MinHash maps set to \([0, 1]\) such that for any two sets:

\[
\Pr(\text{MinHash}(A) = \text{MinHash}(B)) = J(A, B) = \frac{|A \cap B|}{|A \cup B|}
\]

Basic approach to speeding up nearest neighbor search:

- Combine MinHash with a function \(g: [0, 1] \to [m]\)
- Store each set \(A\) in location \(g(\text{MinHash}(A))\) of a table.
- When we want to find sets similar to \(B\), only compare \(B\) to sets stored in location \(g(\text{MinHash}(B))\) of the table.
- Assuming \(g\) has no collisions, the probability that one of the stored sets \(A\) is found in this location is \(J(A, B)\), i.e., near one for similar sets and near zero for dissimilar sets.
LOCALITY SENSITIVE HASHING

- MinHash maps set to $[0, 1]$ such that for any two sets:

$$\Pr(\text{MinHash}(A) = \text{MinHash}(B)) = J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$
• MinHash maps set to [0, 1] such that that for any two sets:

$$\Pr(\text{MinHash}(A) = \text{MinHash}(B)) = J(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

• Basic approach to speeding up nearest neighbor search:
  • Combine MinHash with a function $g : [0, 1] \rightarrow [m]$
  • Store each set $A$ in location $g(\text{MinHash}(A))$ of a table.
  • When we want to find sets similar to $B$, only compare $B$ to sets stored in location $g(\text{MinHash}(B))$ of the table.
  • Assuming $g$ has no collisions, the probability that one of the stored sets $A$ is found in this location is $J(A, B)$, i.e., near one for similar sets and near zero for dissimilar sets.
We want to balance a small probability of false negatives (a high hit rate) with a small probability of false positives (a small query time.)
We want to balance a small probability of false negatives (a high hit rate) with a small probability of false positives (a small query time.)

Create $t$ hash tables. Each is indexed into not with a single MinHash value, but with $r$ values, appended together. A length $r$ signature. Probability that $x$ and $y$ with Jaccard similarity $J(x, y) = s$ match in $\geq 1$ repetition is: $1 - (1 - s^r)^t$. 
Repetition and s-curve tuning can be used for fast similarity search with other similarity metrics:

\[
\cos(\theta(x, y)) = 1 \quad \text{when} \quad \theta(x, y) = 0^\circ \quad \text{and} \quad \cos(\theta(x, y)) = 0 \quad \text{when} \quad \theta(x, y) = 90^\circ, \quad \text{and} \quad \cos(\theta(x, y)) = -1 \quad \text{when} \quad \theta(x, y) = 180^\circ.
\]
Repetition and s-curve tuning can be used for fast similarity search with other similarity metrics:

- LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.
Repetition and s-curve tuning can be used for fast similarity search with other similarity metrics:

- LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.
Repetition and s-curve tuning can be used for fast similarity search with other similarity metrics:

- LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.
Repetition and s-curve tuning can be used for fast similarity search with other similarity metrics:

- LSH schemes exist for many similarity/distance measures: hamming distance, **cosine similarity**, etc.

**Cosine Similarity:** $\cos(\theta(x, y))$
Repetition and $s$-curve tuning can be used for fast similarity search with other similarity metrics:

- LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.

**Cosine Similarity:** $\cos(\theta(x, y))$

- $\cos(\theta(x, y)) = 1$ when $\theta(x, y) = 0^\circ$ and $\cos(\theta(x, y)) = 0$ when $\theta(x, y) = 90^\circ$, and $\cos(\theta(x, y)) = -1$ when $\theta(x, y) = 180^\circ$
Repetition and s-curve tuning can be used for fast similarity search with other similarity metrics:

- LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.

**Cosine Similarity:** \( \cos(\theta(x, y)) = \frac{\langle x, y \rangle}{\|x\|_2 \cdot \|y\|_2} \).

- \( \cos(\theta(x, y)) = 1 \) when \( \theta(x, y) = 0^\circ \) and \( \cos(\theta(x, y)) = 0 \) when \( \theta(x, y) = 90^\circ \), and \( \cos(\theta(x, y)) = -1 \) when \( \theta(x, y) = 180^\circ \)
SimHash Algorithm: LSH for cosine similarity.
SimHash Algorithm: LSH for cosine similarity.
SimHash Algorithm: LSH for cosine similarity.
SimHash Algorithm: LSH for cosine similarity.

SimHash(x) = 1

SimHash(x) = -1

random plane
SimHash Algorithm: LSH for cosine similarity.

SimHash$(x) = \text{sign}(\langle x, t \rangle)$ for a random vector $t$. 

SimHash$(x) = 1$

SimHash$(x) = -1$
**SimHash Algorithm:** LSH for cosine similarity.

\[ \text{SimHash}(x) = \text{sign}(\langle x, t \rangle) \]

for a random vector \( t \).

What is \( \Pr [\text{SimHash}(x) = \text{SimHash}(y)] \)?
What is $\Pr[\text{SimHash}(x) = \text{SimHash}(y)]$?
What is $\Pr[\text{SimHash}(x) = \text{SimHash}(y)]$?

$\text{SimHash}(x) \neq \text{SimHash}(y)$ when the plane separates $x$ from $y$. 

\[
\begin{align*}
\Pr[\text{SimHash}(x) = \text{SimHash}(y)] &= \theta(x, y) \\
\Pr[\text{SimHash}(x) = \text{SimHash}(y)] &= 1 - \theta(x, y) \\
&\approx \cos \theta 
\end{align*}
\] for small $\theta$. 

What is $\Pr[\text{SimHash}(x) = \text{SimHash}(y)]$?

$\text{SimHash}(x) \neq \text{SimHash}(y)$ when the plane separates $x$ from $y$. 

- $\Pr[\text{SimHash}(x) \neq \text{SimHash}(y)] = \theta(x, y)$
- $\Pr[\text{SimHash}(x) = \text{SimHash}(y)] = 1 - \theta(x, y) \approx \cos \theta$ for small $\theta$. 
What is $\Pr [\text{SimHash}(x) = \text{SimHash}(y)]$?

$\text{SimHash}(x) \neq \text{SimHash}(y)$ when the plane separates $x$ from $y$.

- $\Pr [\text{SimHash}(x) \neq \text{SimHash}(y)] = \theta(x, y) / 180$
- $\Pr [\text{SimHash}(x) = \text{SimHash}(y)] = 1 - \theta(x, y) / 180 \approx \cos \theta$ for small $\theta$. 
SIMHASH FOR COSINE SIMILARITY

What is $\Pr [\text{SimHash}(x) = \text{SimHash}(y)]$?

$\text{SimHash}(x) \neq \text{SimHash}(y)$ when the plane separates $x$ from $y$.

\begin{itemize}
  \item $\Pr [\text{SimHash}(x) \neq \text{SimHash}(y)] = \frac{\theta(x,y)}{180}$
  \item $\Pr [\text{SimHash}(x) = \text{SimHash}(y)] = 1 - \frac{\theta(x,y)}{180} \approx \cos \theta$ for small $\theta$.
\end{itemize}
Questions on MinHash and Locality Sensitive Hashing?
The Frequent Items Data Stream Problem

$k$-Frequent Items (Heavy-Hitters) Problem: Consider a stream of $n$ items $x_1, \ldots, x_n$ (with possible duplicates). Return any item at appears at least $\frac{n}{k}$ times.
**k-Frequent Items (Heavy-Hitters) Problem:** Consider a stream of $n$ items $x_1, \ldots, x_n$ (with possible duplicates). Return any item at appears at least $\frac{n}{k}$ times.

\[ \begin{array}{cccccccc}
 x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 \\
 5 & 12 & 3 & 3 & 4 & 5 & 5 & 10 & 3 \\
\end{array} \]
**k-Frequent Items (Heavy-Hitters) Problem:** Consider a stream of $n$ items $x_1, \ldots, x_n$ (with possible duplicates). Return any item at appears at least $\frac{n}{k}$ times.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>$x_7$</th>
<th>$x_8$</th>
<th>$x_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>
**k-Frequent Items (Heavy-Hitters) Problem:** Consider a stream of $n$ items $x_1, \ldots, x_n$ (with possible duplicates). Return any item at appears at least $\frac{n}{k}$ times.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>$x_7$</th>
<th>$x_8$</th>
<th>$x_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

- What is the maximum number of items that must be returned?
  
  a) $n$  
  b) $k$  
  c) $\frac{n}{k}$  
  d) $\log n$
**k-Frequent Items (Heavy-Hitters) Problem:** Consider a stream of $n$ items $x_1, \ldots, x_n$ (with possible duplicates). Return any item at appears at least $\frac{n}{k}$ times.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>$x_7$</th>
<th>$x_8$</th>
<th>$x_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

- What is the maximum number of items that must be returned?
  
a) $n$  
b) $k$  
c) $n/k$  
d) $\log n$
**k-Frequent Items (Heavy-Hitters) Problem:** Consider a stream of $n$ items $x_1, \ldots, x_n$ (with possible duplicates). Return any item at appears at least $\frac{n}{k}$ times.

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>$x_7$</th>
<th>$x_8$</th>
<th>$x_9$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

- **What is the maximum number of items that must be returned?**
  a) $n$  b) $k$  c) $n/k$  d) $\log n$

- **Trivial with $O(n)$ space:** Store the count for each item and return the one that appears $\geq n/k$ times.

- **Can we do it with less space?** I.e., without storing all $n$ items?
Applications of Frequent Items:
Applications of Frequent Items:

- Finding top/viral items (i.e., products on Amazon, videos watched on Youtube, Google searches, etc.)
Applications of Frequent Items:

• Finding top/viral items (i.e., products on Amazon, videos watched on Youtube, Google searches, etc.)

• Finding very frequent IP addresses sending requests (to detect DoS attacks/network anomalies).
Applications of Frequent Items:

- Finding top/viral items (i.e., products on Amazon, videos watched on Youtube, Google searches, etc.)
- Finding very frequent IP addresses sending requests (to detect DoS attacks/network anomalies).
- ‘Iceberg queries’ for all items in a database with frequency above some threshold.
Applications of Frequent Items:

- Finding top/viral items (i.e., products on Amazon, videos watched on Youtube, Google searches, etc.)
- Finding very frequent IP addresses sending requests (to detect DoS attacks/network anomalies).
- ‘Iceberg queries’ for all items in a database with frequency above some threshold.

Generally want very fast detection, without having to scan through database/logs. That is we want to maintain a running list of frequent items that appear in a stream.
**Issue:** No algorithm using $o(n)$ space can output just the items with frequency $\geq n/k$. Hard to tell between an item with frequency $n/k$ (should be output) and $n/k - 1$ (should not be output).
**Issue:** No algorithm using $o(n)$ space can output just the items with frequency $\geq n/k$. Hard to tell between an item with frequency $n/k$ (should be output) and $n/k - 1$ (should not be output).

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>...</th>
<th>$x_{n-k+1}$</th>
<th>...</th>
<th>$x_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>12</td>
<td>9</td>
<td>27</td>
<td>4</td>
<td>101</td>
<td>...</td>
<td>3</td>
<td>...</td>
<td>3</td>
</tr>
</tbody>
</table>

$n/k-1$ occurrences
**Issue:** No algorithm using $o(n)$ space can output just the items with frequency $\geq n/k$. Hard to tell between an item with frequency $n/k$ (should be output) and $n/k - 1$ (should not be output).

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>...</th>
<th>$x_{n-n/k+1}$</th>
<th>...</th>
<th>$x_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>12</td>
<td>9</td>
<td>27</td>
<td>4</td>
<td>101</td>
<td>...</td>
<td>3</td>
<td>...</td>
<td>3</td>
</tr>
</tbody>
</table>

$n/k$-1 occurrences

**$(\epsilon, k)$-Frequent Items Problem:** Consider a stream of $n$ items $x_1, \ldots, x_n$. Return a set $F$ of items, including all items that appear at least $\frac{n}{k}$ times and only items that appear at least $(1 - \epsilon) \cdot \frac{n}{k}$ times.
**Issue:** No algorithm using $o(n)$ space can output just the items with frequency $\geq n/k$. Hard to tell between an item with frequency $n/k$ (should be output) and $n/k - 1$ (should not be output).

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>$x_5$</th>
<th>$x_6$</th>
<th>...</th>
<th>$x_{n-k+1}$</th>
<th>...</th>
<th>$x_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>12</td>
<td>9</td>
<td>27</td>
<td>4</td>
<td>101</td>
<td></td>
<td>3</td>
<td></td>
<td>3</td>
</tr>
</tbody>
</table>

(\(\epsilon, k\))-**Frequent Items Problem:** Consider a stream of \(n\) items \(x_1, \ldots, x_n\). Return a set \(F\) of items, including all items that appear at least \(\frac{n}{k}\) times and only items that appear at least \((1 - \epsilon) \cdot \frac{n}{k}\) times.

- An example of relaxing to a ‘promise problem’: for items with frequencies in \([((1 - \epsilon) \cdot \frac{n}{k}, \frac{n}{k}]\) no output guarantee.
**Today:** Count-min sketch – a random hashing based method closely related to bloom filters.
**Today:** Count-min sketch – a random hashing based method closely related to bloom filters.

\[ A[h(x)] \] to estimate \( f(x) = \left| \{ i : x_i = x \} \right| \), the frequency of \( x \) in the stream.

**random hash function** \( h \)

**m length array** \( A \)

\[
\begin{array}{ccccccccccc}
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\]
**Today:** Count-min sketch – a random hashing based method closely related to bloom filters.
**Today:** Count-min sketch – a random hashing based method closely related to bloom filters.

![Count-min sketch diagram](image)

- **Random hash function** $h$ maps elements $x_1, x_2, x_3, x_4, \ldots, x_n$ to indices.
- **m length array** $A = [1, 1, 1, 0, 0, 0, 0, 0, 0, 0]$ represents the counts of elements in the stream.
**Today:** Count-min sketch – a random hashing based method closely related to bloom filters.

A \[ h(x) \] to estimate \( f(x) = |\{ i : x_i = x \}| \), the frequency of \( x \) in the stream.
**Today:** Count-min sketch – a random hashing based method closely related to bloom filters.

Random hash function \( h \)

\( m \) length array \( A \)

\[
\begin{array}{cccccccc}
1 & 2 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0
\end{array}
\]
**Today:** Count-min sketch – a random hashing based method closely related to bloom filters.

![Diagram of count-min sketch]

- **Random hash function** $h$
- **m length array** $A$
- $x_1, x_2, x_3, x_4, \ldots, x_n$
**Today:** Count-min sketch – a random hashing based method closely related to bloom filters.

Will use $A[h(x)]$ to estimate $f(x) = |\{i : x_i = x\}|$, the frequency of $x$ in the stream.
Use $A[h(x)]$ to estimate $f(x)$.

**Claim 1:** We always have $A[h(x)] \geq f(x)$. Why?

---

$f(x)$: frequency of $x$ in the stream (i.e., number of items equal to $x$). $h$: random hash function. $m$: size of Count-min sketch array.
Use $A[h(x)]$ to estimate $f(x)$.

**Claim 1:** We always have $A[h(x)] \geq f(x)$. Why?

- $A[h(x)]$ counts the number of occurrences of any $y$ with $h(y) = h(x)$, including $x$ itself.

---

$f(x)$: frequency of $x$ in the stream (i.e., number of items equal to $x$). $h$: random hash function. $m$: size of Count-min sketch array.
Use $A[h(x)]$ to estimate $f(x)$.

**Claim 1:** We always have $A[h(x)] \geq f(x)$. Why?

- $A[h(x)]$ counts the number of occurrences of any $y$ with $h(y) = h(x)$, including $x$ itself.
- $A[h(x)] = f(x) + \sum_{y \neq x : h(y) = h(x)} f(y)$.

---

$f(x)$: frequency of $x$ in the stream (i.e., number of items equal to $x$). $h$: random hash function. $m$: size of Count-min sketch array.
COUNT-MIN SKETCH ACCURACY

\[ A[h(x)] = f(x) + \sum_{y \neq x: h(y) = h(x)} f(y) \]

\[ \text{error in frequency estimate} \]

- **f(x):** frequency of \( x \) in the stream (i.e., number of items equal to \( x \)).
- **h:** random hash function.
- **m:** size of Count-min sketch array.
$$A[h(x)] = f(x) + \sum_{y \neq x: h(y) = h(x)} f(y) .$$

**Expected Error:**

$$\mathbb{E} \left[ \sum_{y \neq x: h(y) = h(x)} f(y) \right] =$$

$f(x)$: frequency of $x$ in the stream (i.e., number of items equal to $x$). $h$: random hash function. $m$: size of Count-min sketch array.
**COUNT-MIN SKETCH ACCURACY**

\[ A[h(x)] = f(x) + \sum_{y \neq x \text{ and } h(y) = h(x)} f(y) \]

**Expected Error:**

\[
\mathbb{E}\left[ \sum_{y \neq x \text{ and } h(y) = h(x)} f(y) \right] = \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y)
\]

---

- **f(x):** frequency of x in the stream (i.e., number of items equal to x).
- **h:** random hash function.
- **m:** size of Count-min sketch array.
COUNT-MIN SKETCH ACCURACY

\[ A[h(x)] = f(x) + \sum_{y \neq x : h(y) = h(x)} f(y) \]

Expected Error:

\[
\mathbb{E} \left[ \sum_{y \neq x : h(y) = h(x)} f(y) \right] = \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y) \\
\leq \sum_{y \neq x} \frac{1}{m} \cdot f(y)
\]

- \( f(x) \): frequency of \( x \) in the stream (i.e., number of items equal to \( x \)).
- \( h \): random hash function.
- \( m \): size of Count-min sketch array.
COUNT-MIN SKETCH ACCURACY

\[ A[h(x)] = f(x) + \sum_{y \neq x : h(y) = h(x)} f(y) \].

**Expected Error:**

\[
\mathbb{E} \left[ \sum_{y \neq x : h(y) = h(x)} f(y) \right] = \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y)
\leq \sum_{y \neq x} \frac{1}{m} \cdot f(y) = \frac{1}{m} \cdot (n - f(x)) \leq \frac{n}{m}
\]

\( f(x) \): frequency of \( x \) in the stream (i.e., number of items equal to \( x \)). \( h \): random hash function. \( m \): size of Count-min sketch array.
COUNT-MIN SKETCH ACCURACY

\[ A[h(x)] = f(x) + \sum_{y \neq x: h(y) = h(x)} f(y) \]

**Expected Error:**

\[
\mathbb{E} \left[ \sum_{y \neq x: h(y) = h(x)} f(y) \right] = \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y)
\]

\[
\leq \sum_{y \neq x} \frac{1}{m} \cdot f(y) = \frac{1}{m} \cdot (n - f(x)) \leq \frac{n}{m}
\]

What is a bound on probability that the error is \( \geq \frac{2n}{m} \)?

\( f(x) \): frequency of \( x \) in the stream (i.e., number of items equal to \( x \)). \( h \): random hash function. \( m \): size of Count-min sketch array.
**COUNT-MIN SKETCH ACCURACY**

\[ A[h(x)] = f(x) + \sum_{y \neq x : h(y) = h(x)} f(y) \]

**Expected Error:**

\[
\mathbb{E} \left[ \sum_{y \neq x : h(y) = h(x)} f(y) \right] = \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y)
\leq \sum_{y \neq x} \frac{1}{m} \cdot f(y) = \frac{1}{m} \cdot (n - f(x)) \leq \frac{n}{m}
\]

What is a bound on probability that the error is \( \geq \frac{2n}{m} \)?

**Markov’s inequality:** \( \Pr \left[ \sum_{y \neq x : h(y) = h(x)} f(y) \geq \frac{2n}{m} \right] \leq \frac{1}{2} \).

\( f(x) \): frequency of \( x \) in the stream (i.e., number of items equal to \( x \)). \( h \): random hash function. \( m \): size of Count-min sketch array.
\[ A[h(x)] = f(x) + \sum_{y \neq x: h(y) = h(x)} f(y). \]

**Expected Error:**

\[
\mathbb{E} \left[ \sum_{y \neq x: h(y) = h(x)} f(y) \right] = \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y)
\leq \sum_{y \neq x} \frac{1}{m} \cdot f(y) = \frac{1}{m} \cdot (n - f(x)) \leq \frac{n}{m}
\]

What is a bound on probability that the error is \( \geq \frac{2n}{m} \)?

**Markov’s inequality:**

\[
\Pr \left[ \sum_{y \neq x: h(y) = h(x)} f(y) \geq \frac{2n}{m} \right] \leq \frac{1}{2}.
\]

What property of \( h \) is required to show this bound? a) fully random b) pairwise independent c) 2-universal d) locality sensitive

\( f(x) \): frequency of \( x \) in the stream (i.e., number of items equal to \( x \)). \( h \): random hash function. \( m \): size of Count-min sketch array.
**COUNT-MIN SKETCH ACCURACY**

\[ A[h(x)] = f(x) + \sum_{y \neq x : h(y) = h(x)} f(y) . \]

**Expected Error:**

\[
\mathbb{E} \left[ \sum_{y \neq x : h(y) = h(x)} f(y) \right] = \sum_{y \neq x} \Pr(h(y) = h(x)) \cdot f(y) \\
\leq \sum_{y \neq x} \frac{1}{m} \cdot f(y) = \frac{1}{m} \cdot (n - f(x)) \leq \frac{n}{m}
\]

What is a bound on probability that the error is \( \geq \frac{2n}{m} \)?

**Markov’s inequality:**

\[
\Pr \left[ \sum_{y \neq x : h(y) = h(x)} f(y) \geq \frac{2n}{m} \right] \leq \frac{1}{2}.
\]

What property of \( h \) is required to show this bound?  

a) fully random  

b) pairwise independent  

c) 2-universal  

d) locality sensitive

\( f(x) \): frequency of \( x \) in the stream (i.e., number of items equal to \( x \)).  
\( h \): random hash function.  
\( m \): size of Count-min sketch array.
**Claim:** For any $x$, with probability at least $1/2$,

$$f(x) \leq A[h(x)] \leq f(x) + \frac{2n}{m}.$$
**Claim:** For any $x$, with probability at least $1/2$,

$$f(x) \leq A[h(x)] \leq f(x) + \frac{2n}{m}.$$ 

To solve the $(\epsilon, k)$-Frequent elements problem, set $m = \frac{2k}{\epsilon}$.

*f(x):* frequency of $x$ in the stream (i.e., number of items equal to $x$).  
*h:* random hash function.  
*m:* size of Count-min sketch array.
**Claim:** For any $x$, with probability at least $1/2$,

$$f(x) \leq A[h(x)] \leq f(x) + \frac{2n}{m}.$$ 

To solve the $(\epsilon, k)$-Frequent elements problem, set $m = \frac{2k}{\epsilon}$. How can we improve the success probability?

---

$f(x)$: frequency of $x$ in the stream (i.e., number of items equal to $x$). $h$: random hash function. $m$: size of Count-min sketch array.
**Claim:** For any $x$, with probability at least $1/2$,

$$f(x) \leq A[h(x)] \leq f(x) + \frac{2n}{m}.$$ 

To solve the $(\epsilon, k)$-Frequent elements problem, set $m = \frac{2k}{\epsilon}$. How can we improve the success probability? **Repetition.**

- $f(x)$: frequency of $x$ in the stream (i.e., number of items equal to $x$).
- $h$: random hash function.
- $m$: size of Count-min sketch array.
**COUNT-MIN SKETCH ACCURACY**

Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$. (count-min sketch)

Why min instead of mean or median? The minimum estimate is always the most accurate since they are all overestimates of the true frequency!

<table>
<thead>
<tr>
<th>$x_1$</th>
<th>$x_2$</th>
<th>$x_3$</th>
<th>$x_4$</th>
<th>...</th>
<th>$x_n$</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="https://via.placeholder.com/150" alt="Hash Functions" /></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

$t$ random hash functions $h_1, h_2, \ldots, h_t$

$t$ length $m$ arrays $A_1, A_2, \ldots, A_t$

<table>
<thead>
<tr>
<th>$A_1$</th>
<th>$A_2$</th>
<th>$A_t$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
<td>0 0 0 0 0 0 0 0 0 0 0 0</td>
</tr>
</tbody>
</table>

...
Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i(h_i(x))$. (count-min sketch)

Why min instead of mean or median? The minimum estimate is always the most accurate since they are all overestimates of the true frequency!
Estimate \( f(x) \) with \( \tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)] \). (count-min sketch)

Why min instead of mean or median?

The minimum estimate is always the most accurate since they are all overestimates of the true frequency!
Estimate $f(x)$ with $\hat{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$. (count-min sketch)

Why min instead of mean or median? The minimum estimate is always the most accurate since they are all overestimates of the true frequency!
Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$. (count-min sketch)
Estimate $f(x)$ with $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$. (count-min sketch)
Estimate $f(x)$ with $	ilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$. (count-min sketch)

Why min instead of mean or median?
### Count-Min Sketch Accuracy

Estimate \( f(x) \) with \( \tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)] \). (count-min sketch)

**Why min instead of mean or median?** The minimum estimate is always the most accurate since they are all overestimates of the true frequency!

<table>
<thead>
<tr>
<th>A_1</th>
<th>A_2</th>
<th>A_t</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>1</td>
<td>90</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>52</td>
</tr>
<tr>
<td>0</td>
<td>10</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>78</td>
<td>3</td>
</tr>
<tr>
<td>12</td>
<td>80</td>
<td>12</td>
</tr>
<tr>
<td>104</td>
<td>4</td>
<td>33</td>
</tr>
<tr>
<td>1</td>
<td>11</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>2</td>
</tr>
</tbody>
</table>

The diagram shows the process of estimating frequencies using the count-min sketch method. Each input value is hashed into a set of random hash functions, and the minimum value across all hash functions is used as the estimate.
Estimate $f(x)$ by $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$
Estimate $f(x)$ by $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$

- For every $x$ and $i \in [t]$, we know that for $m = \frac{2k}{\epsilon}$, with probability $\geq 1/2$:
  $$f(x) \leq A_i[h_i(x)] \leq f(x) + \frac{\epsilon n}{k}.$$
Estimate $f(x)$ by $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$

- For every $x$ and $i \in [t]$, we know that for $m = \frac{2k}{\epsilon}$, with probability $\geq 1/2$:
  $$f(x) \leq A_i[h_i(x)] \leq f(x) + \frac{\epsilon n}{k}.$$  

- What is $\Pr[f(x) \leq \tilde{f}(x) \leq f(x) + \frac{\epsilon n}{k}]$?
Estimate $f(x)$ by $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$

- For every $x$ and $i \in [t]$, we know that for $m = \frac{2k}{\epsilon}$, with probability $\geq 1/2$:
  \[ f(x) \leq A_i[h_i(x)] \leq f(x) + \frac{\epsilon n}{k}. \]

- What is $\Pr[f(x) \leq \tilde{f}(x) \leq f(x) + \frac{\epsilon n}{k}]$? $1 - 1/2^t$. 
Estimate $f(x)$ by $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$

- For every $x$ and $i \in [t]$, we know that for $m = \frac{2k}{\epsilon}$, with probability $\geq 1/2$:
  $$f(x) \leq A_i[h_i(x)] \leq f(x) + \frac{\epsilon n}{k}.$$  

- What is $\Pr[f(x) \leq \tilde{f}(x) \leq f(x) + \frac{\epsilon n}{k}]$? $1 - 1/2^t$.

- To get a good estimate with probability $\geq 1 - \delta$, set $t = \log(1/\delta)$. 

Estimate $f(x)$ by $\tilde{f}(x) = \min_{i \in [t]} A_i[h_i(x)]$