CS 473: Algorithms

Chandra Chekuri Ruta Mehta

University of Illinois, Urbana-Champaign

Fall 2016

Streaming Algorithms

Lecture 12

October 5, 2016

Streaming Algorithms

A topic that is both very old, and very current!

Dawn of CS..

Data was stored on tapes, and amount of RAM was very small.

- Too much data, too little space.
- Store only summary or sketch of data.

Streaming Algorithms

A topic that is both very old, and very current!

Dawn of CS..

Data was stored on tapes, and amount of RAM was very small.

- Too much data, too little space.
- Store only summary or sketch of data.

Now..

Terabytes of memory, Gigabytes of RAM.

- Data streams: Humongous amount of data (sometimes never ending)!
- Can go over it at most once, and sometimes not even that!
- Store only summary: sub-linear space-time algorithms.

Examples

An internet router sees a stream of packets, and may want to know,

- which connection is using the most packets
- how many different connections
- median of the file sizes transferred since mid-night
- which connections are using more than 0.1% of the bandwidth.

Computing aggregative information about data streams.

Outline

Computation with data streams.

Heavy-hitters

- Majority element (by R. Boyer and J.S. Moore)
- ϵ-heavy hitters – deterministic
- Approximate counting

Counting using hashing – Count-min Sketch (Cormode-Muthukrishnan'05)

- Variant of Bloom filters.

Data Streams

A stream of data elements, $S = a_1, a_2, \ldots$.

Say a_t arrive at time t. Let us assume that a_t's are numbers for this lecture.

Data Streams

A stream of data elements, $\mathbf{S = a_1, a_2, \ldots}$.

Say $\mathbf{a_t}$ arrive at time $\mathbf{t}$. Let us assume that $\mathbf{a_t}$'s are numbers for this lecture.

Denote $\mathbf{a_{[1..t]} = \langle a_1, a_2, \ldots, a_t \rangle}$.

Given some function we want to compute it continually, while using limited space.

- at any time $\mathbf{t}$ we should be able to query the function value on the stream seen so far, i.e., $\mathbf{a_{[1..t]}}$.

Examples

$$S = 3, 1, 17, 4, -9, 32, 101, 3, -722, 3, 900, 4, 32, \ldots$$

Computing Sum

$$F(a_{[1..t]}) = \sum_{i=1}^{t} a_i$$

Outputs are: 3, 4, 21, 25, 16, 48, 149, 152, -570, ...

Examples

$$S = 3, 1, 17, 4, -9, 32, 101, 3, -722, 3, 900, 4, 32, \ldots$$

Computing Sum

$$F(a_{[1..t]}) = \sum_{i=1}^{t} a_i$$

Outputs are: 3, 4, 21, 25, 16, 48, 149, 152, -570, ...

Keep a counter, and keep adding to it.

After T rounds, the number can be at most $T2^b$. $O(b + \log T)$ space.

Examples

$$S = 3, 1, 17, 4, -9, 32, 101, 3, -722, 3, 900, 4, 32, \ldots$$

Computing max

$$F(a_{[1..t]}) = \max_{i=1}^{t} a_i$$

Outputs are: 3, 3, 17, 17, 17, 32, 101, 101, ...

Just need to store $\mathbf{b}$ bits.

Examples

$$S = 3, 1, 17, 4, -9, 32, 101, 3, -722, 3, 900, 4, 32, \ldots$$

Computing max

$$F(a_{[1..t]}) = \max_{i=1}^{t} a_i$$

Outputs are: 3, 3, 17, 17, 17, 32, 101, 101, ...

Just need to store b bits.

Median?

Examples

$$S = 3, 1, 17, 4, -9, 32, 101, 3, -722, 3, 900, 4, 32, \ldots$$

Computing max

$$F(a_{[1..t]}) = \max_{i=1}^{t} a_i$$

Outputs are: 3, 3, 17, 17, 17, 32, 101, 101, ...

Just need to store b bits.

Median? A lot more tricky

Examples

$$S = 3, 1, 17, 4, -9, 32, 101, 3, -722, 3, 900, 4, 32, \ldots$$

Computing max

$$F(a_{[1..t]}) = \max_{i=1}^{t} a_i$$

Outputs are: 3, 3, 17, 17, 17, 32, 101, 101, ...

Just need to store **b** bits.

Median? A lot more tricky

distinct elements?

Examples

$$S = 3, 1, 17, 4, -9, 32, 101, 3, -722, 3, 900, 4, 32, \ldots$$

Computing max

$$F(a_{[1..t]}) = \max_{i=1}^{t} a_i$$

Outputs are: 3, 3, 17, 17, 17, 32, 101, 101, ...

Just need to store **b** bits.

Median? A lot more tricky

distinct elements? also tricky!

Streaming Algorithms: Framework

$\langle$Initialize summary information$\rangle$

While stream **S** is not done
    x $\leftarrow$ next element in **S**
    $\langle$Do something with **x** and update summary information$\rangle$
    $\langle$Output something if needed$\rangle$

Return $\langle$summary$\rangle$

Streaming Algorithms: Framework

```
⟨Initialize summary information⟩

While stream S is not done
    x ← next element in S
    ⟨Do something with x and update summary information⟩
    ⟨Output something if needed⟩

Return ⟨summary⟩
```

Despite of restrictions, we can compute interesting functions if we can tolerate some error.

Streaming Algorithms: One-sided Error

No false negative

Anything that needs to be considered/counted should be counted.

There may be false positive

We may over count. That is we may consider/count something that shouldn't have been counted.

Part I

Heavy Hitters

Finding the Majority Element

Find the element that occur strictly more than half the time, if any.

Note that at most one such element!

Finding the Majority Element

Find the element that occur strictly more than half the time, if any.

Note that at most one such element!

$$\mathbf{E}, \mathbf{D}, \mathbf{B}, \mathbf{D}, \mathbf{D}_5, \mathbf{D}, \mathbf{B}, \mathbf{B}, \mathbf{B}, \mathbf{B}, \mathbf{B}_{11}, \mathbf{E}, \mathbf{E}, \mathbf{E}, \mathbf{E}, \mathbf{E}_{16}$$

- At time **5**, it is **D**.
- At time **11**, it is **B**
- At time **16**, none!

Finding the Majority Element

Find the element that accrue strictly more than half the time, if any.

R. Boyer and J. S. Moore Algorithm

Initialize: mem=$\emptyset$ and counter=0

Finding the Majority Element

Find the element that accrue strictly more than half the time, if any.

R. Boyer and J. S. Moore Algorithm

Initialize: mem=$\emptyset$ and counter=0

When element $\mathbf{a_t}$ arrives
    if (counter == 0)
        set mem=$\mathbf{a_t}$ and counter=1

Finding the Majority Element

Find the element that accrue strictly more than half the time, if any.

R. Boyer and J. S. Moore Algorithm

Initialize: mem=$\emptyset$ and counter=0

When element $\mathbf{a_t}$ arrives
if (counter == 0)
set mem=$\mathbf{a_t}$ and counter=1
else if ($\mathbf{a_t}$ == mem) then counter++

Finding the Majority Element

Find the element that accrue strictly more than half the time, if any.

R. Boyer and J. S. Moore Algorithm

Initialize: mem=$\emptyset$ and counter=0

When element $\mathbf{a_t}$ arrives
    if (counter == 0)
        set mem=$\mathbf{a_t}$ and counter=1
    else if ($\mathbf{a_t}$ == mem) then counter++
    else counter−− (discard $\mathbf{a_t}$ and a copy of mem)
Return mem.

Finding the Majority Element

Find the element that accrue strictly more than half the time, if any.

R. Boyer and J. S. Moore Algorithm

Initialize: mem=$\emptyset$ and counter=0

When element $\mathbf{a_t}$ arrives
    if (counter == 0)
      set mem=$\mathbf{a_t}$ and counter=1
    else if ($\mathbf{a_t}$ == mem) then counter++
    else counter−− (discard $\mathbf{a_t}$ and a copy of mem)
Return mem.

Even if no majority element, something is returned – False positive.

Finding the Majority Element: Example

R. Boyer and J. S. Moore Algorithm

Initialize: mem=$\emptyset$ and counter=0

Finding the Majority Element: Example

R. Boyer and J. S. Moore Algorithm

Initialize: mem=$\emptyset$ and counter=0

When element $\mathbf{a_t}$ arrives
    if (counter == 0)
        set mem=$\mathbf{a_t}$ and counter=1

Finding the Majority Element: Example

R. Boyer and J. S. Moore Algorithm

Initialize: mem$=\emptyset$ and counter$=0$

When element $\mathbf{a_t}$ arrives

if (counter $==$ 0)

set mem$=\mathbf{a_t}$ and counter$=1$

else if ($\mathbf{a_t}$ $==$ mem) then counter$++$

Finding the Majority Element: Example

R. Boyer and J. S. Moore Algorithm

Initialize: mem=$\emptyset$ and counter=0

When element $\mathbf{a_t}$ arrives
    if (counter == 0)
      set mem=$\mathbf{a_t}$ and counter=1
    else if ($\mathbf{a_t}$ == mem) then counter++
    else counter−− (discard $\mathbf{a_t}$ and a copy of mem)
Return mem.

E, D, B, D, $\mathbf{D_5}$, D, B, B, B, B, $\mathbf{B_{11}}$, E, E, E, E, $\mathbf{E_{16}}$

$\mathbf{a_t}$	E	D	B	D	D	D	B	B	B	B	B	...
mem	E	E	B	B	D	D	D	D	B	B	B	...
counter	1	0	1	0	1	2	1	0	1	2	3	...

Finding a Majority Element

Correctness, if majority element

Lemma

If there is a majority element, the algorithm will output it.

Proof.

- Decreasing counter is like throwing away a copy of element in mem.

Finding a Majority Element

Correctness, if majority element

Lemma

If there is a majority element, the algorithm will output it.

Proof.

- Decreasing counter is like throwing away a copy of element in mem.
- We do this every time a_t is different than mem, and there are less than half such a_t.
- Even if we are throwing away the majority element every time, since they are more than half all cannot be thrown away.

□

Finding a Majority Element

Correctness, if majority element

Lemma

If there is a majority element, the algorithm will output it.

Proof.

- Decreasing counter is like throwing away a copy of element in mem.
- We do this every time $\mathbf{a_t}$ is different than mem, and there are less than half such $\mathbf{a_t}$.
- Even if we are throwing away the majority element every time, since they are more than half all cannot be thrown away.

□

In fact at any time $\mathbf{t}$, mem contains majority element of sub-stream $\mathbf{a_{[1..t]}}$, if any.

Finding a Majority Element

Correctness, if majority element

Gang war interpretation!

Every element is a gang member. When we have two members from different gangs, they shoot each other.

Finding a Majority Element

Correctness, if majority element

Gang war interpretation!

Every element is a gang member. When we have two members from different gangs, they shoot each other.

If there is a gang with more than $\mathbf{n/2}$ members, that will be the only one whose members will survive!

ϵ-Heavy Hitters

Definition

Given a stream $\mathbf{S} = \mathbf{a_1}, \mathbf{a_2}, \ldots$, define count of element $\mathbf{e}$ at any time $\mathbf{t}$ to be

$$\text{count}_{\mathbf{t}}(\mathbf{e}) = |\{\mathbf{i} \leq \mathbf{t} \mid \mathbf{a_i} = \mathbf{e}\}|$$

It is called ϵ-heavy hitter at time $\mathbf{t}$ if $\text{count}_{\mathbf{t}}(\mathbf{e}) > \epsilon \mathbf{t}$.

ϵ-Heavy Hitters

Definition

Given a stream $S = a_1, a_2, \ldots$, define count of element e at any time t to be

$$\text{count}_t(e) = |\{i \leq t \mid a_i = e\}|$$

It is called ϵ-heavy hitter at time t if $\text{count}_t(e) > \epsilon t$.

Goal:

Maintain a structure containing all the ϵ-heavy hitters so far.
At any point there are at most $1/\epsilon$ such elements.

ϵ-Heavy Hitters

Definition

Given a stream $\mathbf{S} = \mathbf{a_1, a_2, \ldots}$, define count of element $\mathbf{e}$ at any time $\mathbf{t}$ to be

$$\mathsf{count_t}(\mathbf{e}) = |\{\mathbf{i} \leq \mathbf{t} \mid \mathbf{a_i} = \mathbf{e}\}|$$

It is called ϵ-heavy hitter at time $\mathbf{t}$ if $\mathsf{count_t}(\mathbf{e}) > \epsilon\mathbf{t}$.

Goal:

Maintain a structure containing all the ϵ-heavy hitters so far.
At any point there are at most $1/\epsilon$ such elements.

Crucial Note: false positive are OK, but no false negative

We are NOT allowed to miss any heavy-hitters, but we could store non-heavy-hitters.

ϵ-Heavy Hitters: Example

If $\epsilon = 1/2$ then the majority element!

ϵ-Heavy Hitters: Example

If $\epsilon = 1/2$ then the majority element!

$$\mathbf{E, D, B, D, D_5, D, B, B, B, B, B_{11}, E, E, E, E, E_{16}}$$

$\mathbf{1/3}$-heavy hitters

- At time **5**, it is **D**.
- At time **11**, both **B** and **D**.
- At time **15**, none!
- At time **16**, it is **E**.

As time passes, the set of heavy hitters may change completely.

ϵ-Heavy Hitters: Algorithm

If $\epsilon = 1/2$ then the majority element!

Set $k = \lceil 1/\epsilon \rceil - 1$. (if $\epsilon = 1/2$ then $k = 1$)

Algorithm

Keep an array $T[1, \ldots, k]$ to hold elements
Keep an array $C[1, \ldots, k]$ to hold their counters

Initialize: $C[j] = 0$ and $T[j] = \emptyset$ for all i.

ϵ-Heavy Hitters: Algorithm

If $\epsilon = 1/2$ then the majority element!

Set $\mathbf{k} = \lceil 1/\epsilon \rceil - 1$. (if $\epsilon = 1/2$ then $\mathbf{k} = 1$)

Algorithm

Keep an array $\mathbf{T[1, \ldots, k]}$ to hold elements
Keep an array $\mathbf{C[1, \ldots, k]}$ to hold their counters

Initialize: $\mathbf{C[j] = 0}$ and $\mathbf{T[j]} = \emptyset$ for all i.

When element $\mathbf{a_t}$ arrives,
    If ($\mathbf{a_t} == \mathbf{T[j]}$ for some $\mathbf{j} \leq \mathbf{k}$), then $\mathbf{C[j]} + +$.

ϵ-Heavy Hitters: Algorithm

If $\epsilon = 1/2$ then the majority element!

Set $\mathbf{k} = \lceil 1/\epsilon \rceil - 1$. (if $\epsilon = 1/2$ then $\mathbf{k} = 1$)

Algorithm

Keep an array $\mathbf{T[1, \ldots, k]}$ to hold elements
Keep an array $\mathbf{C[1, \ldots, k]}$ to hold their counters

Initialize: $\mathbf{C[j] = 0}$ and $\mathbf{T[j]} = \emptyset$ for all i.

When element $\mathbf{a_t}$ arrives,
    If ($\mathbf{a_t} == \mathbf{T[j]}$ for some $\mathbf{j} \leq \mathbf{k}$), then $\mathbf{C[j]}++$.
    Else if ($\mathbf{C[j]} == \mathbf{0}$ for some $\mathbf{j} \leq \mathbf{k}$), then

ϵ-Heavy Hitters: Algorithm

If $\epsilon = 1/2$ then the majority element!

Set $\mathbf{k} = \lceil 1/\epsilon \rceil - 1$. (if $\epsilon = 1/2$ then $\mathbf{k} = 1$)

Algorithm

Keep an array $\mathbf{T[1, \ldots, k]}$ to hold elements
Keep an array $\mathbf{C[1, \ldots, k]}$ to hold their counters

Initialize: $\mathbf{C[j] = 0}$ and $\mathbf{T[j]} = \emptyset$ for all i.

When element $\mathbf{a_t}$ arrives,

If ($\mathbf{a_t} == \mathbf{T[j]}$ for some $\mathbf{j} \leq \mathbf{k}$), then $\mathbf{C[j]}++$.

Else if ($\mathbf{C[j]} == \mathbf{0}$ for some $\mathbf{j} \leq \mathbf{k}$), then

Set $\mathbf{T[j]} \leftarrow \mathbf{a_t}$ and $\mathbf{C[j]} \leftarrow \mathbf{1}$.

ϵ-Heavy Hitters: Algorithm

If $\epsilon = 1/2$ then the majority element!

Set $\mathbf{k} = \lceil 1/\epsilon \rceil - 1$. (if $\epsilon = 1/2$ then $\mathbf{k} = 1$)

Algorithm

Keep an array $\mathbf{T[1, \ldots, k]}$ to hold elements

Keep an array $\mathbf{C[1, \ldots, k]}$ to hold their counters

Initialize: $\mathbf{C[j] = 0}$ and $\mathbf{T[j]} = \emptyset$ for all i.

When element $\mathbf{a_t}$ arrives,

- If ($\mathbf{a_t} == \mathbf{T[j]}$ for some $\mathbf{j} \leq \mathbf{k}$), then $\mathbf{C[j]}++$.
- Else if ($\mathbf{C[j]} == \mathbf{0}$ for some $\mathbf{j} \leq \mathbf{k}$), then
  - Set $\mathbf{T[j]} \leftarrow \mathbf{a_t}$ and $\mathbf{C[j]} \leftarrow \mathbf{1}$.
- Else do $\mathbf{C[j]}--$ for all $\mathbf{j}$. (discard $\mathbf{a_t}$ and a copy of all $\mathbf{T[j]}$)

ϵ-Heavy Hitters: Algorithm

If $\epsilon = 1/2$ then the majority element!

Set $k = \lceil 1/\epsilon \rceil - 1$. (if $\epsilon = 1/2$ then $k = 1$)

Algorithm

Keep an array $T[1, \ldots, k]$ to hold elements
Keep an array $C[1, \ldots, k]$ to hold their counters

Initialize: $C[j] = 0$ and $T[j] = \emptyset$ for all i.

When element a_t arrives,
  If ($a_t == T[j]$ for some $j \leq k$), then $C[j]++$.
  Else if ($C[j] == 0$ for some $j \leq k$), then
    Set $T[j] \leftarrow a_t$ and $C[j] \leftarrow 1$.
  Else do $C[j] --$ for all j. (discard a_t and a copy of all $T[j]$)

Same as the *Majority algorithm* for $\epsilon = 1/2$.

At any time $\mathbf{t}$, our estimates are:

$$
\begin{aligned}
\text{est}_{\mathbf{t}}(\mathbf{e}) &= \mathbf{C[j]} && \text{if } \mathbf{e} = \mathbf{T[j]} \\
&= \mathbf{0} && \text{otherwise}
\end{aligned}
$$

Lemma

Estimates satisfy: $\mathit{est}_{\mathbf{t}}(\mathbf{e}) \leq \mathit{count}_{\mathbf{t}}(\mathbf{e}) \leq \mathit{est}_{\mathbf{t}}(\mathbf{e}) + \epsilon\mathbf{t}$

ϵ-Heavy Hitters

Algorithm Analysis

At any time $\mathbf{t}$, our estimates are:

$$\begin{aligned} \text{est}_{\mathbf{t}}(\mathbf{e}) &= \mathbf{C[j]} && \text{if } \mathbf{e} = \mathbf{T[j]} \\ &= \mathbf{0} && \text{otherwise} \end{aligned}$$

Lemma

Estimates satisfy: $\text{est}_{\mathbf{t}}(\mathbf{e}) \leq \text{count}_{\mathbf{t}}(\mathbf{e}) \leq \text{est}_{\mathbf{t}}(\mathbf{e}) + \epsilon\mathbf{t}$

For each element, count is maintained up to $\epsilon\mathbf{t}$ error!

ϵ-Heavy Hitters

Algorithm Analysis

At any time $\mathbf{t}$, our estimates are:

$$\begin{aligned} \text{est}_{\mathbf{t}}(\mathbf{e}) &= \mathbf{C[j]} && \text{if } \mathbf{e} = \mathbf{T[j]} \\ &= \mathbf{0} && \text{otherwise} \end{aligned}$$

Lemma

Estimates satisfy: $\text{est}_{\mathbf{t}}(\mathbf{e}) \leq \text{count}_{\mathbf{t}}(\mathbf{e}) \leq \text{est}_{\mathbf{t}}(\mathbf{e}) + \epsilon\mathbf{t}$

For each element, count is maintained up to $\epsilon\mathbf{t}$ error!

If $\mathbf{e}$ is not an ϵ-heavy hitter then $\text{count}_{\mathbf{t}}(\mathbf{e}) \leq \epsilon\mathbf{t}$, and hence $\text{est}_{\mathbf{t}}(\mathbf{e}) = \mathbf{0}$ is correct up to $\epsilon\mathbf{t}$ error.

ϵ-Heavy Hitters

Algorithm Analysis

At any time $\mathbf{t}$, our estimates are:

$$\begin{aligned} \mathrm{est}_{\mathbf{t}}(\mathbf{e}) &= \mathbf{C[j]} && \text{if } \mathbf{e} = \mathbf{T[j]} \\ &= \mathbf{0} && \text{otherwise} \end{aligned}$$

Lemma

Estimates satisfy: $\mathit{est}_{\mathbf{t}}(\mathbf{e}) \leq \mathit{count}_{\mathbf{t}}(\mathbf{e}) \leq \mathit{est}_{\mathbf{t}}(\mathbf{e}) + \epsilon\mathbf{t}$

Corollary

For any time $\mathbf{t}$, $\mathbf{T}$ *contains all the* ϵ*-heavy hitters in* $\mathbf{a}_{[1..t]}$.

Proof.

If $\mathbf{e}$ is a heavy hitter at time $\mathbf{t}$ then $\mathrm{count}_{\mathbf{t}}(\mathbf{e}) > \epsilon\mathbf{t}$.

ϵ-Heavy Hitters

Algorithm Analysis

At any time $\mathbf{t}$, our estimates are:

$$\begin{aligned} \mathsf{est}_{\mathbf{t}}(\mathbf{e}) &= \mathbf{C[j]} & \text{if } \mathbf{e} = \mathbf{T[j]} \\ &= \mathbf{0} & \text{otherwise} \end{aligned}$$

Lemma

Estimates satisfy: $est_{\mathbf{t}}(\mathbf{e}) \leq count_{\mathbf{t}}(\mathbf{e}) \leq est_{\mathbf{t}}(\mathbf{e}) + \epsilon\mathbf{t}$

Corollary

For any time $\mathbf{t}$, $\mathbf{T}$ *contains all the* ϵ*-heavy hitters in* $\mathbf{a}_{[1..t]}$.

Proof.

If $\mathbf{e}$ is a heavy hitter at time $\mathbf{t}$ then $\mathsf{count}_{\mathbf{t}}(\mathbf{e}) > \epsilon\mathbf{t}$.Using the lemma,

$$\mathsf{est}_{\mathbf{t}}(\mathbf{e}) \geq \mathsf{count}_{\mathbf{t}}(\mathbf{e}) - \epsilon\mathbf{t}$$

At any time $\mathbf{t}$, our estimates are:

$$\begin{aligned} \text{est}_t(\mathbf{e}) &= \mathbf{C[j]} && \text{if } \mathbf{e} = \mathbf{T[j]} \\ &= \mathbf{0} && \text{otherwise} \end{aligned}$$

Lemma

Estimates satisfy: $\textit{est}_t(\mathbf{e}) \leq \textit{count}_t(\mathbf{e}) \leq \textit{est}_t(\mathbf{e}) + \epsilon\mathbf{t}$

Corollary

For any time $\mathbf{t}$, $\mathbf{T}$ *contains all the* ϵ*-heavy hitters in* $\mathbf{a_{[1..t]}}$.

Proof.

If $\mathbf{e}$ is a heavy hitter at time $\mathbf{t}$ then $\text{count}_t(\mathbf{e}) > \epsilon\mathbf{t}$.Using the lemma,

$$\text{est}_t(\mathbf{e}) \geq \text{count}_t(\mathbf{e}) - \epsilon\mathbf{t} > \mathbf{0}$$

□

ϵ-Heavy Hitters

Algorithm Analysis

At any time $\mathbf{t}$, our estimates are:

$$\begin{aligned} \text{est}_{\mathbf{t}}(\mathbf{e}) &= \mathbf{C[j]} && \text{if } \mathbf{e} = \mathbf{T[j]} \\ &= \mathbf{0} && \text{otherwise} \end{aligned}$$

Lemma

Estimates satisfy: $\mathbf{0} \leq \mathit{count}_{\mathbf{t}}(\mathbf{e}) - \mathit{est}_{\mathbf{t}}(\mathbf{e}) \leq \frac{\mathbf{t}}{\mathbf{k+1}} \leq \epsilon\mathbf{t}$

Proof.

Counter for $\mathbf{e}$ increases only when we see $\mathbf{e}$, $\therefore$ $\text{est}_{\mathbf{t}}(\mathbf{e}) \leq \text{count}_{\mathbf{t}}(\mathbf{e})$.

ϵ-Heavy Hitters

Algorithm Analysis

At any time $\mathbf{t}$, our estimates are:

$$\begin{aligned} \mathrm{est}_{\mathbf{t}}(\mathbf{e}) &= \mathbf{C[j]} && \text{if } \mathbf{e} = \mathbf{T[j]} \\ &= \mathbf{0} && \text{otherwise} \end{aligned}$$

Lemma

Estimates satisfy: $\mathbf{0} \leq \mathit{count}_{\mathbf{t}}(\mathbf{e}) - \mathit{est}_{\mathbf{t}}(\mathbf{e}) \leq \frac{\mathbf{t}}{\mathbf{k+1}} \leq \epsilon\mathbf{t}$

Proof.

Counter for $\mathbf{e}$ increases only when we see $\mathbf{e}$, $\therefore$ $\mathrm{est}_{\mathbf{t}}(\mathbf{e}) \leq \mathrm{count}_{\mathbf{t}}(\mathbf{e})$.
$\mathrm{count}_{\mathbf{t}}(\mathbf{e}) - \mathrm{est}_{\mathbf{t}}(\mathbf{e})$ increases by one,

- when we decrease all $\mathbf{k}$ counters, and see an element outside $\mathbf{T}$

ϵ-Heavy Hitters

Algorithm Analysis

At any time $\mathbf{t}$, our estimates are:

$$\begin{aligned} \mathsf{est}_t(\mathbf{e}) &= \mathbf{C[j]} && \text{if } \mathbf{e} = \mathbf{T[j]} \\ &= \mathbf{0} && \text{otherwise} \end{aligned}$$

Lemma

Estimates satisfy: $\mathbf{0} \leq \mathit{count}_t(\mathbf{e}) - \mathit{est}_t(\mathbf{e}) \leq \frac{t}{k+1} \leq \epsilon t$

Proof.

Counter for $\mathbf{e}$ increases only when we see $\mathbf{e}$, $\therefore$ $\mathsf{est}_t(\mathbf{e}) \leq \mathsf{count}_t(\mathbf{e})$.
$\mathsf{count}_t(\mathbf{e}) - \mathsf{est}_t(\mathbf{e})$ increases by one,

- when we decrease all $\mathbf{k}$ counters, and see an element outside $\mathbf{T}$
- this is like discarding $\mathbf{k+1}$ elements.
- up to time $\mathbf{t}$, we have only $\mathbf{t}$ elements to discard

ϵ-Heavy Hitters

Algorithm Analysis

At any time $\mathbf{t}$, our estimates are:

$$\begin{aligned} \text{est}_t(\mathbf{e}) &= \mathbf{C[j]} && \text{if } \mathbf{e} = \mathbf{T[j]} \\ &= \mathbf{0} && \text{otherwise} \end{aligned}$$

Lemma

Estimates satisfy: $\mathbf{0} \leq \mathit{count}_t(\mathbf{e}) - \mathit{est}_t(\mathbf{e}) \leq \frac{\mathbf{t}}{\mathbf{k+1}} \leq \epsilon \mathbf{t}$

Proof.

Counter for $\mathbf{e}$ increases only when we see $\mathbf{e}$, $\therefore$ $\text{est}_t(\mathbf{e}) \leq \text{count}_t(\mathbf{e})$.
$\text{count}_t(\mathbf{e}) - \text{est}_t(\mathbf{e})$ increases by one,

- when we decrease all $\mathbf{k}$ counters, and see an element outside $\mathbf{T}$
- this is like discarding $\mathbf{k+1}$ elements.
- up to time $\mathbf{t}$, we have only $\mathbf{t}$ elements to discard

So at most $\mathbf{t/(k+1)}$ such increases. □

ϵ-Heavy Hitters: Algorithm

Space usage

Set $\mathbf{k} = \lceil 1/\epsilon \rceil - 1$. (if $\epsilon = 1/2$ then $\mathbf{k} = 1$)

Algorithm

Keep an array $\mathbf{T}[1, \ldots, \mathbf{k}]$ to hold elements
Keep an array $\mathbf{C}[1, \ldots, \mathbf{k}]$ to hold their counters

$\vdots$

Maintains $\mathbf{O}(1/\epsilon)$ counters and elements.

ϵ-Heavy Hitters: Algorithm

Space usage

Set $\mathbf{k} = \lceil 1/\epsilon \rceil - 1$. (if $\epsilon = 1/2$ then $\mathbf{k} = 1$)

Algorithm

Keep an array $\mathbf{T[1, \ldots, k]}$ to hold elements
Keep an array $\mathbf{C[1, \ldots, k]}$ to hold their counters

$\vdots$

Maintains $\mathbf{O(1/\epsilon)}$ counters and elements.
$\mathbf{O(\log t)}$ for each counter. $\mathbf{O(\log \Sigma)}$ for each element, where $\mathbf{\Sigma}$ is the description of largest element.

ϵ-Heavy Hitters: Algorithm

Space usage

Set $\mathbf{k} = \lceil 1/\epsilon \rceil - 1$. (if $\epsilon = 1/2$ then $\mathbf{k} = 1$)

Algorithm

Keep an array $\mathbf{T[1, \ldots, k]}$ to hold elements
Keep an array $\mathbf{C[1, \ldots, k]}$ to hold their counters

$\vdots$

Maintains $\mathbf{O(1/\epsilon)}$ counters and elements.
$\mathbf{O(\log t)}$ for each counter. $\mathbf{O(\log \Sigma)}$ for each element, where $\mathbf{\Sigma}$ is the description of largest element.

Total: $\mathbf{O(1/\epsilon(\log t + \log \Sigma))}$.
Recall: maintains counts for all elements up to $\epsilon \mathbf{t}$ error.

Part II

Use of Hash Functions

Maintaining Counts

Problem Statement:

At any time **t**, given an element **e**, estimate the number of times an **e** appeared so far.

Maintaining Counts

Problem Statement:

At any time $\mathbf{t}$, given an element $\mathbf{e}$, estimate the number of times an $\mathbf{e}$ appeared so far.

If error up to $\epsilon \mathbf{t}$ is OK, then we can use ϵ-heavy hitter algorithm.

Maintaining Counts

Problem Statement:

At any time $\mathbf{t}$, given an element $\mathbf{e}$, estimate the number of times an $\mathbf{e}$ appeared so far.

If error up to $\epsilon \mathbf{t}$ is OK, then we can use ϵ-heavy hitter algorithm.

It takes $\mathbf{O(1/\epsilon(\log t + \log \Sigma))}$ space.

Maintaining Counts

Problem Statement:

At any time $\mathbf{t}$, given an element $\mathbf{e}$, estimate the number of times an $\mathbf{e}$ appeared so far.

If error up to $\epsilon\mathbf{t}$ is OK, then we can use ϵ-heavy hitter algorithm.

It takes $\mathbf{O(1/\epsilon(\log t + \log \Sigma))}$ space.

Can we do better?

Yes – Bloom filter like idea

Recall: Bloom Filter

Storage for inserts and lookups

Sample hash functions $\mathbf{h_1, \ldots, h_d}$ independently and uniformly at random.

```
Insert(e)
  For i = 1...d
    Set T_i[h_i(e)] ← 1
```

```
Lookup(e)
  For i = 1...d
    If (T_i[h_i(e)] == 0) then return "No"
  Return "Yes"
```

If $\mathbf{e}$ inserted, then Lookup($\mathbf{e}$) will always return "Yes".

Recall: Bloom Filter

Storage for inserts and lookups

Sample hash functions $\mathbf{h_1, \ldots, h_d}$ independently and uniformly at random.

```
Insert(e)
  For i = 1...d
    Set T_i[h_i(e)] ← 1
```

```
Lookup(e)
  For i = 1...d
    If (T_i[h_i(e)] == 0) then return "No"
  Return "Yes"
```

If $\mathbf{e}$ inserted, then Lookup($\mathbf{e}$) will always return "Yes".

$\mathbf{e}$ not inserted, but still it can return "Yes" with very low probability.

- Due to some $\mathbf{e'}$s being inserted with $\mathbf{h_i(e') = h_i(e)}$.
- If $\mathbf{Pr[e}$ not inserted and $\mathbf{T_i[h_i(e)] = 1]} \leq \alpha$, then combined error probability would be at most $\alpha^{\mathbf{d}}$.

Count Min-Sketch

By G. Cormode and S. M. Muthukrishnan'05

Keep $\mathbf{d}$ arrays $\mathbf{C_1, \ldots, C_d}$, each to hold $\mathbf{m}$ counters.

$\mathcal{H}$: **2-universal family** of hash functions $\mathbf{h : U \to \{0, \ldots, m-1\}}$.
Sample $\mathbf{h_1, \ldots, h_d}$ independently and uniformly at random from $\mathcal{H}$.

Count Min-Sketch

By G. Cormode and S. M. Muthukrishnan'05

Keep $\mathbf{d}$ arrays $\mathbf{C_1, ..., C_d}$, each to hold $\mathbf{m}$ counters.

$\mathcal{H}$: $\mathbf{2}$-**universal family** of hash functions $\mathbf{h : U \rightarrow \{0, \ldots, m-1\}}$.
Sample $\mathbf{h_1, \ldots, h_d}$ independently and uniformly at random from $\mathcal{H}$.

```
CMInsert(e)
  For i = 1...d
    Do C_i[h_i(e)] + +
```

```
CMEstimate(e)
  est ← ∞
  For i = 1...d
    est ← min{est, C_i[h_i(e)]}
  Return est
```

As element $\mathbf{a_t}$ arrives at time $\mathbf{t}$, call CMInsert($\mathbf{a_t}$).

To get count of $\mathbf{e}$ at any time $\mathbf{t}$, call CMEstimate($\mathbf{e}$).

Count Min-Sketch

By G. Cormode and S. M. Muthukrishnan'05

CMInsert($\mathbf{e}$)
  For $\mathbf{i = 1...d}$
    Do $\mathbf{C_i[h_i(e)] + +}$

CMEstimate($\mathbf{e}$)
  $\mathbf{est} \leftarrow \infty$
  For $\mathbf{i = 1...d}$
    $\mathbf{est} \leftarrow \mathbf{min\{est, C_i[h_i(e)]\}}$
  Return $\mathbf{est}$

At time $\mathbf{t}$, let $\text{est}_\mathbf{t}(\mathbf{e}) = \text{CMEstimate}(\mathbf{e})$.
Observation: $\text{est}_\mathbf{t}(\mathbf{e}) \geq \text{count}_\mathbf{t}(\mathbf{e})$.

Question: How big $(\text{est}_\mathbf{t}(\mathbf{e}) - \text{count}_\mathbf{t}(\mathbf{e}))$ can be?

Count Min-Sketch

By G. Cormode and S. M. Muthukrishnan'05

```
CMInsert(e)
  For i = 1...d
    Do C_i[h_i(e)]++
```

```
CMEstimate(e)
  est ← ∞
  For i = 1...d
    est ← min{est, C_i[h_i(e)]}
  Return est
```

At time $\mathbf{t}$, let $\mathrm{est}_{\mathbf{t}}(\mathbf{e}) = \mathrm{CMEstimate}(\mathbf{e})$.
Observation: $\mathrm{est}_{\mathbf{t}}(\mathbf{e}) \geq \mathrm{count}_{\mathbf{t}}(\mathbf{e})$.

Question: How big $(\mathrm{est}_{\mathbf{t}}(\mathbf{e}) - \mathrm{count}_{\mathbf{t}}(\mathbf{e}))$ can be?

Recall: Any $\mathbf{e}, \mathbf{y} \in \mathbf{U}$, if $\mathbf{e} \neq \mathbf{y}$ then $\mathbf{Pr}[\mathbf{h_i}(\mathbf{y}) = \mathbf{h_i}(\mathbf{e})] = \frac{1}{\mathbf{m}}\ \forall \mathbf{i}$.

Count Min-Sketch: Analysis

By G. Cormode and S. M. Muthukrishnan'05

For simplicity let $\mathbf{f'_e} = \text{est}_\mathbf{t}\mathbf{(e)}$ and $\mathbf{f_e} = \text{count}_\mathbf{t}\mathbf{(e)}$. Bound $\mathbf{f'_e} - \mathbf{f_e}$.

Observations:

Count Min-Sketch: Analysis

By G. Cormode and S. M. Muthukrishnan'05

For simplicity let $f'_e = \text{est}_t(e)$ and $f_e = \text{count}_t(e)$. Bound $f'_e - f_e$.

Observations:

Define indicator variable $X_{i,e,y} = [h_i(y) = h_i(e)]$.

$$E[X_{i,e,y}] = \Pr[h_i(y) = h_i(e)] = 1/m$$

Count Min-Sketch: Analysis

By G. Cormode and S. M. Muthukrishnan'05

For simplicity let $\mathbf{f'_e} = \text{est}_\mathbf{t}\mathbf{(e)}$ and $\mathbf{f_e} = \text{count}_\mathbf{t}\mathbf{(e)}$. Bound $\mathbf{f'_e - f_e}$.

Observations:

Define indicator variable $\mathbf{X_{i,e,y} = [h_i(y) = h_i(e)]}$.

$$\mathbf{E[X_{i,e,y}] = Pr[h_i(y) = h_i(e)] = 1/m}$$

Let $\mathbf{X_{i,e} := \sum_{y \neq e} X_{i,e,y} f_y}$ be the total over counting at $\mathbf{C_i[h_i(e)]}$.

$$\mathbf{C_i[h_i(e)] = X_{i,e} + f_e}$$

Count Min-Sketch: Analysis

By G. Cormode and S. M. Muthukrishnan'05

For simplicity let $\mathbf{f}'_{\mathbf{e}} = \text{est}_{\mathbf{t}}(\mathbf{e})$ and $\mathbf{f}_{\mathbf{e}} = \text{count}_{\mathbf{t}}(\mathbf{e})$. Bound $\mathbf{f}'_{\mathbf{e}} - \mathbf{f}_{\mathbf{e}}$.

Observations:

Define indicator variable $\mathbf{X}_{\mathbf{i,e,y}} = [\mathbf{h}_{\mathbf{i}}(\mathbf{y}) = \mathbf{h}_{\mathbf{i}}(\mathbf{e})]$.

$$\mathbf{E}[\mathbf{X}_{\mathbf{i,e,y}}] = \mathbf{Pr}[\mathbf{h}_{\mathbf{i}}(\mathbf{y}) = \mathbf{h}_{\mathbf{i}}(\mathbf{e})] = \mathbf{1/m}$$

Let $\mathbf{X}_{\mathbf{i,e}} := \sum_{\mathbf{y} \neq \mathbf{e}} \mathbf{X}_{\mathbf{i,e,y}} \mathbf{f}_{\mathbf{y}}$ be the total over counting at $\mathbf{C}_{\mathbf{i}}[\mathbf{h}_{\mathbf{i}}(\mathbf{e})]$.

$$\mathbf{C}_{\mathbf{i}}[\mathbf{h}_{\mathbf{i}}(\mathbf{e})] = \mathbf{X}_{\mathbf{i,e}} + \mathbf{f}_{\mathbf{e}}$$

and since at most $\mathbf{t}$ elements have arrived so far,

$$\mathbf{E}[\mathbf{X}_{\mathbf{i,e}}] = \sum_{\mathbf{y} \neq \mathbf{e}} \mathbf{E}[\mathbf{X}_{\mathbf{i,e,y}}]\, \mathbf{f}_{\mathbf{y}} = \frac{\mathbf{1}}{\mathbf{m}} \sum_{\mathbf{y} \neq \mathbf{e}} \mathbf{f}_{\mathbf{y}} \leq \frac{\mathbf{t}}{\mathbf{m}}$$

Count Min-Sketch: Analysis

By G. Cormode and S. M. Muthukrishnan'05

$\mathbf{C_i[h_i(e)] = X_{i,e} + f_e}$ and $\mathbf{E[X_{i,e}] \leq \frac{t}{m}}$.
For $\epsilon > \mathbf{0}$

$$\mathbf{Pr[C_i[h_i(e)] - f_e \geq \epsilon t]} \;=\; \mathbf{Pr[X_{i,e} \geq \epsilon t]} \qquad \text{[definition]}$$

Count Min-Sketch: Analysis

By G. Cormode and S. M. Muthukrishnan'05

$\mathbf{C_i[h_i(e)] = X_{i,e} + f_e}$ and $\mathbf{E[X_{i,e}] \le \frac{t}{m}}$.
For $\epsilon > \mathbf{0}$

$$\begin{aligned} \mathbf{Pr[C_i[h_i(e)] - f_e \ge \epsilon t]} &= \mathbf{Pr[X_{i,e} \ge \epsilon t]} && \text{[definition]} \\ &\le \frac{\mathbf{E[X_{i,e}]}}{\epsilon \mathbf{t}} && \text{[Markov's inequality]} \end{aligned}$$

Count Min-Sketch: Analysis

By G. Cormode and S. M. Muthukrishnan'05

$C_i[h_i(e)] = X_{i,e} + f_e$ and $E[X_{i,e}] \leq \frac{t}{m}$.
For $\epsilon > 0$

$$\begin{aligned}
\Pr[C_i[h_i(e)] - f_e \geq \epsilon t] &= \Pr[X_{i,e} \geq \epsilon t] && \text{[definition]} \\
&\leq \frac{E[X_{i,e}]}{\epsilon t} && \text{[Markov's inequality]} \\
&\leq \frac{t/m}{\epsilon t} = \frac{1}{m\epsilon} && \text{[derived above]}
\end{aligned}$$

Count Min-Sketch: Analysis

By G. Cormode and S. M. Muthukrishnan'05

$C_i[h_i(e)] = X_{i,e} + f_e$ and $E[X_{i,e}] \leq \frac{t}{m}$.
For $\epsilon > 0$

$$\begin{aligned} \Pr[C_i[h_i(e)] - f_e \geq \epsilon t] &= \Pr[X_{i,e} \geq \epsilon t] && \text{[definition]} \\ &\leq \frac{E[X_{i,e}]}{\epsilon t} && \text{[Markov's inequality]} \\ &\leq \frac{t/m}{\epsilon t} = \frac{1}{m\epsilon} && \text{[derived above]} \end{aligned}$$

Recall: $f'_e = \text{est}_t(e) = \min_{i=1}^{d} C_i[h_i(e)]$.

Count Min-Sketch: Analysis

By G. Cormode and S. M. Muthukrishnan'05

$C_i[h_i(e)] = X_{i,e} + f_e$ and $E[X_{i,e}] \leq \frac{t}{m}$.
For $\epsilon > 0$

$$
\begin{aligned}
Pr[C_i[h_i(e)] - f_e \geq \epsilon t] &= Pr[X_{i,e} \geq \epsilon t] && \text{[definition]} \\
&\leq \frac{E[X_{i,e}]}{\epsilon t} && \text{[Markov's inequality]} \\
&\leq \frac{t/m}{\epsilon t} = \frac{1}{m\epsilon} && \text{[derived above]}
\end{aligned}
$$

Recall: $f'_e = est_t(e) = \min_{i=1}^{d} C_i[h_i(e)]$.

$$Pr\left[f'_e - f_e \geq \epsilon t\right] = Pr[C_i[h_i(e)] - f_e \geq \epsilon t \text{ for all } i]$$

Count Min-Sketch: Analysis

By G. Cormode and S. M. Muthukrishnan'05

$C_i[h_i(e)] = X_{i,e} + f_e$ and $E[X_{i,e}] \le \frac{t}{m}$.
For $\epsilon > 0$

$$\begin{aligned} \Pr[C_i[h_i(e)] - f_e \ge \epsilon t] &= \Pr[X_{i,e} \ge \epsilon t] && \text{[definition]} \\ &\le \frac{E[X_{i,e}]}{\epsilon t} && \text{[Markov's inequality]} \\ &\le \frac{t/m}{\epsilon t} = \frac{1}{m\epsilon} && \text{[derived above]} \end{aligned}$$

Recall: $f'_e = \text{est}_t(e) = \min_{i=1}^{d} C_i[h_i(e)]$.

$$\begin{aligned} \Pr\left[f'_e - f_e \ge \epsilon t\right] &= \Pr[C_i[h_i(e)] - f_e \ge \epsilon t \text{ for all } i] \\ &= \Pr[X_{i,e} \ge \epsilon t \text{ for all } i] \end{aligned}$$

Count Min-Sketch: Analysis

By G. Cormode and S. M. Muthukrishnan'05

$C_i[h_i(e)] = X_{i,e} + f_e$ and $E[X_{i,e}] \leq \frac{t}{m}$.
For $\epsilon > 0$

$$
\begin{aligned}
\Pr[C_i[h_i(e)] - f_e \geq \epsilon t] &= \Pr[X_{i,e} \geq \epsilon t] && \text{[definition]} \\
&\leq \frac{E[X_{i,e}]}{\epsilon t} && \text{[Markov's inequality]} \\
&\leq \frac{t/m}{\epsilon t} = \frac{1}{m\epsilon} && \text{[derived above]}
\end{aligned}
$$

Recall: $f'_e = \text{est}_t(e) = \min_{i=1}^{d} C_i[h_i(e)]$.

$$
\begin{aligned}
\Pr\left[f'_e - f_e \geq \epsilon t\right] &= \Pr[C_i[h_i(e)] - f_e \geq \epsilon t \text{ for all } i] \\
&= \Pr[X_{i,e} \geq \epsilon t \text{ for all } i] \\
&= \Pi_{i=1}^{d} \Pr[X_{i,e} \geq \epsilon t] && \text{[independence of } h_i\text{'s]}
\end{aligned}
$$

Count Min-Sketch: Analysis

By G. Cormode and S. M. Muthukrishnan'05

$C_i[h_i(e)] = X_{i,e} + f_e$ and $E[X_{i,e}] \leq \frac{t}{m}$.
For $\epsilon > 0$

$$\begin{aligned} Pr[C_i[h_i(e)] - f_e \geq \epsilon t] &= Pr[X_{i,e} \geq \epsilon t] && \text{[definition]} \\ &\leq \frac{E[X_{i,e}]}{\epsilon t} && \text{[Markov's inequality]} \\ &\leq \frac{t/m}{\epsilon t} = \frac{1}{m\epsilon} && \text{[derived above]} \end{aligned}$$

Recall: $f'_e = est_t(e) = \min_{i=1}^{d} C_i[h_i(e)]$.

$$\begin{aligned} Pr\left[f'_e - f_e \geq \epsilon t\right] &= Pr[C_i[h_i(e)] - f_e \geq \epsilon t \text{ for all } i] \\ &= Pr[X_{i,e} \geq \epsilon t \text{ for all } i] \\ &= \Pi_{i=1}^{d} Pr[X_{i,e} \geq \epsilon t] && \text{[independence of } h_i\text{'s]} \\ &\leq \left(\frac{1}{\epsilon m}\right)^d && \text{[derived above]} \end{aligned}$$

Count Min-Sketch: Analysis

By G. Cormode and S. M. Muthukrishnan'05

$$\mathbf{Pr}[\mathrm{est}_t(e) - \mathrm{count}_t(e) \geq \epsilon t] \leq \left(\frac{1}{\epsilon m}\right)^d$$

Count Min-Sketch: Analysis

By G. Cormode and S. M. Muthukrishnan'05

$$\mathbf{Pr}[\mathrm{est}_{\mathbf{t}}(\mathbf{e}) - \mathrm{count}_{\mathbf{t}}(\mathbf{e}) \geq \epsilon\mathbf{t}] \leq \left(\frac{\mathbf{1}}{\epsilon\mathbf{m}}\right)^{\mathbf{d}}$$

Set $\mathbf{m} = \lceil \mathbf{2}/\epsilon \rceil$ and $\mathbf{d} = \lceil \mathbf{lg}\, \mathbf{1}/\delta \rceil$ gives us

$$\mathbf{Pr}[\mathrm{est}_{\mathbf{t}}(\mathbf{e}) - \mathrm{count}_{\mathbf{t}}(\mathbf{e}) \geq \epsilon\mathbf{t}] \leq \delta$$

Count Min-Sketch: Analysis

By G. Cormode and S. M. Muthukrishnan'05

$$\Pr[\text{est}_t(e) - \text{count}_t(e) \geq \epsilon t] \leq \left(\frac{1}{\epsilon m}\right)^d$$

Set $m = \lceil 2/\epsilon \rceil$ and $d = \lceil \lg 1/\delta \rceil$ gives us

$$\Pr[\text{est}_t(e) - \text{count}_t(e) \geq \epsilon t] \leq \delta$$

Space: $m * d$ counters each of size $\lg(t) = O(\frac{1}{\epsilon} \lg 1/\delta \lg t)$.

Count Min-Sketch: Analysis

By G. Cormode and S. M. Muthukrishnan'05

Lemma

Given $\epsilon, \delta > \mathbf{0}$, *we can estimate* $count_{\mathbf{t}}(\mathbf{e})$, *at any time* $\mathbf{t}$ *for any element* $\mathbf{e}$, *up to* $\epsilon\mathbf{t}$ *error with probability at least* $(\mathbf{1} - \delta)$ *using* $\mathbf{O}(\frac{1}{\epsilon} \lg \mathbf{1}/\delta)$ *many counters.*