Randomized Algorithms
Randomized Algorithms

• Algorithm can make random decisions
• Why randomized algorithms? Simple and efficient
• Examples: Symmetry-breaking, graph algorithms, quicksort, hashing, load balancing, cryptography, etc
Properties of Random Variables

For independent events $A$ and $B$,

$$\Pr(A \text{ and } B) = \Pr(A) \Pr(B)$$

Union Bound: For any two events $A$ and $B$,

$$\Pr(A \cup B) \leq \Pr(A) + \Pr(B)$$

Conditional Probability: For events $A$ and $B$,

$$\Pr(A \mid B) = \frac{\Pr(A \text{ and } B)}{\Pr(B)}$$
Global Min-Cut

Problem: Given undirected, unweighted graph $G = (V, E)$, find a cut $(A, V - A)$ which has the minimum number of edges across it.

Example:

Note: A global min-cut is not the same as an s-t min cut.

How can we find a global min-cut using $n - 1$ max-flows?

We can do better for the global min-cut.
Karger’s Min-Cut Algorithm

**Problem:** Given undirected, unweighted graph $G = (V, E)$, find a cut $(A, V - A)$ which has the minimum number of edges across it.

**Karger’s Min-Cut Algorithm:**

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges).
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into u}\}$.
Karger’s Min-Cut Algorithm

Problem: Given undirected, unweighted graph \( G = (V, E) \), find a cut \((A, V - A)\) which has the minimum number of edges across it.

Karger’s Min-Cut Algorithm:

1. Repeat until two nodes remain:
   - Pick an edge \( e = (u, v) \) in \( E \) uniformly at random. Collapse \( u \) and \( v \) into a single node (allowing multiple edges).
2. Let \( u, v \) be the nodes. Output \((U, V - U)\), where \( U = \{ \text{nodes that went into } u \} \).

Example:

```
#edges to pick from = 14
Pick (b, f) (probability 1/14)
```
**Problem:** Given undirected, unweighted graph $G = (V, E)$, find a cut $(A, V - A)$ which has the minimum number of edges across it.

**Karger’s Min-Cut Algorithm:**

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

**Example:**

$\#\text{edges to pick from} = 13$
Pick $(g, h)$ (probability $1/13$)
**Karger’s Min-Cut Algorithm**

**Problem:** Given undirected, unweighted graph $G = (V, E)$, find a cut $(A, V - A)$ which has the minimum number of edges across it.

**Karger’s Min-Cut Algorithm:**

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges).
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$.

**Example:**

```
 a ------- c ------- d
  |         |         |
  |  #edges to pick from = 12 |
  |                               |
  | b                               |
  |                                 |
  |  Pick (d, gh) (probability 1/6) |
  |                                 |
  | e ------- g                    |
```

Karger’s Min-Cut Algorithm

**Problem:** Given undirected, unweighted graph \( G = (V, E) \), find a cut \((A, V - A)\) which has the minimum number of edges across it.

**Karger’s Min-Cut Algorithm:**

1. Repeat until two nodes remain:
   - Pick an edge \( e = (u, v) \) in \( E \) uniformly at random. Collapse \( u \) and \( v \) into a single node (allowing multiple edges).
2. Let \( u, v \) be the nodes. Output \((U, V - U)\), where \( U = \{ \text{nodes that went into } u \} \).

**Example:**

\[
\begin{array}{c}
\text{a} \\
\text{b} \\
\text{f} \\
\text{c} \\
\text{e} \\
\text{d} \\
\text{gh}
\end{array}
\]

#edges to pick from = 10
Pick \((a, e)\) (probability 1/10)
Karger’s Min-Cut Algorithm

Problem: Given undirected, unweighted graph $G = (V, E)$, find a cut $(A, V - A)$ which has the minimum number of edges across it.

Karger’s Min-Cut Algorithm:

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges).
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

Example:

#edges to pick from = 9
Pick $(ae, bf)$ (probability $4/9$)
Karger’s Min-Cut Algorithm

**Problem:** Given undirected, unweighted graph $G = (V, E)$, find a cut $(A, V - A)$ which has the minimum number of edges across it.

**Karger’s Min-Cut Algorithm:**

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges).
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$.

**Example:**

![Graph with nodes aebf and dgh connected by lines]

- #edges to pick from = 5
- Pick (c, dgh) (probability 3/5)
Karger’s Min-Cut Algorithm

**Problem:** Given undirected, unweighted graph $G = (V, E)$, find a cut $(A, V - A)$ which has the minimum number of edges across it

**Karger’s Min-Cut Algorithm:**

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

**Example:**

Done! Output $(aebf, cdgh)$

**Original Graph:**
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:
1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

Fact 1. If there are $n$ nodes, then the average degree of a node is $2|E|/n$
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

Fact 1. If there are $n$ nodes, then the average degree of a node is $2|E|/n$

Proof: Total degree of $n$ nodes = $2|E|$, $n$ nodes in total
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

Fact 1. If there are $n$ nodes, then the average degree of a node is $2|E|/n$

Fact 2. The minimum cut size is at most $2|E|/n$

Proof: From Fact 1, there is at least one node $x$ with degree at most $2|E|/n$
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:
1. Repeat until two nodes remain:
   Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

Fact 1. If there are $n$ nodes, then the average degree of a node is $2|E|/n$
Fact 2. The minimum cut size is at most $2|E|/n$
Proof: From Fact 1, there is at least one node $x$ with degree at most $2|E|/n$
The cut $(x, V - x)$ has $\deg(x)$ edges. The minimum cut size is thus at most $\deg(x) \leq 2|E|/n$
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:
1. Repeat until two nodes remain:
   Pick an edge e = (u, v) in E uniformly at random. Collapse u and v into a single node (allowing multiple edges)
2. Let u, v be the nodes. Output (U, V - U), where U = \{nodes that went into u\}

Fact 1. If there are n nodes, then the average degree of a node is 2|E|/n

Fact 2. The minimum cut size is at most 2|E|/n

Fact 3. If we choose an edge uniformly at random (uar), the probability that it lies across the min cut is at most 2/n

Proof: Follows directly from Fact 2
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:
1. Repeat until two nodes remain:
   Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single
   node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

Fact 1. If there are $n$ nodes, then the average degree of a node is $2|E|/n$

Fact 2. The minimum cut size is at most $2|E|/n$

Fact 3. If we choose an edge uniformly at random (uar), the probability that it lies
across the min cut is at most $2/n$
Karger’s Algorithm: Analysis

Karger’s Min-Cut Algorithm:

1. Repeat until two nodes remain:
   Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

Fact 1. If there are $n$ nodes, then the average degree of a node is $2|E|/n$

Fact 2. The minimum cut size is at most $2|E|/n$

Fact 3. If we choose an edge uniformly at random (uar), the probability that it lies across the min cut is at most $2/n$

Observe: Bad case is when the algorithm selects an edge $e$ across the min-cut
**Karger’s Algorithm: Analysis**

Karger’s Min-Cut Algorithm:

1. Repeat until two nodes remain:
   - Pick an edge $e = (u, v)$ in $E$ uniformly at random. Collapse $u$ and $v$ into a single node (allowing multiple edges)
2. Let $u, v$ be the nodes. Output $(U, V - U)$, where $U = \{\text{nodes that went into } u\}$

**Fact 1.** If there are $n$ nodes, then the average degree of a node is $2|E|/n$

**Fact 2.** The minimum cut size is at most $2|E|/n$

**Fact 3.** If we choose an edge uniformly at random (uar), the probability that it lies across the min cut is at most $2/n$

Observe: Bad case is when the algorithm selects an edge $e$ across the min-cut

$$\Pr[\text{Output cut = min-cut}] = \Pr[\text{First selected edge not in min-cut}] \times \Pr[\text{Second selected edge not in min-cut} | \text{1st edge}] \times \ldots$$

$$\geq \left(1 - \frac{2}{n}\right) \left(1 - \frac{2}{n-1}\right) \left(1 - \frac{2}{n-2}\right) \ldots \left(1 - \frac{2}{3}\right)$$

$$= \frac{n-2}{n} \cdot \frac{n-3}{n-1} \cdot \frac{n-4}{n-2} \cdot \frac{2}{4} \cdot \frac{1}{3} = \frac{2}{n(n-1)}$$

Thus, outputs min-cut w.p. $2/n^2$; can run it $O(n^2)$ times and pick the best of the outputs
Types of Randomized Algorithms

Monte Carlo Algorithm:
Always has the same running time
Not guaranteed to return the correct answer (returns a correct answer only with some probability)

Las Vegas Algorithm:
Always guaranteed to return the correct answer
Running time fluctuates (probabilistically)

Fact: Suppose a Monte Carlo algorithm succeeds w.p. $p$. Then, it can be made to succeed w.p. $1 - t$ for any (small) $t$ by running it $O(\log (1/t)/p)$ time

Proof: Suppose we run the algorithm $k$ times. Then,
\[ \Pr[\text{Algorithm is wrong every time}] = (1 - p)^k < t \]
when $k = O(\log (1/t)/p)$
### Expectation

Given discrete random variable $X$, which takes $m$ values $x_i$ w.p. $p_i$, the expectation $E[X]$ is defined as:

$$E[X] = \sum_{i=1}^{m} x_i \cdot \Pr[X = x_i] = \sum_{i=1}^{m} x_i p_i$$

#### Examples:

1. Let $X = 1$ if a fair coin toss comes up heads, 0 ow. What is $E[X]$?

2. We are tossing a coin with head probability $p$, tail probability $1 - p$. Let $X = \#$independent flips until first head. What is $E[X]$?

   $$\Pr[X = j] = \frac{p}{1 - p} \cdot (1 - p)^{j-1}$$

   - head on toss $j$
   - first $j-1$ tails

   $$E[X] = \sum_{j=1}^{\infty} j \cdot p(1 - p)^{j-1} = \frac{p}{1 - p} \sum_{j=0}^{\infty} j(1 - p)^j = \frac{p}{1 - p} \cdot \frac{1 - p}{p^2} = \frac{1}{p}$$
Given discrete random variable $X$, which takes $m$ values $x_i$ w.p. $p_i$, the expectation $E[X]$ is defined as:

$$E[X] = \sum_{i=1}^{m} x_i \cdot \Pr[X = x_i] = \sum_{i=1}^{m} x_i p_i$$

**Linearity of Expectation:** $E[X + Y] = E[X] + E[Y]$
Expectation

Given discrete random variable $X$, which takes $m$ values $x_i$ w.p. $p_i$, the expectation $E[X]$ is defined as:

$$E[X] = \sum_{i=1}^{m} x_i \cdot \Pr[X = x_i] = \sum_{i=1}^{m} x_i p_i$$

Linearity of Expectation: $E[X + Y] = E[X] + E[Y]$

Example: Guessing a card

Shuffle $n$ cards, then turn them over one by one. Guess what the next card is. How many guesses are correct on expectation?

Let $X_i = 1$ if guess $i$ is correct, 0 otherwise

$$\Pr[X_i = 1] = \frac{1}{n - i + 1} \quad E[X_i] = \frac{1}{n - i + 1}$$

Expected # of correct guesses = $E[\sum_{i=1}^{n} X_i] = \sum_{i=1}^{n} E[X_i] = \sum_{i=1}^{n} \frac{1}{i} = \Theta(\log n)$

What if we insert the selected card into the pile randomly and pull another?
Expectation

Given discrete random variable $X$, which takes $m$ values $x_i$ w.p. $p_i$, the expectation $E[X]$ is defined as:

$$E[X] = \sum_{i=1}^{m} x_i \cdot \Pr[X = x_i] = \sum_{i=1}^{m} x_i p_i$$

**Linearity of Expectation:** $E[X + Y] = E[X] + E[Y]

**Example: Coupon Collector’s Problem**

Balls tossed randomly into $n$ bins. How many balls in expectation before each bin has a ball?

Let $X_j = \text{time spent when there are exactly } j \text{ non-empty bins (Phase } j)$
Let $X = \text{total steps} = X_1 + X_2 + \ldots + X_{n-1}$

We move from phase $j$ to $j+1$ when a ball hits one of $n-j$ bins, so w.p. $(n-j) / n$

Therefore, $E[X_j] = n/(n - j)$ [From previous slide on expected waiting times]

$E[X] = E[X_1 + \ldots + X_{n-1}] = E[X_1] + \ldots + E[X_{n-1}] = n + n/2 + \ldots + n/(n-1) = \Theta(n \log n)$
Expectation

Given discrete random variable $X$, which takes $m$ values $x_i$ w.p. $p_i$, the expectation $E[X]$ is defined as:

$$E[X] = \sum_{i=1}^{m} x_i \cdot Pr[X = x_i] = \sum_{i=1}^{m} x_i p_i$$

Linearity of Expectation: $E[X + Y] = E[X] + E[Y]$

Example: Birthday Paradox

$m$ balls tossed randomly into $n$ bins. What is the expected #collisions?

For $1 \leq i < j \leq m$, let $X_{ij} = 1$ if balls $i$ and $j$ land in the same bin, 0 otherwise

$Pr[ X_{ij} = 1 ] = 1/n$, so $E[X_{ij}] = 1/n$

So, expected number of collisions from tossing $m$ balls = $\sum_{i,j} E[X_{ij}] = \left(\frac{m}{2}\right) = \frac{m(m-1)}{2n}$

So when $m < \sqrt{2n}$, expected #collisions < 1; otherwise, it's more
Variance

Given a random variable $X$, its variance $\text{Var}[X]$ is defined as:
\[ \text{Var}(X) = \mathbb{E}[(X - \mathbb{E}[X])^2] \]

Properties of Variance:

1. $\text{Var}(X) = \mathbb{E}[X^2] - (\mathbb{E}[X])^2$
2. If $X$ and $Y$ are independent random variables, then, $\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$
3. For any constants $a$ and $b$, $\text{Var}(aX + b) = a^2 \text{Var}(X)$

Variance of a random variable measures its “spread”

High Variance Distribution

Low Variance Distribution
Computing Percentiles

An f-th percentile is the value below which f percent of observations fall.

Given array A[1..n], find the k-th smallest element in A.

Example: Median = n/2-th smallest element = 50th percentile

How to compute the median in O(n log n) time?
Randomized Selection

Given array A[1..n], find the k-th smallest element in A

A Divide and Conquer Algorithm:
Select(A, k)
  1. Pick an item v in A
  2. Let:
      \( A_L \) = all elements in A that are < v
      \( A_M \) = all elements in A that are = v
      \( A_R \) = all elements in A that are > v
  3. Return:
      Select(\( A_L \), k)                   if k <= |\( A_L \) |
      v                                     if |\( A_L \) | < k <= |\( A_L \) | + |\( A_M \) |
      Select(\( A_R \), k - |\( A_L \) | - |\( A_M \) |)  otherwise

Example:
\[
A = \begin{bmatrix} 2 & 36 & 5 & 21 & 8 & 13 & 11 & 20 & 5 & 4 & 1 \end{bmatrix} \quad v = 5
\]
\[
A_L = \begin{bmatrix} 2 & 4 & 1 \end{bmatrix} \quad A_M = \begin{bmatrix} 5 & 5 \end{bmatrix} \quad A_R = \begin{bmatrix} 36 & 21 & 8 & 13 & 11 & 20 \end{bmatrix}
\]
Randomized Selection

Given array A[1..n], find the k-th smallest element in A

A Divide and Conquer Algorithm:

Select(A, k)
1. Pick an item v in A
2. Let:
   \( A_L = \) all elements in A that are \(< v \)
   \( A_M = \) all elements in A that are \(= v \)
   \( A_R = \) all elements in A that are \(> v \)
3. Return:
   \( \text{Select}(A_L, k) \) if \( k \leq |A_L| \)
   \( v \) if \( |A_L| < k \leq |A_L| + |A_M| \)
   \( \text{Select}(A_R, k - |A_L| - |A_M|) \) otherwise

How to select v?
Pick v uniformly at random in 1..n

Worst case: v = smallest or largest element
Time \( T(n) = O(n) \) (for splitting) + \( T(n-1) \)
Solving the recurrence, \( T(n) = O(n^2) \)
\[
\Pr(\text{Worst Case}) = \frac{2}{n} \cdot \frac{2}{n-1} \cdot \ldots \cdot \frac{2}{2} \approx \frac{2^n}{n!}
\]

Best case: v is the k-th element
Time taken \( T(n) = O(n) \) (for splitting) + O(1)
\( T(n) = O(n) \)
\[
\Pr(\text{Best Case}) \geq \frac{1}{n}
\]
Randomized Selection

Given array $A[1..n]$, find the $k$-th smallest element in $A$

A Divide and Conquer Algorithm:

Select($A, k$)
1. Pick an item $v$ in $A$
2. Let:
   - $A_L$ = all elements in $A$ that are $<$ $v$
   - $A_M$ = all elements in $A$ that are $=$ $v$
   - $A_R$ = all elements in $A$ that are $>$ $v$
3. Return:
   - Select($A_L, k$)                     if $k \leq |A_L|$
   - $v$                                     if $|A_L| < k \leq |A_L| + |A_M|$
   - Select($A_R, k - |A_L| - |A_M|$)  otherwise

How to select $v$?
Pick $v$ uniformly at random in $1..n$

Average case: Let $T(n)$ be the expected running time on an array of size $n$

Lucky split: $v$ is the $m$-th smallest element, for $n/4 \leq m \leq 3n/4$. $\Pr[$Lucky Split$] = 1/2$

$$T(n) \leq \text{Time to split} + \Pr[$Lucky Split$] \times T(\text{array of size} \leq 3n/4)$$
$$+ \Pr[$Unlucky Split$] \times T(\text{array of size} \leq n)$$
$$\leq n + (1/2) T(3n/4) + (1/2) T(n)$$

Solving, $T(n) \leq T(3n/4) + 2n = O(n)$
Randomized Sorting

Given array $A[1..n]$, sort $A$

QuickSort:
Sort($A$)
1. Pick an item $v$ in $A$
2. Let:
   - $A_L$ = all elements in $A$ that are $< v$
   - $A_M$ = all elements in $A$ that are $= v$
   - $A_R$ = all elements in $A$ that are $> v$
3. Return:
   - Sort($A_L$) + $A_M$ + Sort($A_R$)

How to select $v$?
Pick $v$ uniformly at random in $1..n$

Best case: $[n/2, n/2]$ split
Running Time $T(n)$ = (time to split) + $2T(n/2) = n + 2T(n/2)$
Solving, $T(n) = O(n \log n)$

Worst case: $[1, n-1]$ split
Running Time $T(n)$ = (time to split) + $T(1) + T(n-1) = n + 1 + T(n-1)$
Solving, $T(n) = O(n^2)$
Given array A[1..n], sort A

**QuickSort:**
Sort(A)
1. Pick an item v in A
2. Let:
   - \( A_L \) = all elements in A that are < v
   - \( A_M \) = all elements in A that are = v
   - \( A_R \) = all elements in A that are > v
3. Return:
   \( \text{Sort}(A_L) + A_M + \text{Sort}(A_R) \)

How to select v?
Pick v uniformly at random in 1..n

**Average case:** Let \( T(n) \) be the expected running time on an array of size n

\[
T(n) = \text{Time to split} + \text{expected time to sort } A_L \text{ and } A_R
\]

\[
= n + \sum_{i=1}^{n} \Pr[v \text{ is the } i^{th} \text{ smallest element in } A] \cdot (T(i) + T(n - i))
\]

\[
= n + \frac{1}{n} \sum_{i=1}^{n} (T(i) + T(n - i))
\]

**Exercise:** Solve the recurrence to \( T(n) = O(n \log n) \). Use:

\[
\sum_{k=1}^{n-1} k \log k \leq \frac{1}{2} n^2 \log n - \frac{1}{8} n^2
\]
**MAX 3SAT**

**3-SAT Problem:** Given a boolean formula $F$ consisting of:
- $n$ variables $x_1, x_2, ..., x_n$
- $m$ clauses of size 3 of the form $x_i \lor x_j \lor x_k$ or $\neg(x_i) \lor x_j \lor x_k$
Is there an assignment of true/false values to variables s.t. all clauses are true?

**Example:**
3 variables: $x_1, ..., x_3$

Clauses:
$x_1 \lor x_2 \lor x_3$, $\neg(x_1) \lor x_2 \lor x_3$, $x_1 \lor \neg(x_2) \lor x_3$, $x_1 \lor x_2 \lor \neg(x_3)$, $\neg(x_1) \lor \neg(x_2) \lor x_3$, $\neg(x_1) \lor x_2 \lor \neg(x_3)$, $x_1 \lor \neg(x_2) \lor \neg(x_3)$, $\neg(x_1) \lor \neg(x_2) \lor \neg(x_3)$

Unsatisfiable!

**MAX-3SAT Problem:** Given a boolean formula $F$ consisting of:
- $n$ variables $x_1, x_2, ..., x_n$
- $m$ clauses of size 3 of the form $x_i \lor x_j \lor x_k$ or $\neg(x_i) \lor x_j \lor x_k$
Find an assignment of true/false values to variables to satisfy the most clauses

**Example:**
Any assignment satisfies 7 out of 8 clauses
**MAX-3SAT**

**MAX-3SAT Problem:** Given a boolean formula $F$ consisting of:
- $n$ variables $x_1, x_2, ..., x_n$
- $m$ clauses of size 3 of the form $x_i \lor x_j \lor x_k$ or $\neg(x_i) \lor x_j \lor x_k$
Find an assignment of true/false values to variables to satisfy the most clauses

**A Randomized MAX-3SAT Algorithm:**
Set each variable to 0/1 independently with probability 1/2 each

Define: $Z_i = 1$, if clause $i$ is satisfied by the assignment
$Z_i = 0$, otherwise

$\Pr[Z_i = 0] = (1/2) \cdot (1/2) \cdot (1/2) = 1/8$
$E[Z_i] = 7/8$

Let $Z = Z_1 + Z_2 + ... + Z_m = \#satisfied$ clauses

$E[Z] = E[Z_1 + Z_2 + ... + Z_m] = E[Z_1] + E[Z_2] + ... + E[Z_m] = 7m/8 = E[\#satisfied clauses]$

How to get a solution with $\geq 7m/8$ satisfied clauses?

**Fact:** $P = \Pr[Solution has \geq 7m/8 satisfied clauses] \geq 1/8m$

**Proof:** Let $p_j = \Pr[Solution has j satisfied clauses]$, $k = \text{largest integer} < 7m/8$

$$\frac{7m}{8} = E[Z] = \sum_{j=0}^{k} jp_j + \sum_{j=k+1}^{m} jp_j \leq k + mP \quad \rightarrow \quad P \geq \frac{\frac{7m}{8} - k}{m} \geq \frac{1}{8m}$$

As $m, k$ are integers, $7m/8 - k \geq 1/8$
MAX-3SAT Problem: Given a boolean formula $F$ consisting of:
  - $n$ variables $x_1, x_2, \ldots, x_n$
  - $m$ clauses of size 3 of the form $x_i \lor x_j \lor x_k$ or $\neg(x_i) \lor x_j \lor x_k$

Find an assignment of true/false values to variables to satisfy the most clauses

A Randomized MAX-3SAT Algorithm:
Set each variable to 0/1 independently with probability 1/2 each

Define: $Z_i = 1$, if clause $i$ is satisfied by the assignment
$Z_i = 0$, otherwise

$\Pr[Z_i = 0] = (1/2) \cdot (1/2) \cdot (1/2) = 1/8$
$E[Z_i] = 7/8$

Let $Z = Z_1 + Z_2 + \ldots + Z_m = \text{#satisfied clauses}$
$E[Z] = E[Z_1 + Z_2 + \ldots + Z_m] = E[Z_1] + E[Z_2] + \ldots + E[Z_m] = 7m/8 = E[\text{#satisfied clauses}]$

How to get a solution with $\geq 7m/8$ satisfied clauses?

Fact: $\Pr[\text{Solution has } \geq 7m/8 \text{ satisfied clauses}] \geq 1/8m$

Solution: Run algorithm $8m \log(1/t)$ times independently. With probability $1 - t$, there will be a solution with at least $7m/8$ satisfied clauses.
Variance

Given a random variable $X$, its variance $\text{Var}[X]$ is defined as:

$$\text{Var}(X) = E[(X - E[X])^2]$$

If $X$ and $Y$ are independent random variables, then,

$$\text{Var}(X + Y) = \text{Var}(X) + \text{Var}(Y)$$

What if $X$ and $Y$ are not independent?

$$\text{Cov}(X, Y) = E(XY) - E(X)E(Y) = E((X - E[X])(Y - E[Y]))$$

$\text{Cov}(X, Y)$ measures how closely $X, Y$ are “correlated” (in a loose sense)

$$\text{Var}(X+Y) = \text{Var}(X) + \text{Var}(Y) + 2 \text{Cov}(X,Y) \quad \text{[for general r.v. } X \text{ and } Y\text{]}$$

What is $\text{Cov}(X,Y)$ if $X$ and $Y$ are independent?
Inequality 1: Markov’s Inequality

If $X$ is a random variable which takes non-negative values, and $a > 0$, then

$$\Pr[X \geq a] \leq \frac{E[X]}{a}$$

Example: $n$ tosses of an unbiased coin. $X = \#\text{heads}$

$E[X] = n/2$. Let $a = 3n/4$. By Markov’s Inequality, $\Pr(X \geq a) \leq 2/3$. But what is it really?

$$\Pr[X \geq \frac{3n}{4}] \leq \left(\binom{n}{3n/4} + \binom{n}{3n/4+1} + \ldots + \binom{n}{n}\right) \cdot 2^{-n} \leq n \cdot 2^{-n} \cdot \binom{n}{n/4}$$

$$\leq n \cdot 2^{-n} \cdot (4e)^{n/4} \leq n(e/4)^{n/4} < (e/3)^{n/4} \quad \text{for large } n$$

Fact: If $n \geq k$

$$\left(\frac{n}{k}\right)^k \leq \left(\frac{n}{k}\right) \leq \left(\frac{ne}{k}\right)^k$$

Summary: Markov’s inequality can be weak, but it only requires $E[X]$ to be finite!
Inequality 2: Chebyshev’s Inequality

If \( X \) is a random variable and \( a > 0 \), then

\[
\Pr( X \geq E[X] + a ) \\ 
\Pr( X \leq E[X] - a ) \\ 
\Pr( X \geq E[X] + a ) \\ 
\Pr( X \leq E[X] - a )
\]

\[
E[X] = \frac{n}{2} \quad \text{Var}[X] = \frac{n}{4} \quad \text{(how would you compute this?)}
\]

Example: \( n \) tosses of an unbiased coin. \( X = \# \text{heads} \)

\[
E[X] = \frac{n}{2} \quad \text{Var}[X] = \frac{n}{4}
\]

From last slide, \( \Pr(X \geq 3n/4) \leq c^{n/4} \) for some constant \( c < 1 \), and large enough \( n \)

Let \( a = n/4 \), so that we compute \( \Pr(X \geq 3n/4) \). By Chebyshev, \( \Pr(X \geq 3n/4) \leq 4/n \)

Summary: Chebyshev’s inequality can also be weak, but only requires finite \( \text{Var}[X], E[X] \)
Let $X_1, .., X_n$ be independent 0/1 random variables, and $X = X_1 + .. + X_n$. Then, for any $t>0$,

$$
Pr(X \geq (1 + t)E[X]) \leq \left( \frac{e^t}{(1 + t)^{1+t}} \right)^{E[X]}
$$

Moreover, for $t < 1$,

$$
Pr(X \leq (1 - t)E[X]) \leq e^{-\frac{1}{2}t^2E[X]}
$$

**Example:** $n$ tosses of an unbiased coin. $X=#$heads$= X_1 + .. + X_n$ where $X_i=1$ if toss $i =$head

$E[X] = n/2$. $Pr[ X >= 3n/4] = Pr[ X >= (1 + 1/2) E[X])$, so $t = 1/2$

Thus from Chernoff Bounds,

$$
Pr(X \geq 3n/4) \leq \left( e^{1/2} \cdot (2/3)^{3/2} \right)^{n/2} \leq (0.88)^{n/2}
$$

**Summary:** Stronger bound, but needs independence!
Let $X_1, \ldots, X_n$ be independent 0/1 random variables, and $X = X_1 + \ldots + X_n$. Then, for any $t > 0$,

$$\Pr(X \geq (1 + t)E[X]) \leq \left( \frac{e^t}{(1 + t)^{1+t}} \right)^{E[X]}$$

Moreover, for $t < 1$,

$$\Pr(X \leq (1 - t)E[X]) \leq e^{-\frac{1}{2}t^2E[X]}$$

**Simplified Version:**

Let $X_1, \ldots, X_n$ be independent 0/1 random variables, and $X = X_1 + \ldots + X_n$. Then, for $t < 2e - 1$,

$$\Pr(X > (1 + t)E[X]) \leq e^{-t^2E[X]/4}$$
Randomized Algorithms

- Contention Resolution
- Some Facts about Random Variables
- Global Minimum Cut Algorithm
- Randomized Selection and Sorting
- Max 3-SAT
- Three Concentration Inequalities
- Hashing and Balls and Bins
Hashing and Balls-n-Bins

**Problem:** Given a large set $S$ of elements $x_1, \ldots, x_n$, store them using $O(n)$ space s.t it is easy to determine whether a query item $q$ is in $S$ or not

**Popular Data Structure:** A Hash table

**Algorithm:**
1. Pick a completely random function $h : U \rightarrow \{1, \ldots, n\}$
2. Create a table of size $n$, initialize it to null
3. Store $x_i$ in the linked list at position $h(x_i)$ of table
4. For a query $q$, look at the linked list at location $h(q)$ of table to see if $q$ is there

What is the query time of the algorithm?
Hashing and Balls-n-Bins

**Problem:** Given a large set $S$ of elements $x_1, .., x_n$, store them using $O(n)$ space s.t it is easy to determine whether a query item $q$ is in $S$ or not.

![Table]

**Algorithm:**
1. Pick a completely random function $h$
2. Create a table of size $n$, initialize it to null
3. Store $x_i$ in the linked list at position $h(x_i)$ of table
4. For a query $q$, check the linked list at location $h(q)$

**Average Query Time:** Suppose $q$ is picked at random s.t it is equally likely to hash to $1, .., n$. What is the expected query time?

**Expected Query Time**

$$
\text{Expected Query Time} = \sum_{i=1}^{n} \Pr[q \text{ hashes to location } i] \cdot (\text{length of list at } T[i])
$$

$$
= \frac{1}{n} \sum_{i} (\text{length of list at } T[i]) = \frac{1}{n} \cdot n = 1
$$
**Problem:** Given a large set $S$ of elements $x_1, .., x_n$, store them using $O(n)$ space s.t it is easy to determine whether a query item $q$ is in $S$ or not

**Algorithm:**
1. Pick a completely random function $h$
2. Create a table of size $n$, initialize it to null
3. Store $x_i$ in the linked list at position $h(x_i)$ of table
4. For a query $q$, check the linked list at location $h(q)$

**Worst Case Query Time:** For any $q$, what is the query time? (with high probability over the choice of hash functions)

**Equivalent to the following Balls and bins Problem:**
Suppose we toss $n$ balls u.a.r into $n$ bins. What is the max #balls in a bin with high probability?

With high probability (w.h.p) = With probability $1 - 1/poly(n)$
Balls and Bins, again

Suppose we toss \( n \) balls u.a.r into \( n \) bins. What is the max load of a bin with high probability?

Some Facts:

1. The expected load of each bin is 1

2. What is the probability that each bin has load 1?

\[
\text{Probability} = \frac{\text{# permutations}}{\text{# ways of tossing } n \text{ balls to } n \text{ bins}} = \frac{n!}{n^n}
\]

3. What is the expected number of empty bins?

\[
\Pr[\text{Bin } i \text{ is empty}] = \left(1 - \frac{1}{n}\right)^n
\]

\[
E[\text{# empty bins}] = n \left(1 - \frac{1}{n}\right)^n = \Theta(n) \quad (1 - 1/n)^n \text{ lies between } 1/4 \text{ and } 1/e \text{ for } n \geq 2)
\]
Balls and Bins

Suppose we toss $n$ balls u.a.r into $n$ bins. What is the max load of a bin with high probability?

Let $X_i$ = #balls in bin $i$

$$
\Pr(X_i \geq t) \leq \binom{n}{t} \frac{1}{n^t} \leq \left(\frac{ne}{t}\right)^t \frac{1}{n^t} \leq \left(\frac{e}{t}\right)^t \leq \frac{1}{n^2}
$$

From Fact

Would like this for whp condition

Let $t = \frac{c \log n}{\log \log n}$ for constant $c$

$$
\log \left(\frac{t}{e}\right)^t = t \log t - t = \frac{c \log n}{\log \log n} \cdot (\log c + \log \log n - \log \log \log n)
$$

\[ \geq \frac{c}{2} \log n \geq 2 \log n, \text{ for } c \geq 4
\]

Therefore, w.p. $1/n^2$, there are at least $t$ balls in Bin $i$. What is $\Pr(\text{All bins have } \leq t \text{ balls})$?

Applying Union Bound, $\Pr(\text{All bins have } \leq t \text{ balls}) \geq 1 - 1/n$
Balls and Bins

Suppose we toss $n$ balls u.a.r into $n$ bins. What is the max load of a bin with high probability?

Fact: W.p. $1 - 1/n$, the maximum load of each bin is at most $O(\log n/\log \log n)$

Fact: The max loaded bin has $(\log n/3\log \log n)$ balls with probability at least $1 - \text{const.}/n^{(1/3)}$

Let $X_i = \#\text{balls in bin }i$

\[
Pr(X_i \geq t) \geq \binom{n}{t} \frac{1}{n^t} \left(1 - \frac{1}{n}\right)^{n-t} \geq \left(\frac{n}{t}\right)^t \cdot \frac{1}{n^t} \cdot e^{-1} \geq \frac{1}{et^t}
\]

At least $1/en^{1/3}$ for $t = \log n/3\log \log n$

Let $Y_i = 1$ if bin $i$ has load $t$ or more, $= 0$ otherwise

\[Y = Y_1 + Y_2 + .. + Y_n\]

\[Pr(Y = 0) = Pr(\text{No bin has load } t \text{ or more}) \leq Pr(|Y - E[Y]| \geq E[Y])\]

Using Chebyshev, \[Pr(|Y - E[Y]| \geq E[Y]) \leq \frac{\text{Var}(Y)}{E(Y)^2}\]
**Balls and Bins**

Suppose we toss $n$ balls u.a.r into $n$ bins. What is the max load of a bin with high probability?

**Fact:** W.p. $1 - 1/n$, the maximum load of each bin is at most $O(\log n / \log \log n)$

**Fact:** The max loaded bin has $(\log n / 3 \log \log n)$ balls with probability at least $1 - \text{const.}/n^{(1/3)}$

Let $Y_i = 1$ if bin $i$ has load $t$ or more, $= 0$ otherwise

$Y = Y_1 + Y_2 + \ldots + Y_n$

$E(Y) = n^{2/3}/e$

$\Pr(Y = 0) = \Pr(\text{No bin has load } \geq t) \leq \Pr(|Y - E[Y]| \geq E[Y]) \leq \frac{\text{Var}(Y)}{E(Y)^2}$ \hspace{1cm} \text{(Chebyshev)}

$\text{Var}[Y] = \text{Var}[(Y_1 + \ldots + Y_n)^2] = \sum_i \text{Var}(Y_i) + 2 \sum_{i \neq j} (E[Y_i Y_j] - E[Y_i] E[Y_j])$

Now if $i$ is not $j$, $Y_i$ and $Y_j$ are negatively correlated, which means that $E[Y_i Y_j] < E[Y_i] E[Y_j]$

Thus,

$\text{Var}(Y) \leq \sum_{i=1}^{n} \text{Var}(Y_i) \leq n \cdot 1$

$\Pr(Y = 0) \leq \frac{\text{Var}(Y)}{E(Y)^2} \leq \frac{ne^2}{n^{4/3}} \leq \frac{e^2}{n^{1/3}}$
The Power of Two Choices

**Problem:** Given a large set $S$ of elements $x_1, .., x_n$, store them using $O(n)$ space s.t it is easy to determine whether a query item $q$ is in $S$ or not

![Table](image)

Linked list of all $x_i$ s.t $h(x_i) = 2$

**Algorithm:**
1. Pick two completely random functions $h_1 : \mathcal{U} \rightarrow \{1, \ldots, n\}$, and $h_2 : \mathcal{U} \rightarrow \{1, \ldots, n\}$
2. Create a table of size $n$, initialize it to null
3. Store $x_i$ at linked list at position $h_1(x_i)$ or $h_2(x_i)$, whichever is shorter
4. For a query $q$, look at the linked list at location $h_1(q)$ and $h_2(q)$ of table to see if $q$ is there

Equivalent to the following **Balls and Bins Problem:** Toss $n$ balls into $n$ bins. For each ball, pick two bins u.a.r and put the ball into the lighter of the two bins.

What is the worst case query time? **Answer:** $O(\log \log n)$ (proof not in this class)