DS 503: Advanced Data Analytics

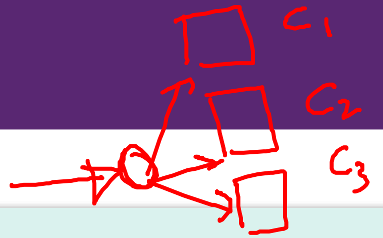
# Lecture 9: Streams

Dr. Gagan Raj Gupta

# More algorithms for streams:

- **More algorithms for streams:**
  - **(1) Counting distinct elements: KMV**
    - *Number of distinct elements* seen so far in the stream
  - **(2) Frequent Items: MG, Space Saving, Count-Min**
    - Estimate the *k most frequent elements* in the stream
  - **(3) Estimating moments: AMS method**
    - Estimate $\boldsymbol{k}^{th}$ *moment of element frequencies*
  - **(4) Histograms/Quantiles: KLL** ( Not in the Tierce)
    - Estimate what *% of elements are below a threshold*

# 1: Counting Distinct Elements




$c_1 + c_2 + c_3$

- **Problem:**
- Data stream consists of a universe of elements chosen from a set of size ***N***
- Maintain a count of the number of distinct elements seen so far
- **Obvious approach:**
  - That is, keep a hash table of all the distinct elements seen so far
- It will take up a lot of storage

- **How many different words are found among the Web pages being crawled at a site?**
  - Unusually low or high numbers could indicate artificial pages (spam?)
- **How many different Web pages does each customer request in a week?**
- **How many distinct products have we sold in the last week?**
- **Number of unique users in the system**
- **Number of users affected by an outage**
- **Number of users using a particular service**

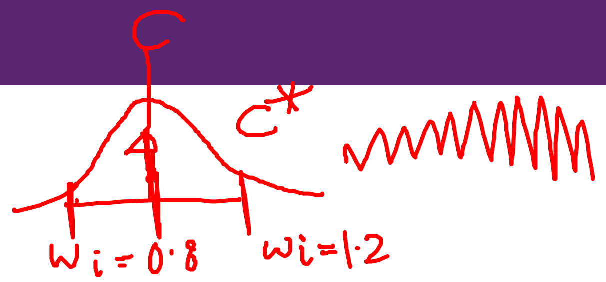
Allow for **intersections and unions**
How many products/users in both categories or at least one?

# Naïve approach to distinct counting

- How many distinct users saw a news clip?
- Database query: select count(distinct user) from very_large_stream
- Execution (Naive):
  - Insert each user into a hash table
  - Compute the number of keys in the table
- Cost:
  - N = 1 million, 64 bits / user
  - Hash table size per website/issue: 16 MB
  - 100 websites/issues/ day ⇒ 1.6 GB / day
  - Over 1 month ⇒ 50 GB / month
  - With dimension breakdowns (user demographics), much higher costs
- Uniform sampling will be misleading, if some user watched it many times.

# Using Small storage

$\min(f(c^*))$




- **Real problem:** **What if we do not have space to maintain the set of elements seen so far?**
- **Estimate the count in an unbiased way (E[estimate]=actual)**
- **Accept that the count may have a little error, but limit the probability that the error is large**
- **Several algorithms have been proposed for this problem, all of them try to make use of (one or more) hash functions and design an estimator that is unbiased and has low variance.**

# KMV Sketch

**Data (sequence of items):** $x_1, x_2, x_3, \dots, x_t$

**Universal hash function 'h':**$x_i \rightarrow Z_i \sim Uniform(0,1)$

- Results in **N** distinct values (eg. $2^{32}$ $or$ $2^{64}$ )
- Distribution is known and depends only on the desired value N

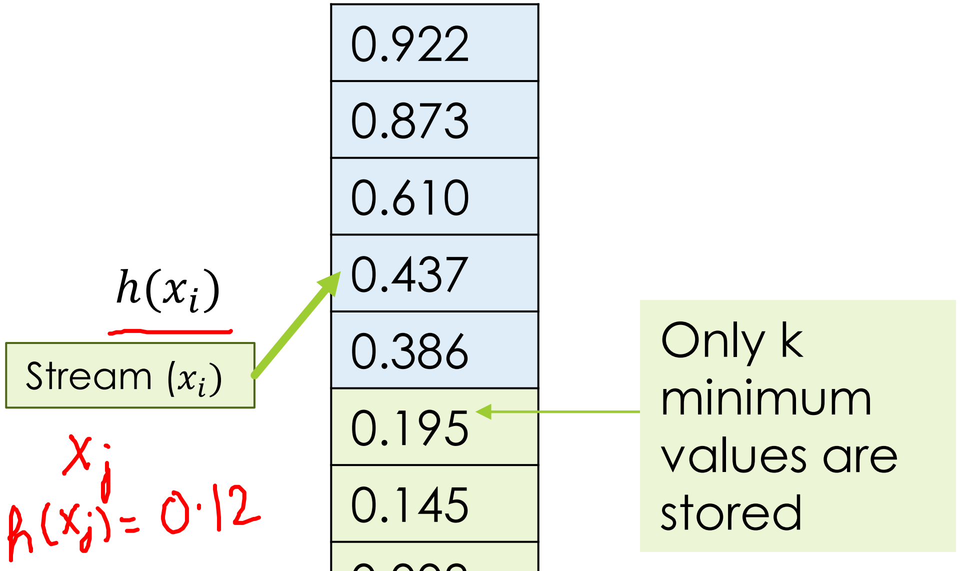
**Summary**: Maintain k minimum values of h($x_k$);

$$Z_1 \leq Z_2 \leq Z_3 \dots \leq Z_k$$

**Estimate**: $N_{hat} = \frac{k-1}{Z_k}$ ✓

$\frac{3-1}{0.145} = \frac{2}{0.145} \sim 12$

**Error**: std dev $\sigma(N_{hat}) = \frac{N_{hat}}{\sqrt{k-2}}$



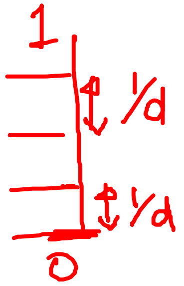

Mergeable:
Distributed computations, over time and space are allowed
Operations**:**
Union, Intersection, Set Differences
Example: Multiple dimensions can be implemented as intersections, but errors increase

# Intuition

$x = [1 \cdots n] \longrightarrow y = \frac{1}{x} \quad [\frac{1}{n} \cdots 1]$

0

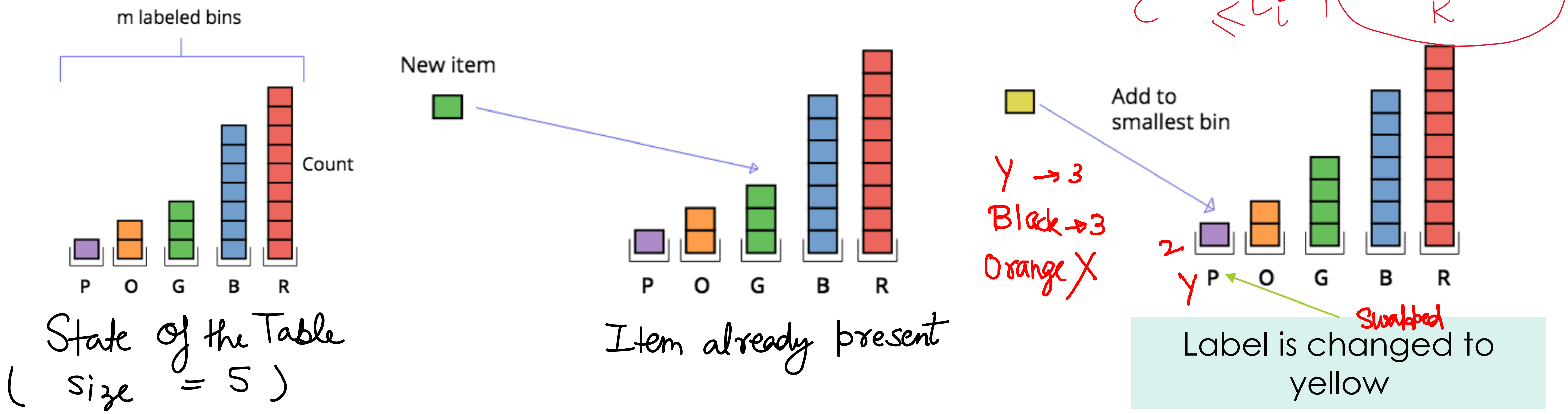
- Hash function maps the element to any of the n values with equal probability
- After d items, the spacing between them is roughly 1/d
- The value of the minimum is close to 1/d
- Thus estimate of d = 1/(min)
- But, there may be a lot of variation in this. We may get a small values, which may disturb the estimate
- Keep many values (k), and the estimate is k/(kth minimum) $\sim$ 1000
  - Because we have computed the minimum, hence lost one degree of freedom
  - To make it unbiased, we have to use $\frac{k-1}{Z_k}$



# Problem 2: Top-K/Frequent Items/Heavy Hitters

- **Monitoring:** Detect a DDoS attack on a destination
- **Analytics:** Find the heaviest users, top websites, most-significant failure reasons etc.
- **Problem:** Given a stream of (key, increment) pairs, find the keys with the largest total sums
- **MG and Space Saving techniques:** Maintain only fixed number of bins and delete infrequent items
  + increments.
- **Count Min:** Hash into multiple bins and report the minimum value when queried
  + - increments

$$c^{est} \le c_i^{*\,\text{true}} + \frac{\text{Total Counts}}{K}$$



# MG Sketch

- GUARANTEE: Given parameter $\epsilon$ , stores $\frac{1}{\epsilon}$ items and count/weights. Guarantees that returns the count/weights are within $\epsilon$ of the total count/weight, C, over all items
- INITIALIZE: Create empty table, T, with space k= $\frac{1}{\epsilon}$
- UPDATE:

weight

If $i \in T, c_i = c_i + 1$ (increment count)
Else T = T ∪ {i} ; $c_i = 1$ (insert into the Table)

weight

If Size (T) > k:

min weight

$\forall\, i \in T, c_i = c_i - 1$ (decrement counts)
If $c_i = 0$, T = T \ i (remove i from the table)

a b c d e
5 + 5 + 5 + 5 + 5

$\frac{25}{4}$ ~ 6

5 5 5 5 5 . . 5

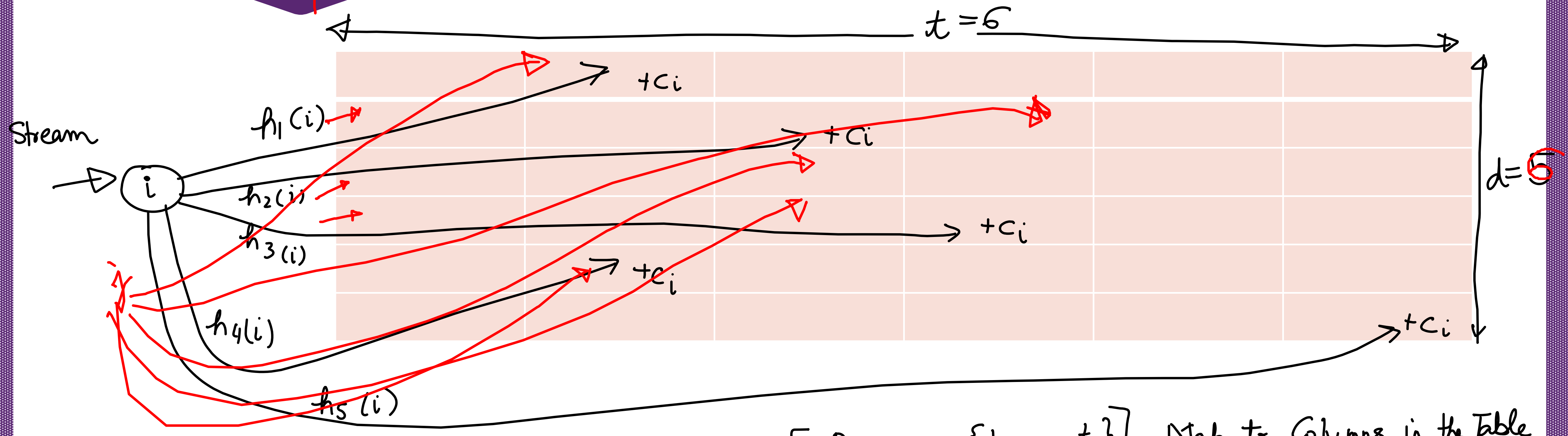
Error Bound Proof Sketch:

a. We decrement counts only when the table is full ✓
b. A decrement operation reduces each of the k counts by 1 (min count)
c. There are C total elements seen in the stream (W)
d. Maximum number of decrement operations possible: C/k
e. Maximum error in the estimate: C/k = $C\epsilon$
f. Works extremely well when the distribution is skewed (e.g. Words in English, Pop of city)

a a a . . a
b b b . .
c c . .
d . .
e e e e e

5 4 3 2
5 4 3 2
5 4 3 2
5 4 3 2
e 1
0
0

# Count-Min Sketch




Eg:- $d=5$; $h_1 \dots h_5$ are independent hash functions. [ Range :- $\{1, \dots t\}$ ] Map to Columns in the Table

UPDATE :- Add the counts to the Table Entries for each hash function. $T[h(i)] = T[h(i)] + c_i$

QUERY :- For a given item 'i', its estimated Count/Weight is

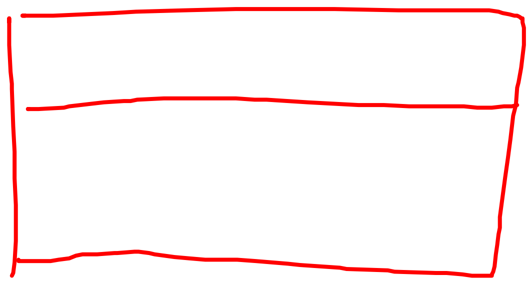
$$c_i^* = \min_{h(i)} T[h_i]$$

Guarantee to be within $\frac{c}{t}$ with probability $\left(1 - \frac{1}{d}\right)$

Note :-
Counts can be negative!

Each hf is responsible for a row

↳ maps an element to the Column



Updates the count of (row, col) – $+c_i$

Query Cells for each hf.

10, 12, 14

↑

min

# Bonus Question: 10 marks for the course

- Read relevant sections of Chapter 3 of SSBD book (http://dimacs.rutgers.edu/~graham/ssbd/ssbd3.pdf)
- Can read any other reference you like
- Write the proofs for the correctness and error bounds on MG, Space Saving, Count-Min Sketch in your own words.
- Proofs must be neat, Preferably type it in LaTeX.
  - Use https://www.overleaf.com/
- First two correct submissions will receive the credit.

# Problem 3: Computing Moments

- **Suppose a stream has elements chosen from a set *A* of *N* values**
- **Let $m_i$ be the number of times value *i* occurs in the stream**
- **The $k^{th}$ (frequency) *moment* is :** $\sum_{i \in A}(m_i)^k$
  - This is the same way as moments are defined in statistics. But there one typically "centers" the moment by subtracting the mean
- **0th moment =** number of distinct elements
  - Problem we considered (solved using KMV)
- **1st moment =** count of the numbers of elements = length of the stream
  - Easy to compute, so not particularly useful
- **2nd moment = *surprise number S* =** a measure of how uneven the distribution is
  - Very useful

a a b a

$a \to 3$

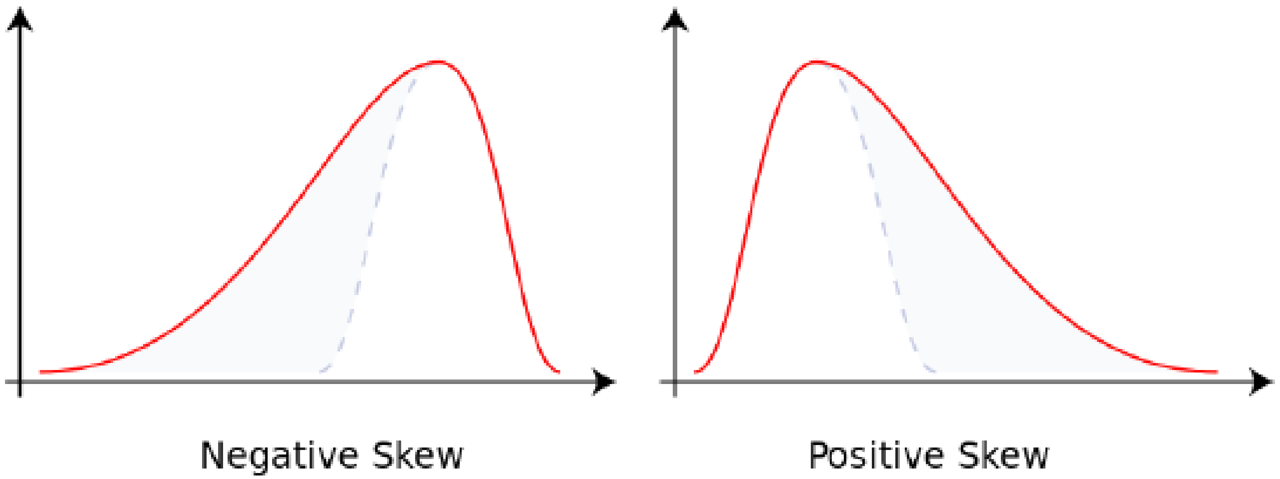
$b \to 1$

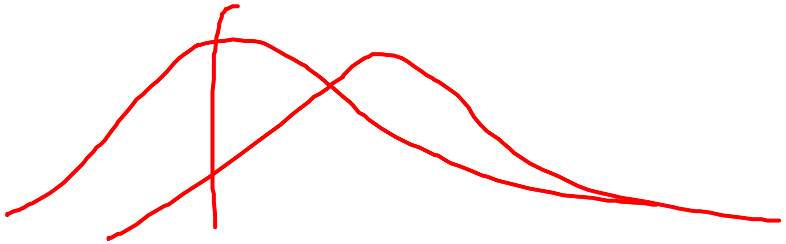
$2^{nd} \to 3^2 + 1^2$

$= 10$

# Moments

- Third Moment is Skew:



- Fourth Moment: Kurtosis
  - Peakedness (width of peak), tail weight, and lack of shoulders (distribution primarily peak and tails, not in between).

# Example: Surprise Number

- **Measure of how uneven the distribution is**
- **Stream of length 100 (sum of first moment)**
  - **11 distinct values**
- Item **counts mi**: **10, 9, 9, 9, 9, 9, 9, 9, 9, 9, 9**

  **Surprise *S* = 910**
- Item **counts mi** : **90, 1, 1, 1, 1, 1, 1, 1 ,1, 1, 1**

  **Surprise *S* = 8,110**

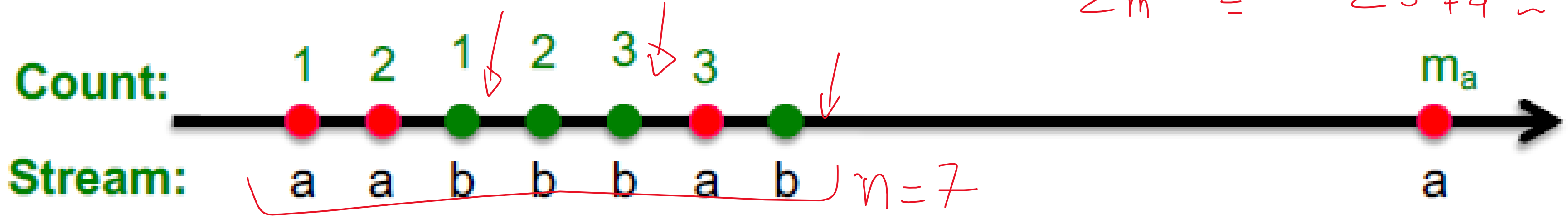
# AMS Method

- **AMS method works for all moments**
- **Gives an unbiased estimate**
- **We will just concentrate on the 2nd moment**
  - Can be generalized
- **We pick and keep track of many variables *X*:**
  - For each variable ***X*** we store ***X.el*** and ***X.val***
    - ***X.el*** corresponds to the item ***i***
    - ***X.val*** corresponds to the **count** $\boldsymbol{m_i}$ of item ***i***
- Note this requires a count in main memory, so number of ***X***s is limited
- **Our goal is to compute** $\boldsymbol{S} = \sum_i m_i^2$

# One Random Variable (X)

- **How to set *X.val* and *X.el*?**
  - Assume stream has length ***n*** (we relax this later)
  - Pick some random time ***t*** (***t*<*n***) to start, so that any time is equally likely
  - Let at time ***t*** the stream have item ***i***. ***We set X.el = i***
  - Then we maintain count ***c*** (***X.val = c***) of the number of ***i*s** in the stream starting from the chosen time ***t***
- **Then the estimate of the 2nd moment ($\sum_i m_i^2$) is:**
  - $S = f(X) = n\,(2 \cdot c - 1)$
  - Note, we will keep track of multiple **X**s, $(X_1, X_2, \ldots, X_k)$ and our final estimate will be $S = \frac{1}{k}\sum_j^k f(X_j)$

# Expectation Analysis

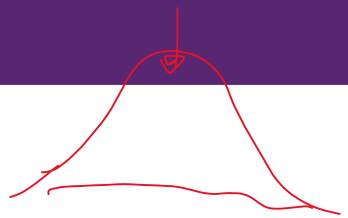



$n=7$

$c_3 = 4$

$\Sigma m^2 = \Sigma 3^2 + 4^2 \sim 25$

$f(x) = 7(2 \cdot 4 - 1) = 49$ ✓



$n = 7$

$c_5 = 2$

$f(x) = 7(2 \cdot 2 - 1)$

$= 7(3)$

$= 21$

- **2nd moment is $S = \sum_i m_i^2$**
- $c_t$ … number of times item at time ***t*** appears from time **t** onwards

$(c_1 = m_a, c_2 = m_a - 1, c_3 = m_b)$

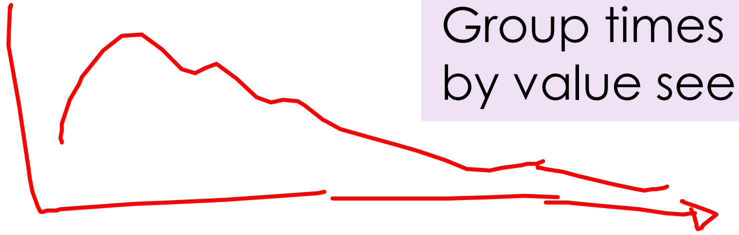
$m_i$ … total count of item i in the stream
(we are assuming stream has length n)

- $E[f(X)] = \frac{1}{n}\sum_{t=1}^{n} n(2c_t - 1) \;\; = \frac{1}{n}\sum_i n(1 + 3 + 5 + \cdots + 2m_i - 1)$

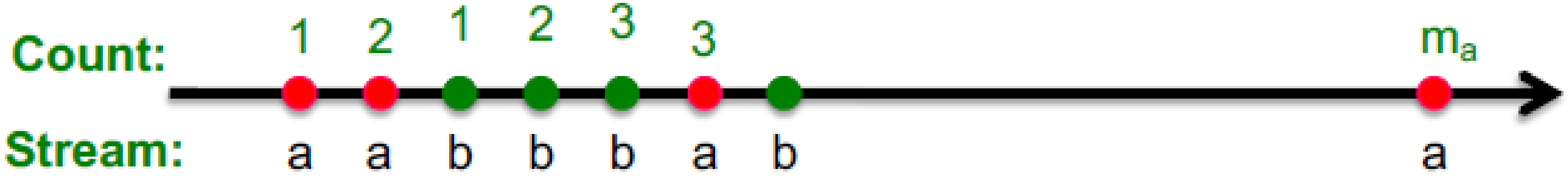
Group times
by value seen

Time t when
the last item i is
seen ($c_t = 1$)

Time t when
the first i is seen
($c_t = m_i$)



# Expectation Analysis




- $E[f(X)] = \frac{1}{n}\sum_{t=1}^{n} n(2c_t - 1) \ = \frac{1}{n}\sum_i n(1 + 3 + 5 + \cdots + 2m_i - 1)$
  - $1 + 3 + 5 + \cdots + 2m_i - 1 = \sum_{i=1}^{m_i}(2i - 1) = \frac{2m_i(m_i+1)}{2} - m_i = (m_i)^2$
- Then $E[f(X)] = \frac{1}{n}\sum_i n(m_i)^2$
- So, $E[f(X)] = \sum_i (m_i)^2 \ = S$
- We have the second moment (in expectation)!

# Higher-Order Moments

- **For estimating k^th^ moment we essentially use the same algorithm but change the estimate f(X):**
  - For **k=2** we used ***n* (2·c – 1)**
  - For **k=3** we use: $n(3.c^2 - 3.c + 1)$ (where **c=X.val**)
- **Why?**
  - **For k=2:** $(1 + 3 + 5 + \cdots + 2m_i - 1)$ sum to $m_i^2$
    - $\sum_{c=1}^{m}(2c-1) = \sum_{c=1}^{m} c^2 - \sum_{c=1}^{m}(c-1)^2 = m^2$
    - **Because:** $2c - 1 = c^2 - (c-1)^2$
  - **For k=3:** $c^3 - (c-1)^3 = 3c^2 - 3c + 1$
- **Generally:** Estimate f(X) = $n(c^k - (c-1)^k)$

Bonus Question

Prove by induction

$$\sum_{c=1}^{m}(3c^2 - 3c + 1) = m^3$$

# Combining Samples

- **In practice:**
  - Compute $f(X) = n(2c - 1)$ for as many variables $\boldsymbol{X}$ as you can fit in memory
  - Average them in groups
  - Take median of averages

some

- **Problem: Streams never end**
  - We assumed there was a number $\boldsymbol{n}$, the number of positions in the stream
  - Real streams go on forever, $\boldsymbol{n}$ is a variable – the number of inputs seen so far

# Never ending streams: Solution

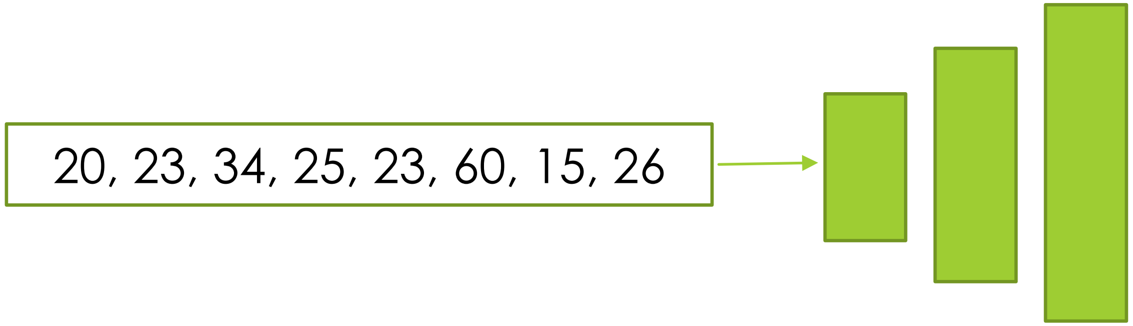
- The variables ***X*** have ***n*** as a factor – keep ***n*** separately; just hold the count in ***X***
- Suppose we can only store ***k*** counts. We must throw some ***X***s out as time goes on:
- **Objective:** Each starting time ***t*** is selected with probability ***k/n***
- **Solution: (fixed-size / reservoir sampling!)**
  - Choose the first ***k*** times for ***k*** variables
  - When the $n^{th}$ element arrives ($n > k$), choose it with probability ***k/n***
  - If you choose it, throw one of the previously stored variables **X** out, with equal probability

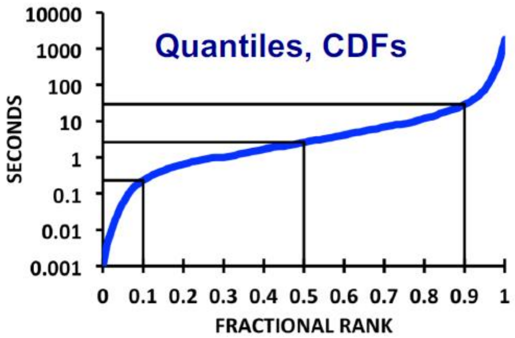
# Problem 4: Generating Quantiles

- Fundamental in any SLA monitoring system
- Compute quality of service metrics
  - 99th percentile latency / TTFB/ etc.
- Robust metrics (median, interquartile range)
  - Much more accurate than sampling methods
- Set anomaly detection thresholds

| Partial list of quantile sketches | Systems |
|---|---|
| Greenwald-Khanna (2001) | Spark |
| MRL / RANDOM (Manku et al 1999, Agarwal et al 2012) | |
| Q-digest (Shrivastava 2004) | Presto |
| T-digest (v1 Dunning 2013, v2 + Ertl 2019) | Dynatrace, Splunk, various |
| KLL (Karnin et al, 2016) | Apache Datasketches |
| DDSketch (Masson et al 2019) | Datadog |
| Moments sketch (Gan et a 2018) | Druid extension |
| Relative error streaming quantiles (Arxiv 2020) | |

**KLL Sketch:** Basic idea is to store the data in series of buffers which are compacted successively (by moving odd/even values to next level) to provide estimates of the quantiles (what fraction of values are less than x)



# Bonus Question: 5 marks

- Read sections 3 of Chapter 4 of SSBD book (http://dimacs.rutgers.edu/~graham/ssbd/ssbd4.pdf)
- Can read any other reference you like
- Write the proofs for the correctness and error bounds on KLL sketch along with examples explaining the compaction process.
- Proofs must be neat, Preferably type it in LaTeX.
  - Use https://www.overleaf.com/
- First two correct submissions will receive the credit.