Algorithms for Big Data (Fall 25)

Lecture 2

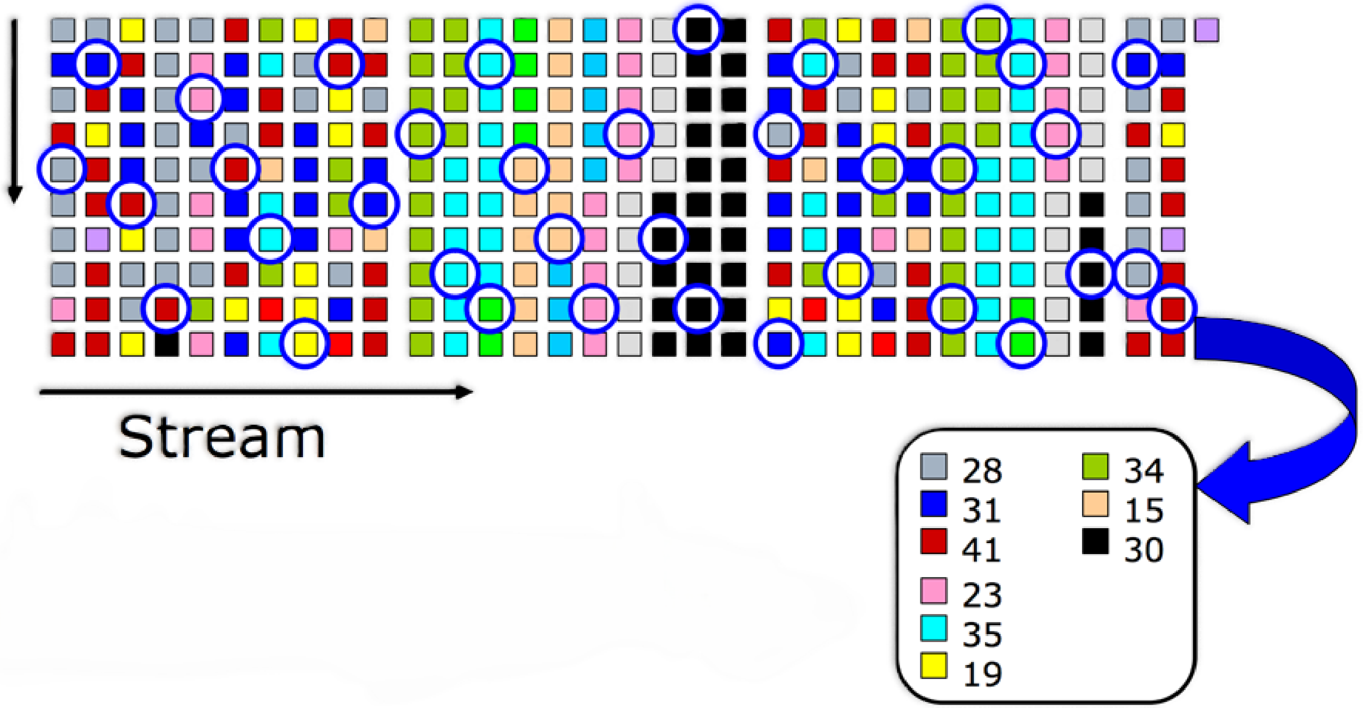
Streaming Model and Sampling

Ali Vakilian (vakilian@vt.edu)

Streaming Model

can't store the whole stream

- **Input:** m items $e_1, \cdots, e_N$ arrive **one by one**
- **Memory constraint:** only B tokens of space, with (typically) $B \ll N$
- **Goal:** compute interesting functions/statistics over the stream under limited memory



Streaming Model

can't store the whole stream

- **Input:** m items $e_1, \cdots, e_N$ arrive **one by one**
- **Memory constraint:** only B tokens of space, with (typically) $B \ll N$
- **Goal:** compute interesting functions/statistics over the stream under limited memory

Examples of streams

- A number in $[n]$
- **Network packets:** (src IP, dst IP, payload)
- **Graph stream:** each token is an edge
- **Geometric stream:** each token is a point in a feature space
- **Matrix stream:** each token is a row/column of a matrix (or an entry of a matrix)

Streaming Algorithms: Motivations

- **Storage gap:** Very large but slow media (e.g., tape and remote storage) are best for sequential access, while main memory is small but fast.

process data in **one or few passes**

Streaming Algorithms: Motivations

- **Storage gap:** Very large but slow media (e.g., tape and remote storage) are best for sequential access, while main memory is small but fast.
- **High-velocity sources:** Network switches, logs, sensors/IoT where data is flying by; raw storage is infeasible due to rate, cost, or privacy/compliance.

keep only **high-level statistics**

Streaming Algorithms: Motivations

- **Storage gap:** Very large but slow media (e.g., tape and remote storage) are best for sequential access, while main memory is small but fast.
- **High-velocity sources:** Network switches, logs, sensors/IoT where data is flying by; raw storage is infeasible due to rate, cost, or privacy/compliance.
- **Distributed data:** Data lives on many devices. Shipping everything to a central place is costly/impossible.

need **small-size** summaries

sketching techniques

Streaming Algorithms: Motivations

- **Storage gap:** Very large but slow media (e.g., tape and remote storage) are best for sequential access, while main memory is small but fast.
- **High-velocity sources:** Network switches, logs, sensors/IoT where data is flying by; raw storage is infeasible due to rate, cost, or privacy/compliance.
- **Distributed data:** Data lives on many devices. Shipping everything to a central place is costly/impossible.
- **Resource constraints:** *Often* per-item update time is required to be $O(1)$ or $O(\log n)$; memory $\ll$ data size; often running with power/latency limits.

Key Question in Streaming Algorithms

- Trade-off between **memory size**, **accuracy**, and ***number of passes***

Ideal scenario. compute the quantity of interest, over a stream of $\boldsymbol{n}$ items

- Near optimally (say $(\mathbf{1} \pm \boldsymbol{\epsilon})$**-approximation**),
- In **one pass** over the stream,
- Using $\mathbf{poly}(\mathbf{log}\, \boldsymbol{n})$ space

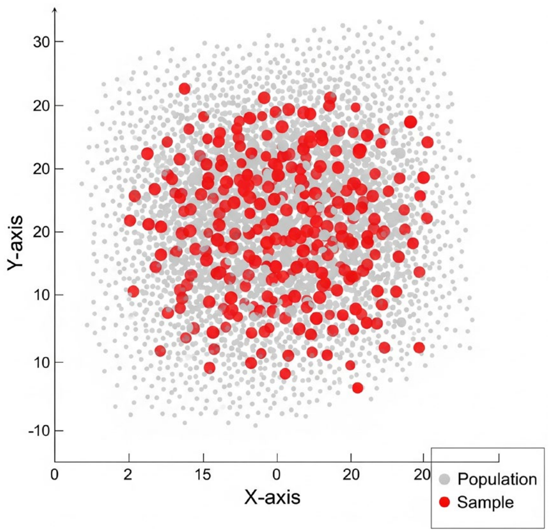
− For many, **not possible** to do in less than $O(n)$ space if one wants **exact** answer

+ **Randomization** and **approximation** leads to many interesting results

Technique I: Sampling

Sampling

- **Why:** Random sampling is a powerful, general tool for data analysis. We'll see several variants and applications.
- **Core idea:** pick a small random subset S from a large dataset D and estimate the quantity of interest using S instead of all of D.
- **What matters:** the sampling strategy, sample size, and estimator used in analysis.



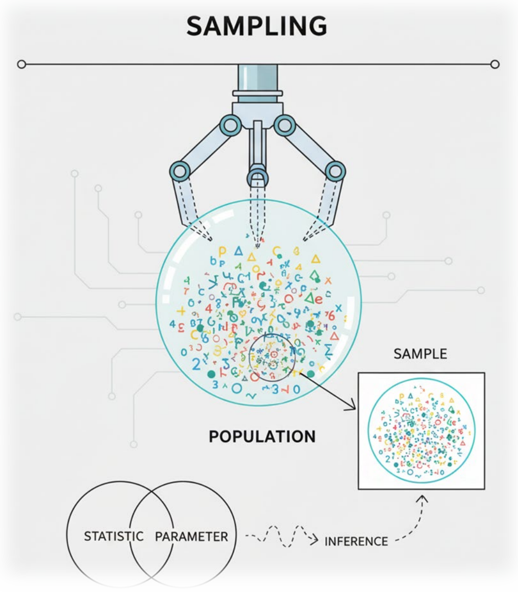
Sampling: Common Variants

- **Simple random sampling:** with/without replacement
- **Reservoir sampling:** maintain a uniform k-sample from a stream (single pass)
- **Stratified sampling:** sample within groups/segments to reduce variance
- **Weighted sampling**: assign a weight to each item; sample accordingly.

Estimation and Accuracy Analysis

- *Unbiased* **estimators** with variance you can compute/upper bound
- **Concentration:** error ϵ with failure prob. δ typically needs sample size $O(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$
  - Typically apply Chebyshev or Hoeffding/Chernoff to bound accuracy and uncertainty

- $\mathbb{E}[\hat{\theta}] = \theta$, and $\mathrm{Var}(\hat{\theta})$ is bounded (estimator is correct on average but no reliable)
- **Variance reduction via averaging.** Consider k independent copies of unbiased $\hat{\theta}_1, \cdots, \hat{\theta}_k$; then $\hat{\theta}_{\mathrm{avg}} = \sum \hat{\theta}_i \,/k$
  - $\mathbb{E}[\hat{\theta}_{\mathrm{avg}}] = \theta$, but $\mathrm{Var}(\hat{\theta}_{\mathrm{avg}})$= $\mathrm{Var}(\hat{\theta})/k$
  - Chebyshev's inequality: $\Pr\left[\left|\hat{\theta}_{\mathrm{avg}} - \theta\right| \geq \epsilon\right] \leq \frac{\mathrm{Var}(\theta)}{k\epsilon^2}$. So, set $k = \frac{\sigma^2}{\epsilon^2\delta}$ to get probability down to δ
  - Chernoff bound: $\Pr\left[\left|\hat{\theta}_{\mathrm{avg}} - \theta\right| \geq \epsilon\right] \leq 2^{-\Omega(\mathrm{k}\epsilon^2)}$. So, set $k = O(\frac{1}{\epsilon^2}\log\frac{1}{\delta})$ to get probability down to δ



How to perform sampling?

Basic Sampling Strategies

- **Setup:** dataset of size m; goal is a uniform sample of size k

With replacement

- **Procedure:** repeat k times, draw $i \sim \mathbf{Uniform}\{1, \dots, m\}$ and include item e_i
- Duplicates allowed; draws are independent (*i.i.d.*)
- Use when independence simplifies analysis or when sampling cost per draw is small

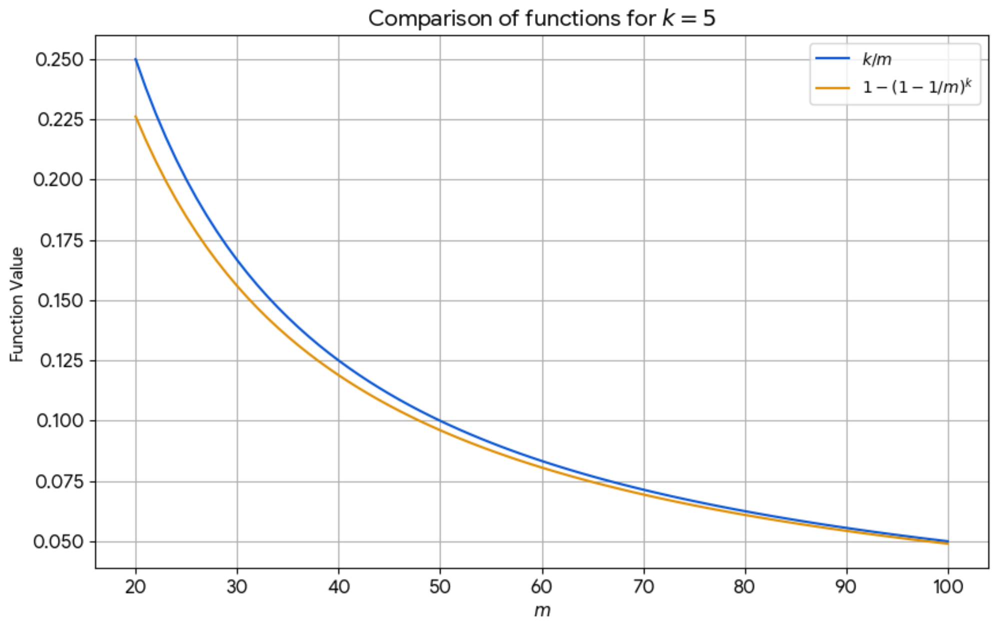
Without replacement

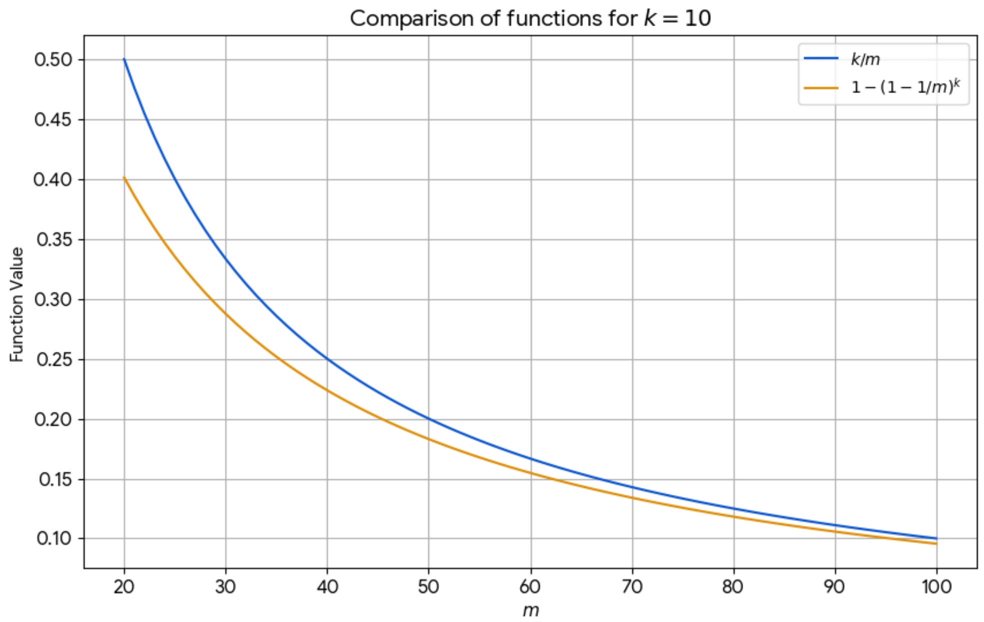
- **Procedure:** choose one k-subset uniformly from all $\binom{m}{k}$ subsets
- No duplicates; draws are dependent
- Marginal inclusion probability for any item: k/m

Compare the marginal inclusion probability of w. replacement and w/o. replacement

Marginal Inclusion Probability Computation

Marginal Inclusion Probability Computation





Reservoir Sampling

- How to draw a uniform sample w/o knowing length of stream in advance?

Key tool: rejection sampling

Simple example. Choose a random integer r in $\{1, \cdots, m\}$

- Let $k = \lceil \log m \rceil$
- Use k random bits to generate an integer r uniformly in $\{1, \cdots, 2^k\}$
- If $r \leq m$, output r; otherwise, **reject** r and repeat

Expected number of iterations? **(HW0)**

Reservoir Sampling

- How to draw a uniform sample w/o knowing length of stream in advance?

Claim. Let m be the length of the stream.

The output sample is uniform. i.e.,

$$\forall i \in [m], \Pr[\text{sample} = e_i] = 1/m$$

Proof. (by induction)

ReservoirSample (stream):

```
sample ← ∅, t ← 0
foreach item x in stream:
        t ← t + 1
        // Replace with probability 1/t
        if RandomUniform(0,1) < 1/t:
                sample ← x
return sample
```

Reservoir Sampling

- How to draw a uniform sample w/o knowing length of stream in advance?

Claim. Let m be the length of the stream.

The output sample is uniform. I.e.,

$$\forall i \in [m], \Pr[\text{sample} = e_i] = 1/m$$

Question. How to pick k samples?

How to sample k items **without replacement**?

ReservoirSample (stream):

sample $\leftarrow \emptyset, t \leftarrow 0$

foreach item x in stream:

$t \leftarrow t + 1$

// Replace with probability $1/t$

if RandomUniform$(0,1) < 1/t$:

sample $\leftarrow x$

return sample

Reservoir Sampling without Replacement

- How to draw a uniform sample w/o knowing length of stream in advance?

Claim. Let m be the length of the stream.
The probability of selecting all items are equal,

$$\forall i \in [m], \Pr[e_i \in S] = k/m$$

Proof.

ReservoirSample (stream):

S[1...k]← Stream[1...k], $t \leftarrow k$

foreach x in stream after position k:

$t \leftarrow t + 1$

// Replace with probability k/t

$r \leftarrow$ RandomUniformInt$[1, t]$

if $r \leq k$:

S[r] $\leftarrow x$

return S

Reservoir Sampling without Replacement

- How to draw a uniform sample w/o knowing length of stream in advance?

Claim. Let m be the length of the stream.
The probability of selecting all items are equal,

$$\forall i \in [m], \Pr[e_i \in S] = k/m$$

A different implementation in **HW 1**

ReservoirSample (stream):

S[1…k]← Stream[1…k], $t \leftarrow k$

foreach x in stream after position k:

$t \leftarrow t + 1$

// Replace with probability k/t

$r \leftarrow$ RandomUniformInt$[1, t]$

if $r \leq k$:

S[r] $\leftarrow x$

return S

Weighted Sampling

- Now, each item x_i in the stream is assigned with a weight $w_i > 0$. Goal is sample item i proportional to its weight; i.e., w_i/W where $W = \sum_{i=1}^{m} w_i$.
- How to implement in streaming?

Correctness Proof.

ReservoirSample (stream):

$\mathsf{sample} \leftarrow \emptyset, t \leftarrow 0, W \leftarrow 0$

foreach item x_i in stream:

$t \leftarrow t + 1, W \leftarrow W + w_i$

// Replace with probability w_i/W

if RandomUniform$(0,1) < w_i/W$:

$\mathsf{sample} \leftarrow x$

return sample

Mean and Median via Sampling

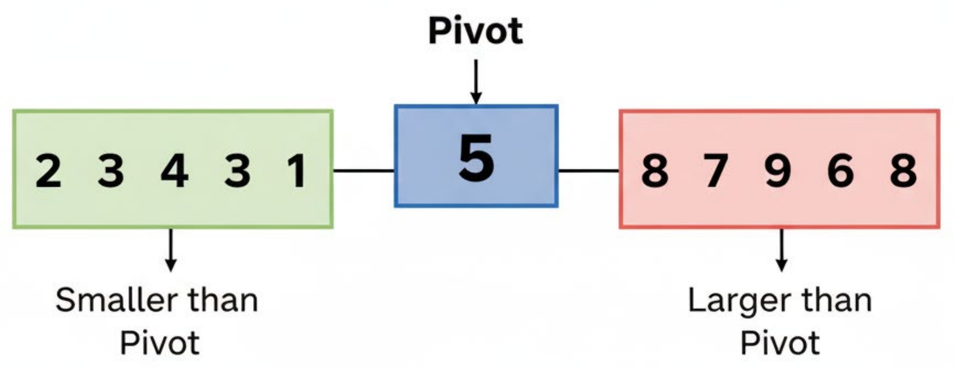
Mean and Median Statistics

- Given a list of n numbers $x_1, \cdots, x_n$:
  - **Mean:** average value = $\sum_{i=1}^{n} x_i \,/n$
  - **Median:** the middle number after sorting (if n is even; the average of the two middle ones)

Mean can be computed easily in $O(n)$ time. Similarly, for **Median** (but much more involved).

How to compute them in streaming setting?

- **Mean** is still easy! What about **Median**?



How to Compute Median in $O(n)$

- Medians of Median algorithm