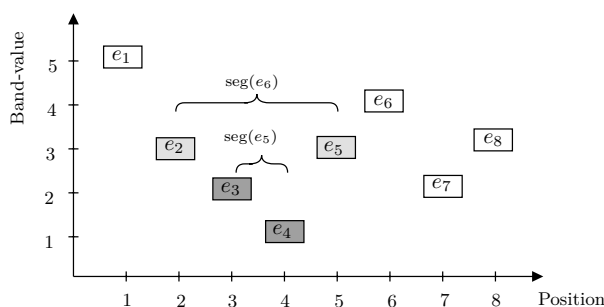
3 A non-trivial extension of GK algorithm for weighted streams

In this section, we present our extension of the GK algorithm for weighted streams. As a warm-up to our main algorithm, we explicitly analyze the special case of the algorithm when all weights are 1 (the unweighted setting) in Section 4.2. Along the way, we also present a very simple and greedy way of maintaining unweighted quantile summaries in $O(\frac{1}{\varepsilon} \cdot \log^2(\varepsilon n))$ space in Section 4.1. There, we shed further light on some counter-intuitive choices in GK summaries which turn out to be a basis for their tighter $O(\frac{1}{\varepsilon} \log(\varepsilon n))$ space. In particular, we motivate the following definition:

► **Definition 10 (Segment).** *The **segment** of an element e_i in WQS, denoted by $\text{seg}(e_i)$, is defined as the maximal set of consecutive elements $e_j, e_{j+1}, \dots, e_{i-1}$ in WQS with b-value strictly less than $\text{b-value}(e_i)$. We let G_i^* be the sum of the G -values of e_i and its segment, i.e., $G_i^* = G_i + \sum_{e_k \in \text{seg}(e_i)} G_k$.*

See Figure 2 below for an illustration.

At any step, the algorithm first inserts the arriving element into WQS; we call this the *insertion step*. It then only deletes an element from WQS if it can be deleted *together* with its entire segment without violating Invariant 1. While there is any such element whose deletion



■ **Figure 2** An illustration of Definition 10. The ranks of elements increase along the horizontal axis. The segment of the element e_5 contains e_3 and e_4 . The segment of e_6 contains e_2, e_3, e_4 and e_5 .

(along with its segment) does not violate Invariant 1 (and another simple but important condition on b-value), the algorithm deletes it from WQS; we call this the *deletion step*. We now give a formal description of the algorithm.

► **Algorithm 1.** A generalization of the GK algorithm for weighted streams:

For each arriving item $(x_j, w(x_j))$:

- (i) Run **Insert** $(x_j, w(x_j))$;
- (ii) While there exists an element e_i in WQS satisfying:

$$(1) \text{b-value}(e_i) \leq \text{b-value}(e_{i+1}) \quad \text{and} \quad (2) G_i^* + g_{i+1} + \Delta_{i+1} \leq t_j$$

Run **Delete** (e_k) for e_k in $\{e_i\} \cup \text{seg}(e_i)$.

► **Theorem 11.** For any $\varepsilon > 0$ and a weighted stream of length n with total weight W_n , Algorithm 1 maintains an ε -approximate quantile summary in $O(\frac{1}{\varepsilon} \cdot \log(\varepsilon W_n))$ space. Also, there is an implementation of Algorithm 1 that takes $O(\log(1/\varepsilon) + \log \log(\varepsilon W_n) + \frac{\log^2(\varepsilon W_n)}{\varepsilon n})$ worst-case update time per element.

We remark here that as long as the weights are $\text{poly}(n)$ bounded and $\varepsilon \geq 1/n^{1-\delta}$ for any fixed $\delta \in (0, 1)$, the space used by the algorithm will be $O((1/\varepsilon) \log(\varepsilon n))$ and its update time will be $O(\log(1/\varepsilon) + \log \log(\varepsilon n))$. This matches the space and time complexities of the implementation of the GK summary described in [13]. Note that the interesting regime for ε is at least a small constant, because when $\varepsilon < 1/n^{1-\delta}$, the information-theoretic lower bound of $(1/2\varepsilon)$ on the summary size already implies that we need to store $\Omega(n^{1-\delta})$ elements even for original GK summaries on unweighted inputs, which is prohibitive for most applications.

Algorithm 1 maintains a valid ε -approximate summary since it may only delete an element e_i along with its segment if the condition (ii): $G_i^* + g_{i+1} + \Delta_{i+1} \leq t_k$ is satisfied. Thus, Invariant 1 is satisfied for the element e_{i+1} after the deletion of e_i (other g and Δ values are unaffected by this). We now focus on bounding the space used by the algorithm in the following. Then, in Section 3.2, we give an efficient implementation to finalize the proof of Theorem 11.

3.1 Space Analysis

In this subsection, we prove a bound on the space used by Algorithm 1. Formally, we have the following:

19:10 Generalizing GK Summaries for Weighted Inputs

► **Lemma 12.** *For any $\varepsilon > 0$ and a stream of length n with the total weight W_n , Algorithm 1 maintains an ε -approximate quantile summary in $O(\frac{1}{\varepsilon} \cdot \log(\varepsilon W_n))$ space.*

We first make a critical observation.

► **Observation 13.** *Elements from $\text{Band}_{\leq \alpha}$ in WQS only cover elements of $\text{Band}_{\leq \alpha}$ at any time.*

This is because when e_i and $\text{seg}(e_i)$ get covered by e_{i+1} , Algorithm 1 ensures that $\text{b-value}(e_i) \leq \text{b-value}(e_{i+1})$. From Definition 10, $\text{seg}(e_i)$ contains elements with b-value less than $\text{b-value}(e_i)$. Thus, $C(e_{i+1})$ contains elements with b-value at most $\text{b-value}(e_{i+1})$ and this continues to be the case at a later time by Observation 8.

Another important observation is that, after executing a deletion step after k insertions, an element e_i present in WQS either satisfies $\text{b-value}(e_i) > \text{b-value}(e_{i+1})$ or $G_i^* + g_{i+1} + \Delta_{i+1} > t_k$; otherwise Algorithm 1 would have deleted this element. We refer to the elements in WQS satisfying the former condition as *type-1* elements and the ones satisfying only the latter condition as *type-2* elements. Thus, each element is exactly one of the two types (except only $e_s = +\infty$ which we can ignore). It will therefore suffice to obtain a bound on the number of type-1 and type-2 elements to bound the space complexity of WQS. Let us first bound the number of type-1 elements in the following lemma.

► **Lemma 14.** *After the deletion step when k elements have been seen, the number of type-1 elements stored in WQS is $O(\ell \cdot \log t_k)$.*

Proof. We first partition the type-1 elements into $B^{(k)}$ sets $Y_0, \dots, Y_{B^{(k)}}$ where for any band-value α :

$$Y_\alpha := \{e_i \in \text{WQS} \mid e_i \text{ is type-1 and } \text{b-value}(e_{i+1}) = \alpha\};$$

(notice that elements in Y_α are such that the band-value of their next element is α , not themselves²) We will show that the size of any set Y_α is at most $O(\ell)$. We map each element of e_i to the smallest element e_j with b-value greater than α ; see Figure 3a for an illustration. Let T_α be the set of all such elements e_j . Also, it is easy to see that the mapping from Y_α to T_α is one to one; giving us $|Y_\alpha| = |T_\alpha|$. Note that e_{j-1} must be a type-2 element. Hence,

$$G_{j-1}^* + g_j + \Delta_j > t_k. \tag{7}$$

Since $\text{b-value}(e_j)$ is greater than $\text{b-value}(e_{i+1}) = \alpha$, by Observation 8, one can argue that e_j is inserted in WQS before e_{i+1} . Let g'_j be the g -value of e_j when e_{i+1} got inserted. By Invariant 1,

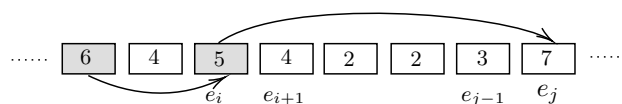
$$g'_j + \Delta_j \leq t_0(e_{i+1}) \quad (\Delta \text{ value does not change over time}). \tag{8}$$

Subtracting Equation (8) from Equation (7) and using the bounds from Definition 7 we conclude that,

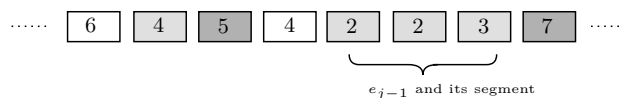
$$G_{j-1}^* + (g_j - g'_j) > t_k - t_0(e_{i+1}) \geq 2^{\alpha-1} - 2. \tag{9}$$

In the above equation:

² While this may sound counter-intuitive at first glance, recall that the criterion for defining the type of an element is a function of both this element and the next one; this definition allows us to take this into account.



(a) The shaded blocks are elements of Y_4 . The arrows indicate the mapping from elements in Y_4 to elements in T_4 . Each element e_i in Y_4 is mapped to the first larger element e_j with a band-value higher than 4.



(b) Each dark gray block represents an element e_j in T_4 . All elements which are either e_{j-1} or are in the $\text{seg}(e_{j-1})$ are shaded light gray.

■ **Figure 3** The two figures represent a section of the summary with each block representing an element. The number inside the block is the element's band-value.

- (i) The term $(g_j - g'_j)$ counts the sum of weights of the elements covered by e_j after e_{i+1} is inserted. Claim 15 will show that these elements are in $\text{Band}_{\leq \alpha}$.
- (ii) The term G_{j-1}^* counts the sum of the weights of the elements covered by e_{j-1} and $\text{seg}(e_{j-1})$. By Definition 10, e_{j-1} and all elements in $\text{seg}(e_{j-1})$ have b-value $\leq \alpha$. Observation 13 allows us to conclude that the sum of weights of elements counted by G_{j-1}^* are in $\text{Band}_{\leq \alpha}$ as well.
- (iii) Additionally, it is easy to see that for distinct e_{j_1} and e_{j_2} in T_α , the segments of e_{j_1-1} and e_{j_2-1} do not overlap (as can be observed in Figure 3). Thus, by Equation (5), the elements covered by e_{j_1} and its segment are distinct from the elements covered by and e_{j_2} and its segment.

Finally, from the above discussion, we conclude that the LHS of Equation (9), summed over all T_α , is proportional to the total weight of all the elements in $\text{Band}_{\leq \alpha}$. Formally,

$$|T_\alpha| \cdot (2^{\alpha-1} - 2) \leq \sum_{e_j \in T_\alpha} G_{j-1}^* + \sum_{e_j \in T_\alpha} (g_j - g'_j) \leq \sum_{x_k \in \text{Band}_{\leq \alpha}} w(x_k) + \sum_{x_k \in \text{Band}_{\leq \alpha}} w(x_k) \leq O(\ell \cdot 2^{\alpha+2}),$$

where the last inequality follows from Equation (2). Hence, $|T_\alpha| = |Y_\alpha| = O(\ell)$ for $\alpha \geq 3$. We have $O(\ell)$ elements for $\alpha = 0, 1, 2$ anyway. Since there are $B^{(k)} = \log t_k$ possible values of α , the number of type-1 elements is $O(\ell \log t_k)$.

▷ **Claim 15.** All the elements covered by e_j after e_{i+1} was inserted have b-value at most α currently.

Proof. Let us assume that there exists an element which currently has b-value $> \alpha$ but gets covered by e_j after e_{i+1} was inserted. All such elements are less than e_j and greater than e_{i+1} since they get covered by e_j . Now consider the smallest such element e . Clearly, e_{i+1} belongs to the segment of e just after the insertion of e_{i+1} . Since e does not belong to the summary right now, it must have been deleted. This implies that its segment, which contained e_{i+1} , got deleted. This means e_{i+1} is also deleted, which is a contradiction. ◁

This finalizes the proof of Lemma 14. ◀

It now remains to bound the number of type-2 elements in WQS which we do in the following lemma.

19:12 Generalizing GK Summaries for Weighted Inputs

► **Lemma 16.** *After the deletion step when k elements of the stream have been seen, the number of type-2 elements is $O(\ell \cdot \log t_k)$.*

Proof. Any type-2 element e_i in WQS , has the property that $G_i^* + g_{i+1} + \Delta_i > t_k$. This will give a lower bound on $G_i^* + g_{i+1}$ in terms of the $\text{b-value}(e_{i+1})$.

▷ **Claim 17.** After seeing k elements, for any type-2 element e_i , $G_i^* + g_{i+1} \geq 2^{\text{b-value}(e_{i+1})-1} - 2$.

Proof. As e_i is a type-2 element, $G_i^* + g_{i+1} + \Delta_{i+1} > t_k$. By Equation (6), $\Delta_{i+1} \leq t_0(e_{i+1})$ and therefore,

$$G_i^* + g_{i+1} > t_k - t_0(e_{i+1}) \geq 2^{\text{b-value}(e_{i+1})-1} - 2,$$

where the second inequality is by Definition 7. ◁

Claim 17 gives us a lower bound on the G^* -value of each type-2 element e_i as a function of the g -value and band-value of the *next* element e_{i+1} . Therefore, we partition the type-2 elements into sets $X_0, \dots, X_{B(k)}$ such that for any band-value α ,

$$X_\alpha := \{e_i \in \text{WQS} \mid e_i \text{ is type-2 and } \text{b-value}(e_{i+1}) = \alpha\}.$$

Moreover, for any $e_i \in X_\alpha$, since e_i is a type-2 element, $\text{b-value}(e_i) \leq \text{b-value}(e_{i+1}) = \alpha$. Summing over the inequality of Claim 17 for each element in X_α , we obtain:

$$|X_\alpha| \cdot (2^{\alpha-1} - 2) \leq \sum_{e_i \in X_\alpha} G_i^* + g_{i+1}. \quad (10)$$

We next show an upper bound on the right-hand side of Equation (10) which will imply the necessary bound on $|X_\alpha|$.

▷ **Claim 18.** After seeing k elements, for any $\alpha \geq 0$, $\sum_{e_i \in X_\alpha} G_i^* \leq 2 \sum_{e_j \in \text{WQS} \cap \text{Band}_{\leq \alpha}} G_j$.

Proof. We partition X_α into two disjoint sets $X_\alpha \cap \text{Band}_\alpha$ and $X_\alpha \cap \text{Band}_{\leq \alpha-1}$ and observe that two elements from one of these two sets must have disjoint segments. Also, the elements in their segments must all be in $\text{Band}_{\leq \alpha}$. Therefore,

$$\begin{aligned} \sum_{e_i \in X_\alpha} G_i^* &= \sum_{e_i \in X_\alpha \cap \text{Band}_\alpha} G_i^* + \sum_{e_i \in X_\alpha \cap \text{Band}_{\leq \alpha-1}} G_i^* \leq \sum_{e_j \in \text{WQS} \cap \text{Band}_{\leq \alpha}} G_j + \sum_{e_j \in \text{WQS} \cap \text{Band}_{\leq \alpha}} G_j \\ &= 2 \sum_{e_j \in \text{WQS} \cap \text{Band}_{\leq \alpha}} G_j. \end{aligned} \quad \triangleleft$$

The next claim bounds the sum of G -values of the elements in WQS from $\text{Band}_{\leq \alpha}$.

▷ **Claim 19.** After seeing k elements, for any $\alpha \geq 0$, $\sum_{e_i \in \text{WQS} \cap \text{Band}_{\leq \alpha}} G_i \leq O(\ell \cdot 2^{\alpha+1})$.

Proof. An element is only deleted by Algorithm 1 if condition (1) is satisfied. By Observation 8, this continues to be the case at any later point in the algorithm. Therefore, $C(e_i)$ only contains elements whose b-value is at most $\text{b-value}(e_i)$. Therefore,

$$\begin{aligned} \sum_{e_i \in \text{WQS} \cap \text{Band}_{\leq \alpha}} G_i &= \sum_{e_i \in \text{WQS} \cap \text{Band}_{\leq \alpha}} \sum_{x_j \in C(e_i)} w(x_j) \\ &\quad \text{(as } G_i = \sum_{x_j \in C(e_i)} w(x_j) \text{ by Equation (5))} \\ &\leq \sum_{x_j \in \text{Band}_{\leq \alpha}} w(x_j) \\ &\quad \text{(as } C(e_i)\text{'s are disjoint and their elements belong to } \text{Band}_{\leq \alpha}\text{)} \\ &= O(\ell \cdot 2^{\alpha+1}), \end{aligned} \quad \text{(by the bound in Equation (2))}$$

completing the argument. ◁

By plugging the bounds of Claim 18 and Claim 19 in Equation (10) and using the fact that a G value of an element is at least its g value, we have that,

$$|X_\alpha| \cdot (2^{\alpha-1} - 2) \leq \sum_{e_i \in X_\alpha} G_i^* + \sum_{e_j \in \text{WQS} \cap \text{Band}_{\leq \alpha}} g_j \leq 2 \sum_{e_j \in \text{WQS} \cap \text{Band}_{\leq \alpha}} G_j + \sum_{e_j \in \text{WQS} \cap \text{Band}_{\leq \alpha}} G_j \leq 3 \cdot O(\ell \cdot 2^{\alpha+1}),$$

which implies $|X_\alpha| = O(\ell)$ for $3 \leq \alpha \leq B^{(k)}$. There can be $O(\ell)$ elements each in X_0 , X_1 and X_2 since there are at most $O(\ell)$ elements in $\text{Band}_{\leq 2}$. By Equation (2) we have $B^{(k)} = O(\log t_k)$ and therefore that the number of type-2 elements is $O(\ell \cdot \log t_k)$. ◀

We have now shown that, after performing the deletion step after k elements have been seen, the number of type-1 elements in WQS is $O(\ell \cdot \log t_k)$ by Lemma 14 and the number of type-2 elements in WQS is $O(\ell \cdot \log t_k)$ by Lemma 16. Since each element in WQS (other than $+\infty$) is either type-1 or type-2, the total number of elements in WQS is $O(\ell \cdot \log t_k)$.

This finalizes the proof of Lemma 12 since $t_n = O(\varepsilon W_n)$ and $\ell = O(\frac{1}{\varepsilon})$. We conclude the discussion of the space complexity with the following remark;

► **Remark 20 (Delaying Deletions).** Suppose in Algorithm 1, instead of running the **deletion step** in Line (ii) after each element, we run it only after inserting c elements $c > 1$; then, the space complexity of the algorithm only increases by an additive term $O(c)$.

Performing the deletion step after k elements, the number of elements reduces to $O(\ell \cdot \log t_k)$ as proved earlier as long as we have been satisfying both the conditions of the deletions of Algorithm 1 while performing every deletion. Thus, the extra space is only due to storing the additional $O(c)$ elements that are inserted in WQS .

The above remark will be useful in proposing an implementation of Algorithm 1 which has an asymptotically faster update time per element, which we do in the following.

3.2 An Efficient Implementation of Algorithm 1

In this section, we present an efficient implementation of Algorithm 1. This is similar to the implementation of the GK summary proposed in [13]. The key idea is that the deletion step is slow and therefore performing it after every time step is rather time inefficient. However, not performing the deletion step for too long blows up the space. The fast implementation we present deals with this trade-off and chooses the delay between consecutive deletion steps so that both the time and space complexity of the algorithm are optimized. Formally, we show the following:

► **Lemma 21.** *There is an implementation of Algorithm 1 that takes $O(\log(1/\varepsilon) + \log \log(\varepsilon W_n) + \frac{\log^2(\varepsilon W_n)}{\varepsilon n})$ worst case processing time per element.*

Part I: Storing WQS

We store our summary WQS as a balanced binary search tree (BST), where each node contains an element of WQS along with its metadata. For each element e we store $w(e)$, $g(e)$, $\Delta(e)$ and $t_0(e)$. The sorting key of the BST is the value of elements. The **Insert** and **Delete** operations insert elements into and delete elements from the BST respectively.

Part II: Performing a Deletion Step

The deletion step involves the deletion of elements in the summary that satisfy the two conditions of Algorithm 1. Checking condition (ii) requires that we know the G^* values corresponding to each element of the summary, which we show how to do efficiently in the following.

19:14 Generalizing GK Summaries for Weighted Inputs

Computing G^* values. First, we perform an inorder traversal of WQS and store the elements e_i in sorted order as a temporary linked list. The G^* value computation will use a stack and will make one pass over the list from the smallest to the largest element. We describe the computation when the traversal reaches the element e_i in the list. To obtain G_i^* , we sum up the G^* values of all elements on the top of the stack with $\text{b-value} < \text{b-value}(e_i)$ and add the sum to G_i . All these elements are popped from the stack and then e_i along with its computed G^* value is pushed onto the stack. We claim that at this point G_i^* has been correctly computed. Since each element is pushed and popped from the stack at most once, the G^* values of all elements can be computed in time linear in the size of WQS .

We now describe how each deletion step is performed.

► **Algorithm.** Performing a deletion step efficiently:

1. Perform an inorder traversal of WQS (which is a BST) to obtain a temporary (doubly-linked) list of elements sorted by value.
2. Compute b-values of all elements of WQS using Definition 7.
3. Compute the G^* value of all elements using the algorithm described above.
4. Traverse the list from larger elements to smaller ones. For each element e_i , delete it from BST (as well as the list), if it satisfies both the deletion conditions mentioned in Algorithm 1.

Having described an insertion step and a deletion step, below is an implementation of Algorithm 1 with fast amortized update time. We also describe how to modify this implementation to also get the same bound on the worst-case update time.

► **Implementation 1.** Efficient Implementation of Algorithm 1.

- Initialize WQS to be an empty balanced binary search tree.
- $\text{DeleteTime} \leftarrow 2$.
- For each arriving item $(x_k, w(x_k))$:
 - (i) Run **Insert** $(x_k, w(x_k))$.
 - (ii) If $(k = \text{DeleteTime})$:
 - Execute the deletion step and update $\text{DeleteTime} \leftarrow \text{DeleteTime} + \lceil \ell \log t_k \rceil$.

Space Analysis

The space complexity of the above implementation is still $O(\frac{1}{\varepsilon} \log(\varepsilon W_n))$. This follows from the fact that, after performing a deletion when k elements have been seen, we wait for another $O(\ell \cdot \log t_k)$ elements only, which increases the space complexity by only a constant factor due to Remark 20. Thus, the space complexity, after n insertions, remains $O(\ell \cdot \log t_n) = O(\frac{1}{\varepsilon} \log(\varepsilon W_n))$, as $\ell = 1/\varepsilon$ and $t_n = O(\varepsilon W_n)$.

Time Analysis

The main purpose behind storing WQS as a BST was to decrease the time required to perform an **Insert** and **Delete** operation on WQS . This takes only $O(\log s)$, where s is the summary size which is at most $O(\frac{1}{\varepsilon} \log \varepsilon W_n)$. Thus, we now have the following observation, which is directly implied by the fact that we perform **Insert** and **Delete** at most once per element.

► **Observation 22.** *Over a stream of length n , the total time taken by the fast implementation of Implementation 1 to perform all **Insert** and **Delete** operations is $O(n \cdot (\log(1/\varepsilon) + \log \log(\varepsilon W_n)))$.*

Note that the only time taken by Implementation 1 **not** taken into account in Observation 22 is the part that determines *which elements* to delete, which we bound in the following.

► **Lemma 23.** *Over a stream of length n , the total time taken by Implementation 1 to decide which elements need to be deleted over all the executed deletion steps is $O(n + \frac{1}{\varepsilon} \log^2(\varepsilon W_n))$.*

Proof. The time taken to decide which elements need to be deleted inside one deletion step (when k elements have been seen) step is $O(s) = O(\ell \cdot \log t_k)$. This is because creating a linked list, followed by computation of b -value and G^* -value of all elements can be performed in $O(s)$ time, as discussed before. Finally, making a linear pass over the list from the largest to the smallest element (to check if the deletion conditions hold) requires $O(s)$ time.

Next, we obtain a bound on the number of deletion steps performed by the algorithm. Consider the deletion steps performed when t_k is in the intervals $[2^i, 2^{i+1})$, for $1 \leq i \leq \lceil \log(\varepsilon W_n) \rceil$. Let $d(i)$ be the number of such deletion steps and $n(i)$ denote the number of elements x_k of the stream for which t_k is in the range $[2^i, 2^{i+1})$. After the deletion step when k elements have been seen, we wait for $\lceil \ell \log t_k \rceil$ insertions. Therefore, there are at least $\ell \cdot i$ elements inserted between two consecutive deletion steps that happen in the considered interval. Therefore, we get the following bound on the number of deletion steps that are performed during the interval.

$$d(i) \leq \frac{n(i)}{\ell \cdot i} + 1. \quad (11)$$

The time spent deciding which element to delete in a deletion step (after seeing k elements) is at most $O(\ell \log t_k) = O(\ell \cdot i)$, when t_k is in the interval $[2^i, 2^{i+1})$. This and Equation (11), give the following bound on the total time spent to decide which elements to delete over all deletions steps.

$$\begin{aligned} O \left(\sum_{i=1}^{\lceil \log(\varepsilon W_n) \rceil} d(i) \cdot \ell i \right) &= O \left(\sum_{i=1}^{\lceil \log(\varepsilon W_n) \rceil} (n(i) + \ell i) \right) \\ &= O \left(n + \frac{1}{\varepsilon} \log^2(\varepsilon W_n) \right) \end{aligned}$$

This finalizes the proof of the lemma. ◀

Observation 22 and Lemma 23 together clearly imply that the total time taken by Implementation 1 over a stream of length n is $O(n \cdot (\log(1/\varepsilon) + \log \log(\varepsilon W_n)) + \frac{1}{\varepsilon} \log^2(\varepsilon W_n))$. Thus, the amortized update time per element is $O(\log(1/\varepsilon) + \log \log(\varepsilon W_n) + \frac{\log^2(\varepsilon W_n)}{\varepsilon n})$.

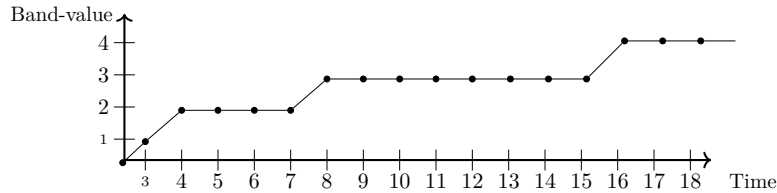
We can obtain the same bound on the worst-case update time per element using standard ideas of distributing time of inefficient operations over multiple time steps. The idea is to process the deletion step over all the following time steps before executing the next deletion step. Formally, we have the following:

▷ **Claim 24.** There is an implementation of Algorithm 1 with worst-case update time $O(\log(1/\varepsilon) + \log \log(\varepsilon W_n) + \frac{\log^2(\varepsilon W_n)}{\varepsilon n})$.

4 Unweighted Quantiles

For a definition of an unweighted quantile summary, see Definition 1. This is a special case of the weighted quantiles problem where each element of the stream arrives with a weight of 1. The GK-algorithm [9] solves this problem optimally [6] by proposing a summary of size $O(\frac{1}{\epsilon} \log(\epsilon n))$. In the following sections, we attempt to simplify the GK algorithm while still being able to prove similar space guarantees.

Towards this end, we describe two algorithms. These algorithms also use the notion of time steps and bands which are formally defined in Section 2.1 (see Definition 5 and Definition 6). The n elements of the stream S are processed in $O(\epsilon n)$ chunks each of size $\ell = O(1/\epsilon)$. We refer to each such chunk of ℓ elements as a time step. (Contrast this with Algorithm 1 in the context of which a time step was defined to be a chunk of $O(1/\epsilon)$ weight). When an element x of the stream arrives in the i -th time step, we set $t_0(x) = i$. Elements are further grouped geometrically based on what time step they appear in into $O(\log t)$ bands, where t is the current time-step (see Figure 4).



(a) Progression of the band-values of elements inserted at time step 2.



(b) Distribution of band values at $t = 15$.

■ **Figure 4** An illustration of band-values and bands.

We use QS to represent the unweighted summary and e_i to be the i -th largest element stored in QS. For each element e_i , we wish to maintain lower and upper bounds on $\text{rank}(e_i)$ denoted by $r\text{-min}(e_i)$ and $r\text{-max}(e_i)$ respectively. This is done implicitly by storing g_i and Δ_i values as described in Equation (3) in Section 2.2. Unlike the weighted setting, we do not need G_i values. (To see why, note that in Equation (4) when the weight of each element is 1, $G_i = g_i$.) Any summary which maintains the following invariant is guaranteed to be an ϵ -approximate quantile summary.

► **Invariant 2.** After t times steps of the stream S , each element $e_i \in \text{QS}$ satisfies $g_i + \Delta_i \leq t$.

Note that this is just a special case of Invariant 1 when all weights are one i.e. $W_n = n$. The goal of our algorithms is to maintain this invariant using limited space.

4.1 A Greedy $O(\frac{1}{\epsilon} \log^2(\epsilon n))$ Size Summary

As a warm-up to our main algorithm, we first present a very simple and greedy way of updating the quantile summary QS to maintain Invariant 2 in $O(\frac{1}{\epsilon} \cdot \log^2(\epsilon n))$ space.

► **Algorithm 2.** A greedy algorithm for updating the quantile summary.

For each time step t with arriving items $(x_1^{(t)}, \dots, x_\ell^{(t)})$:

- (i) Run **Insert** $(x_j^{(t)})$ for each element of the chunk.
- (ii) Repeatedly run **Delete** (e_i) for any (arbitrarily chosen) element e_i in QS satisfying:

$$(1) \text{ b-value}(e_i) \leq \text{b-value}(e_{i+1}) \quad \text{and} \quad (2) g_i + g_{i+1} + \Delta_{i+1} \leq t$$

► **Theorem 25.** For any $\varepsilon > 0$ and a stream of length n , Algorithm 2 maintains an ε -approximate quantile summary in $O(\frac{1}{\varepsilon} \cdot \log^2(\varepsilon n))$ space. Also, there is an implementation of Algorithm 2 that takes $O(\log(1/\varepsilon) + \log \log(\varepsilon n))$ worst-case processing time per element. Finally, quantile queries can be answered in $O(\log(1/\varepsilon) + \log \log(\varepsilon n))$ worst-case time per query.

Algorithm 2 maintains Invariant 2 since it may only delete an element e_i if $g_i + g_{i+1} + \Delta_{i+1} \leq t$, which then implies that $g_{i+1} + \Delta_{i+1} \leq t$ after the deletion. The other (g, Δ) -values remain unchanged. As argued, maintaining Invariant 2 directly implies that QS is an ε -approximate quantile summary throughout the stream. Below we discuss some important insights on showing the space complexity of the algorithm. For formal proofs and its efficient implementation, we refer the reader to the full version of the paper.

After performing the deletion step at time t , we classify the elements into either *type-1* or *type-2*, and bound each one of them separately as we did previously. An element e_i present in QS satisfies $\text{b-value}(e_i) > \text{b-value}(e_{i+1})$ then it is of type-1, or it must satisfy $g_i + g_{i+1} + \Delta_{i+1} > t$ and it is of type-2. We first bound the number of type-2 elements using a similar counting argument as in Section 3.1.

► **Lemma 26.** After the deletion step at time step t , the number of type-2 elements stored in QS is $O(\ell \log t)$.

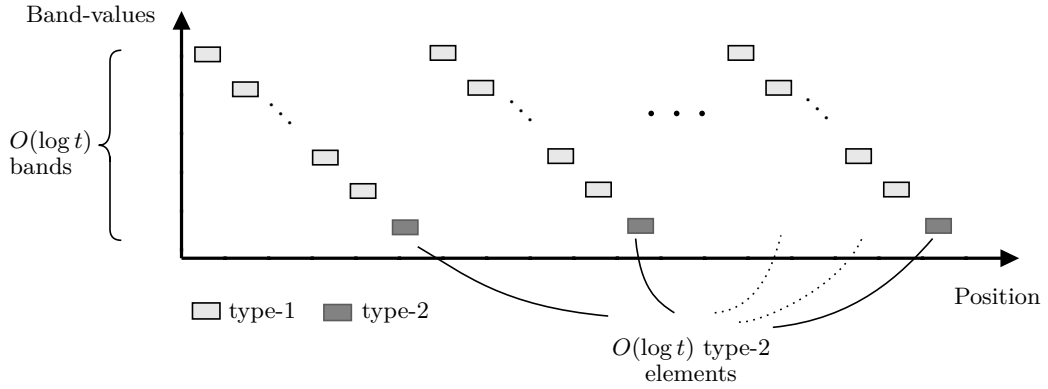
A more interesting is the bound on the number of type-2 elements. Observe that we do not delete an element with its segment (unlike Algorithm 1), which was crucial in proving a bound on type-1 element. However, as we show, the number of type-1 elements cannot be much larger than the type-2 ones even for this deletion strategy. This is simply because the band-values of consecutive type-1 elements strictly decrease from one element to the next and thus we cannot have many type-1 elements next to each other.

► **Lemma 27.** After the deletion step at time step t , the number of type-1 elements stored in QS is $O(\log t)$ times larger than the type-2 elements.

► **Remark 28.** In the space analysis, we bounded the number of type-2 elements in the summary after the deletion step by $O(\ell \cdot \log t) = O((1/\varepsilon) \cdot \log(\varepsilon n))$, which is quite efficient on its own. However, in the worst case, there can be $O(\log t)$ type-1 elements for every type-2 element as shown in Figure 5. Thus, Algorithm 2 may end up storing as many as $O(\ell \cdot \log^2 t) = O((1/\varepsilon) \cdot \log^2(\varepsilon n))$ type-1 elements in the summary, leading to its sub-optimal space requirement.

4.2 The Simplified GK $O(\frac{1}{\varepsilon} \cdot \log(\varepsilon n))$ Size Summary

We give our description of GK summaries. As we say in Remark 28, one source of sub-optimality of Algorithm 2 was a large number of type-1 elements stored in the summary compared to the type-2 ones. A way to improve this is to *actively* try to decrease the number



■ **Figure 5** Each block in the figure represents an element stored in QS. The ranks of elements increase along the horizontal axis. The figure illustrates why Algorithm 2 might end up storing $O(\ell \log^2 t)$ elements in QS. By Lemma 26, there could be as many as $O(\ell \cdot \log t)$ type-2 elements in QS. Each of these type-2 elements could be preceded by a sequence of $O(\log t)$ type-1 elements (since there are $O(\log t)$ bands).

of stored type-1 elements. Roughly speaking, this is done by deleting type-2 elements from the summary only if it does not contribute to creating a long sequence of type-1 elements (e.g., as in Figure 5). Roughly speaking, while there is an element whose deletion *together* with its entire segment (Definition 10) doesn't violate Invariant 2 (and the same condition on b-values), we delete the element and its entire segment. Let g_i^* denote the sum of g -values of the elements in $\text{seg}(e_i)$. Below is a formal description of the algorithm and the theorem, whose proof we include in the full version.

▶ **Algorithm 3.** An improved algorithm for updating the quantile summary.

For each time step t with arriving items $(x_1^{(t)}, \dots, x_\ell^{(t)})$:

- (i) Run **Insert** $(x_j^{(t)})$ for each element of the chunk.
- (ii) While there exists an element e_i in QS satisfying:

$$(1) \text{b-value}(e_i) \leq \text{b-value}(e_{i+1}) \quad \text{and} \quad (2) g_i^* + g_{i+1} + \Delta_{i+1} \leq t$$

Run **Delete** (e_k) for e_k in $\{e_i\} \cup \text{seg}(e_i)$.

▶ **Theorem 29.** For any $\varepsilon > 0$ and a stream of length n , Algorithm 3 maintains an ε -approximate quantile summary in $O(\frac{1}{\varepsilon} \cdot \log(\varepsilon n))$ space. Also, there is an implementation of Algorithm 3 that takes $O(\log(\frac{1}{\varepsilon}) + \log \log(\varepsilon n))$ worst case update time per element.

We shall note that even though our description of Algorithm 3 varies from the presentation of GK summaries in [9], the two algorithms behave in an almost identical way.

References

- 1 Pankaj K. Agarwal, Graham Cormode, Zengfeng Huang, Jeff M. Phillips, Zhewei Wei, and Ke Yi. Mergeable summaries. In *Proceedings of the 31st ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, PODS 2012, Scottsdale, AZ, USA, May 20-24, 2012*, 2012.

- 2 Noga Alon, Omri Ben-Eliezer, Yuval Dagan, Shay Moran, Moni Naor, and Eylon Yogev. Adversarial laws of large numbers and optimal regret in online classification. In *Proceedings of the 53rd Annual ACM SIGACT Symposium on Theory of Computing*, pages 447–455, 2021.
- 3 Noga Alon, Yossi Matias, and Mario Szegedy. The space complexity of approximating the frequency moments. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing, Philadelphia, Pennsylvania, USA, May 22-24, 1996*, pages 20–29, 1996.
- 4 Tianqi Chen and Carlos Guestrin. XGBoost. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, August 2016. doi:10.1145/2939672.2939785.
- 5 Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining*, pages 785–794, 2016.
- 6 Graham Cormode and Pavel Veselý. A tight lower bound for comparison-based quantile summaries. In *Proceedings of the 39th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems, PODS 2020, Portland, OR, USA, June 14-19, 2020*, pages 81–93, 2020.
- 7 Anna C Gilbert, Brett Hemenway, Atri Rudra, Martin J Strauss, and Mary Wootters. Recovering simple signals. In *2012 Information Theory and Applications Workshop*, pages 382–391. IEEE, 2012.
- 8 Anna C Gilbert, Brett Hemenway, Martin J Strauss, David P Woodruff, and Mary Wootters. Reusable low-error compressive sampling schemes through privacy. In *2012 IEEE Statistical Signal Processing Workshop (SSP)*, pages 536–539. IEEE, 2012.
- 9 Michael Greenwald and Sanjeev Khanna. Space-efficient online computation of quantile summaries. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data, Santa Barbara, CA, USA, May 21-24, 2001*, pages 58–66, 2001.
- 10 Moritz Hardt and David P Woodruff. How robust are linear sketches to adaptive inputs? In *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*, pages 121–130, 2013.
- 11 Regant Y. S. Hung and Hing-Fung Ting. An $\Omega(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ space lower bound for finding ϵ -approximate quantiles in a data stream. In *Frontiers in Algorithmics, 4th International Workshop, FAW 2010, Wuhan, China, August 11-13, 2010. Proceedings*, pages 89–100, 2010.
- 12 Zohar S. Karnin, Kevin J. Lang, and Edo Liberty. Optimal quantile approximation in streams. In *IEEE 57th Annual Symposium on Foundations of Computer Science, FOCS 2016, 9-11 October 2016, Hyatt Regency, New Brunswick, New Jersey, USA*, pages 71–78, 2016.
- 13 Ge Luo, Lu Wang, Ke Yi, and Graham Cormode. Quantiles over data streams: experimental comparisons, new analyses, and further improvements. *VLDB J.*, 25(4):449–472, 2016.
- 14 Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Approximate medians and other quantiles in one pass and with limited memory. In *SIGMOD 1998, Proceedings ACM SIGMOD International Conference on Management of Data, June 2-4, 1998, Seattle, Washington, USA*, pages 426–435, 1998.
- 15 Gurmeet Singh Manku, Sridhar Rajagopalan, and Bruce G. Lindsay. Random sampling techniques for space efficient online computation of order statistics of large datasets. In *SIGMOD 1999, Proceedings ACM SIGMOD International Conference on Management of Data, June 1-3, 1999, Philadelphia, Pennsylvania, USA*, pages 251–262, 1999.
- 16 Ilya Mironov, Moni Naor, and Gil Segev. Sketching in adversarial environments. *SIAM Journal on Computing*, 40(6):1845–1870, 2011.
- 17 J. Ian Munro and Mike Paterson. Selection and sorting with limited storage. In *19th Annual Symposium on Foundations of Computer Science, Ann Arbor, Michigan, USA, 16-18 October 1978*, pages 253–258, 1978.
- 18 Moni Naor and Eylon Yogev. Bloom filters in adversarial environments. In *Annual Cryptology Conference*, pages 565–584. Springer, 2015.
- 19 List of open problems in sublinear algorithms – problem 2: Quantiles. <https://sublinear.info/2>.