Want to store a set $S$ of items from a massive universe of possible items (e.g., images, text documents, IP addresses).

- Allow small probability $\delta > 0$ of false positives. I.e., for any $x$,
  \[
  \Pr(query(x) = 1 \text{ and } x \notin S) \leq \delta.
  \]
Approximately Maintaining a Set

Want to store a set $S$ of items from a massive universe of possible items (e.g., images, text documents, IP addresses).

**Goal:** support $\text{insert}(x)$ to add $x$ to the set and $\text{query}(x)$ to check if $x$ is in the set. Both in $O(1)$ time.

- Allow small probability $\delta > 0$ of false positives. I.e., for any $x$,

$$\Pr(\text{query}(x) = 1 \text{ and } x \notin S) \leq \delta.$$
Want to store a set $S$ of items from a massive universe of possible items (e.g., images, text documents, IP addresses).

**Goal:** support $\text{insert}(x)$ to add $x$ to the set and $\text{query}(x)$ to check if $x$ is in the set. Both in $O(1)$ time.

- Allow small probability $\delta > 0$ of false positives. I.e., for any $x$,

$$\Pr(\text{query}(x) = 1 \text{ and } x \notin S) \leq \delta.$$
Want to store a set $S$ of items from a massive universe of possible items (e.g., images, text documents, IP addresses).

**Goal:** support $\text{insert}(x)$ to add $x$ to the set and $\text{query}(x)$ to check if $x$ is in the set. Both in $O(1)$ time.

- Allow small probability $\delta > 0$ of false positives. I.e., for any $x$,

$$\Pr(\text{query}(x) = 1 \text{ and } x \notin S) \leq \delta.$$
Want to store a set $S$ of items from a massive universe of possible items (e.g., images, text documents, IP addresses).

**Goal:** support $\text{insert}(x)$ to add $x$ to the set and $\text{query}(x)$ to check if $x$ is in the set. Both in $O(1)$ time.

- Allow small probability $\delta > 0$ of false positives. I.e., for any $x$,

  $$\Pr(\text{query}(x) = 1 \text{ and } x \not\in S) \leq \delta.$$ 

**Solution:** Bloom filters (repeated random hashing). Will use much less space than a hash table.
Chose $k$ independent random hash functions $h_1, \ldots, h_k$ mapping the universe of elements $U \rightarrow [m]$. No false negatives. False positives more likely with more insertions.
Chose $k$ independent random hash functions $h_1, \ldots, h_k$ mapping the universe of elements $U \rightarrow [m]$.

- Maintain an array $A$ containing $m$ bits, all initially 0.
BLOOM FILTERS

Chose $k$ independent random hash functions $h_1, \ldots, h_k$ mapping the universe of elements $U \rightarrow [m]$.

- Maintain an array $A$ containing $m$ bits, all initially 0.
- $\text{insert}(x)$: set all bits $A[h_1(x)] = \ldots = A[h_k(x)] := 1$. 

No false negatives. False positives more likely with more insertions.
Bloom Filters

Chose $k$ independent random hash functions $h_1, \ldots, h_k$ mapping the universe of elements $U \rightarrow [m]$.

- Maintain an array $A$ containing $m$ bits, all initially 0.
- $\text{insert}(x)$: set all bits $A[h_1(x)] = \ldots = A[h_k(x)] := 1$.
- $\text{query}(x)$: return 1 only if $A[h_1(x)] = \ldots = A[h_k(x)] = 1$.

No false negatives. False positives more likely with more insertions.
BLOOM FILTERS

Chose \( k \) independent random hash functions \( h_1, \ldots, h_k \) mapping the universe of elements \( U \rightarrow [m] \).

- Maintain an array \( A \) containing \( m \) bits, all initially 0.
- \( \text{insert}(x) \): set all bits \( A[h_1(x)] = \ldots = A[h_k(x)] := 1 \).
- \( \text{query}(x) \): return 1 only if \( A[h_1(x)] = \ldots = A[h_k(x)] = 1 \).

\[
\begin{array}{cccccccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]
BLOOM FILTERS

Chose $k$ independent random hash functions $h_1, \ldots, h_k$ mapping the universe of elements $U \to [m]$.

- Maintain an array $A$ containing $m$ bits, all initially 0.
- $\text{insert}(x)$: set all bits $A[h_1(x)] = \ldots = A[h_k(x)] := 1$.
- $\text{query}(x)$: return 1 only if $A[h_1(x)] = \ldots = A[h_k(x)] = 1$.

\begin{center}
\begin{tabular}{c|cccccccccccc}
\textbf{Insertions} \\
\hline
m bit array $A$ & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\textbf{Queries:} & & & & & & & & & & & &
\end{tabular}
\end{center}
Bloom Filters

Chose $k$ independent random hash functions $h_1, \ldots, h_k$ mapping the universe of elements $U \rightarrow [m]$.

- Maintain an array $A$ containing $m$ bits, all initially 0.
- $\text{insert}(x)$: set all bits $A[h_1(x)] = \ldots = A[h_k(x)] := 1$.
- $\text{query}(x)$: return 1 only if $A[h_1(x)] = \ldots = A[h_k(x)] = 1$.

No false negatives. False positives more likely with more insertions.

**Insertions:**

**Queries:**

$$
\begin{array}{cccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
$$
BLOOM FILTERS

Chose $k$ independent random hash functions $h_1, \ldots, h_k$ mapping the universe of elements $U \rightarrow [m]$.

- Maintain an array $A$ containing $m$ bits, all initially 0.
- $insert(x)$: set all bits $A[h_1(x)] = \ldots = A[h_k(x)] := 1$.
- $query(x)$: return 1 only if $A[h_1(x)] = \ldots = A[h_k(x)] = 1$.

No false negatives. False positives more likely with more insertions.
Bloom Filters

Chose $k$ independent random hash functions $h_1, \ldots, h_k$ mapping the universe of elements $U \rightarrow [m]$.

- Maintain an array $A$ containing $m$ bits, all initially 0.
- $\text{insert}(x)$: set all bits $A[h_1(x)] = \ldots = A[h_k(x)] := 1$.
- $\text{query}(x)$: return 1 only if $A[h_1(x)] = \ldots = A[h_k(x)] = 1$.
Chose $k$ independent random hash functions $h_1, \ldots, h_k$ mapping the universe of elements $U \to [m]$.

- Maintain an array $A$ containing $m$ bits, all initially 0.
- $\text{insert}(x)$: set all bits $A[h_1(x)] = \ldots = A[h_k(x)] := 1$.
- $\text{query}(x)$: return 1 only if $A[h_1(x)] = \ldots = A[h_k(x)] = 1$. 
Chose \( k \) independent random hash functions \( h_1, \ldots, h_k \) mapping the universe of elements \( U \rightarrow [m] \).

- Maintain an array \( A \) containing \( m \) bits, all initially 0.
- \textit{insert}(x): set all bits \( A[h_1(x)] = \ldots = A[h_k(x)] := 1 \).
- \textit{query}(x): return 1 only if \( A[h_1(x)] = \ldots = A[h_k(x)] = 1 \).

No false negatives. False positives more likely with more insertions.
Chose \( k \) independent random hash functions \( h_1, \ldots, h_k \) mapping the universe of elements \( U \rightarrow [m] \).

- Maintain an array \( A \) containing \( m \) bits, all initially 0.
- \textit{insert}(x): set all bits \( A[h_1(x)] = \ldots = A[h_k(x)] := 1 \).
- \textit{query}(x): return 1 only if \( A[h_1(x)] = \ldots = A[h_k(x)] = 1 \).
Bloom Filters

Chose $k$ independent random hash functions $h_1, \ldots, h_k$ mapping the universe of elements $U \rightarrow [m]$.

- Maintain an array $A$ containing $m$ bits, all initially 0.
- $insert(x)$: set all bits $A[h_1(x)] = \ldots = A[h_k(x)] := 1$.
- $query(x)$: return 1 only if $A[h_1(x)] = \ldots = A[h_k(x)] = 1$. No false negatives. False positives more likely with more insertions.
Chose $k$ independent random hash functions $h_1, \ldots, h_k$ mapping the universe of elements $U \rightarrow [m]$.

- Maintain an array $A$ containing $m$ bits, all initially 0.
- $\text{insert}(x)$: set all bits $A[h_1(x)] = \ldots = A[h_k(x)] := 1$.
- $\text{query}(x)$: return 1 only if $A[h_1(x)] = \ldots = A[h_k(x)] = 1$. 

No false negatives. False positives more likely with more insertions.
BLOOM FILTERS

Chose \( k \) independent random hash functions \( h_1, \ldots, h_k \) mapping the universe of elements \( U \to [m] \).

- Maintain an array \( A \) containing \( m \) bits, all initially 0.
- \( insert(x) \): set all bits \( A[h_1(x)] = \ldots = A[h_k(x)] := 1 \).
- \( query(x) \): return 1 only if \( A[h_1(x)] = \ldots = A[h_k(x)] = 1 \).
Chose \( k \) independent random hash functions \( h_1, \ldots, h_k \) mapping the universe of elements \( U \rightarrow [m] \).

- Maintain an array \( A \) containing \( m \) bits, all initially 0.
- \( \text{insert}(x) \): set all bits \( A[h_1(x)] = \ldots = A[h_k(x)] := 1 \).
- \( \text{query}(x) \): return 1 only if \( A[h_1(x)] = \ldots = A[h_k(x)] = 1 \).
Chose $k$ independent random hash functions $h_1, \ldots, h_k$ mapping the universe of elements $U \rightarrow [m]$.

- Maintain an array $A$ containing $m$ bits, all initially 0.
- $\text{insert}(x)$: set all bits $A[h_1(x)] = \ldots = A[h_k(x)] := 1$.
- $\text{query}(x)$: return 1 only if $A[h_1(x)] = \ldots = A[h_k(x)] = 1$. 

No false negatives. False positives more likely with more insertions.
Bloom Filters

Chose $k$ independent random hash functions $h_1, \ldots, h_k$ mapping the universe of elements $U \rightarrow [m]$.

- Maintain an array $A$ containing $m$ bits, all initially 0.
- $\text{insert}(x)$: set all bits $A[h_1(x)] = \ldots = A[h_k(x)] := 1$.
- $\text{query}(x)$: return 1 only if $A[h_1(x)] = \ldots = A[h_k(x)] = 1$.

No false negatives. False positives more likely with more insertions.
For a bloom filter with $m$ bits and $k$ hash functions, the insertion and query time is $O(k)$. 

How does the false positive rate $\delta$ depend on $m$, $k$, and the number of items inserted? Step 1: What is the probability that after inserting $n$ elements, the $i$th bit of the array $A$ is still 0?

$$\Pr(A[i] = 0) = \Pr(h_1(x_1) \neq i \cap \ldots \cap h_k(x_k) \neq i \cap \ldots) = \Pr(h_1(x_1) \neq i) \times \ldots \times \Pr(h_k(x_k) \neq i) \times ...$$

$n \times k$ events each occurring with probability $1 - 1/m = \left(1 - \frac{1}{m}\right)^k n$.
For a bloom filter with $m$ bits and $k$ hash functions, the insertion and query time is $O(k)$. How does the false positive rate $\delta$ depend on $m$, $k$, and the number of items inserted?
For a bloom filter with $m$ bits and $k$ hash functions, the insertion and query time is $O(k)$. How does the false positive rate $\delta$ depend on $m$, $k$, and the number of items inserted?

**Step 1:** What is the probability that after inserting $n$ elements, the $i^{th}$ bit of the array $A$ is still 0?
For a bloom filter with $m$ bits and $k$ hash functions, the insertion and query time is $O(k)$. How does the false positive rate $\delta$ depend on $m$, $k$, and the number of items inserted?

**Step 1:** What is the probability that after inserting $n$ elements, the $i^{th}$ bit of the array $A$ is still 0? $n \times k$ total hashes must not hit bit $i$.

$$\Pr(A[i] = 0) = \Pr(h_1(x_1) \neq i \cap \ldots \cap h_k(x_k) \neq i$$

$$\cap h_1(x_2) \neq i \ldots \cap h_k(x_2) \neq i \cap \ldots)$$
For a bloom filter with $m$ bits and $k$ hash functions, the insertion and query time is $O(k)$. How does the false positive rate $\delta$ depend on $m$, $k$, and the number of items inserted?

**Step 1:** What is the probability that after inserting $n$ elements, the $i^{th}$ bit of the array $A$ is still 0? $n \times k$ total hashes must not hit bit $i$.

\[
\Pr(A[i] = 0) = \Pr(h_1(x_1) \neq i \cap \ldots \cap h_k(x_k) \neq i \\
\cap h_1(x_2) \neq i \ldots \cap h_k(x_2) \neq i \cap \ldots \\
= \Pr(h_1(x_1) \neq i) \times \ldots \times \Pr(h_k(x_1) \neq i) \times \Pr(h_1(x_2) \neq i) \ldots \\
k \cdot n \text{ events each occurring with probability } 1 - 1/m
\]
For a bloom filter with $m$ bits and $k$ hash functions, the insertion and query time is $O(k)$. How does the false positive rate $\delta$ depend on $m$, $k$, and the number of items inserted?

**Step 1:** What is the probability that after inserting $n$ elements, the $i^{th}$ bit of the array $A$ is still 0? $n \times k$ total hashes must not hit bit $i$.

$$\Pr(A[i] = 0) = \Pr(h_1(x_1) \neq i \land \ldots \land h_k(x_k) \neq i$$

$$\land h_1(x_2) \neq i \land \ldots \land h_k(x_2) \neq i \land \ldots)$$

$$= \Pr(h_1(x_1) \neq i) \times \ldots \times \Pr(h_k(x_1) \neq i) \times \Pr(h_1(x_2) \neq i) \ldots$$

$$= \underbrace{1 - \frac{1}{m}}_{k \cdot n \text{ events each occurring with probability } 1 - 1/m} \times \underbrace{1 - \frac{1}{m}}_{\ldots} \times \underbrace{1 - \frac{1}{m}}_{\ldots}$$

$$= \left(1 - \frac{1}{m}\right)^{kn}$$
How does the false positive rate $\delta$ depend on $m$, $k$, and the number of items inserted?

What is the probability that after inserting $n$ elements, the $i^{th}$ bit of the array $A$ is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn}$$

$n$: total number items in filter, $m$: number of bits in filter, $k$: number of random hash functions, $h_1, \ldots, h_k$: hash functions, $A$: bit array, $\delta$: false positive rate.
How does the false positive rate $\delta$ depend on $m$, $k$, and the number of items inserted?

What is the probability that after inserting $n$ elements, the $i^{th}$ bit of the array $A$ is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

$n$: total number items in filter, $m$: number of bits in filter, $k$: number of random hash functions, $h_1, \ldots, h_k$: hash functions, $A$: bit array, $\delta$: false positive rate.
How does the false positive rate $\delta$ depend on $m$, $k$, and the number of items inserted?

What is the probability that after inserting $n$ elements, the $i^{th}$ bit of the array $A$ is still 0?

$$\Pr(A[i] = 0) = \left(1 - \frac{1}{m}\right)^{kn} \approx e^{-\frac{kn}{m}}$$

Let $T$ be the number of zeros in the array after $n$ inserts. Then,

$$E[T] = m \left(1 - \frac{1}{m}\right)^{kn} \approx me^{-\frac{kn}{m}}$$

$n$: total number items in filter, $m$: number of bits in filter, $k$: number of random hash functions, $h_1, \ldots, h_k$: hash functions, $A$: bit array, $\delta$: false positive rate.
If $T$ is the number of 0 entries, for a non-inserted element $w$:

$$\text{Pr}(A[h_1(w)] = \ldots = A[h_k(w)] = 1)$$

$$= \text{Pr}(A[h_1(w)] = 1) \times \ldots \times \text{Pr}(A[h_k(w)] = 1)$$

$$= (1 - T/m) \times \ldots \times (1 - T/m)$$

$$= (1 - T/m)^k$$
If \( T \) is the number of 0 entries, for a non-inserted element \( w \):

\[
\Pr(A[h_1(w)] = \ldots = A[h_k(w)] = 1) = \Pr(A[h_1(w)] = 1) \times \ldots \times \Pr(A[h_k(w)] = 1) = (1 - \frac{T}{m})^k
\]

- How small is \( \frac{T}{m} \)? Note that \( \frac{T}{m} \geq \frac{m-nk}{m} \approx e^{-\frac{kn}{m}} \) when \( kn \ll m \). More generally, it can be shown that \( \frac{T}{m} = \Omega(e^{-\frac{kn}{m}}) \) via Theorem 2 of:

  cglab.ca/~morin/publications/ds/bloom-submitted.pdf
False Positive Rate: with $m$ bits of storage, $k$ hash functions, and $n$ items inserted $\delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k$. 

- Can differentiate to show optimal number of hashes is $k = \ln 2 \cdot \frac{m}{n}$.
- Balances between filling up the array with too many hashes and having enough hashes so that even when the array is pretty full, a new item is unlikely to have all its bits set (yield a false positive).
False Positive Rate: with $m$ bits of storage, $k$ hash functions, and $n$ items inserted $\delta \approx \left(1 - e^{-kn/m}\right)^k$. 
False Positive Rate: with \( m \) bits of storage, \( k \) hash functions, and \( n \) items inserted, \( \delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k \).
False Positive Rate: with $m$ bits of storage, $k$ hash functions, and $n$ items inserted $\delta \approx \left(1 - e^{-kn/m}\right)^k$.
False Positive Rate: with $m$ bits of storage, $k$ hash functions, and $n$ items inserted $\delta \approx \left(1 - e^{-kn/m}\right)^k$.

- Can differentiate to show optimal number of hashes is $k = \ln 2 \cdot \frac{m}{n}$.
**False Positive Rate:** with $m$ bits of storage, $k$ hash functions, and $n$ items inserted $\delta \approx \left(1 - e^{-\frac{kn}{m}}\right)^k$.

- Can differentiate to show optimal number of hashes is $k = \ln 2 \cdot \frac{m}{n}$.
- Balances between filling up the array with too many hashes and having enough hashes so that even when the array is pretty full, a new item is unlikely to have all its bits set (yield a false positive)
Questions on Bloom Filters?
**Stream Processing:** Have a massive dataset $X$ with $n$ items $x_1, x_2, \ldots, x_n$ that arrive in a continuous stream. Not nearly enough space to store all the items (in a single location).

- Still want to analyze and learn from this data.
**Stream Processing:** Have a massive dataset $X$ with $n$ items $x_1, x_2, \ldots, x_n$ that arrive in a continuous stream. Not nearly enough space to store all the items (in a single location).

- Still want to analyze and learn from this data.
- Typically must compress the data on the fly, storing a data structure from which you can still learn useful information.
**Stream Processing**: Have a massive dataset $X$ with $n$ items $x_1, x_2, \ldots, x_n$ that arrive in a continuous stream. Not nearly enough space to store all the items (in a single location).

- Still want to analyze and learn from this data.
- Typically must compress the data on the fly, storing a data structure from which you can still learn useful information.
- Often the compression is randomized. E.g., bloom filters.
Stream Processing: Have a massive dataset $X$ with $n$ items $x_1, x_2, \ldots, x_n$ that arrive in a continuous stream. Not nearly enough space to store all the items (in a single location).

- Still want to analyze and learn from this data.
- Typically must compress the data on the fly, storing a data structure from which you can still learn useful information.
- Often the compression is randomized. E.g., bloom filters.
- Compared to traditional algorithm design, which focuses on minimizing runtime, the big question here is how much space is needed to answer queries of interest.
SOME EXAMPLES

- **Sensor data**: images from telescopes (15 terabytes per night from the Large Synoptic Survey Telescope), readings from seismometer arrays monitoring and predicting earthquake activity, traffic cameras and travel time sensors (Smart Cities), electrical grid monitoring.
SOME EXAMPLES

• **Sensor data:** images from telescopes (15 terabytes per night from the Large Synoptic Survey Telescope), readings from seismometer arrays monitoring and predicting earthquake activity, traffic cameras and travel time sensors (Smart Cities), electrical grid monitoring.
SOME EXAMPLES

- **Sensor data**: images from telescopes (15 terabytes per night from the Large Synoptic Survey Telescope), readings from seismometer arrays monitoring and predicting earthquake activity, traffic cameras and travel time sensors (Smart Cities), electrical grid monitoring.

- **Internet Traffic**: 500 million Tweets per day, 5.6 billion Google searches, billions of ad-clicks and other logs from instrumented webpages, IPs routed by network switches, ...
SOME EXAMPLES

• **Sensor data:** images from telescopes (15 terabytes per night from the Large Synoptic Survey Telescope), readings from seismometer arrays monitoring and predicting earthquake activity, traffic cameras and travel time sensors (Smart Cities), electrical grid monitoring.

• **Internet Traffic:** 500 million Tweets per day, 5.6 billion Google searches, billions of ad-clicks and other logs from instrumented webpages, IPs routed by network switches, ...

• **Datasets in Machine Learning:** When training e.g. a neural network on a large dataset (ImageNet with 14 million images), the data is typically processed in a stream due to storage limitations.
**SOME EXAMPLES**

- **Sensor data**: images from telescopes (15 terabytes per night from the Large Synoptic Survey Telescope), readings from seismometer arrays monitoring and predicting earthquake activity, traffic cameras and travel time sensors (Smart Cities), electrical grid monitoring.

- **Internet Traffic**: 500 million Tweets per day, 5.6 billion Google searches, billions of ad-clicks and other logs from instrumented webpages, IPs routed by network switches, ...

- **Datasets in Machine Learning**: When training e.g. a neural network on a large dataset (ImageNet with 14 million images), the data is typically processed in a stream due to storage limitations.
Distinct Elements (Count-Distinct) Problem: Given a stream \( x_1, \ldots, x_n \), output the number of distinct elements in the stream.
Distinct Elements (Count-Distinct) Problem: Given a stream \( x_1, \ldots, x_n \), output the number of distinct elements in the stream. E.g.,

\[ 1, 5, 7, 5, 2, 1 \rightarrow 4 \text{ distinct elements} \]
Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements in the stream. E.g.,

$1, 5, 7, 5, 2, 1 \rightarrow 4$ distinct elements
Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements in the stream. E.g.,

$$1, 5, 7, 5, 2, 1 \rightarrow 4 \text{ distinct elements}$$

Applications:

- Distinct IP addresses clicking on an ad or visiting a site.
- Distinct values in a database column (for estimating sizes of joins and group bys).
- Number of distinct search engine queries.
- Counting distinct motifs in large DNA sequences.
Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements in the stream. E.g.,

$$1, 5, 7, 5, 2, 1 \rightarrow 4 \text{ distinct elements}$$

Applications:

- Distinct IP addresses clicking on an ad or visiting a site.
- Distinct values in a database column (for estimating sizes of joins and group bys).
- Number of distinct search engine queries.
- Counting distinct motifs in large DNA sequences.

Google Sawzall, Facebook Presto, Apache Drill, Twitter Algebird
DISTINCT ELEMENTS IDEAS
Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.
Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

- Let $h : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, h(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$
Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

- Let $h : U \rightarrow [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, h(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$
Distinct Elements (Count-Distinct) Problem: Given a stream \( x_1, \ldots, x_n \), estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

- Let \( h : U \rightarrow [0, 1] \) be a random hash function (with a real valued output)
- \( s := 1 \)
- For \( i = 1, \ldots, n \)
  - \( s := \min(s, h(x_i)) \)
- Return \( \tilde{d} = \frac{1}{s} - 1 \)
Distinct Elements (Count-Distinct) Problem: Given a stream \(x_1, \ldots, x_n\), estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

- Let \(h : U \rightarrow [0, 1]\) be a random hash function (with a real valued output)
- \(s := 1\)
- For \(i = 1, \ldots, n\)
  - \(s := \min(s, h(x_i))\)
- Return \(\tilde{d} = \frac{1}{s} - 1\)
Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

- Let $h : U \rightarrow [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, h(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$
Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

- Let $h : U \rightarrow [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, h(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$
Distinct Elements (Count-Distinct) Problem: Given a stream \( x_1, \ldots, x_n \), estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

- Let \( h : U \to [0, 1] \) be a random hash function (with a real valued output)
- \( s := 1 \)
- For \( i = 1, \ldots, n \)
  - \( s := \min(s, h(x_i)) \)
- Return \( \tilde{d} = \frac{1}{s} - 1 \)
Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

- Let $h : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, h(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$
Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

- Let $h : U \to [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, h(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$
Distinct Elements (Count-Distinct) Problem: Given a stream $x_1, \ldots, x_n$, estimate the number of distinct elements.

Min-Hashing for Distinct Elements (variant of Flajolet-Martin):

- Let $h : U \rightarrow [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, h(x_i))$
- Return $\tilde{d} = \frac{1}{s} - 1$
Min-Hashing for Distinct Elements:

- Let \( h : U \rightarrow [0, 1] \) be a random hash function (with a real valued output)
- \( s := 1 \)
- For \( i = 1, \ldots, n \)
  - \( s := \min(s, h(x_i)) \)
- Return \( \hat{d} = \frac{1}{s} - 1 \)
Min-Hashing for Distinct Elements:

- Let \( h : U \rightarrow [0, 1] \) be a random hash function (with a real valued output)
- \( s := 1 \)
- For \( i = 1, \ldots, n \)
  - \( s := \min(s, h(x_i)) \)
- Return \( \hat{d} = \frac{1}{s} - 1 \)

After all items are processed, \( s \) is the minimum of \( d \) points chosen uniformly at random on \([0, 1]\). Where \( d = \# \) distinct elements.
Min-Hashing for Distinct Elements:

- Let $h : U \rightarrow [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, h(x_i))$
- Return $\hat{d} = \frac{1}{s} - 1$

- After all items are processed, $s$ is the minimum of $d$ points chosen uniformly at random on $[0, 1]$. Where $d = \#$ distinct elements.
- Intuition: The larger $d$ is, the smaller we expect $s$ to be.
Min-Hashing for Distinct Elements:

- Let $h : U \rightarrow [0, 1]$ be a random hash function (with a real valued output)
- $s := 1$
- For $i = 1, \ldots, n$
  - $s := \min(s, h(x_i))$
- Return $\hat{d} = \frac{1}{s} - 1$

- After all items are processed, $s$ is the minimum of $d$ points chosen uniformly at random on $[0, 1]$. Where $d = \#$ distinct elements.
- Intuition: The larger $d$ is, the smaller we expect $s$ to be.
- Same idea as Flajolet-Martin algorithm and HyperLogLog, except they use discrete hash functions.
$s$ is the minimum of $d$ points chosen uniformly at random on $[0, 1]$. Where $d = \#$ distinct elements.
s is the minimum of $d$ points chosen uniformly at random on $[0, 1]$. Where $d = \#$ distinct elements.

$\mathbb{E}[s] =$
s is the minimum of d points chosen uniformly at random on [0, 1]. Where d = # distinct elements.

\[ \mathbb{E}[s] = \frac{1}{d + 1} \] (using \( \mathbb{E}(s) = \int_0^{\infty} \Pr(s > x)dx \) + calculus)
s is the minimum of d points chosen uniformly at random on [0, 1]. Where d = \# distinct elements.

\[
\mathbb{E}[s] = \frac{1}{d + 1} \quad \text{(using } \mathbb{E}(s) = \int_0^\infty \Pr(s > x)dx \text{)} + \text{calculus)}
\]

- So estimate of \( \hat{d} = \frac{1}{s} - 1 \) output by the algorithm is correct if s exactly equals its expectation.
**PERFORMANCE IN EXPECTATION**

$s$ is the minimum of $d$ points chosen uniformly at random on $[0, 1]$. Where $d = \#$ distinct elements.

\[ E[s] = \frac{1}{d + 1} \]  
(using $E(s) = \int_0^\infty \Pr(s > x)dx$ + calculus)

- So estimate of $\hat{d} = \frac{1}{s} - 1$ output by the algorithm is correct if $s$ exactly equals its expectation. **Does this mean $E[\hat{d}] = d$?**
s is the minimum of d points chosen uniformly at random on [0, 1]. Where d = \# distinct elements.

\[ \mathbb{E}[s] = \frac{1}{d + 1} \] (using \( \mathbb{E}(s) = \int_0^\infty \Pr(s > x)dx \) + calculus)

- So estimate of \( \hat{d} = \frac{1}{s} - 1 \) output by the algorithm is correct if s exactly equals its expectation. Does this mean \( \mathbb{E}[\hat{d}] = d \)? No, but:
s is the minimum of d points chosen uniformly at random on [0, 1]. Where d = # distinct elements.

\[ E[s] = \frac{1}{d + 1} \] (using \( E(s) = \int_0^\infty \Pr(s > x)dx \) + calculus)

- So estimate of \( \hat{d} = \frac{1}{s} - 1 \) output by the algorithm is correct if s exactly equals its expectation. Does this mean \( E[\hat{d}] = d \)? No, but:

- **Approximation is robust**: if \( |s - E[s]| \leq \epsilon \cdot E[s] \) for any \( \epsilon \in (0, 1/2) \) and a small constant \( c \leq 4 \):

\[ (1 - c\epsilon)d \leq \hat{d} \leq (1 + c\epsilon)d \]