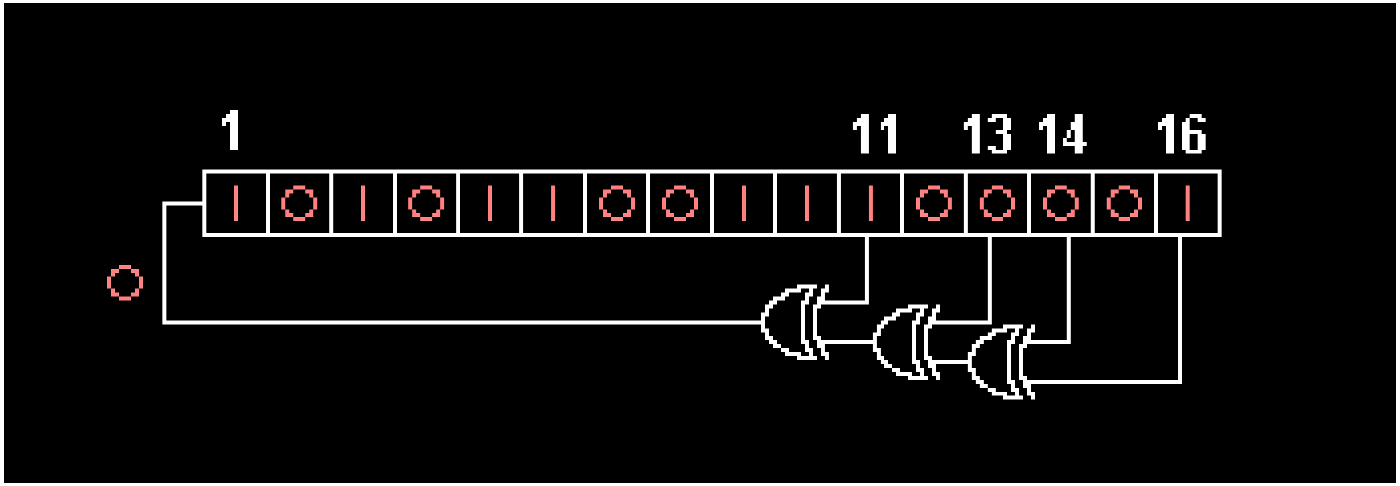
Pseudo-Randomness
and
Hashing

Pseudo-random number generators

- In previous class we assumed that a randomized algorithm has access to a sequence of IID RV.
- In reality, the program calls a pseudo-random function such as **RAND().**
- For the sake of simplicity lets assume that each call to Rand generates a single bit.
- The function **RAND** has a persistent d-bit variable called state and it operates in two modes:
  - **RAND(seed)** :
    - Set State=seed.
    - Return F(State)   # F returns a bit.
  - **RAND()** :
    - Update State=G(State) # G returns a new state.
    - Return F(State)

A simple random number generator: Linear Feedback Shift Register (LFSR)



- State: 16bit number.
- G: At each step the bits shift one step to the right and the least significant bit is replaced by the output of the circuit.
- F: output the most significant bit (bit 16) as the output of the random number generator;

What does pseudo-random mean?

- We assume that F and G are public knowledge.
- If we know the seed, then we know exactly what the sequence would be.
- Recall what it means that $X_1, X_2, \ldots, X_n$ are IID Binary RV with p= $\frac{1}{2}$:
  - $P(X_1 = x_1, X_2 = x_2, \ldots, X_n = x_n) = \frac{1}{2^n}$
  - $P(X_t = x_t | X_1 = x_1, \ldots, X_{t-1} = x_{t-1}) = 1/2$
- In words: it is impossible to predict the bit value of X_t from the values of the sequence so far: $X_1, \ldots, X_{t-1}$

Is it possible to predict the output of a pseudo-random number generator?

- The sequence is determined by the seed.
- If the state has d bits, there are at most 2^d possible seeds and therefor 2^d possible sequences.
- A brute force prediction algorithm: make a list of all possible seeds and the corresponding sequences, after each bit is revealed: delete the seeds whose sequences are inconsistent with the bit.
- It is not hard to show that predicting with the majority of surviving seeds will make no more than d mistakes.
- So pseudo random number generators **can** be predicted!
- Why do we call them (pseudo) random?
- Because the process of predicting them requires compute resources that are $\Omega(2^d)$.
- In other words, it is computational complexity that gives us pseudo-random number generators!

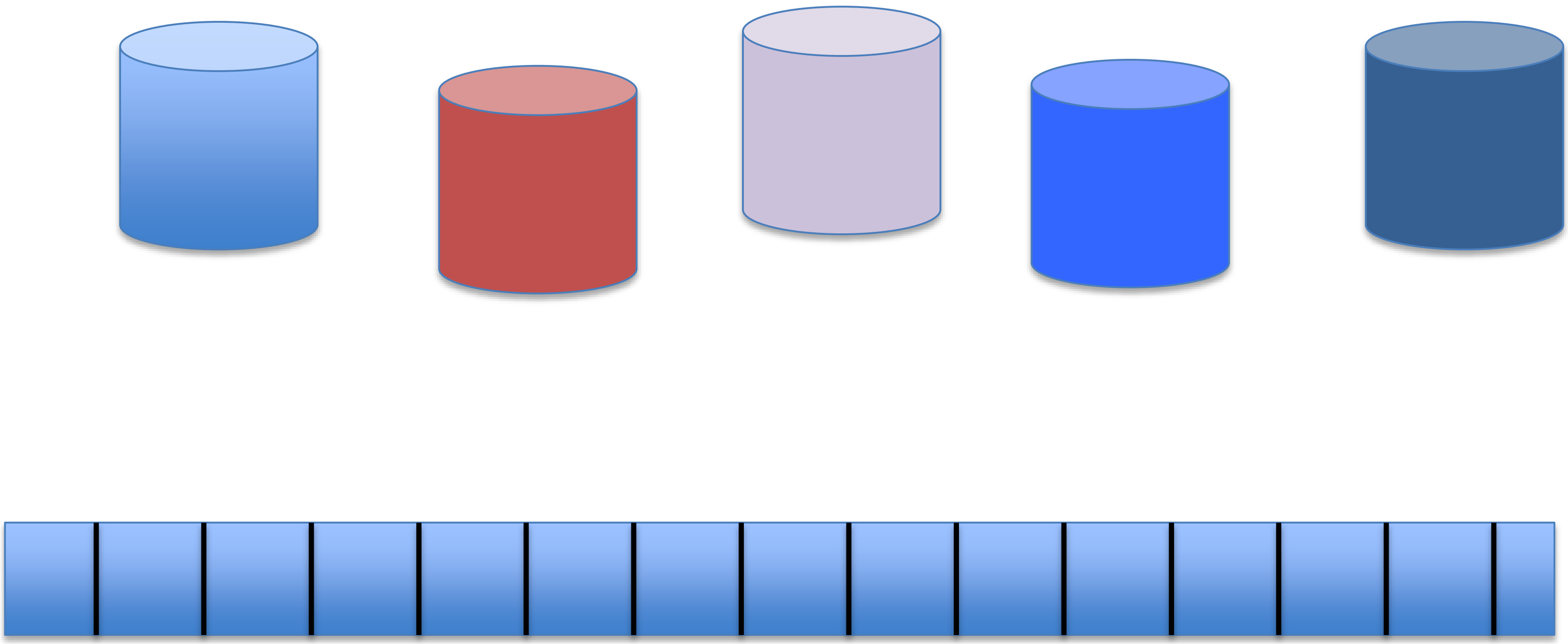
Pseudo-random vs random

- Random sequence: cannot be predicted.
- Pseudo-Random sequence: Takes prohibitive memory and time to predict.
- LFSR: a very weak pseudo-random number generator: sufficient for applications in signal processing.
- Cryptographically secure pseudo-random number generators: generators for which difficulty of prediction is mathematically proven.
  - Used in cryptographic systems such as RSA, OpenSSL,…
  - Volunerabilities sometimes related to problems with the pseudo-random generators.

Hash Functions and Hash Tables

Hashing

- A collection of n complex items from a very large set A (names, sounds, images)



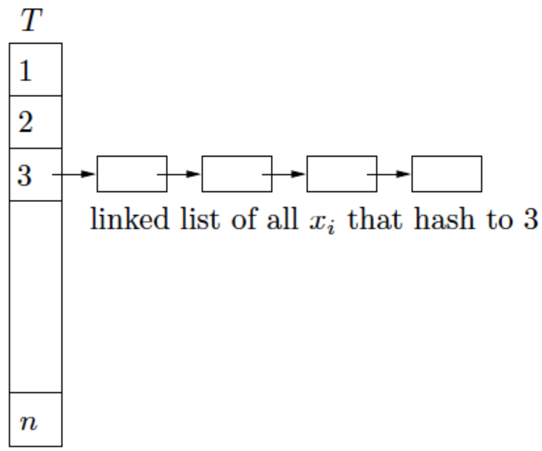
Hash Table: Array of pointers of size m

Hash Functions

- Hash functions map items from a very large space (images) to the number 1,…,n
- We want the mapping to **behave like a random function**: each item $a \in A$ is mapped to $\mathrm{H}(a) = i \in \{1, \dots, n\}$ with probability 1/n independently of $H(b), H(c), \dots$
- But, we also want the mapping **not to be random** in that $\mathrm{H}(a)$ always gives the same number.
- Solution: instead of one function, we use a **family** of hash functions, indexed by i: $H_i(a)$ at the start of the program we choose i at random to be a d-bit integer.
- We use a pseudo-random number generator **RAND()** to construct the hash function (assume RAND() outputs numbers in the range 1..n)
  - $H_i(a) = Rand(i \boxplus a)$ -- $i \boxplus a$ is the binary number created by concatenating the bits of the index i with the bits of the item a

Linked list Hash

- In the single occupancy hash table we discussed last class, each bin in the hash table contains (the pointer to) at most one item.
  - Inserting a new items requires finding an empty bin.
- In the a linked list hash table each bin contains a pointer to a linked list of items



- Unlike single occupancy hash tables, a new item is added to the end of the list starting at the bin to which it is hashed.
- One can store more items than the number of bins in the table.
- The time to add or fetch an item is proportional to the length of the list.

Expected number of items in each bin

m =number of bins $\qquad$ n = number of items

$X_i =$ number of items in bin i

Note: $X_1 + X_2 + \cdots + X_m = n$

$E(X_1 + X_2 + \cdots + X_m) = n$

What is $\mathrm{E}(X_i)$?

A. $\mathrm{E}(X_i) = n / n = 1$

B. $\mathrm{E}(X_i) = m / m = 1$

C. $\mathrm{E}(X_i) = n / m$

C. $\mathrm{E}(X_i) = n^2 / m$

Expected running time vs. worst case running time

- As the expected occupancy is n/m and we expect $m > n$, we define $c = \frac{n}{m}$
- Note that any hashing function achieves expected occupancy c. Even one which maps all items to the same bin.
- Beyond expected occupancy, we would like a guarantee that the maximal occupancy is small.
- An upper bound on $\max(X_1, X_2, \ldots, X_m)$ that holds with high probability over the random choice of the hashing function.

Bounding the max

If $\max\left(X_1, X_2, \ldots, X_m\right) \geq l$ then there must be i such that $X_i \geq l$

$$P(\max\left(X_1, X_2, \ldots, X_m\right) \geq l) \leq P(X_1 \geq l) + P(X_2 \geq l) + \cdots + P(X_m \geq l) = mP(X_1 \geq l)$$

Using the union bound.

Suppose n (the number of items) grows to infinity, $m = \frac{n}{c}$ and $l(n)$ is a function of n

c is fixed

Our goal, when computing an upper bound on the maximal occupancy is to

A. Find the fastest increasing $l(n)$ such that $P(X_1 \geq l(n))$ decreases to zero with n.

B. Find the slowest increasing $l(n)$ such that $mP(X_1 \geq l(n))$ decreases to zero with n.

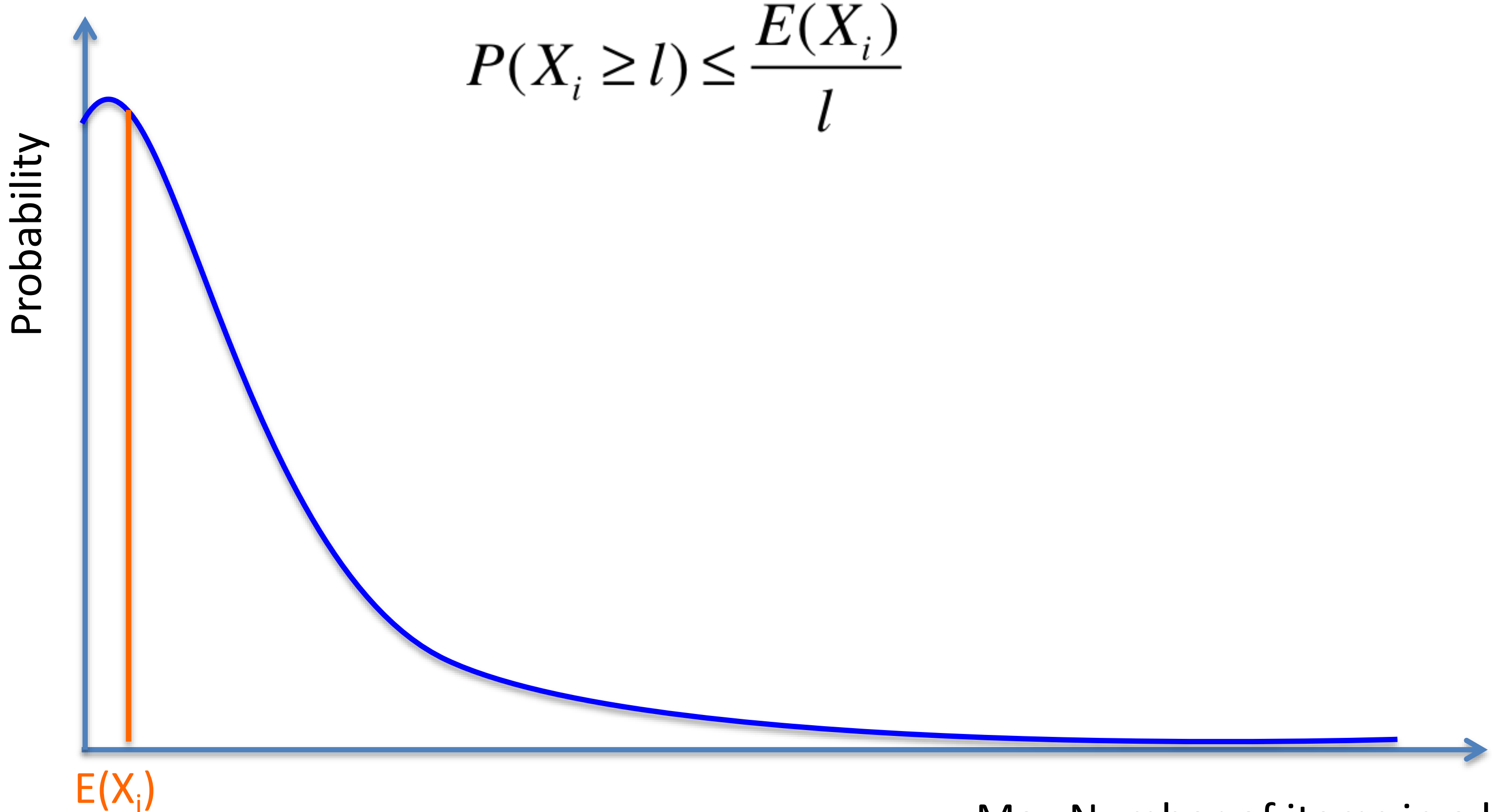
C. Find the slowest increasing $l(n)$ such that $P(X_1 \geq l(n))$ decreases to zero with n.

Bounding the deviation from the mean

- We know that the mean occupancy is n/m
- We want to show that the probability of a much higher occupancy is small.
- To do this we need to upper bound the deviation of the actual occupancy from the mean.
- We will do this using a sequence of better and better bounds, starting with Markov, then Chebyshev, and finally using the binomial coefficient.

Bounding using Markov inequality

$$P(X_i \geq l) \leq \frac{E(X_i)}{l}$$



Using Markov

$E(X_i) = n / m = 1 / c$

$P(X_i \geq l(n)) \leq \dfrac{E(X_i)}{l(n)} = \dfrac{1}{cl(n)}$

To get that $m(n)P(X_i \geq l(n)) \to 0$ we need that $\dfrac{m(n)}{cl(n)} \to 0$

$m(n) = cn$, so $\dfrac{n}{l(n)} \to 0$

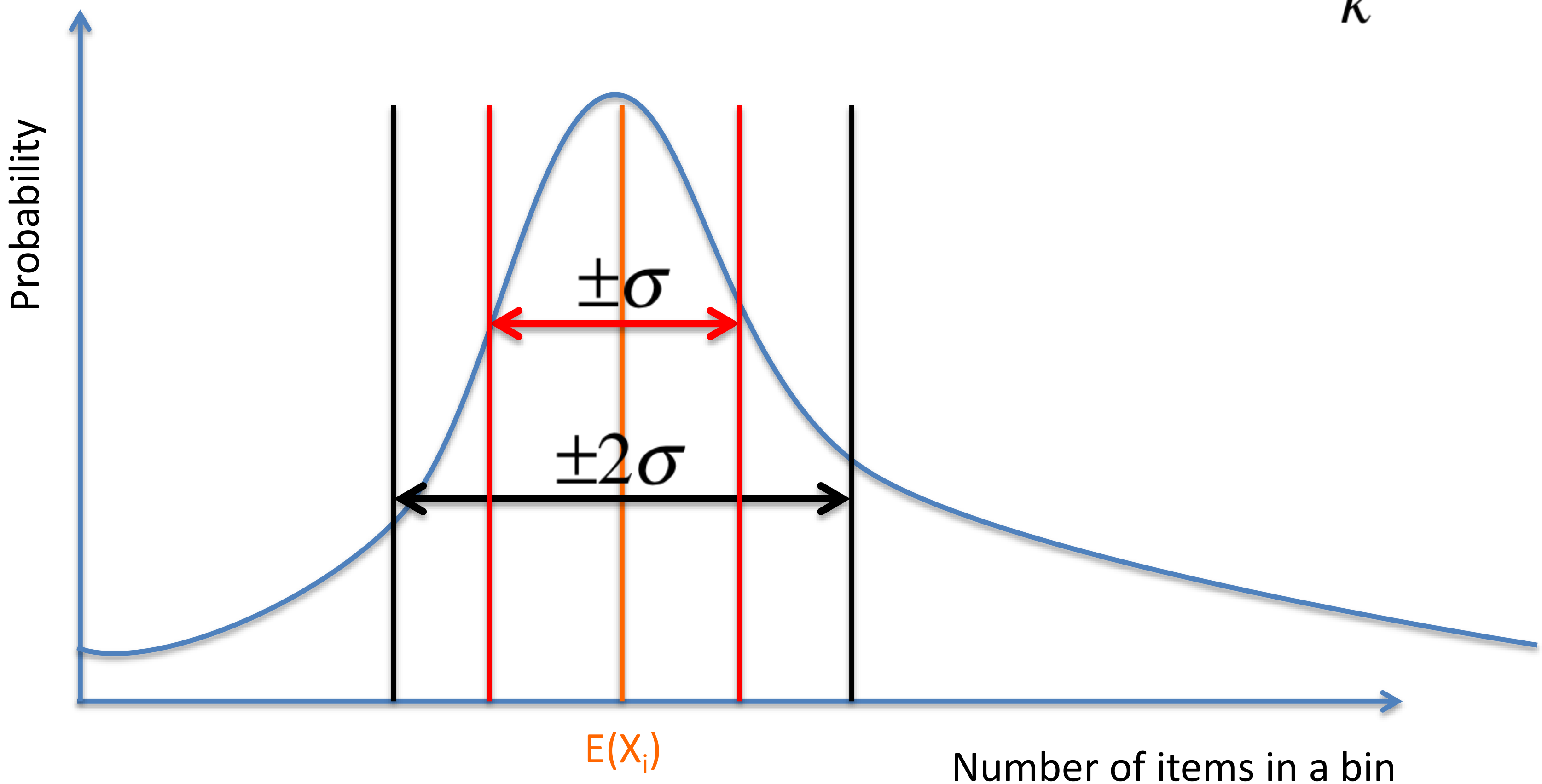
In other words, $l(n)$ increases faster than n.

But the max occupancy of any bin is n - the number of items

Using Markov generates a trivial bound.

Worst case bound using variance

Chebyshev's bound: $P\left(|X_i - E(X_i)| \ge k\sigma\right) \le \frac{1}{k^2}$



$\mathrm{var}(X_i) = E(X_i^2) - E(X_i)^2; \quad E(X_i) = 1/c$

To bound $E(X_i^2)$ we think of X_i as a sum:

$$X_i = X_{i1} + \cdots X_{in}; \qquad X_{ij} = \begin{cases} 1 & \text{if item } j \text{ is in bin } i \\ 0 & \text{otherwise} \end{cases}$$

X_{ij}, X_{ik} are independent if $j \neq k$

If two random variables X, Y are independent

then $E(XY) = E(X)E(Y)$

$E(X_{ij}) = P(X_{ij} = 1) = 1/m$

Which of the following is true?

A. $E(X_{ij}X_{ik}) = 1/m^2$

B. $E(X_{ij}X_{ik}) = 1/m$ if $j = k$; $\quad 1/m^2$ otherwise

C. $E(X_{ij}X_{ik}) = 1/m$

Breaking up the square into a sum

$$E(X_i^2) = E((X_{i1} + \cdots + X_{in})^2) = ?$$

A. $n^2 E(X_{ij}^2)$ B. $E(X_{i1} + \cdots + X_{in})^2$

C. $nE(X_{i1}^2) + n(n-1)E(X_{ij}X_{ik})$ D. $nE(X_{ii})$

Bounding the variance

$$nE(X_{i1}^2) + n(n-1)E(X_{ij}X_{ik})$$

$$E(X_{i1}^2) = E(X_{i1}) = P(\text{item 1 in bin } i) = 1/m$$

$$E(X_{ij}X_{ik}) = P(\text{item j in bin } i \text{ and item k in bin } i) =$$

$$= P(\text{item j in bin } i)P(\text{item k in bin } i) = 1/m^2$$

$$nE(X_{i1}^2) + n(n-1)E(X_{ij}X_{ik}) \le \frac{n}{m} + \frac{n(n-1)}{m^2} \le \frac{n}{m} + \frac{n^2}{m^2}$$

$$\text{if } m = cn, c \ge 1 \text{ we get } \operatorname{var}(X_i) \le \frac{1}{c} + \frac{1}{c^2} \le \frac{2}{c}; \quad \sigma \le \sqrt{\frac{2}{c}}$$

$$mP(X_i \geq l\sigma) \leq \frac{m}{l^2}$$

What is the slowest rate

that l can increase so that $(m/l^2) \to 0$ as $m \to \infty$?

A. $l = O(m^\alpha), \alpha > 1$ B. $l = O(\log m)$

C. $l = O(m^\alpha), \alpha > 1/2$ C. $l = O(m^\alpha), \ \alpha > 1/4$

A tighter bound using binomial coefficients

Instead of considering the mean and variance of X_i,

We upper bound the probability using the binomial coefficient:

$$P\left(X_i \geq l\right) = \sum_{i=l}^{n} \binom{n}{i} \left(\frac{1}{m}\right)^{i} \left(\frac{m-1}{m}\right)^{n-i} \leq \binom{n}{l} \left(\frac{1}{m}\right)^{l}$$

Because:

We can first select the set of l that falls in the bin i

and then select the remaining $n-$ in an arbitrary way.

Standard inequality (see cheat-Sheets) : $\binom{n}{l} \leq \left(\frac{ne}{l}\right)^{l}$

$$\binom{n}{l}\left(\frac{1}{m}\right)^{l} \leq \left(\frac{ne}{l}\right)^{l}\left(\frac{1}{m}\right)^{l} = \left(\frac{ne}{lm}\right)^{l} = \left(\frac{e}{lc}\right)^{l}$$

We need $\quad n\left(\frac{e}{lc}\right)^{l} \xrightarrow{n\to\infty} 0$

How fast does l need to grow in order to guarantee this?

A. $l = \log n$ B. $l = \log^2 n$ C. $l = \log\log n$ D. $l = \sqrt{n}$

The power of two choices

- We have shown that, if the ratio between items n and bins m is a constant $c = \frac{n}{m}$, and if $n \to \infty, m \to \infty$ then the probability that the highest occupancy is larger than $O(\log n)$ goes to zero.
- Is this the best that can be done?
- No!
- Mitzenmacher 1996, proposed the following ingenious method:
  - Instead of one hash function, use two (two different indices).
  - Given a new item a, compute both hash functions: $H_1(a), H_2(a)$
  - Compare the length of the lists in bin $H_1(a)$ and bin $H_2(a)$, add a to the end of the shorter list.
  - When retrieving search both lists for item a.
- Performance: maximal occupancy is $O(\log \log n)$ instead of $O(\log n)$

A general bound for binomial tails

Let $X_1, X_2, \ldots, X_n$ be independent, identically distributed

binary variables: $X_i = \begin{cases} 1 & \text{with prob. } p \\ 0 & \text{with prob. } 1-p \end{cases}$

Let $Y = \frac{1}{n}(X_1 + X_2 + \cdots + X_n)$; $E(Y) = p$; $\operatorname{var}(Y) = \frac{p(1-p)}{n}$

Then $P(Y \le k) = \sum_{i=0}^{k} \binom{n}{i} p^i (1-p)^{n-i}$ The binomial tail.

We can bound the binomial tail using any of the following

for any $k > 1$, $P(Y \ge kp) \le e^{-\frac{1}{3}np(k-1)^2}$

for any $k > 1$, $P\left(Y \le \frac{p}{k}\right) \le e^{-\frac{1}{2}np(1-1/k)^2}$

for any $\epsilon > 0$, $P(|Y - p| \ge \epsilon) \le 2e^{-2n\epsilon^2}$