

# Lecture 5: Streaming

Rafael Oliveira

University of Waterloo  
Cheriton School of Computer Science

rafael.oliveira.teaching@gmail.com

September 23, 2020

# Overview

- Introduction
  - Data Streaming
  - Basic Examples
- Main Examples
  - Heavy hitters
  - Distinct Elements
- Acknowledgements

## Why streaming?

In today's world we have to deal with *big data*. But not all big data are created equal. Today we will study one way in which massive data can appear in our lives: *streaming*.

## Why streaming?

In today's world we have to deal with *big data*. But not all big data are created equal. Today we will study one way in which massive data can appear in our lives: *streaming*.

- ① Data stream: *massive* sequence of data, too large to store in memory.

# Why streaming?

In today's world we have to deal with *big data*. But not all big data are created equal. Today we will study one way in which massive data can appear in our lives: *streaming*.

- 1 Data stream: *massive* sequence of data, too large to store in memory.
  - 1 Network traffic (source/destination)
  - 2 Internet search logs
  - 3 Database transactions
  - 4 sensor networks
  - 5 satellite data feeds

# Why streaming?

In today's world we have to deal with *big data*. But not all big data are created equal. Today we will study one way in which massive data can appear in our lives: *streaming*.

- ① Data stream: *massive* sequence of data, too large to store in memory.
  - ① Network traffic (source/destination)
  - ② Internet search logs
  - ③ Database transactions
  - ④ sensor networks
  - ⑤ satellite data feeds
- ② **Does not** come to us at once.

# Why streaming?

In today's world we have to deal with *big data*. But not all big data are created equal. Today we will study one way in which massive data can appear in our lives: *streaming*.

- ① Data stream: *massive* sequence of data, too large to store in memory.
  - ① Network traffic (source/destination)
  - ② Internet search logs
  - ③ Database transactions
  - ④ sensor networks
  - ⑤ satellite data feeds
- ② **Does not** come to us at once.
- ③ Essentially can only look at each piece of data once (or constantly many times)

# Why streaming?

In today's world we have to deal with *big data*. But not all big data are created equal. Today we will study one way in which massive data can appear in our lives: *streaming*.

- 1 Data stream: *massive* sequence of data, too large to store in memory.
  - 1 Network traffic (source/destination)
  - 2 Internet search logs
  - 3 Database transactions
  - 4 sensor networks
  - 5 satellite data feeds
- 2 **Does not** come to us at once.
- 3 Essentially can only look at each piece of data once (or constantly many times)

How can we deal with it/model it? What can we do if we cannot even see the whole input?



# What is streaming?

## Definition (Basic Data Stream model)

In the data stream model:

# What is streaming?

## Definition (Basic Data Stream model)

In the data stream model:

- receive a stream of elements  $a_1, a_2, \dots, a_N$  each from a known alphabet  $\Sigma$ . Each element of  $\Sigma$  takes  $b$  bits to represent.
  - usually assume that  $N$  is known

# What is streaming?

## Definition (Basic Data Stream model)

In the data stream model:

- receive a stream of elements  $a_1, a_2, \dots, a_N$  each from a known alphabet  $\Sigma$ . Each element of  $\Sigma$  takes  $b$  bits to represent.
  - usually assume that  $N$  is known
- Basic operations (comparison, arithmetic, bitwise) take  $\Theta(1)$  time

# What is streaming?

## Definition (Basic Data Stream model)

In the data stream model:

- receive a stream of elements  $a_1, a_2, \dots, a_N$  each from a known alphabet  $\Sigma$ . Each element of  $\Sigma$  takes  $b$  bits to represent.
  - usually assume that  $N$  is known
- Basic operations (comparison, arithmetic, bitwise) take  $\Theta(1)$  time
- Single or small number of passes over data

# What is streaming?

## Definition (Basic Data Stream model)

In the data stream model:

- receive a stream of elements  $a_1, a_2, \dots, a_N$  each from a known alphabet  $\Sigma$ . Each element of  $\Sigma$  takes  $b$  bits to represent.
  - usually assume that  $N$  is known
- Basic operations (comparison, arithmetic, bitwise) take  $\Theta(1)$  time
- Single or small number of passes over data
- Bounded storage
  - Typically  $\log^c(N)$  for  $c = O(1)$  or  $N^\alpha$  for some  $0 < \alpha < 1$

# What is streaming?

## Definition (Basic Data Stream model)

In the data stream model:

- receive a stream of elements  $a_1, a_2, \dots, a_N$  each from a known alphabet  $\Sigma$ . Each element of  $\Sigma$  takes  $b$  bits to represent.
  - usually assume that  $N$  is known
- Basic operations (comparison, arithmetic, bitwise) take  $\Theta(1)$  time
- Single or small number of passes over data
- Bounded storage
  - Typically  $\log^c(N)$  for  $c = O(1)$  or  $N^\alpha$  for some  $0 < \alpha < 1$
- We are allowed to use randomness (almost always necessary)
  - *Probabilistic model*: our algorithm must succeed most of the time

# What is streaming?

## Definition (Basic Data Stream model)

In the data stream model:

- receive a stream of elements  $a_1, a_2, \dots, a_N$  each from a known alphabet  $\Sigma$ . Each element of  $\Sigma$  takes  $b$  bits to represent.
  - usually assume that  $N$  is known
- Basic operations (comparison, arithmetic, bitwise) take  $\Theta(1)$  time
- Single or small number of passes over data
- Bounded storage
  - Typically  $\log^c(N)$  for  $c = O(1)$  or  $N^\alpha$  for some  $0 < \alpha < 1$
- We are allowed to use randomness (almost always necessary)
  - *Probabilistic model*: our algorithm must succeed most of the time
- (usually) want *approximate answers* to the true answer

# What is streaming?

## Definition (Basic Data Stream model)

In the data stream model:

- receive a stream of elements  $a_1, a_2, \dots, a_N$  each from a known alphabet  $\Sigma$ . Each element of  $\Sigma$  takes  $b$  bits to represent.
  - usually assume that  $N$  is known
- Basic operations (comparison, arithmetic, bitwise) take  $\Theta(1)$  time
- Single or small number of passes over data
- Bounded storage
  - Typically  $\log^c(N)$  for  $c = O(1)$  or  $N^\alpha$  for some  $0 < \alpha < 1$
- We are allowed to use randomness (almost always necessary)
  - *Probabilistic model*: our algorithm must succeed most of the time
- (usually) want *approximate answers* to the true answer

*Goal*: minimize space complexity (in bits) and the processing time.



# Examples of Streaming Problems

## Example (Sum of elements)

- **Input stream:**  $a_1, \dots, a_N$  be integers from the set  $[-2^b + 1, 2^b - 1]$
- **Task:** maintain the current sum of the elements we have seen so far

# Examples of Streaming Problems

## Example (Sum of elements)

- **Input stream:**  $a_1, \dots, a_N$  be integers from the set  $[-2^b + 1, 2^b - 1]$
- **Task:** maintain the current sum of the elements we have seen so far

## Example (Median)

- **Input stream:**  $a_1, \dots, a_N$  be integers from the set  $[-2^b + 1, 2^b - 1]$
- **Task:** maintain the current median of elements we have seen so far

# Examples of Streaming Problems

## Example (Distinct elements)

- **Input stream:**  $a_1, \dots, a_N$  be integers from the set  $[-2^b + 1, 2^b - 1]$
- **Task:** maintain current # of distinct elements we have seen so far

# Examples of Streaming Problems

## Example (Distinct elements)

- **Input stream:**  $a_1, \dots, a_N$  be integers from the set  $[-2^b + 1, 2^b - 1]$
- **Task:** maintain current # of distinct elements we have seen so far

## Example (Heavy hitters)

- **Input stream:**  $a_1, \dots, a_N$  integers from  $[-2^b + 1, 2^b - 1]$ ,  $\epsilon > 0$
- **Task:** maintain set of elements that contains elements that have appeared at least  $\epsilon$ -fraction of the time (a.k.a. *heavy hitters*)
- **Constraint:** allowed to also output *false positives* (low hitters), but not allowed to miss any heavy hitter!

# Majority Element - Algorithm

Setup: heavy hitters with  $\epsilon = 1/2$ .

- At time  $t$ , we will maintain set  $S_t$  which contains the element that has appeared at least  $N/2$  times, if any.

# Majority Element - Algorithm

Setup: heavy hitters with  $\epsilon = 1/2$ .

- At time  $t$ , we will maintain set  $S_t$  which contains the element that has appeared at least  $N/2$  times, if any.
- $S_0 = \emptyset$ ,  $c \leftarrow 0$  ( $c$  is a counter)

# Majority Element - Algorithm

Setup: heavy hitters with  $\epsilon = 1/2$ .

- At time  $t$ , we will maintain set  $S_t$  which contains the element that has appeared at least  $N/2$  times, if any.
- $S_0 = \emptyset$ ,  $c \leftarrow 0$  ( $c$  is a counter)
- when element  $a_t$  arrives:

# Majority Element - Algorithm

Setup: heavy hitters with  $\epsilon = 1/2$ .

- At time  $t$ , we will maintain set  $S_t$  which contains the element that has appeared at least  $N/2$  times, if any.
- $S_0 = \emptyset$ ,  $c \leftarrow 0$  ( $c$  is a counter)
- when element  $a_t$  arrives:
  - If  $c == 0$ 
    - $S_t = \{a_t\}$  and  $c \leftarrow 1$



# Majority Element - Algorithm

Setup: heavy hitters with  $\epsilon = 1/2$ .

- At time  $t$ , we will maintain set  $S_t$  which contains the element that has appeared at least  $N/2$  times, if any.
- $S_0 = \emptyset$ ,  $c \leftarrow 0$  ( $c$  is a counter)
- when element  $a_t$  arrives:
  - If  $c == 0$ 
    - $S_t = \{a_t\}$  and  $c \leftarrow 1$
  - Else

# Majority Element - Algorithm

Setup: heavy hitters with  $\epsilon = 1/2$ .

- At time  $t$ , we will maintain set  $S_t$  which contains the element that has appeared at least  $N/2$  times, if any.
- $S_0 = \emptyset$ ,  $c \leftarrow 0$  ( $c$  is a counter)
- when element  $a_t$  arrives:
  - If  $c == 0$ 
    - $S_t = \{a_t\}$  and  $c \leftarrow 1$
  - Else
    - if  $a_t \in S_{t-1}$ , set  $c \leftarrow c + 1$
    - else  $c \leftarrow c - 1$  and discard  $a_t$

# Majority Element - Algorithm

Setup: heavy hitters with  $\epsilon = 1/2$ .

- At time  $t$ , we will maintain set  $S_t$  which contains the element that has appeared at least  $N/2$  times, if any.
- $S_0 = \emptyset$ ,  $c \leftarrow 0$  ( $c$  is a counter)
- when element  $a_t$  arrives:
  - If  $c == 0$ 
    - $S_t = \{a_t\}$  and  $c \leftarrow 1$
  - Else
    - if  $a_t \in S_{t-1}$ , set  $c \leftarrow c + 1$
    - else  $c \leftarrow c - 1$  and discard  $a_t$
- At end of stream, return element in  $S_N$

## Majority Element - Analysis

- If there is no majority element, we could still output a false positive (low hitter), which is fine.

## Majority Element - Analysis

- If there is no majority element, we could still output a false positive (low hitter), which is fine.
- What happens when there is a majority element?

## Majority Element - Analysis

- If there is no majority element, we could still output a false positive (low hitter), which is fine.
- What happens when there is a majority element?
  - Every time that we discard a copy of the majority element, we throw away a different element.

3 1 2 1 1

memory : $\emptyset$	count : 0	beginning
memory : 3	count : 1	on 3
memory : 3	count : 0	on 1
memory : 2	count : 1	on 2
memory : 2	count : 0	on 1
memory : 1	count : 1	on 1

## Majority Element - Analysis

- If there is no majority element, we could still output a false positive (low hitter), which is fine.
- What happens when there is a majority element?
  - Every time that we discard a copy of the majority element, we throw away a different element.
  - Majority element appears more than half the time, so we cannot throw away all the majority elements

## Majority Element - Analysis

- If there is no majority element, we could still output a false positive (low hitter), which is fine.
- What happens when there is a majority element?
  - Every time that we discard a copy of the majority element, we throw away a different element.
  - Majority element appears more than half the time, so we cannot throw away all the majority elements
- Space used:  $O(b)$  (stored set  $S_t$  which has at most one element and counter)



- Introduction
  - Data Streaming
  - Basic Examples
- Main Examples
  - Heavy hitters
  - Distinct Elements
- Acknowledgements

# Heavy hitters Problem

## Example (Heavy hitters)

- **Input stream:**  $a_1, \dots, a_N$  integers from  $[-2^b + 1, 2^b - 1]$ ,  $\epsilon > 0$
- **Task:** maintain set of elements that contains elements that have appeared at least  $\epsilon$ -fraction of the time (a.k.a. *heavy hitters*)
- **Constraint:** allowed to also output *false positives* (low hitters), but not allowed to miss any heavy hitter!

# Heavy Hitters Algorithm

- 1 Set  $k = \lceil 1/\epsilon \rceil - 1$

# Heavy Hitters Algorithm

- 1 Set  $k = \lceil 1/\epsilon \rceil - 1$
- 2 Set array  $T$  of length  $k$  where each entry  $T[i]$  can hold an element of  $\Sigma$  ( $= [-2^b + 1, 2^b - 1]$ ).

# Heavy Hitters Algorithm

- 1 Set  $k = \lceil 1/\epsilon \rceil - 1$
- 2 Set array  $T$  of length  $k$  where each entry  $T[i]$  can hold an element of  $\Sigma$  ( $= [-2^b + 1, 2^b - 1]$ ).
- 3 Set array  $C$  of length  $k$  where each entry can hold non-negative integer (counter)

# Heavy Hitters Algorithm

- 1 Set  $k = \lceil 1/\epsilon \rceil - 1$
- 2 Set array  $T$  of length  $k$  where each entry  $T[i]$  can hold an element of  $\Sigma$  ( $= [-2^b + 1, 2^b - 1]$ ).
- 3 Set array  $C$  of length  $k$  where each entry can hold non-negative integer
- 4 Initialize  $T[i] \leftarrow \text{NaN}$  and  $C[i] \leftarrow 0$  for  $i \in [k]$ .

# Heavy Hitters Algorithm

- 1 Set  $k = \lceil 1/\epsilon \rceil - 1$
- 2 Set array  $T$  of length  $k$  where each entry  $T[i]$  can hold an element of  $\Sigma$  ( $= [-2^b + 1, 2^b - 1]$ ).
- 3 Set array  $C$  of length  $k$  where each entry can hold non-negative integer
- 4 Initialize  $T[i] \leftarrow NaN$  and  $C[i] \leftarrow 0$  for  $i \in [k]$ .
- 5 When receive element  $a_t$ :

# Heavy Hitters Algorithm

- 1 Set  $k = \lceil 1/\epsilon \rceil - 1$
- 2 Set array  $T$  of length  $k$  where each entry  $T[i]$  can hold an element of  $\Sigma$  ( $= [-2^b + 1, 2^b - 1]$ ).
- 3 Set array  $C$  of length  $k$  where each entry can hold non-negative integer
- 4 Initialize  $T[i] \leftarrow NaN$  and  $C[i] \leftarrow 0$  for  $i \in [k]$ .
- 5 When receive element  $a_t$ :
  - 1 If there is  $j \in [k]$  such that  $a_t = T[j]$ , then  $C[j] \leftarrow C[j] + 1$

*increase counter*



# Heavy Hitters Algorithm

- 1 Set  $k = \lceil 1/\epsilon \rceil - 1$
- 2 Set array  $T$  of length  $k$  where each entry  $T[i]$  can hold an element of  $\Sigma$  ( $= [-2^b + 1, 2^b - 1]$ ).
- 3 Set array  $C$  of length  $k$  where each entry can hold non-negative integer
- 4 Initialize  $T[i] \leftarrow NaN$  and  $C[i] \leftarrow 0$  for  $i \in [k]$ .
- 5 When receive element  $a_t$ :
  - 1 If there is  $j \in [k]$  such that  $a_t = T[j]$ , then  $C[j] \leftarrow C[j] + 1$
  - 2 Else, if there is  $j \in [k]$  such that  $C[j] = 0$ , then  $T[j] \leftarrow a_t$  and  $C[j] \leftarrow 1$

# Heavy Hitters Algorithm

- 1 Set  $k = \lceil 1/\epsilon \rceil - 1$
- 2 Set array  $T$  of length  $k$  where each entry  $T[i]$  can hold an element of  $\Sigma$  ( $= [-2^b + 1, 2^b - 1]$ ).
- 3 Set array  $C$  of length  $k$  where each entry can hold non-negative integer
- 4 Initialize  $T[i] \leftarrow NaN$  and  $C[i] \leftarrow 0$  for  $i \in [k]$ .
- 5 When receive element  $a_t$ :
  - 1 If there is  $j \in [k]$  such that  $a_t = T[j]$ , then  $C[j] \leftarrow C[j] + 1$
  - 2 Else, if there is  $j \in [k]$  such that  $C[j] = 0$ , then  $T[j] \leftarrow a_t$  and  $C[j] \leftarrow 1$
  - 3 Else make all  $C[j] \leftarrow C[j] - 1$  and discard  $a_t$

*decrease every counter*

# Heavy Hitters Algorithm

- 1 Set  $k = \lceil 1/\epsilon \rceil - 1$
- 2 Set array  $T$  of length  $k$  where each entry  $T[i]$  can hold an element of  $\Sigma$  ( $= [-2^b + 1, 2^b - 1]$ ).
- 3 Set array  $C$  of length  $k$  where each entry can hold non-negative integer
- 4 Initialize  $T[i] \leftarrow NaN$  and  $C[i] \leftarrow 0$  for  $i \in [k]$ .
- 5 When receive element  $a_t$ :
  - 1 If there is  $j \in [k]$  such that  $a_t = T[j]$ , then  $C[j] \leftarrow C[j] + 1$
  - 2 Else, if there is  $j \in [k]$  such that  $C[j] = 0$ , then  $T[j] \leftarrow a_t$  and  $C[j] \leftarrow 1$
  - 3 Else make all  $C[j] \leftarrow C[j] - 1$  and discard  $a_t$
- 6 Return the array  $T$  with the counter array  $C$

## Heavy hitters proof

- For element  $e \in \Sigma$ , let  $est(e) = \begin{cases} C[j], & \text{if } e = T[j] \\ 0, & \text{otherwise. } (e \notin T) \end{cases}$

## Heavy hitters proof

- For element  $e \in \Sigma$ , let  $est(e) = \begin{cases} C[j], & \text{if } e = T[j] \\ 0, & \text{otherwise.} \end{cases}$

### Lemma

*Let  $count(e)$  be the number of occurrences of  $e$  in stream up to time  $N$ .*

$$0 \leq count(e) - est(e) \leq \frac{N}{k+1} \leq \epsilon N$$

## Heavy hitters proof

- For element  $e \in \Sigma$ , let  $est(e) = \begin{cases} C[j], & \text{if } e = T[j] \\ 0, & \text{otherwise.} \end{cases}$

### Lemma

Let  $count(e)$  be the number of occurrences of  $e$  in stream up to time  $N$ .

$$0 \leq count(e) - est(e) \leq \frac{N}{k+1} \leq \epsilon N$$

- $count(e) \geq est(e)$  because never increase  $C[j]$  for  $e$  unless we see  $e$

## Heavy hitters proof

- For element  $e \in \Sigma$ , let  $est(e) = \begin{cases} C[j], & \text{if } e = T[j] \\ 0, & \text{otherwise.} \end{cases}$

### Lemma

Let  $count(e)$  be the number of occurrences of  $e$  in stream up to time  $N$ .

$$0 \leq count(e) - est(e) \leq \frac{N}{k+1} \leq \epsilon N$$

- $count(e) \geq est(e)$  because never increase  $C[j]$  for  $e$  unless we see  $e$
- If we don't increase  $est(e)$  by 1 when we see an update to  $e$  then we decrement  $k$  counters and discard current update to  $e$

## Heavy hitters proof

- For element  $e \in \Sigma$ , let  $est(e) = \begin{cases} C[j], & \text{if } e = T[j] \\ 0, & \text{otherwise.} \end{cases}$

### Lemma

Let  $count(e)$  be the number of occurrences of  $e$  in stream up to time  $N$ .

$$0 \leq count(e) - est(e) \leq \frac{N}{k+1} \leq \epsilon N$$

- $count(e) \geq est(e)$  because never increase  $C[j]$  for  $e$  unless we see  $e$
- If we don't increase  $est(e)$  by 1 when we see an update to  $e$  then we decrement  $k$  counters and discard current update to  $e$
- So we drop  $k+1$  distinct stream updates, but there are  $N$  updates, so we won't increase  $est(e)$  by 1 (when we should) at most

$$\frac{N}{k+1} \leq \epsilon N \text{ times.}$$

$$\rightarrow (k = \lceil \frac{1}{\epsilon} \rceil - 1)$$

my discard + k decrements



## Heavy hitters proof

- At any time  $N$ , all heavy hitters  $e$  are in  $T$

## Heavy hitters proof

- At any time  $N$ , all heavy hitters  $e$  are in  $T$ 
  - For an  $\epsilon$ -heavy hitter  $e$ , we have  $\text{count}(e) > \epsilon \cdot N$

## Heavy hitters proof

- At any time  $N$ , all heavy hitters  $e$  are in  $T$ 
  - For an  $\epsilon$ -heavy hitter  $e$ , we have  $count(e) > \epsilon \cdot N$
  - $est(e) \geq count(e) - \epsilon \cdot N > 0$  (by lemma)

## Heavy hitters proof

- At any time  $N$ , all heavy hitters  $e$  are in  $T$ 
  - For an  $\epsilon$ -heavy hitter  $e$ , we have  $\text{count}(e) > \epsilon \cdot N$
  - $\text{est}(e) \geq \text{count}(e) - \epsilon \cdot N > 0$
  - $\text{est}(e) > 0 \Rightarrow e$  is in  $T$

(definition of  $\text{est}$ )

$$\text{est}(e) = \begin{cases} C[i] & \text{if } e \in T \text{ ( } e = T[i] \text{ )} \\ 0 & \text{otherwise} \end{cases}$$

# Heavy hitters proof

- At any time  $N$ , all heavy hitters  $e$  are in  $T$ 
    - For an  $\epsilon$ -heavy hitter  $e$ , we have  $\text{count}(e) > \epsilon \cdot N$
    - $\text{est}(e) \geq \text{count}(e) - \epsilon \cdot N > 0$
    - $\text{est}(e) > 0 \Rightarrow e$  is in  $T$
    - Space used is  $O(k \cdot (\log(\Sigma) + \log N)) = O((1/\epsilon) \cdot (b + \log N))$  bits
- Handwritten notes:*
- $k$  is underlined in green.
  - $b$  is underlined in green.
  - A green arrow points to  $b$ .
  - $\log(\Sigma)$  is underlined in red, with "store  $T$ " written below it.
  - $\log N$  is underlined in green, with "one ctx" written below it.
  - $k \log N$  is underlined in blue, with "store counters" written below it.

## Heavy hitters with weights

### Example (Weighted heavy hitters)

- **Input stream:**  $(a_1, w_1), \dots, (a_N, w_N)$  tuples of integers from  $\Sigma = [-2^b + 1, 2^b - 1]$ , parameter  $q \in \mathbb{N}$

# Heavy hitters with weights

## Example (Weighted heavy hitters)

- **Input stream:**  $(a_1, w_1), \dots, (a_N, w_N)$  tuples of integers from  $\Sigma = [-2^b + 1, 2^b - 1]$ , parameter  $q \in \mathbb{N}$ 
  - Total weight

$$Q = \sum_{t=1}^N w_t$$

# Heavy hitters with weights

## Example (Weighted heavy hitters)

- **Input stream:**  $(a_1, w_1), \dots, (a_N, w_N)$  tuples of integers from  $\Sigma = [-2^b + 1, 2^b - 1]$ , parameter  $q \in \mathbb{N}$ 
  - Total weight

$$Q = \sum_{t=1}^N w_t$$

- Total weight of  $e \in \Sigma$ :

$$Q(e) = \sum_{t:a_t=e} w_t$$

take  $(a_t, w_t)$

if  $a_t = e$  then  $Q(e) += w_t$



# Heavy hitters with weights

## Example (Weighted heavy hitters)

- **Input stream:**  $(a_1, w_1), \dots, (a_N, w_N)$  tuples of integers from  $\Sigma = [-2^b + 1, 2^b - 1]$ , parameter  $q \in \mathbb{N}$ 
  - Total weight

$$Q = \sum_{t=1}^N w_t$$

- Total weight of  $e \in \Sigma$ :

$$Q(e) = \sum_{t:a_t=e} w_t$$

- **Task:** find all elements  $e$  such that  $Q(e) \geq q$

# Heavy hitters with weights

## Example (Weighted heavy hitters)

- **Input stream:**  $(a_1, w_1), \dots, (a_N, w_N)$  tuples of integers from  $\Sigma = [-2^b + 1, 2^b - 1]$ , parameter  $q \in \mathbb{N}$ 
  - Total weight

$$Q = \sum_{t=1}^N w_t$$

- Total weight of  $e \in \Sigma$ :

$$Q(e) = \sum_{t:a_t=e} w_t$$

- **Task:** find all elements  $e$  such that  $Q(e) \geq q$
- **Constraint:** allowed to also output *false positives* (low hitters), but not allowed to miss any heavy hitter!

## Weighted heavy hitters - algorithm setup

We will see an algorithm that gives us the following guarantees:

## Weighted heavy hitters - algorithm setup

We will see an algorithm that gives us the following guarantees:

- 1 All heavy hitters are reported

## Weighted heavy hitters - algorithm setup

We will see an algorithm that gives us the following guarantees:

- 1 All heavy hitters are reported
- 2 if  $Q(e) \leq \underline{q - \epsilon \cdot Q}$ , then  $e$  is reported with probability at most  $\underline{\delta}$ 
  - That is, have low probability of reporting a really low hitter *small*

## Weighted heavy hitters - algorithm setup

We will see an algorithm that gives us the following guarantees:

- 1 All heavy hitters are reported
- 2 if  $Q(e) \leq q - \epsilon \cdot Q$ , then  $e$  is reported with probability at most  $\delta$ 
  - That is, have low probability of reporting a really low hitter

**Idea:** Use 2-universal hash functions!

## Weighted heavy hitters - algorithm setup

We will see an algorithm that gives us the following guarantees:

- 1 All heavy hitters are reported
- 2 if  $Q(e) \leq q - \epsilon \cdot Q$ , then  $e$  is reported with probability at most  $\delta$ 
  - That is, have low probability of reporting a really low hitter

Use 2-universal hash functions!

- $k, \ell$  are parameters to be chosen later

## Weighted heavy hitters - algorithm setup

We will see an algorithm that gives us the following guarantees:

- 1 All heavy hitters are reported
- 2 if  $Q(e) \leq q - \epsilon \cdot Q$ , then  $e$  is reported with probability at most  $\delta$ 
  - That is, have low probability of reporting a really low hitter

Use 2-universal hash functions!

- $k, \ell$  are parameters to be chosen later
- Pick  $k$  hash functions  $h_1, \dots, h_k$  where  $h_i : \Sigma \rightarrow [0, \ell - 1]$

*uniformly at random*

*independently*



## Weighted heavy hitters - algorithm setup

We will see an algorithm that gives us the following guarantees:

- 1 All heavy hitters are reported
- 2 if  $Q(e) \leq q - \epsilon \cdot Q$ , then  $e$  is reported with probability at most  $\delta$ 
  - That is, have low probability of reporting a really low hitter

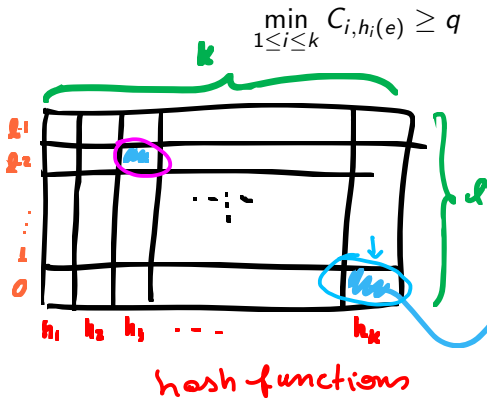
Use 2-universal hash functions!

- $k, \ell$  are parameters to be chosen later
- Pick  $k$  hash functions  $h_1, \dots, h_k$  where  $h_i : \Sigma \rightarrow [0, \ell - 1]$
- Let's maintain  $k \cdot \ell$  counters  $C_{i,j}$ , where each  $C_{i,j}$  adds the weight of items that are mapped to  $j^{\text{th}}$  entry by the  $i^{\text{th}}$  hash function. Start with  $C_{i,j} = 0$  for all  $1 \leq i \leq k$  and  $1 \leq j \leq \ell$ .

# Weighted heavy hitters - algorithm

- Given  $(a_t, w_t)$ , for each  $1 \leq i \leq k$  set  $C_{i, h_i(a_t)} \leftarrow C_{i, h_i(a_t)} + w_t$  *increase*
- At the end,<sup>1</sup> report all elements  $e$  with  $\min_{1 \leq i \leq k} C_{i, h_i(e)} \geq q$  *( $\geq 0$ )*

data structure  
Counters  
 range of hash fcn



sum of all  $w_t$ 's  
 s.t.  
 $h_k(a_t) = 0$   
 (i.e.  $C_{k,l}$ )

<sup>1</sup>In this version need to do second pass over data. But this can be fixed. Practice problem: fix this so that we can report on the fly.

## Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large

if  $a$  is heavy hitter

$$C_{i, h_i(a)} \geq \underbrace{\sum (\text{weights of } a)}_{Q(a)} \geq q$$

↑  
because  $a$   
heavy hitter

## Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large
- Need to show now that if  $e$  is not a heavy hitter, with high probability we will have one counter  $C_{i,h_i(e)} < q$ .

## Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large
- Need to show now that if  $e$  is not a heavy hitter, with high probability we will have one counter  $C_{i,h_i(e)} < q$ .
- If  $Q(e) \leq q - \epsilon \cdot Q$ , what is prob.  $e$  will be reported as heavy hitter?

*far from being heavy hitter*

## Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large
- Need to show now that if  $e$  is not a heavy hitter, with high probability we will have one counter  $C_{i,h_i(e)} < q$ .
- If  $Q(e) \leq q - \epsilon \cdot Q$ , what is prob.  $e$  will be reported as heavy hitter?
  - Look at counter  $C_{i,h_i(e)}$ . Since  $e$  is reported, must have  $C_{i,h_i(e)} \geq q$

## Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large
- Need to show now that if  $e$  is not a heavy hitter, with high probability we will have one counter  $C_{i,h_i(e)} < q$ .
- If  $Q(e) \leq q - \epsilon \cdot Q$ , what is prob.  $e$  will be reported as heavy hitter?
  - Look at counter  $C_{i,h_i(e)}$ . Since  $e$  is reported, must have  $C_{i,h_i(e)} \geq q$
  - Contribution from  $e$  is  $Q(e) \leq q - \epsilon \cdot Q$ . So other elements that map to  $h_i(e)$  must have contributed  $\geq \epsilon \cdot Q$ .

## Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large
- Need to show now that if  $e$  is not a heavy hitter, with high probability we will have one counter  $C_{i,h_i(e)} < q$ .
- If  $Q(e) \leq q - \epsilon \cdot Q$ , what is prob.  $e$  will be reported as heavy hitter?
  - Look at counter  $C_{i,h_i(e)}$ . Since  $e$  is reported, must have  $C_{i,h_i(e)} \geq q$
  - Contribution from  $e$  is  $Q(e) \leq q - \epsilon \cdot Q$ . So other elements that map to  $h_i(e)$  must have contributed  $\geq \epsilon \cdot Q$ .
  - Let  $Z_i$  be the value of  $C_{i,h_i(e)}$  that was added by other elements



## Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large
- Need to show now that if  $e$  is not a heavy hitter, with high probability we will have one counter  $C_{i,h_i(e)} < q$ .
- If  $Q(e) \leq q - \epsilon \cdot Q$ , what is prob.  $e$  will be reported as heavy hitter?
  - Look at counter  $C_{i,h_i(e)}$ . Since  $e$  is reported, must have  $C_{i,h_i(e)} \geq q$
  - Contribution from  $e$  is  $Q(e) \leq q - \epsilon \cdot Q$ . So other elements that map to  $h_i(e)$  must have contributed  $\geq \epsilon \cdot Q$ .
  - Let  $Z_i$  be the value of  $C_{i,h_i(e)}$  that was added by other elements
  - $h_i$  chosen from 2-universal hash family then probability that another element  $f$  is mapped to  $h_i(e)$  is  $\leq 1/\ell$ .

## Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large
- Need to show now that if  $e$  is not a heavy hitter, with high probability we will have one counter  $C_{i,h_i(e)} < q$ .
- If  $Q(e) \leq q - \epsilon \cdot Q$ , what is prob.  $e$  will be reported as heavy hitter?
  - Look at counter  $C_{i,h_i(e)}$ . Since  $e$  is reported, must have  $C_{i,h_i(e)} \geq q$
  - Contribution from  $e$  is  $Q(e) \leq q - \epsilon \cdot Q$ . So other elements that map to  $h_i(e)$  must have contributed  $\geq \epsilon \cdot Q$ .
  - Let  $Z_i$  be the value of  $C_{i,h_i(e)}$  that was added by other elements
  - $h_i$  chosen from 2-universal hash family then probability that another element  $f$  is mapped to  $h_i(e)$  is  $\leq 1/\ell$ .
  - Thus  $\mathbb{E}[Z_i] \leq Q/\ell$ . By Markov:

$$\Pr[Z_i \geq \epsilon \cdot Q] \leq \frac{\mathbb{E}[Z_i]}{\epsilon \cdot Q} \leq \frac{1}{\epsilon \ell}$$

2-universal  
hash  
fcn

## Weighted heavy hitters - analysis

- Heavy hitter always reported, as all their counters are large
- Need to show now that if  $e$  is not a heavy hitter, with high probability we will have one counter  $C_{i, h_i(e)} < q$ .
- If  $Q(e) \leq q - \epsilon \cdot Q$ , what is prob.  $e$  will be reported as heavy hitter?
  - Look at counter  $C_{i, h_i(e)}$ . Since  $e$  is reported, must have  $C_{i, h_i(e)} \geq q$
  - Contribution from  $e$  is  $Q(e) \leq q - \epsilon \cdot Q$ . So other elements that map to  $h_i(e)$  must have contributed  $\geq \epsilon \cdot Q$ .
  - Let  $Z_i$  be the value of  $C_{i, h_i(e)}$  that was added by other elements
  - $h_i$  chosen from 2-universal hash family then probability that another element  $f$  is mapped to  $h_i(e)$  is  $\leq 1/\ell$ .
  - Thus  $\mathbb{E}[Z_i] \leq Q/\ell$ . By Markov:

$$\Pr[Z_i \geq \epsilon \cdot Q] \leq \frac{\mathbb{E}[Z_i]}{\epsilon \cdot Q} \leq \frac{1}{\epsilon \ell}$$

- Hash functions  $h_i$  chosen independently  $\Rightarrow$

$$\Pr \left[ \min_{1 \leq i \leq k} Z_i \geq \epsilon \cdot Q \right] \leq \left( \frac{1}{\epsilon \ell} \right)^k = \prod \Pr[Z_i]$$

## Weighted heavy hitters - analysis

We have

$$\Pr \left[ \min_{1 \leq i \leq k} Z_i \geq \epsilon \cdot Q \right] \leq \left( \frac{1}{\epsilon \ell} \right)^k$$

## Weighted heavy hitters - analysis

We have

$$\Pr \left[ \min_{1 \leq i \leq k} Z_i \geq \epsilon \cdot Q \right] \leq \left( \frac{1}{\epsilon \ell} \right)^k \leq \delta$$

- Setting  $\ell = 2/\epsilon$  and  $k = \log(\delta)$  we get that probability above  $\leq \delta$ .

## Weighted heavy hitters - analysis

We have

$$\Pr \left[ \min_{1 \leq i \leq k} Z_i \geq \epsilon \cdot Q \right] \leq \left( \frac{1}{\epsilon \ell} \right)^k$$

- Setting  $\ell = 2/\epsilon$  and  $k = \log^{y_\delta}(\frac{1}{\delta})$  we get that probability above  $\leq \delta$ .
- Space requirement for counters  $O(1/\epsilon \cdot \log(1/\delta))$

$Q \cdot k$  table

## Weighted heavy hitters - analysis

We have

$$\Pr \left[ \min_{1 \leq i \leq k} Z_i \geq \epsilon \cdot Q \right] \leq \left( \frac{1}{\epsilon \ell} \right)^k$$

- Setting  $\ell = 2/\epsilon$  and  $k = \log(\delta)$  we get that probability above  $\leq \delta$ .
- Space requirement for counters  $O(1/\epsilon \cdot \log(1/\delta)) \cdot \log(N)$
- Space required to store all hash functions and evaluation time  $O(k \cdot \ell)$

# Distinct Elements

## Example (Distinct elements)

- **Input stream:**  $a_1, \dots, a_N$  be integers from  $[0, 2^b - 1]$ .  $m := 2^b$
- **Task:** maintain current # of distinct elements  $D$  we have seen so far

true task: maintain approx. number of distinct elements

input: parameter  $\epsilon$  (approx parameter)

goal: output  $Y \in \mathbb{N}$  s.t.

$$(1-\epsilon)D \leq Y \leq (1+\epsilon)D$$



# Distinct Elements

## Example (Distinct elements)

- **Input stream:**  $a_1, \dots, a_N$  be integers from  $[0, 2^b - 1]$ .  $m := 2^b$
- **Task:** maintain current # of distinct elements  $D$  we have seen so far

**Idea!** Use strongly 2-universal hash function!

# Distinct Elements

## Example (Distinct elements)

- **Input stream:**  $a_1, \dots, a_N$  be integers from  $[0, 2^b - 1]$ .  $m := 2^b$
- **Task:** maintain current # of distinct elements  $D$  we have seen so far

Use strongly 2-universal hash function!

- Take strongly 2-universal hash function  $h : [0, m - 1] \rightarrow [0, m^3]$ .

# Distinct Elements

## Example (Distinct elements)

- **Input stream:**  $a_1, \dots, a_N$  be integers from  $[0, 2^b - 1]$ .  $m := 2^b$
- **Task:** maintain current # of distinct elements  $D$  we have seen so far

Use strongly 2-universal hash function!

- Take strongly 2-universal hash function  $h : [0, m - 1] \rightarrow [0, m^3]$ .
- From previous lecture, w.h.p. no collisions!

# Distinct Elements

## Example (Distinct elements)

- **Input stream:**  $a_1, \dots, a_N$  be integers from  $[0, 2^b - 1]$ .  $m := 2^b$
- **Task:** maintain current # of distinct elements  $D$  we have seen so far

Use strongly 2-universal hash function!

- Take strongly 2-universal hash function  $h : [0, m - 1] \rightarrow [0, m^3]$ .
- From previous lecture, w.h.p. no collisions!
- Suppose there are  $D$  distinct elements  $b_1, \dots, b_D$

# Distinct Elements

## Example (Distinct elements)

- **Input stream:**  $a_1, \dots, a_N$  be integers from  $[0, 2^b - 1]$ .  $m := 2^b$
- **Task:** maintain current # of distinct elements  $D$  we have seen so far

Use strongly 2-universal hash function!

- Take strongly 2-universal hash function  $h : [0, m - 1] \rightarrow [0, m^3]$ .
- From previous lecture, w.h.p. no collisions!
- Suppose there are  $D$  distinct elements  $b_1, \dots, b_D$ 
  - If the  $D$  hash values  $h(b_1), \dots, h(b_D)$  are *evenly distributed* in  $[0, m^3]$ , then  $t^{\text{th}}$  smallest hash value should be close to  $\frac{tm^3}{D}$ .

# Distinct Elements

## Example (Distinct elements)

- **Input stream:**  $a_1, \dots, a_N$  be integers from  $[0, 2^b - 1]$ .  $m := 2^b$
- **Task:** maintain current  $\#$  of distinct elements  $D$  we have seen so far

Use strongly 2-universal hash function!

- Take strongly 2-universal hash function  $h : [0, m - 1] \rightarrow [0, m^3]$ .
- From previous lecture, w.h.p. no collisions!
- Suppose there are  $D$  distinct elements  $b_1, \dots, b_D$ 
  - If the  $D$  hash values  $h(b_1), \dots, h(b_D)$  are evenly distributed in  $[0, m^3]$ , then  $t^{\text{th}}$  smallest hash value should be close to  $\frac{tm^3}{D}$ .
  - If we know that  $t^{\text{th}}$  smallest value is  $T$ , then  $T \approx \frac{tm^3}{D} \Rightarrow D \approx \frac{tm^3}{T}$

## Distinct Elements - algorithm

- Choose a random hash function  $h$  from strongly 2-universal hash family
- For each item  $a_i$  in the stream:
  - Compute  $h(a_i)$
  - update list that stores the  $t$  smallest hash values
  - After all data has read, let  $T$  be  $t^{\text{th}}$  smallest hash value in data stream.

$$\text{Return } Y = \frac{tm^3}{T}$$

in perfect world  
this would be  $D$

# Distinct Elements Analysis

- What are our space requirements?



# Distinct Elements Analysis

- What are our space requirements?
  - Not going to store the whole hash table, only store hash function  $h$  and  $t$  numbers (the  $t$  smallest values we have seen)

# Distinct Elements Analysis

- What are our space requirements?
  - Not going to store the whole hash table, only store hash function  $h$  and  $t$  numbers (the  $t$  smallest values we have seen)
  - Need to find good value of  $t$  for have high probability of success

*(and hope that  $t$  is small)*

# Distinct Elements Analysis

- What are our space requirements?
  - Not going to store the whole hash table, only store hash function  $h$  and  $t$  numbers (the  $t$  smallest values we have seen)
  - Need to find good value of  $t$  for have high probability of success

## Theorem

Setting  $t = O(1/\epsilon^2)$  we have that

$$(1 - \epsilon) \cdot D \leq Y \leq (1 + \epsilon) \cdot D$$

with constant probability.

↑ our output

# Distinct Elements Analysis

## Theorem

Setting  $t = O(1/\epsilon^2)$  we have that  $Y = \frac{tm^3}{T}$  satisfies:

$$(1 - \epsilon) \cdot D \leq Y \leq (1 + \epsilon) \cdot D$$

*with constant probability.*

# Distinct Elements Analysis

## Theorem

Setting  $t = O(1/\epsilon^2)$  we have that  $Y = \frac{tm^3}{T}$  satisfies:

$$(1 - \epsilon) \cdot D \leq Y \leq (1 + \epsilon) \cdot D$$

with constant probability.

Upper Bound:  $\Pr[Y > (1 + \epsilon) \cdot D]$  is small

# Distinct Elements Analysis

## Theorem

Setting  $t = O(1/\epsilon^2)$  we have that  $Y = \frac{tm^3}{T}$  satisfies:

$$(1 - \epsilon) \cdot D \leq Y \leq (1 + \epsilon) \cdot D$$

with constant probability.

Upper Bound:  $\Pr[Y > (1 + \epsilon) \cdot D]$

- $Y > (1 + \epsilon) \cdot D \Rightarrow T < \frac{tm^3}{(1 + \epsilon) \cdot D} \leq \frac{(1 - \epsilon/2) \cdot tm^3}{D}$   
 $\hookrightarrow \frac{1}{1 + \epsilon} \leq 1 - \frac{\epsilon}{2}$  when  $\epsilon \leq 1$

$$(1 + \epsilon)D < Y = \frac{tm^3}{T} \Rightarrow T < \frac{tm^3}{(1 + \epsilon)D}$$

# Distinct Elements Analysis

## Theorem

Setting  $t = O(1/\epsilon^2)$  we have that  $Y = \frac{tm^3}{T}$  satisfies:

$$(1 - \epsilon) \cdot D \leq Y \leq (1 + \epsilon) \cdot D$$

with constant probability.

Upper Bound:  $\Pr[Y > (1 + \epsilon) \cdot D]$

- $Y > (1 + \epsilon) \cdot D \Rightarrow T < \frac{tm^3}{(1 + \epsilon) \cdot D} \leq \frac{(1 - \epsilon/2) \cdot tm^3}{D}$
- At least  $t$  hash values smaller than  $\frac{(1 - \epsilon/2) \cdot tm^3}{D}$

by definition of  $T$ :  $T =$  value of  $t^{\text{th}}$  smallest hash value

# Distinct Elements Analysis

## Theorem

Setting  $t = O(1/\epsilon^2)$  we have that  $Y = \frac{tm^3}{T}$  satisfies:

$$(1 - \epsilon) \cdot D \leq Y \leq (1 + \epsilon) \cdot D$$

with constant probability.

Upper Bound:  $\Pr[Y > (1 + \epsilon) \cdot D]$

- $Y > (1 + \epsilon) \cdot D \Rightarrow T < \frac{tm^3}{(1 + \epsilon) \cdot D} \leq \frac{(1 - \epsilon/2) \cdot tm^3}{D}$

- At least  $t$  hash values smaller than  $\frac{(1 - \epsilon/2) \cdot tm^3}{D}$

- Random variable  $X_i = \begin{cases} 1, & \text{if } h(a_i) \leq \frac{(1 - \epsilon/2) \cdot tm^3}{D} \\ 0, & \text{otherwise} \end{cases}$

*small*

*hash value of  $a_i$*



## Distinct Elements Analysis

Upper Bound:  $\Pr[Y > (1 + \epsilon) \cdot D]$

- Random variable  $X_i = \begin{cases} 1, & \text{if } h(a_i) \leq \frac{(1 - \epsilon/2) \cdot tm^3}{D} \\ 0, & \text{otherwise} \end{cases}$

## Distinct Elements Analysis

Upper Bound:  $\Pr[Y > (1 + \epsilon) \cdot D]$

- Random variable  $X_i = \begin{cases} 1, & \text{if } h(a_i) \leq \frac{(1 - \epsilon/2) \cdot tm^3}{D} \\ 0, & \text{otherwise} \end{cases}$

- $\mathbb{E}[X_i] = \Pr \left[ \underline{h(a_i)} \leq \frac{(1 - \epsilon/2) \cdot tm^3}{\underline{D}} \right] = \frac{(1 - \epsilon/2) \cdot t}{D}$

Each  $h(a_i)$  uniformly random in  $[0, m^3]$ .

$\rightarrow \Pr[X_i = 1] \cdot 1 + \Pr[X_i = 0] \cdot 0$

## Distinct Elements Analysis

Upper Bound:  $\Pr[Y > (1 + \epsilon) \cdot D]$

- Random variable  $X_i = \begin{cases} 1, & \text{if } h(a_i) \leq \frac{(1 - \epsilon/2) \cdot tm^3}{D} \\ 0, & \text{otherwise} \end{cases}$

- $\mathbb{E}[X_i] = \Pr \left[ h(a_i) \leq \frac{(1 - \epsilon/2) \cdot tm^3}{D} \right] = \frac{(1 - \epsilon/2) \cdot t}{D}$

Each  $h(a_i)$  uniformly random in  $[0, m^3]$ .

- If there are  $D$  distinct elements,

$$\mathbb{E} \left[ \# \text{ elements with hash value} \leq \frac{(1 - \epsilon/2) \cdot tm^3}{D} \right] \leq t(1 - \epsilon/2)$$

$$X = \sum X_i$$

linearity of expectation

## Distinct Elements Analysis

Upper Bound:  $\Pr[Y > (1 + \epsilon) \cdot D]$

- Random variable  $X_i = \begin{cases} 1, & \text{if } h(a_i) \leq \frac{(1 - \epsilon/2) \cdot tm^3}{D} \\ 0, & \text{otherwise} \end{cases}$

- $\mathbb{E}[X_i] = \Pr \left[ h(a_i) \leq \frac{(1 - \epsilon/2) \cdot tm^3}{D} \right] = \frac{(1 - \epsilon/2) \cdot t}{D}$

Each  $h(a_i)$  uniformly random in  $[0, m^3]$ .

- If there are  $D$  distinct elements,

$$\mathbb{E} \left[ \# \text{ elements with hash value} \leq \frac{(1 - \epsilon/2) \cdot tm^3}{D} \right] \leq t(1 - \epsilon/2)$$

- but we assumed we have at least  $t$  such elements! Now need to show that this cannot happen with high probability

(concentration of measure)

$t$  far from expectation

## Distinct Elements Analysis

Upper Bound:  $\Pr[Y > (1 + \epsilon) \cdot D]$

*↷ # elements with small hash value*

- If there are  $D$  distinct elements, let  $X = \sum_{i=1}^D X_i$

$$\mathbb{E}[X] \leq t(1 - \epsilon/2)$$

*previous slide*

## Distinct Elements Analysis

Upper Bound:  $\Pr[Y > (1 + \epsilon) \cdot D]$

- If there are  $D$  distinct elements, let  $X = \sum_{i=1}^D X_i$

$$\mathbb{E}[X] \leq t(1 - \epsilon/2)$$

- Probability we will see  $\geq t$  elements smaller than  $\frac{(1 - \epsilon/2) \cdot tm^3}{D}$

## Distinct Elements Analysis

Upper Bound:  $\Pr[Y > (1 + \epsilon) \cdot D]$

- If there are  $D$  distinct elements, let  $X = \sum_{i=1}^D X_i$

$$\mathbb{E}[X] \leq t(1 - \epsilon/2)$$

- Probability we will see  $\geq t$  elements smaller than  $\frac{(1 - \epsilon/2) \cdot tm^3}{D}$ 
  - $\text{Var}[X] = \sum_{i=1}^D \text{Var}[X_i]$  (pairwise independence)

*we will see this in next lecture!*

## Distinct Elements Analysis

Upper Bound:  $\Pr[Y > (1 + \epsilon) \cdot D]$

- If there are  $D$  distinct elements, let  $X = \sum_{i=1}^D X_i$

$$\mathbb{E}[X] \leq t(1 - \epsilon/2)$$

- Probability we will see  $\geq t$  elements smaller than  $\frac{(1 - \epsilon/2) \cdot tm^3}{D}$ 
  - $\text{Var}[X] = \sum_{i=1}^D \text{Var}[X_i]$  (pairwise independence)
  - $\text{Var}[X_i] = \mathbb{E}[(X_i - \mathbb{E}[X_i])^2] = \mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2 \leq \mathbb{E}[X_i]$  (indicator variable)

definition of variance

simple computation

$$\therefore \text{Var}[X] = \sum_{i=1}^D \text{Var}[X_i] \leq \sum_{i=1}^D \mathbb{E}[X_i] = \mathbb{E}[X] \leq t(1 - \epsilon/2)$$

lin. expectation



## Distinct Elements Analysis

Upper Bound:  $\Pr[Y > (1 + \epsilon) \cdot D] \leq \Pr[X > t] \leq \frac{4}{\epsilon^2 t}$

- If there are  $D$  distinct elements, let  $X = \sum_{i=1}^D X_i$

$$\mathbb{E}[X] \leq t(1 - \epsilon/2)$$

- Probability we will see  $\geq t$  elements smaller than  $\frac{(1 - \epsilon/2) \cdot tm^3}{D}$ 
  - $\text{Var}[X] = \sum_{i=1}^D \text{Var}[X_i]$  (pairwise independence)
  - $\text{Var}[X_i] = \mathbb{E}[(X_i - \mathbb{E}[X_i])^2] = \mathbb{E}[X_i^2] - \mathbb{E}[X_i]^2 \leq \mathbb{E}[X_i]$  (indicator variable)
  - Chebyshev's inequality:

$$\begin{aligned} \Pr[X > t] &= \Pr[X > \underbrace{t \cdot (1 - \epsilon/2)}_{\mathbb{E}[X]} + \epsilon \cdot t/2] \\ &\leq \Pr[|X - \mathbb{E}[X]| > \epsilon \cdot t/2] \leq \frac{4 \cdot \text{Var}[X]}{\epsilon^2 t^2} \leq \frac{4}{\epsilon^2 t} \end{aligned}$$

*because of absolute value* (orange arrow pointing to the absolute value in the inequality)

*Chebyshev* (green arrow pointing to the inequality step)

*plugging in  $\text{Var}[X] \leq t(1 - \epsilon/2)$*  (blue arrow pointing to the  $\text{Var}[X]$  term)

## Distinct Elements Analysis

Lower Bound:  $\Pr[Y < (1 - \epsilon) \cdot D]$ . *small*

Similar calculation as previous slide.<sup>2</sup>

Practice problem: do this part of the proof.

---

<sup>2</sup>replacing  $1 - \epsilon$  by  $1 + \epsilon$  and using Chebyshev

## Distinct Elements Analysis

Lower Bound:  $\Pr[Y < (1 - \epsilon) \cdot D]$ .

Similar calculation as previous slide.<sup>2</sup>

Practice problem: do this part of the proof.

- $\Pr[Y > (1 + \epsilon) \cdot D] \leq \frac{4}{\epsilon^2 t}$
- $\Pr[Y < (1 - \epsilon) \cdot D] \leq \frac{4}{\epsilon^2 t}$

} bad events  
(small prob.)

---

<sup>2</sup>replacing  $1 - \epsilon$  by  $1 + \epsilon$  and using Chebyshev

## Distinct Elements Analysis

Lower Bound:  $\Pr[Y < (1 - \epsilon) \cdot D]$ .

Similar calculation as previous slide.<sup>2</sup>

Practice problem: do this part of the proof.

- $\Pr[Y > (1 + \epsilon) \cdot D] \leq \frac{4}{\epsilon^2 t}$
- $\Pr[Y < (1 - \epsilon) \cdot D] \leq \frac{4}{\epsilon^2 t}$
- Setting  $t = 24/\epsilon^2$  gives us

$$\Pr[\underbrace{(1 - \epsilon) \cdot D \leq Y \leq (1 + \epsilon) \cdot D}_{\text{Alg. correct}}] \geq 1 - \frac{8}{\epsilon^2 t} = 2/3$$
$$\underbrace{1 - \Pr[Y < (1 - \epsilon) \cdot D] - \Pr[Y > (1 + \epsilon) \cdot D]}_{\text{bad events}}$$

---

<sup>2</sup>replacing  $1 - \epsilon$  by  $1 + \epsilon$  and using Chebyshev

## Distinct Elements Analysis

Lower Bound:  $\Pr[Y < (1 - \epsilon) \cdot D]$ .

Similar calculation as previous slide.<sup>2</sup>

Practice problem: do this part of the proof.

- $\Pr[Y > (1 + \epsilon) \cdot D] \leq \frac{4}{\epsilon^2 t}$
- $\Pr[Y < (1 - \epsilon) \cdot D] \leq \frac{4}{\epsilon^2 t}$
- Setting  $t = 24/\epsilon^2$  gives us

$$\Pr[(1 - \epsilon) \cdot D \leq Y \leq (1 + \epsilon) \cdot D] \geq 1 - \frac{8}{\epsilon^2 t} = 2/3$$

Homework problem # 6: how can we make the success probability much higher?

---

<sup>2</sup>replacing  $1 - \epsilon$  by  $1 + \epsilon$  and using Chebyshev

## Space requirements and running time

- Total space used:  $O\left(\frac{1}{\epsilon^2} \log m\right)$  bits

## Space requirements and running time

- Total space used:  $O\left(\frac{1}{\epsilon^2} \log m\right)$  bits
  - we stored  $O(1/\epsilon^2)$  hash values each of  $\log(m)$  bits
  - hash function only requires  $O(\log m)$  bits to store.

## Space requirements and running time

- Total space used:  $O\left(\frac{1}{\epsilon^2} \log m\right)$  bits
  - we stored  $O(1/\epsilon^2)$  hash values each of  $\log(m)$  bits
  - hash function only requires  $O(\log m)$  bits to store.
- Running time per operation:  $O(\log(m) + 1/\epsilon^2)$  steps



## Space requirements and running time

- Total space used:  $O\left(\frac{1}{\epsilon^2} \log m\right)$  bits
  - we stored  $O(1/\epsilon^2)$  hash values each of  $\log(m)$  bits
  - hash function only requires  $O(\log m)$  bits to store.
- Running time per operation:  $O(\log(m) + 1/\epsilon^2)$  steps
  - compute hash in  $O(\log m)$  time
  - Since we keep track of  $O(1/\epsilon^2)$  elements, and need to update the list, this takes  $O(1/\epsilon^2)$  time (though there are smarter ways)

# Acknowledgement

- Lecture based largely on Lap Chi's notes and David Woodruff's notes.
- See Lap Chi's notes at  
<https://cs.uwaterloo.ca/~lapchi/cs466/notes/L05.pdf>
- See David's notes at  
<https://www.cs.cmu.edu/~15451-s20/lectures/lec6.pdf>