• Problem Set 1 is due Thursday in Gradescope.
• My office hours today are 1:15pm-2:15pm.
• Problem Set 1 is due Thursday in Gradescope.
• My office hours today are 1:15pm-2:15pm.

**Lecture Pace:** Piazza poll results for last class:

• 18%: too fast
• 48%: a bit too fast
• 26%: perfect
• 8%: (a bit) too slow

So will try to slow down a bit.
Last Class:

- Hashing for Jaccard Similarity
  - MinHash for estimating the Jaccard similarity
  - Application to fast similarity search
  - Locality sensitive hashing (LSH)

This Class:

- Finish up MinHash and LSH
- The Frequent Elements (heavy-hitters) problem
- Misra-Gries summaries
Last Class: Hashing for Jaccard Similarity

- MinHash for estimating the Jaccard similarity.
- Application to fast similarity search.
- Locality sensitive hashing (LSH).
Last Class: Hashing for Jaccard Similarity

- MinHash for estimating the Jaccard similarity.
- Application to fast similarity search.
- Locality sensitive hashing (LSH).

This Class:

- Finish up MinHash and LSH.
- The Frequent Elements (heavy-hitters) problem.
- Misra-Gries summaries.
Jaccard Similarity: \[ J(A, B) = \frac{|A \cap B|}{|A \cup B|} = \frac{\# \text{ shared elements}}{\# \text{ total elements}}. \]

Two Common Use Cases:

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

- **All-pairs Similarity Search**: Have \( n \) different sets/bit strings. Want to find all pairs with high similarity. Naively \( O(n^2) \) time.
MinHashing

MinHash(A) = min_{a \in A} h(a) where h : U \rightarrow [0, 1] is a random hash.
MinHashing

MinHash(A) = \min_{a \in A} h(a) where h : U \rightarrow [0, 1] is a random hash.

**Locality Sensitivity:** \( \Pr(\text{MinHash}(A) = \text{MinHash}(B)) = J(A, B) \).
MinHash(A) = min_{a \in A} h(a) where h : U \rightarrow [0, 1] is a random hash.

**Locality Sensitivity:** \( \Pr(\text{MinHash}(A) = \text{MinHash}(B)) = J(A, B) \).

Represents a set with a single number that captures Jaccard similarity information!
MinHash($A$) = $\min_{a \in A} h(a)$ where $h : U \rightarrow [0, 1]$ is a random hash.

**Locality Sensitivity:** $\Pr(\text{MinHash}(A) = \text{MinHash}(B)) = J(A, B)$.

Represents a set with a single number that captures Jaccard similarity information!

Given a collision free hash function $g : [0, 1] \rightarrow [m]$, 
\[
\Pr [g(\text{MinHash}(A)) = g(\text{MinHash}(B))] = J(A, B).
\]
MinHash(A) = \min_{a \in A} h(a) \text{ where } h : U \rightarrow [0, 1] \text{ is a random hash.}

**Locality Sensitivity:**  \( \Pr(\text{MinHash}(A) = \text{MinHash}(B)) = J(A, B) \).

Represents a set with a **single number** that captures Jaccard similarity information!

Given a collision free hash function \( g : [0, 1] \rightarrow [m] \),

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

What happens to \( \Pr [g(\text{MinHash}(A)) = g(\text{MinHash}(B))] \) if \( g \) is not collision free?
**MINHASHING**

\[
\text{MinHash}(A) = \min_{a \in A} h(a) \text{ where } h : U \to [0, 1] \text{ is a random hash.}
\]

**Locality Sensitivity:** \( \Pr(\text{MinHash}(A) = \text{MinHash}(B)) = J(A, B) \).

Represents a set with a **single number** that captures Jaccard similarity information!

Given a collision free hash function \( g : [0, 1] \to [m] \),

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

What happens to \( \Pr[g(\text{MinHash}(A)) = g(\text{MinHash}(B))] \) if \( g \) is not collision free? Collision probability will be larger than \( J(A, B) \).
When searching for similar items only search for matches that land in the same hash bucket.

- False Negative: A similar pair doesn't appear in the same bucket.
- False Positive: A dissimilar pair is hashed to the same bucket.

Need to balance a small probability of false negatives (a high hit rate) with a small probability of false positives (a small query time.)
When searching for similar items only search for matches that land in the same hash bucket.

- **False Negative**: A similar pair doesn’t appear in the same bucket.
- **False Positive**: A dissimilar pair is hashed to the same bucket.
When searching for similar items only search for matches that land in the same hash bucket.

- **False Negative:** A similar pair doesn’t appear in the same bucket.
- **False Positive:** A dissimilar pair is hashed to the same bucket.

Need to balance a small probability of false negatives (a high hit rate) with a small probability of false positives (a small query time.)
Consider a pairwise independent random hash function $h : U \to [m]$. Is this locality sensitive?
Consider a pairwise independent random hash function $h : U \rightarrow [m]$. Is this locality sensitive?

$$\Pr (h(x) = h(y)) = \frac{1}{m} \text{ for all } x, y \in U.$$ Not locality sensitive!
Consider a pairwise independent random hash function $h : U \rightarrow [m]$. Is this locality sensitive?

$$\Pr (h(x) = h(y)) = \frac{1}{m} \text{ for all } x, y \in U. \text{ Not locality sensitive!}$$

- Random hash functions (for load balancing, fast hash table look ups, bloom filters, distinct element counting, etc.) aim to evenly distribute elements across the hash range.
- Locality sensitive hash functions (for similarity search) aim to distribute elements in a way that reflects their similarities.
Balancing False Negatives/Positives with MinHash via repetition.
Balancing False Negatives/Positives with MinHash via repetition.

Create $t$ hash tables. Each is indexed into not with a single MinHash value, but with $r$ values, appended together. A length $r$ signature:

$$MH_{i,1}(x), MH_{i,2}(x), \ldots, MH_{i,r}(x).$$
For $A, B$ with Jaccard similarity $J(A, B) = s$, probability their length $r$ MinHash signatures collide:

$$
Pr \left( [\text{MH}_{i,1}(A), \ldots, \text{MH}_{i,r}(A)] = [\text{MH}_{i,1}(B), \ldots, \text{MH}_{i,r}(B)] \right) = ?.
$$
For $A, B$ with Jaccard similarity $J(A, B) = s$, probability their length $r$ MinHash signatures collide:

$$\Pr \left( [\text{MH}_{i,1}(A), \ldots, \text{MH}_{i,r}(A)] = [\text{MH}_{i,1}(B), \ldots, \text{MH}_{i,r}(B)] \right) = s^r.$$
For $A, B$ with Jaccard similarity $J(A, B) = s$, probability their length $r$ MinHash signatures collide:

$$\Pr \left( [MH_{i,1}(A), \ldots, MH_{i,r}(A)] = [MH_{i,1}(B), \ldots, MH_{i,r}(B)] \right) = s^r.$$ 

Probability the signatures don’t collide:

$$\Pr \left( [MH_{i,1}(A), \ldots, MH_{i,r}(A)] \neq [MH_{i,1}(B), \ldots, MH_{i,r}(B)] \right) = 1 - s^r.$$ 

$MH_{i,j}$: $(i, j)^{th}$ independent instantiation of MinHash. $t$ repetitions ($i = 1, \ldots t$), each with $r$ hash functions ($j = 1, \ldots r$) to make a length $r$ signature.
For $A, B$ with Jaccard similarity $J(A, B) = s$, probability their length $r$ MinHash signatures collide:

$$\Pr\left([\text{MH}_{i,1}(A), \ldots, \text{MH}_{i,r}(A)] = [\text{MH}_{i,1}(B), \ldots, \text{MH}_{i,r}(B)]\right) = s^r.$$ 

Probability the signatures don’t collide:

$$\Pr\left([\text{MH}_{i,1}(A), \ldots, \text{MH}_{i,r}(A)] \neq [\text{MH}_{i,1}(B), \ldots, \text{MH}_{i,r}(B)]\right) = 1 - s^r.$$ 

Probability there is at least one collision in the $t$ hash tables:

$$\Pr\left(\exists i : [\text{MH}_{i,1}(A), \ldots, \text{MH}_{i,r}(A)] = [\text{MH}_{i,1}(B), \ldots, \text{MH}_{i,r}(B)]\right) = 1 - (1 - s^r)^t.$$ 

**MH_{i,j}:** $(i, j)^{th}$ independent instantiation of MinHash. $t$ repetitions ($i = 1, \ldots t$), each with $r$ hash functions ($j = 1, \ldots r$) to make a length $r$ signature.
Using $t$ repetitions each with a signature of $r$ MinHash values, the probability that $x$ and $y$ with Jaccard similarity $J(x, y) = s$ match in at least one repetition is: $1 - (1 - s^r)^t$. 
Using $t$ repetitions each with a signature of $r$ MinHash values, the probability that $x$ and $y$ with Jaccard similarity $J(x, y) = s$ match in at least one repetition is: $1 - (1 - s^r)^t$. 

$r = 5, t = 10$
Using $t$ repetitions each with a signature of $r$ MinHash values, the probability that $x$ and $y$ with Jaccard similarity $J(x, y) = s$ match in at least one repetition is: $1 - (1 - s^r)^t$. 

$r = 10, t = 10$
Using $t$ repetitions each with a signature of $r$ MinHash values, the probability that $x$ and $y$ with Jaccard similarity $J(x, y) = s$ match in at least one repetition is: $1 - (1 - s^r)^t$. 

$\text{r = 5, t = 30}$
Using $t$ repetitions each with a signature of $r$ MinHash values, the probability that $x$ and $y$ with Jaccard similarity $J(x, y) = s$ match in at least one repetition is: $1 - (1 - s^r)^t$.

$r$ and $t$ are tuned depending on application. ‘Threshold’ when hit probability is 1/2 is $\approx (1/t)^{1/r}$. E.g., $\approx (1/30)^{1/5} = .51$ in this case.
For example: Consider a database with 10,000,000 audio clips. You are given a clip $x$ and want to find any $y$ in the database with $J(x, y) \geq .9$. 

Expected Number of Items Scanned: (proportional to query time)
For example: Consider a database with 10,000,000 audio clips. You are given a clip $x$ and want to find any $y$ in the database with $J(x, y) \geq .9$.

- There are 10 true matches in the database with $J(x, y) \geq .9$.
- There are 1000 near matches with $J(x, y) \in [.7, .9]$. 
For example: Consider a database with 10,000,000 audio clips. You are given a clip $x$ and want to find any $y$ in the database with $J(x, y) \geq .9$.

- There are 10 true matches in the database with $J(x, y) \geq .9$.
- There are 1000 near matches with $J(x, y) \in [.7, .9]$.

With signature length $r = 25$ and repetitions $t = 50$, hit probability for $J(x, y) = s$ is $1 - (1 - s^{25})^{50}$. 
For example: Consider a database with 10,000,000 audio clips. You are given a clip $x$ and want to find any $y$ in the database with $J(x, y) \geq .9$.

- There are 10 true matches in the database with $J(x, y) \geq .9$.
- There are 1000 near matches with $J(x, y) \in [.7, .9]$.

With signature length $r = 25$ and repetitions $t = 50$, hit probability for $J(x, y) = s$ is $1 - (1 - s^{25})^{50}$.

- Hit probability for $J(x, y) \geq .9$ is $\geq 1 - (1 - .9^{25})^{50} \approx .98$. 
For example: Consider a database with 10,000,000 audio clips. You are given a clip $x$ and want to find any $y$ in the database with $J(x, y) \geq .9$.

- There are 10 true matches in the database with $J(x, y) \geq .9$.
- There are 1000 near matches with $J(x, y) \in [.7, .9]$.

With signature length $r = 25$ and repetitions $t = 50$, hit probability for $J(x, y) = s$ is $1 - (1 - s^{25})^{50}$.

- Hit probability for $J(x, y) \geq .9$ is $\geq 1 - (1 - .9^{25})^{50} \approx .98$
- Hit probability for $J(x, y) \in [.7, .9]$ is $\leq 1 - (1 - .9^{25})^{50} \approx .98$
For example: Consider a database with 10,000,000 audio clips. You are given a clip $x$ and want to find any $y$ in the database with $J(x, y) \geq .9$.

- There are 10 true matches in the database with $J(x, y) \geq .9$.
- There are 1000 near matches with $J(x, y) \in [.7, .9]$.

With signature length $r = 25$ and repetitions $t = 50$, hit probability for $J(x, y) = s$ is $1 - (1 - s^{25})^{50}$.

- Hit probability for $J(x, y) \geq .9$ is $\geq 1 - (1 - .9^{25})^{50} \approx .98$
- Hit probability for $J(x, y) \in [.7, .9]$ is $\leq 1 - (1 - .9^{25})^{50} \approx .98$
- Hit probability for $J(x, y) \leq .7$ is $\leq 1 - (1 - .7^{25})^{50} \approx .007$
For example: Consider a database with 10,000,000 audio clips. You are given a clip \( x \) and want to find any \( y \) in the database with \( J(x, y) \geq .9 \).

- There are 10 true matches in the database with \( J(x, y) \geq .9 \).
- There are 1000 near matches with \( J(x, y) \in [.7, .9] \).

With signature length \( r = 25 \) and repetitions \( t = 50 \), hit probability for \( J(x, y) = s \) is \( 1 - (1 - s^{25})^{50} \).

- Hit probability for \( J(x, y) \geq .9 \) is \( \geq 1 - (1 - .9^{25})^{50} \approx .98 \)
- Hit probability for \( J(x, y) \in [.7, .9] \) is \( \leq 1 - (1 - .9^{25})^{50} \approx .98 \)
- Hit probability for \( J(x, y) \leq .7 \) is \( \leq 1 - (1 - .7^{25})^{50} \approx .007 \)

Expected Number of Items Scanned: (proportional to query time)

\[ .98 \times 10 + .98 \times 1000 + .007 \times 9,998,990 \approx 80,000 \ll 10,000,000. \]
Repetition and s-curve tuning can be used for search with any similarity metric, given a locality sensitive hash function for that metric.

Cosine Similarity:

\[
\cos(\langle x; y \rangle) = \frac{\langle x; y \rangle}{\|x\|_2 \|y\|_2}.
\]

- \(\cos(\langle x; y \rangle) = 1\) when \(\langle x; y \rangle = 0\)° and \(\cos(\langle x; y \rangle) = 0\) when \(\langle x; y \rangle = 90\)°, and \(\cos(\langle x; y \rangle) = 1\) when \(\langle x; y \rangle = 180\)°.
Repetition and s-curve tuning can be used for search with any similarity metric, given a locality sensitive hash function for that metric.

• LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.
Repetition and s-curve tuning can be used for search with any similarity metric, given a locality sensitive hash function for that metric.

- LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.
Repetition and s-curve tuning can be used for search with any similarity metric, given a locality sensitive hash function for that metric.

- LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.
Repetition and s-curve tuning can be used for search with any similarity metric, given a locality sensitive hash function for that metric.

- 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 search with any similarity metric, given a locality sensitive hash function for that metric.

- 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 search with any similarity metric, given a locality sensitive hash function for that metric.

- 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(x) = \text{sign}(\langle x; t \rangle) \text{ for a random vector } t:

\Pr[\text{SimHash}(x) = \text{SimHash}(y)] = \frac{1}{2} (\langle x; y \rangle + 1):
SimHash Algorithm: LSH for cosine similarity.
SimHash Algorithm: LSH for cosine similarity.

\[ \text{SimHash}(x) = \text{sign}(\langle x, t \rangle) \] for a random vector \( t \).
SimHash Algorithm: LSH for cosine similarity.

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

for a random vector \( t \).

\[
\text{Pr}[\text{SimHash}(x) = \text{SimHash}(y)] = 1 - \frac{\theta(x, y)}{\pi} \approx \frac{\cos(\theta(x, y)) + 1}{2}.
\]
Many applications outside traditional similarity search. E.g., approximate neural net computation (Anshumali Shrivastava).
Many applications outside traditional similarity search. E.g., approximate neural net computation (Anshumali Shrivastava).

\[ n_i = \sigma \left( \sum_{j=1}^{m} w(x_j, n_i) \cdot x_j \right) = \sigma(\langle w_i, x \rangle) \]
Many applications outside traditional similarity search. E.g., approximate neural net computation (Anshumali Shrivastava).

\[ n_i = \sigma \left( \sum_{j=1}^{m} w(x_j, n_i) \cdot x_j \right) = \sigma(\langle w_i, x \rangle) \]

- Evaluating \( \mathcal{N}(x) \) requires \(|x| \cdot |\text{layer 1}| + |\text{layer 1}| \cdot |\text{layer 2}| + \ldots \) multiplications if fully connected.
Many applications outside traditional similarity search. E.g., approximate neural net computation (Anshumali Shrivastava).

- Evaluating $\mathcal{N}(x)$ requires $|x| \cdot |\text{layer 1}| + |\text{layer 1}| \cdot |\text{layer 2}| + \ldots$ multiplications if fully connected.
- Can be expensive, especially on constrained devices like cellphones, cameras, etc.
Many applications outside traditional similarity search. E.g., approximate neural net computation (Anshumali Shrivastava).

- Evaluating $\mathcal{N}(x)$ requires $|x| \cdot |\text{layer 1}| + |\text{layer 1}| \cdot |\text{layer 2}| + \ldots$ multiplications if fully connected.
- Can be expensive, especially on constrained devices like cellphones, cameras, etc.
- For approximate evaluation, suffices to identify the neurons in each layer with high activation when $x$ is presented.
Many applications outside traditional similarity search. E.g., approximate neural net computation (Anshumali Shrivastava).

- Evaluating $\mathcal{N}(x)$ requires $|x| \cdot |\text{layer 1}| + |\text{layer 1}| \cdot |\text{layer 2}| + \ldots$ multiplications if fully connected.
- Can be expensive, especially on constrained devices like cellphones, cameras, etc.
- For approximate evaluation, suffices to identify the neurons in each layer with **high activation** when $x$ is presented.
Important neurons have high activation $\langle w_i, x \rangle$. Since $\cos$ is typically monotonic, this means large $\langle w_i, x \rangle$.

$\cos(\langle w_i, x \rangle) = \langle w_i, x \rangle \left\| w_i \right\| \left\| x \right\|$. Thus these neurons can be found very quickly using LSH for cosine similarity search.

Nonlinearity $\sigma$

$$n_i = \sigma \left( \sum_{j=1}^{m} w(x_j, n_i) \cdot x_j \right) = \sigma(\langle w_i, x \rangle)$$
Important neurons have high activation $\sigma(\langle w_i, x \rangle)$. 
• Important neurons have high activation $\sigma(\langle w_i, x \rangle)$.
• Since $\sigma$ is typically monotonic, this means large $\langle w_i, x \rangle$. 
Important neurons have high activation $\sigma(\langle w_i, x \rangle)$.
Since $\sigma$ is typically monotonic, this means large $\langle w_i, x \rangle$.
$\cos(\theta(w_i, x)) = \frac{\langle w_i, x \rangle}{\|w_i\| \|x\|}$. Thus these neurons can be found very quickly using LSH for cosine similarity search.
# Hashing for Duplicate Detection

<table>
<thead>
<tr>
<th>Goal</th>
<th>Bloom Filters</th>
<th>Hash Table</th>
<th>MinHash</th>
<th>Distinct Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check if x is a duplicate of y in database.</td>
<td>Check if x is a duplicate of any y in database and return y.</td>
<td>Check if x is a duplicate of any y in database and return y.</td>
<td>Count # of items, excluding duplicates.</td>
<td></td>
</tr>
<tr>
<td>Approximate Duplicates?</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

15
## Hashing for Duplicate Detection

<table>
<thead>
<tr>
<th>Goal</th>
<th>Bloom Filters</th>
<th>Hash Table</th>
<th>MinHash</th>
<th>Distinct Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check if x is a duplicate of y in database.</td>
<td>Check if x is a duplicate of any y in database and return y.</td>
<td>Check if x is a duplicate of any y in database and return y.</td>
<td>Count # of items, excluding duplicates.</td>
<td></td>
</tr>
<tr>
<td>Approximate Duplicates?</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

All different variants of detecting duplicates/finding matches in large datasets. An important problem in many contexts!
All different variants of detecting duplicates/finding matches in large datasets. An important problem in many contexts!

*MinHash*(A) is a single number sketch, that can be used both to estimate the number of items in A and the Jaccard similarity between A and other sets.

<table>
<thead>
<tr>
<th>Goal</th>
<th>Bloom Filters</th>
<th>Hash Table</th>
<th>MinHash</th>
<th>Distinct Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Check if x is a duplicate of y in database.</td>
<td>Check if x is a duplicate of any y in database and return y.</td>
<td>Check if x is a duplicate of any y in database and return y.</td>
<td>Count # of items, excluding duplicates.</td>
<td></td>
</tr>
<tr>
<td>Approximate Duplicates?</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>
Questions on MinHash and Locality Sensitive Hashing?
$k$-Frequent Items (Heavy-Hitters) Problem: Consider a stream of $n$ items $x_1, \ldots, x_n$ (with possible duplicates). Return any item that 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.

<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>
\textit{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{table}[h]
\centering
\begin{tabular}{cccccccc}
\hline
\(x_1\) & \(x_2\) & \(x_3\) & \(x_4\) & \(x_5\) & \(x_6\) & \(x_7\) & \(x_8\) & \(x_9\) \\
\hline
5 & 12 & 3 & 3 & 4 & 5 & 5 & 10 & 3 \\
\hline
\end{tabular}
\end{table}

• What is the maximum number of items that must be returned? At most \(k\) items with frequency \(\frac{n}{k}\).

• Trivial with \(O(n)\) space – store the count for each item and return the one that appears \(\frac{n}{k}\) times.

• Can we do it with less space? I.e., without storing all \(n\) items? Same challenge as with the distinct elements problem.
**THE FREQUENT ITEMS PROBLEMS**

$k$-Frequent Items (Heavy-Hitters) Problem: Consider a stream of $n$ items $x_1, \ldots, x_n$ (with possible duplicates). Return any item that 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?
**$k$-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? At most $k$ items with frequency $\geq \frac{n}{k}$.
\textbf{$k$-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>

\begin{itemize}
  \item What is the maximum number of items that must be returned? At most $k$ items with frequency $\geq \frac{n}{k}$.
  \item Trivial with $O(n)$ space – store the count for each item and return the one that appears $\geq n/k$ times.
\end{itemize}
**$k$-Frequent Items (Heavy-Hitters) Problem**: Consider a stream of $n$ items $x_1, \ldots, x_n$ (with possible duplicates). Return any item that 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? At most $k$ items with frequency $\geq \frac{n}{k}$.
- 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?
**k-Frequent Items (Heavy-Hitters) Problem:** Consider a stream of \( n \) items \( x_1, \ldots, x_n \) (with possible duplicates). Return any item that 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? At most \( k \) items with frequency \( \geq \frac{n}{k} \).
- 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?
- Same challenge as with the distinct elements problem.
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. I.e., want to maintain a running list of frequent items that appear in a stream.
Majority: Consider a stream of $n$ items $x_1, \ldots, x_n$, where a single item appears a majority of the time. Return this item.
**MAJORITY IN DATA STREAMS**

**Majority:** Consider a stream of \( n \) items \( x_1, \ldots, x_n \), where a single item appears a majority of the time. Return this item.

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_1 )</td>
<td>( x_2 )</td>
<td>( x_3 )</td>
<td>( x_4 )</td>
<td>( x_5 )</td>
<td>( x_6 )</td>
<td>( x_7 )</td>
<td>( x_8 )</td>
<td>( x_9 )</td>
<td>( x_{10} )</td>
</tr>
<tr>
<td>5</td>
<td>12</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
**MAJORITY IN DATA STREAMS**

**Majority:** Consider a stream of $n$ items $x_1, \ldots, x_n$, where a single item appears a majority of the time. Return this item.

<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>
<th>$x_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>

- Basically $k$-Frequent items for $k = 2$ (except assume a single item has a strict majority.)
Boyer-Moore Voting Algorithm: (our first deterministic algorithm)

• Initialize count $c := 0$, majority element $m := \bot$
Boyer-Moore Voting Algorithm: (our first deterministic algorithm)

- Initialize count $c := 0$, majority element $m := \bot$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.  
  - Else if $m \neq x_i$, set $c := c - 1$.  

Just requires $O(\log n)$ bits to store $c$ and space to store $m$.  

Boyermoore Voting Algorithm: (our first deterministic algorithm)

- Initialize count $c := 0$, majority element $m := \bot$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.
  - Else if $m \neq x_i$, set $c := c - 1$.

Just requires $O(\log n)$ bits to store $c$ and space to store $m$. 
Boyer-Moore Voting Algorithm: (our first deterministic algorithm)

- Initialize count $c := 0$, majority element $m := \bot$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.
  - Else if $m \neq x_i$, set $c := c - 1$.

Just requires $O(\log n)$ bits to store $c$ and space to store $m$. 

<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>
<th>$x_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Boyer-Moore Voting Algorithm: (our first *deterministic algorithm*)

- Initialize count $c := 0$, majority element $m := \bot$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.
  - Else if $m \neq x_i$, set $c := c - 1$.

Just requires $O(\log n)$ bits to store $c$ and space to store $m$. 

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

**c=1, m=5**

**c=0, m=⊥**
Boyer-Moore Voting Algorithm: (our first deterministic algorithm)

- Initialize count $c := 0$, majority element $m := \perp$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.
  - Else if $m \neq x_i$, set $c := c - 1$.

Just requires $O(\log n)$ bits to store $c$ and space to store $m$. 

<table>
<thead>
<tr>
<th></th>
<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>
<th>$x_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>5</td>
<td>12</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Boyer-Moore Voting Algorithm: (our first deterministic algorithm)

- Initialize count \( c := 0 \), majority element \( m := \perp \)
- For \( i = 1, \ldots, n \)
  - If \( c = 0 \), set \( m := x_i \)
  - Else if \( m = x_i \), set \( c := c + 1 \).
  - Else if \( m \neq x_i \), set \( c := c - 1 \).

Just requires \( O(\log n) \) bits to store \( c \) and space to store \( m \).
Boyer-Moore Voting Algorithm: (our first deterministic algorithm)

• Initialize count \( c := 0 \), majority element \( m := \bot \)
• For \( i = 1, \ldots, n \)
  • If \( c = 0 \), set \( m := x_i \)
  • Else if \( m = x_i \), set \( c := c + 1 \).
  • Else if \( m \neq x_i \), set \( c := c - 1 \).

Just requires \( O(\log n) \) bits to store \( c \) and space to store \( m \).
Boyer-Moore Voting Algorithm: (our first deterministic algorithm)

- Initialize count $c := 0$, majority element $m := \bot$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.
  - Else if $m \neq x_i$, set $c := c - 1$.

Just requires $O(\log n)$ bits to store $c$ and space to store $m$. 

<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>
<th>$x_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Boyer-Moore Voting Algorithm: (our first *deterministic algorithm*)

- Initialize count $c := 0$, majority element $m := \bot$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.
  - Else if $m \neq x_i$, set $c := c - 1$.

Just requires $O(\log n)$ bits to store $c$ and space to store $m$. 
Boyer-Moore Voting Algorithm: (our first *deterministic algorithm*)

- Initialize count $c := 0$, majority element $m := \bot$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.
  - Else if $m \neq x_i$, set $c := c - 1$.

Just requires $O(\log n)$ bits to store $c$ and space to store $m$. 

<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>
<th>$x_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Boyer-Moore Voting Algorithm: (our first deterministic algorithm)

• Initialize count \( c := 0 \), majority element \( m := \bot \)
• For \( i = 1, \ldots, n \)
  • If \( c = 0 \), set \( m := x_i \)
  • Else if \( m = x_i \), set \( c := c + 1 \).
  • Else if \( m \neq x_i \), set \( c := c - 1 \).

Just requires \( O(\log n) \) bits to store \( c \) and space to store \( m \).
Boyer-Moore Voting Algorithm: (our first deterministic algorithm)

- Initialize count $c := 0$, majority element $m := \bot$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$
  - Else if $m \neq x_i$, set $c := c - 1$

Just requires $O(\log n)$ bits to store $c$ and space to store $m$. 
Boyer-Moore Voting Algorithm: (our first deterministic algorithm)

• Initialize count \( c := 0 \), majority element \( m := \perp \)
• For \( i = 1, \ldots, n \)
  • If \( c = 0 \), set \( m := x_i \)
  • Else if \( m = x_i \), set \( c := c + 1 \).
  • Else if \( m \neq x_i \), set \( c := c - 1 \).

Just requires \( O(\log n) \) bits to store \( c \) and space to store \( m \).
Boyer-Moore Voting Algorithm:

- Initialize count $c := 0$, majority element $m := \bot$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.
  - Else if $m \neq x_i$, set $c := c - 1$.

Claim: The Boyer-Moore algorithm always outputs the majority element, regardless of what order the stream is presented in.

Proof: Let $M$ be the true majority element. Let $s = c$ when $m = M$ and $s = c$ otherwise.

- $s$ is incremented each time $M$ appears. So it is incremented more than it is decremented and ends at a positive value.

The algorithm ends with $m = M$. 

21
Boyer-Moore Voting Algorithm:

- Initialize count $c := 0$, majority element $m :=\bot$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.
  - Else if $m \neq x_i$, set $c := c - 1$.

**Claim:** The Boyer-Moore algorithm always outputs the majority element, regardless of what order the stream is presented in.
Boyer-Moore Voting Algorithm:

- Initialize count $c := 0$, majority element $m := \bot$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$
  - Else if $m \neq x_i$, set $c := c - 1$

**Claim:** The Boyer-Moore algorithm always outputs the majority element, regardless of what order the stream is presented in.

**Proof:** Let $M$ be the true majority element. Let $s = c$ when $m = M$ and $s = -c$ otherwise.
Boyer-Moore Voting Algorithm:

- Initialize count $c := 0$, majority element $m := \bot$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.
  - Else if $m \neq x_i$, set $c := c - 1$.

**Claim:** The Boyer-Moore algorithm always outputs the majority element, regardless of what order the stream is presented in.

**Proof:** Let $M$ be the true majority element. Let $s = c$ when $m = M$ and $s = -c$ otherwise.

<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>
<th>$x_{10}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>12</td>
<td>3</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>5</td>
<td>5</td>
</tr>
</tbody>
</table>
Boyer-Moore Voting Algorithm:

- Initialize count $c := 0$, majority element $m := \perp$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.
  - Else if $m \neq x_i$, set $c := c - 1$.

Claim: The Boyer-Moore algorithm always outputs the majority element, regardless of what order the stream is presented in.

Proof: Let $M$ be the true majority element. Let $s = c$ when $m = M$ and $s = -c$ otherwise.
Boyer-Moore Voting Algorithm:

- Initialize count $c := 0$, majority element $m := \perp$

- For $i = 1, \ldots, n$
  
  - If $c = 0$, set $m := x_i$
  
  - Else if $m = x_i$, set $c := c + 1$.
  
  - Else if $m \neq x_i$, set $c := c - 1$.

Claim: The Boyer-Moore algorithm always outputs the majority element, regardless of what order the stream is presented in.

Proof: Let $M$ be the true majority element. Let $s = c$ when $m = M$ and $s = -c$ otherwise.
Boyer-Moore Voting Algorithm:

- Initialize count $c := 0$, majority element $m := \bot$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.
  - Else if $m \neq x_i$, set $c := c - 1$.

Claim: The Boyer-Moore algorithm always outputs the majority element, regardless of what order the stream is presented in.

Proof: Let $M$ be the true majority element. Let $s = c$ when $m = M$ and $s = -c$ otherwise.

- $s$ is incremented each time $M$ appears. So it is incremented more than it is decremented and ends at a positive value.
**Boyer-Moore Voting Algorithm:**

- Initialize count \( c := 0 \), majority element \( m := \bot \)
- For \( i = 1, \ldots, n \)
  - If \( c = 0 \), set \( m := x_i \)
  - Else if \( m = x_i \), set \( c := c + 1 \).
  - Else if \( m \neq x_i \), set \( c := c - 1 \).

**Claim:** The Boyer-Moore algorithm always outputs the majority element, regardless of what order the stream is presented in.

**Proof:** Let \( M \) be the true majority element. Let \( s = c \) when \( m = M \) and \( s = -c \) otherwise.

- \( s \) is incremented each time \( M \) appears. So it is incremented more than it is decremented and ends at a positive value. \( \implies \) algorithm ends with \( m = M \).
**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.

**Boyer-Moore Voting Algorithm:**

- Initialize count $c := 0$, majority element $m := \bot$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.
  - Else if $m \neq x_i$, set $c := c - 1$. 
**k-Frequent Items (Heavy-Hitters) Problem:** Consider a stream of $n$ items $x_1, \ldots, x_n$ (with possible duplicates). Return any item that appears at least $\frac{n}{k}$ times.

**Misra-Gries Summary:**

- Initialize count $c := 0$, majority element $m := \bot$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.
  - Else if $m \neq x_i$, set $c := c - 1$. 
$k$-Frequent Items (Heavy-Hitters) Problem: Consider a stream of $n$ items $x_1, \ldots, x_n$ (with possible duplicates). Return any item that appears at least $\frac{n}{k}$ times.

Misra-Gries Summary:

- Initialize counts $c_1, \ldots, c_k := 0$, elements $m_1, \ldots, m_k := \bot$
- For $i = 1, \ldots, n$
  - If $c = 0$, set $m := x_i$
  - Else if $m = x_i$, set $c := c + 1$.
  - Else if $m \neq x_i$, set $c := c - 1$. 
$k$-Frequent Items (Heavy-Hitters) Problem: Consider a stream of $n$ items $x_1, \ldots, x_n$ (with possible duplicates). Return any item that appears at least $\frac{n}{k}$ times.

**Misra-Gries Summary:**

- Initialize counts $c_1, \ldots, c_k := 0$, elements $m_1, \ldots, m_k := \perp$
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some $j$, set $c_j := c_j + 1$.
  - Else let $t = \arg \min c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all $j$. 
Misra-Gries Summary:

- Initialize counts $c_1, \ldots, c_k := 0$, elements $m_1, \ldots, m_k := \perp$.
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some $j$, set $c_j := c_j + 1$.
  - Else let $t = \arg\min c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all $j$.

Claim: At the end of the stream, all items with frequency $n_k$ are stored.
**Misra-Gries Summary:**

- Initialize counts $c_1, \ldots, c_k := 0$, elements $m_1, \ldots, m_k := \bot$.
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some $j$, set $c_j := c_j + 1$.
  - Else let $t = \arg\min c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all $j$.

<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>
Misra-Gries Summary:

- Initialize counts \( c_1, \ldots, c_k := 0 \), elements \( m_1, \ldots, m_k := \bot \).
- For \( i = 1, \ldots, n \):
  - If \( m_j = x_i \) for some \( j \), set \( c_j := c_j + 1 \).
  - Else let \( t = \arg \min c_j \). If \( c_t = 0 \), set \( m_t := x_i \) and \( c_t := 1 \).
  - Else \( c_j := c_j - 1 \) for all \( j \).

<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>
Misra-Gries Summary:

- Initialize counts $c_1, \ldots, c_k := 0$, elements $m_1, \ldots, m_k := \bot$.
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some $j$, set $c_j := c_j + 1$.
  - Else let $t = \arg \min c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all $j$.

<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>
Misra-Gries Summary:

- Initialize counts $c_1, \ldots, c_k := 0$, elements $m_1, \ldots, m_k := \bot$.
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some $j$, set $c_j := c_j + 1$.
  - Else let $t = \arg \min c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all $j$.

<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>
MISRA-GRIES ALGORITHM

Misra-Gries Summary:

- Initialize counts $c_1, \ldots, c_k := 0$, elements $m_1, \ldots, m_k := \bot$.
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some $j$, set $c_j := c_j + 1$.
  - Else let $t = \arg \min c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all $j$.

Claim: At the end of the stream, all items with frequency $n_k$ are stored.

<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>
Misra-Gries Algorithm

**Misra-Gries Summary:**

- Initialize counts $c_1, \ldots, c_k := 0$, elements $m_1, \ldots, m_k := \perp$.
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some $j$, set $c_j := c_j + 1$.
  - Else let $t = \arg\min c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all $j$.

<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>
Misra-Gries Summary:

- Initialize counts $c_1, \ldots, c_k := 0$, elements $m_1, \ldots, m_k := \perp$.
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some $j$, set $c_j := c_j + 1$.
  - Else let $t = \arg \min c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all $j$.

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
 x_1 & x_2 & x_3 & x_4 & x_5 & x_6 & x_7 & x_8 & x_9 \\
 5   & 12  & 3   & 3   & 4   & 5   & 5   & 10  & 3 \\
\end{array}
\]
Misra-Gries Summary:

- Initialize counts $c_1, \ldots, c_k := 0$, elements $m_1, \ldots, m_k := \bot$.
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some $j$, set $c_j := c_j + 1$.
  - Else let $t = \arg \min c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all $j$.

<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>
Misra-Gries Summary:

- Initialize counts $c_1, \ldots, c_k := 0$, elements $m_1, \ldots, m_k := \bot$.
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some $j$, set $c_j := c_j + 1$.
  - Else let $t = \arg \min c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all $j$.

<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>
Misra-Gries Summary:

- Initialize counts $c_1, \ldots, c_k := 0$, elements $m_1, \ldots, m_k := \bot$.
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some $j$, set $c_j := c_j + 1$.
  - Else let $t = \arg \min c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all $j$.

Claim: At the end of the stream, all items with frequency $n_k$ are stored.
Misra-Gries Summary:

- Initialize counts $c_1, \ldots, c_k := 0$, elements $m_1, \ldots, m_k := \bot$.
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some $j$, set $c_j := c_j + 1$.
  - Else let $t = \arg \min c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all $j$.

Claim: At the end of the stream, all items with frequency $\geq \frac{n}{k}$ are stored.
**Claim:** At the end of the stream, the Misra-Gries algorithm stores $k$ items, including all those with frequency $\geq \frac{n}{k}$.
Claim: At the end of the stream, the Misra-Gries algorithm stores $k$ items, including all those with frequency $\geq \frac{n}{k}$.

Intuition:

- If there are exactly $k$ items, each appearing exactly $n/k$ times, all are stored.
Claim: At the end of the stream, the Misra-Gries algorithm stores $k$ items, including all those with frequency $\geq \frac{n}{k}$.

Intuition:

- If there are exactly $k$ items, each appearing exactly $n/k$ times, all are stored.
- If there are $k/2$ items each appearing $\geq n/k$ times, there are $\leq n/2$ irrelevant items, being inserted into $k/2$ ‘free slots’.
Claim: At the end of the stream, the Misra-Gries algorithm stores $k$ items, including all those with frequency $\geq \frac{n}{k}$.

Intuition:

- If there are exactly $k$ items, each appearing exactly $n/k$ times, all are stored.
- If there are $k/2$ items each appearing $\geq n/k$ times, there are $\leq n/2$ irrelevant items, being inserted into $k/2$ ‘free slots’.
- May cause $\frac{n/2}{k/2} = \frac{n}{k}$ decrement operations. Few enough that the heavy items still have positive counts.
Claim: At the end of the stream, the Misra-Gries algorithm stores $k$ items, including all those with frequency $\geq \frac{n}{k}$.

Intuition:

- If there are exactly $k$ items, each appearing exactly $n/k$ times, all are stored.
- If there are $k/2$ items each appearing $\geq n/k$ times, there are $\leq n/2$ irrelevant items, being inserted into $k/2$ ‘free slots’.
- May cause $\frac{n/2}{k/2} = \frac{n}{k}$ decrement operations. Few enough that the heavy items still have positive counts.

Anything undesirable about the Misra-Gries output guarantee?
**Claim:** At the end of the stream, the Misra-Gries algorithm stores \( k \) items, including all those with frequency \( \geq \frac{n}{k} \).

**Intuition:**

- If there are exactly \( k \) items, each appearing exactly \( n/k \) times, all are stored.
- If there are \( k/2 \) items each appearing \( \geq n/k \) times, there are \( \leq n/2 \) irrelevant items, being inserted into \( k/2 \) ‘free slots’.
- May cause \( \frac{n/2}{k/2} = \frac{n}{k} \) decrement operations. Few enough that the heavy items still have positive counts.

Anything undesirable about the Misra-Gries output guarantee?
May have false positives – infrequent items that are stored.
Issue: Misra-Gries algorithm stores $k$ items, including all with frequency $\geq n/k$. But may include infrequent items.
**Issue:** Misra-Gries algorithm stores $k$ items, including all with frequency $\geq n/k$. But may include infrequent items.

- In fact, no algorithm using $o(n)$ space can output just the items with frequency $\geq n/k$. To hard to tell e.g., between an item with frequency $n/k$ (should be output) and $n/k - 1$ (should not be output).


**Issue:** Misra-Gries algorithm stores \(k\) items, including all with frequency \(\geq n/k\). But may include infrequent items.

- In fact, no algorithm using \(o(n)\) space can output just the items with frequency \(\geq n/k\). To hard to tell e.g., between an item with frequency \(n/k\) (should be output) and \(n/k - 1\) (should not be output).

\((\epsilon, k)\)-**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:** Misra-Gries algorithm stores $k$ items, including all with frequency $\geq n/k$. But may include infrequent items.

- In fact, no algorithm using $o(n)$ space can output just the items with frequency $\geq n/k$. To hard to tell e.g., between an item with frequency $n/k$ (should be output) and $n/k - 1$ (should not be output).

**$(\epsilon, k)$-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.
Misra-Gries Summary: ($\epsilon$-error version)

- Let $r := \lceil k/\epsilon \rceil$
- Initialize counts $c_1, \ldots, c_r := 0$, elements $m_1, \ldots, m_r := \bot$.
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some $j$, set $c_j := c_j + 1$.
  - Else let $t = \text{arg min} \ c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all $j$.
- Return any $m_j$ with $c_j \geq (1 - \epsilon) \cdot \frac{n}{k}$. 
**Misra-Gries Summary:** (\(\varepsilon\)-error version)

- Let \( r := \lceil k/\varepsilon \rceil \)
- Initialize counts \( c_1, \ldots, c_r := 0 \), elements \( m_1, \ldots, m_r := \bot \).
- For \( i = 1, \ldots, n \)
  - If \( m_j = x_i \) for some \( j \), set \( c_j := c_j + 1 \).
  - Else let \( t = \arg \min c_j \). If \( c_t = 0 \), set \( m_t := x_i \) and \( c_t := 1 \).
  - Else \( c_j := c_j - 1 \) for all \( j \).
- Return any \( m_j \) with \( c_j \geq (1 - \varepsilon) \cdot \frac{n}{k} \).

**Claim:** For all \( m_j, f(m_j) - \frac{\varepsilon n}{k} \leq c_j \leq f(m_j) \), where \( f(m_j) \) is the true frequency. \( r \) is large so relatively few decrements.
Misra-Gries Summary: ($\epsilon$-error version)

- Let $r := \lceil k/\epsilon \rceil$
- Initialize counts $c_1, \ldots, c_r := 0$, elements $m_1, \ldots, m_r := \bot$.
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some $j$, set $c_j := c_j + 1$.
  - Else let $t = \arg \min c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all $j$.
- Return any $m_j$ with $c_j \geq (1 - \epsilon) \cdot \frac{n}{k}$.

Claim: For all $m_j$, $f(m_j) - \frac{\epsilon n}{k} \leq c_j \leq f(m_j)$, where $f(m_j)$ is the true frequency. $r$ is large so relatively few decrements.

Implication: If $f(m_j) \geq \frac{n}{k}$, then $c_j \geq (1 - \epsilon) \cdot \frac{n}{k}$ so the item is returned. If $f(m_j) < (1 - \epsilon) \cdot \frac{n}{k}$, then $c_j < (1 - \epsilon) \cdot \frac{n}{k}$ so the item is not returned.
Misra-Gries Summary: ($\epsilon$-error version)

- Let $r := \lceil k/\epsilon \rceil$
- Initialize counts $c_1, \ldots, c_r := 0$, elements $m_1, \ldots, m_r := \bot$
- For $i = 1, \ldots, n$
  - If $m_j = x_i$ for some $j$, set $c_j := c_j + 1$.
  - Else let $t = \text{arg min} c_j$. If $c_t = 0$, set $m_t := x_i$ and $c_t := 1$.
  - Else $c_j := c_j - 1$ for all $j$.
- Return any $m_j$ with $c_j \geq (1 - \epsilon) \cdot \frac{n}{K}$.

Claim: For all $m_j$, $f(m_j) - \frac{\epsilon n}{K} \leq c_j \leq f(m_j)$, where $f(m_j)$ is the true frequency. $r$ is large so relatively few decrements.

Implication: If $f(m_j) \geq \frac{n}{K}$, then $c_j \geq (1 - \epsilon) \cdot \frac{n}{K}$ so the item is returned. If $f(m_j) < (1 - \epsilon) \cdot \frac{n}{K}$, then $c_j < (1 - \epsilon) \cdot \frac{n}{K}$ so the item is not returned. Solves $(\epsilon, k)$-Frequent Items.
**Upshot:** The \((\epsilon, k)\)-Frequent Items problems can be solved via the Misra-Gries approach.
Upshot: The $(\epsilon, k)$-Frequent Items problems can be solved via the Misra-Gries approach.

- Space usage is $\lceil k/\epsilon \rceil$ counts – $O \left( \frac{k \log n}{\epsilon} \right)$ bits and $\lceil k/\epsilon \rceil$ items.
- Deterministic approximation algorithm.
Questions?