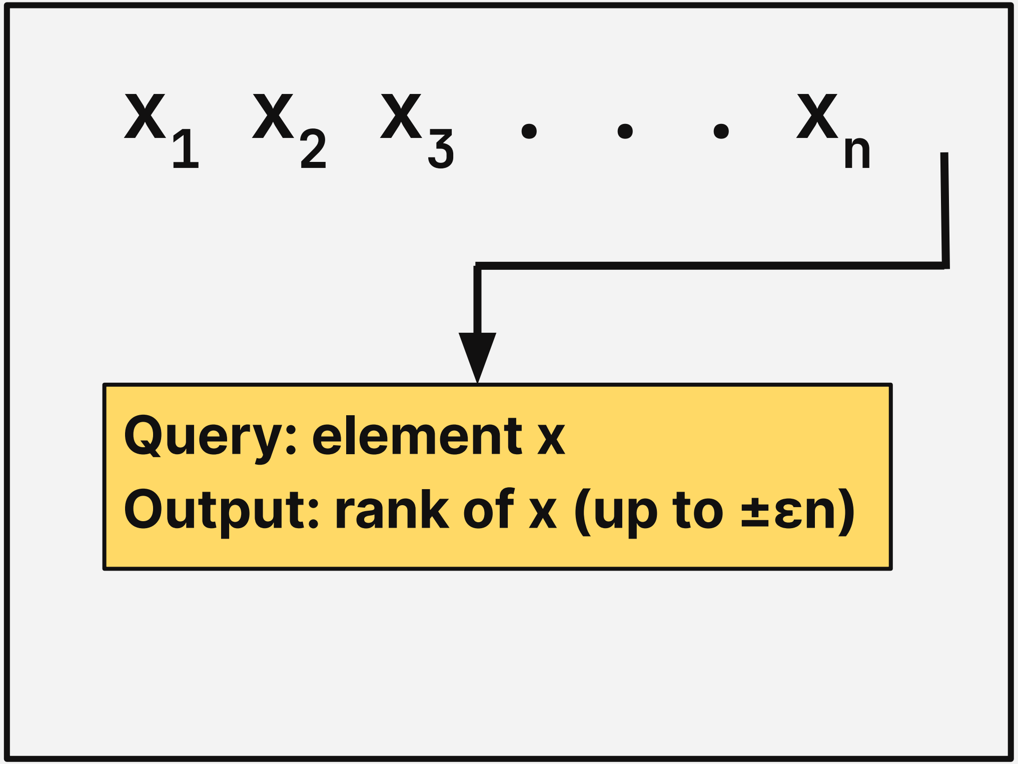
Optimal quantile estimation for streams

Meghal Gupta, **Mihir Singhal**, Hongxun Wu

Quantile estimation problem



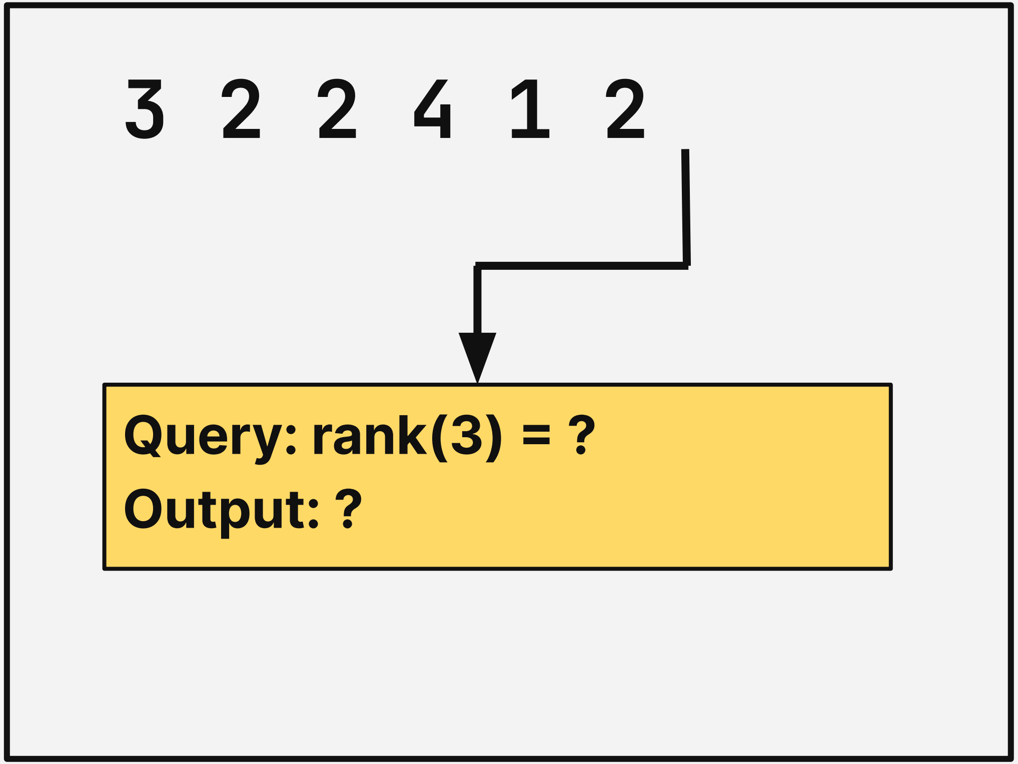
Input: Given parameters n, U, ε, receive a stream x_1, x_2, x_3,..., x_n of numbers in {1,...,U}.

In memory: Store a running sketch of the data that uses as little memory as possible.

Answering queries: Receive an element x of {1,...,U} and output its rank in the stream, that is, the number of elements that were less than x, up to additive error εn.

- Equivalently: receive a rank r as a query, and output the element whose rank is between r – εn and r + εn

Quantile estimation problem



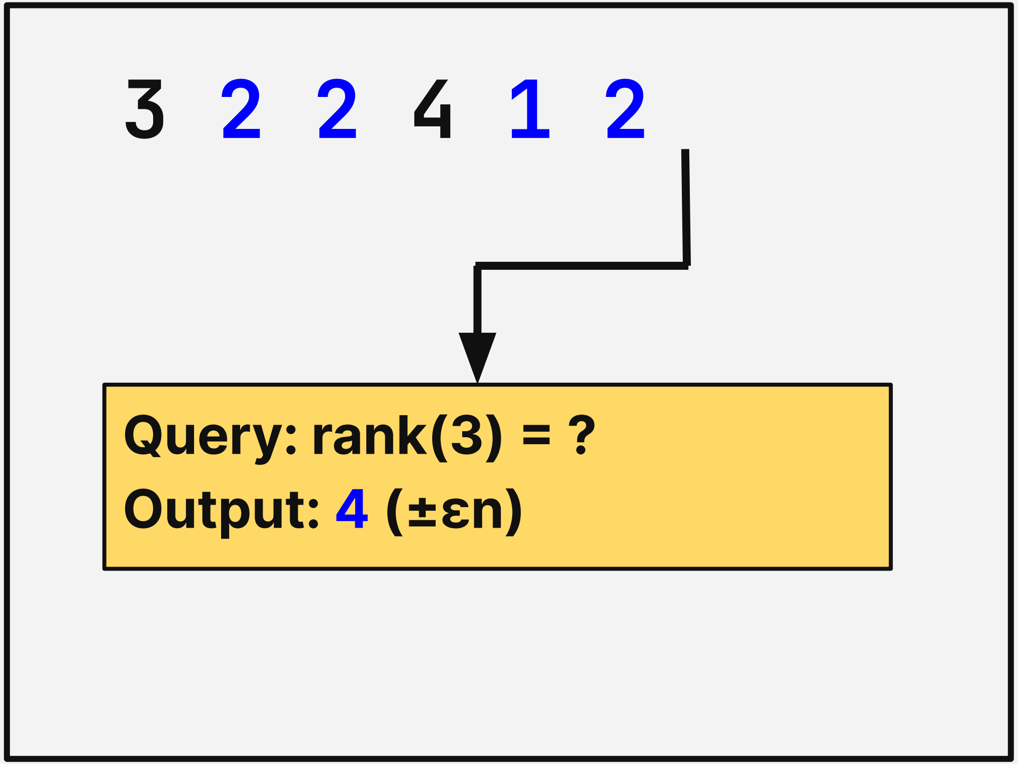
Input: Given parameters n, U, ε, receive a stream $x_1, x_2, x_3, \ldots, x_n$ of numbers in {1,…,U}.

In memory: Store a running sketch of the data that uses as little memory as possible.

Answering queries: Receive an element x of {1,…,U} and output its rank in the stream, that is, the number of elements that were less than x, up to additive error εn.

- Equivalently: receive a rank r as a query, and output the element whose rank is between r – εn and r + εn

Quantile estimation problem



Input: Given parameters n, U, ε, receive a stream $x_1, x_2, x_3, \ldots, x_n$ of numbers in {1,...,U}.

In memory: Store a running sketch of the data that uses as little memory as possible.

Answering queries: Receive an element x of {1,...,U} and output its rank in the stream, that is, the number of elements that were less than x, up to additive error εn.

- Equivalently: receive a rank r as a query, and output the element whose rank is between $r - \varepsilon n$ and $r + \varepsilon n$

Applications of quantile estimation

- Estimating system reliability — for example, estimating the 99th percentile latency of a system.
- User analytics for websites/online content.
  - Median or 90th percentile time for watching a YouTube video.
  - Estimating time a typical user spends on different screens in the app.
  - Companies like Amplitude exist for this very purpose.
- There are many libraries for quantile sketches: Spark-SQL, Apache DataSketches project, GoogleSQL, and XGBoost.

History of quantiles

n = length of stream
U = size of universe
εn = additive error

(1 word = log n + log U bits)

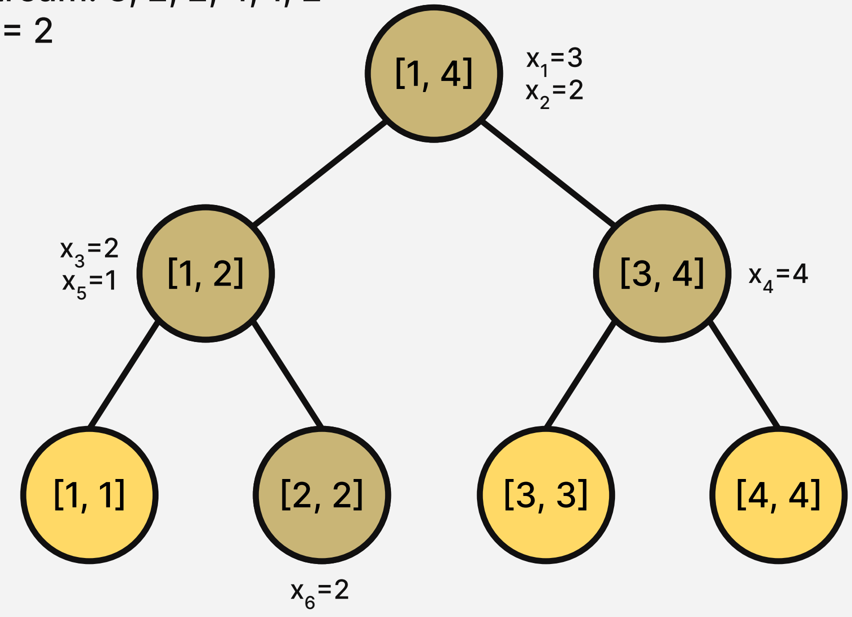
	Space (in words)	Properties
MRL Sketch [1998]	$O(\varepsilon^{-1} \log^2 n)$	comparison-based, deterministic
GK Sketch [2001]	$O(\varepsilon^{-1} \log n)$	comparison-based, deterministic
Q-digest [2004]	$O(\varepsilon^{-1} \log U)$	deterministic
KLL Sketch [2016]	$O(\varepsilon^{-1} \log \log(1/\delta))$	comparison-based, randomized
This talk	$O(\varepsilon^{-1})$	deterministic

(This is optimal)

Q-digest sketch

n = length of stream
U = size of universe
εn = additive error
α = εn / log U
= capacity of each node

Stream: 3, 2, 2, 4, 1, 2
α = 2



Structure: binary tree, where each node is a subinterval of [1, U]. Each node stores some subset of stream elements. Each node has a "capacity" α = εn / log U.

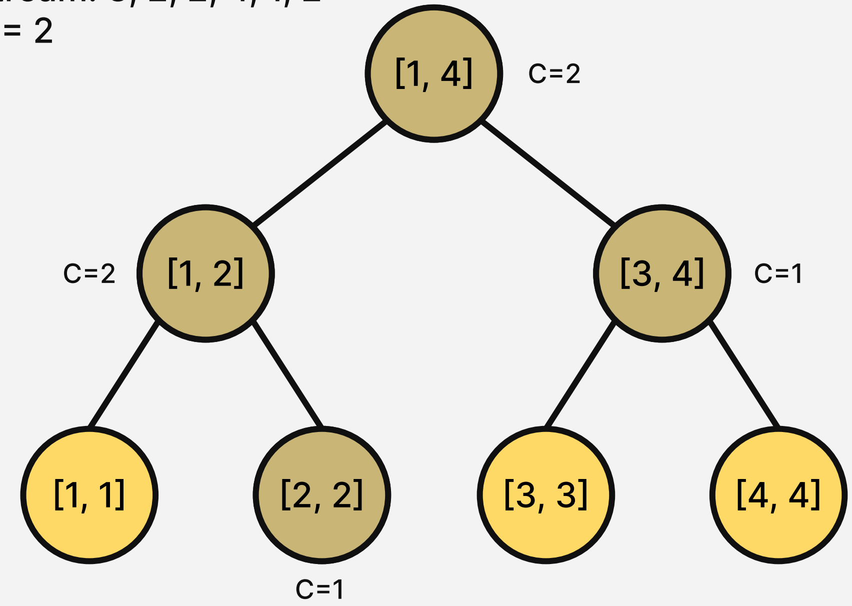
Insertion: insert each element into the tree into the top-most node whose interval contains it, that is not yet at its capacity.

Storage: for each node, only store the number of elements it has.

Q-digest sketch

n = length of stream
U = size of universe
εn = additive error
α = εn / log U
= capacity of each node

Stream: 3, 2, 2, 4, 1, 2
α = 2



Structure: binary tree, where each node is a subinterval of [1, U]. Each node stores some subset of stream elements. Each node has a “capacity” α = εn / log U.

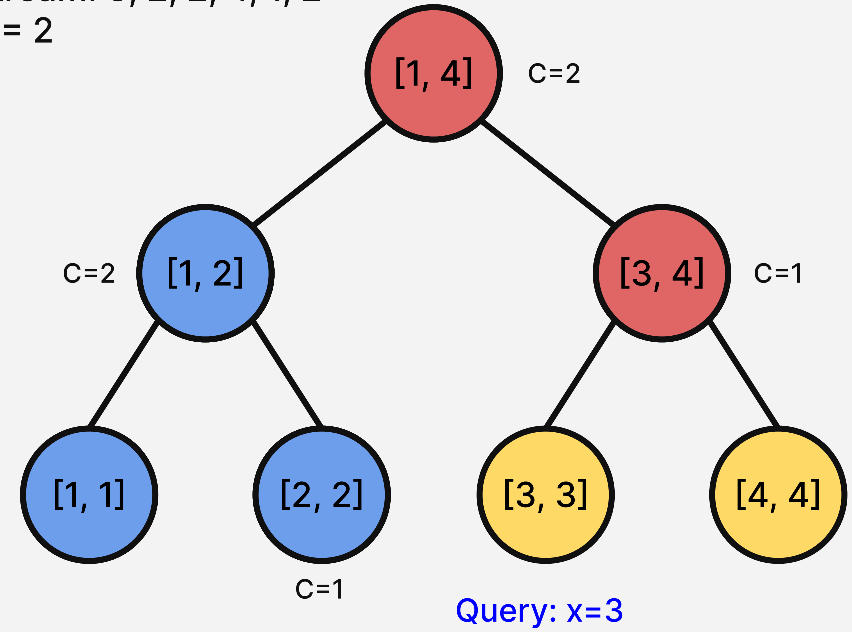
Insertion: insert each element into the tree into the top-most node whose interval contains it, that is not yet at its capacity.

Storage: for each node, only store the number of elements it has.

Q-digest sketch

n = length of stream
U = size of universe
εn = additive error
α = εn / log U
= capacity of each node

Stream: 3, 2, 2, 4, 1, 2
α = 2



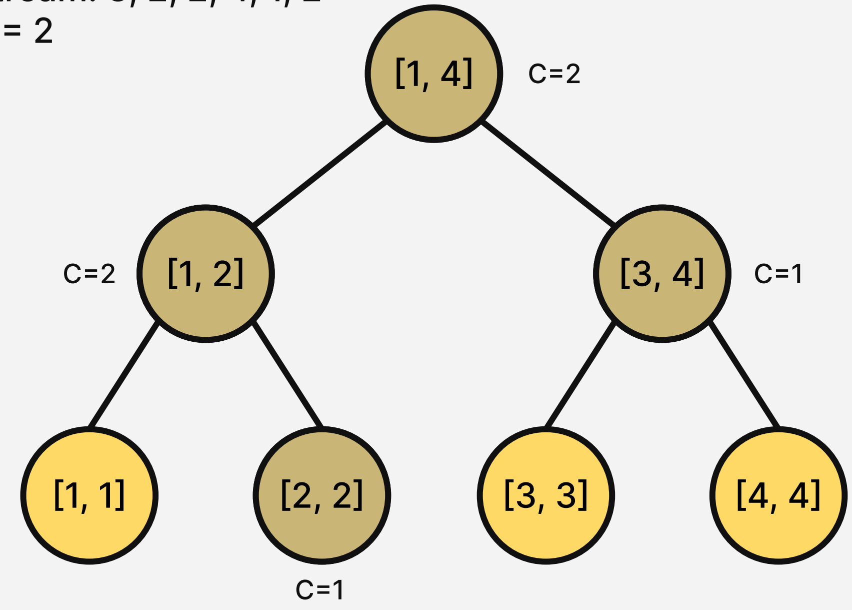
Answering queries: To find the rank of x, add up the counts of all nodes whose intervals contain only elements less than x.

- The only possible missed elements are in ancestors of x
- log U ancestors, so total error in rank is at most α log U = εn

Q-digest sketch

n = length of stream
U = size of universe
εn = additive error
α = εn / log U
= capacity of each node

Stream: 3, 2, 2, 4, 1, 2
α = 2



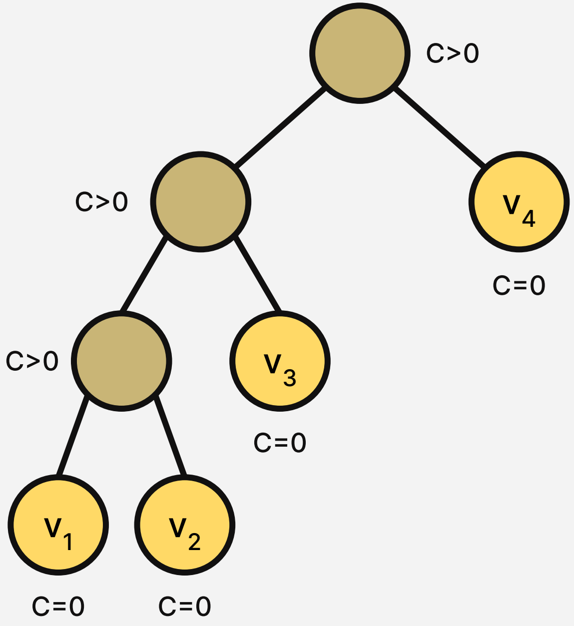
Space complexity: need to store list of all nonempty nodes and their counts.

- Number of nonempty nodes: O(n/α) = $O(\varepsilon^{-1} \log U)$
- List of all nodes: only need to store tree topology, so 1 bit per node
- Counts: log α ≈ O(log n) bits per node
- Total storage: $O(\varepsilon^{-1} \log U \log n)$ bits

Goal of our algorithm: reduce the space needed to store counts

Reducing memory of counts

n = length of stream
U = size of universe
εn = additive error
$\alpha = \varepsilon n / \log U$
= capacity of each node



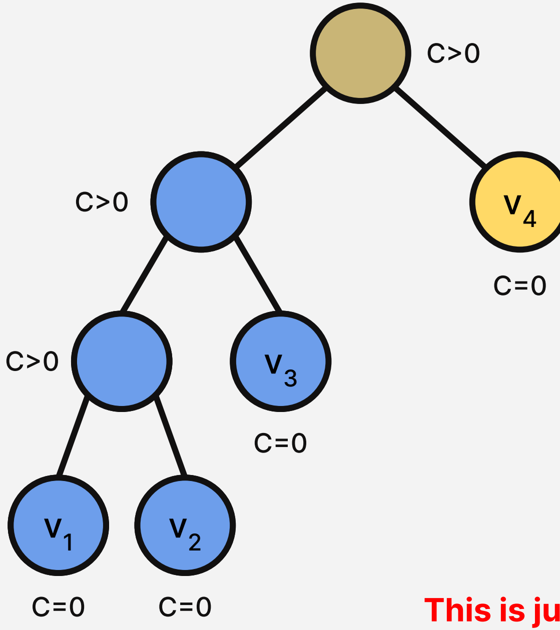
Idea: Only store approximate counts of nodes

How do we insert an element if we only have approximate counts?

- Consider intermediate stage of q-digest tree, and let $v_1, \ldots, v_m$ be children of nonempty nodes. Each insertion will increment the count of some v_i (or its parent).
- Process insertions in **batches** of size α. After each batch, increment counts of v_i.

Reducing memory of counts

n = length of stream
U = size of universe
εn = additive error
$\alpha = \varepsilon n / \log U$
= capacity of each node



Process insertions in **batches** of size α. After each batch, increment counts of v_i.

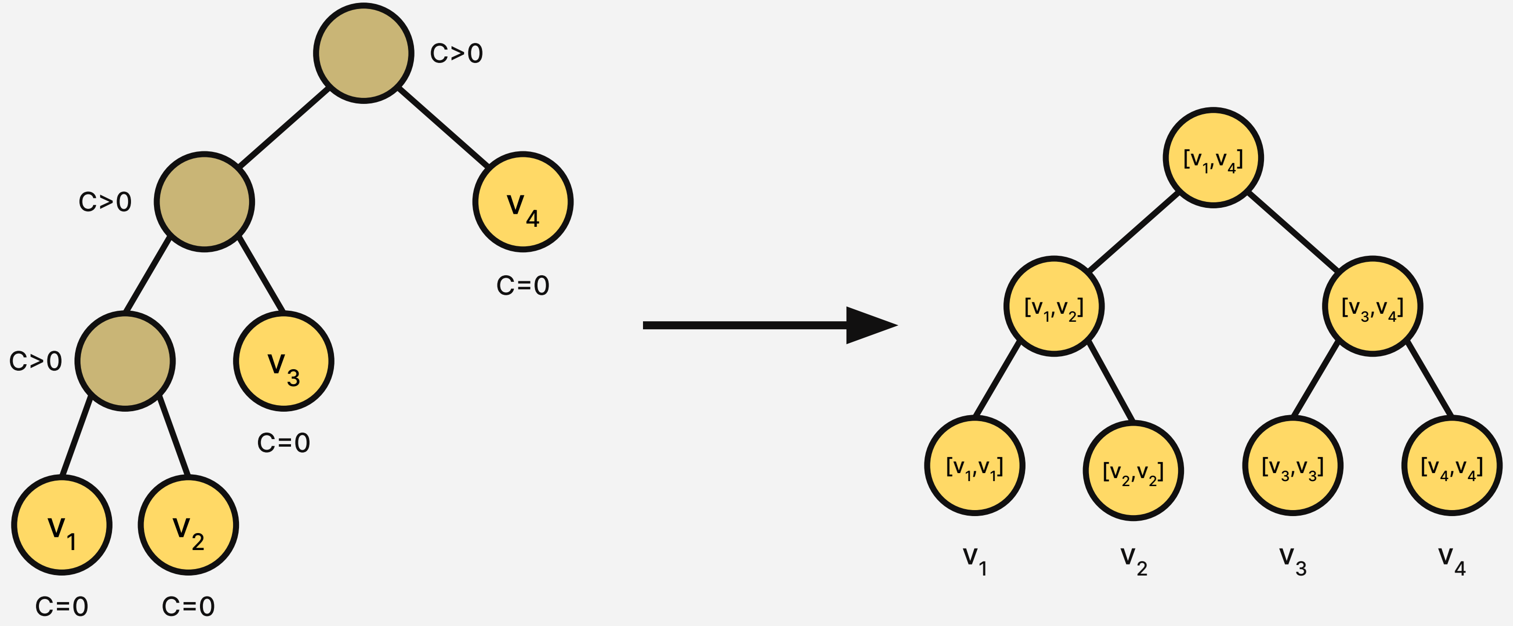
How to process a batch?

- Each stream element will go into v_i for some i.
- Need to obtain approximate counts C_i' which are close to the true counts C_i.
- Rank query adds up C_1 to C_k for some k, so additional error introduced to rank query is
$$C_1' + \ldots + C_k' - (C_1 + \ldots + C_k).$$
- In other words, need an approximation to
$$C_1 + \ldots + C_k$$
for each k (must be accurate within $\varepsilon\alpha$).
- (Can recover $C_1, \ldots, C_m$ from their prefix sums.)

This is just a quantile sketch!

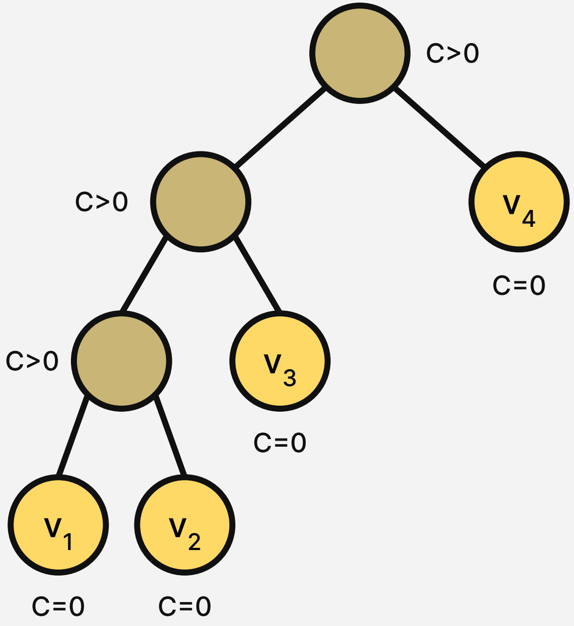
Reducing memory of counts

n = length of stream
U = size of universe
εn = additive error
α = εn / log U
   = capacity of each node



Reducing memory of counts

n = length of stream
U = size of universe
εn = additive error
α = εn / log U
= capacity of each node

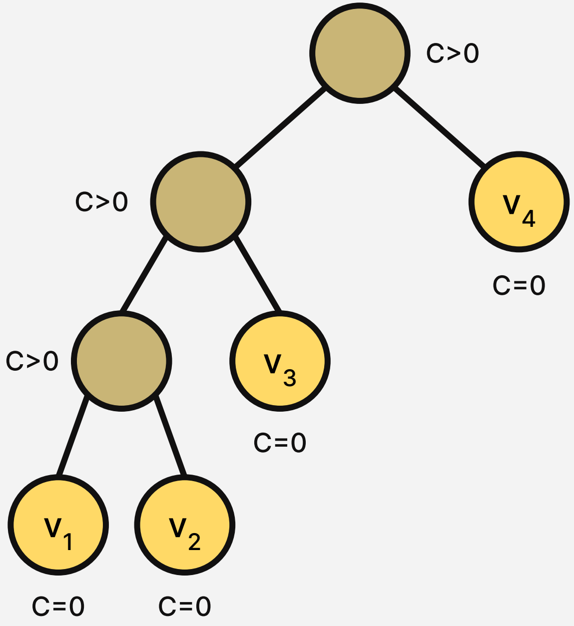


Process insertions in **batches** of size n' = α. After each batch, increment counts of v_i.

- Obtain approximate counts from a smaller q-digest on $v_1, \ldots, v_m$
- Parameters of inner q-digest:
  - $n' = \alpha \le n$
  - $U' \le n / \alpha = O(\varepsilon^{-1} \log U)$
  - $\varepsilon' = \varepsilon$
  - Total space: $O((\varepsilon')^{-1} \log n' \log U') =$ $\tilde{O}(\varepsilon^{-1} \log n \log\log U)$ bits
- Outer q-digest stores counts in multiples of εα up to α, so $O(\log \varepsilon^{-1}) = \tilde{O}(1)$ bits per node
  - Total space: $\tilde{O}(\varepsilon^{-1} \log U)$ bits
- Overall space: $\tilde{O}(\varepsilon^{-1} \log\log U)$ words

Reducing memory of counts

n = length of stream
U = size of universe
εn = additive error
α = εn / log U
= capacity of each node



Process insertions in **batches** of size n' = α. After each batch, increment counts of v_i.

Overall space: $\tilde{O}(\varepsilon^{-1} \log \log U)$ words

Can recursively replace inner q-digest with another instance of our algorithm

- Can do this roughly log* U times, but need to reduce error parameter ε to ε / log* U per recursive layer
- Total space: $\tilde{O}(\varepsilon^{-1} \log^* U)$ words

Getting optimal space

n = length of stream
U = size of universe
εn = additive error
$\alpha = \varepsilon n / \log U$
= capacity of each node

- To get rid of log* U term, give each recursive layer a different error parameter ε: the top and bottom layers get $O(\varepsilon)$ and the rest are slightly smaller
- Losing the polylog(ε^{-1}) term is harder
  - In order to have O(1) space per node instead of O(log ε^{-1}), can only store approximate values of C=0 and C=α
  - Batches will then need to be much larger than α, so will need to keep track of the descendants of v_i as well in case v_i fills up during a batch
  - Finally, also need to change shape of q-digest from one tree of height log(U) to 1/ε trees of height log(εU); will need this for the deeper layers of recursion.

Lower bounds

n = length of stream
U = size of universe
εn = additive error
α = εn / log U
= capacity of each node

More precisely, our algorithm uses $O(\varepsilon^{-1} (\log(\varepsilon U) + \log(\varepsilon n)))$ bits.

Easy lower bound of $\Omega(\varepsilon^{-1} \log(\varepsilon U))$ bits, since if algorithm receives εn copies each of 1/ε elements, it must remember them all.

Theorem [Wang, preprint via private communication]: Deterministic quantile sketches require $\Omega(\varepsilon^{-1} \log(\varepsilon n))$ bits.