Hashing for Distinct Elements:

- Let \( h_1, h_2, \ldots, h_k : U \rightarrow [0, 1] \) be random hash functions
- \( s_1, s_2, \ldots, s_k := 1 \)
- For \( i = 1, \ldots, n \)
  - For \( j = 1, \ldots, k \), \( s_j := \min(s_j, h_j(x_i)) \)
- \( s := \frac{1}{k} \sum_{j=1}^{k} s_j \)
- Return \( \hat{d} = \frac{1}{s} - 1 \)

- Setting \( k = \frac{1}{\epsilon^2 \delta} \), algorithm returns \( \hat{d} \) with \( |d - \hat{d}| \leq 4\epsilon \cdot d \) with probability at least \( 1 - \delta \).

- Space complexity is \( k = \frac{1}{\epsilon^2 \delta} \) real numbers \( s_1, \ldots, s_k \).
- \( \delta = 5\% \) failure rate gives a factor 20 overhead in space complexity.
How can we improve our dependence on the failure rate $\delta$?
How can we improve our dependence on the failure rate $\delta$?

**The median trick:** Run $t = O(\log 1/\delta)$ trials each with failure probability $\delta' = 1/4$ – each using $k = \frac{1}{\delta'\epsilon^2} = \frac{4}{\epsilon^2}$ hash functions.
How can we improve our dependence on the failure rate $\delta$?

**The median trick:** Run $t = O(\log 1/\delta)$ trials each with failure probability $\delta' = 1/4$ – each using $k = \frac{1}{\delta' \epsilon^2} = \frac{4}{\epsilon^2}$ hash functions.

- Letting $\hat{d}_1, \ldots, \hat{d}_t$ be the outcomes of the $t$ trials, return
  $$\hat{d} = \text{median}(\hat{d}_1, \ldots, \hat{d}_t).$$
How can we improve our dependence on the failure rate $\delta$?

**The median trick:** Run $t = O(\log 1/\delta)$ trials each with failure probability $\delta' = 1/4$ – each using $k = \frac{1}{\delta' \epsilon^2} = \frac{4}{\epsilon^2}$ hash functions.

- Letting $\hat{d}_1, \ldots, \hat{d}_t$ be the outcomes of the $t$ trials, return
  $$\hat{d} = median(\hat{d}_1, \ldots, \hat{d}_t).$$

[Diagram showing the distribution of $\hat{d}$ and its median.]
How can we improve our dependence on the failure rate $\delta$?

**The median trick:** Run $t = O(\log 1/\delta)$ trials each with failure probability $\delta' = 1/4$ – each using $k = \frac{1}{\delta' \epsilon^2} = \frac{4}{\epsilon^2}$ hash functions.

- Letting $\hat{d}_1, \ldots, \hat{d}_t$ be the outcomes of the $t$ trials, return
  $$\hat{d} = median(\hat{d}_1, \ldots, \hat{d}_t).$$

> 1/2

- If > 1/2 of trials fall in $[(1 - 4\epsilon)d, (1 + 4\epsilon)d]$, then the median will.
• $\hat{d}_1, \ldots, \hat{d}_t$ are the outcomes of the $t$ trials, each falling in

$\left[(1 - 4\epsilon)d, (1 + 4\epsilon)d\right]$

with probability at least $3/4$. Let $\hat{d} = median(\hat{d}_1, \ldots, \hat{d}_t)$.

What is the probability that the median $\hat{d}$ falls in $\left[(1 - 4\epsilon)d, (1 + 4\epsilon)d\right]$?
• \( \hat{d}_1, \ldots, \hat{d}_t \) are the outcomes of the \( t \) trials, each falling in
\[
[(1 - 4\epsilon)d, (1 + 4\epsilon)d]
\]
with probability at least \( 3/4 \). Let \( \hat{d} = median(\hat{d}_1, \ldots, \hat{d}_t) \).

What is the probability that the median \( \hat{d} \) falls in \( [(1 - 4\epsilon)d, (1 + 4\epsilon)d] \)?

• Let \( X \) be the \# of trials falling in \( [(1 - 4\epsilon)d, (1 + 4\epsilon)d] \).
THE MEDIAN TRICK

• $\hat{d}_1, \ldots, \hat{d}_t$ are the outcomes of the $t$ trials, each falling in
  $$[(1 - 4\epsilon)d, (1 + 4\epsilon)d]$$
  with probability at least $3/4$. Let $\hat{d} = \text{median}(\hat{d}_1, \ldots, \hat{d}_t)$.

What is the probability that the median $\hat{d}$ falls in $[(1 - 4\epsilon)d, (1 + 4\epsilon)d]$?

• Let $X$ be the number of trials falling in $[(1 - 4\epsilon)d, (1 + 4\epsilon)d]$.

$$\Pr \left( \hat{d} \notin [(1 - 4\epsilon)d, (1 + 4\epsilon)d] \right) \leq \Pr \left( X \leq \frac{1}{2} \cdot t \right)$$
THE MEDIAN TRICK

• \( \hat{d}_1, \ldots, \hat{d}_t \) are the outcomes of the \( t \) trials, each falling in

\[
[(1 - 4\epsilon)d, (1 + 4\epsilon)d]
\]

with probability at least \( 3/4 \). Let \( \hat{d} = median(\hat{d}_1, \ldots, \hat{d}_t) \).

What is the probability that the median \( \hat{d} \) falls in \( [(1 - 4\epsilon)d, (1 + 4\epsilon)d] \)?

• Let \( X \) be the \# of trials falling in \( [(1 - 4\epsilon)d, (1 + 4\epsilon)d] \). \( \mathbb{E}[X] \geq \).

\[
\Pr\left( \hat{d} \notin [(1 - 4\epsilon)d, (1 + 4\epsilon)d] \right) \leq \Pr\left( X \leq \frac{1}{2} \cdot t \right)
\]
• \( \hat{d}_1, \ldots, \hat{d}_t \) are the outcomes of the \( t \) trials, each falling in 
\[
[(1 - 4\epsilon)d, (1 + 4\epsilon)d]
\]
with probability at least \( 3/4 \). Let \( \hat{d} = \text{median}(\hat{d}_1, \ldots, \hat{d}_t) \).

What is the probability that the median \( \hat{d} \) falls in \( [(1 - 4\epsilon)d, (1 + 4\epsilon)d] \)?

• Let \( X \) be the \# of trials falling in \( [(1 - 4\epsilon)d, (1 + 4\epsilon)d] \). \( \mathbb{E}[X] \geq \frac{3}{4} \cdot t \).

\[
\Pr\left( \hat{d} \notin [(1 - 4\epsilon)d, (1 + 4\epsilon)d] \right) \leq \Pr\left( X \leq \frac{1}{2} \cdot t \right)
\]
THE MEDIAN TRICK

• \( \hat{d}_1, \ldots, \hat{d}_t \) are the outcomes of the \( t \) trials, each falling in 

\[
[(1 - 4\epsilon)d, (1 + 4\epsilon)d]
\]

with probability at least \( 3/4 \). Let \( \hat{d} = median(\hat{d}_1, \ldots, \hat{d}_t) \).

What is the probability that the median \( \hat{d} \) falls in \([ (1 - 4\epsilon)d, (1 + 4\epsilon)d] \)?

• Let \( X \) be the \# of trials falling in \([ (1 - 4\epsilon)d, (1 + 4\epsilon)d] \). \( \mathbb{E}[X] \geq \frac{3}{4} \cdot t \).

\[
\Pr(\hat{d} \notin [(1 - 4\epsilon)d, (1 + 4\epsilon)d]) \leq \Pr(X \leq \frac{1}{2} \cdot t)
\]
THE MEDIAN TRICK

• \(\hat{d}_1, \ldots, \hat{d}_t\) are the outcomes of the \(t\) trials, each falling in 
  \[
  [(1 - 4\epsilon)d, (1 + 4\epsilon)d]
  \]
  with probability at least 3/4. Let \(\hat{d} = median(\hat{d}_1, \ldots, \hat{d}_t)\).

What is the probability that the median \(\hat{d}\) falls in \([(1 - 4\epsilon)d, (1 + 4\epsilon)d]\)?

• Let \(X\) be the \# of trials falling in \([(1 - 4\epsilon)d, (1 + 4\epsilon)d]\). \(\mathbb{E}[X] \geq \frac{3}{4} \cdot t\).

\[
\Pr(\hat{d} \notin [(1 - 4\epsilon)d, (1 + 4\epsilon)d]) \leq \Pr(X \leq \frac{1}{2} \cdot t) \leq \Pr(|X - \mathbb{E}[X]| \geq \frac{1}{4}t)
\]
• $\hat{d}_1, \ldots, \hat{d}_t$ are the outcomes of the $t$ trials, each falling in $[(1 - 4\epsilon)d, (1 + 4\epsilon)d]$ with probability at least $3/4$. Let $\hat{d} = \text{median}(\hat{d}_1, \ldots, \hat{d}_t)$.

What is the probability that the median $\hat{d}$ falls in $[(1 - 4\epsilon)d, (1 + 4\epsilon)d]$?

• Let $X$ be the # of trials falling in $[(1 - 4\epsilon)d, (1 + 4\epsilon)d]$. $\mathbb{E}[X] \geq \frac{3}{4} \cdot t$.

$$\Pr(\hat{d} \notin [(1 - 4\epsilon)d, (1 + 4\epsilon)d]) \leq \Pr\left(X \leq \frac{1}{2} \cdot t\right) \leq \Pr\left(|X - \mathbb{E}[X]| \geq \frac{1}{4} t\right)$$

Apply Chernoff bound:
THE MEDIAN TRICK

• $\hat{d}_1, \ldots, \hat{d}_t$ are the outcomes of the $t$ trials, each falling in

$$[(1 - 4\epsilon)d, (1 + 4\epsilon)d]$$

with probability at least $3/4$. Let $\hat{d} = median(\hat{d}_1, \ldots, \hat{d}_t)$.

What is the probability that the median $\hat{d}$ falls in $[(1 - 4\epsilon)d, (1 + 4\epsilon)d]$?

• Let $X$ be the # of trials falling in $[(1 - 4\epsilon)d, (1 + 4\epsilon)d]$. $\mathbb{E}[X] \geq \frac{3}{4} \cdot t$.

$$Pr\left(\hat{d} \notin [(1 - 4\epsilon)d, (1 + 4\epsilon)d]\right) \leq Pr\left(X \leq \frac{1}{2} \cdot t\right) \leq Pr\left(|X - \mathbb{E}[X]| \geq \frac{1}{4} t\right)$$

Apply Chernoff bound:

$$Pr\left(|X - \mathbb{E}[X]| \geq \frac{1}{3} \mathbb{E}[X]\right) \leq 2 \exp\left(- \frac{12 \cdot 3}{4} \frac{t}{2 + 1/3}\right) = O\left(e^{-O(t)}\right).$$
THE MEDIAN TRICK

- \(\hat{d}_1, \ldots, \hat{d}_t\) are the outcomes of the \(t\) trials, each falling in \([ (1 - 4\epsilon)d, (1 + 4\epsilon)d ]\)
  
  with probability at least \(3/4\). Let \(\hat{d} = median(\hat{d}_1, \ldots, \hat{d}_t)\).

  What is the probability that the median \(\hat{d}\) falls in \([ (1 - 4\epsilon)d, (1 + 4\epsilon)d ]\)?

- Let \(X\) be the \# of trials falling in \([ (1 - 4\epsilon)d, (1 + 4\epsilon)d ]\). \(\mathbb{E}[X] \geq \frac{3}{4} \cdot t\).

\[
\Pr\left(\hat{d} \notin [(1 - 4\epsilon)d, (1 + 4\epsilon)d]\right) \leq \Pr\left(X \leq \frac{1}{2} \cdot t\right) \leq \Pr\left(|X - \mathbb{E}[X]| \geq \frac{1}{4} t\right)
\]

Apply Chernoff bound:

\[
\Pr\left(|X - \mathbb{E}[X]| \geq \frac{1}{3} \mathbb{E}[X]\right) \leq 2 \exp\left(-\frac{\frac{1}{3}^2 \cdot \frac{3}{4} t}{2 + 1/3}\right) = O\left(e^{-O(t)}\right).
\]

- Setting \(t = O(\log(1/\delta))\) gives failure probability \(e^{-\log(1/\delta)} = \delta\).
**Upshot:** The median of $t = O(\log(1/\delta))$ independent runs of the hashing algorithm for distinct elements returns

$$\hat{d} \in [(1 - 4\epsilon)d, (1 + 4\epsilon)d]$$

with probability at least $1 - \delta$. 
**Upshot:** The median of $t = O(\log(1/\delta))$ independent runs of the hashing algorithm for distinct elements returns

$$\hat{d} \in [(1 - 4\epsilon)\delta, (1 + 4\epsilon)\delta]$$

with probability at least $1 - \delta$.

**Total Space Complexity:** $t$ trials, each using $k = \frac{1}{\epsilon^2 \delta'}$ hash functions, for $\delta' = 1/4$. Space is $\frac{4t}{\epsilon^2} = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ real numbers (the minimum value of each hash function).
**Upshot:** The median of $t = O(\log(1/\delta))$ independent runs of the hashing algorithm for distinct elements returns

$$\hat{d} \in [(1 - 4\epsilon)d, (1 + 4\epsilon)d]$$

with probability at least $1 - \delta$.

**Total Space Complexity:** $t$ trials, each using $k = \frac{1}{\epsilon^2 \delta'}$ hash functions, for $\delta' = 1/4$. Space is $\frac{4t}{\epsilon^2} = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right)$ real numbers (the minimum value of each hash function).

No dependence on the number of distinct elements $d$ or the number of items in the stream $n!$ Both can be very large.
**Upshot:** The median of \( t = O(\log(1/\delta)) \) independent runs of the hashing algorithm for distinct elements returns

\[
\hat{d} \in [(1 - 4\epsilon)d, (1 + 4\epsilon)d]
\]

with probability at least \( 1 - \delta \).

**Total Space Complexity:** \( t \) trials, each using \( k = \frac{1}{\epsilon^2 \delta'} \) hash functions, for \( \delta' = 1/4 \). Space is \( \frac{4t}{\epsilon^2} = O\left(\frac{\log(1/\delta)}{\epsilon^2}\right) \) real numbers (the minimum value of each hash function).

No dependence on the number of distinct elements \( d \) or the number of items in the stream \( n! \). Both can be very large.

**A note on the median:** The median is often used as a robust alternative to the mean, when there are outliers (e.g., heavy tailed distributions, corrupted data).
Our algorithm uses continuous valued fully random hash functions.
Our algorithm uses continuous valued fully random hash functions. Can’t be implemented...
Our algorithm uses continuous valued fully random hash functions. Can’t be implemented...

- The idea of using the minimum hash value of $x_1, \ldots, x_n$ to estimate the number of distinct elements naturally extends to when the hash functions map to discrete values.
Our algorithm uses continuous valued fully random hash functions. Can’t be implemented...

- The idea of using the minimum hash value of $x_1, \ldots, x_n$ to estimate the number of distinct elements naturally extends to when the hash functions map to discrete values.
- Flajolet-Martin (LogLog) algorithm and HyperLogLog.
Our algorithm uses continuous valued fully random hash functions. Can’t be implemented...

- The idea of using the minimum hash value of $x_1, \ldots, x_n$ to estimate the number of distinct elements naturally extends to when the hash functions map to discrete values.
- Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<table>
<thead>
<tr>
<th>$h(x_1)$</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(x_2)$</td>
<td>1001100</td>
</tr>
<tr>
<td>$h(x_3)$</td>
<td>1001110</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>$h(x_n)$</td>
<td>1011000</td>
</tr>
</tbody>
</table>
Our algorithm uses continuous valued fully random hash functions. Can’t be implemented...

- The idea of using the minimum hash value of $x_1, \ldots, x_n$ to estimate the number of distinct elements naturally extends to when the hash functions map to discrete values.
- Flajolet-Martin (LogLog) algorithm and HyperLogLog.

| $h(x_1)$ | 1010010  |
| $h(x_2)$ | 1001100  |
| $h(x_3)$ | 1001110  |
| $\vdots$ | $\vdots$ |
| $h(x_n)$ | 1011000  |

Estimate \# distinct elements based on maximum number of trailing zeros $m$. 
Our algorithm uses continuous valued fully random hash functions. Can’t be implemented...

- The idea of using the minimum hash value of $x_1, \ldots, x_n$ to estimate the number of distinct elements naturally extends to when the hash functions map to discrete values.
- Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<table>
<thead>
<tr>
<th>$h(x_1)$</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(x_2)$</td>
<td>1001100</td>
</tr>
<tr>
<td>$h(x_3)$</td>
<td>1001110</td>
</tr>
<tr>
<td>\vdots</td>
<td>\vdots</td>
</tr>
<tr>
<td>$h(x_n)$</td>
<td>1011000</td>
</tr>
</tbody>
</table>

Estimate # distinct elements based on maximum number of trailing zeros $m$.
The more distinct hashes we see, the higher we expect this maximum to be.
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

| \( h(x_1) \) | 1010010 |
| \( h(x_2) \) | 1001100 |
| \( h(x_3) \) | 1001110 |
| \( \vdots \) | \( \vdots \) |
| \( h(x_n) \) | 1011000 |

Estimate the number of distinct elements based on the maximum number of trailing zeros \( m \).
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<table>
<thead>
<tr>
<th>$h(x_1)$</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(x_2)$</td>
<td>1001100</td>
</tr>
<tr>
<td>$h(x_3)$</td>
<td>1001110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$h(x_n)$</td>
<td>1011000</td>
</tr>
</tbody>
</table>

Estimate $\#$ distinct elements based on maximum number of trailing zeros $m$.

With $d$ distinct elements, roughly what do we expect $m$ to be?

a) $O(1)$  b) $O(\log d)$  c) $O(\sqrt{d})$  d) $O(d)$
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<table>
<thead>
<tr>
<th>h(x₁)</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(x₂)</td>
<td>1001100</td>
</tr>
<tr>
<td>h(x₃)</td>
<td>1001110</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>h(xₙ)</td>
<td>1011000</td>
</tr>
</tbody>
</table>

Estimate # distinct elements based on maximum number of trailing zeros $m$.

With $d$ distinct elements, roughly what do we expect $m$ to be?

a) $O(1)$  
 b) $O(\log d)$  
 c) $O(\sqrt{d})$  
 d) $O(d)$
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<table>
<thead>
<tr>
<th>$h(x_i)$</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(x_2)$</td>
<td>1001100</td>
</tr>
<tr>
<td>$h(x_3)$</td>
<td>1001110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>$h(x_n)$</td>
<td>1011000</td>
</tr>
</tbody>
</table>

Estimate the number of distinct elements based on the maximum number of trailing zeros $m$.

With $d$ distinct elements, roughly what do we expect $m$ to be?

$$\Pr(h(x_i) \text{ has } x \text{ trailing zeros}) =$$
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

| $h(x_1)$ | 1010010 |
| $h(x_2)$ | 1001100 |
| $h(x_3)$ | 1001100 |
| $\vdots$ | $\vdots$ |
| $h(x_n)$ | 1011000 |

With $d$ distinct elements, roughly what do we expect $m$ to be?

$$\Pr(h(x_i) \text{ has } x \text{ trailing zeros}) = \frac{1}{2^x}$$

Estimate $\#$ distinct elements based on maximum number of trailing zeros $m$. 

Note: Careful averaging of estimates from multiple hash functions.
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<table>
<thead>
<tr>
<th>h(x_1)</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(x_2)</td>
<td>1001100</td>
</tr>
<tr>
<td>h(x_3)</td>
<td>1001110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>h(x_n)</td>
<td>1011000</td>
</tr>
</tbody>
</table>

With $d$ distinct elements, roughly what do we expect $m$ to be?

$$\Pr(h(x_i) \text{ has } \log d \text{ trailing zeros}) = \frac{1}{2^{\log d}}$$
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<table>
<thead>
<tr>
<th>h(x₁)</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(x₂)</td>
<td>1001100</td>
</tr>
<tr>
<td>h(x₃)</td>
<td>1001110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>h(xₙ)</td>
<td>1011000</td>
</tr>
</tbody>
</table>

Estimate the number of distinct elements based on the maximum number of trailing zeros \( m \).

With \( d \) distinct elements, roughly what do we expect \( m \) to be?

\[
\Pr(h(x_i) \text{ has } \log d \text{ trailing zeros}) = \frac{1}{2^{\log d}} = \frac{1}{d}.
\]
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<table>
<thead>
<tr>
<th>h(x₁)</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(x₂)</td>
<td>1001100</td>
</tr>
<tr>
<td>h(x₃)</td>
<td>1001110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>h(xₙ)</td>
<td>1011000</td>
</tr>
</tbody>
</table>

Estimate number of distinct elements based on maximum number of trailing zeros \( m \).

With \( d \) distinct elements, roughly what do we expect \( m \) to be?

\[
\Pr(h(x_i) \text{ has } \log d \text{ trailing zeros}) = \frac{1}{2^{\log d}} = \frac{1}{d}.
\]

So with \( d \) distinct hashes, expect to see 1 with \( \log d \) trailing zeros. Expect \( m \approx \log d \).
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<table>
<thead>
<tr>
<th>h(x₁)</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(x₂)</td>
<td>1001100</td>
</tr>
<tr>
<td>h(x₃)</td>
<td>1001110</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>h(xₙ)</td>
<td>1011000</td>
</tr>
</tbody>
</table>

Estimate # distinct elements based on maximum number of trailing zeros \( m \).

With \( d \) distinct elements, roughly what do we expect \( m \) to be?

\[
\Pr(h(x_i) \text{ has } \log d \text{ trailing zeros}) = \frac{1}{2^{\log d}} = \frac{1}{d}.
\]

So with \( d \) distinct hashes, expect to see 1 with \( \log d \) trailing zeros. Expect \( m \approx \log d \). \( m \) takes \( \log \log d \) bits to store.
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<table>
<thead>
<tr>
<th>$h(x_1)$</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>$h(x_2)$</td>
<td>1001100</td>
</tr>
<tr>
<td>$h(x_3)$</td>
<td>1001110</td>
</tr>
<tr>
<td>$\vdots$</td>
<td>$\ddots$</td>
</tr>
<tr>
<td>$h(x_n)$</td>
<td>1011000</td>
</tr>
</tbody>
</table>

Estimate \# distinct elements based on maximum number of trailing zeros $m$.

With $d$ distinct elements, roughly what do we expect $m$ to be?

$$\Pr(h(x_i) \text{ has } \log d \text{ trailing zeros}) = \frac{1}{2^{\log d}} = \frac{1}{d}.$$  

So with $d$ distinct hashes, expect to see 1 with $\log d$ trailing zeros. Expect $m \approx \log d$. $m$ takes $\log \log d$ bits to store.

**Total Space:** $O\left(\frac{\log \log d}{\epsilon^2} + \log d\right)$ for an $\epsilon$ approximate count.
Flajolet-Martin (LogLog) algorithm and HyperLogLog.

<table>
<thead>
<tr>
<th>h(x₁)</th>
<th>1010010</th>
</tr>
</thead>
<tbody>
<tr>
<td>h(x₂)</td>
<td>1001100</td>
</tr>
<tr>
<td>h(x₃)</td>
<td>1001110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>h(xₙ)</td>
<td>1011000</td>
</tr>
</tbody>
</table>

Estimate the number of distinct elements based on the maximum number of trailing zeros \( m \).

With \( d \) distinct elements, roughly what do we expect \( m \) to be?

\[
\Pr(h(x_i) \text{ has } \log d \text{ trailing zeros}) = \frac{1}{2^{\log d}} = \frac{1}{d}.
\]

So with \( d \) distinct hashes, expect to see 1 with \( \log d \) trailing zeros. Expect \( m \approx \log d \). \( m \) takes \( \log \log d \) bits to store.

**Total Space:** \( O \left( \frac{\log \log d}{\epsilon^2} + \log d \right) \) for an \( \epsilon \) approximate count.

**Note:** Careful averaging of estimates from multiple hash functions.
Using HyperLogLog to count 1 billion distinct items with 2% accuracy:

\[
\text{space used} = O\left(\frac{\log \log d}{\epsilon^2} + \log d\right)
\]
Using HyperLogLog to count 1 billion distinct items with 2% accuracy:

\[
\text{space used} = O\left(\frac{\log \log d}{\epsilon^2} + \log d\right)
\]

\[
= 1.04 \cdot \left\lceil \log_2 \log_2 d \right\rceil + \left\lfloor \log_2 d \right\rfloor \text{ bits}^1
\]

1. 1.04 is the constant in the HyperLogLogLog analysis. Not important!
Using HyperLogLog to count 1 billion distinct items with 2% accuracy:

\[
\text{space used} = O\left(\frac{\log \log d}{\epsilon^2} + \log d\right)
\]

\[
= \frac{1.04 \cdot \lceil \log_2 \log_2 d \rceil}{\epsilon^2} + \lceil \log_2 d \rceil \text{ bits}^1
\]

\[
= \frac{1.04 \cdot 5}{.02^2} + 30 = 13030 \text{ bits} \approx 1.6 \text{ kB}!
\]

1. 1.04 is the constant in the HyperLogLog analysis. Not important!
Using HyperLogLog to count 1 billion distinct items with 2% accuracy:

\[
\text{space used} = O \left( \frac{\log \log d}{\epsilon^2} + \log d \right)
\]

\[
= \frac{1.04 \cdot \lceil \log_2 \log_2 d \rceil}{\epsilon^2} + \lceil \log_2 d \rceil \text{ bits}^1
\]

\[
= \frac{1.04 \cdot 5}{.02^2} + 30 = 13030 \text{ bits} \approx 1.6 \text{ kB}!
\]

**Mergeable Sketch:** Consider the case (essentially always in practice) that the items are processed on different machines.

1. 1.04 is the constant in the HyperLogLogLog analysis. Not important!
Using HyperLogLog to count 1 billion distinct items with 2% accuracy:

\[
\text{space used} = O\left( \frac{\log \log d}{\epsilon^2} + \log d \right)
\]

\[
= \frac{1.04 \cdot \lceil \log_2 \log_2 d \rceil}{\epsilon^2} + \lceil \log_2 d \rceil \text{ bits}\]

\[
= \frac{1.04 \cdot 5}{0.02^2} + 30 = 13030 \text{ bits} \approx 1.6 \text{ kB!}
\]

**Mergeable Sketch:** Consider the case (essentially always in practice) that the items are processed on different machines.

- Given data structures (sketches) \( HLL(x_1, \ldots, x_n), HLL(y_1, \ldots, y_n) \) it is easy to merge them to give \( HLL(x_1, \ldots, x_n, y_1, \ldots, y_n) \).

---

1. 1.04 is the constant in the HyperLogLog analysis. Not important!
Using HyperLogLog to count 1 billion distinct items with 2% accuracy:

\[
\text{space used} = O \left( \frac{\log \log d}{\epsilon^2} + \log d \right) = 1.04 \cdot \left\lceil \log_2 \log_2 d \right\rceil + \left\lceil \log_2 d \right\rceil \text{ bits} = \frac{1.04 \cdot 5}{0.02^2} + 30 = 13030 \text{ bits} \approx 1.6 \text{ kB}!
\]

**Mergeable Sketch:** Consider the case (essentially always in practice) that the items are processed on different machines.

- Given data structures (sketches) \( HLL(x_1, \ldots, x_n), HLL(y_1, \ldots, y_n) \) it is easy to merge them to give \( HLL(x_1, \ldots, x_n, y_1, \ldots, y_n) \). How?

1. 1.04 is the constant in the HyperLogLog analysis. Not important!
Using HyperLogLog to count 1 billion distinct items with 2% accuracy:

\[
\text{space used} = O\left(\frac{\log \log d}{\epsilon^2} + \log d\right)
\]

\[
= 1.04 \cdot \lceil \log_2 \log_2 d \rceil + \lceil \log_2 d \rceil \text{ bits}\]

\[
= 1.04 \cdot 5.02 + 30 = 13030 \text{ bits} \approx 1.6 \text{ kB!}
\]

**Mergeable Sketch:** Consider the case (essentially always in practice) that the items are processed on different machines.

- Given data structures (sketches) \(HLL(x_1, \ldots, x_n), HLL(y_1, \ldots, y_n)\) it is easy to merge them to give \(HLL(x_1, \ldots, x_n, y_1, \ldots, y_n)\). How?
- Set the maximum \# of trailing zeros to the maximum in the two sketches.

1. 1.04 is the constant in the HyperLogLog analysis. Not important!
Questions on distinct elements counting?
**Jaccard Index:** A similarity measure between two sets.

\[
J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\# \text{ shared elements}}{\# \text{ total elements}}.
\]

Natural measure for similarity between bit strings – interpret an \( n \) bit string as a set, containing the elements corresponding the positions of its ones.

\[
J(x, y) = \frac{\# \text{ shared ones}}{\text{total ones}}.
\]
SEARCH WITH JACCARD SIMILARITY

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\text{# shared elements}}{\text{# total elements}}.$$  

Want Fast Implementations For:

- **Near Neighbor Search:** Have a database of \( n \) sets/bit strings and given a set \( A \), want to find if it has high Jaccard similarity to anything in the database. \( \Omega(n) \) time with a linear scan.

- **All-pairs Similarity Search:** Have \( n \) different sets/bit strings and want to find all pairs with high Jaccard similarity. \( \Omega(n^2) \) time if we check all pairs explicitly.

Will speed up via randomized locality sensitive hashing.
J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\# \text{ shared elements}}{\# \text{ total elements}}.

Want Fast Implementations For:

- **Near Neighbor Search:** Have a database of \( n \) sets/bit strings and given a set \( A \), want to find if it has high Jaccard similarity to anything in the database. \( \Omega(n) \) time with a linear scan.

- **All-pairs Similarity Search:** Have \( n \) different sets/bit strings and want to find all pairs with high Jaccard similarity. \( \Omega(n^2) \) time if we check all pairs explicitly.

Will speed up via randomized **locality sensitive hashing**.

What approaches might you use here to speed up search?
Document Similarity:

- E.g., to detect plagiarism, copyright infringement, duplicate webpages, spam.
- Use Shingling + Jaccard similarity.
Document Similarity:

- E.g., to detect plagiarism, copyright infringement, duplicate webpages, spam.
- Use Shingling + Jaccard similarity. ($n$-grams, $k$-mers)
Document Similarity:

- E.g., to detect plagiarism, copyright infringement, duplicate webpages, spam.
- Use Shingling + Jaccard similarity. \((n\text{-grams, } k\text{-mers})\)
Online recommendation systems are often based on **collaborative filtering**. Simplest approach: find similar users and make recommendations based on those users.
Online recommendation systems are often based on **collaborative filtering**. Simplest approach: find similar users and make recommendations based on those users.

- **Twitter**: represent a user as the set of accounts they follow. Match similar users based on the Jaccard similarity of these sets. Recommend that you follow accounts followed by similar users.
Online recommendation systems are often based on **collaborative filtering**. Simplest approach: find similar users and make recommendations based on those users.

- **Twitter**: represent a user as the set of accounts they follow. Match similar users based on the Jaccard similarity of these sets. Recommend that you follow accounts followed by similar users.

- **Netflix**: look at sets of movies watched. Amazon: look at products purchased, etc.
**Entity Resolution Problem:** Want to combine records from multiple data sources that refer to the same entities.
Entity Resolution Problem: Want to combine records from multiple data sources that refer to the same entities.

- E.g. data on individuals from voting registrations, property records, and social media accounts. Names and addresses may not exactly match, due to typos, nicknames, moves, etc.
- Still want to match records that all refer to the same person using all pairs similarity search.

See Section 3.8.2 of *Mining Massive Datasets* for a discussion of a real world example involving 1 million customers. Naively this would be $1000000^2 \approx 500$ billion pairs of customers to check!
**Entity Resolution Problem:** Want to combine records from multiple data sources that refer to the same entities.

- E.g. data on individuals from voting registrations, property records, and social media accounts. Names and addresses may not exactly match, due to typos, nicknames, moves, etc.

- Still want to match records that all refer to the same person using all pairs similarity search.

See Section 3.8.2 of *Mining Massive Datasets* for a discussion of a real world example involving 1 million customers. Naively this would be \( \binom{1000000}{2} \approx 500 \text{ billion pairs of customers to check!} \)
Many applications to spam/fraud detection. E.g.
Many applications to spam/fraud detection. E.g.

- **Fake Reviews**: Very common on websites like Amazon. Detection often looks for (near) duplicate reviews on similar products, which have been copied. ‘Near duplicate’ measured with shingles + Jaccard similarity.
Many applications to spam/fraud detection. E.g.

- **Fake Reviews**: Very common on websites like Amazon. Detection often looks for (near) duplicate reviews on similar products, which have been copied. ‘Near duplicate’ measured with shingles + Jaccard similarity.

- **Lateral phishing**: Phishing emails sent to addresses at a business coming from a legitimate email address at the same business that has been compromised.
  - One method of detection looks at the recipient list of an email and checks if it has small Jaccard similarity with any previous recipient lists. If not, the email is flagged as possible spam.
Goal: Speed up Jaccard similarity search (near neighbor and all-pairs similarity search).

Minhashing

- Let $h: U \rightarrow [0, 1]$ be a random hash function
- $s := 1$
- For $x_1, \ldots, x_{|A|} \in A$
  - $s := \min(s, h(x_k))$
- Return $s$

Identical to our distinct elements sketch!
**Goal:** Speed up Jaccard similarity search (near neighbor and all-pairs similarity search).

**Strategy:** Use random hashing to map each set to a very compressed representation. Jaccard similarity can be estimated from these representations.
**Goal:** Speed up Jaccard similarity search (near neighbor and all-pairs similarity search).

**Strategy:** Use random hashing to map each set to a very compressed representation. Jaccard similarity can be estimated from these representations.

**MinHash(A):** [Andrei Broder, 1997 at Altavista]

- Let $h : U \rightarrow [0, 1]$ be a random hash function
- $s := 1$
- For $x_1, \ldots, x_{|A|} \in A$
  - $s := \min(s, h(x_k))$
- Return $s$
**Goal:** Speed up Jaccard similarity search (near neighbor and all-pairs similarity search).

**Strategy:** Use random hashing to map each set to a very compressed representation. Jaccard similarity can be estimated from these representations.

**MinHash(A):** [Andrei Broder, 1997 at Altavista]

- Let $h: U \rightarrow [0, 1]$ be a random hash function
- $s := 1$
- For $x_1, \ldots, x_{|A|} \in A$
  - $s := \min(s, h(x_k))$
- Return $s$
**Goal:** Speed up Jaccard similarity search (near neighbor and all-pairs similarity search).

**Strategy:** Use random hashing to map each set to a very compressed representation. Jaccard similarity can be estimated from these representations.

**MinHash(A):** [Andrei Broder, 1997 at Altavista]

- Let \( h : U \rightarrow [0, 1] \) be a random hash function
- \( s := 1 \)
- For \( x_1, \ldots, x_{|A|} \in A \)
  - \( s := \min(s, h(x_k)) \)
- Return \( s \) Identical to our distinct elements sketch!
For two sets $A$ and $B$, what is $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$?
For two sets $A$ and $B$, what is $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$?

- Since we are hashing into the continuous range $[0, 1]$, we will never have $h(x) = h(y)$ for $x \neq y$ (i.e., no spurious collisions)
For two sets $A$ and $B$, what is $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$?

- Since we are hashing into the continuous range $[0, 1]$, we will never have $h(x) = h(y)$ for $x \neq y$ (i.e., no spurious collisions)
For two sets $A$ and $B$, what is $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$?

- Since we are hashing into the continuous range $[0, 1]$, we will never have $h(x) = h(y)$ for $x \neq y$ (i.e., no spurious collisions).
For two sets $A$ and $B$, what is $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$?

- Since we are hashing into the continuous range $[0, 1]$, we will never have $h(x) = h(y)$ for $x \neq y$ (i.e., no spurious collisions)
For two sets $A$ and $B$, what is $\text{Pr}(\text{MinHash}(A) = \text{MinHash}(B))$?

- Since we are hashing into the continuous range $[0, 1]$, we will never have $h(x) = h(y)$ for $x \neq y$ (i.e., no spurious collisions)
For two sets $A$ and $B$, what is $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$?

- Since we are hashing into the continuous range $[0, 1]$, we will never have $h(x) = h(y)$ for $x \neq y$ (i.e., no spurious collisions).

- $\text{MinHash}(A) = \text{MinHash}(B)$ only if an item in $A \cap B$ has the minimum hash value in both sets.
For two sets $A$ and $B$, what is $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$?

**Claim:** $\text{MinHash}(A) = \text{MinHash}(B)$ only if an item in $A \cap B$ has the minimum hash value in both sets.
For two sets $A$ and $B$, what is $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$?

**Claim:** $\text{MinHash}(A) = \text{MinHash}(B)$ only if an item in $A \cap B$ has the minimum hash value in both sets.

$\Pr(\text{MinHash}(A) = \text{MinHash}(B)) = \frac{|A \cap B|}{\text{total # items hashed}} = \frac{|A \cap B|}{|A \cup B|} = J(A, B)$. 

$Pr(\text{MinHash}(A) = \text{MinHash}(B)) =$
For two sets $A$ and $B$, what is $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$?

**Claim:** $\text{MinHash}(A) = \text{MinHash}(B)$ only if an item in $A \cap B$ has the minimum hash value in both sets.

$$\Pr(\text{MinHash}(A) = \text{MinHash}(B)) = \frac{|A \cap B|}{\text{total } \# \text{ items hashed}}$$
For two sets $A$ and $B$, what is $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$?

**Claim:** $\text{MinHash}(A) = \text{MinHash}(B)$ only if an item in $A \cap B$ has the minimum hash value in both sets.

\[
\Pr(\text{MinHash}(A) = \text{MinHash}(B)) = \frac{|A \cap B|}{\text{total } \# \text{ items hashed}} = \frac{|A \cap B|}{|A \cup B|}
\]
For two sets $A$ and $B$, what is $\Pr(\text{MinHash}(A) = \text{MinHash}(B))$?

**Claim:** $\text{MinHash}(A) = \text{MinHash}(B)$ only if an item in $A \cap B$ has the minimum hash value in both sets.

$$\Pr(\text{MinHash}(A) = \text{MinHash}(B)) = \frac{|A \cap B|}{\text{total \# items hashed}} = \frac{|A \cap B|}{|A \cup B|} = J(A, B).$$
**Upshot:** MinHash reduces estimating the Jaccard similarity to checking equality of a *single number*.

\[ \Pr(\text{MinHash}(A) = \text{MinHash}(B)) = J(A, B). \]
Upshot: MinHash reduces estimating the Jaccard similarity to checking equality of a *single number*.

\[
\Pr(\text{MinHash}(A) = \text{MinHash}(B)) = J(A, B).
\]

- An instance of *locality sensitive hashing* (LSH).
**Upshot:** MinHash reduces estimating the Jaccard similarity to checking equality of a *single number.*

\[ \Pr(\text{MinHash}(A) = \text{MinHash}(B)) = J(A, B). \]

- An instance of **locality sensitive hashing (LSH).**
- A hash function where the collision probability is higher when two inputs are more similar (can design different functions for different similarity metrics.)
Upshot: MinHash reduces estimating the Jaccard similarity to checking equality of a *single number*.

\[
\Pr(\text{MinHash}(A) = \text{MinHash}(B)) = J(A, B).
\]

- An instance of **locality sensitive hashing** (LSH).
- A hash function where the collision probability is higher when two inputs are more similar (can design different functions for different similarity metrics.)
**Upshot:** MinHash reduces estimating the Jaccard similarity to checking equality of a *single number*.

\[ \Pr(\text{MinHash}(A) = \text{MinHash}(B)) = J(A, B). \]

- An instance of *locality sensitive hashing* (LSH).
- A hash function where the collision probability is higher when two inputs are more similar (can design different functions for different similarity metrics.)
How does locality sensitive hashing (LSH) help with similarity search?

[Diagram of locality sensitive hash function with data points and hash buckets]

- **Near Neighbor Search:**
  - Given item $x$, compute $h(x)$.
  - Only search for similar items in the $h(x)$ bucket of the hash table.

- **All-pairs Similarity Search:**
  - Scan through all buckets of the hash table and look for similar pairs within each bucket.
How does locality sensitive hashing (LSH) help with similarity search?

- **Near Neighbor Search**: Given item $x$, compute $h(x)$. Only search for similar items in the $h(x)$ bucket of the hash table.
How does locality sensitive hashing (LSH) help with similarity search?

- **Near Neighbor Search**: Given item $x$, compute $h(x)$. Only search for similar items in the $h(x)$ bucket of the hash table.

- **All-pairs Similarity Search**: Scan through all buckets of the hash table and look for similar pairs within each bucket.
Questions?