# Streaming Algorithms

Short recall of last lecture

# Space vs Accuracy

# Example applications

- Are there very frequent search queries (or ip addresses)?
- How many different search queries?
- A third one today

# The setting

- A stream consisting of *m* elements: $\sigma = \langle a_1, a_2, \ldots, a_m \rangle$
- Every element $a_i$ takes a value in $[n] = \{1, 2, \ldots, n\}$
- A stream defines a frequency vector $f = (f_1, f_2, \ldots, f_n)$ where

$$f_i = |\{j : a_j = i\}|$$ number of items of value *i*

In the search example, we have m searches, each search is represented by an integer from *1* to *n* and $f_i$ denotes the number of searchers of value *i*

# Finding very frequent queries

**FREQUENT** problem with parameter *k*:

output the set $\{j : f_j > m/k\}$

## Misra-Gries Algorithm
with parameter ***k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3

| | |
|---|---|
| A[1] | 0 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |

**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**

## Misra-Gries Algorithm
with parameter ***k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3

| | |
|---|---|
| A[1] | 0 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |

**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**

## Misra-Gries Algorithm
with parameter ***k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3

| | |
|---|---|
| A[1] | 1 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |

**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**

# Misra-Gries Algorithm

with parameter ***k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3

| | |
|---|---|
| A[1] | 1 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |

**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**


# Misra-Gries Algorithm

with parameter ***k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 2 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |


**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**



# Misra-Gries Algorithm

with parameter ***k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 2 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |


**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**



# Misra-Gries Algorithm

with parameter ***k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 2 |
| A[2] | 1 |
| A[3] | 0 |
| A[4] | 0 |


**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**

# Misra-Gries Algorithm

with parameter *k*

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3

| | |
|---|---|
| A[1] | 2 |
| A[2] | 2 |
| A[3] | 0 |
| A[4] | 0 |

**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**


# Misra-Gries Algorithm

with parameter *k*

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 2 |
| A[2] | 2 |
| A[3] | 0 |
| A[4] | 0 |


**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**

# Misra-Gries Algorithm

with parameter *k*

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 1 |
| A[2] | 1 |
| A[3] | 0 |
| A[4] | 0 |


**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**

# Misra-Gries Algorithm

with parameter *k*

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 0 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |


**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**

# Misra-Gries Algorithm

**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3

| | |
|---|---|
| A[1] | 0 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |

**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**


# Misra-Gries Algorithm

**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 1 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |


**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**

# Misra-Gries Algorithm

**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 1 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 1 |


**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**

# Misra-Gries Algorithm

**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 1 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 2 |


**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**

# Misra-Gries Algorithm

**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3

| | |
|---|---|
| A[1] | 2 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 2 |

**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**


# Misra-Gries Algorithm

**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3

| | |
|---|---|
| A[1] | 2 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 2 |

**Algorithm only stores non-zero counters and at most k-1 counters will be non-zero at any time**


Output: $\hat{f}_1 = 2, \hat{f}_2 = 0, \hat{f}_3 = 0, \hat{f}_4 = 2$


# Misra-Gries Algorithm

**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.


Space usage:

- At most k-1 counters
- Each counter can be stored using log n bits for key and log m bits for value
- Total space: O(k (log n + log m))



# Misra-Gries Algorithm

**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.


**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3

| | |
|---|---|
| A[1] | 0 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |

# Misra-Gries Algorithm

**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3

| | |
|---|---|
| A[1] | 0 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |


# Misra-Gries Algorithm

**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 1 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |


# Misra-Gries Algorithm

**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 1 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |


# Misra-Gries Algorithm

**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 2 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |

# Misra-Gries Algorithm

**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3

| | |
|---|---|
| A[1] | 2 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |


# Misra-Gries Algorithm

**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 2 |
| A[2] | 1 |
| A[3] | 0 |
| A[4] | 0 |


# Misra-Gries Algorithm

**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 2 |
| A[2] | 2 |
| A[3] | 0 |
| A[4] | 0 |


# Misra-Gries Algorithm

**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 2 |
| A[2] | 2 |
| A[3] | 0 |
| A[4] | 0 |

# Misra-Gries Algorithm
with parameter *k*

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3

| | |
|---|---|
| A[1] | 2 |
| A[2] | 2 |
| A[3] | 0 |
| A[4] | 1 |

**"Fake" add**


# Misra-Gries Algorithm
with parameter *k*

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 1 |
| A[2] | 1 |
| A[3] | 0 |
| A[4] | 0 |

**"Fake" add**
**... then remove immediately**


# Misra-Gries Algorithm
with parameter *k*

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 1 |
| A[2] | 1 |
| A[3] | 0 |
| A[4] | 0 |


# Misra-Gries Algorithm
with parameter *k*

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 1 |
| A[2] | 1 |
| A[3] | 0 |
| A[4] | 1 |

**"Fake" add**

# Misra-Gries Algorithm
with parameter *k*

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3

| | |
|---|---|
| A[1] | 0 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |

**"Fake" add**
**… then remove immediately**


# Misra-Gries Algorithm
with parameter *k*

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 0 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |


# Misra-Gries Algorithm
with parameter *k*

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 1 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 0 |


# Misra-Gries Algorithm
with parameter *k*

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 1 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 1 |

## Misra-Gries Algorithm
**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3

| | |
|---|---|
| A[1] | 1 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 2 |


## Misra-Gries Algorithm
**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 2 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 2 |


## Misra-Gries Algorithm
**with parameter *k***

**Initialization:** $A \leftarrow$ (empty associative array).

**Process** $j$**:**

1. If $j \in keys(A)$ then
2. $\quad A[j] = A[j] + 1$
3. Else if $|keys(A)| < k - 1$ then
4. $\quad A[j] = 1$
5. Else foreach $\ell \in keys(A)$ do
6. $\quad A[\ell] = A[\ell] - 1$ if $A[\ell] = 0$ then remove $\ell$ from $A$.

**Output:** On query $a$, if $a \in keys(A)$, then report $\hat{f}_a = A[a]$, else report $\hat{f}_a = 0$.

**Guarantee on output?**

Small reinterpretation

Example: *n=4,* stream <*1,1,2,2,4,4,1,4,4,1*> and k=3


| | |
|---|---|
| A[1] | 2 |
| A[2] | 0 |
| A[3] | 0 |
| A[4] | 2 |

Output: $\hat{f}_1 = 2, \hat{f}_2 = 0, \hat{f}_3 = 0, \hat{f}_4 = 2$

# Guarantee on output

- Total # increments (with fake adds) = m
- Total # iterations where we decrement <= ?
  - Each time we decrement we decrease a counter we decrease k counters

# Guarantee on output

- Total # increments (with fake adds) = m
- Total # iterations where we decrement <= m/k
  - Each time we decrement we decrease a counter we decrease k counters

# Guarantee on output

- Total # increments (with fake adds) = m
- Total # iterations where we decrement <= m/k
  - Each time we decrement we decrease a counter we decrease k counters

**Every time it goes up, it goes up one floor**

# Guarantee on output

- Total # increments (with fake adds) = m
- Total # iterations where we decrement <= m/k
  - Each time we decrement we decrease a counter we decrease k counters

**Every time it goes up, it goes up one floor**

# Guarantee on output

- Total # increments (with fake adds) = m
- Total # iterations where we decrement <= m/k
  - Each time we decrement we decrease a counter we decrease k counters

**Every time it goes up, it goes up one floor**

# Guarantee on output

- Total # increments (with fake adds) = m
- Total # iterations where we decrement <= m/k
  - Each time we decrement we decrease a counter we decrease k counters

**Every time it goes up, it goes up one floor**

# Guarantee on output

- Total # increments (with fake adds) = m
- Total # iterations where we decrement <= m/k
  - Each time we decrement we decrease a counter we decrease k counters

**Every time it goes up, it goes up one floor**

**Every time it goes down, it goes down k floors**

# Guarantee on output

- Total # increments (with fake adds) = m
- Total # iterations where we decrement <= m/k
  - Each time we decrement we decrease a counter we decrease k counters

**Every time it goes up, it goes up one floor**

**Every time it goes down, it goes down k floors**

# Guarantee on output

- Total # increments (with fake adds) = m
- Total # iterations where we decrement <= m/k
  - Each time we decrement we decrease a counter we decrease k counters
- It follows that for every *j:*

$$f_j - m/k \leq \hat{f}_j \leq f_j$$

- To solve FREQUENT problem run a second pass over the stream to exactly count the frequencies of items $\{j : \hat{f}_j > 0\}$

# How many distinct queries

**DISTINCT-ELEMENTS** problem:

output the number of distinct elements on stream

$$d(\sigma) = |\{j : f_j > 0\}|$$

# Randomization and approximation

- No exact or deterministic low space algorithm
- We shall use randomization to output estimate $\hat{d}$ satisfying

$$\Pr[d/3 \leq \hat{d} \leq 3d] \geq 1 - \delta$$

- The space requirement will be

$$O(\log(1/\delta) \log n)$$

# Ingredients

- Family of pairwise independent hash functions h: [n] -> [n]
  - We assume n is a power of 2 so h(a) maps to a random bit string of length $\log_2(n)$
- The zeros function

$$zeros(p) = \max\{i : 2^i \text{ divides } p\}$$

$$zeros(2) = 1, zeros(3) = 0, zeros(4) = 2, zeros(6) = 1, zeros(7) = 0$$

**#0's to the right in the binary presentation**

# Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process** $j$**:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

**Intuition**

- The probability that a random number *x* has *zeros(x)* >= *log(d)* is *1/d*
- So if we have *d* distinct numbers then we would expect that *zeros(h(j))>= log d* for some element *j*

## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process $j$:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

Example: *n=4,* stream *<1,1,2,2,4,4,1,4,4,1>, h = identity*

z 0


## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process $j$:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

Example: *n=4,* stream *<1,1,2,2,4,4,1,4,4,1>, h = identity*


z 0 $zeros(h(1)) = zeros(1) = 0$


## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process $j$:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

Example: *n=4,* stream *<1,1,2,2,4,4,1,4,4,1>, h = identity*


z 0 $zeros(h(1)) = zeros(1) = 0$


## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process $j$:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

Example: *n=4,* stream *<1,1,2,2,4,4,1,4,4,1>, h = identity*


z 0 $zeros(h(2)) = zeros(2) = 1$

## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process** $j$**:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

Example: *n=4,* stream *<1,1,2,2,4,4,1,4,4,1>, h = identity*

z | 1

$zeros(h(2)) = zeros(2) = 1$


## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process** $j$**:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

Example: *n=4,* stream *<1,1,2,2,4,4,1,4,4,1>, h = identity*

z | 1

$zeros(h(2)) = zeros(2) = 1$



## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process** $j$**:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

Example: *n=4,* stream *<1,1,2,2,4,4,1,4,4,1>, h = identity*


z | 1

$zeros(h(4)) = zeros(4) = 2$


## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process** $j$**:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

Example: *n=4,* stream *<1,1,2,2,4,4,1,4,4,1>, h = identity*


z | 2

$zeros(h(4)) = zeros(4) = 2$

## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process $j$:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

Example: *n=4,* stream *<1,1,2,2,4,4,1,4,4,1>, h = identity*

z 2

$$zeros(h(4)) = zeros(4) = 2$$


## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process $j$:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

Example: *n=4,* stream *<1,1,2,2,4,4,1,4,4,1>, h = identity*


z 2

$$zeros(h(1)) = zeros(1) = 0$$


## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process $j$:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

Example: *n=4,* stream *<1,1,2,2,4,4,1,4,4,1>, h = identity*


z 2

$$zeros(h(4)) = zeros(4) = 2$$


## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process $j$:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

Example: *n=4,* stream *<1,1,2,2,4,4,1,4,4,1>, h = identity*


z 2

$$zeros(h(4)) = zeros(4) = 2$$

## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process** $j$**:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

Example: *n=4,* stream *<1,1,2,2,4,4,1,4,4,1>, h = identity*

z 2 $zeros(h(1)) = zeros(1) = 0$

## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process** $j$**:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

Example: *n=4,* stream *<1,1,2,2,4,4,1,4,4,1>, h = identity*

z 2 Output: $\hat{d} = 2^{2+1/2} \approx 5.66$

## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process** $j$**:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

Example: *n=4,* stream *<1,1,2,2,4,4,1,4,4,1>, h = identity*

z 2 Output: $\hat{d} = 2^{2+1/2} \approx 5.66$

Space: hash function + z = O(log n)


## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process** $j$**:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

**Quality of estimate**

- Let *t* be final value of *z* and so $\hat{d} = 2^{t+1/2}$
- Let $X_{j,r}$ be random indicator variable for the event "*zeros(h(j))>= r*"
- Let $Y_r$ be the number of distinct values with more than *r* zeros:

$$Y_r = \sum_{j : f_j > 0} X_{r,j}$$

- Note that $Y_r > 0 \iff t \geq r$
- Similarly $Y_r = 0 \iff t \leq r - 1$

## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process** $j$**:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

**Quality of estimate**

- Upper bounding $\Pr[\hat{d} > 3d]$
- Since *h(j)* is uniformly distributed over *log(n)* bit strings

$$\mathbb{E}[X_{r,j}] = \Pr[zeros(h(j)) \geq r] = \Pr[2^r \text{ divides } h(j)] = \frac{1}{2^r}.$$

- Hence

$$\mathbb{E}[Y_r] = \sum_{j:f_j>0} \mathbb{E}[X_{r,j}] = \frac{d}{2^r}$$

- Let *a* be smallest integer so that $2^{a+1/2} > 3d$
- Then by Markov's inequality

$$\Pr[t \geq a] = \Pr[Y_a > 0] = \Pr[Y_a \geq 1] \leq \frac{\mathbb{E}[Y_a]}{1} = \frac{d}{2^a} \leq \frac{\sqrt{2}}{3}$$

*t is bigger than a if at least one value hashes to more than a 0's*

## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process** $j$**:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

**Quality of estimate**

- Upper bounding $\Pr[\hat{d} < d/3]$
- Let b be the largest integer so that $2^{b+1/2} < d/3$
- Then, $\Pr[\hat{d} < d/3] = \Pr[t \leq b] = \Pr[Y_{b+1} = 0]$

*t is at most than b if no value hashes to more than b+1 0's*

- We use Chebychev's inequality to analyze this:

$$\Pr[Y_{b+1} = 0] \leq \Pr\left[|Y_{b+1} - \mathbb{E}[Y_{b+1}]| \geq \mathbb{E}[Y_{b+1}]\right] \leq \frac{\text{Var}[Y_{b+1}]}{(\mathbb{E}[Y_{b+1}])^2}$$

*It is sufficient to bound the probability that $Y_{b+1}$ is smaller than its expectation but it turns out to be easier to bound both sides using Chebychev's inequality*

- Recall b is the largest integer so that $2^{b+1/2} < d/3$
- And $\Pr[\hat{d} < d/3] \leq \frac{\text{Var}[Y_{b+1}]}{(\mathbb{E}[Y_{b+1}])^2} \leq \frac{1}{\mathbb{E}[Y_{b+1}]}$

**Bounding variance:**

$$\begin{aligned}
\text{Var}[Y_r] &= \mathbb{E}[Y_r^2] - \mathbb{E}[Y_r]^2 \\
&= \mathbb{E}[\sum_{j,j':f_j,f_{j'}>0} X_{r,j}X_{r,j'}] - \sum_{j,j':f_j,f_{j'}>0} \mathbb{E}[X_{r,j}]\,\mathbb{E}[X_{r,j'}] \\
\text{Pairwise independence!} \quad &= \sum_{j:f_j>0} \left(\mathbb{E}[X_{r,j}^2] - \mathbb{E}[X_{r,j}]^2\right) \\
&\leq \sum_{j:f_j>0} \mathbb{E}[X_{r,j}^2] = \sum_{j:f_j>0} \mathbb{E}[X_{r,j}] = \mathbb{E}[Y_r]
\end{aligned}$$

- Recall b is the largest integer so that $2^{b+1/2} < d/3$
- And $\Pr[\hat{d} < d/3] \leq \frac{\text{Var}[Y_{b+1}]}{(\mathbb{E}[Y_{b+1}])^2} \leq \frac{1}{\mathbb{E}[Y_{b+1}]} \leq \frac{2^{b+1}}{d} = \frac{2^{b+1/2}\sqrt{2}}{d} \leq \frac{\sqrt{2}}{3}$

**Calculating expectation:**

$$\mathbb{E}[Y_r] = \frac{d}{2^r} \text{ and so } \mathbb{E}[Y_{b+1}] = \frac{d}{2^{b+1}}$$

## Algorithm

**Initialization:** Choose a random hash function $h : [n] \to [n]$ from a pairwise independent family[2]. Let $z = 0$.

**Process** $j$**:** If $zeros(h(j)) > z$ then $z = zeros(h(j))$.

**Output:** $2^{z+1/2}$.

**Quality of estimate**

- We proved that

$$\Pr[\hat{d} > 3d] \leq \frac{\sqrt{2}}{3} \approx 0.47$$

- And

$$\Pr[\hat{d} < d/3] \leq \frac{\sqrt{2}}{3} \approx 0.47$$

- But we wish to have for a tiny $\delta$

$$\Pr[d(\sigma)/3 \leq \hat{d} \leq 3d(\sigma)] \geq 1 - \delta,$$

# !!Median!! Trick

- Run estimator T times independently in parallel to get estimates

$$\hat{d}_1, \hat{d}_2, \ldots, \hat{d}_T$$

- Output the median $\tilde{d}$

- The space requirement increases by a factor of T.
- We can select T to be $O(\log(1/\delta))$

- What is $\Pr[\tilde{d} < d/3]$?
- Let $Z_i$ be the indicator variable that $\hat{d}_i < d/3$
- Let $Z = \sum_{i=1}^{T} Z_i$

- $Z$ is a sum of *independent* 0/1 random variables and $\mathbb{E}[Z] \leq \frac{\sqrt{2}}{3}T \approx 0.47T$
- Hence, by Chernoff bound

$$\Pr[\tilde{d} < d/3] \leq \Pr[Z \geq T/2] \leq e^{\Omega(-T)}$$

Thus by selecting $T = C\log(1/\delta)$ for a large enough constant $C$ we get that this is at most $\delta/2$

Same analysis shows that $\Pr[\tilde{d} > 3d] \leq \delta/2$ for $T = C\log(1/\delta)$

# Summarize

- Using space $O(\log(1/\delta) \cdot \log(n))$

- We get estimator for #distinct elements with correctness guarantee

$$\Pr[d(\sigma)/3 \leq \tilde{d} \leq 3d(\sigma)] \geq 1 - \delta,$$