Generalizing Greenwald-Khanna Streaming Quantile Summaries for Weighted Inputs

March 13, 2023




Sepehr Assadi
Rutgers University

Nirmit Joshi
Northwestern University

Milind Prabhu
University of Michigan

Vihan Shah
Rutgers University

Acknowledgement

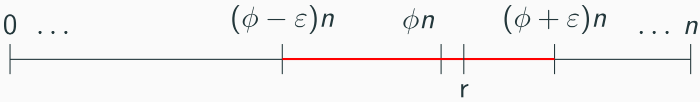
Rajiv Gandhi
Rutgers University–Camden

Introduction

Streaming Quantile Estimation Problem

- **Input:**
  1. $S = \{x_1, \ldots, x_n\}$ of elements from an ordered universe (in the streaming fashion).
  2. Fixed approximation parameter $\varepsilon > 0$.
- **Goal:** At the end of the stream, for any $\phi \in (0, 1]$, we want to estimate ϕ-quantile of S up to an additive error of ε.
- On queried for any $\phi \in (0, 1]$, we want to be able to return $x \in S$ such that

$$(\phi - \varepsilon)n \leq rank(x, S) \leq (\phi + \varepsilon)n.$$



- Rank of an element:

$$rank(x, S) = |\{y \in S \mid y \leq x\}|$$

Weighted Generalized Problem

- **Input**:
  1. A weighted stream $S_w = \{(x_1, w_1), \ldots, (x_n, w_n)\}$.
  2. $w(x)$ is a positive integer.

$$W_n = \sum_{i=1}^{n} w(x_i)$$

  3. Fixed $\varepsilon > 0$

1	1	1	1	3	3	6	6	6	6	6	10	10	10	10

Weighted Generalized Problem

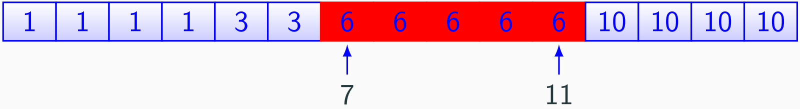
- **Input**:
  1. A weighted stream $S_w = \{(x_1, w_1), \ldots, (x_n, w_n)\}$.
  2. $w(x)$ is a positive integer.

$$W_n = \sum_{i=1}^{n} w(x_i)$$

  3. Fixed $\varepsilon > 0$

1	1	1	1	3	3	6	6	6	6	6	10	10	10	10

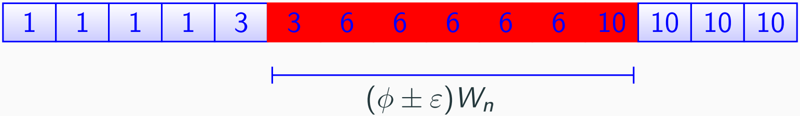
- **Range of rank:**



Weighted Generalization Problem

- **Goal:** At the end, for any $\phi \in (0, 1]$, we want to return an element x_j such that

$$\text{(Range of Ranks of } x_j) \cap \left[(\phi - \varepsilon)W_n, (\phi + \varepsilon)W_n\right] \neq \emptyset$$

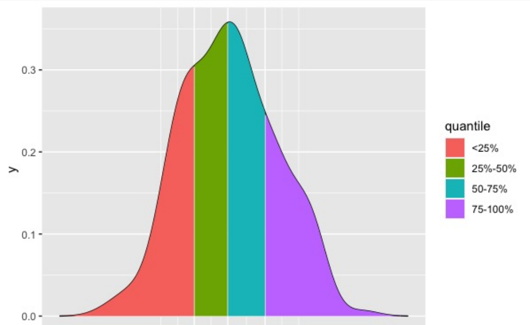


Motivation

A fundamental problem in:

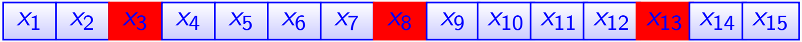
- Data Mining and Data Science
- Machine Learning
- Computer Science

Quantiles provide concise information about the data distribution as they allow us to estimate the CDF of the underlying distribution.

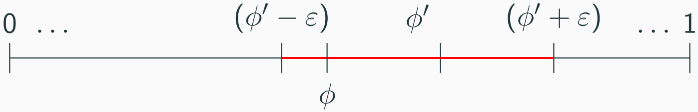


Information Theoretic Lower Bound (Unweighted)

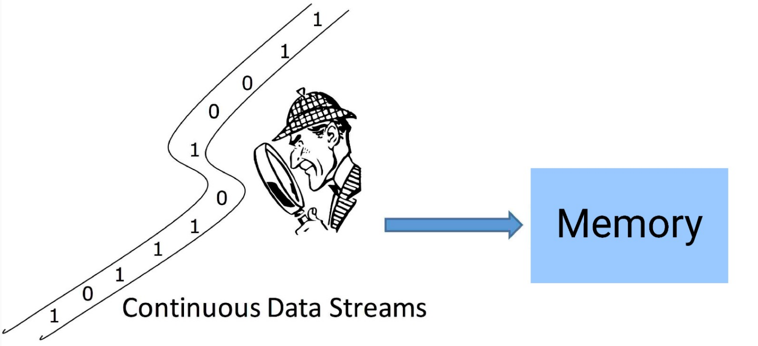
- $S = \{x_1, \ldots, x_n\}$ known apriori
- Which elements to store to approximately answer quantiles queries?
- Store ε-quantile, 3ε-quantile, 5ε-quantile, ... This requires only $O(1/\varepsilon)$ elements.



- This is also necessary!! We must store $\Omega(1/\varepsilon)$ elements.



Streaming Setting



Memory $\lll$ Input Size

Elements $x_1, \ldots, x_n$ come one by one in any arbitrary order.

Related Work

Deterministic Algorithms:

- Manku, Rajagopalan and Lindsay [MRL'98, SIGMOD] the MRL algorithm: uses $O(\frac{1}{\varepsilon}\log^2(\varepsilon n))$ space.
- Greenwald and Khanna [GK'01, SIGMOD] proposed the GK algorithm: uses $O(\frac{1}{\varepsilon}\log(\varepsilon n))$ space.
- The best-known 22-year-old deterministic quantile summary.

Randomized Algorithm:

- [KLL'16, FOCS]: answers with probabilistic guarantee $(1-\delta)$ and achieves $O((\frac{1}{\varepsilon})\log\log(1/\varepsilon\delta))$ space.

Related Work: GK Algorithm

Question 1

Can we improve this space-bound of $O(\frac{1}{\varepsilon}\log(\varepsilon n))$ bound? Or is this optimal?

Answer

Cormode and Vesleý [CV'20, PODS] recently resolved this question by proving $\Omega(\frac{1}{\varepsilon}\log(\varepsilon n))$ lower bound.

Related Work: GK Algorithm

Question 2

Can we simplify the GK Algorithm so that it allows for generalization to related problems, such as the weighted quantile problem?

Answer

This paper!!

Results

Results for the Unweighted Setting

Result 1

A simple and greedy algorithm that admits $O(\frac{1}{\varepsilon}\log^2(\varepsilon n))$ space guarantee.

- Similar to the "GK-Adaptive" [LWYC'16, VLDB] that has no theoretical guarantee.
- Leads to intuitions behind the counter-intuitive choices of the GK algorithm.

Result 2

A new simpler description of the GK algorithm, which requires $O(\frac{1}{\varepsilon}\log(\varepsilon n))$ space.

Result for the Weighted Setting

- **Trivial way:** Feed multiple copies of the same element.
- Update time $O(\max_i w_i)$– prohibitively large.

Result 3

A non-trivial extension of the GK algorithm for weighted inputs that uses

- $O(\frac{1}{\varepsilon}\log(\varepsilon n))$ space.
- $O(\log(1/\varepsilon) + \log\log(\varepsilon n))$ update time per element.

assuming weights are $\mathrm{poly}(n)$ and $\varepsilon \geq 1/n^{1-\delta}$ for any $\delta > 0$

- If $\varepsilon \approx 1/n$, even information-theoretically $\Omega(1/\varepsilon) = \Omega(n)$ elements needed.

This matches the best unweighted case guarantees.

Application

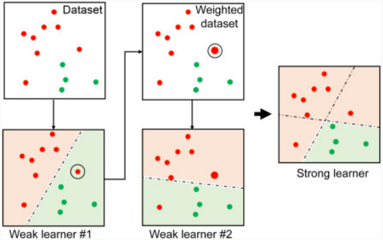
XGBoost

Weak
Learning

XGBoost

Weak Learning → Strong Learning

- Combine weak predictors to boost the confidence and accuracy



- Algorithm: XGBoost library by Nvidia
- Uses the weighted extension of the MRL algorithm: $O(\frac{1}{\varepsilon}\log^2(\varepsilon n))$ space
- Our GK extension can be used here....

Basic Setup: Unweighted to Weighted Extension

Unweighted Quantile Summary (QS)

- QS: A data structure that allows us to answer ε-approximate quantile queries
- It simply stores a subset of elements seen so far.

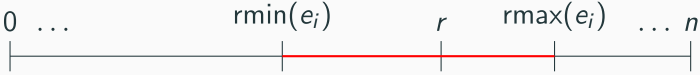
$$\texttt{QS} = \{e_1, \ldots, e_s\}$$

$$e_1 < e_2 < \cdots < e_s$$

- For each element $e \in \texttt{QS}$:

$$\text{rmin}(e_i) = \text{lower bound on the rank of } e_i$$

$$\text{rmax}(e_i) = \text{upper bound on the rank of } e_i$$



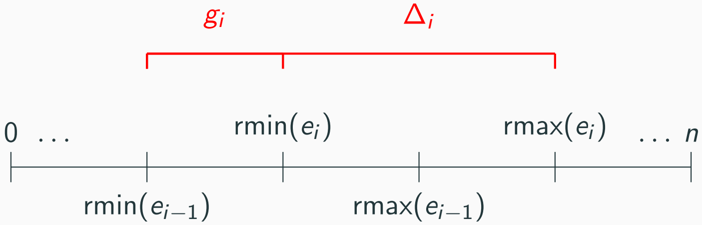
- Space Complexity=$|\texttt{QS}|$ = # of elements stored at any time.

(g, Δ): **Indirectly Handling** (rmin, rmax)

For each element $e_i \in \texttt{QS}$:

$$g_i := \text{rmin}(e_i) - \text{rmin}(e_{i-1})$$

$$\Delta_i = \text{rmax}(e_i) - \text{rmin}(e_i)$$



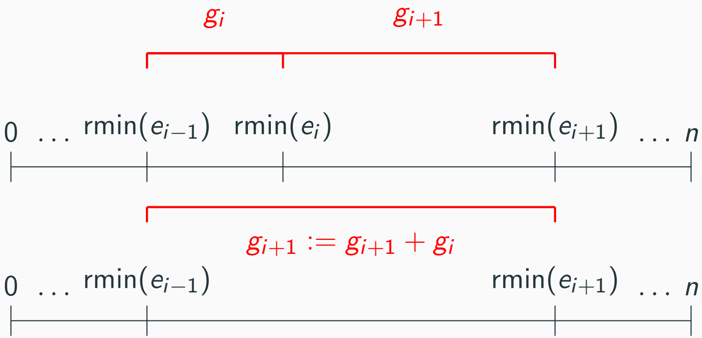
$$\boxed{(\text{rmin}, \text{rmax}) \iff (g, \Delta)}$$

Interpretation

High $(g, \Delta) \iff$ High uncertainty in the Ranks

Insert and Delete

- We can define **Insert(x)** operation
- **Delete(e_i)**: Just forget e_i and keep rmin and rmax values unchanged.



Insert and Delete

Delte(e_i)

1. Delete e_i from `QS`.
2. Keep rmin and rmax values changed
3. Update $g_{i+1} = g_{i+1} + g_i$.

Weighted Quantile Summary WQS

- WQS: A data structure that allows us to answer ε-approximate weighted quantile queries
- It simply stores a subset of elements seen so far.

$$\texttt{WQS} = \{e_1, \ldots, e_s\}$$

$$e_1 < e_2 < \cdots < e_s$$

- Store $w(e_i)$ for each element
- For each element $e \in \texttt{WQS}$:

$$\text{rmin}(e_i) = \text{lower bound on the rank of first copy } e_i$$

$$\text{rmax}(e_i) = \text{upper bound on the rank of first copy } e_i$$

Weighted Quantile Summary `WQS`

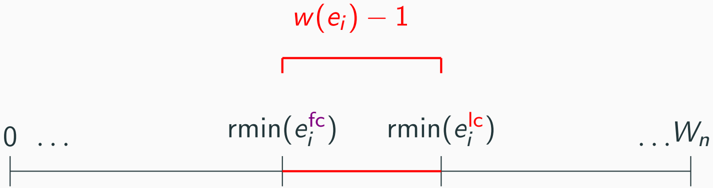
$$\text{rmin}(j\text{-th copy of } e_i) = \text{rmin}(e_i) + j - 1$$

$$\text{rmax}(j\text{-th copy of } e_i) = \text{rmax}(e_i) + j - 1$$

In particular,

$$\text{rmin}(e_i^{\text{lc}}) = \text{rmin}(e_i) + w(e_i) - 1$$

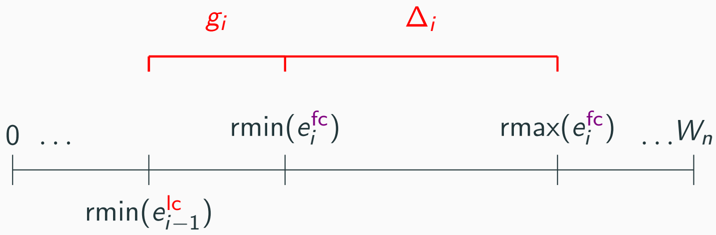
$$\text{rmax}(e_i^{\text{lc}}) = \text{rmax}(e_i) + w(e_i) - 1$$



(g, Δ)

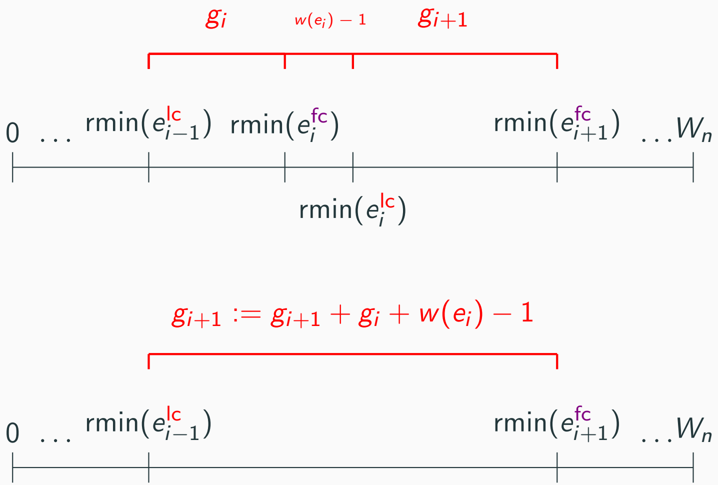
$$\Delta_i := \text{rmax}(e_i) - \text{rmin}(e_i)$$

$$g_i := \text{rmin}(e_i^{\text{fc}}) - \text{rmin}(e_{i-1}^{\text{lc}})$$



Insert-Delete

You can also define **Insert(x)** operation.



G-value and Delete

$$G_i = g_i + w(e_i) - 1$$

Delte(e_i)

1. Delete e_i from `QS`.
2. Update $g_{i+1} = g_{i+1} + g_i + w(e_i) - 1 = g_{i+1} + G_i$.

Note: $G_i = g_i$ in the unweighted case

Approach

Algorithm Sketch

1. **Insertion Step:** Insert all arriving elements x in the chunk using *Insert(x)*.
2. **Deletion Step:** Delete a few elements from `QS`, according to some rule.

The only cleverness of the algorithm is in the deletion step!

Recalling the Goal....

1. To be able to answer the quantile queries

2. Minimize the space

Let's focus on the first...

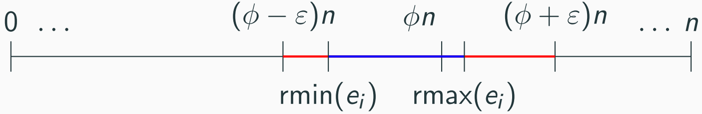
Quantitatively, what $(g, \Delta) \rightarrow$ allows answering quantile queries??

Invariant: Sufficient Condition

Unweighted: After n insertions, for all elements $e_i \in \texttt{QS}$

$$g_i + \Delta_i \leq \varepsilon n$$

then we can answer any ϕ-quantile query with ε-precision.



Weighted:

$$\boxed{g_i + \Delta_i \leq \varepsilon W_n}$$

Obvious Algorithm

Delete elements as long as (g, Δ) invariant holds....

Allows us to answer quantile queries... ✓

Space complexity $\times$

What is the magical GK deletion rule?

Simplified GK for the Unweighted Case

Bands

Bands ≈ Geometric grouping of elements

- Band α contains $\approx 2^{\alpha}$ chunks of $1/\varepsilon$ elements
- After n insertions we have, # of bands $= O(\log(\varepsilon n))$.

Band number:	1		2				3							
Chunk number:	14	13	12	11	10	9	8	7	6	5	4	3	2	1

Segment

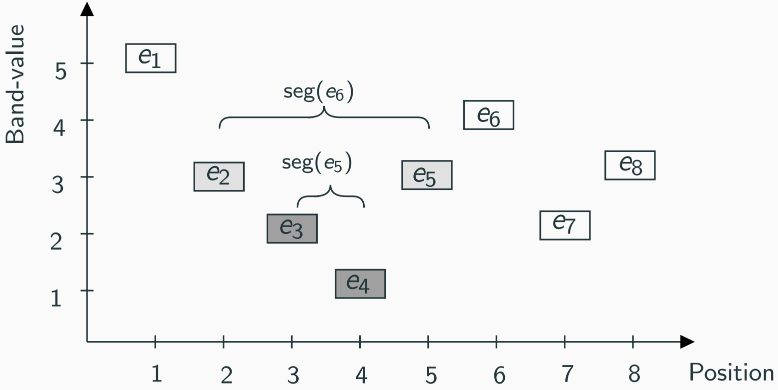
Segment

The **segment** of an element e_i in `QS`, denoted by $\text{seg}(e_i)$, is defined as the maximal set of consecutive elements

$$e_j, e_{j+1}, \cdots, e_{i-1}$$

in `QS` with b-value strictly less than b-value(e_i).

Segment


Band-value
5
4
3
2
1
e_1
seg(e_6)
e_6
e_2
seg(e_5)
e_5
e_8
e_3
e_7
e_4
1
2
3
4
5
6
7
8
Position

Simplified GK Algorithm: $O(\frac{1}{\varepsilon}\log(\varepsilon n))$ space

Treat the element and its segment as one unit!

Simpler GK Algorithm

For arriving item x_k:

1. **Insertion Step:** Insert arriving element x_k using *Insert*(x_k).
2. **Deletion Step:** For any $e_i \in \mathtt{QS}$, delete e_i along with $\text{seg}(e_i)$ if the following two conditions hold:

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

$$g_i^* = g_i + \sum_{j \in \text{seg}(e_i)} g_j$$

Analysis

$(g + \Delta) \leq \varepsilon n$ invariant holds after the deletion!

- Only $O(1/\varepsilon)$ elements per band:

$$\boxed{\text{Total elements} = O(\log(\varepsilon n)/\varepsilon).}$$

Delete element without segment?

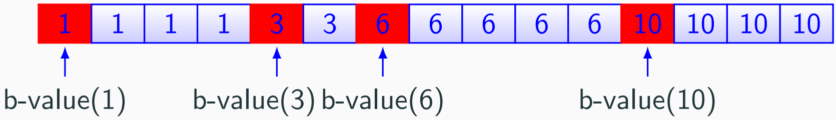
$O(\frac{1}{\varepsilon} \log^2(\varepsilon n))$ space

Greedy!!

Non-trivial extension of the GK for Weighted Inputs

Bands (Weighted)

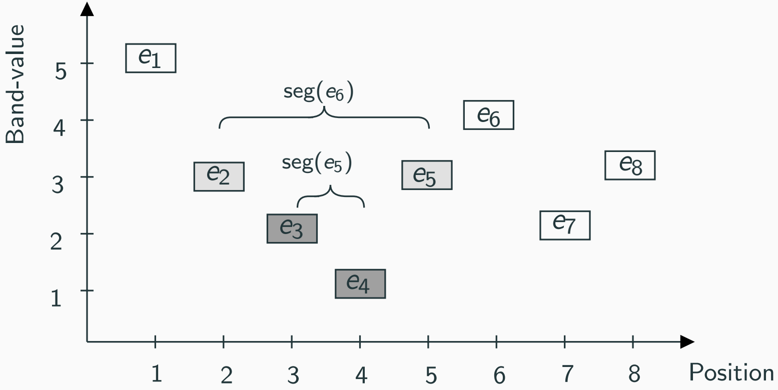
$$\text{b-value}(x) = \text{b-value}(\text{first copy of } x)$$



Different copies may have different b-values in the corresponding unweighted stream because of weights!!

$$\# \text{ bands} = O(\log \varepsilon W_n)$$

Segment (Weighted)



Weighted Extension of GK Algorithm

Treat the element and its segment as one unit!

Weighted extension GK Algorithm

For any arriving item $(x_k, w(x_k))$:

1. **Insertion Step:** Run *Insert*(x_k).
2. **Deletion Step:** For any $e_i \in \texttt{WQS}$, delete e_i along with $\text{seg}(e_i)$ if the following two conditions hold:

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

$$\boxed{G_i^* = G_i + \sum_{j \in \text{seg}(e_i)} G_j}$$

$$(G = g + w - 1)$$

Space Analysis

- W_n = Total weight of n elements
- Using similar counting argument:

$$\text{Space Complexity} = O((1/\varepsilon)\log(\varepsilon W_n)).$$

Under assumption weights are $\mathrm{poly}(n)$:

$$\boxed{\text{Space Complexity} = O\left(\frac{1}{\varepsilon}\log(\varepsilon n)\right).}$$

- This is not just some "smart" implementation of the "trivial" GK extension!!

Difference from Trivial GK

Trivial extension GK Algorithm

For any arriving item $(x_k, w(x_k))$:

1. **Insertion Step:** Insert $w(x_k)$ copies of x_k in `QS`.
2. **Deletion step:** Run unweighted GK deletion rule on `QS`.
3. At the end, collapse multiple copies of the remaining elements into one element.

May delete a partial number of copies of one element into the other and then delete remaining copies later.....

Runtime

Weighted extension GK Algorithm

For any arriving item $(x_k, w(x_k))$:

1. **Insertion Step:** Run *Insert*(x_k).
2. **Deletion Step:** For any $e_i \in \texttt{WQS}$, delete e_i along with seg(e_i) if the following two conditions hold:

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

Runtime: Already Update time doesn't depend on $\max_{i \in [n]} w_i$.

Goal Accomplished!

Faster Runtime

- We can get $O(\log|\texttt{WQS}|) = O(\log(1/\varepsilon) + \log\log(\varepsilon n))$ runtime.
- Store WQS as a balanced Binary Search Tree (BST).
- Insert and Delete takes $O(\log|\texttt{WQS}|)$ time.
- But still, deciding which elements to delete takes time linear in $|\texttt{WQS}|$......
- Perform deletion only after $|\texttt{WQS}|$ doubles by delaying deletions (the space increases only by a constant factor)
- Total time spent over n insertion is

$$O\left(n \cdot (\log(1/\varepsilon) + \log\log(\varepsilon n))\right)$$

$O\left(\log(1/\varepsilon) + \log\log(\varepsilon n)\right)$ amortized update-time

Amortized to Worst-Case Conversion

- Standard Techniques of delaying deletions
- Spread the time required for the deletion step which is linear in $|\texttt{WQS}|$
- Over the next few insertions.
- Interleave between Insertions and Deletions!!
- More details in the full version: arXiv 2303.06288

Main result

A non-trivial extension of the GK algorithm for weighted inputs, under mild assumptions:

- $O(\frac{1}{\varepsilon}\log(\varepsilon n))$ space.
- $O(\log(1/\varepsilon) + \log\log(\varepsilon n))$ update time per element.

Open Question

- Stream S_1 of length n_1 and S_2 of length n_2
- $\texttt{QS}_1 = \mathcal{A}(S_1)$ and $\texttt{QS}_2 = \mathcal{A}(S_2)$ with size $f(n_1)$ and $f(n_2)$.

Mergeable Summaris

An algorithm $\mathcal{A}$ creates mergeable summaries if we can create

$$\texttt{QS} = \mathcal{A}(S_1 \cup S_2)$$

just using $\texttt{QS}_1$ and $\texttt{QS}_2$. We then have $|\texttt{QS}| = f(n_1 + n_2)$.

- The MRL summaries are mergeable: $O((1/\varepsilon)\log^2(\varepsilon n))$ space.
- Is the GK summary also mergeable?

Any algorithm that uses optimal space, i.e.
$f(n) = O(\frac{1}{\varepsilon}\log(\varepsilon n))$ and produce mergeable summaries?

- Important to parallelize the algorithm.

Summary

`QS` : (rmin, rmax, g, Δ)

Insert(x) and Delete(x)

Increases g value

→

`WQS` : (rmin, rmax, g, Δ, G)

Insert(x) and Delete(x)

Increases $G := g + w - 1$ value

Bands and Segment

→

Bands (of the first copies) and Segment

Delete element+segment

$O(\frac{1}{\varepsilon}\log(\varepsilon n))$ space

→

Same strategy $O(\frac{1}{\varepsilon}\log(\varepsilon n))$ space

$O(\log(1/\varepsilon) + \log\log(\varepsilon n))$ update time

GK mergeable? Optimal mergeable summaries?

Thank you!

Questions?