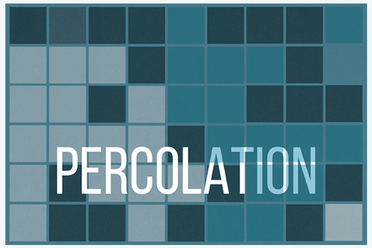

https://algs4.cs.princeton.edu

# RANDOMNESS

- *what it is and what it isn't*
- *Las Vegas and Monte Carlo*
- *rejection sampling*
- *Karger's algorithm*

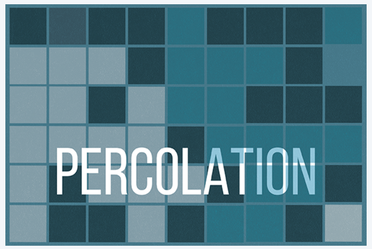
# A brief recap: where we've already encountered randomness

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

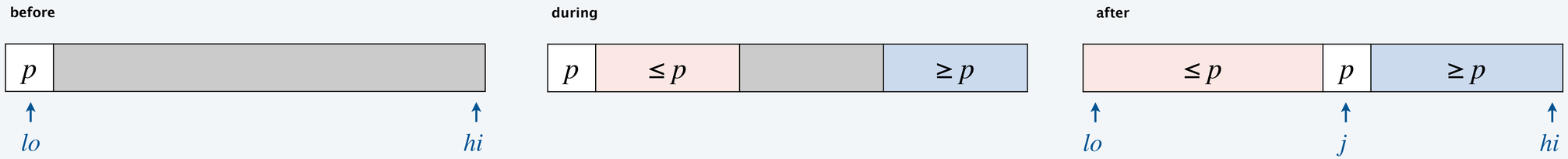

```
Tests 1-8 make random intermixed calls to addFirst(), addLast(),
removeFirst(), removeLast(), isEmpty(), and size(), and iterator().
Test 12: check iterator() after random calls to addFirst(), addLast(),
         removeFirst(), and removeLast() with probabilities (p1, p2, p3, p4)
Tests 1-6 make random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

Shuffling is needed for performance guarantee.




Hash tables.


https://algs4.cs.princeton.edu

# RANDOMNESS

- *what it is and what it isn't*
- *Las Vegas and Monte Carlo*
- *rejection sampling*
- *Karger's algorithm*

# 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.

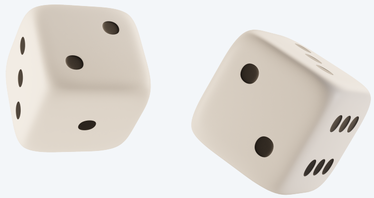
**E.** Both B and C.

# The uniform distribution

Coin flip.


$\mathbb{P}[C \text{ lands heads}] = \mathbb{P}[C \text{ lands tails}] = \frac{1}{2}$. ← *uniform over 2 outcomes*

Roll of a die.



$\mathbb{P}[D \text{ rolls } 1] = \mathbb{P}[D \text{ rolls } 2] = \cdots = \mathbb{P}[D \text{ rolls } 6] = \frac{1}{6}$. ← *uniform over 6 outcomes*

Independent coin flips.


$\mathbb{P}[C_1 \text{ heads}, C_2 \text{ tails}, \ldots C_k \text{ heads}] = \frac{1}{2} \times \frac{1}{2} \cdots \times \frac{1}{2} = \frac{1}{2^k}$. ← *uniform over $2^k$ outcomes*

Terminology and notation.

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

**Distribution**: all outcome–probability pairs.

[uniform distribution: all probabilities equal]

| outcome | probability |
|---|---|
| heads | 1/2 |
| tails | 1/2 |

**distribution of unbiased coin**

| outcome | probability |
|---|---|
| 1 | 1/6 |
| 2 | 1/6 |
| 3 | 1/6 |
| 4 | 1/6 |
| 5 | 1/6 |
| 6 | 1/6 |

**distribution of 6-sided die**

# Pseudorandomness

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



Pseudorandom functions.

**Class Random**

java.lang.Object
    java.util.Random

**All Implemented Interfaces:**

`Serializable`

**Direct Known Subclasses:**

`SecureRandom`, `ThreadLocalRandom`

# Randomness: quiz 2

**Flip a coin 6 times and count how often it lands heads. Which count is most likely?**

A. 2

B. 3

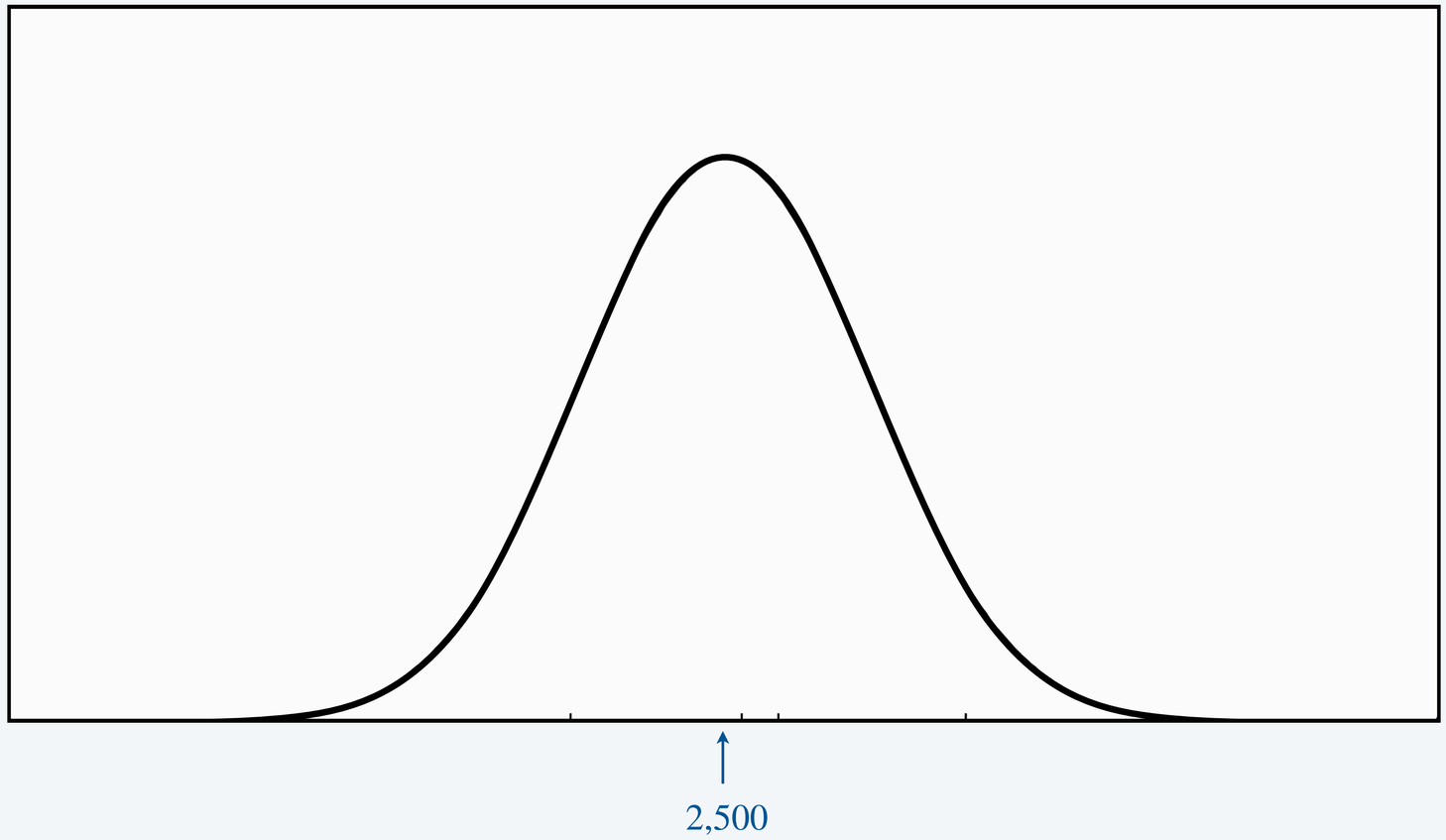
C. 4

D. All of the above.

E. None of the above.

# Binomial distribution

Experiment. Flip 5000 coins, count # of heads.




https://algs4.cs.princeton.edu

# RANDOMNESS

- *what it is and what it isn't*
- *Las Vegas and Monte Carlo*
- *rejection sampling*
- *Karger's algorithm*

# A toy problem

Treasure hunt. Length-$n$ array with 50% treasures, 50% duds.

Deterministic algorithms.

- scan the array left-to-right; return once treasure found. $\longleftarrow \frac{n}{2} + 1$ *accesses in worst case*

# A toy problem

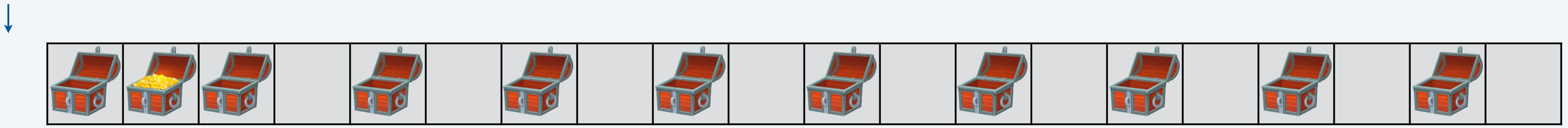
Treasure hunt. Length-$n$ array with 50% treasures, 50% duds.

Deterministic algorithms.

- scan the array left-to-right; return once treasure found. ⟵ $\frac{n}{2} + 1$ *accesses in worst case*
- scan the array right-to-left; return once treasure found. ⟵ $\frac{n}{2} + 1$ *accesses in worst case*

# A toy problem

Treasure hunt. Length-$n$ array with 50% treasures, 50% duds.



Deterministic algorithms.

- scan the array left-to-right; return once treasure found. ⟵ $\frac{n}{2} + 1$ *accesses in worst case*
- scan the array right-to-left; return once treasure found. ⟵ $\frac{n}{2} + 1$ *accesses in worst case*
- look at even entries, then odd; return once treasure found. ⟵ $\frac{n}{2} + 1$ *accesses in worst case*

# A toy problem

Treasure hunt. Length-$n$ array with 50% treasures, 50% duds.

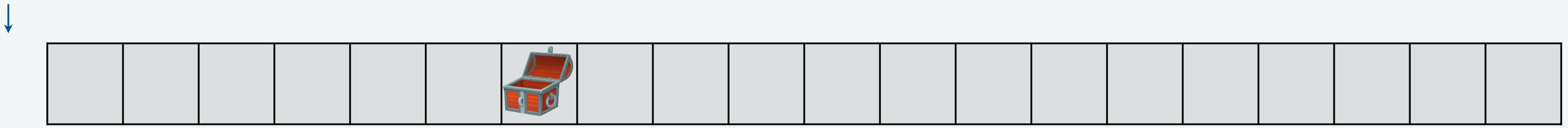
Proposition. For every deterministic algorithm, there is a 50%-treasure array where it makes $\frac{n}{2} + 1$ accesses.

Pf. Consider the sequence of $\frac{n}{2}$ accesses it makes when all are duds.

The array with duds there and treasures elsewhere requires $\frac{n}{2} + 1$ accesses. ▪

# A toy problem

Treasure hunt. Length-$n$ array with 50% treasures, 50% duds.



Randomized algorithms (Monte Carlo):

- look at a[StdRandom.uniformInt(n)], return treasure (if found). ⟵ *fails ≡ 1 flip lands tails*

Fails with probability $\frac{n/2}{n} = \frac{1}{2}$.

# A toy problem

Treasure hunt. Length-$n$ array with 50% treasures, 50% duds.

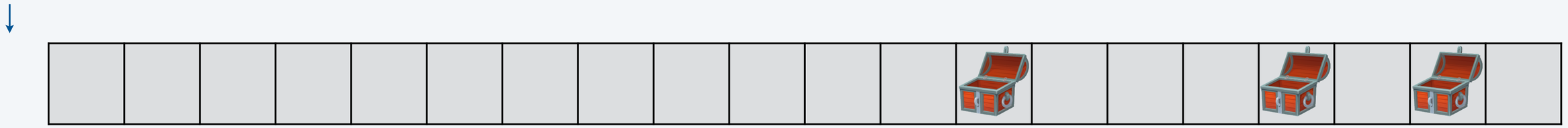
Randomized algorithms (Monte Carlo):

- look at a[StdRandom.uniformInt(n)], return treasure (if found). ← *fails ≡ 1 flip lands tails*
- look at two uniformly random entries, return 1st treasure found (if any). ← *fails ≡ 2 flips land tails*

Fails with probability $\frac{1}{2} \times \frac{1}{2}$.

# A toy problem

Treasure hunt. Length-$n$ array with 50% treasures, 50% duds.



Randomized algorithms (Monte Carlo):

- look at a[StdRandom.uniformInt(n)], return treasure (if found). ← *fails ≡ 1 flip lands tails*
- look at two uniformly random entries, return 1st treasure found (if any). ← *fails ≡ 2 flips land tails*
- look at three uniformly random entries, return 1st treasure found (if any). ← *fails ≡ 3 flips land tails*

Fails with probability $\frac{1}{2} \times \frac{1}{2} \times \frac{1}{2}$.

# A toy problem

Treasure hunt. Length-$n$ array with 50% treasures, 50% duds.

Randomized algorithms (Monte Carlo):

- look at a[StdRandom.uniformInt(n)], return treasure (if found). ← *fails ≡ 1 flip lands tails*
- look at two uniformly random entries, return 1st treasure found (if any). ← *fails ≡ 2 flips land tails*
- look at three uniformly random entries, return 1st treasure found (if any). ← *fails ≡ 3 flips land tails*
- look at $k$ uniformly random entries, return 1st treasure found (if any). ← *fails ≡ k flips land tails*

Fails with probability $\frac{1}{2^k}$.

# Randomness: quiz 3

**Suppose 1% of the array contains treasure and 99% contain duds. Then `a[StdRandom.uniformInt(n)]` finds a treasure with probability**

A. 1%

B. 10%

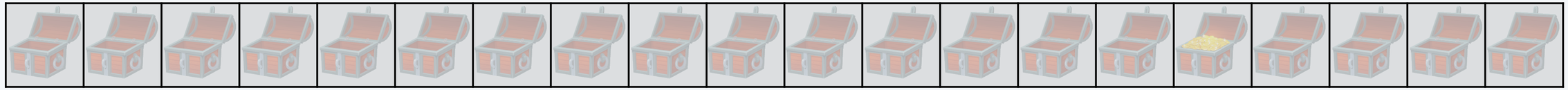
C. 50%

D. 99%

E. None of the above.

# Rare treasures and biased coins

Treasure hunt. Length-$n$ array with 1% treasures, 99% duds.



Randomized algorithm (Monte Carlo):

- look at $k$ uniformly random entries, return treasure (if found).

Failure probability $= \mathbb{P}[k$ biased coin flips land tails$]$

$= (0.99)^k$.

| outcome | probability |
|---|---|
| heads | 1/100 |
| tails | 99/100 |

**distribution of 99%-1% biased coin**

Remark. For $0.99^k < 1\,\%$, setting $k = 459$ suffices.

# Monte Carlo algorithms

Monte Carlo.

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

Error reduction. If $\mathbb{P}[A \text{ fails}] = p$ but want $\leq q$, repeat $k \geq \log_p q$ times and return best solution:

$$\mathbb{P}[A \text{ fails } k \text{ times}] = p^k \leq p^{\log_p q} = q$$

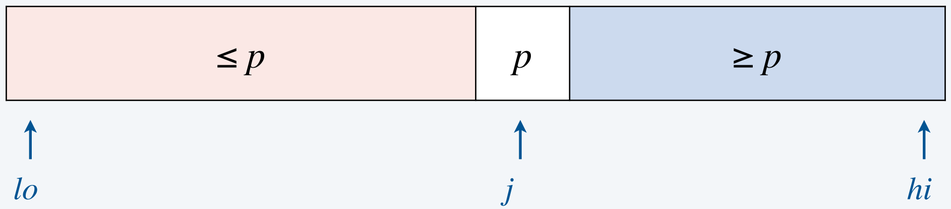
*independence*

# Las Vegas algorithms

Las Vegas.

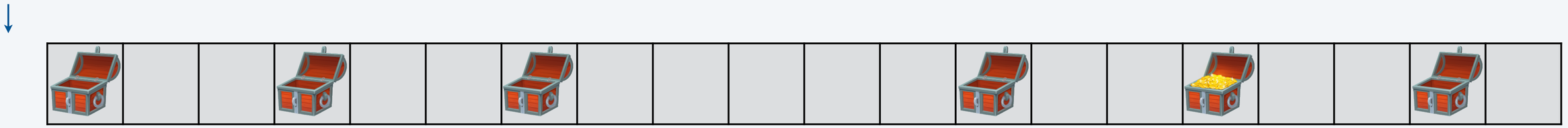
- Guaranteed to be correct.
- Running time depends on outcomes of coin flips.

Ex. Quicksort, quickselect.




# Las Vegas vs. Monte Carlo

Treasure hunt. Length-$n$ array with 50% treasures, 50% duds.



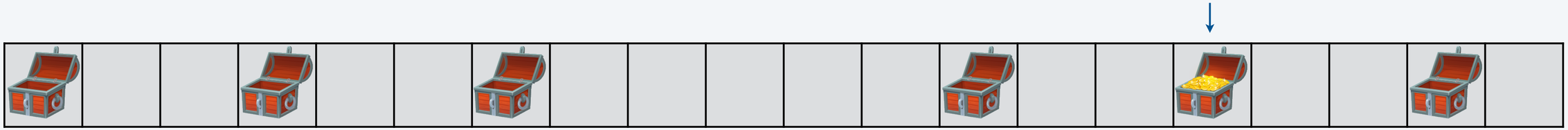
Randomized algorithm (Las Vegas):

- repeatedly look at uniformly random entry; return *only when* treasure found.

Returns in 1st try with probability 1/2.

# Las Vegas vs. Monte Carlo

Treasure hunt. Length-$n$ array with 50% treasures, 50% duds.



Randomized algorithm (Las Vegas):

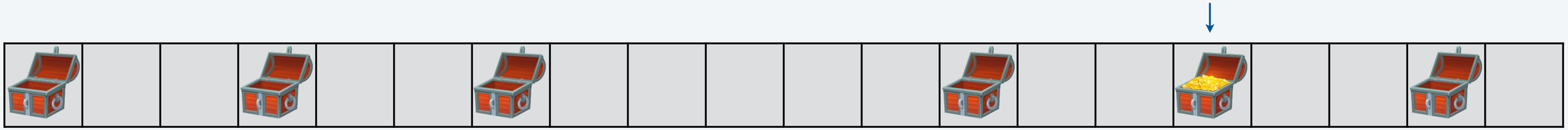
- repeatedly look at uniformly random entry; return *only when* treasure found.

Returns in 1st try with probability 1/2.

Returns in 2nd try with probability 1/4.

# Las Vegas vs. Monte Carlo

Treasure hunt. Length-$n$ array with 50% treasures, 50% duds.



Randomized algorithm (Las Vegas):

- repeatedly look at uniformly random entry; return *only when* treasure found.

Returns in 1st try with probability $1/2$.

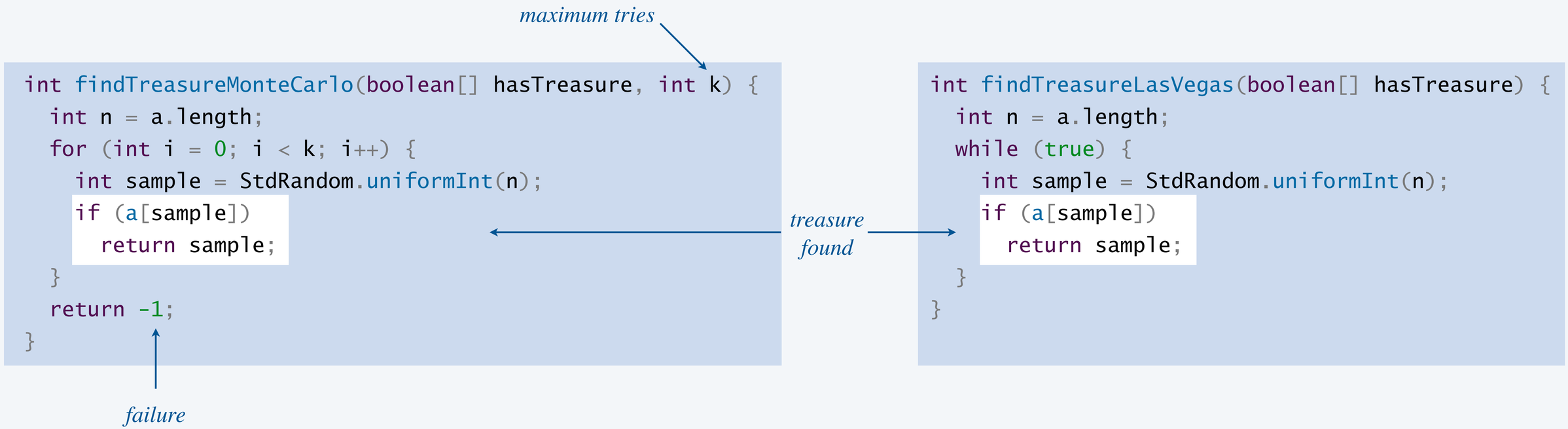
Returns in 2nd try with probability $1/4$.

⋮

Returns in $k^{\text{th}}$ try with probability $1/2^k$.

Expected # of array accesses: $1 \times \mathbb{P}[1 \text{ access}] \; + \; 2 \times \mathbb{P}[2 \text{ accesses}] \; + \; 3 \times \mathbb{P}[3 \text{ accesses}] \; + \; \cdots$

# Monte Carlo and Las Vegas in Java



# Randomness: quiz 4

**In the worst case, how many array accesses made by Las Vegas treasure hunt on length-$n$ array? (Recall: samples with replacement.)**

A. 1

B. 2

C. $n/2$

D. $n$

E. None of the above.

# Las Vegas Analysis

Treasure hunt. Length-$n$ array with a $p$ fraction of treasures, $1 - p$ of duds.

*geometric random variable with success probability p*

Proposition. Expected number of array accesses of Las Vegas algorithm is $1/p$.

Pf. Call $\mathbb{E}[A]$ the expected number of accesses.

- With probability $p$, first try succeeds $\Longrightarrow$ total is $1$.
- With probability $1 - p$, first try fails $\Longrightarrow$ total is $\mathbb{E}[A] + 1$.

Therefore,

$$\mathbb{E}[A] \;=\; p \cdot 1 \;+\; (1 - p) \cdot \big(\mathbb{E}[A] + 1\big),$$

whose solution is $\mathbb{E}[A] = 1/p$. ▪

# Randomness: quiz 5

**How many expected array accesses made by Las Vegas treasure hunt on length-$n$ array with 99% duds?**

A. 2

B. 99

C. 100

D. $n$

E. None of the above.


https://algs4.cs.princeton.edu

# RANDOMNESS

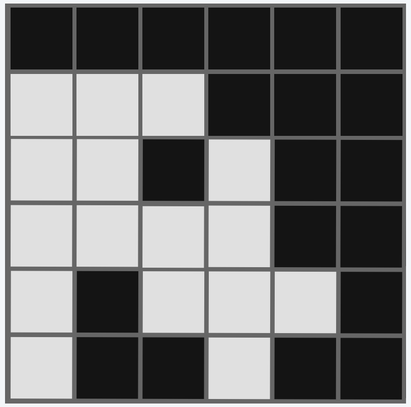
- *what it is and what it isn't*
- *Las Vegas and Monte Carlo*
- *rejection sampling*
- *Karger's algorithm*

# Uniform distribution of closed cells

Goal. Generate a random $(i, j)$ with `open[i,j] == false`.

Rejection sampling.

- Generate a random cell in $n$-by-$n$ grid without replacement.
- If cell is closed, use it; otherwise, repeat. ← *à la Las Vegas!*



```
int i, j;
do {
   i = StdRandom.uniformInt(n);
   j = StdRandom.uniformInt(n);
} while (isOpen(i, j));
```

*random $(i, j)$ in grid*

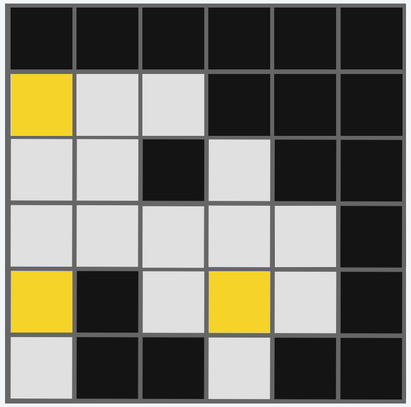
*repeat until closed*

# Uniform distribution of closed cells

Goal. Generate a random $(i, j)$ with `open[i,j] == false`.

Rejection sampling.

- Generate a random cell in $n$-by-$n$ grid without replacement.
- If cell is closed, use it; otherwise, repeat.

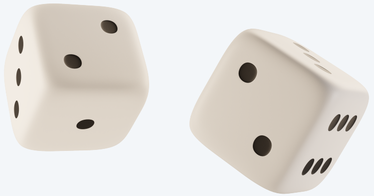


```
int i, j;
do {
   i = StdRandom.uniformInt(n);
   j = StdRandom.uniformInt(n);
} while (isOpen(i, j));
```

Expected iterations. $1/\alpha$ if $\alpha$ fraction closed. ⟵ *geometric random variable with success $\alpha$*

# Uniform distribution of bounded integers

Goal. Simulate roll of 6-sided die with uniform bits.



Rejection sampling.

- Sample 3 independent bits.
- Repeat or output as follows:
  - if TTT, output 1
  - if TTH, output 2
  - if THT, output 3
  - if THH, output 4
  - if HTT, output 5
  - if HTH, output 6
  - if HHT or HHH, repeat

# Uniform distribution of bounded integers

Goal. Generate an integer $0 \le r < n$ using uniform bits.

Rejection sampling.

- Sample $\lceil \log_2 n \rceil$ independent bits, call $r$ the integer with corresponding binary representation.
- If $r < n$, use it; otherwise, repeat.

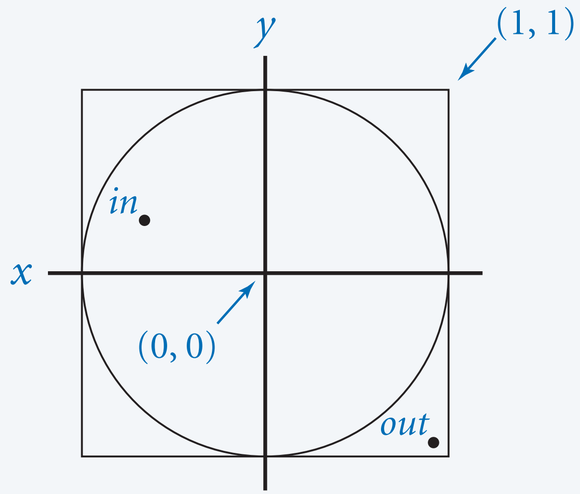
```
int r;
do {
   r = 0;
   for (int p = 1; p < n; p *= 2)
      r += p * StdRandom.bernoulli();
} while (r >= n);
```

*random* $0 \le r < 2^{\lceil \log_2 n \rceil}$

*repeat until* $< n$

# Uniform distribution in unit circle

Goal. Generate a random point in unit circle. ← *used in Fraud Detection!*




Rejection sampling.

- Generate a random point in 2-by-2 square centered at origin.
- If point is inside circle, use it; otherwise, repeat.

```
double x, y;
do {
   x = StdRandom.uniformDouble(-1.0, 1.0);
   y = StdRandom.uniformDouble(-1.0, 1.0);
} while (x*x + y*y > 1.0);
```

*random (x, y) in square*

*repeat until in the circle*

Remark. If $s$ out of $t$ samples in unit circle, $\frac{s}{t} \approx \frac{\pi}{4}$.


https://algs4.cs.princeton.edu

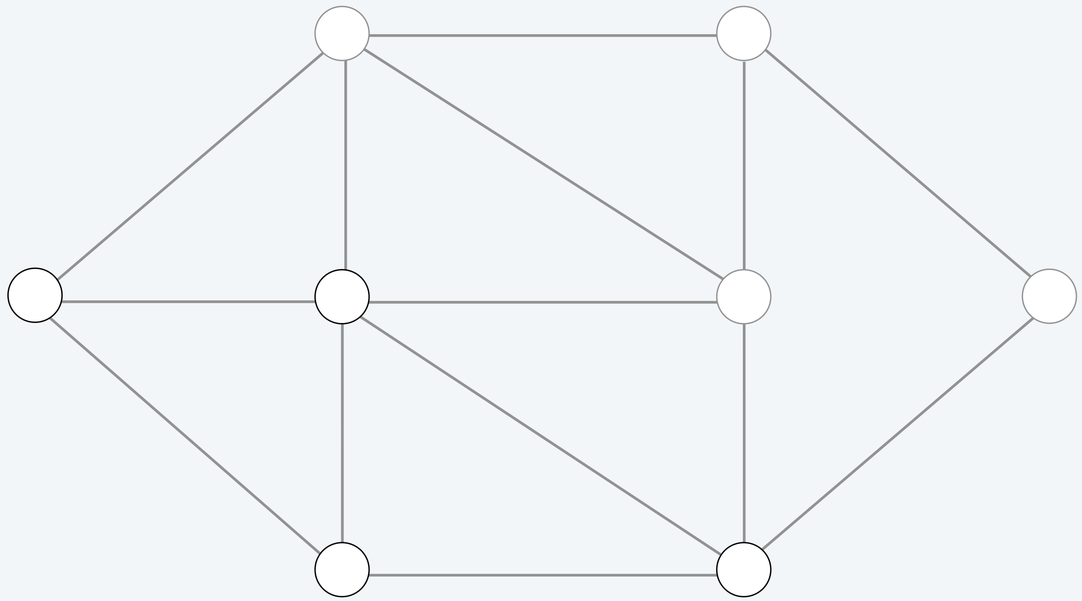
# RANDOMNESS

- *what it is and what it isn't*
- *Las Vegas and Monte Carlo*
- *rejection sampling*
- *Karger's algorithm*

# Global mincut problem

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

Equivalent. Smallest min $st$-cut among all pairs $(s, t)$, antiparallel edges of capacity 1.

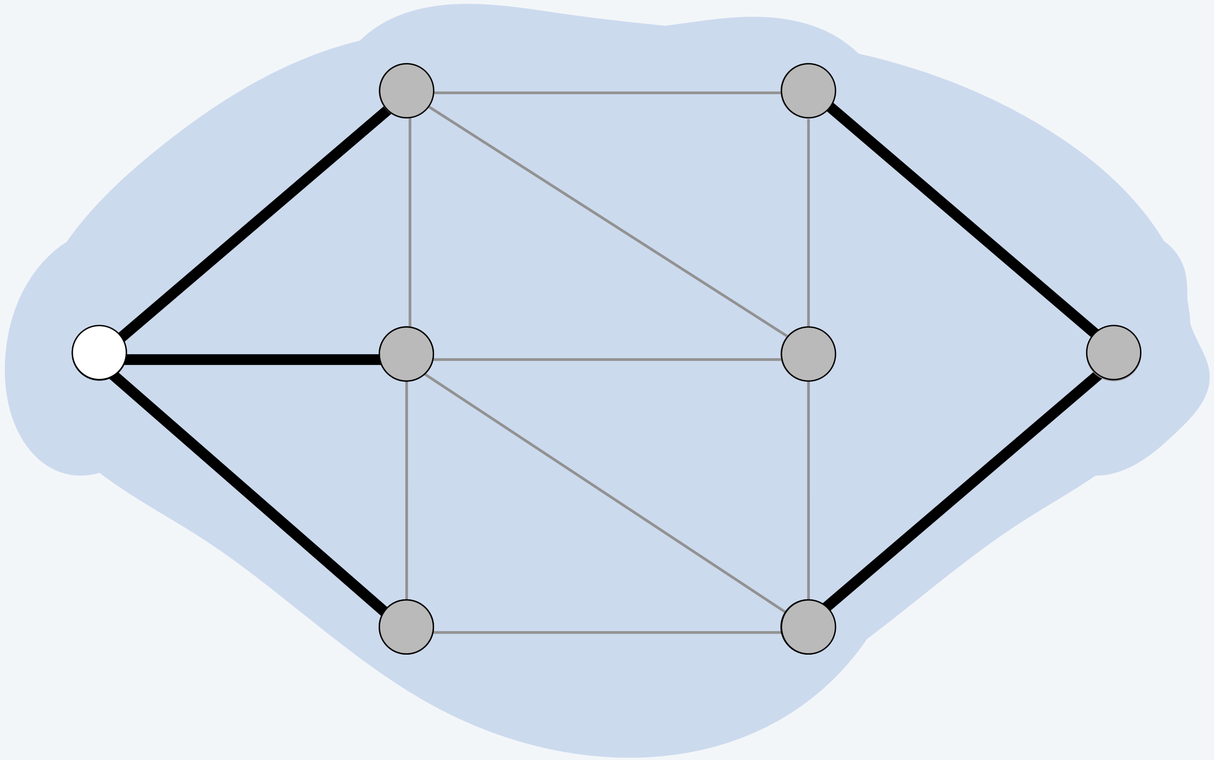


# Global mincut problem

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

Deterministic algorithms.

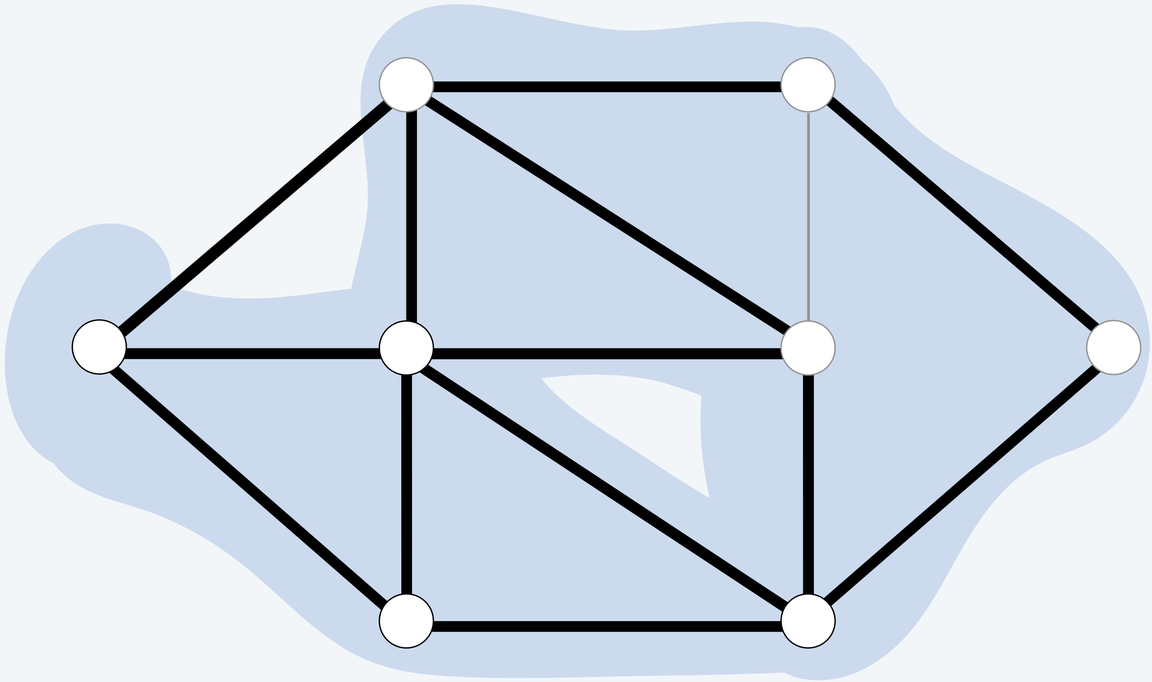
- Brute-force: iterate over all cuts, return smallest. [$2^{V-1} - 1$ cuts $\Longrightarrow$ exponential time!]
- Ford-Fulkerson-based: pick any $s$ as source, try every $t$ as target. [$V - 1$ runs of FF $\Longrightarrow$ $\Theta(V^2E^2)$ runtime]



# Global mincut problem

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

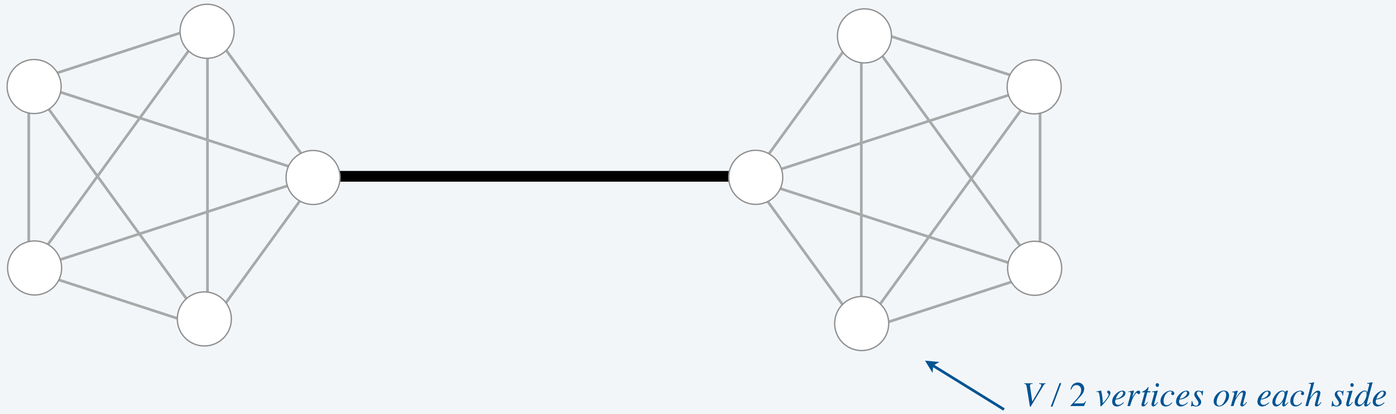
Idea. Pick a random cut.



# Mincut of dumbbell graph

Uniformly? There may be 1 mincut but $2^{V-1} - 1$ total cuts — takes a *lot* of luck to find it.

Example.



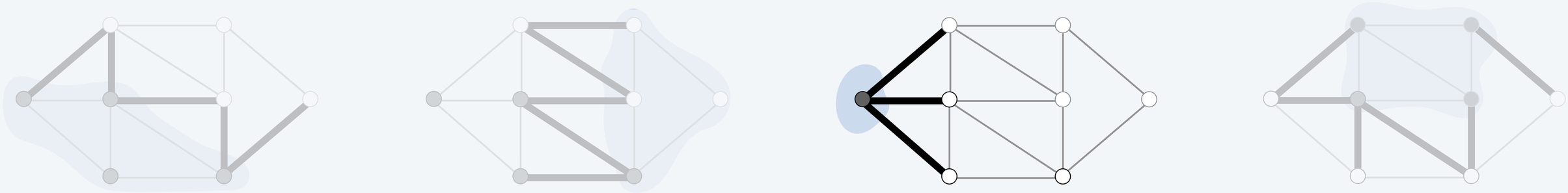
# Karger's global mincut algorithm

Algorithm.

- Assign a random weight (uniform between 0 and 1) to each edge $e$. ← *or: shuffle edges, build MST in resulting order*
- Run Kruskal's MST algorithm until 2 connected components left.
- Return cut defined by connected components.

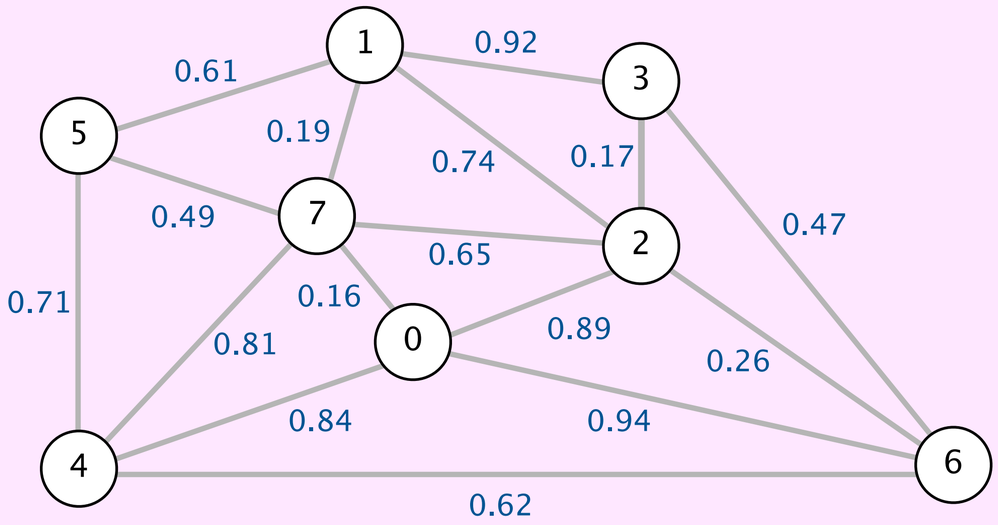
Probability of finding a mincut $\geq \frac{2}{V^2}$. [no mincut edges in either connected component]

Run algorithm many times and return best cut.



# Karger's algorithm demo

- **Assign random edge weights.**
- Run Kruskal's algorithm until 2 connected components left.

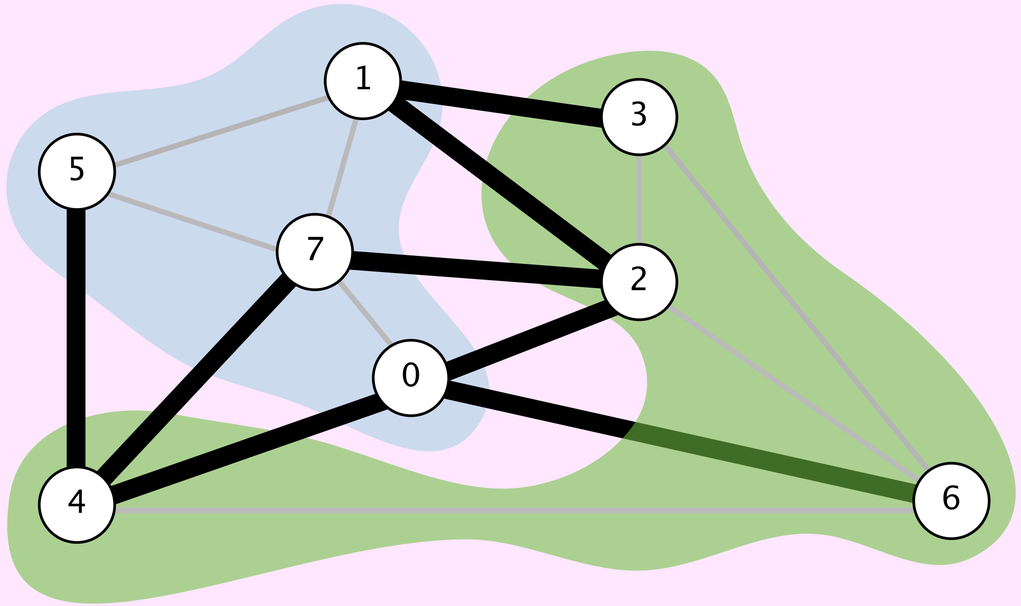



```
0-2  0.89
0-4  0.84
0-7  0.16
0-6  0.94
1-2  0.74
1-3  0.92
1-5  0.61
1-7  0.19
2-3  0.17
2-6  0.26
2-7  0.65
3-6  0.47
4-6  0.62
4-5  0.71
4-7  0.81
5-7  0.49
```

# Karger's algorithm demo

Consider edges in ascending order of weight.

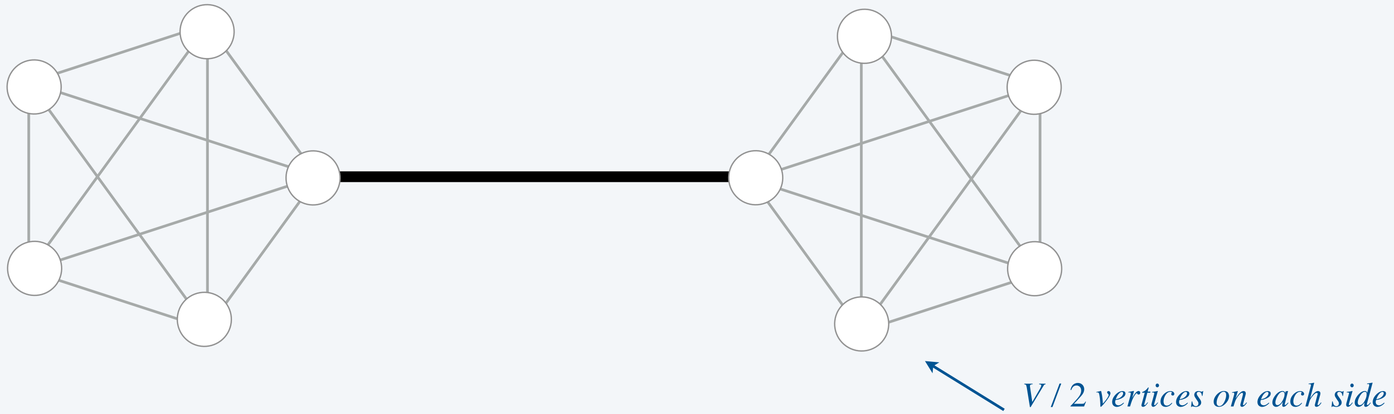
- Add next edge to $T$ unless doing so would create a cycle.
- Stop if $T$ contains $V - 2$ edges.




**a cut:** $\{0,1,5,7\}$ **and** $\{2,3,4,6\}$

```
0-7  0.16
2-3  0.17
1-7  0.19
2-6  0.26
3-6  0.47
5-7  0.49
1-5  0.61
4-6  0.62
2-7  0.65
4-5  0.71
1-2  0.74
4-7  0.81
0-4  0.84
0-2  0.89
1-3  0.92
0-6  0.94
```

# Karger on dumbbell graph




Karger succeeds if middle edge not in MST.

- with probability $1/E$, its weight is the largest.
- each side has $< (V/2)^2$ edges, so $E \leq V^2/2$.

$\Longrightarrow$ success with $\geq \frac{2}{V^2}$ probability!

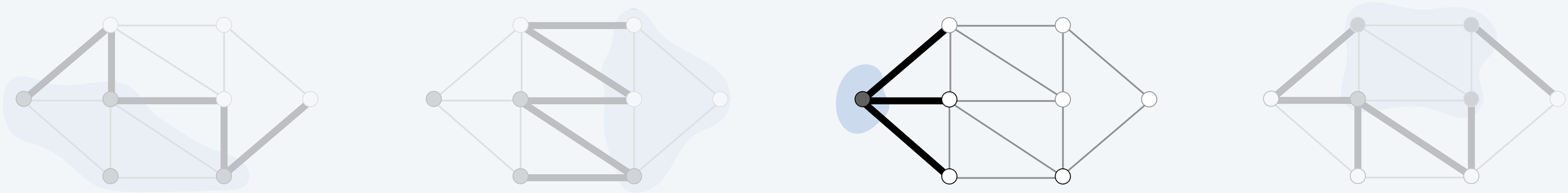
# Karger's global mincut algorithm

Algorithm.

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

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

Run algorithm many times and return best cut.



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

Remark 2. With clever idea, improved to $\Theta(E \log^3 V)$ time (still randomized).

# Randomness: quiz 6

**Smallest # of repetitions of Karger's algorithm to get correct answer with 99% probability?**

A. $\Theta(1)$

B. $\Theta(V)$

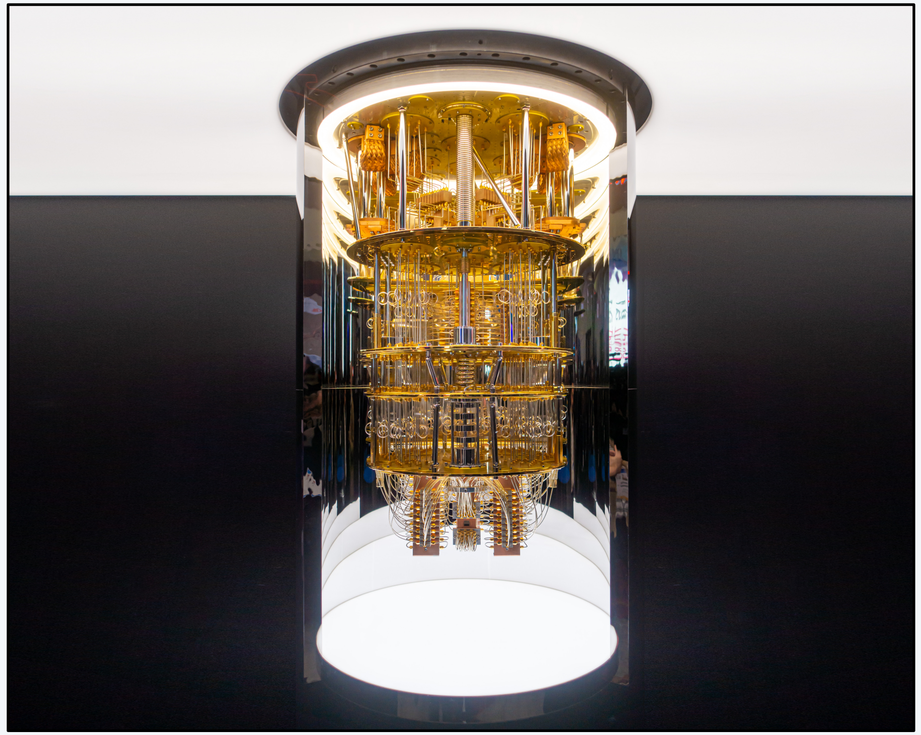
C. $\Theta(V^2)$

D. $\Theta(V^3)$

E. None of the above.

# Beyond this course

- Approximation algorithms [intractability: stay tuned!]
- Machine learning [randomized MW]
- Optimization [stochastic gradient descent]
- Cryptography [average-case hardness]
- Complexity theory [derandomization]
- Quantum computation [Shor's factoring algorithm]
- Networking [load balancing]
- Graphics [procedural generation]
- Physics [Monte Carlo simulation]
- Health sciences [randomized control trials]
- Mathematics [probabilistic method]



**IBM Quantum System One**

ORF 309. Probability and Stochastic Systems

COS 330. Great Ideas in Theoretical Computer Science

COS 433. Cryptography

# 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 |
| *Treasure chests* | 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/