

# Problémy a algoritmy.

Algoritmus. Vlastnosti algoritmu. Analýza efektivity algoritmu.  
Asymptotická složitost.

Tomáš Bayer | bayertom@fsv.cvut.cz

Katedra geomatiky, fakulta stavební ČVUT.

# 1. Plán přednášek

## Syllabus:

- 1 Algoritmy, datové struktury, časová/paměťová složitost (TB).
- 2 Komprese a kompresní algoritmy pro rastrová data (TB).
- 3 Prostorová indexace dat (TB).
- 4 Clusterizační algoritmy (MP).
- 5 Analýza, klasifikace a extrakce objektů z bodových mračen (MP).
- 6 Dekorelace dat, metoda hlavních komponent (LH/MP).
- 7 Filtrace obrazu (LH/MP).
- 8 Obrazová korelace (LH/MP).
- 9 Automatizované rozpoznávání objektů v mapách (JC/ TJ).
- 10 Vybrané grafové algoritmy a jejich implementace (TB).
- 11 Rekonstrukce 3D objektů z bodových mračen (MP/LH).

## Požadavky na zápočet:

Účast na cvičení.

Odevzdání úloh v daných termínech.

## Zkouška:

Dle dosaženého bodového skóre.

## Literatura:

Ke každé části specifická.

## 2. Geoinformatika

Multidisciplinární obor, napříč přírodními, technickými i exaktními vědami.

Syntéza teoretické informatiky, matematiky, počítačové grafiky, výpočetní geometrie, statistiky.

V rámci kurzu diskutovány pouze *vybrané problémy*.

Snaha o vysvětlení podstaty, principu, funkcionality.

Nestačí verbální popis, nutná multioborová syntéza.

Důležité stanovení míry detailu.

### Cíle předmětu:

- Chápání teoretické podstaty geoinformatických problémů.
- Rozvoj logického a abstraktního myšlení.
- Schopnost dekomponovat složité problémy na jednodušší.
- Automatizace vybraných geoinformatických problémů (Python, Matlab).

Rule-based vs AI/ML přístup.

### *Current research problems:*

Raster/vector data, models and storage. Collecting/measuring data. Big data. On-line access and processing. Distributed/parallel computing. Spatial data structures and algorithms. Spatial queries, indexing. Spatial interpolation. Visualization and 3D modelling. Artificial intelligence. Machine learning.

### 3. Použité technologie (cvičení)

Python:

- výpočty: numpy, scipy,
- rastrová data: gdal,
- strojové učení: PyTorch, scikit-learn,
- skriptování: Jupyter notebook.

Matlab + toolboxes:

- Image Processing Toolbox.

GNU/Linux:

- GIS.LAB.

Ostatní:

- Google Eath Engine,
- Javascript,
- github client.

## 4. Problém & geoinformatika

Dynamický rozvoj přírodních/technických věd přináší řadu nových problémů.

Většina nějak řešitelná s využitím knihoven či specializovaného software.

Stávající řešení nemusí být efektivní, problém lze modifikovat.

Poptávka po odbornících:

- schopnost stávající problémy efektivně řešit (\*),
- schopnost hledat řešení nová řešení stávajících problémů (\*\*),
- schopnost hledat řešení nových problémů (\*\*\*)

Zajímají nás problémy, které lze přesně formulovat s využitím matematického aparátu.

Jejich řešení lze automatizovat (např. s využitím počítače).

U řady problémů neexistuje exaktní řešení (kartografie).

Řešení pak založeno na kombinaci *exaktních* a *subjektivních* přístupů.

Snaha omezovat vliv lidského faktoru při zpracování geodat.

Neexistuje univerzální technika vedoucí k nalezení řešení.

Někdy požadováno přesné řešení, jindy pouze přibližné (NP problémy).

V závislosti na typu problému/řešení/vstupní množiny nutné zvolit vhodnou strategii.

## 5. Popis problému (#)

“Problém” lze z pohledu geoinformatiky formalizovat:

*NÁZEV: Slovní popis problému*

*IN: Popis přípustného vstupu (množina vstupních dat).*

*OUT: Popis výsledku, který je pro daný vstup očekáván.*

Musí existovat funkce  $f$  přiřazující vstupním datům požadovaný výstup.

Nalezení řešení problému  $\Rightarrow$  nalezení příslušné funkce  $f$ .

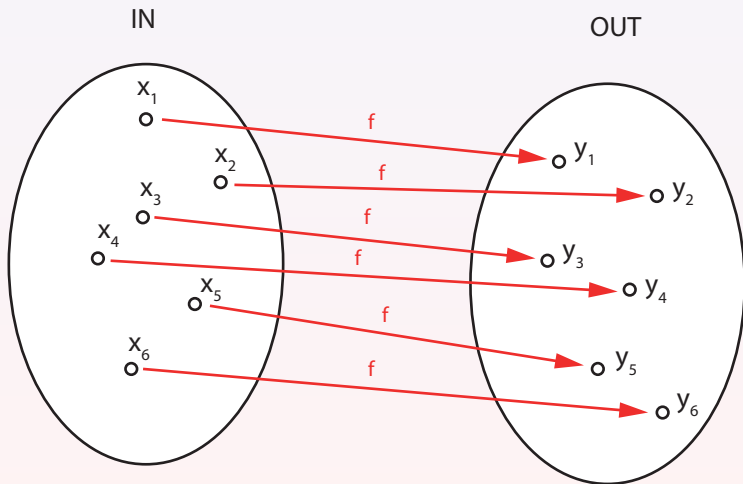
*Každý problém  $P$  určen uspořádanou trojicí  $P(IN, OUT, f)$*

$$f : IN \rightarrow OUT,$$

*IN množina přípustných vstupů, OUT množina očekávaných výstupů,  $f$  přiřazuje každému vstupu očekávaný výstup.*

IN/OUT: Kombinace znaků, celých čísel či přirozených čísel představující kódování.

## 6. Znázornění problému (#)



## 7. Algoritmus a jeho vlastnosti (#)

Obecný předpis sloužící pro řešení zadaného problému.

Představuje posloupnost kroků doplněných jednoznačnými pravidly.

Algoritmicky řešitelný problém:

*Algoritmus  $A$  řeší problém  $P$ , pokud libovolnému vstupu  $x$ ,  $x \in IN$ , přiřazuje v konečném počtu kroků (alespoň jeden) výstup  $y$ ,  $y \in OUT$ , tak, že platí:  $y = f(x)$ .*

Pro zadaný vstup  $x$  může existovat více než jedno řešení  $y$ .

Algoritmus  $A$  by měl nalézt alespoň jedno řešení.

Vlastnosti algoritmu:

- *Determinovanost*: Algoritmus jednoznačný jako celek i v každém dílčím kroku.
- *Rezultativnost*: Vede vždy ke správnému výsledku v konečném počtu kroků.
- *Hromadnost*: Lze použít pro řešení stejné třídy problémů s různým vstupem.
- *Opakovatelnost*: Při opakovaném použití stejného vstupu poskytne tentýž výsledek.
- *Efektivita*: Každý krok algoritmu by měl být efektivní.

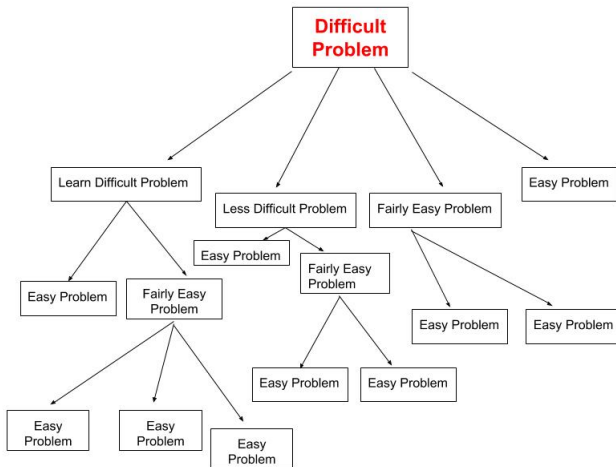
**Strategie řešení:** Rozkládání problému na dílčí podproblémy.

Dekompozice s využitím *Top-Bottom Approach*.

**Strategie implementace:** opačná, *Bottom-up Approach*.



## 8. Top-Down decomposition (#)



## 9. Efektivita algoritmu

Efektivita algoritmu je důležitou *výkonostní charakteristikou*.

Výkonostní charakteristiky algoritmů nelze ignorovat.

Časové a materiální úspory.

Efektivní algoritmus řeší problém s minimálními nároky na HW.

Strategie "Time Is Money".

Optimální využití existujících prostředků".

Pozor na zdánlivě nepodstatné detaily.

### **Nejrychlejší vs. optimální řešení:**

Nejjednodušší zpravidla nejpomalejší, ale implementačně jednoduché.

Nejrychlejší velmi náročné na implementaci (časově kritické aplikace).

V praxi používáno optimální, kompromis (běžné aplikace).

Hodnocení efektivity empiricky/matematickou analýzou.

# 10. Analýza efektivity algoritmu

Efektivita algoritmu funkcí velikosti dat a jejich typu.

*Empirická analýza:*

Srovnáním běhu více algoritmů, různé množiny:

- 1) Random: ověření funkcionality.
- 2) Worst: schopnost zpracovat libovolná data.
- 3) Best: nejlepší případ.

*Exaktní analýza:*

Matematická analýza řad, hledání asymptotických funkcí.

Cíle analýzy algoritmů:

- Porovnání > algoritmů řešící problém: výběr optimálního.
- Odhad výkonnosti algoritmu: lze ho použít pro problém & data?
- Nastavení parametrů algoritmu: co nejefektivnější běh.

**Chybou** soustředění se pouze na výkonnostní charakteristiky.

Implementace a odladění rychlého algoritmu pro libovolný vstup složité.

Lépe pomalejší, ale univerzálnější algoritmus.

# 11. Posuzování složitosti algoritmu

Měřitelné charakteristiky: doba běhu, výpočtu, počet instrukcí, množství paměti...

*Časová složitost algoritmu (Time Complexity):*

Doba zpracování vstupních dat  $D$  algoritmem  $A$  v čase  $T$

$$T = \tau(A(D)).$$

*Paměťová složitost algoritmu (Space Complexity):*

Množství paměti  $M$  pro zpracování vstupních dat  $D$  algoritmem  $A$

$$M = \mu(A(D)).$$

Složitost algoritmu funkcí velikosti a typu vstupních dat.

## A) Velikost vstupu $n$

Složitost jako funkce velikosti vstupu  $n$ .

Algebraický tvar složitý (např.  $4n^3 - 9n^2 + 20n + 27$ ), asymptotický (limitní) odhad (např.  $O(n^3)$ ).

## B) Charakteristika vstupních dat

Pro vstup  $n$  se složitost závisí na hodnotách vstupu (např. náhodná seříděná nebo reverzně seříděná data).

Složitost lze posuzovat empiricky:

- dle nejhoršího možného případu (Worst Case).
- dle průměrné doby běhu (Average Case).
- dle nejlepšího možného případu (Best Case).

# 12. Worst/Best/Average Cases

## Worst Case:

Maximum z dob běhu algoritmu pro všechny vstupy velikosti  $n$

$$T_{\text{WORST}} = \max(T_1(n), T_2(n), \dots, T_n(n)).$$

Nevhodná konfigurace vstupních dat.

O několik řádů vyšší než Average Case.

## Best Case:

Minimum z dob běhu algoritmu pro všechny vstupy velikosti  $n$

$$T_{\text{BEST}} = \min(T_1(n), T_2(n), \dots, T_n(n)).$$

Ideální konfiguraci vstupních dat.

Až o několik řádů lepší než Average Case.

V praxi k takové situaci nemusí dojít (nebo jen ve velmi řídkých případech).

Vlastnosti algoritmu mohou být tímto odhadem zkresleny.

## Average Case:

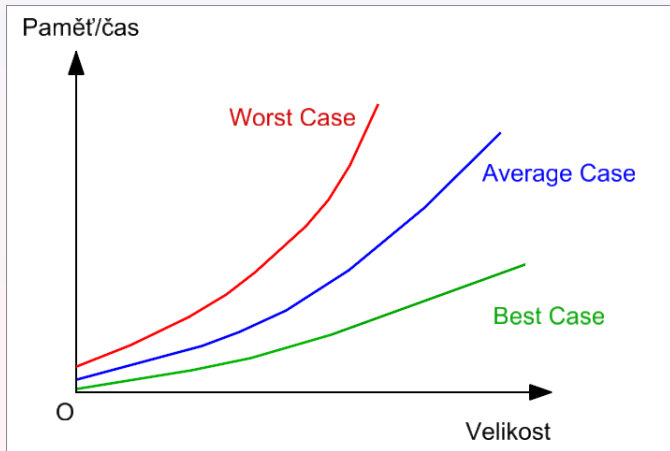
Průměrná doba běhu na (běžných) datech velikosti  $n$

$$T_{\text{AVERAGE}} = E(T_1(n), T_2(n), \dots, T_n(n)).$$

Běžná konfigurace vstupních dat.

Může být o několik řádů lepší/horší než Worst/Best Case .

## 13. Ukázka BC/WC/AC



U dobře navržených algoritmů rozdíl mezi Best Case a Worst Case malý.

Optimalizace algoritmu: snížení rozpětí mezi Best Case a Worst Case.

Pokud Best Case = Worst Case, nelze dále optimalizovat.

# 14. Asymptotická složitost

Empirické srovnání algoritmů nepostačuje, relativní.

Algebraické (přesné) vyjádření složitosti matematicky náročné.

Umíme určit jen u jednoduchých problémů.

Nahrazení algebraické složitosti asymptotickým (limitním) odhadem.

Asymptotická složitost pro  $n \rightarrow \infty$  odpovídá algebraické složitosti.

Zajímá nás chování algoritmu pro velká  $n$ .

Popisuje řád růstu funkce, zjednodušení.

## Zásady:

- 1) Zanedbání konstant (aditivní/multiplikativní).
- 2) Zanedbání funkcí s nízkým růstem řádu ("rychlé" části algoritmu).

Hodnoty  $f_1(n) = 0.1n^2$  pro velká  $n$  podobné  $f_2(n) = 100n^2 + 90n$ .

$$\lim_{n \rightarrow \infty} f_1(n) = \lim_{n \rightarrow \infty} f_2(n).$$

# 15. Zanedbání konstant

**Předpoklad 1:** Zanedbání multiplikativní konstanty  $c$ :

Nechť  $f(n)$  je libovolná funkce a  $c$  libovolná konstanta,  $c > 0$ . Pak funkce  $f(n)$  a  $c \cdot f(n)$  jsou označovány jako (asymptoticky) stejně rychle rostoucí.

$$\lim_{n \rightarrow \infty} c \cdot f(n) = \lim_{n \rightarrow \infty} f(n)$$

**Předpoklad 2:** Zanedbání aditivní konstanty  $d$ :

Nechť  $f(n)$  je libovolná funkce a  $d$  libovolná konstanta,  $d > 0$ . Pak funkce  $f(n)$  a  $f(n) + d$  jsou označovány jako (asymptoticky) stejně rychle rostoucí.

$$\lim_{n \rightarrow \infty} f(n) + d = \lim_{n \rightarrow \infty} f(n)$$

**Důsledek:**

Funkce  $f(n)$  a  $c \cdot f(n) + d$  jsou (asymptoticky) stejně rychle rostoucí.  
Stejný řád růstu.

$$\lim_{n \rightarrow \infty} c \cdot f(n) + d = \lim_{n \rightarrow \infty} f(n).$$

Funkce  $f_1(n) = 0.1n^2$  a  $f_2(n) = 100n^2 + 90n$  patří do stejné třídy (kvadratické)



# 16. Přehled asymptotických odhadů

5 asymptotických odhadů složitosti:

- 1 Asymptotický horní odhad složitosti ostrý  $O(g(N))$ .
- 2 Asymptotický dolní odhad složitosti ostrý  $\Omega(g(N))$ .
- 3 Asymptotický oboustranný odhad složitosti:  $\Theta(g(N))$ .
- 4 Asymptotický horní odhad časové neostrý  $o(g(N))$ .
- 5 Asymptotický dolní odhad časové neostrý  $\omega(g(N))$ .

# 17. Asymptotický horní odhad $O(g(n))$

Ilustruje nejhorší možný případ doby běhu algoritmu.

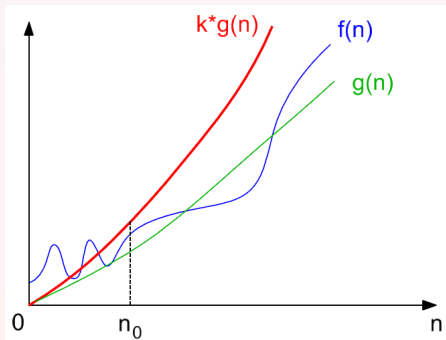
Tzv. O-notace (Big O-notation).

Pro libovolné funkce  $f, g: \mathbb{N} \rightarrow \mathbb{N}$  platí:  $f(n) \in O(g(n)) \Leftrightarrow \exists k \in \mathbb{R}, \exists n_0 \in \mathbb{N}, \forall n > n_0:$

$$f(n) \leq k \cdot g(n).$$

Interpretace:  $f$  roste nejvýše tak rychle jako  $g$ .

Nejčastější typ odhadu, informuje o nejpomalejším možném řešení problému.



Příklad: Platí, že  $20n^2 \in O(n^2)$ ? Řešení:  $20n^2 \leq k \cdot n^2, k \geq 20$

# 18. Asymptotický dolní odhad $\Omega(g(n))$

Ilustruje nejlepší možný případ doby běhu algoritmu.

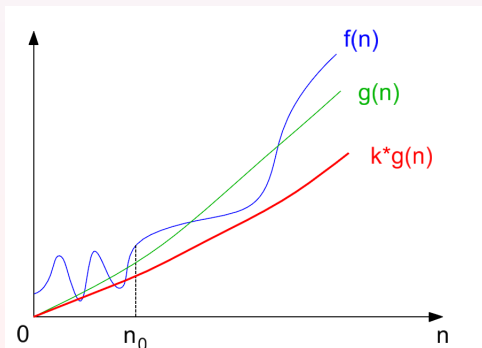
Tzv.  $\Omega$ -notace.

Pro libovolné funkce  $f, g: \mathbb{N} \rightarrow \mathbb{N}$  platí:  $f(n) \in \Omega(g(n)) \Leftrightarrow \exists k \in \mathbb{R}, \exists n_0 \in \mathbb{N}, \forall n > n_0$ :

$$f(n) \geq k \cdot g(n).$$

Interpretace:  $f$  roste nejméně tak rychle jako  $g$ .

Používán méně často, zpravidla nás nezajímá, jak nejrychleji problém vyřešíme.



# 19. Asymptotický oboustranný odhad $\Theta(g(n))$

Popisuje očekávanou složitost (tj. průměrný případ).

Tzv.  $\Theta$ -notace.

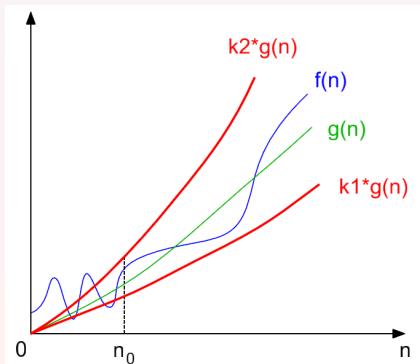
Pro libovolné funkce  $f, g: \mathbb{N} \rightarrow \mathbb{N}$  platí:  $f(n) \in \Theta(g(n)) \Leftrightarrow \exists k_1 \in \mathbb{R}, \exists k_2 \in \mathbb{R}, \exists n_0 \in \mathbb{N}, \forall n > n_0$ :

$$0 \leq k_1 \cdot g(n) \leq f(n) \leq k_2 \cdot g(n).$$

Odhad stejné rychlosti růstu,  $f$  roste stejně rychle jako  $g$ .

Nalezení funkce, která roste asymptoticky stejně rychle až na konstantu.

Odpovídá původní definici “efektivity” algoritmu.



## 20. Ukázka $\Theta(g(N))$ odhadu

Platí, že  $9n^2 - 5n + 6 \in \Theta(n^2)$ ?

Nalezneme vhodné konstanty  $k_1, k_2$

$$k_1 n^2 \leq 9n^2 - 5n + 6 \leq k_2 n^2,$$

např.:  $k_1 = 8, k_2 = 9$ .

Pak  $8n^2 \leq 9n^2 - 5n + 6 \Rightarrow n^2 - 5n + 6 \geq 0, n \geq 3$ .

Pak  $9n^2 - 5n + 6 \leq 9n^2 \Rightarrow 5n \geq 6, n \geq \frac{6}{5}$ .

Protože  $n \geq 3 \wedge n \geq \frac{6}{5} \Rightarrow n \geq 3, n_0 = 3$ .

# 21. Charakteristika algoritmů dle časové složitosti

Složitost	Vyjádření	Charakteristika
Konstantní	1	Konstantní doba běhu programu. Nezávisí na vstupních datech.
Logaritmická	$\log(n)$	Doba běhu se mírně zvětšuje v závislosti na $N$ . Řešení hledáno opakovaným dělením vstupní množiny na menší množiny (hledání v binárním stromu).
Lineární	$n$	Doba běhu programu roste lineárně s $N$ . Zpracováván každý prvek, např. cyklus.
$n \log(n)$	$n \log(n)$	Doba běhu roste téměř lineárně. Opakované dělení vstupního problému na menší problémy, které jsou řešeny nezávisle (Divide and Conquer, např. třídění).
Kvadratická	$n^2$	Doba běhu roste kvadraticky, vhodný pro menší množiny dat. Vnořený cyklus.
Kubická	$n^3$	Doba běhu roste s třetí mocninou, dvojnásobně vnořený cyklus. V praxi snaha nahrazovat algoritmus předchozími dvěma kategoriemi (Greedy algoritmy)
Exponenciální	$2^n$	Exponenciální doba běhu. Použitelné pro množiny do $n=30$ Aplikace v kryptografii.

## 22. Ukázka časové složitosti algoritmů

Vstupní množina  $n = 10, 100, 1000$  prvků.

Počet operací nutných pro řešení problému.

Složitost	$n = 10$	$n = 100$	$n = 1000$
Logaritmická složitost	1	2	3
Lineární složitost	10	100	1000
Kvadratická složitost	100	10000	$1.0 \cdot 10^6$
Kubická složitost	1000	$1.0 \cdot 10^6$	$1.0 \cdot 10^9$
Bikvadratická složitost	10000	$1.0 \cdot 10^8$	$1.0 \cdot 10^{12}$
Exponenciální složitost	1024	$1.3 \cdot 10^{30}$	$1.1 \cdot 10^{301}$
Faktoriální složitost	$3.6 \cdot 10^6$	$9.3 \cdot 10^{157}$	$4.0 \cdot 10^{2567}$

## 23. Ukázka doby běhu algoritmů

Ukázka doby běhu algoritmu pro  $n = 10^9$ .

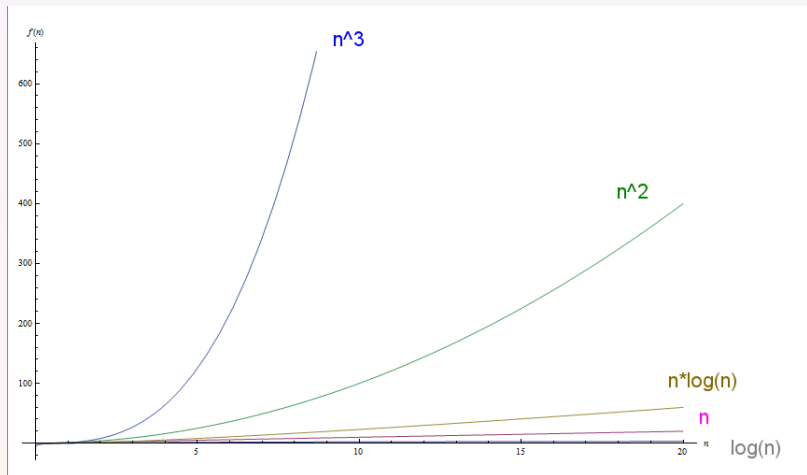
CPU: Počet operací/s  $\frac{10^6}{s}$ ,  $\frac{10^9}{s}$ ,  $\frac{10^{12}}{s}$ .

Složitost	CPU $\frac{10^6}{s}$	CPU $\frac{10^9}{s}$	CPU $\frac{10^{12}}{s}$
Logaritmická složitost	hodiny	vteřiny	okamžitě
Lineární složitost	hodiny	vteřiny	okamžitě
$n \log(n)$	hodiny	vteřiny	okamžitě
Kvadratická složitost	nikdy	roky	týdny
Kubická složitost	nikdy	nikdy	měsíce
Bikvadratická složitost	nikdy	nikdy	roky
Exponenciální složitost	nikdy	nikdy	nikdy
Faktoriální složitost	nikdy	nikdy	nikdy



# 24. Grafické znázornění algoritmů dle časové složitosti

Vstupní množina:  $n \in (0, 20)$



## 25. Analýza složitosti algoritmu, Bubble Sort (1/2)

```
def bubbleSort(x):  
    for i in range(0, len(x)):  
        for j in range(0, len(x) - i - 1):  
            if x[j] > x[j + 1]:  
                temp = x[j];  
                x[j] = x[j + 1];  
                x[j + 1] = temp;
```

Hledáme časovou funkci  $\tau(f(n))$  a asymptotický horní odhad  $O(g(n))$ .

## 26. Analýza složitosti algoritmu, Bubble Sort (2/2)

Závislost na vstupních datech.

Vnější cyklus for, proměnná  $i$ , proběhne  $n$  krát.

Vnitřní cyklus for, proměnná  $j$ , proběhne nejvýše  $n - 1$  krát.

Pro setříděnou posloupnost neprovedeno žádné prohození  $x[i]$  a  $x[i + 1]$ .

Pro neseříděnou posloupnost provedeno  $n - 1$  prohození  $x[i]$  a  $x[i + 1]$ .

Pro běžná data bude počet prohození v intervalu  $(0, n - 1)$ .

Časová funkce

$$\begin{aligned} f(n) &= n - 1 + n - 2 + \dots + 2 + 1 + 0, \\ &= (0 + n - 1) \frac{n - 1}{2}, \\ &= \frac{n^2}{2} - n + \frac{1}{2}. \end{aligned}$$

Zanedbání konstant a nevýznamných členů:

$$g(n) \equiv n^2.$$

Algoritmus náleží do kvadratické třídy  $O(n^2)$ .

## 27. Třída složitosti algoritmů

Zaváděny pro kategorizaci problémů dle jejich složitosti.

Rozčlenění problémů do tříd téže složitosti umožňuje:

- Odhalovat vzájemné podobnosti těchto problémů.
- Řešení jednoho problému převodem na jiný problém.

Problém se snažíme “zařadit” do třídy s co nejnižší složitostí.

Většina problémů má limitní hranici složitosti, po jejímž dosažení již algoritmus nelze urychlit.

Dvě základní třídy složitosti:

- třída složitosti PTIME (problémy rychle řešitelné),
- třída složitosti NPTIME (problémy rychle verifikovatelné).

### Třída složitosti PTIME:

Do této třídy patří problémy řešitelné algoritmy s polynomiální složitostí  $O(n^c)$ .

$$PTIME = \bigcup_{c=0}^{\infty} \tau(n^c)$$

Výpočetně snadné, zvládnutelné, problémy.

V praxi nejčastěji používány algoritmy se složitostí:  $O(n)$ ,  $O(n \log(n))$ ,  $O(n^2)$ ,  $O(n^3)$ .

Příklady: aritmetické operace, třídění, vyhledávání, některé grafové či geometrické algoritmy.

## 28. Třída složitosti NPTIME

Třída algoritmů

$$NPTIME = \bigcup_{c=0}^{\infty} \tau(n^c)$$

NP problémy obtížně řešitelné ale verifikovatelné.

Řešitelné nedeterministickými algoritmy v polynomiálním čase  $O(n^c)$ ,  $c > 0$ .

Avšak existuje verifikační algoritmus pracující v polynomiálně omezeném čase.

Vztah mezi PTIME a NPTIME:

$$PTIME \subseteq NPTIME \text{ nebo } PTIME = NPTIME?$$

Další třídy:

- *NP-complete*  
Nelze nalézt exaktní řešení, lze verifikovat.
- *NP-hard*  
Nelze nalézt exaktní řešení ani verifikovat.

## 29. Příklady NP-úplných problémů

V současné době existuje přes 1000 NP-úplných problémů.

Není znám polynomiální algoritmus jejich řešení.

Často velmi podobné problémům, u kterých existuje polynomiální algoritmus.

Příklady NP-úplných problémů:

- TSP (problém obchodního cestujícího): navštívení všech vrcholů grafu právě jednou, délka cesty nejkratší.
- Problém  $k$  barev: obarvení vrcholů grafu tak, aby žádné dva sousední vrcholy nebyly obarveny dvěma stejnými barvami.
- Problém batohu: Jak naskládat do batohu předměty známého tvaru tak, aby zabíraly co nejméně místa?
- Loupežnický problém: rozdělení množiny na dvě podmnožiny se stejným součtem.
- Faktorizace prvočísel.

NP-úplné problémy lze řešit **přibližně**: aproximační algoritmy.