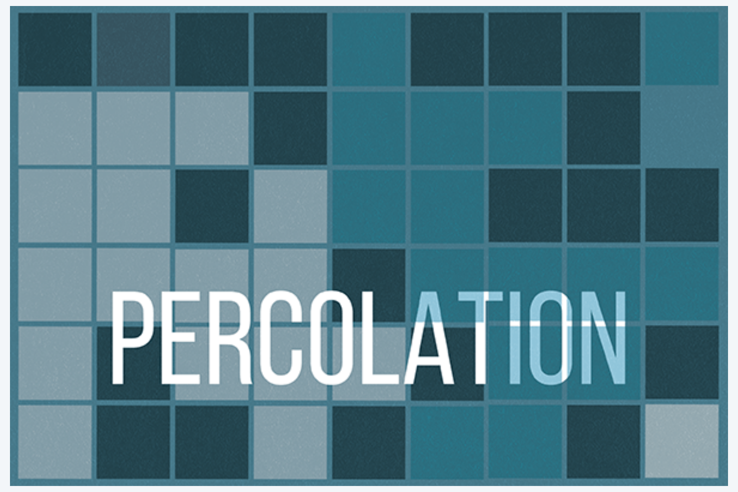
https://algs4.cs.princeton.edu

Randomness

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

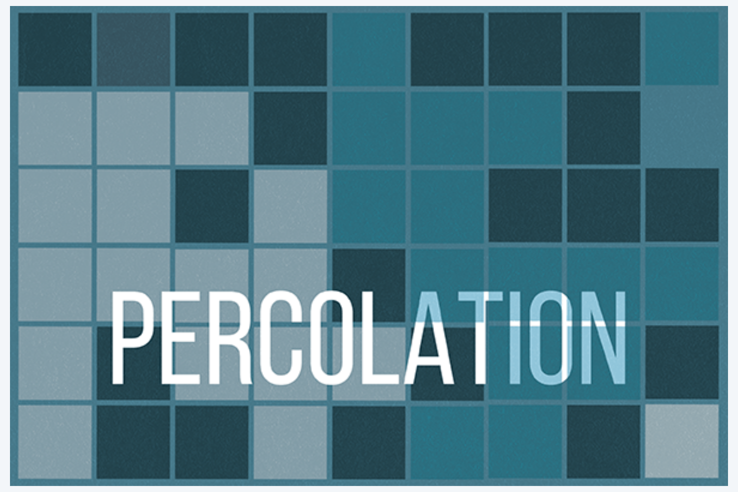
A brief recap: where we've already encountered randomness

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

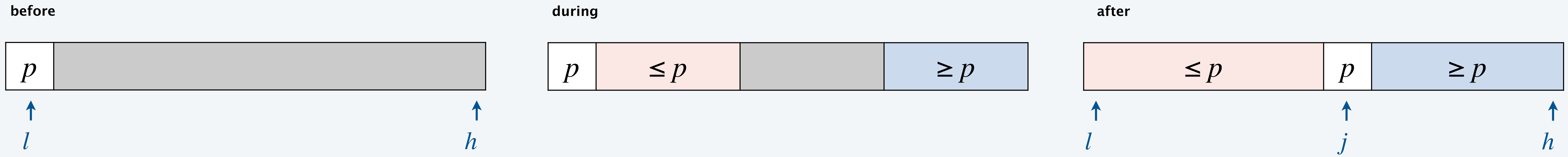

```
Tests 1-8 make random intermixed calls to addFirst(), addLast(),
removeFirst(), removeLast(), isEmpty(), and size(), and iterator().
Test 12: check iterator() after random calls to addFirst(), addLast(),
         removeFirst(), and removeLast() with probabilities (p1, p2, p3, p4)
Tests 1-6 make random 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

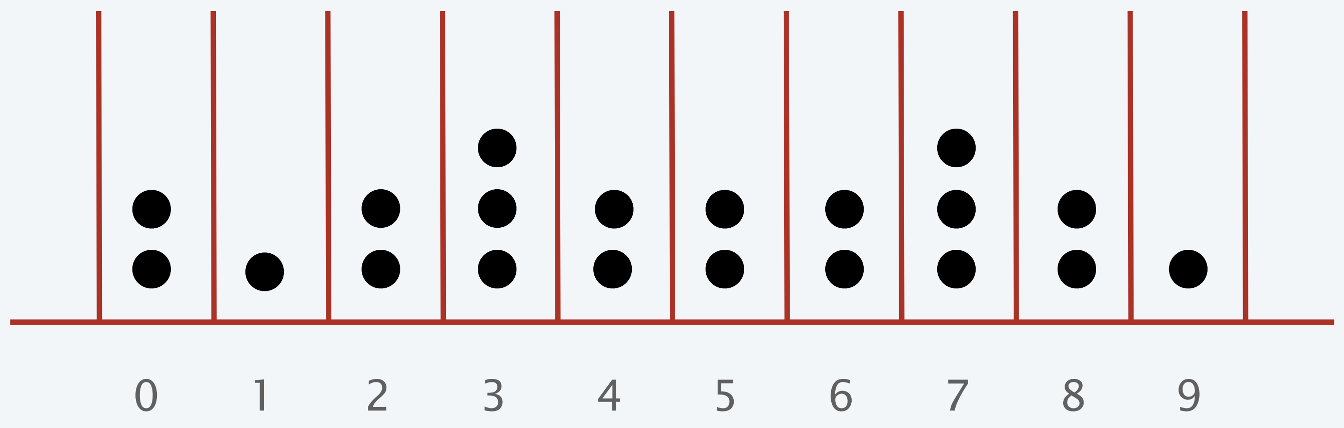
Quicksort is a (Las Vegas) randomized algorithm.

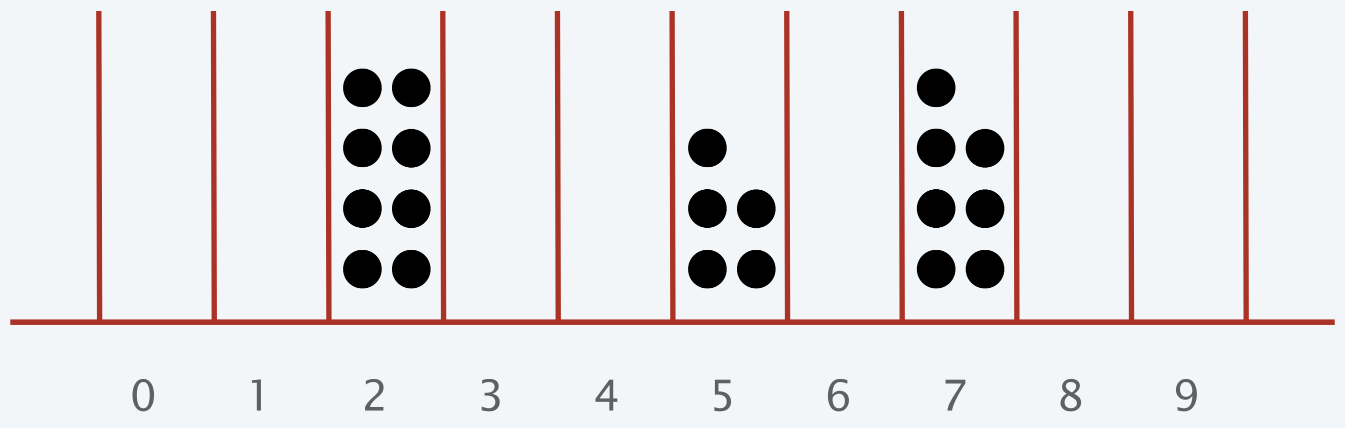
Shuffling is needed for performance guarantee.

Equivalent alternative: pick a random pivot in each subarray.



Hash tables.





https://algs4.cs.princeton.edu

Randomness

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

Randomness: quiz 1

Which of these outcomes is most likely to occur in a sequence of 6 coin flips?

A. 

B. 

C. 

D. All of the above.

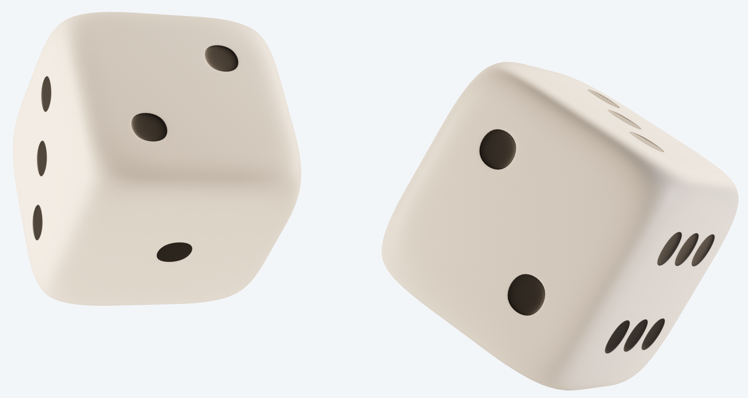
E. Both B and C.

The uniform distribution

Coin flip.

$\mathbb{P}[C \text{ lands heads}] = \mathbb{P}[C \text{ lands tails}] = \dfrac{1}{2}.$

Roll of a die.



$\mathbb{P}[D = 1] = \mathbb{P}[D = 2] = \cdots = \mathbb{P}[D = 6] = \dfrac{1}{6}.$

$\mathbb{P}[D = 1] = \cdots = \mathbb{P}[D = 20] = \dfrac{1}{20}.$

Notation.

C and D are **random variables**.

"C lands heads," "$D = 4$" or "D is even" are **events** with probabilities $\mathbb{P}[C \text{ lands heads}]$, 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.

1	2	3	4	5	6

H	T

- Over large domains:
  - Bit strings of length n: [size 2^n]

    flip n coins, output sequence of outcomes ($H = 0, T = 1$).
  - Permutations of n items: [size $n!$]

    sample n elements from $\{1, 2, \ldots, n\}$ without replacement.
  - Spanning trees of n-vertex graph? [size $\leq n^{n-2}$]

Randomness: quiz 2

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.

Pseudorandomness

Computers can’t generate randomness (without specialized hardware).

Pseudorandom functions.

random — Generate pseudo-random numbers

Source code: Lib/random.py

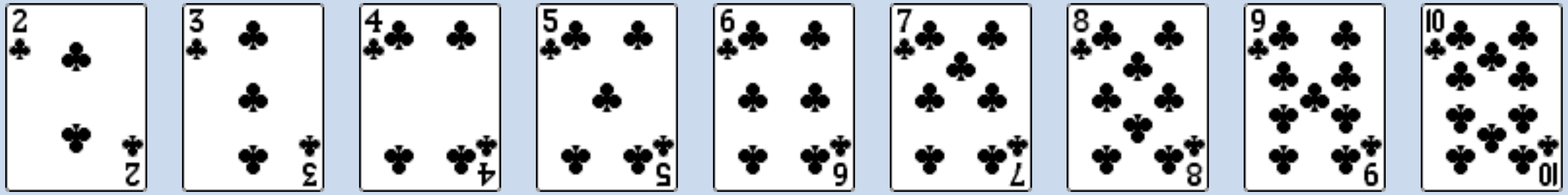
This module implements pseudo-random number generators for various distributions.

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.

Interview question: shuffle an array

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

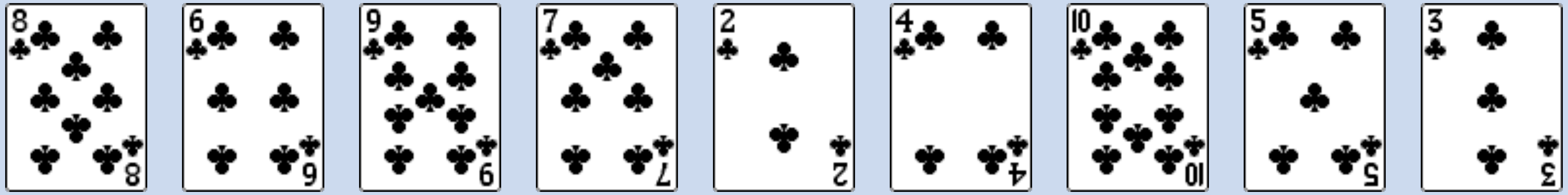
all $n!$ permutations equally likely



Interview question: shuffle an array

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

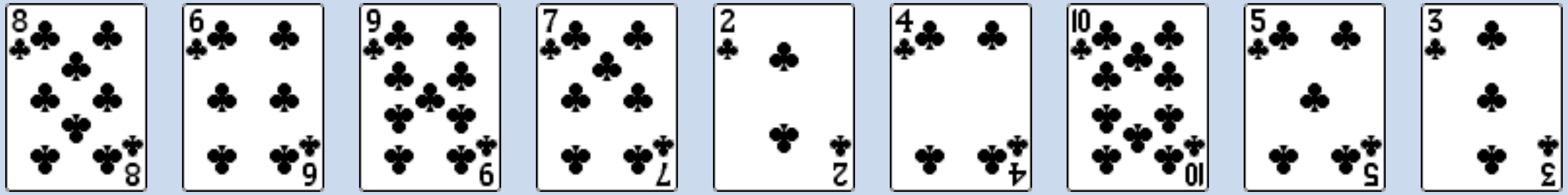
all n! permutations equally likely



Interview question: shuffle an array

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

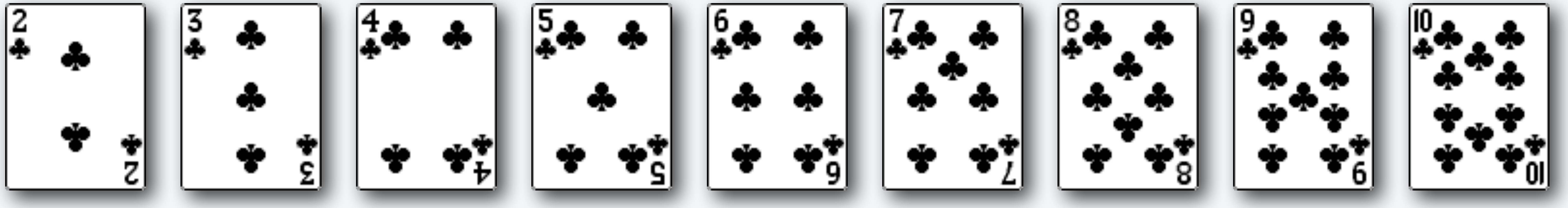
all n! permutations equally likely



Challenge. Design a linear-time algorithm.

Knuth shuffle

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



Proposition. [Fisher–Yates 1938] Knuth shuffling algorithm produces a uniformly random permutation of the input array in linear time.

assuming integers uniformly at random

Knuth shuffle

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

common bug: between 0 and n – 1
correct variant: between i and n – 1

```
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);
       }
   }
}
```

between 0 and i

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

Broken Knuth shuffle

Q. What happens if integer is chosen between 0 and $n - 1$?

A. Not uniformly random!

instead of between 0 and i

permutation	Knuth shuffle	broken shuffle
A B C	1 / 6	4 / 27
A C B	1 / 6	5 / 27
B A C	1 / 6	5 / 27
B C A	1 / 6	5 / 27
C A B	1 / 6	4 / 27
C B A	1 / 6	4 / 27

$3^3 = 27$ possible outcomes (but 27 is not a multiple of 6)

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

Industry story (online poker)

Texas hold'em poker. Software must shuffle electronic cards.

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

https://www.developer.com/tech/article.php/616221/How-We-Learned-to-Cheat-at-Online-Poker-A-Study-in-Software-Security.htm

Industry story (online poker)

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

between 1 and 51

Shuffling algorithm in FAQ at www.planetpoker.com

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

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

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

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

" *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.

RANDOM.ORG

Bottom line. Shuffling a deck of cards is hard!

https://algs4.cs.princeton.edu

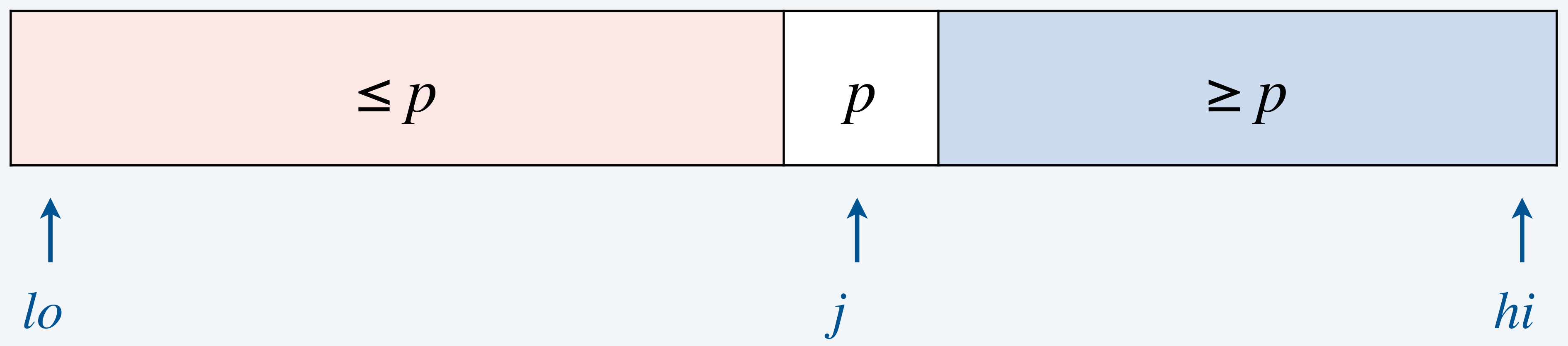
Randomness

- *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.



Monte Carlo algorithms

Monte Carlo algorithm.

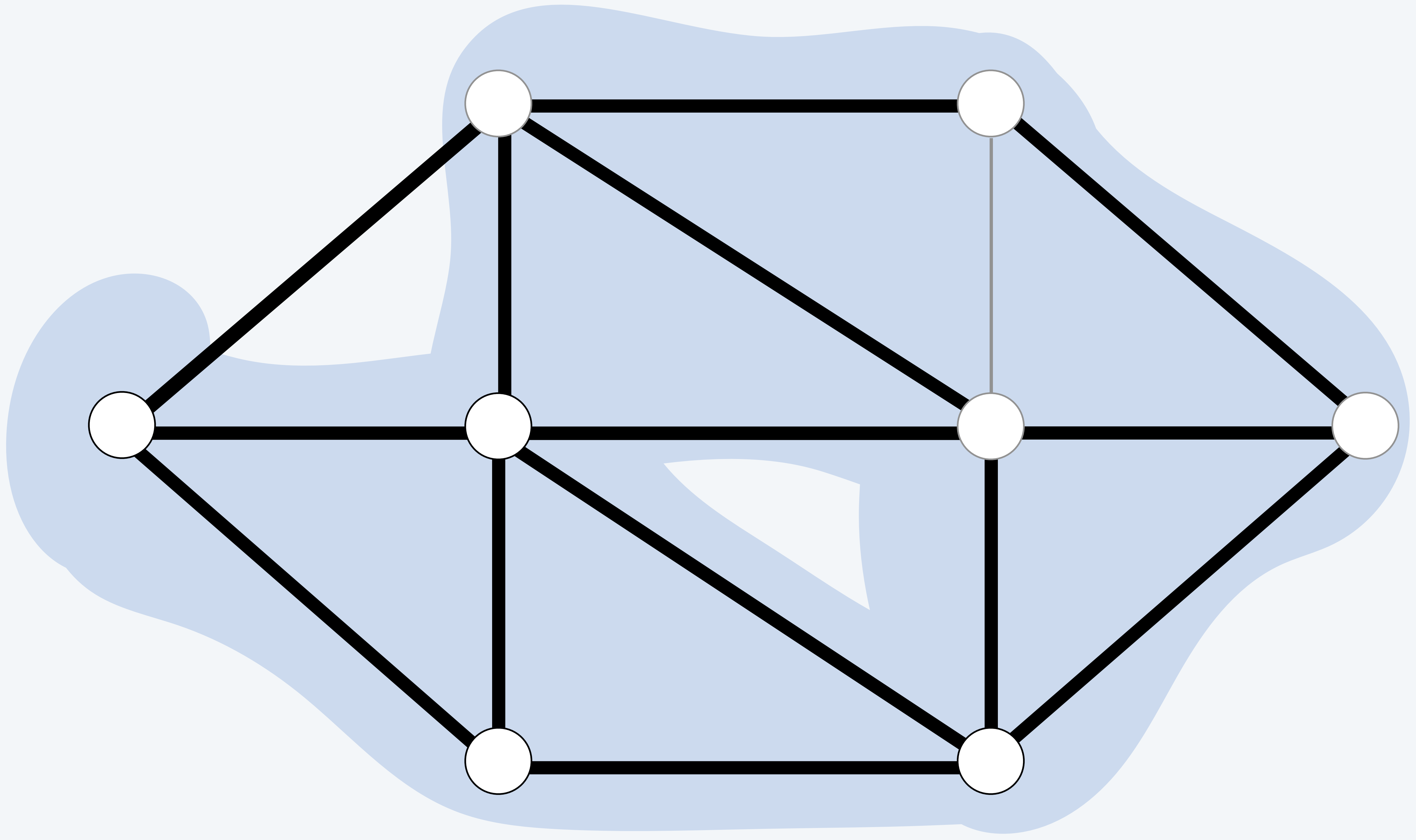
- Not guaranteed to be correct.
- Running time is deterministic.
  [doesn't depend on coin flips]

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

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

Global mincut problem

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 $\sim \dfrac{1}{2^V}$ probability.

Global mincut problem

Example. Bad graph for the "pick a uniformly random cut" algorithm.

suppose there are V / 2 vertices on each side

Problem. There is only 1 mincut, but $2^V - 1$ total cuts, we need to be lucky to find it.

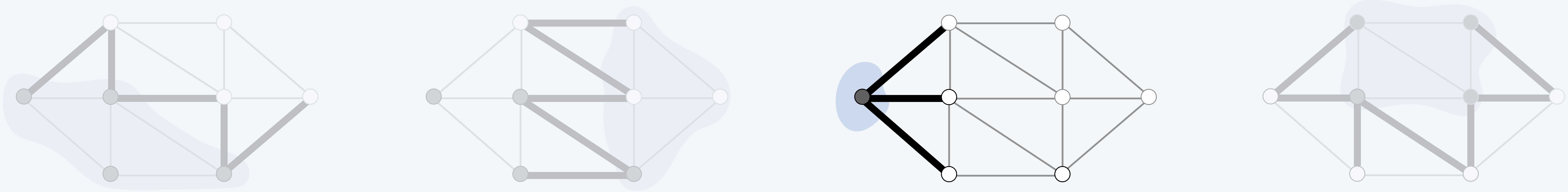
Karger's global mincut algorithm

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: $\geq \dfrac{1}{V^2}$. [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. $\Theta(V^3)$

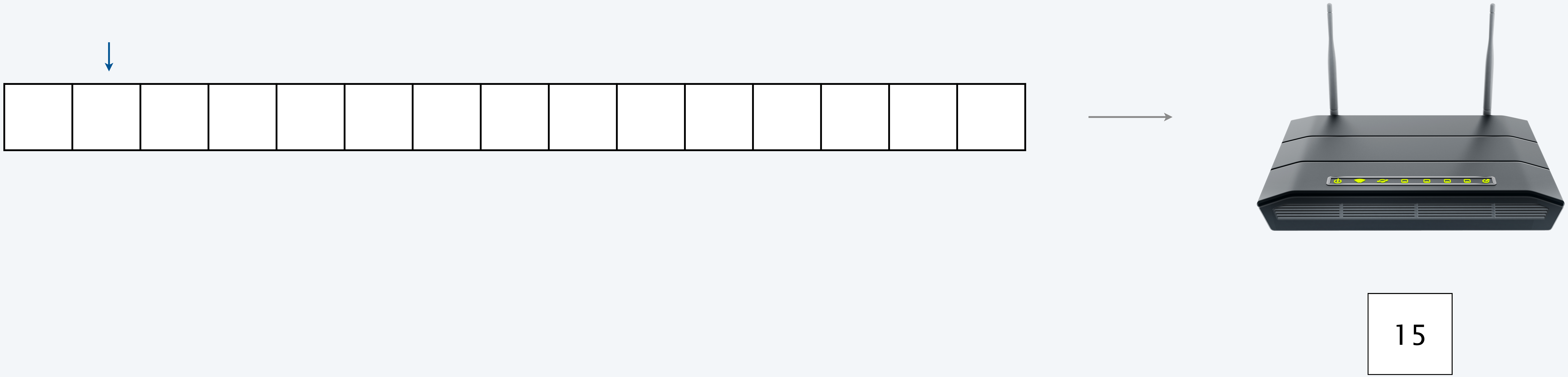
E. None of the above.

https://algs4.cs.princeton.edu

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. $\lfloor \log_2 n \rfloor$ $\longleftarrow$ *round down*

C. $\lceil \log_2 n \rceil$ $\longleftarrow$ *round up*

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

E. n

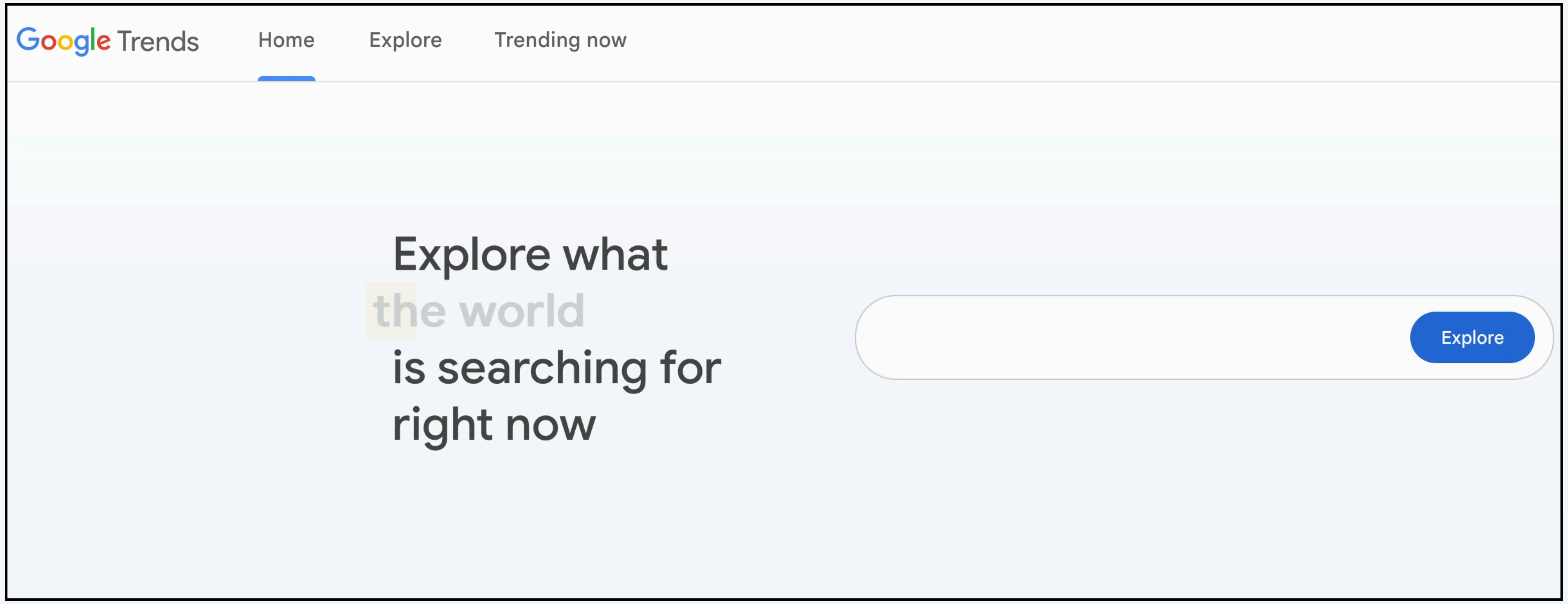
Approximate counting

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

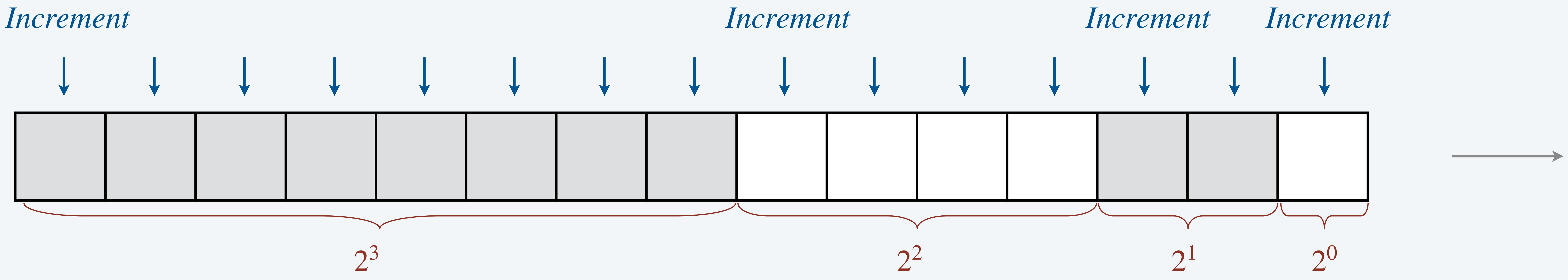
Why bother?

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.



Approximate counting



4

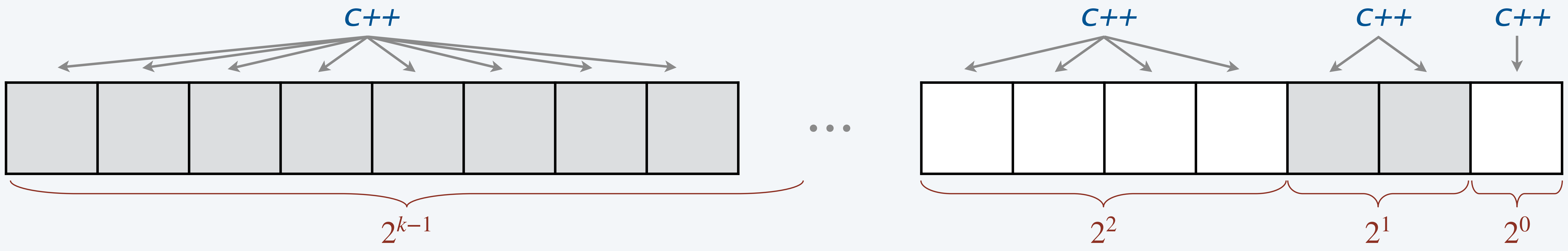
Approximate counting

```
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: $\sim \log_2 k \sim \log_2 \log_2 n$.

Approximate counting: probabilistic analysis

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

Pf. [by induction on n]

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

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

$$P_{n+1,k} = \underset{\substack{\uparrow\\ \textit{increased}}}{\frac{1}{2^{k-1}}} \times \overset{\substack{\textit{counter was } k-1\\ \downarrow}}{P_{n,k-1}} + \left(1 \underset{\substack{\uparrow\\ \textit{didn't increase}}}{-} \frac{1}{2^k}\right) \times \overset{\substack{\textit{counter was } k\\ \downarrow}}{P_{n,k}}$$

analysis beyond
scope of this course

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]

$$P_{n+1,k} = \underbrace{\frac{1}{2^{k-1}}}_{\text{increased}} \times \overbrace{P_{n,k-1}}^{\text{counter was } k-1} + \underbrace{\left(1 - \frac{1}{2^k}\right)}_{\text{didn't increase}} \times \overbrace{P_{n,k}}^{\text{counter was } k}$$

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

$$\mathbb{E}\,[2^{c_{n+1}}] = \sum_{k=0}^{n+1} 2^k \times \boxed{P_{n+1,k}}$$

analysis beyond scope of this course

Approximate counting: probabilistic analysis

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

Pf. [by induction on n]

$$P_{n+1,k} = \underset{\textit{increased}}{\underbrace{\frac{1}{2^{k-1}}}} \times \overset{\textit{counter was } k-1}{P_{n,k-1}} + \left(1 - \underset{\textit{didn't increase}}{\underbrace{\frac{1}{2^k}}}\right) \times \overset{\textit{counter was } k}{P_{n,k}}$$

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

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

analysis beyond
scope of this course

Approximate counting: probabilistic analysis

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

Pf. [by induction on n]

$$P_{n+1,k} = \underbrace{\frac{1}{2^{k-1}}}_{\text{increased}} \times \overbrace{P_{n,k-1}}^{\text{counter was } k-1} + \underbrace{\left(1 - \frac{1}{2^k}\right)}_{\text{didn't increase}} \times \overbrace{P_{n,k}}^{\text{counter was } k}$$

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

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

analysis beyond scope of this course

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]

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

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

increased *didn't increase*

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

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

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

analysis beyond scope of this course

Approximate counting: probabilistic analysis

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

Pf. [by induction on n]

$$P_{n+1,k} = \underset{\text{increased}}{\underset{\uparrow}{\frac{1}{2^{k-1}}}} \times \overset{\text{counter was } k-1}{\overset{\downarrow}{P_{n,k-1}}} + \underset{\text{didn't increase}}{\underset{\uparrow}{\left(1 - \frac{1}{2^k}\right)}} \times \overset{\text{counter was } k}{\overset{\downarrow}{P_{n,k}}}$$

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

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

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

analysis beyond scope of this course

Approximate counting: probabilistic analysis

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

Pf. [by induction on n]

$$P_{n+1,k} = \underbrace{\frac{1}{2^{k-1}}}_{\text{increased}} \times \overbrace{P_{n,k-1}}^{\text{counter was } k-1} + \underbrace{\left(1 - \frac{1}{2^k}\right)}_{\text{didn't increase}} \times \overbrace{P_{n,k}}^{\text{counter was } k}$$

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

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

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

analysis beyond scope of this course

Approximate counting: probabilistic analysis

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

Pf. [by induction on n]

$$P_{n+1,k} = \underbrace{\frac{1}{2^{k-1}}}_{\text{increased}} \times \overbrace{P_{n,k-1}}^{\text{counter was } k-1} + \underbrace{\left(1 - \frac{1}{2^k}\right)}_{\text{didn't increase}} \times \overbrace{P_{n,k}}^{\text{counter was } k}$$

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

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

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

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

analysis beyond scope of this course

Approximate counting: probabilistic analysis

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

Pf. [by induction on n]

$$P_{n+1,k} = \underset{\textit{increased}}{\frac{1}{2^{k-1}}} \times \overset{\textit{counter was } k-1}{P_{n,k-1}} + \underset{\textit{didn't increase}}{\left(1 - \frac{1}{2^k}\right)} \times \overset{\textit{counter was } k}{P_{n,k}}$$

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

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

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

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

inductive hypothesis

$$= \boxed{(n+1)} + 1$$

analysis beyond scope of this course

https://algs4.cs.princeton.edu

Randomness

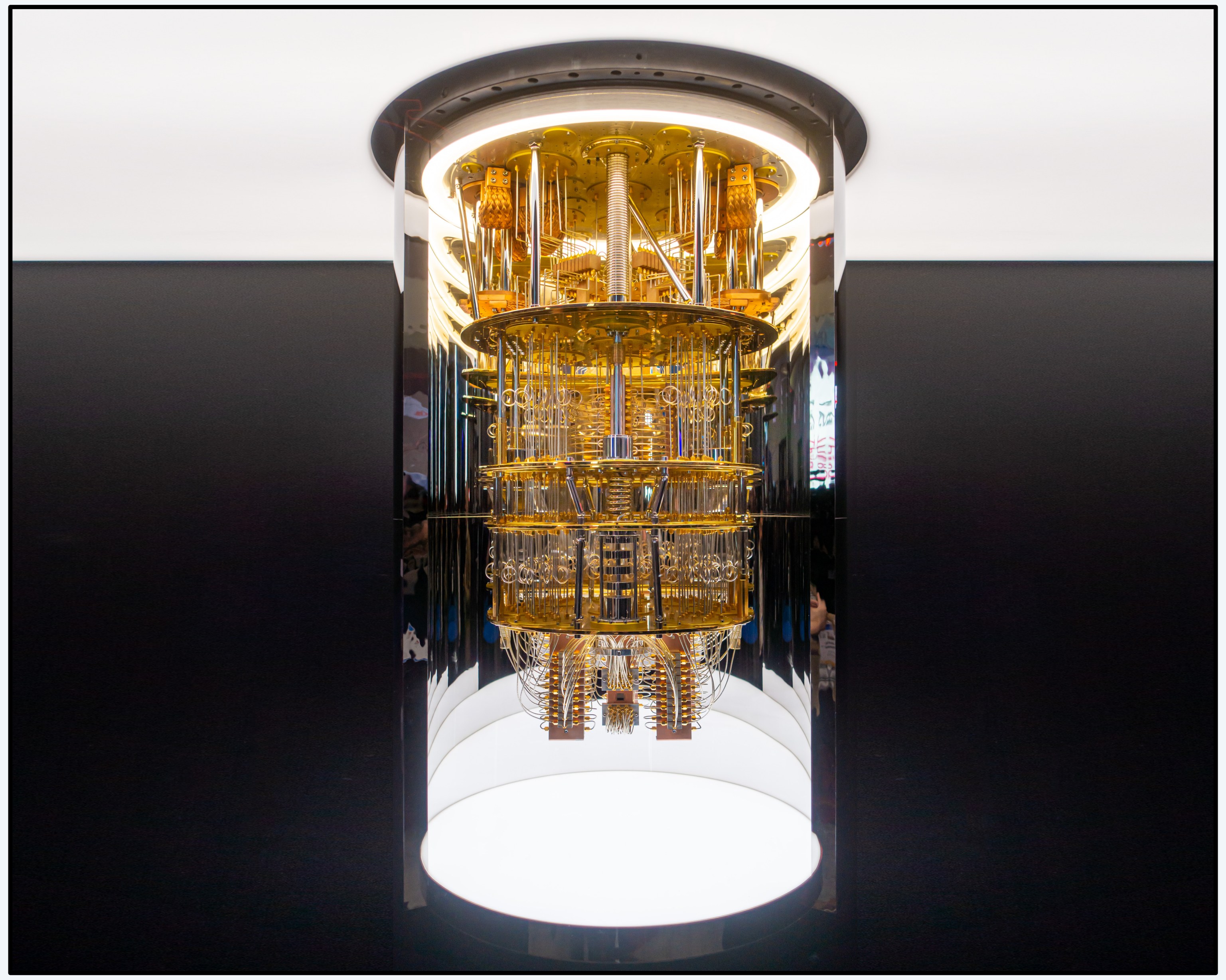
- *what it is and what it isn't*
- *Las Vegas and Monte Carlo*
- *approximate counting*
- *context*

Beyond this course

- Approximation algorithms [intractability: stay tuned!]
- Cryptography [average-case hardness]
- Complexity theory: P $\stackrel{?}{=}$ 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 $\mathbb{P}[T \text{ is satisfied}] > 0$.
- Quantum computation



IBM Quantum System One

ORF 309. Probability and Stochastic Systems.

Credits

image	source	license
Quarter	Adobe Stock	Education License
6-sided dice	Adobe Stock	Education License
20-sided die	Adobe Stock	Education License
Lava lamps	Fast Company	
Coin Toss	clipground.com	CC BY 4.0
IDQ Quantum Key Factory	idquantique.com	
SG100	protego.bytehost16.com	
Las Vegas	Adobe Stock	Education License
Monte Carlo	Adobe Stock	Education License
Router	Adobe Stock	Education License
Random number generator	XKCD	CC BY-NC 2.5

```
int getRandomNumber()
{
    return 4;     // chosen by fair dice roll.
                  // guaranteed to be random.
}
```

https://xkcd.com/221/