CSCI 4030/6220 Randomized Algorithms

Monte Carlo vs. Las Vegas, the $A(x; r)$ model, and amplification

Alex Gittens
Department of Computer Science, Rensselaer Polytechnic Institute

Fall 2026

Why randomized algorithms?

Randomness is useful when a deterministic choice is brittle, expensive, or hard to analyze.

In this lecture, randomness appears in two different roles:

- as a way to make a fast verifier;
- as a way to avoid adversarial inputs in a sorting algorithm.

Those two examples will define the course vocabulary for Monte Carlo and Las Vegas algorithms.

Today's lecture

1. Two concrete algorithms: Freivalds and randomized Quicksort.

Today's lecture

1. Two concrete algorithms: Freivalds and randomized Quicksort.
2. Monte Carlo vs. Las Vegas as guarantee structures.

Today's lecture

1. Two concrete algorithms: Freivalds and randomized Quicksort.
2. Monte Carlo vs. Las Vegas as guarantee structures.
3. The unified seed model $A(x; r)$.

Today's lecture

1. Two concrete algorithms: Freivalds and randomized Quicksort.
2. Monte Carlo vs. Las Vegas as guarantee structures.
3. The unified seed model $A(x; r)$.
4. One-sided amplification by repetition.

PART 1

A First Randomized Algorithm

1

Challenge: verify a matrix product

You are given $A, B, C \in \mathbb{F}^{n \times n}$ and a claim that

$$AB = C.$$

Challenge: verify a matrix product

You are given $A, B, C \in \mathbb{F}^{n \times n}$ and a claim that

$$AB = C.$$

Can we check the claim faster than recomputing AB?

Challenge: verify a matrix product

You are given $A, B, C \in \mathbb{F}^{n \times n}$ and a claim that

$$AB = C.$$

Can we check the claim faster than recomputing AB?

For any vector $r \in \mathbb{F}^n$,

$$AB = C \quad \Longrightarrow \quad A(Br) = Cr.$$

The test is to pick one random vector and check whether the claim survives that projection.

Freivalds' matrix verification (1977)

A cheap randomized verifier

Monte Carlo

Input: $A, B, C \in \mathbb{F}^{n \times n}$.
Random choice: draw $r \in \{0,1\}^n$ uniformly.
Test: compute $y = Br$, $z = Ay$, and $w = Cr$.
Output: accept iff $z = w$.

Freivalds' matrix verification (1977)

A cheap randomized verifier

Monte Carlo

Input: $A, B, C \in \mathbb{F}^{n \times n}$.
Random choice: draw $r \in \{0,1\}^n$ uniformly.
Test: compute $y = Br$, $z = Ay$, and $w = Cr$.
Output: accept iff $z = w$.

Three matrix-vector products cost $O(n^2)$, far less than recomputing the full matrix product.

Freivalds' matrix verification (1977)

A cheap randomized verifier

Monte Carlo

Input: $A, B, C \in \mathbb{F}^{n \times n}$.
Random choice: draw $r \in \{0,1\}^n$ uniformly.
Test: compute $y = Br$, $z = Ay$, and $w = Cr$.
Output: accept iff $z = w$.

Three matrix-vector products cost $O(n^2)$, far less than recomputing the full matrix product.

First observation

If $AB = C$, the test always accepts. Any error can occur only when $AB \neq C$.

When can the output be incorrect?

Let

$$D = AB - C.$$

If $AB \neq C$, then $D \neq 0$ and Freivalds fails exactly when $Dr = 0$.

When can the output be incorrect?

Let

$$D = AB - C.$$

If $AB \neq C$, then $D \neq 0$ and Freivalds fails exactly when $Dr = 0$.

Pick a nonzero row $D_{i,:}$ and an index j with $D_{ij} \neq 0$. Failure implies

$$\sum_{\ell=1}^{n} D_{i\ell} r_\ell = 0.$$

When can the output be incorrect?

Let

$$D = AB - C.$$

If $AB \neq C$, then $D \neq 0$ and Freivalds fails exactly when $Dr = 0$.

Pick a nonzero row $D_{i,:}$ and an index j with $D_{ij} \neq 0$. Failure implies

$$\sum_{\ell=1}^{n} D_{i\ell} r_\ell = 0.$$

Condition on all coordinates of r except r_j. Then the equation becomes

$$r_j = -\alpha / D_{ij}$$

for some fixed α. At most one of $\{0, 1\}$ can satisfy that equality.

Freivalds' error bound

Fix r_ℓ for all $\ell \neq j$. The relevant row-dot-product has the form

$$\alpha + D_{ij} r_j,$$

where $D_{ij} \neq 0$.

Freivalds' error bound

Fix r_ℓ for all $\ell \neq j$. The relevant row-dot-product has the form

$$\alpha + D_{ij} r_j,$$

where $D_{ij} \neq 0$.
The conditional equation has at most one solution in $\{0, 1\}$, so

$$\mathbb{P}\big[D_{i,:} r = 0 \mid (r_\ell)_{\ell \neq j}\big] \leq \frac{1}{2}.$$

Freivalds' error bound

Fix r_ℓ for all $\ell \neq j$. The relevant row-dot-product has the form

$$\alpha + D_{ij} r_j,$$

where $D_{ij} \neq 0$.
The conditional equation has at most one solution in $\{0, 1\}$, so

$$\mathbb{P}\big[D_{i,:} r = 0 \mid (r_\ell)_{\ell \neq j}\big] \leq \frac{1}{2}.$$

By averaging over the conditioned coordinates,

$$\mathbb{P}[Dr = 0] \leq \frac{1}{2}.$$

Freivalds' error bound

Fix r_ℓ for all $\ell \neq j$. The relevant row-dot-product has the form

$$\alpha + D_{ij} r_j,$$

where $D_{ij} \neq 0$.
The conditional equation has at most one solution in $\{0, 1\}$, so

$$\mathbb{P}\big[D_{i,:} r = 0 \mid (r_\ell)_{\ell \neq j}\big] \leq \frac{1}{2}.$$

By averaging over the conditioned coordinates,

$$\mathbb{P}[Dr = 0] \leq \frac{1}{2}.$$

Freivalds' guarantee

If $AB = C$, accept with probability 1. If $AB \neq C$, accept with probability at most $1/2$.

Freivalds' error bound

Fix r_ℓ for all $\ell \neq j$. The relevant row-dot-product has the form

$$\alpha + D_{ij} r_j,$$

where $D_{ij} \neq 0$.
The conditional equation has at most one solution in $\{0, 1\}$, so

$$\mathbb{P}\big[D_{i,:} r = 0 \mid (r_\ell)_{\ell \neq j}\big] \leq \frac{1}{2}.$$

By averaging over the conditioned coordinates,

$$\mathbb{P}[Dr = 0] \leq \frac{1}{2}.$$

Freivalds' guarantee

If $AB = C$, accept with probability 1. If $AB \neq C$, accept with probability at most $1/2$.

Thus a Yes answer may be wrong, but a No answer is conclusive. This is **false-positive one-sided error**.

PART 2

A Second Randomized Algorithm

2

Challenge: sort an array

Sorting problem

Input: an array $a = [a_1, \ldots, a_n]$ of distinct comparable elements.
Output: a permutation of a in increasing order.

Challenge: sort an array

Sorting problem

Input: an array $a = [a_1, \ldots, a_n]$ of distinct comparable elements.
Output: a permutation of a in increasing order.

Divide-and-conquer suggests choosing a pivot, separating smaller and larger elements, and recursively sorting the two sides.

Challenge: sort an array

Sorting problem

Input: an array $a = [a_1, \ldots, a_n]$ of distinct comparable elements.
Output: a permutation of a in increasing order.

Divide-and-conquer suggests choosing a pivot, separating smaller and larger elements, and recursively sorting the two sides.

The design question

How much can the pivot rule affect the amount of recursive work?

Quicksort (Hoare, 1961)

Quicksort

Input: array a of distinct comparable elements.
Base case: if $|a| \leq 1$, return a.
Pivot: let p be the first element of a.
Partition: form $L = [x \in a : x < p]$ and $R = [x \in a : x > p]$.
Recurse: let $L' = \text{Quicksort}(L)$ and $R' = \text{Quicksort}(R)$.
Output: return $L' \,\|\, [p] \,\|\, R'$.

One partition step

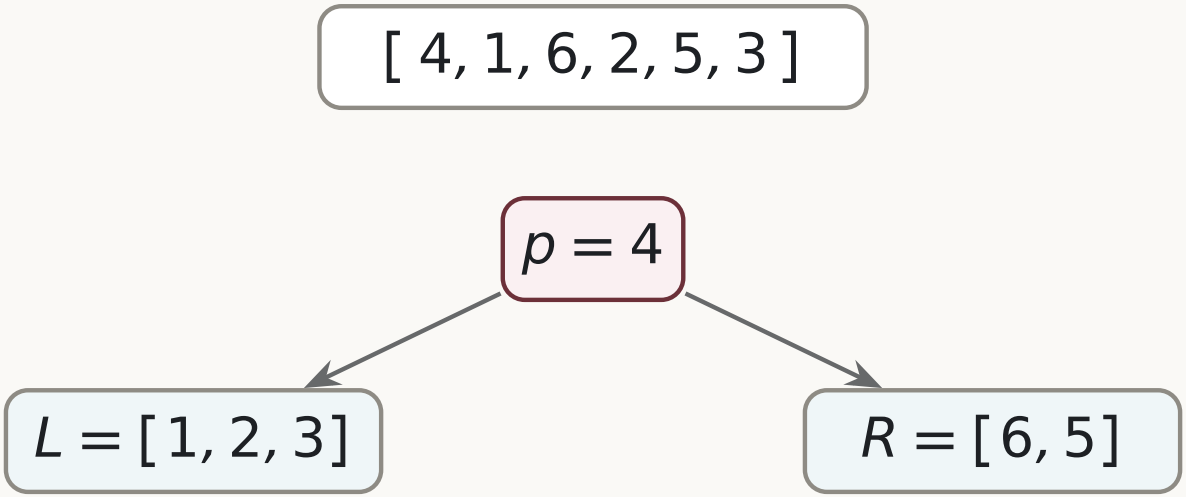
[4, 1, 6, 2, 5, 3]

One partition step

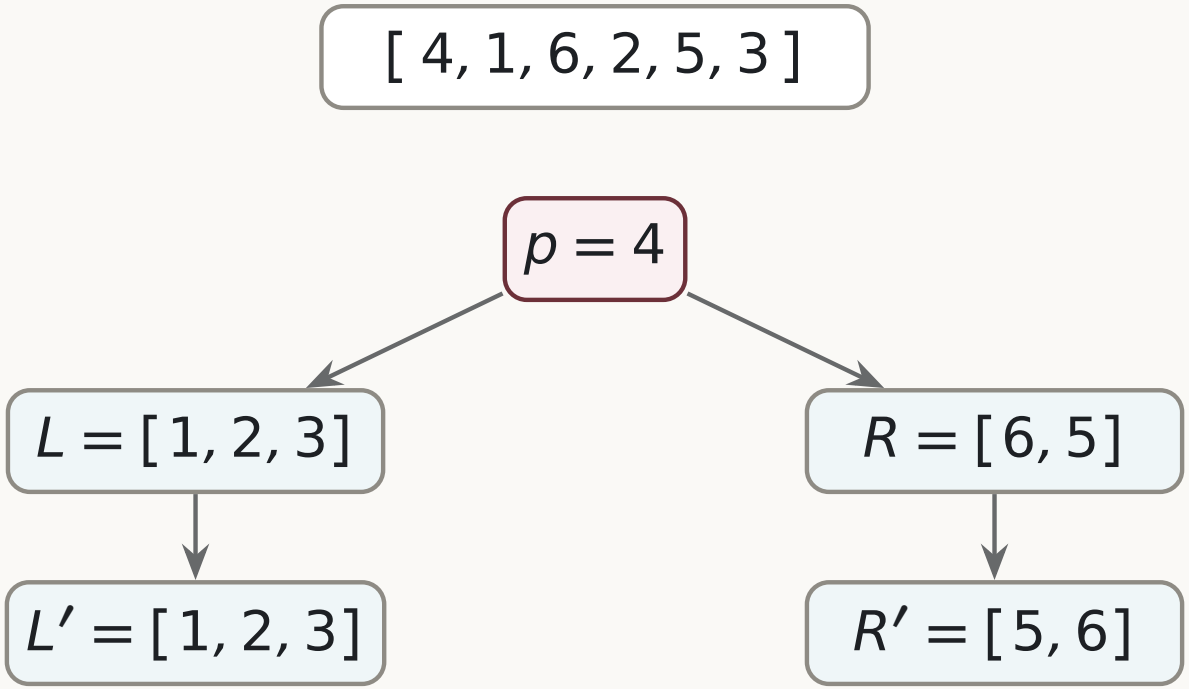
[4, 1, 6, 2, 5, 3]

$p = 4$

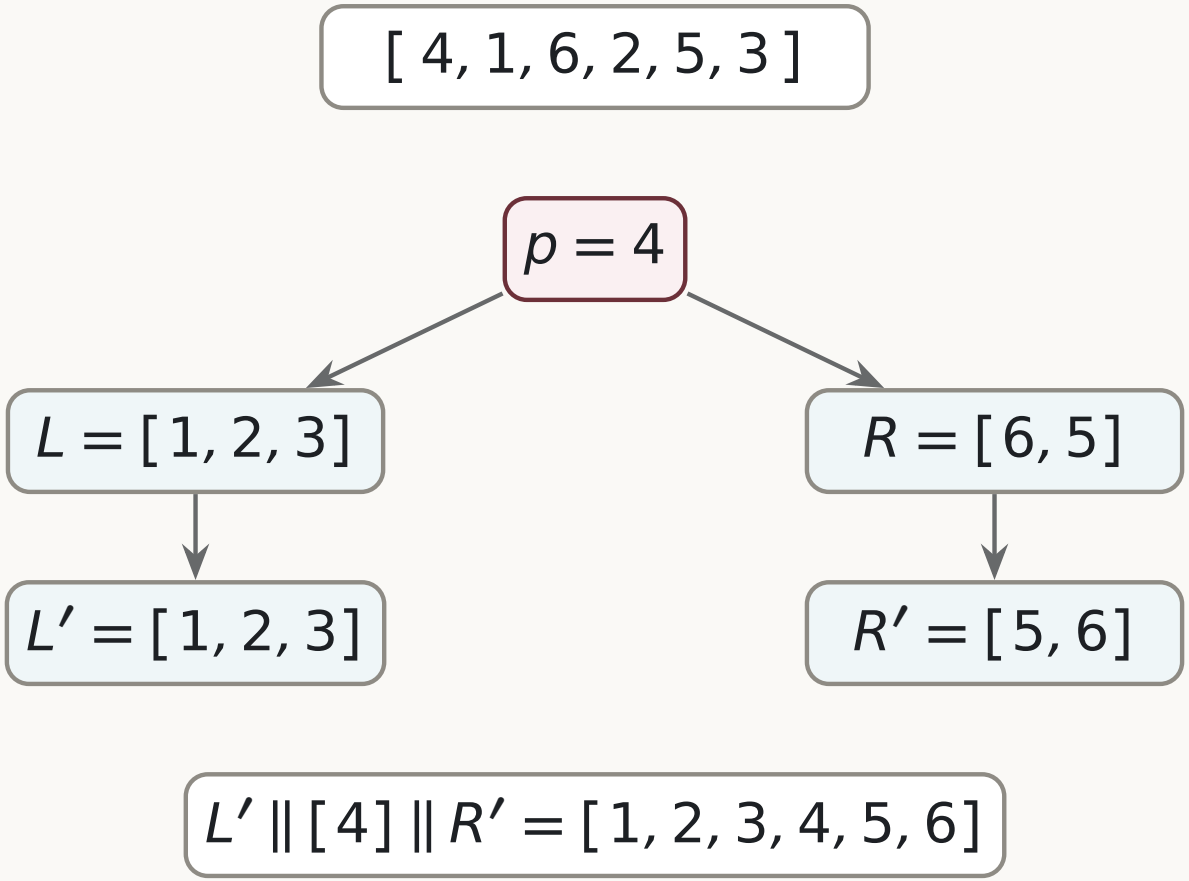
One partition step



One partition step



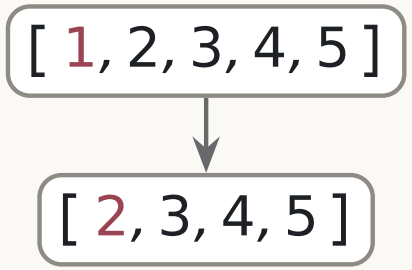
One partition step



First-element pivots can repeat the worst split

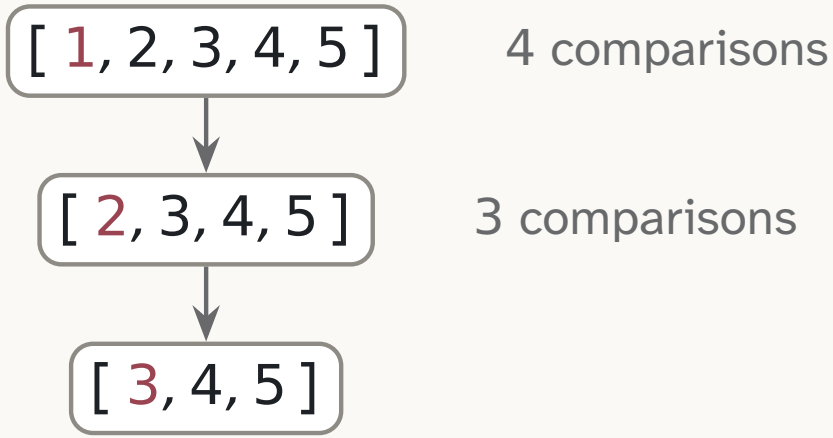
[1, 2, 3, 4, 5]

First-element pivots can repeat the worst split

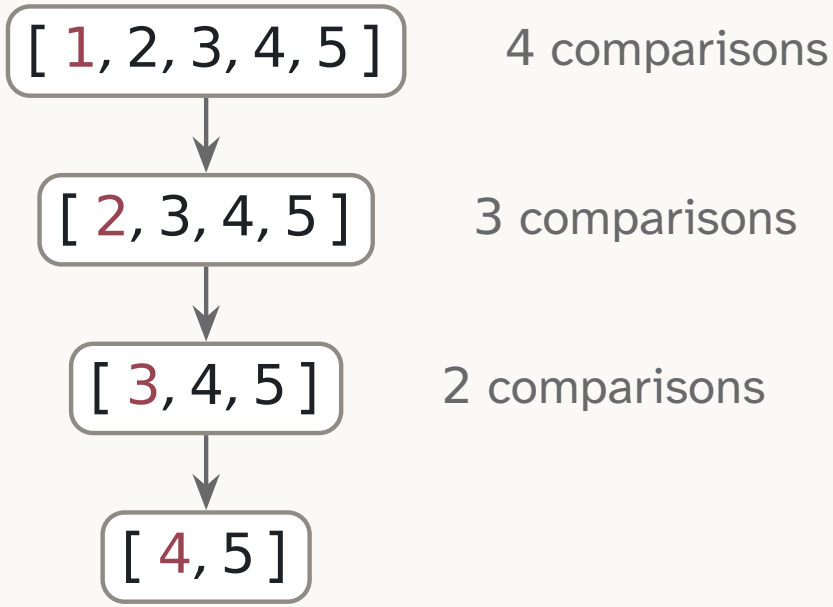


4 comparisons

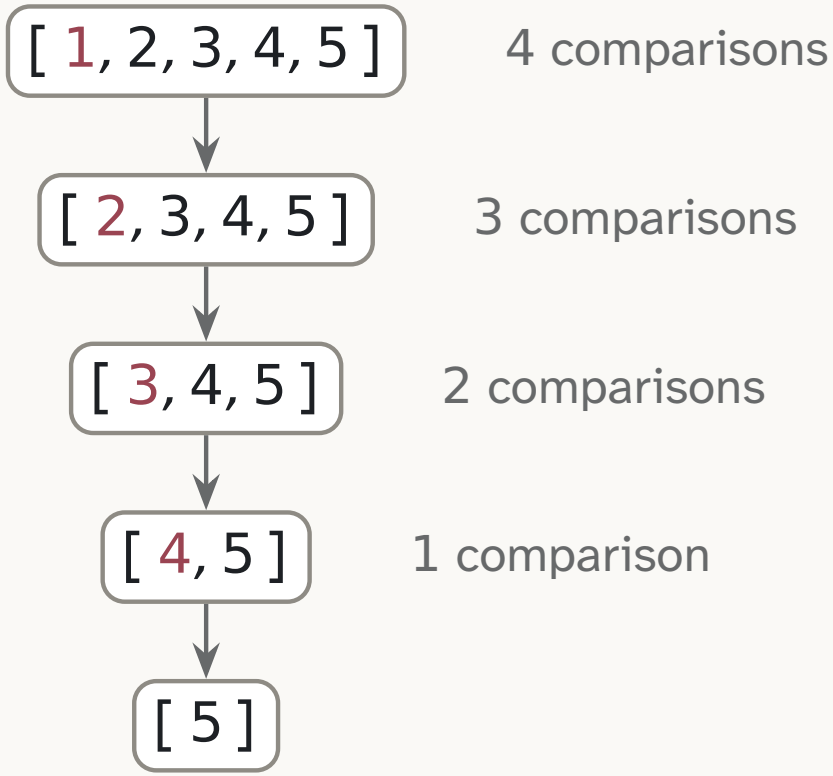
First-element pivots can repeat the worst split



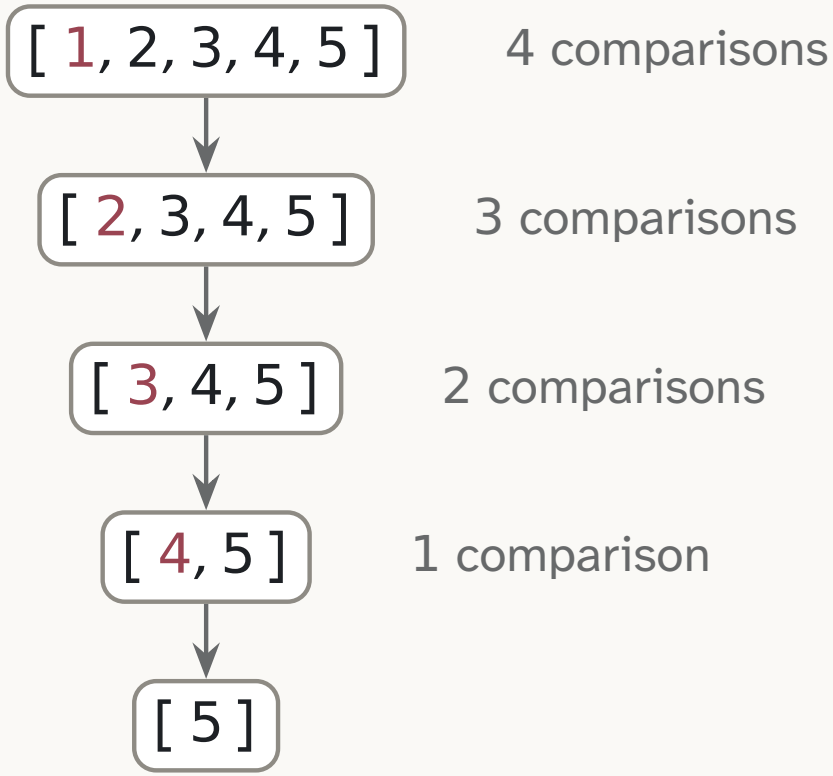
First-element pivots can repeat the worst split



First-element pivots can repeat the worst split



First-element pivots can repeat the worst split



$$T(n) = T(n-1) + \Theta(n) \quad \Longrightarrow \quad T(n) = \Theta(n + (n-1) + \cdots + 1) = \Theta(n^2).$$

Randomized Quicksort

Change only the pivot rule

Randomized Quicksort

Input: array a of n elements.
Base case: if $|a| \leq 1$, return a.
Random choice: pick a pivot index uniformly from the current subarray.
Partition: form $L = [x \in a : x < p]$ and $R = [x \in a : x > p]$.
Recurse: let $L' = \text{Quicksort}(L)$ and $R' = \text{Quicksort}(R)$.
Output: return $L' \,\|\, [p] \,\|\, R'$.

Randomized Quicksort

Change only the pivot rule

Randomized Quicksort

Input: array a of n elements.
Base case: if $|a| \leq 1$, return a.
Random choice: pick a pivot index uniformly from the current subarray.
Partition: form $L = [x \in a : x < p]$ and $R = [x \in a : x > p]$.
Recurse: let $L' = \text{Quicksort}(L)$ and $R' = \text{Quicksort}(R)$.
Output: return $L' \,\|\, [p] \,\|\, R'$.

For a current subarray of size m, partitioning uses exactly $m - 1 = |L| + |R|$ pivot comparisons. Thus

$$T(m) = T(|L|) + T(|R|) + (m - 1).$$

Random pivot choices determine the subproblem sizes, and therefore the total number of comparisons.

Why does randomization help?

Random pivot on any fixed input:

- the pivot rank is random;
- consistently extreme pivots are unlikely;
- the expected number of comparisons is $O(n \log n)$.

Why does randomization help?

Random pivot on any fixed input:

- the pivot rank is random;
- consistently extreme pivots are unlikely;
- the expected number of comparisons is $O(n \log n)$.

Key insight

Randomization prevents a fixed input order from forcing the same bad split at every recursive call.

Expected number of comparisons

Randomized Quicksort guarantee

Assume distinct keys. Then $\mathbb{E}[\text{comparisons}] = O(n \log n)$.

Expected number of comparisons

Randomized Quicksort guarantee

Assume distinct keys. Then $\mathbb{E}[\text{comparisons}] = O(n \log n)$.

The worst-case number of comparisons is still $O(n^2)$, but that requires repeatedly picking an extreme element at every recursive stage.

Expected number of comparisons

Randomized Quicksort guarantee

Assume distinct keys. Then $\mathbb{E}[\text{comparisons}] = O(n \log n)$.

The worst-case number of comparisons is still $O(n^2)$, but that requires repeatedly picking an extreme element at every recursive stage.
The proof of the expected bound uses indicator variables and linearity of expectation, which are the topic of Lecture 2.

PART 3

Monte Carlo and Las Vegas

3

Two algorithms, two guarantee types

	Freivalds' matrix check	**Quicksort**
Runtime	Always $O(n^2)$	$\mathbb{E} = O(n \log n)$; worst $O(n^2)$
Correctness	May be wrong ($\mathbb{P} \leq 1/2$)	Always correct
Randomness affects	Correctness	Runtime

Two algorithms, two guarantee types

	Freivalds' matrix check	**Quicksort**
Runtime	Always $O(n^2)$	$\mathbb{E} = O(n \log n)$; worst $O(n^2)$
Correctness	May be wrong ($\mathbb{P} \leq 1/2$)	Always correct
Randomness affects	Correctness	Runtime

These are fundamentally different guarantee structures. What should we call them?

Two canonical guarantee models

A randomized algorithm consults random bits during execution. The key distinction is what randomness is allowed to affect.

Monte Carlo

Running time is bounded.
The answer may be wrong.
Randomness affects **correctness**.

Las Vegas

The answer is always correct.
Running time is random.
Randomness affects **work**.

Two canonical guarantee models

A randomized algorithm consults random bits during execution. The key distinction is what randomness is allowed to affect.

Monte Carlo

Running time is bounded.
The answer may be wrong.
Randomness affects **correctness**.

Las Vegas

The answer is always correct.
Running time is random.
Randomness affects **work**.

Mnemonic

Monte Carlo: Maybe Correct. **Las Vegas:** Length Varies.

PART 4

A Unified Model

4

A unifying model: $A(x; r)$

A randomized algorithm can be written as a deterministic function

$$A(x; r),$$

where r is the string of random bits used by the algorithm.

A unifying model: $A(x; r)$

A randomized algorithm can be written as a deterministic function

$$A(x; r),$$

where r is the string of random bits used by the algorithm.

For each fixed r, define

$$A_r(x) := A(x; r).$$

Then

randomized algorithm $\longleftrightarrow$ distribution over deterministic A_r.

A unifying model: $A(x; r)$

A randomized algorithm can be written as a deterministic function

$$A(x; r),$$

where r is the string of random bits used by the algorithm.

For each fixed r, define

$$A_r(x) := A(x; r).$$

Then

$$\boxed{\text{randomized algorithm}} \quad \longleftrightarrow \quad \boxed{\text{distribution over deterministic } A_r}.$$

In this language,

$$\mathbb{P}_R[A(x; R) \text{ is correct}] \geq 1 - \delta$$

is a Monte Carlo guarantee.

The order of the quantifiers matters

Fix an input length n. Compare

$$\exists r_n \; \forall x \in \{0,1\}^n : A(x; r_n) \text{ is correct}$$

with

$$\forall x \in \{0,1\}^n : \mathbb{P}_{R_n}[A(x; R_n) \text{ is correct}] \geq 1 - \delta.$$

The order of the quantifiers matters

Fix an input length n. Compare

$$\exists r_n \; \forall x \in \{0,1\}^n : A(x; r_n) \text{ is correct}$$

with

$$\forall x \in \{0,1\}^n : \mathbb{P}_{R_n}[A(x; R_n) \text{ is correct}] \geq 1 - \delta.$$

Different quantifier patterns

The first statement asks for one seed that works on every length-n input. The second allows the good seeds to depend on the input.

Why the seed model matters

The object $A(x; r)$ gives a common language for later techniques.

- **Amplification:** use independent seeds $r_1, \ldots, r_k$.

Why the seed model matters

The object $A(x; r)$ gives a common language for later techniques.

- **Amplification:** use independent seeds $r_1, \ldots, r_k$.
- **Derandomization:** replace random choice of r by a deterministic construction or search.

Why the seed model matters

The object $A(x; r)$ gives a common language for later techniques.

- **Amplification:** use independent seeds $r_1, \ldots, r_k$.
- **Derandomization:** replace random choice of r by a deterministic construction or search.

Why the seed model matters

The object $A(x; r)$ gives a common language for later techniques.

- **Amplification:** use independent seeds $r_1, \ldots, r_k$.
- **Derandomization:** replace random choice of r by a deterministic construction or search.

For every fixed input x, many seeds may work well. That does not imply that one seed works well for every input simultaneously.

PART 5

Amplification

5

False-positive amplification: from q to q^k

A **false positive** is an incorrect Yes answer. Suppose No answers are always correct, while every No instance satisfies

$$\mathbb{P}_R[A(x;R) = \text{Yes}] \leq q.$$

False-positive amplification: from q to q^k

A **false positive** is an incorrect Yes answer. Suppose No answers are always correct, while every No instance satisfies

$$\mathbb{P}_R[A(x;R) = \text{Yes}] \leq q.$$

Run k independent copies and return Yes iff all k copies return Yes (the And rule).

False-positive amplification: from q to q^k

A **false positive** is an incorrect Yes answer. Suppose No answers are always correct, while every No instance satisfies

$$\mathbb{P}_R[A(x;R) = \text{Yes}] \leq q.$$

Run k independent copies and return Yes iff all k copies return Yes (the And rule).
Every Yes instance is still accepted. A No instance produces a false positive only if all copies return Yes, so its error probability is at most

$$q^k.$$

False-positive amplification: from q to q^k

A **false positive** is an incorrect Yes answer. Suppose No answers are always correct, while every No instance satisfies

$$\mathbb{P}_R[A(x;R) = \text{Yes}] \leq q.$$

Run k independent copies and return Yes iff all k copies return Yes (the And rule).
Every Yes instance is still accepted. A No instance produces a false positive only if all copies return Yes, so its error probability is at most

$$q^k.$$

For Freivalds, $q = 1/2$.

False-positive amplification: from q to q^k

A **false positive** is an incorrect Yes answer. Suppose No answers are always correct, while every No instance satisfies

$$\mathbb{P}_R[A(x;R) = \text{Yes}] \leq q.$$

Run k independent copies and return Yes iff all k copies return Yes (the And rule).
Every Yes instance is still accepted. A No instance produces a false positive only if all copies return Yes, so its error probability is at most

$$q^k.$$

For Freivalds, $q = 1/2$.

Freivalds

With $k = 20$ repetitions, the error is at most $2^{-20} < 10^{-6}$.

Summary

1. A randomized algorithm is rigorous when we can identify its failure event and prove that event unlikely.

Summary

1. A randomized algorithm is rigorous when we can identify its failure event and prove that event unlikely.
2. Freivalds replaces full matrix multiplication by a random projection test and has false-positive one-sided error at most $1/2$ per run.

Summary

1. A randomized algorithm is rigorous when we can identify its failure event and prove that event unlikely.
2. Freivalds replaces full matrix multiplication by a random projection test and has false-positive one-sided error at most $1/2$ per run.
3. Monte Carlo algorithms randomize correctness; Las Vegas algorithms preserve correctness and randomize runtime.

Summary

1. A randomized algorithm is rigorous when we can identify its failure event and prove that event unlikely.
2. Freivalds replaces full matrix multiplication by a random projection test and has false-positive one-sided error at most $1/2$ per run.
3. Monte Carlo algorithms randomize correctness; Las Vegas algorithms preserve correctness and randomize runtime.
4. The seed model $A(x; r)$ views a randomized algorithm as a distribution over deterministic algorithms.

Summary

1. A randomized algorithm is rigorous when we can identify its failure event and prove that event unlikely.
2. Freivalds replaces full matrix multiplication by a random projection test and has false-positive one-sided error at most $1/2$ per run.
3. Monte Carlo algorithms randomize correctness; Las Vegas algorithms preserve correctness and randomize runtime.
4. The seed model $A(x; r)$ views a randomized algorithm as a distribution over deterministic algorithms.
5. Independent repetition amplifies false-positive one-sided error using the And rule and a direct product bound.

Summary

1. A randomized algorithm is rigorous when we can identify its failure event and prove that event unlikely.
2. Freivalds replaces full matrix multiplication by a random projection test and has false-positive one-sided error at most $1/2$ per run.
3. Monte Carlo algorithms randomize correctness; Las Vegas algorithms preserve correctness and randomize runtime.
4. The seed model $A(x; r)$ views a randomized algorithm as a distribution over deterministic algorithms.
5. Independent repetition amplifies false-positive one-sided error using the And rule and a direct product bound.

Summary

1. A randomized algorithm is rigorous when we can identify its failure event and prove that event unlikely.
2. Freivalds replaces full matrix multiplication by a random projection test and has false-positive one-sided error at most $1/2$ per run.
3. Monte Carlo algorithms randomize correctness; Las Vegas algorithms preserve correctness and randomize runtime.
4. The seed model $A(x; r)$ views a randomized algorithm as a distribution over deterministic algorithms.
5. Independent repetition amplifies false-positive one-sided error using the And rule and a direct product bound.

Next lecture

Indicator variables, linearity of expectation, and the union bound: proving the $O(n \log n)$ expected comparison bound for randomized Quicksort.

References

- M. Mitzenmacher and E. Upfal. *Probability and Computing*. Cambridge University Press, 2nd ed., 2017. Chapters 1.1–1.3 and 2.
- R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge University Press, 1995. Chapters 1 and 5.
- R. Freivalds. Probabilistic machines can use less running time. *IFIP Congress*, 1977.
- C. A. R. Hoare. Quicksort. *The Computer Journal*, 1962.

PART 6

Appendix: Taxonomy and conversions

6

Exact vs. approximate randomized algorithms

All examples so far are **exact-correctness** problems: the answer is either right or wrong.

Exact vs. approximate randomized algorithms

All examples so far are **exact-correctness** problems: the answer is either right or wrong.

Decision problems

Answer is Yes/No.

Freivalds is a decision problem.

Function problems

Output the unique correct value.

Randomized Quicksort is a function problem.

Exact vs. approximate randomized algorithms

All examples so far are **exact-correctness** problems: the answer is either right or wrong.

Decision problems

Answer is Yes/No.

Freivalds is a decision problem.

Function problems

Output the unique correct value.

Randomized Quicksort is a function problem.

Many later randomized algorithms are for approximation or estimation, where quality lives on a continuum rather than a binary right/wrong scale.

Challenge: determine the majority bit by sampling

Majority-bit problem

Input: a bit string $b = (b_1, \ldots, b_n)$ with noticeably more 0's than 1's, or noticeably more 1's than 0's.
Output: the bit that occurs more often.

Challenge: determine the majority bit by sampling

Majority-bit problem

Input: a bit string $b = (b_1, \ldots, b_n)$ with noticeably more 0's than 1's, or noticeably more 1's than 0's.
Output: the bit that occurs more often.

Reading the entire string determines the answer exactly. Can a small random sample determine it with low error?

Randomized Majority Determination

Randomized Majority Determination

Input: bit string b, odd sample count k, and promised gap $\gamma > 0$.
Promise: if $p = \frac{1}{n}\sum_i b_i$, then $|p - 1/2| \geq \gamma$.
Random choice: independently sample k indices uniformly with replacement.
Output: return the majority bit among the k sampled positions.

Randomized majority has two-sided error

Guarantee

If $|p - 1/2| \geq \gamma$, then

$$\mathbb{P}[\text{sampled majority is incorrect}] \leq \exp(-2\gamma^2 k).$$

Randomized majority has two-sided error

Guarantee

If $|p - 1/2| \geq \gamma$, then

$$\mathbb{P}[\text{sampled majority is incorrect}] \leq \exp(-2\gamma^2 k).$$

If 1 is the true majority, the sample can incorrectly return 0. If 0 is the true majority, the sample can incorrectly return 1.

Randomized majority has two-sided error

Guarantee

If $|p - 1/2| \geq \gamma$, then

$$\mathbb{P}[\text{sampled majority is incorrect}] \leq \exp(-2\gamma^2 k).$$

If 1 is the true majority, the sample can incorrectly return 0. If 0 is the true majority, the sample can incorrectly return 1.

Two-sided Monte Carlo

The algorithm has bounded running time, but either possible answer can be wrong.

We can convert Las Vegas to Monte Carlo!

Let $\tau(x)$ be the random running time of a Las Vegas algorithm, and suppose

$$\mathbb{E}[\tau(x)] \leq T(n).$$

We can convert Las Vegas to Monte Carlo!

Let $\tau(x)$ be the random running time of a Las Vegas algorithm, and suppose

$$\mathbb{E}[\tau(x)] \leq T(n).$$

Abort after $2T(n)$ steps and output an arbitrary answer on timeout. By Markov's inequality,

$$\mathbb{P}[\tau(x) > 2T(n)] \leq \frac{1}{2}.$$

We can convert Las Vegas to Monte Carlo!

Let $\tau(x)$ be the random running time of a Las Vegas algorithm, and suppose

$$\mathbb{E}[\tau(x)] \leq T(n).$$

Abort after $2T(n)$ steps and output an arbitrary answer on timeout. By Markov's inequality,

$$\mathbb{P}[\tau(x) > 2T(n)] \leq \frac{1}{2}.$$

Conversion

A known expected-time bound yields a bounded-time Monte Carlo algorithm by truncation.

Can we convert Monte Carlo to Las Vegas?

Not in general. We need a way to recognize when a trial has succeeded.

Can we convert Monte Carlo to Las Vegas?

Not in general. We need a way to recognize when a trial has succeeded.

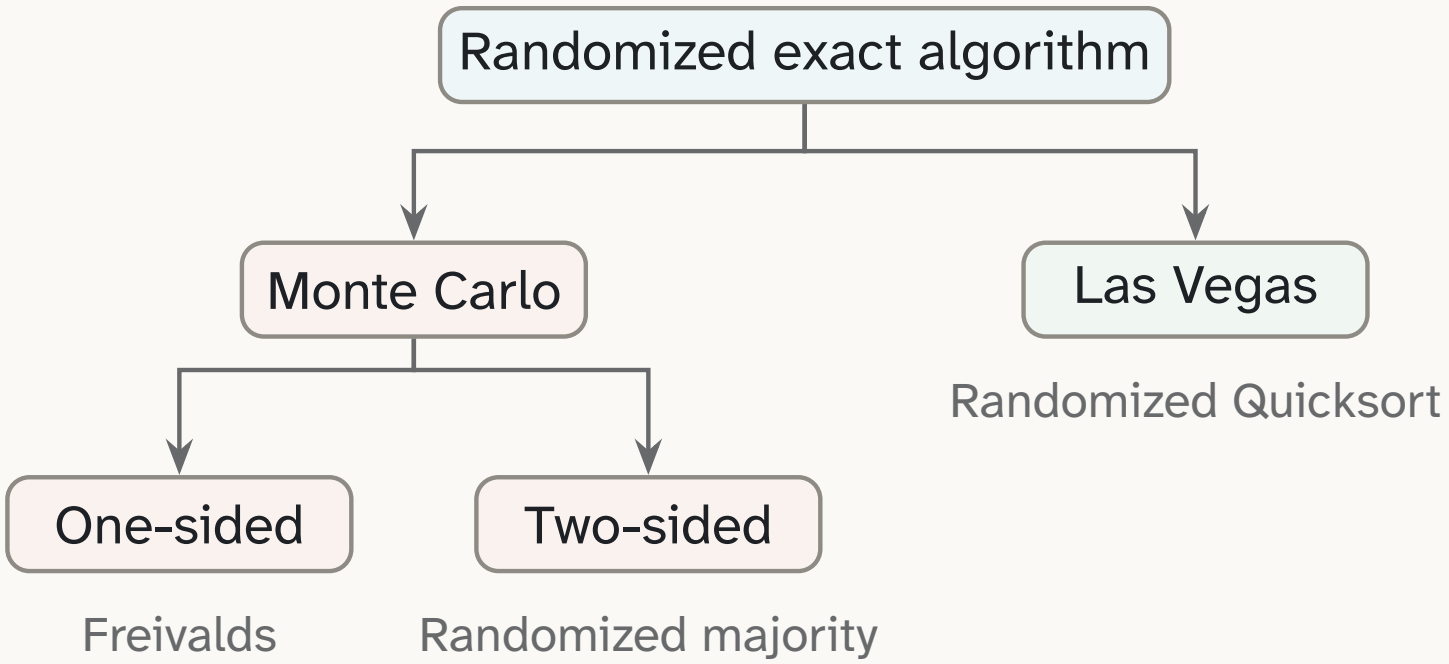
Search problems

If one trial returns a verifiably correct solution with probability at least $p > 0$, independent restart until verification succeeds gives a Las Vegas algorithm with expected $1/p$ trials.

Decision problems

In a one-sided test, only one answer is conclusive. Without a certificate for the other answer, restart does not tell us when that answer is correct.

Taxonomy of randomized algorithms



Classification check

For each guarantee, decide whether it is Monte Carlo, Las Vegas, both, or neither.

1. Always returns the correct answer; expected runtime is $O(n^2)$.

Classification check

For each guarantee, decide whether it is Monte Carlo, Las Vegas, both, or neither.

1. Always returns the correct answer; expected runtime is $O(n^2)$.
2. Always halts in $O(n^2)$ time; returns the wrong answer with probability at most 10^{-3}.

Classification check

For each guarantee, decide whether it is Monte Carlo, Las Vegas, both, or neither.

1. Always returns the correct answer; expected runtime is $O(n^2)$.
2. Always halts in $O(n^2)$ time; returns the wrong answer with probability at most 10^{-3}.
3. Always returns the correct answer and always halts in $O(n^2)$ time.

Two-sided amplification by majority vote

Suppose a binary-output algorithm has advantage $\gamma > 0$: on every input,

$$\mathbb{P}[\text{one run is correct}] \geq \frac{1}{2} + \gamma.$$

Two-sided amplification by majority vote

Suppose a binary-output algorithm has advantage $\gamma > 0$: on every input,

$$\mathbb{P}[\text{one run is correct}] \geq \frac{1}{2} + \gamma.$$

Run an odd number k of independent copies and return the majority answer. Let X_i indicate that run i is correct. The majority is wrong only if

$$\sum_{i=1}^{k} X_i \leq \frac{k}{2}, \qquad \mathbb{E}\left[\sum_{i=1}^{k} X_i\right] \geq \left(\frac{1}{2} + \gamma\right) k.$$

Two-sided amplification by majority vote

Suppose a binary-output algorithm has advantage $\gamma > 0$: on every input,

$$\mathbb{P}[\text{one run is correct}] \geq \frac{1}{2} + \gamma.$$

Run an odd number k of independent copies and return the majority answer. Let X_i indicate that run i is correct. The majority is wrong only if

$$\sum_{i=1}^{k} X_i \leq \frac{k}{2}, \qquad \mathbb{E}\left[\sum_{i=1}^{k} X_i\right] \geq \left(\frac{1}{2} + \gamma\right) k.$$

By Hoeffding's inequality,

$$\mathbb{P}[\text{majority is wrong}] \leq e^{-2\gamma^2 k}.$$

Amplifying one-sided error

The name identifies which *reported answer* can be wrong.

Error type	**Meaning**	**Combination**	**Error**
False positive	A Yes answer may be wrong	Return Yes iff all do (And)	q^k
False negative	A No answer may be wrong	Return Yes iff any does (Or)	q^k

Amplifying one-sided error

The name identifies which *reported answer* can be wrong.

Error type	Meaning	Combination	Error
False positive	A Yes answer may be wrong	Return Yes iff all do (And)	q^k
False negative	A No answer may be wrong	Return Yes iff any does (Or)	q^k

Freivalds has false-positive error, so it uses the And rule.