Lecture 4 (part 1) — Eliminating the Adversary

> Course **2-INF-135/15 Pravdepodobnostné algoritmy**, LS 2025/26. Source slides: `04_metody1.pdf` (12 pages) + start of `04_metody23.pdf` (slides 1–4).

Where we are: the methods block

Lectures 1–3 built individual algorithms and then the *complexity-class* scaffolding (RP, BPP, ZPP, …). Now begins the **methods** block — the recurring *tricks* that keep producing randomized algorithms. The whole catalogue, in one breath:

Method (Slovak)	Idea	Examples
Eliminating the adversary (eliminácia protihráča)	Randomly choose the strategy/element that drives the computation, so the worst case becomes unlikely → focus on **expected** cost.	DB equality, QuickSort, universal hashing, **online problems**
Fingerprints (odtlačky)	Replace comparing complex objects by comparing simpler *fingerprints*.	databases, hashing, Freivalds
Error reduction by repetition (znižovanie chyby)	Amplify confidence — but sometimes it is smarter *not* to repeat the whole computation.	(recurring throughout)
Witnesses (svedkovia)	Randomly hunt for a *witness* that confirms/refutes a property.	primality — a divisor witnesses compositeness
Optimization + random rounding	Relax a discrete problem ($\mathbb{Z}, \mathbb{N}$) into $\mathbb{R}$, solve efficiently there, round back.	LP-rounding
Random walks (náhodné prechádzky)	Find an object with a property by walking randomly through the object space.	2-SAT walk, s-t connectivity

This part of the lecture is entirely the **first** method, applied to its most striking arena: **online problems**.

---

1. The core idea — eliminating the adversary

> By **randomly choosing** the strategy (or the element) that steers a computation, the worst case stops being a *fixed* target the adversary can aim at. We trade *worst-case* guarantees for *expected-case* ones — and that trade is often a landslide win.

The mental model is a **game**: you (the algorithm) move, an **adversary** picks the input to hurt you most. A *deterministic* algorithm is fully predictable, so the adversary can compute your single worst input in advance and hand it to you. A *randomized* algorithm makes a coin-flip the adversary cannot see — so it cannot tune one input to be bad for all your possible coin outcomes. The adversary is **eliminated** not by being beaten head-on, but by being denied a fixed target.

Online problems — the natural battlefield

In an **online** problem the input is **not given all at once**. It dribbles in:

$$x_1, x_2, x_3, \dots$$

and you must irrevocably react to x_t knowing only $x_1, \dots, x_t$ — the past, never the future. The functions $f_1(x_1), f_2(x_1, x_2), \dots$ model "your decision so far." This is exactly where adversaries thrive: they choose the **next** input *after* watching what you just did.

Competitive ratio — grading an online algorithm

We compare an online algorithm A to the **optimal offline** solution $\mathrm{OPT}(x)$ — the cost an all-knowing solver, who sees the *whole* input in advance, would pay. The **competitive ratio** on input x is the worse of the two directions:

$$\mathrm{comp}_A(x) = \max\left\{ \frac{A(x)}{\mathrm{OPT}(x)},\ \frac{\mathrm{OPT}(x)}{A(x)} \right\} \ \geq\ 1.$$

- For a **minimization** problem (paging, scheduling: smaller cost is better) this is just $\dfrac{A(x)}{\mathrm{OPT}(x)} \geq 1$ — how many times worse than optimal you are.
- For a **maximization** problem (selection: bigger value is better) people instead say A is **α-competitive** if $E[A(x)] \geq \alpha \cdot \mathrm{OPT}(x)$ with $\alpha \leq 1$ — how large a *fraction* of optimum you secure. *(Watch this flip of convention; both appear in this very lecture.)*

> **δ-hard problem** (δ-ťažký): a problem with **no** d-competitive algorithm for any $d < \delta$. In words, δ is a *barrier* — a proven floor on how good ANY online algorithm can be. Proving δ-hardness means: *for every algorithm there is an input that drags its ratio up to* δ. That "for every A, there

> exists a bad I" is the adversary talking.

Two flavours of "probabilistic approximation"

For a randomized online algorithm A we ask it to **always** output a feasible solution, and grade the *quality* either in expectation or with a guarantee:

- A is **probabilistic $E[\delta]$-approximation** if
  $\Pr[A(x) \in \mathrm{Sol}(x)] = 1 \quad \text{and} \quad E[\mathrm{comp}_A(x)] \leq \delta \quad \forall x.$ *(Expected ratio is good.)*
- A is **probabilistic δ-approximation** if
  $\Pr[A(x) \in \mathrm{Sol}(x)] = 1 \quad \text{and} \quad \Pr[\mathrm{comp}_A(x) \leq \delta] \geq \frac{1}{2} \quad \forall x.$ *(Ratio is good at least half the time — then amplify.)*

Both demand a **correct** (feasible) answer with probability 1; only the *quality* is random. This is the online analogue of a Las Vegas algorithm.

---

2. Warm-up: Paging is k-hard

The problem

A **cache** (cash) of size k holds pages; everything else lives in main memory. Page requests arrive online. If a requested page is **not** in the cache, that is a **fault** (chyba) — you must bring it in and **evict** one of the k residents. We count the number of swaps (faults).

> **Claim. Paging is k-hard:** no deterministic online algorithm is better than k-competitive.

The adversary's construction

There are $k+1$ pages in play but only k cache slots — so at every moment **exactly one** page is missing. The adversary's rule is brutally simple:

> **Always request the page the algorithm just evicted** (equivalently, the one page currently *not* in cache).

Then the online algorithm faults on **every single request**. Walk it through with $k = 4$, pages $\{1, 2, 3, 4\}$ in cache and page 5 entering:

request	online cache (faults every step)	optimal offline (1 fault per block)
start	`{1,2,3,4}`	`{1,2,3,4}`
5	fault → evict 3 → `{1,2,5,4}`	fault → evict 4 → `{1,2,3,5}`
3	fault → evict 1 → `{3,2,5,4}`	hit ✓
1	fault → evict 2 → `{3,1,5,4}`	hit ✓
2	fault → evict … → `{3,1,5,?}`	hit ✓

The online algorithm pays $k = 4$ faults; the optimal **offline** algorithm, seeing the future, evicts page 4 first (it is the one page never requested again in this block), and so pays only **1** fault. Hence

$$\text{comp}(x) = \frac{k}{1} = k.$$

The deep point

The offline optimum is **Bélády's rule**: *evict the page whose next use is furthest in the future*. The online algorithm cannot run it because it does not know the future — and the adversary weaponizes exactly that ignorance. The gap is *fundamentally about information*, not cleverness.

> **Punchline.** Determinism online is hopeless here: a k-fold gap, and k is the whole cache. *(Randomization — the "marking algorithm" — later cuts this to $O(\log k)$; that is the payoff of eliminating the adversary, though it is beyond this slide.)*

---

3. The main event: two-job scheduling

This example is the heart of the lecture, because it shows the **whole arc**: a deterministic lower bound, a deterministic upper bound, a hardness theorem, and then a randomized algorithm that *smashes through* the deterministic barrier.

Setup

- m types of machines $A_1, \ldots, A_m$.
- A **job** = a permutation of machine indices: the *order* in which it wants to visit the machines. Each visit takes **1 time unit**.
- For simplicity the input is **2 jobs**: $\alpha = (1, 2, \ldots, m), \qquad \beta = (i_1, i_2, \ldots, i_m)$. Job α visits machines in order $1, 2, \ldots, m$; job β in the order β.

Two jobs **collide** when they want the **same machine at the same time** — one must wait. A machine serves one job at a time.

The schedule as a lattice path

Picture an $m \times m$ grid. Being at cell (i, j) means "α has finished i steps, β has finished j steps." A schedule S is a monotone path from $(0,0)$ to (m,m) built from three moves:

- $\rightarrow$ advance α only (β waits this tick),
- $\downarrow$ advance β only (α waits),
- $\searrow$ advance **both** in one tick — allowed only when their next machines **differ**, $\alpha(i{+}1) \neq \beta(j{+}1)$ (no collision, so they truly run in parallel).

A **collision cell** is one where $\alpha(i{+}1) = \beta(j{+}1)$: there you *cannot* go diagonal, you must detour $\rightarrow$ then $\downarrow$ (two ticks instead of one).

> **Cost.** The time is the number of ticks. With g diagonal steps you spend $\mathbf{time} = 2m - g = m + (m - g)$. Writing $\mathbf{del}(S) = m - g$ for the number of collisions your path was forced through, $\boxed{\mathbf{time}(S) = m + \mathbf{del}(S).}$ The ideal is m (all diagonal, perfect parallelism); every unavoidable collision adds 1.

Crucial counting fact. Each machine v appears once in α and once in β, so there is **exactly one** collision cell per machine — m **collision cells total**, and each one lies on a single grid diagonal. Remember this; it powers the upper bound.

3a. Deterministic lower bound: $\mathrm{time} \geq m + \frac{m}{8}$

> $\forall A \; \exists I : \; \mathbf{time}(A(I)) \geq m + \frac{m}{8}.$

The adversary builds β **online, reacting to the algorithm's moves**, to force a collision **every other step**. Sketch of the rule (collisions (i, j) with $i, j \leq m/2$):

- start $\beta(1) = \alpha(1) = 1$ (immediate collision);
- whenever A resolves a collision by going $\rightarrow$, set $\beta(i{+}1) := j + 2$;
- whenever A goes $\downarrow$, set $\beta(i{+}1) := m - k$, where k = number of $\downarrow$'s so far.

Whichever way A dodges, the adversary plants the next collision right in its path. Over the first $\sim m/2$ steps each forces an expected delay of $\sim \frac{1}{2}$, and only on $\sim \frac{1}{2}$ of the grid does the trap apply, giving a guaranteed delay

$$\mathbf{del} \;\geq\; \frac{m}{2} \cdot \frac{1}{2} \cdot \frac{1}{2} \;=\; \frac{m}{8}.$$

So *every* deterministic online algorithm can be pushed to $\mathbf{time} \geq m + m/8$.

3b. Deterministic upper bound: $\mathrm{OPT} \leq m + \sqrt{m}$

Now the *offline* side — how good is the **best** schedule (it may inspect all of β)? Consider a family of **shifted-diagonal** strategies A_j, for $j \in \{-\sqrt{m}, \ldots, \sqrt{m}\}$ ($2\sqrt{m}+1$ of them):

$$A_j = \begin{cases} (\rightarrow^{|j|})(\searrow^{m-|j|})(\downarrow^{|j|}), & j \leq 0, \\ (\downarrow^{j})(\searrow^{m-j})(\rightarrow^{j}), & j > 0. \end{cases}$$

A_0 is the pure main diagonal; A_j slides the diagonal sideways by j to **dodge collisions sitting on the main diagonal**. Its cost splits into a fixed **offset** $|j|$ (stepping off the diagonal and back) plus the collisions still on its shifted track:

$$\mathbf{time}(A_j) = m + |j| + \mathbf{del}(A_j).$$

The averaging argument (this is the elegant part). Each of the m collision cells lies on **one** diagonal, so it can burden **at most one** strategy A_j. Therefore

$$\sum_{j=-\sqrt{m}}^{\sqrt{m}} \mathbf{del}(A_j) \;\leq\; m.$$

Add up the total excess over the ideal m across all strategies:

$$\sum_{j=-\sqrt{m}}^{\sqrt{m}} \big(|j| + \mathbf{del}(A_j)\big) \;\leq\; \underbrace{(m+\sqrt{m})}_{\sum |j|} + \underbrace{m}_{\sum \mathbf{del}} \;=\; 2m + \sqrt{m}.$$

So the **average** excess over the $2\sqrt{m}+1$ strategies is

$$\frac{2m+\sqrt{m}}{2\sqrt{m}+1} = \frac{\sqrt{m}\,(2\sqrt{m}+1)}{2\sqrt{m}+1} = \sqrt{m}.$$

Since *some strategy is no worse than the average*, there exists one with $\mathbf{time} \leq m + \sqrt{m}$. The offline optimum can only be better:

$$\boxed{\mathrm{OPT}(x) \leq m + \sqrt{m}.}$$

> **The averaging trick** ("there exists one no worse than the average") is the probabilistic method in miniature — and it is the bridge to the randomized algorithm below.

3c. Hardness: the problem is $(9/8-\varepsilon)$-hard

Combine the two bounds. The lower bound gives a bad instance with $\mathrm{cost}_A(I) \geq m + m/8 = \frac{9}{8}m$; the upper bound gives $\mathrm{OPT} \leq m + \sqrt{m}$. Hence

$$\mathrm{comp}_A(I) = \frac{\mathrm{cost}_A(I)}{\mathrm{OPT}(I)} \geq \frac{\frac{9}{8}m}{m+\sqrt{m}} = \frac{9}{8} \cdot \frac{1}{1+1/\sqrt{m}} = \frac{9}{8}\left(1 - \frac{1}{\sqrt{m}+1}\right) \xrightarrow{m\to\infty} \frac{9}{8}.$$

> **No deterministic online algorithm beats $9/8-\varepsilon$.** A hard, *fixed* floor — because a deterministic algorithm is a fixed target.

3d. The randomized algorithm DIAG — through the barrier

> **DIAG:** pick $i \in_R \{-\sqrt{m}, \ldots, \sqrt{m}\}$ uniformly, then run strategy A_i.

That is the entire algorithm: **a random diagonal.** The analysis is just the averaging argument re-read as an expectation. Because the m collisions are spread over the $2\sqrt{m}+1$ strategies and the offsets average out,

$$E[\#\text{delays of } A_i] \;\leq\; \lceil\sqrt{m}\rceil + \tfrac{1}{2}, \qquad E[\text{time}] \;\leq\; m + \lceil\sqrt{m}\rceil + \tfrac{1}{2}.$$

Using the trivial $\mathrm{OPT} \geq m$,

$$E[\mathrm{comp}_{\mathrm{DIAG}}] \;\leq\; \frac{m + \lceil\sqrt{m}\rceil + \frac{1}{2}}{m} \;=\; 1 + \frac{1}{\sqrt{m}} + \frac{1}{2m} \;\xrightarrow{m\to\infty}\; 1.$$

> **Punchline.** Deterministic is stuck at $\geq 9/8$; randomized **DIAG** drives the expected ratio all the way to 1. *Same strategies, the only change is choosing which diagonal at random.* The adversary built its trap assuming it knew your diagonal — randomizing the diagonal **eliminates** that knowledge, and the forced delay collapses from $m/8$ down to $\sqrt{m}$. This is "eliminating the adversary" in its purest form.

---

4. The selection problem and Yao's minimax principle

The problem (the secretary problem in disguise)

Values $v_1, \dots, v_n$ arrive online (in random order). When you see a value you must **immediately** decide to take it or pass forever; you want to end up holding the **maximum**. How well can you do?

Result 1 — determinism is worthless: no deterministic algorithm beats 0-competitive. Suppose some deterministic A were α-competitive, $\alpha > 0$ (i.e. $E[v(A)] \geq \alpha \max_i v_i$). The adversary watches A's reaction to seeing $v_1 = 1$ first:

- **If A takes $v_1 = 1$:** feed input $I = (1,\ \frac{2}{\alpha},\ 0, \dots, 0)$. Then $\mathrm{OPT} = 2/\alpha$ but A holds 1, ratio $= \frac{1}{2/\alpha} = \frac{\alpha}{2} < \alpha$. ✗
- **If A passes v_1:** feed input $II = (1,\ 0, 0, \dots, 0)$. The only good value is gone; A ends with 0, ratio $= 0 < \alpha$. ✗

A deterministic algorithm's decision on seeing "1" is *fixed*, so the adversary picks whichever input punishes that fixed decision. Determinism loses completely.

Result 2 — a trivial randomized algorithm is $\frac{1}{n}$-competitive. Pick a position $x \in_R \{1, \dots, n\}$ and output v_x. You hit the maximum with probability exactly $\frac{1}{n}$. *Can we do better?* — This is the question Yao answers.

Yao's minimax principle

> **Yao (for minimization).** Let A be a random variable over deterministic algorithms $\mathcal{A}$, and X a random variable over inputs $\mathcal{X}$. Then $\max_{x \in \mathcal{X}} E[c(A, x)] \ \geq \ \min_{a \in \mathcal{A}} E[c(a, X)]$.

Read it as a sentence: **the best randomized algorithm's cost on its worst input is at least the best deterministic algorithm's cost against a (well-chosen) random input.** This is the workhorse for **lower bounds on randomized algorithms**, and it makes them *easy*: instead of reasoning about all possible coin-flip distributions, you just

1. **invent one input distribution X** (your choice — make it nasty), and
2. show **every deterministic** algorithm is expensive on average against it.

That number is then a valid lower bound for *every* randomized algorithm.

Proof — two one-line inequalities. With $E[c(A, x)] = \sum_a \Pr[A{=}a]\, c(a, x)$ and $E[c(a, X)] = \sum_x \Pr[X{=}x]\, c(a, x)$:

$$
\begin{aligned}
\max_x E[c(A,x)] &= \max_x \sum_a \Pr[A{=}a]\, c(a,x) \\
&\geq \sum_x \Pr[X{=}x] \sum_a \Pr[A{=}a]\, c(a,x) && \textbf{(a max is} \geq \textbf{any weighted average)} \\
&= \sum_a \Pr[A{=}a] \sum_x \Pr[X{=}x]\, c(a,x) && \textbf{(swap the sums)} \\
&\geq \min_a \sum_x \Pr[X{=}x]\, c(a,x) && \textbf{(a weighted average is} \geq \textbf{the min)} \\
&= \min_a E[c(a,X)].
\end{aligned}
$$

> The two steps are just **"max $\geq$ average"** and **"average $\geq$ min."** That is all Yao is — yet it is exactly von Neumann's minimax / LP duality for the zero-sum game *you vs. the input*. For a **maximization** problem, set $c = -p$ to flip it: $\min_x E[p(A,x)] \;\leq\; \max_a E[p(a,X)]$.

Applying Yao: you cannot select the maximum with probability $> \frac{1}{n}$

Work over values $\{0, 1, \ldots, n\}$. Let $\mathcal{X}$ = all permutations, $\mathcal{A}$ = deterministic online algorithms,

- $s(a,x)$ = the position at which a selects (if it never selects, $s \leftarrow n$),
- $p(a,x) = 1$ iff a selected the maximum.

We want $\min_x E[p(A,x)] \leq \frac{1}{n}$; by Yao (max-form) it suffices to exhibit a distribution X with $\max_a E[p(a,X)] = \frac{1}{n}$. **The hard distribution:**

$$
x(t) := (1, 2, \ldots, t, 0, \ldots, 0), \qquad T \in_R \{1, \ldots, n\}, \qquad X = x(T).
$$

A random-length **increasing prefix**, then zeros. Take any deterministic a and let $s := s(a, x(n))$ be where it selects on the full increasing input $x(n) = (1, \ldots, n)$. For a prefix $t \neq n$:

- if $s \leq t$: the prefixes $x(t)$ and $x(n)$ **agree** up to position s, so a selects the same position s — and that is the maximum of $x(t)$ **iff** $s = t$ (the peak sits at t);
- if $s > t$: by then a is in the all-zeros tail, so it selects a 0 — never the max.

Either way, $p(a, x(t)) = 1 \iff s = t$. Therefore

$$
E[p(a,X)] = \Pr[T = s] = \frac{1}{n} \quad \text{for } \backslash\text{emph}\{\text{every}\} \text{ deterministic } a.
$$

So $\max_a E[p(a,X)] = \frac{1}{n}$, and Yao gives $\min_x E[p(A,x)] \leq \frac{1}{n}$ for every randomized A. Combined with Result 2 (which *achieves* $\frac{1}{n}$), $\frac{1}{n}$ **is tight.**

> **Why the increasing prefix is diabolical.** When the algorithm stands at position i seeing value i, it cannot tell whether the sequence stops here ($t = i$, so this *is* the max) or keeps climbing ($t > i$, more is coming). Every position looks identically tempting — that ambiguity is precisely

what caps it at $1/n$.

(Oral-exam nuance.) This is the *select-the-exact-maximum* version. The classic secretary problem reaches $\approx 1/e$ — but that relies on a **uniformly random arrival order** and **rank comparisons**; Yao's adversary is allowed to choose this particular *non*-uniform monotone distribution, against which no strategy beats $1/n$.

Competitive version: no randomized algorithm is α-competitive for $\alpha > \frac{1}{n} + \varepsilon$

The "probability of hitting the max" bound upgrades to a **competitive-ratio** bound by scaling the values exponentially so that missing the max is nearly worthless.

Suppose A is $(\frac{1}{n} + \varepsilon)$-competitive, $\varepsilon > 0$, and feed it $v_i = M^{x_i}$ with $x_i \in \{0, \dots, n\}$ and $M \gg 1$. Let $v^* = \max_i v_i$. If A misses the top, the best it can hold is the runner-up, $\leq v^*/M$. So

$$E[v(A)] \;\leq\; v^* \cdot \Pr[A \text{ selects } v^*] + \frac{v^*}{M}.$$

Competitiveness says $E[v(A)] \geq (\frac{1}{n} + \varepsilon)\, v^*$. Divide by v^*:

$$\tfrac{1}{n} + \varepsilon \;\leq\; \Pr[A \text{ selects } v^*] + \tfrac{1}{M} \quad\Longrightarrow\quad \Pr[A \text{ selects } v^*] \;\geq\; \tfrac{1}{n} + \varepsilon - \tfrac{1}{M}.$$

Choosing $M = 2/\varepsilon$ gives $\Pr[A \text{ selects } v^*] \geq \frac{1}{n} + \frac{\varepsilon}{2} > \frac{1}{n}$ — contradicting the theorem above. Hence:

> **No randomized online algorithm for selection is α-competitive for $\alpha > \frac{1}{n} + \varepsilon$.** The trivial "pick a random position" algorithm is essentially **optimal**.

> **Punchline of Yao.** A lower bound *over all randomized algorithms* — a quantifier over infinitely many coin-flip distributions — collapses into analyzing **deterministic** algorithms against **one** input distribution *you* design. Hard becomes easy because the design freedom moves to your side of the game.

Recurring themes from this part

Theme	Where it appeared
Eliminate the adversary = deny it a fixed target by randomizing your strategy	DIAG (§3d), random-position selection (§4)
Online ⇒ the cost of not knowing the future	paging vs. Bélády (§2), scheduling lower bound (§3a)
Averaging / probabilistic method ("∃ one no worse than the average")	scheduling upper bound (§3b) → DIAG (§3d)
Deterministic barrier, randomized breakthrough (9/8 → 1)	scheduling (§3)
Yao's minimax (randomized lower bound ⇐ deterministic-vs-random-input)	selection (§4)
Exponential value-scaling to turn "hit the max" into a competitive ratio	selection competitive bound (§4)

The one sentence tying it together:

> **A deterministic online algorithm is a fixed target the adversary aims at; a coin flip the adversary cannot see turns the worst case into a merely unlikely case — and Yao's principle tells us exactly how far that trick can ever go.**