RANDOMNESS

- what it is and what it isn’t
- Las Vegas and Monte Carlo
- approximate counting
- context

https://algs4.cs.princeton.edu
A brief recap: where we’ve already encountered randomness

**Percolation.** Monte Carlo simulation: open random blocked sites.

**Randomized queues.** Remove item chosen uniformly at random.
A brief recap: where we’ve already encountered randomness

Test 2: open random sites until the system percolates
Test 7: open random sites with large n
Test 12: call open(), isOpen(), and numberOfOpenSites() in random order until just before system percolates
Test 13: call open() and percolates() in random order until just before system percolates
Test 14: call open() and isFull() in random order until just before system percolates
Test 15: call all methods in random order until just before system percolates
Test 16: call all methods in random order until almost all sites are open (with inputs not prone to backwash)
Test 20: call all methods in random order until all sites are open (these inputs are prone to backwash)
A brief recap: where we’ve already encountered randomness

Tests 1-8 make **random** intermixed calls to addFirst(), addLast(), removeFirst(), removeLast(), isEmpty(), and size(), and iterator().

Test 12: check iterator() after **random** calls to addFirst(), addLast(), removeFirst(), and removeLast() with probabilities (p1, p2, p3, p4)

Tests 1-6 make **random** intermixed calls to enqueue(), dequeue(), sample(), isEmpty(), size(), and iterator().

Test 16: **check randomness** of sample() by enqueueing n items, repeatedly calling sample(), and counting the frequency of each item

Test 17: **check randomness** of dequeue() by enqueueing n items, dequeueing n items, and seeing whether each of the n! permutations is equally likely

Test 18: **check randomness** of iterator() by enqueueing n items, iterating over those n items, and seeing whether each of the n! permutations is equally likely
A brief recap: where we’ve already encountered randomness

**Quicksort** is a (Las Vegas) randomized algorithm.
Shuffling is needed for performance guarantee.
Equivalent alternative: pick a random pivot in each subarray.

---

**Hash tables.**
RANDOMNESS

- what it is and what it isn’t
- Las Vegas and Monte Carlo
- approximate counting
- context

https://algs4.cs.princeton.edu
Which of these outcomes is most likely to occur in a sequence of 6 coin flips?

A. 

B. 

C. 

D. All of the above.

E. Both B and C.
The uniform distribution

Coin flip.

\[ \Pr(C \text{ lands heads}) = \Pr(C \text{ lands tails}) = \frac{1}{2}. \]

Roll of a die.

\[ \Pr(D = 1) = \Pr(D = 2) = \cdots = \Pr(D = 6) = \frac{1}{6}. \]

\[ \Pr(D = 1) = \cdots = \Pr(D = 20) = \frac{1}{20}. \]

Notation.

\( C \) and \( D \) are random variables.

“\( C \) lands heads,” “\( D = 4 \)” or “\( D \) is even” are events with probabilities \( \Pr(C \text{ lands heads}) \), etc.

A distribution consists of all outcome–probability pairs.

[uniform distribution: all probabilities equal]
The uniform distribution

Generating uniform distributions.

- Over (small) domain of size $n$:
  place outcomes in array, return random element.

- Over large domains:
  - Bit strings of length $n$: \[ \text{size } 2^n \]
    flip $n$ coins, output sequence of outcomes ($H = 0$, $T = 1$).
  - Permutations of $n$ items: \[ \text{size } n! \]
    sample $n$ elements from \{1,2,...,n\} without replacement.
  - Spanning trees of $n$-vertex graph? \[ \text{size } \leq n^{n-2} \]

\[ \text{see precept!} \]
Flip a coin 6 times and count how often it lands heads. Which count is most likely?

A. 2
B. 3
C. 4
D. All of the above.
E. None of the above.
Concentration and the Central Limit Theorem

Normal distribution (a.k.a. Gaussian)
Normal distribution (a.k.a. Gaussian)

Mean $\mu$: peak

Standard deviation $\sigma$: spread [68% within $\pm\sigma$, 95% within $\pm2\sigma$]
Randomness: quiz 3

What is your height?

A. Less than 5’4”
B. At least 5’4” and less than 5’6”
C. At least 5’6” and less than 5’8”
D. At least 5’8” and less than 5’10”
E. At least 5’10”
Fast on average vs. fast with high probability

**Quicksort:** $C_n = \mathbb{E}[c_n] \sim 2n \ln n$. [fast on average]

**Stronger guarantee:** $\mathbb{P}[c_n \leq 3n \ln n] \geq 99\%$. [fast with high probability]

**Tail bounds toolkit.** Markov, Chebyshev, Chernoff, …
Computers can’t generate randomness (without specialized hardware).

Pseudorandom functions.

```
random — Generate pseudo–random numbers

Source code: Lib/random.py

This module implements pseudo–random number generators for various distributions.

For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random element, a function to generate a random permutation of a list in–place, and a function for random sampling without replacement.
```
**Goal.** Rearrange array so that result is a uniformly random permutation.

(all $n!$ permutations equally likely)
Goal. Rearrange array so that result is a uniformly random permutation.

all $n!$ permutations equally likely
Interview question: shuffle an array

Goal. Rearrange array so that result is a uniformly random permutation.

Challenge. Design a linear-time algorithm.
Knuth shuffle demo

- In iteration $i$, pick integer $r$ between 0 and $i$ uniformly at random.
- Swap $a[i]$ and $a[r]$. 

![Playing cards](image-url)
Knuth shuffle

- In iteration $i$, pick integer $r$ between 0 and $i$ uniformly at random.
- Swap $a[i]$ and $a[r]$.

Knuth shuffle

- In iteration $i$, pick integer $r$ between 0 and $i$ uniformly at random.
- Swap $a[i]$ and $a[r]$.

```
public class Knuth {
    public static void shuffle(Object[] a) {
        int n = a.length;
        for (int i = 0; i < n; i++) {
            int r = StdRandom.uniform(i + 1);
            exch(a, i, r);
        }
    }
}
```

http://algs4.cs.princeton.edu/11model/Knuth.java.html

common bug: between 0 and $n - 1$
correct variant: between $i$ and $n - 1$
Broken Knuth shuffle

Q. What happens if integer is chosen between 0 and $n - 1$?
A. Not uniformly random!

<table>
<thead>
<tr>
<th>permutation</th>
<th>Knuth shuffle</th>
<th>broken shuffle</th>
</tr>
</thead>
<tbody>
<tr>
<td>A B C</td>
<td>1 / 6</td>
<td>4 / 27</td>
</tr>
<tr>
<td>A C B</td>
<td>1 / 6</td>
<td>5 / 27</td>
</tr>
<tr>
<td>B A C</td>
<td>1 / 6</td>
<td>5 / 27</td>
</tr>
<tr>
<td>B C A</td>
<td>1 / 6</td>
<td>5 / 27</td>
</tr>
<tr>
<td>C A B</td>
<td>1 / 6</td>
<td>4 / 27</td>
</tr>
<tr>
<td>C B A</td>
<td>1 / 6</td>
<td>4 / 27</td>
</tr>
</tbody>
</table>

probability of each permutation when shuffling \{ A, B, C \}

$3^3 = 27$ possible outcomes (but 27 is not a multiple of 6) instead of between 0 and i
Texas hold’em poker. Software must shuffle electronic cards.

How We Learned to Cheat at Online Poker: A Study in Software Security

Industry story (online poker)

Bug 1. Random number $r$ is never 52 $\Rightarrow$ 52\textsuperscript{nd} card can’t end up in 52\textsuperscript{nd} place.

Bug 2. Shuffle not uniform (should be between 1 and $i$).

Bug 3. \texttt{random()} uses 32-bit seed $\Rightarrow$ $2^{32}$ possible shuffles.

Bug 4. Seed = milliseconds since midnight $\Rightarrow$ 86.4 million shuffles.

```
for i := 1 to 52 do begin
    r := random(51) + 1;
    swap := card[r];
    card[r] := card[i];
    card[i] := swap;
end;
```

Shuffling algorithm in FAQ at www.planetpoker.com

“\textit{The generation of random numbers is too important to be left to chance.}”
— Robert R. Coveyou
Industry story (online poker)

Best practices for shuffling (if your business depends on it).

- Use a hardware random-number generator that has passed both the FIPS 140–2 and the NIST statistical test suites.
- Continuously monitor statistical properties: hardware random-number generators are fragile and fail silently.
- Use an unbiased shuffling algorithm.

Bottom line. Shuffling a deck of cards is hard!
RANDOMNESS

- what it is and what it isn’t
- Las Vegas and Monte Carlo
- approximate counting
- context
Las Vegas algorithms

- Guaranteed to be correct.
- Running time depends on outcomes of random coin flips.

Ex. Quicksort, quickselect.

\[
\begin{array}{ccc}
\leq p & \bar{} & \geq p \\
\uparrow & & \uparrow \\
lo & j & hi
\end{array}
\]
Monte Carlo algorithms

Monte Carlo algorithm.
- Not guaranteed to be correct.
- Running time is deterministic.
  [doesn’t depend on coin flips]

Amplification. If \( \mathbb{P}[A \text{ is correct}] = 1\% \), repeat 500 times.

Then, \( \mathbb{P}[A_1, A_2, \ldots, A_{500} \text{ are all incorrect}] \leq \left( \frac{99}{100} \right)^{500} < 1\% \)

\[ \text{independence} \]
Karger’s global mincut algorithm

**Goal.** Find cut in undirected graph with fewest edges (for any source and sink).

**Idea.** Pick a random cut.

**Uniformly?** Since there are $2^V - 1$ cuts, may succeed with only $\sim \frac{1}{2^V}$ probability.
Karger’s global mincut algorithm

Algorithm.

- Assign a random weight (uniform between 0 and 1) to each edge $e$.
- Run Kruskal’s MST algorithm until 2 connected components left.
- 2 connected components defines the cut.

**Probability of finding a mincut:** $\geq \frac{1}{V^2}$. [no mincut edges in each connected component]

Run algorithm many times and return best cut.

**Remark 1.** Finds global mincut in $\Theta(EV^2 \log E)$ time — better than $\Theta(V)$ runs of Ford–Fulkerson!

**Remark 2.** With clever idea, improved to $\Theta(V^2 \log^3 V)$ time (still randomized).
Smallest # of repetitions of Karger’s algorithm to get correct answer with 99% probability?

A. $\Theta(1)$

B. $\Theta(V)$

C. $\Theta(V^2)$

D. $\Theta(V^3)$

E. None of the above.
RANDOMNESS

- what it is and what it isn’t
- Las Vegas and Monte Carlo
- approximate counting
- context
Packet counting
Fix $n \in \mathbb{N}$. How many bits must a counter have to count from 0 to $n - 1$?

A. $\log_2 n$

B. $\lfloor \log_2 n \rfloor$  \hspace{1cm} \text{round down}

C. $\lceil \log_2 n \rceil$  \hspace{1cm} \text{round up}

D. $\lfloor \log_2 n \rfloor + 1$

E. $n$
Approximate counting

**Goal.** Count with less memory: from $\sim \log_2 n$ to $\Theta(\log \log n)$.

**Why bother?**
Database with 1 billion entries: $\log_2 10^9 \approx 30$ bits, but $\log_2 \log_2 10^9 \approx 5$ bits.
Factor–6 improvement matters a lot.

$\log_2 \log_2 10^9 \approx \frac{30}{\log_2 10} \approx 5$
Approximate counting
Approximate counting

public class ApproximateCounter() {
    private byte c;

    public void increment() {
        if (StdRandom.uniformInt(1 << c) == 0)
            c++;
    }

    public int count() {
        return (1 << c) - 1;
    }
}

Returns $2^c$

Value of counter around $k$ after $n = 2^0 + 2^1 + \cdots + 2^{k-1} = 2^k - 1$ packets.

Memory requirement: $\sim \log_2 k \sim \log_2 \log_2 n$. 
Which of the following distributions makes $b = 1$ with probability $1/8$?

A. Flip 3 coins, set $b = 1$ if two flips land heads.

B. Flip 3 coins, set $b = 1$ if all land heads.

C. Flip 3 coins, set $b = 1$ if all land tails.

D. Both A and B.

E. Both B and C.
Approximate counting: probabilistic analysis

**Proposition.** The value $c_n$ of the counter after $n$ packets satisfies $\mathbb{E}[2^{c_n}] = n + 1$.

**Pf.** [by induction on $n$]

Base case: initially, $\mathbb{E}[2^{c_0}] = 2^0 = 0 + 1$.

Define $P_{n,k} = \mathbb{P}[c_n = k]$. They satisfy the recurrence $P_{0,0} = P_{0,1} = 1$ and

$$P_{n+1,k} = \frac{1}{2^{k-1}} \times P_{n,k-1} + \left(1 - \frac{1}{2^k}\right) \times P_{n,k}$$

- **counter was $k-1$**
- **increased**

- **counter was $k$**
- **didn’t increase**

*analysis beyond scope of this course*
**Proposition.** The value $c_n$ of the counter after $n$ packets satisfies $\mathbb{E}[2^{c_n}] = n + 1$.

**Pf.** [by induction on $n$]

Decompose $\mathbb{E}[2^{c_n}]$ and rearrange:

\[
\mathbb{E}[2^{c_n+1}] = \sum_{k=0}^{n+1} 2^k \times P_{n+1,k}
\]
Approximate counting: probabilistic analysis

**Proposition.** The value $c_n$ of the counter after $n$ packets satisfies $\mathbb{E}[2^{c_n}] = n + 1$.

**Pf.** [by induction on $n$]

\[
P_{n+1,k} = \frac{1}{2^{k-1}} \times P_{n,k-1} + \left(1 - \frac{1}{2^k}\right) \times P_{n,k}
\]

Decompose $\mathbb{E}[2^{c_{n+1}}]$ and rearrange:

\[
\mathbb{E}[2^{c_{n+1}}] = \sum_{k=0}^{n+1} 2^k \times \left(\frac{1}{2^{k-1}} \times P_{n,k-1} + \left(1 - \frac{1}{2^k}\right) \times P_{n,k}\right)
\]

*analysis beyond scope of this course*
Proposition. The value $c_n$ of the counter after $n$ packets satisfies $\mathbb{E}[2^{c_n}] = n + 1$.

Pf. [by induction on $n$]

Decompose and rearrange:

$$P_{n+1,k} = \frac{1}{2^{k-1}} \times P_{n,k-1} + \left(1 - \frac{1}{2^k}\right) \times P_{n,k}$$

Counter was $k$ - 1

Counter was $k$

Increased

Didn't increase

Decompose $\mathbb{E}[2^{c_{n+1}}]$ and rearrange:

$$\mathbb{E}[2^{c_{n+1}}] = \sum_{k=0}^{n+1} 2^k \times \left(\frac{1}{2^{k-1}} \times P_{n,k-1} + \left(1 - \frac{1}{2^k}\right) \times P_{n,k}\right)$$

Analysis beyond scope of this course
**Proposition.** The value $c_n$ of the counter after $n$ packets satisfies $\mathbb{E}[2^{c_n}] = n + 1$.

**Pf.** [by induction on $n$]

\[
P_{n+1,k} = \frac{1}{2^{k-1}} \times P_{n,k-1} + \left(1 - \frac{1}{2^k}\right) \times P_{n,k}
\]

Decompose $\mathbb{E}[2^{c_n+1}]$ and rearrange:

\[
\mathbb{E}[2^{c_n+1}] = \sum_{k=0}^{n+1} 2^k \times \left(\frac{1}{2^{k-1}} \times P_{n,k-1} + \left(1 - \frac{1}{2^k}\right) \times P_{n,k}\right)
\]

\[
= 2 \sum_{k=0}^{n+1} P_{n,k-1}
\]
Proposition. The value $c_n$ of the counter after $n$ packets satisfies $\mathbb{E}[2^{c_n}] = n + 1$.

Pf. [by induction on $n$]

Decompose and rearrange:

$$P_{n+1,k} = \frac{1}{2^{k-1}} \times P_{n,k-1} + \left(1 - \frac{1}{2^k}\right) \times P_{n,k}$$

Increased
didn’t increase

$$\mathbb{E}[2^{c_{n+1}}] = \sum_{k=0}^{n+1} 2^k \times \left(\frac{1}{2^{k-1}} \times P_{n,k-1} + \left(1 - \frac{1}{2^k}\right) \times P_{n,k}\right)$$

$$= 2 \sum_{k=0}^{n+1} P_{n,k-1} + \sum_{k=0}^{n+1} 2^k \times P_{n,k}$$

Analysis beyond scope of this course
Proposition. The value $c_n$ of the counter after $n$ packets satisfies $\mathbb{E}[2^{c_n}] = n + 1$.

Pf. [by induction on $n$]

Decompose and rearrange:

$$P_{n+1,k} = \frac{1}{2^{k-1}} \times P_{n,k-1} + \left(1 - \frac{1}{2^k}\right) \times P_{n,k}$$

- Counter was $k - 1$ and increased
- Counter was $k$ and didn’t increase

Decompose $\mathbb{E}[2^{c_{n+1}}]$ and rearrange:

$$\mathbb{E}[2^{c_{n+1}}] = \sum_{k=0}^{n+1} 2^k \times \left(\frac{1}{2^{k-1}} \times P_{n,k-1} + \left(1 - \frac{1}{2^k}\right) \times P_{n,k}\right)$$

$$= 2 \sum_{k=0}^{n+1} P_{n,k-1} + \sum_{k=0}^{n+1} 2^k \times P_{n,k} - \sum_{k=0}^{n+1} P_{n,k}$$
**Proposition.** The value $c_n$ of the counter after $n$ packets satisfies $\mathbb{E}[2^{c_n}] = n + 1$.

**Pf.** [by induction on $n$]

Decompose and rearrange:

\[
P_{n+1,k} = \frac{1}{2^{k-1}} \times P_{n,k-1} + \left(1 - \frac{1}{2^k}\right) \times P_{n,k}
\]

Approximate counting: probabilistic analysis

Decompose $\mathbb{E}[2^{c_{n+1}}]$ and rearrange:

\[
\mathbb{E}[2^{c_{n+1}}] = \sum_{k=0}^{n+1} 2^k \times \left(\frac{1}{2^{k-1}} \times P_{n,k-1} + \left(1 - \frac{1}{2^k}\right) \times P_{n,k}\right)
\]

\[
= 2 \sum_{k=0}^{n+1} P_{n,k-1} + \sum_{k=0}^{n+1} 2^k \times P_{n,k} - \sum_{k=0}^{n+1} P_{n,k}
\]

\[
= 2 + \mathbb{E}[2^{c_n}] - 1
\]

Analysis beyond scope of this course
Proposition. The value $c_n$ of the counter after $n$ packets satisfies $\mathbb{E}[2^{c_n}] = n + 1$.

Pf. [by induction on $n$]

Decompose and rearrange:

\[
P_{n+1, k} = \frac{1}{2^{k-1}} \times P_{n, k-1} + \left(1 - \frac{1}{2^k}\right) \times P_{n, k}
\]

Counter was $k - 1$

Counter was $k$

Increase

didn't increase

Approximate counting: probabilistic analysis

\[
\mathbb{E}[2^{c_{n+1}}] = \sum_{k=0}^{n+1} 2^k \times \left(\frac{1}{2^{k-1}} \times P_{n, k-1} + \left(1 - \frac{1}{2^k}\right) \times P_{n, k}\right)
\]

\[
= 2 \sum_{k=0}^{n+1} P_{n, k-1} + \sum_{k=0}^{n+1} 2^k \times P_{n, k} - \sum_{k=0}^{n+1} P_{n, k}
\]

= $2 + \mathbb{E}[2^{c_n}] - 1$

= $(n + 1) + 1$

Analysis beyond scope of this course
RANDOMNESS

- what it is and what it isn’t
- Las Vegas and Monte Carlo
- approximate counting
- context
Beyond this course

- Approximation algorithms  [intractability: stay tuned!]
- Cryptography  [average-case hardness]
- Complexity theory: $P \equiv BPP$  [derandomization]
- Mathematics: the Probabilistic Method
  E.g., graph with $E$ edges has a cut with $E/2$ edges.  [approximate maxcut]
  To prove that there exists an object with property $T$:
  - sample a random object;
  - show that $\Pr[T$ is satisfied] > 0.
- Quantum computation

**COS 495.**  Probability in Computer Science.

**ORF 309.**  Probability and Stochastic Systems.
<table>
<thead>
<tr>
<th>image</th>
<th>source</th>
<th>license</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quarter</td>
<td>Adobe Stock</td>
<td>Education License</td>
</tr>
<tr>
<td>6-sided dice</td>
<td>Adobe Stock</td>
<td>Education License</td>
</tr>
<tr>
<td>20-sided die</td>
<td>Adobe Stock</td>
<td>Education License</td>
</tr>
<tr>
<td>Lava lamps</td>
<td>Fast Company</td>
<td></td>
</tr>
<tr>
<td>Coin Toss</td>
<td>clipground.com</td>
<td></td>
</tr>
<tr>
<td>IDQ Quantum Key Factory</td>
<td>idquantique.com</td>
<td>Education License</td>
</tr>
<tr>
<td>SG100</td>
<td>protego.bytehost16.com</td>
<td>Education License</td>
</tr>
<tr>
<td>Las Vegas</td>
<td>Adobe Stock</td>
<td>Education License</td>
</tr>
<tr>
<td>Monte Carlo</td>
<td>Adobe Stock</td>
<td>Education License</td>
</tr>
<tr>
<td>Router</td>
<td>Adobe Stock</td>
<td>Education License</td>
</tr>
<tr>
<td>Random number generator</td>
<td>XKCD</td>
<td>CC BY-NC 2.5</td>
</tr>
</tbody>
</table>
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
    // guaranteed to be random.
}