Lecture 1 — Broader Introduction (Širší úvod)

> Course **2-INF-135/15 Pravdepodobnostné algoritmy**, LS 2025/26. Source slides: `slidy_01_sirsi_uvod.pdf` (26 pages).

Course outline

The whole course is structured into five blocks:

1. **Broader introduction** — motivational examples (this lecture).
2. **Basic models and complexity classes** — what a randomized algorithm *is* formally, and the classes RP, coRP, BPP, ZPP, PP.
3. **Methods** — the recurring tricks (probability amplification, the probabilistic method, fingerprinting, random walks, …).
4. **More about complexity classes** — relationships between them.
5. **Derandomization** — removing the randomness while keeping the speed.

This first lecture is just a *gallery*. Each example is a small story whose punchline is the same: **a few random coin flips can replace a lot of deterministic work, at the price of a tiny, controllable probability of error.** The recurring questions are always:

- What is the **resource** we save (communication bits? comparisons? time?)
- What is the **probability of error**, and can we make it as small as we like?

---

1. Database equivalence — fingerprinting with primes

The problem

Two computers, `RI` and `RII`, each hold an n-bit string:

- `RI` holds $X = x_1 x_2 \dots x_n \in \{0,1\}^n$,
- `RII` holds $Y = y_1 y_2 \dots y_n \in \{0,1\}^n$.

We want to test whether $X = Y$ while sending as few bits across the network as possible. Read the bit strings as integers:

$$x = \mathrm{bin}(X), \qquad y = \mathrm{bin}(Y), \qquad 0 \le x, y < 2^n.$$

The trivial deterministic solution is to ship all n bits of one string to the other side. We want to do dramatically better.

The protocol

`RI` picks a **random prime p** from the primes up to n^2:

$$p \in_R \text{Primes}(n^2).$$

Then:

1. `RI` computes the *fingerprint* $s = x \bmod p$ and sends the pair (p, s) to `RII`.
2. `RII` computes $r = y \bmod p$ and answers **"equal"** iff $s = r$.

Both p and s are numbers below n^2, so each needs about $\log(n^2) = 2\log n$ bits.

> **Communication cost:** $\approx 4 \log n$ bits, versus n bits for the deterministic protocol. An exponential saving.

Correctness

- If $X = Y$, then $x = y$, so $x \bmod p = y \bmod p$ **always** — the answer is correct.
- If $X \neq Y$, the protocol errs **only when** $x \equiv y \pmod p$, i.e. when p divides $x - y$.

So the one-sided error appears only on unequal inputs, and only for the "unlucky" primes that happen to divide the difference.

Probability of error

Suppose $X \neq Y$, so $x - y \neq 0$. The fingerprint fails exactly for primes p dividing $x - y$:

$$p_{\text{err}} = \frac{\#\{p \in \text{Primes}(n^2) : p \mid (x - y)\}}{|\text{Primes}(n^2)|}.$$

How many bad primes are there? Write the prime factorization

$$|x - y| = p_{i_1}^{j_1} p_{i_2}^{j_2} \cdots p_{i_k}^{j_k}, \qquad j_t > 0.$$

Each prime is ≥ 2, and $|x - y| < 2^n$, so the number of *distinct* prime factors satisfies

$$2^k \leq |x - y| < 2^n \implies k \leq n - 1.$$

So **at most $n - 1$ primes** can divide $x - y$.

How many primes are there to choose from? By the prime-counting estimate

$$|\mathrm{Primes}(m)| \sim \frac{m}{\ln m}, \qquad \text{here } m = n^2,$$

so $|\mathrm{Primes}(n^2)| \sim \dfrac{n^2}{\ln n^2} = \dfrac{n^2}{2\ln n}$.

Putting it together:

$$p_{\text{err}} \le \frac{n-1}{n^2/(2\ln n)} < \frac{2\ln n}{n}.$$

> **Punchline.** With only $O(\log n)$ communicated bits we get error $O\left(\dfrac{\ln n}{n}\right)$, which already vanishes as n grows — and could be driven down further by repetition.

Why primes? The bound $k \le n-1$ uses only that distinct primes multiply up fast. Choosing the modulus from a pool ($\sim n^2/\ln n$ primes) much larger than the number of bad ones ($\le n-1$) is exactly what makes a random pick almost surely good. This trick — replacing an object by its *fingerprint modulo a random prime* — comes back again and again.

---

2. Two-way probabilistic finite automaton (2PFA) for $a^n b^n$

> **Naming.** *2PFA* = **two-way** probabilistic finite automaton. The "2" means the read head can move **both directions**, so the machine can re-scan the input as many times as it wants. That two-way ability is exactly what makes this algorithm possible — and it returns as the punchline at the end.

What we are up against

$L = \{a^n b^n \mid n \in \mathbb{N}\}$ is the classic **non-regular** language. A deterministic finite automaton has *fixed, finite* memory, but comparing two arbitrary counts n and m needs unbounded memory — so no DFA can do it.

Our model adds two small powers:

- **two-way head** — it can go back and re-scan the input as often as it likes;
- **a fair coin** — equivalently, an extra tape of random $0/1$ bits it may read.

With these we *can* recognize L, but only with **bounded error**. Constants k and L control how small that error is.

The one idea: compare 2^{-n} and 2^{-m} instead of n and m

We cannot store n or m, but we can turn a *count* into a *probability*:

> Sweep over the block of a's, flipping a coin above each one. The event **"all n coins came up heads"** has probability exactly 2^{-n}.

Do the same over the b's and you get an event of probability 2^{-m}. Now the comparison is easy in spirit:

- if $n = m$, the two rare events 2^{-n} and 2^{-m} are **equally likely**;
- if $n \neq m$, the **larger** block has the **rarer** "all heads" event.

The machine never learns n or m — it only *feels* which of the two events fires more often.

The algorithm

Input $w \in \{a, b\}^*$; constants k, L control the error.

1. **Cheap deterministic sieve.** Check that w has the shape $a^n b^m$ (all a's, then all b's) **and** that $n \equiv m \pmod{k}$. Both need only finite memory (counting mod k is k states). If either fails, **reject**.

2. **The coin-flip race.** Repeatedly sweep the input, flipping a coin over each symbol. Per sweep:

   - **a-success** = every one of the n a's flipped `1` (and the b-block came out *mixed*, both `0` and `1`) — probability $\approx 2^{-n}$;
   - **b-success** = every one of the m b's flipped `1` (and the a-block came out mixed) — probability $\approx 2^{-m}$.

   *(The "other block is mixed" clause only makes a-success and b-success **mutually exclusive** — at most one per sweep — so the race below is well defined. It barely changes the probabilities.)*

3. **The decision.** Watch the sequence of successes. **Reject** if L a-successes occur before any b-success, *or* L b-successes occur before any a-success. Otherwise **accept**.

In words: if one side runs away with the race, the counts are unequal → reject; if the race stays balanced, the counts are equal → accept.

Why step 1 (the mod-k check) is not optional

It does two jobs:

1. It instantly rejects wrong-shape strings and everything with $n \not\equiv m \pmod{k}$ — all of which are genuinely **not** in L.

2. **The crucial one:** if a string *survives* step 1 but still has $n \neq m$, then $n - m$ is a nonzero multiple of k, so

$$|n - m| \geq k.$$

That guarantees a **minimum gap** of k between the counts — exactly what makes the two rare events differ by a usable factor. Without it, n and m could differ by just 1, so 2^{-n} and 2^{-m} would differ by only a factor of 2 — too weak to detect reliably.

Correctness — the two cases

Case $n = m$ (string *is* in L → we want to accept). a-success and b-success are equally likely, so each success in the race is a-type or b-type with probability $\frac{1}{2}$, independently. We *wrongly* reject only if the first L successes are all the same type:

$$p_{\text{reject}} = \underbrace{2}_{\text{either side}} \cdot \left(\tfrac{1}{2}\right)^{L} = 2^{\,1-L}.$$

With $L = 3$: $p_{\text{reject}} = 0.25$, i.e. we correctly **accept with probability 0.75**.

Case $n > m$ (string *not* in L → we want to reject). Because step 1 forces $n \equiv m \pmod{k}$, if $n \neq m$ then $n = m + ik \geq m + k$. An a-success now needs k *extra* heads compared to a b-success, so it is at least 2^k times rarer:

$$\frac{\Pr[a\text{-success}]}{\Pr[a\text{- or }b\text{-success}]} \leq \frac{1}{2^k + 1}.$$

So b-successes dominate the race, and we correctly reject with probability

$$p_{\text{reject}} \geq \left(1 - \frac{1}{2^k + 1}\right)^{L}.$$

With $L = 3,\ k = 2$: $\left(\frac{4}{5}\right)^3 = 0.512$.

input	should	correct with prob.
$a^n b^n$	accept	≥ 0.75
wrong counts	reject	≥ 0.512

Both are bounded away from $\frac{1}{2}$, so this is genuine **bounded error**. The gap is modest; amplify it the usual way — run many independent copies and take a majority vote (and tune k, L).

> **Subtle tension (good oral-exam point).** Raising L shrinks the $n = m$ error 2^{1-L} — but it *also* shrinks the reject probability $\left(1 - \frac{1}{2^k+1}\right)^L$ in the $n \neq m$ case (a base below 1 raised to a higher power). So you cannot just crank L; you must raise k alongside it (a bigger guaranteed gap ⇒ a bigger per-success bias) to keep **both** errors small.

The catch: correct but exponentially slow — the deep point

An a-success has probability $\approx 2^{-n}$ per sweep, so you expect to wait about 2^n sweeps just to *see* one. Hence the machine recognizes $\{a^n b^n\}$ with bounded error but in **expected exponential time**.

This is not a flaw of this particular construction — it is **unavoidable** (Dwork–Stockmeyer, building on Freivalds):

> Any bounded-error 2PFA recognizing a **non-regular** language must run in **expected exponential time**.

> **Punchline.** A two-way finite automaton plus a coin is *just barely* powerful enough to escape regularity — and it pays for that power with exponential time. Randomness buys a new capability here, and the price tag is explicit.

---

3. Randomized min-cut — Karger's contraction algorithm

The problem

- **Input:** an unweighted multigraph $G = (V, E)$, $|V| = n$.
- **Output:** a partition of V into V_1, V_2 minimizing the number of crossing edges $|E^*| = |(V_1 \times V_2) \cap E|$.

Best deterministic algorithms run in $O(n^3)$ (or $O(|V|\,|E| \log(|V|^2/|E|))$). The randomized algorithm is strikingly simple.

The algorithm: random edge contraction

```
1.  label(v) ← v              for every vertex
2.  while more than 2 vertices remain:
        pick e = (x, y) ∈_R E uniformly at random
        contract e: merge x and y into a single vertex z   (G ← contract(G, e))
        label(z) ← label(x) ∪ label(y)    (keep parallel edges, drop self-loops)
3.  now G has exactly two vertices u, v:
        return (label(u), label(v))
```

Each contraction merges the two endpoints of a random edge into one super-vertex, keeping multi-edges (they represent "how strongly connected" the groups are) but deleting loops. When only two super-vertices remain, their labels are the two sides of the cut. **Complexity** $O(n^2)$ per run.

Analysis

Fix a particular minimum cut $C_{\min}$ of size k (assume for simplicity it is unique). The algorithm returns $C_{\min}$ **iff it never contracts one of its** k **edges.**

Key fact — minimum degree. Every vertex has degree $\geq k$ (otherwise the single-vertex cut around it would be smaller than k). Hence

$$|E| = \frac{1}{2}\sum_{v} \deg(v) \geq \frac{nk}{2}.$$

Let E_i be the event "$C_{\min}$ has survived the first i contractions", and let G/F_i be the graph after contracting the edges F_i of the first i steps. The same degree argument on G/F_i (which has $n-i$ super-vertices) gives

$$|E(G/F_i)| \geq \frac{(n-i)\,k}{2}.$$

Step 1 survives: we must avoid the k cut-edges out of $\geq nk/2$ edges,

$$\Pr[E_1] = \frac{|E|-k}{|E|} = 1 - \frac{k}{|E|} \geq 1 - \frac{2}{n}.$$

Step i survives, given the cut survived so far:

$$\Pr\left[E_i \,\middle|\, \bigcap_{1\leq j\leq i-1} E_j\right] = \frac{|E(G/F_{i-1})|-k}{|E(G/F_{i-1})|} \geq 1 - \frac{2}{n-i+1}.$$

The cut survives all $n-2$ **contractions** (telescoping product):

$$\Pr\left[\bigcap_j E_j\right] \geq \prod_{1\leq j\leq n-2}\left(1-\frac{2}{n-j+1}\right) = \prod_{1\leq j\leq n-2}\frac{n-j-1}{n-j+1} = \cdots = \frac{2}{n(n-1)} > \frac{2}{n^2}.$$

> **One run finds a fixed minimum cut with probability $> 2/n^2$.** That looks tiny, but it is *polynomially* small, so a polynomial number of independent runs makes failure exponentially small.

Amplification by repetition. Failure of one run is $\leq 1 - 2/n^2$, so after t independent runs (keeping the best cut found),

$$\Pr[\text{all fail}] \leq \left(1-\frac{2}{n^2}\right)^t \leq e^{-2t/n^2}.$$

Taking $t = \frac{n^2}{2}\log n$ gives error $\leq e^{-\log n} = \frac{1}{n} = O(1/n)$.

- **Total time:** $O(n^2)\cdot t = O(n^4 \log n)$. 🙂 correct with high probability, 😟 slower than the deterministic algorithm as written.

> **Punchline.** A laughably simple "keep merging random edges" routine has a *guaranteed* good success probability per run, and repetition turns that into high confidence. (Later improvements — Karger–Stein — cut the time dramatically by recognizing that early contractions are safe and only the *late* ones are risky.)

---

4. Randomized QuickSort (RQS)

The algorithm

```
RQS(A):
  if A = {b}: return b
  else:
    pick pivot b ∈_R A          (uniformly random)
    S< = { a ∈ A : a < b }
    S> = { a ∈ A : a > b }
    return ( RQS(S<), b, RQS(S>) )
```

Assume the elements are distinct. The **cost = number of comparisons**, which depends entirely on the random pivot choices:

- always picking an **extreme** element: $T(n) = \sum_{2 \le i \le n} i = O(n^2)$;
- always picking the **median**: $T(n) \le 2T(n/2) + n - 1 = O(n \log n)$;
- even a *lopsided but balanced-ish* split works:
  $T(n) \le T(n/8) + T(7n/8) + n - 1 = O(n \log n)$.

The point: any split that is "not too extreme" already gives $O(n \log n)$. With a random pivot, balanced-enough splits are the typical case. Let's prove the **expected** cost is $O(n \log n)$.

Analysis via indicator variables

Let $s_1 < s_2 < \cdots < s_n$ be the sorted output. For a particular computation C and a pair $i < j$, define the **indicator**

$$X_{ij}(C) = \begin{cases} 1 & \text{if } s_i \text{ and } s_j \text{ are compared during } C, \\ 0 & \text{otherwise.} \end{cases}$$

Total comparisons: $T(C) = \sum_{1 \le i \le n-1} \sum_{j > i} X_{ij}(C)$. By linearity of expectation,

$$E[T] = \sum_{i<j} E[X_{ij}] = \sum_{i<j} p_{ij}, \qquad p_{ij} := \Pr[s_i, s_j \text{ are compared}].$$

(The crucial move: $E[X_{ij}] = 1 \cdot p_{ij} + 0 \cdot (1 - p_{ij}) = p_{ij}$. Linearity lets us add up these probabilities even though the X_{ij} are highly dependent.)

Computing p_{ij} — the key combinatorial insight

Consider the set $\{s_i, s_{i+1}, \ldots, s_j\}$ of $j - i + 1$ consecutive elements.

> s_i and s_j are compared **if and only if one of them is the first pivot chosen from this whole set.**

Why: if some middle element r with $s_i < r < s_j$ is picked first, it separates s_i and s_j into different sub-arrays, and they never meet again. They are compared only if s_i or s_j itself is the first pivot among the $j - i + 1$ candidates. Since the first pivot in that set is uniform over all $j - i + 1$ of them, exactly 2 of those choices (s_i or s_j) cause a comparison:

$$p_{ij} = \frac{2}{j - i + 1}.$$

Summing up

$$E[T] = \sum_{i=1}^{n-1} \sum_{j>i} \frac{2}{j-i+1} = \sum_{i=1}^{n-1} \sum_{2 \le k \le n-i+1} \frac{2}{k} \;\le\; 2nH_n \approx 2n \ln n.$$

(Here $k = j - i + 1$ ranges over $2, 3, \ldots$, and $H_n = \sum_{k=1}^{n} 1/k \approx \ln n$ is the harmonic number.)

> **Punchline.** Expected $\approx 2n \ln n$ comparisons. The art is: (1) charge cost to *pairs*, (2) use linearity of expectation to ignore dependencies, (3) reduce each pair's probability to a clean "who is picked first" question. This indicator-plus-linearity pattern is one of the most reused tools in the whole course.

---

5. Freivalds' test: is $AB = C$?

The problem

Given three $n \times n$ matrices A, B, C, decide whether $AB = C$. Recomputing AB costs $O(n^{2.37\cdots})$ (or naively $O(n^3)$). **Freivalds (1977)** *verifies* a claimed product in only $O(n^2)$.

The algorithm

1. Pick a random vector $x \in_R \{0,1\}^n$.
2. Check whether $A(Bx) = Cx$.

Why $O(n^2)$: never form the matrix product. Compute Bx first (a matrix–vector product, $O(n^2)$), then $A(Bx)$ ($O(n^2)$), and Cx ($O(n^2)$). Three cheap matrix–vector multiplies.

Correctness

- If $AB = C$: then $ABx = Cx$ for every x — **always correct**.
- If $AB \neq C$: the test wrongly says "equal" only if $(AB - C)x = 0$ for the random x we happened to pick.

Error probability (the $\{0,1\}$ version)

Let $D = AB - C \neq 0$, so some entry $D_{ij} \neq 0$. Look at coordinate i of $y = Dx$:

$$y_i = \sum_k D_{ik}\, x_k = D_{ij}\, x_j + \sum_{k \neq j} D_{ik}\, x_k.$$

Principle of deferred decisions: fix all coordinates x_k ($k \neq j$) first. Then $y_i = 0$ forces exactly one value of x_j:

$$x_j = -\frac{\sum_{k \neq j} D_{ik}\, x_k}{D_{ij}}.$$

There is at most one such value, and x_j is a fair coin over $\{0,1\}$, so it equals that value with probability $\leq 1/2$:

$$\Pr[\text{error}] \leq \Pr[y_i = 0] \leq \frac{1}{2}.$$

Repeating with independent random x drives the error to 2^{-t}.

Real-valued version (Vandermonde / polynomial view)

Instead of a $0/1$ vector, pick a random real $r \in_R \mathbb{R}$ and set $x = (1, r, r^2, \ldots, r^{n-1})^T$. Then coordinate i becomes a **polynomial in** r:

$$y_i = p_i(r) = \sum_{k=0}^{n-1} D_{ik}\, r^k, \qquad \deg p_i \leq n-1.$$

If $D \neq 0$, some p_i is a nonzero polynomial of degree $\leq n-1$, hence has at most $n-1$ roots. So

$$\Pr[\text{error}] \leq \Pr[r \text{ is a root of } p_i] = \frac{n-1}{|R|},$$

where we draw r from a finite set $R \subseteq \mathbb{R}$. This is a baby case of the **Schwartz–Zippel lemma**.

> **Punchline.** Verifying is cheaper than computing. A single random vector "probes" the matrix difference, and a nonzero difference is almost surely exposed. This is the prototypical **fingerprinting** of a linear-algebra identity.

---

6. "Derandomizing" $AB = C$ for integer matrices

Can we remove the randomness entirely? For **integer-coefficient** matrices, yes — by *choosing one cleverly large evaluation point* instead of a random one.

Cauchy's root bound (1829)

> **Theorem (Cauchy).** Let $P(x) = a_k x^k + \cdots + a_1 x + a_0$ be a real polynomial. If x is a root of P, then $|x| < 1 + \frac{A}{|a_k|}, \qquad A = \max_{0 \leq i \leq k} |a_i|.$

In words: *all roots of a polynomial live inside a disk whose radius is controlled by its coefficients.* So if we evaluate at a point **bigger than this bound**, we are guaranteed not to be sitting on a root — unless the polynomial is identically zero.

The derandomized check

The row polynomials $p_i(r) = \sum_k D_{ik} r^k$ of $D = AB - C$ have integer coefficients we can bound. If every entry of A, B, C is bounded by

$$c_{\max} = \max\{|a_{ij}|, |b_{ij}|, |c_{ij}|\},$$

then each entry of $D = AB - C$ is at most $n\, c_{\max}^2 + c_{\max}$ in absolute value (sum of n products, each $\le c_{\max}^2$, plus one subtracted entry $\le c_{\max}$). Plugging into Cauchy's bound, any real root has magnitude $< 1 + n c_{\max}^2 + c_{\max}$. So pick

$$\alpha = n\, c_{\max}^2 + c_{\max} + 1, \qquad r \leftarrow \alpha, \qquad x = (1, r, \ldots, r^{n-1})^T,$$

and check $ABx \stackrel{?}{=} Cx$ **deterministically**. Because α exceeds every possible root, the only way $Dx = 0$ is $D = 0$.

- **Complexity:** $O(n^2)$ algebraic operations.

> **Punchline.** Randomness was only used to "dodge the roots". Once we can *bound where the roots are*, a single well-chosen point dodges them for free. This is the spirit of **derandomization** — the topic of the last block of the course.

---

7. Nondeterministic matrix multiplication

Now turn the verifier into a way to *certify a guessed product* — i.e. put matrix multiplication into a nondeterministic setting.

Over $\mathbb{Q}$

- **Guess** the result C.
- Compute α as above and $x = (1, \alpha, \ldots, \alpha^{n-1})^T$.
- **Verify** $ABx = Cx$ deterministically (single point suffices, by §6).

Over $\mathbb{R}$

- **Guess** C.
- Take distinct reals $r_1, \ldots, r_n$ and build $x_i = (1, r_i, \ldots, r_i^{n-1})^T$.
- Then $AB = C \iff ABx_i = Cx_i$ for all $i = 1, \ldots, n$.

Complexity: the vectors x_i are Vandermonde, so Bx_i and Cx_i can be batched by divide-and-conquer in $O(n^2 \log^2 n)$, or with the **FFT** in $O(n^2 \log n)$.

> **Caveat (`!!`):** once you compute $A(Bx)$, the inner result Bx no longer has the nice Vandermonde "good form", so you cannot recursively exploit the structure on the outer multiply. The trick has to be applied carefully.

The bilinear form trick

Fold both sides into a single quadratic form. Since $D = AB - C$:

$$D = 0 \iff x^T D x = 0 \text{ for enough } x \iff (x^T A)(Bx) \overset{?}{=} x^T C x.$$

Corollary. Let D be a real $n \times n$ matrix, let $r_1, \dots, r_{2n-1}$ be distinct reals, and $x_i = (1, r_i, \dots, r_i^{n-1})^T$. Then

$$D = 0 \iff \forall i : \; x_i^T D x_i = 0.$$

Why $2n-1$ points? The scalar $x_i^T D x_i = \sum_{a,b} D_{ab}\, r_i^{a+b}$ is a polynomial in r_i of degree at most $2n-2$. If $D \neq 0$ this polynomial is nonzero, hence has at most $2n-2$ roots — so checking $2n-1$ distinct points is enough to be sure.

> **Theorem.** Multiplication of real matrices can be realized on an **N-RealRAM** (nondeterministic real RAM) using $O(n^2 \log n)$ (resp. $O(n^2 \log^2 n)$) algebraic operations.

> **Punchline.** "Guess the answer, then verify it cheaply" — the verifier from §5–6 is exactly the certificate-checker that places the problem in a nondeterministic class.

---

8. Nondeterministic multiplication over $\mathbb{Z}$ and $\mathbb{Z}_p$

Why move to modular arithmetic

The evaluation points blow up: r^{n-1} with $r \approx 2n-1$ is a number $(2n-1)^{n-1}$, i.e. about $O(n \log n)$ **bits** long. Arithmetic on such giant integers is expensive. **Fix:** work modulo a prime p in the field $\mathbb{Z}_p$, keeping numbers small.

Two subtleties arise:

- **Cauchy's bound no longer holds** in $\mathbb{Z}_p$ — there is no notion of "magnitude" to bound roots.

- A value that is *not* a root over $\mathbb{Z}$ might become a root over $\mathbb{Z}_p$. Example: x^2+1 has no integer root, but over $\mathbb{Z}_5$ it has roots 2 and 3.
- **However**, a polynomial of degree n still has **at most n roots** in any field $\mathbb{Z}_p$. That is all we need.

> **Lemma.** Let D be an integer $n \times n$ matrix with $\max |d_{ij}| \le \delta$. Let $\mathbb{Z}_p$ be a field with $p > \max\{\delta,\, 2n-1\}$. Let $r_1, \ldots, r_{2n-1} < p$ be distinct, and $x_i = (1, r_i, \ldots, r_i^{n-1})^T$. Then $D = 0 \ (\text{in } \mathbb{Z}) \iff \forall i:\ x_i^T D x_i \equiv 0 \ (\text{in } \mathbb{Z}_p)$.

Choosing p larger than the entries (δ) *and* larger than the degree ($2n-1$) prevents both spurious roots and overflow-collapse, so testing over $\mathbb{Z}_p$ faithfully decides the integer question.

The sign problem when guessing C

When we guess C nondeterministically we must handle signs. Split each matrix into a positive-entry part and a negative-entry part, $A = A^+ + A^-$, $B = B^+ + B^-$:

$$(A^+ + A^-)(B^+ + B^-) = \underbrace{A^+B^+ + A^-B^-}_{C^+} + \underbrace{A^-B^+ + A^+B^-}_{C^-}.$$

This costs **4 multiplications instead of 1**, but each now has consistent signs.

How to find a suitable prime p

- **Guess p.** Bertrand's postulate guarantees one nearby: $\forall k > 1$ $\exists$ prime p with $k < p \le 2k$.
- **Verify p is prime** by guessing a short **Pratt certificate** (1975): verification takes $O(\log^2 p)$ modular multiplications on a unit-cost RAM. (Details in §9.)

Complexity of the whole nondeterministic multiplication:

- unit-cost RAM: $O(n^2 \log^2 n)$;
- log-cost RAM: $O(n^2 \log n \cdot M(\log p))$, where $M(\log p)$ is the cost of multiplying $\log p$-bit numbers.

---

9. Nondeterministic primality verification (Pratt certificates)

The previous section needed to *certify* that a number p is prime. Pratt's theorem says this can always be done with a short, checkable proof.

> **Theorem (Pratt).** Every prime has a **short certificate**.

The idea

If p is prime, then $\mathbb{Z}_p^*$ is cyclic and has a **generator** x of order $p-1$. Being a generator (a *primitive root*) is exactly the witness of primality, and it can be checked by a few modular exponentiations.

The axiomatic proof system

Triples (p, x, a) mean "so far we have verified that the order of x is a multiple of a":

- **Axiom A:** $(p, x, 1)$ is an axiom.
- **Rule R_1:** $(p, x, a),\ q \ \mapsto\ (p, x, qa)$ provided $q \mid (p-1)$ and $x^{(p-1)/q} \not\equiv 1 \pmod{p}$.
- **Rule R_2:** $(p, x, p-1) \ \mapsto\ p$ provided $x^{p-1} \equiv 1 \pmod{p}$.

We prove three things:

1. **p is prime $\iff$ p is a theorem of the system.**
2. The proof of primality of p has $O(\log p)$ lines.
3. Verifying a proof on a unit-cost RAM costs $O(\log^3 p \cdot \log\log p)$ operations.

(1 ⇒) If p is prime, then p is a theorem

Let x be a generator of $\mathbb{Z}_p^*$ and $p-1 = q_1 q_2 \cdots q_k$ the factorization into primes. Build the proof **by induction**:

- Bases $2, 3$ are prime (handled directly).
- Let D_i be the proof that each factor q_i is prime.
- Construction:
  - A: $(p, x, 1)$
  - the sub-proofs $D_1, \ldots, D_k$
  - apply R_1 k times:
    $(p, x, 1) \mapsto (p, x, q_1) \mapsto (p, x, q_1 q_2) \mapsto \cdots \mapsto (p, x, q_1 \cdots q_k) = (p, x, p-1)$
  - R_2: $(p, x, p-1) \mapsto p$, since $x^{p-1} \equiv 1 \pmod{p}$.

Each R_1 step confirms $x^{(p-1)/q_i} \neq 1$, certifying that the order of x is divisible by q_i; together they force the order to be exactly $p-1$, so x really is a generator.

(1 ⇐) If p is a theorem, then p is prime

By contradiction. Suppose p is *not* prime but is a theorem. Then $\mathbb{Z}_p$ has **no generator**. The last proof line must be $(p, x, p-1) \mapsto p$ with $x^{p-1} \equiv 1 \pmod{p}$. Since x is not a generator, $x^j \equiv 1 \pmod{p}$ for some $j < p-1$ with $j \mid (p-1)$. But to reach $(p, x, p-1)$ from $(p, x, 1)$ the proof had to apply R_1, which requires $x^{(p-1)/q_i} \neq 1$ at each step — contradicting that the true order j is a proper divisor of $p-1$. Hence p must be prime.

(2) The proof has at most $\lceil 4 \log p \rceil$ lines

By induction.

- **Base** $p = 2, 3$: 5 lines suffice.
- **Hypothesis:** the proof of any prime q has $\leq \lfloor 4 \log q \rfloor - 4$ lines.
- The construction in (1) for p with $p - 1 = q_1 \cdots q_k$ uses: A + R_2 + (proofs of all q_i) + $k \times R_1$. Counting lines:

$$2 + k + \sum_{1 \leq u \leq k} (\lfloor 4 \log q_u \rfloor - 4) < 2 + k + 4 \lfloor \log q_1 + \cdots + \log q_k \rfloor - 4k < \lfloor 4 \log p \rfloor - 4.$$

(using $\log q_1 + \cdots + \log q_k = \log(q_1 \cdots q_k) = \log(p-1) < \log p$). Adding the 5 base lines for the $2, 3$ cases:

$$\text{length} < \lfloor 4 \log p \rfloor - 4 + 5 \leq \lceil 4 \log p \rceil.$$

So the certificate is **logarithmic** in the size of p — genuinely short.

(3) Verifying the proof is cheap

The only nontrivial operation is computing $x^b \bmod p$, done deterministically by **repeated squaring**:

- init: $(x, b, 1)$;
- b even: $x^b = (x^2)^{b/2}$, i.e. $(u, v, w) \vdash (u^2, v/2, w)$;
- b odd: $x^b = x \cdot x^{b-1}$, i.e. $(u, v, w) \vdash (u, v-1, uw)$.

Every operation keeps numbers to $O(\log p)$ bits, and there are $O(\log^2 p)$ multiplications/squarings. Multiplying $\log p$-bit numbers costs $O(\log p \log\log p)$ via the **Schönhage–Strassen** algorithm. Total: $O(\log^3 p \cdot \log\log p)$ operations.

Corollary

> $$\textbf{Primes} \in \textbf{NP} \cap \textbf{coNP}.$$

- $\textbf{Primes} \in \textbf{NP}$: a Pratt certificate (the generator x + recursive factor proofs) is a short, polynomially-checkable witness of primality.
- $\textbf{Primes} \in \textbf{coNP}$: a *factor* is a short witness of compositeness.

> **Historical note.** This $\textbf{NP} \cap \textbf{coNP}$ result long predates the unconditional **AKS** algorithm (2002), which finally showed $\textbf{Primes} \in \textbf{P}$ deterministically. But the randomized tests (Miller–Rabin, Solovay–Strassen) remain the practical choice — another instance of randomness winning on

> efficiency.

Recurring themes to carry into the rest of the course

Theme	Where it appeared
Fingerprinting (test a big identity via a small random/modular projection)	§1 primes, §5–8 matrix product
One-sided error (`yes` -instances never lie; `no` -instances err with small prob.)	§1, §5 — these are RP/coRP-style algorithms
Probability amplification by independent repetition	§1, §3 min-cut, §5 Freivalds
Indicator variables + linearity of expectation	§4 QuickSort
Polynomial / Schwartz–Zippel ("a nonzero low-degree polynomial has few roots")	§5–8
Derandomization (bound the bad set, then pick deterministically)	§6 Cauchy, §7
Guess-and-verify / short certificates (nondeterminism)	§7–9, Pratt

The single sentence that ties them all together:

> **A small amount of randomness lets us *probe* a large object so that any "flaw" is almost surely exposed — and when we can describe *where* the flaws can hide, we can often remove the randomness altogether.**