Úvod do složitosti algoritmů

prof. Ing. Pavel Tvrdík CSc.

Katedra počítačových systémů FIT
České vysoké učení technické v Praze

DSA, ZS 2009/10, Předn. 1

http://service.felk.cvut.cz/courses/X36DSA/

Organizace předmětu DSA

- Základní předmět informatického kurikula: Cílem je
  - seznámit se s vlastnostmi složitějších datových struktur a jejich využitím při řešení úloh v informatice,
  - seznámit se se základními algoritmy a naučit se je navrhovat a analyzovat, zvl. pro řazení a hledání.
- Cvičení: seminární, u tabule, 6 písemek po 8b = **48b**, zápočet min **26b** .
- Zkouška:
  - Roztřel: **24b**, min **16b**.
  - Velká písemka: max. **26b**.
  - Ústní: není.

Problémy a algoritmy

Výpočetní problém V: úkol zpracovat/transformovat vstupní data In na výstupní data Out s požadovanými vlastnostmi.

Algoritmus A: výpočetní postup řešení problému V

čili

posloupnost výpočetních kroků, která vezme vstupní data a vyprodukuje výstupní data, požadovaných vlastností podle zadání problému.

Instance problému: problém s konkrétními daty potřebnými pro jeho řešení.

Příklad

Problém řazení čísel (Sorting Problem).

Vstup: Posloupnost čísel $In = (a_1, a_2, \ldots, a_n)$.

Výstup: Permutace $Out = (a'_1, a'_2, \ldots, a'_n)$ posloupnosti In taková, že $a'_1 \leq a'_2 \leq \ldots \leq a'_n$.

Instance problému řazení:

Seřaďte vzestupně posloupnost $In = (75, 11, 34, 176, 59, 6, 54)$.

Korektnost algoritmů

Korektní algoritmus A pro problém V: pro každou instanci problému V, algoritmus A skončí v **konečném čase** se **správným** výstupem. Pak říkáme, že algoritmus A **řeší problém** V.

Čili:
A není korektní algoritmus, pokud:

- pro některou instanci problému **neskončí** (např. se "zacyklí") nebo
- pro některou instanci problému **skončí s nesprávným výstupem**.

Úmluva: v DSA uvažujeme pouze korektní algoritmy.

Zápis algoritmů

- Kritériem je srozumitelnost a stručnost zápisu.
- Způsoby zápisu algoritmů:
  - slovní popis,
  - vývojový diagram,
  - **pseudokód** (C-like, Java-like, Pascal-like), s případnými částmi (českých, anglických) vět,
  - kód v nějakém programovacím jazyku:
    - vyšší programovací jazyk,
    - mezijazyk (bytekód),
    - strojový jazyk,
  - logický obvod (např. FPGA).

Program

Program P: zápis (implementace) algoritmu A v programovacím jazyku, **spustitelný** (**proveditelný**) na nějakém počítači s nějakým OS a SW prostředím.

- U návrhu a implementace programů nás zajímají SW inženýrské otázky, jako např.
  - modularita,
  - ošetření vyjímek,
  - datová abstrakce a ošetření vstupů a výstupů,
  - dokumentace.
- Jednomu algoritmu A mohou odpovídat různé programy $P_1, P_2, \ldots$ v různých programovacích jazycích.
- Složitosti provedení jednotlivých programů téhož algoritmu se mohou lišit svojí složitostí v závislosti na architektuře počítače — nikoli ale řádově.

Různé algoritmy mohou mít různou složitost

- Pro **jeden problém** V mohou existovat **různé** algoritmy $A_1, A_2, \ldots$, pro jeho řešení.
- Jejich **složitosti** se mohou lišit výrazně (= **řádově**) svými nároky na **výpočetní prostředky**.
- Zvláště se mohou lišit svojí rychlostí (operační složitostí).

Jak měřit a určovat složitost algoritmů?

Analýza složitosti algoritmu: **výpočet/odhad/predikce** požadavků na výpočetní prostředky, které provedení algoritmu bude vyžadovat v závislosti na **velikosti vstupních dat**.

Velikost (složitost) vstupních dat závisí na specifikaci daného problému :

- počet bitů vstupního čísla,
- délka vstupní posloupnosti čísel,
- rozměry vstupní matice,
- počet znaků vstupního textu,
- ...

Výpočetní prostředky

- cykly procesoru $\Rightarrow$ **operační** složitost,
- paměťové buňky $\Rightarrow$ **paměťová** složitost,
- počet logických hradel FPGA při HW implementaci algoritmu,
- síťové kapacity komunikačních prostředků $\Rightarrow$ **komunikační** složitost.

Výpočetní (operační) složitost

Varianty:

- doba běhu algoritmu,
- doba výpočtu,
- počet provedených operací,
- počet provedených instrukcí (aritmetických, logických, paměťových),
- počet transakcí nad databází.

Případová analýza složitosti: Typický algoritmus obsahuje podmíněné příkazy $\Rightarrow$ složitost algoritmu **může obecně** záviset **nejen na velikosti** vstupních dat, ale také na jejich **hodnotách** (pak říkáme, že algoritmus je **citlivý na vstupní data**).

Příklad algoritmů s různou složitostí

Příklad

Zjistěte, zda jsou v poli $a[0, \ldots, n-1]$ všechny hodnoty navzájem různé.

8	1	4	6	3	10	-2

$\Rightarrow$ `true`

8	1	3	6	3	10	-2

$\Rightarrow$ `false`

1. algoritmus A_1 (intuitivní, jednoduchý)

- **Myšlenka:** porovnat každý prvek s každým, zda se shodují.

```
bool ruzne = true;
for (i=0; (i<n); i++)
    for (j=0; (j<n); j++)
        if ((i!=j) && (a[i]==a[j]))
            ruzne = false;
```

- **Počet iterací:** n^2.
- **Počet porovnání:** přibližně $h_1(n) \doteq 4n^2$.
- Kromě toho se provádí řada dalších operací, např. čtení z paměti.

2. lepší algoritmus A_2

- **Myšlenka:** ukončit cyklus, pokud jsou nalezeny stejné hodnoty.

```
bool ruzne = true;
for (i=0; (i<n) && ruzne; i++)
    for (j=0; (j<n) && ruzne; j++)
        if ((i!=j) && (a[i]==a[j]))
                    ruzne = false;
```

- **Počet iterací:** 1 až n^2.
- **Počet porovnání:** $h_2(n) \doteq 4$ až přibližně $4n^2$.

3. ještě lepší algoritmus A_3

- **Myšlenka:** Neporovnávat každý prvek s každým prvkem dvakrát (rovnost je symetrická relace)
- & elegantněji vynechat diagonálu.

```
bool ruzne = true;
for (i=1; (i<n) && ruzne; i++)
    for (j=0; (j<i) && ruzne; j++)
        if ( /* ... */ (a[i]==a[j]))
            ruzne = false;
```

- **Počet iterací:** 1 až $n(n-1)/2$.
- **Počet porovnání:** $h_3 \doteq 3$ až přibližně $3n(n-1)/2$.

4. ještě lepší algoritmus A_4

- **Myšlenka:** Hodnoty v poli nejprve seřadit.

```
bool ruzne = true;
seřaď vstupní pole a[0,...,n-1];
for (i=1; (i<n) && ruzne; i++)
    if (a[i]==a[i-1])
        ruzne = false;
```

- **Počet iterací:** 1 až $(n-1)$.
- **Počet porovnání:** 3 až přibližně $3n$.
- **Celková složitost:** nejvýše přibližně $h_4(n) \doteq kn \log n + 3n$, k je nějaká konstanta.

Vyjádření složitosti algoritmu

- Viděli jsme, že složitost algoritmu **může obecně** záviset na **hodnotách** vstupních dat.
- Proto je **obecně** potřebné vyjádřit složitost algoritmu:
  - v **nejlepším** případě,
  - v **nejhorším** případě,
  - v **průměrném** případě.
- Poslední případ je nejobtížnější, protože není vždy zřejmé, co to jsou vstupní data v průměrném případě. Typicky náhodně vygenerovaná, ale to nemusí být v praxi splněno.
- Platí pro časovou, paměťovou, komunikační, transakční,. . . .

Umění analýzy složitosti algoritmu

Umění analyzovat složitost algoritmů vyžaduje řadu znalostí a schopností:

- znalosti odhadů řádu rychlosti růstu funkcí,
- schopnost **abstrakce** výpočetního modelu,
- matematické znalosti z diskétní matematiky, kombinatoriky,
- znalosti sčítání řad a řešení rekurentních rovnic,
- znalosti základů teorie pravděpodobnosti,
- a další.

RAM model

- Pro analýzu složitosti algoritmu potřebujeme **model výpočetního systému**, na kterém algoritmus bude hypoteticky implementován.
- Nejčastější je **jednoprocesorový Random Access Machine** (RAM) model:
  - data uložena v adresovatelné paměti,
  - aritmetickologické, řídící, paměťové instrukce,
  - časová složitost instrukcí je jednotková nebo konstatní,
  - intrukce jsou prováděny postupně (sekvenčně).

Asymptotická složitost

- Přestože lze v případě jednodušších algoritmů určit složitost přesně,
  - ve většině případů to nestojí za tu námahu,
  - v řadě případů složitých algoritmů je to velmi obtížné.
- Proto se to typicky nedělá.
- Z praktického hlediska nás zajímá především, jak se bude algoritmus chovat pro **velké** ($\to \infty$) instance problému.
- Typicky nám stačí vyjádřit **řád růstu** funkce složitosti se zanedbáním příspěvků nižších řádu.

Příklad

Složitost 4. algoritmu je nejvýše $h_4(n) = kn \log n + 3n \doteq kn \log n$ pro velká n ($\Rightarrow$ řazení dominuje zbytku algoritmu).

Asymptotická složitost (pokr.)

- Zajímá nás, jak složitost algoritmu závisí na velkosti vstupních dat v **limitě**, pokud velikost instance problému poroste **nade všechny meze**.
- Pro první porozumění chování algoritmu nás nemusejí zajímat ani **multiplikativní** konstanty $\Rightarrow$ říkáme, že nás zajímá **asymptotická složitost**.
- Je to složitost vyjádřená pro instance problému dostatečně velké, aby se jasně projevil **řád růstu** funkce složitosti v závislosti na velikosti vstupních dat.
- **Asymptoticky lepší** algoritmus bude **lepší pro všechny** instance problému **kromě konečného počtu menších** instancí.

Příklad

Hodnoty fce $f(n) = 1.5n^2 + 140\sqrt{n} + 30$ pro velká n budou velmi blízké hodnotám fce $g(n) = 1.5n^2$ nebo fce $h(n) = 1.5n^2 - 25n + 44$.

Asymptotická aritmetika

- $\mathcal{N}^+ =$ množina přirozených čísel
- $R^+ =$ množina kladných reálných čísel.
- Uvažujeme pouze kladné funkce jedné přirozené proměnné n: $f, g, \ldots : \mathcal{N}^+ \rightarrow \Re^+$.

Asymptotická těsná mez : Θ-notace

Definice

(Relační) Jsou-li dány funkce $f(n)$ a $g(n)$, pak řekneme, že $f(n)$ je **téhož řádu jako** *$g(n)$, psáno $f(n) = \Theta(g(n))$, jestliže*

$\exists c_1, c_2 \in \Re^+ \quad \exists n_0 \in \mathcal{N}^+; \quad \forall n \geq n_0 : \quad c_1.g(n) \leq f(n) \leq c_2.g(n).$

Alternativní definice.

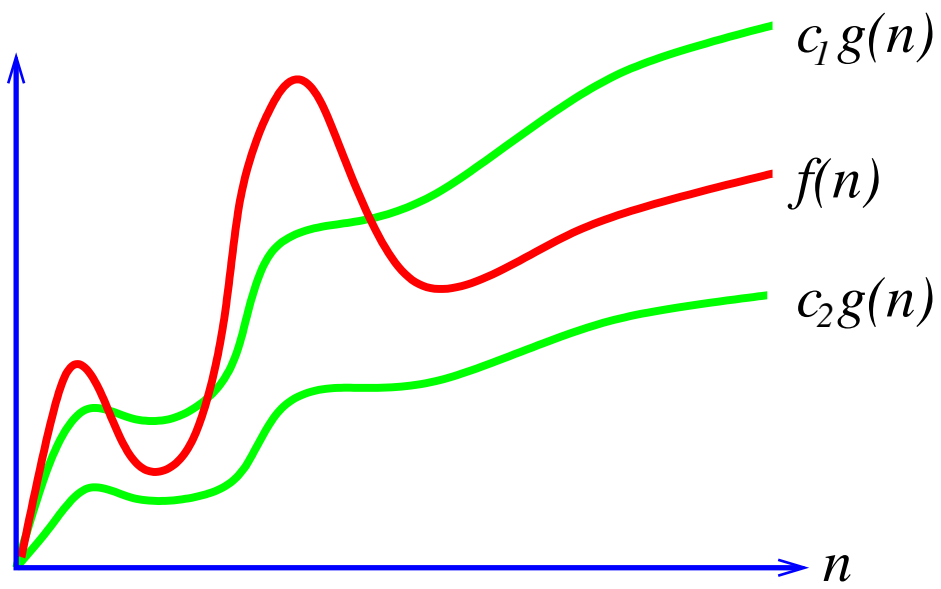
Definice

(Množinová) Je-li dána funkce $g(n)$, pak $\Theta(g(n))$ je definována jako **(nekonečná) množina funkcí**

$\{f(n) : \exists c_1, c_2 \in \Re^+ \, \exists n_0 \in \mathcal{N}^+ \, \forall n \geq n_0 : \, c_1.g(n) \leq f(n) \leq c_2.g(n)\}.$

- Zápisy $f(n) \in \Theta(g(n))$ a $f(n) = \Theta(g(n))$ jsou rovnocenné a záleží pouze na kontextu, který je vhodnější. (To samé platí pro O, Ω, o, ω).
- Θ-notaci používáme pro vyjádření faktu, že 2 funkce jsou asymptoticky stejné až na **multiplikativní** konstantu.

Asymptotická těsná mez : Θ-notace (pokr.)



Příklad

- $3n^2 - 5n - 15 = \Theta(n^2)$
- Obecně pro každý polynom $\sum_{i=0}^{d} a_i n^i = \Theta(n^d)$, pokud $a_d > 0$.
- $\frac{1}{3}\log^2 n + 2\sqrt[3]{n} = \Theta(\sqrt[3]{n})$: funkce $\frac{1}{3}\log^2 n + 2\sqrt[3]{n}$ a $\sqrt[3]{n}$ rostou řádově stejně rychle.

Asymptotická horní mez : O-notace

Definice

(Relační) Jsou-li dány funkce $f(n)$ a $g(n)$, pak řekneme, že $f(n)$ je **nejvýše řádu** *$g(n)$, psáno $f(n) = O(g(n))$, jestliže*
$\exists c_2 \in \Re^+ \quad \exists n_0 \in \mathcal{N}^+; \quad \forall n \geq n_0 : \quad f(n) \leq c_2.g(n).$

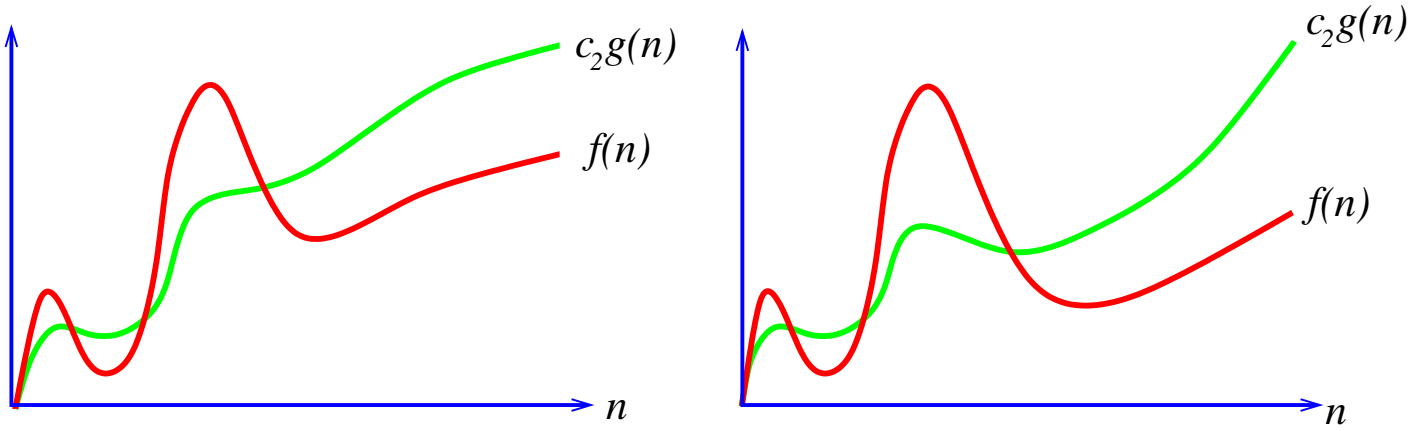
Alternativní definice.

Definice

(Množinová) Je-li dána funkce $g(n)$, pak $O(g(n))$ je definována jako **(nekonečná) množina funkcí**
$\{f(n) : \exists c_2 \in \Re^+ \quad \exists n_0 \in \mathcal{N}^+; \quad \forall n \geq n_0 : \quad f(n) \leq c_2.g(n)\}.$

- O-notaci používáme pro vyjádření horní meze funkce až na **multiplikativní** konstantu.

Asymptotická horní mez : O-notace (pokr.)



- $f(n) = \Theta(g(n))$ implikuje $f(n) = O(g(n))$
  (Θ je silnější podmínka než O neboli $\Theta(g(n)) \subset O(g(n))$).
- Zápisem $f(n) = O(g(n))$ vyjadřujeme, že $g(n)$ je pro $f(n)$ asymp. horní mezí **stejného nebo vyššího** řádu.
- O-výrazy používáme pro **odhady** složitostí algoritmů v **nejhorších** případech.

Příklad

- $3n^2 - 5n - 15 = O(n^2)$, ale také $3n^2 - 5n - 15 = O(n^3)$

Asymptotická dolní mez : Ω-notace

Definice

(Relační) Jsou-li dány funkce $f(n)$ a $g(n)$, pak řekneme, že $f(n)$ je **nejméně řádu** *$g(n)$, psáno $f(n) = \Omega(g(n))$, jestliže*
$\exists c_1 \in \Re^+ \quad \exists n_0 \in \mathcal{N}^+; \quad \forall n \geq n_0 : \quad c_1.g(n) \leq f(n).$

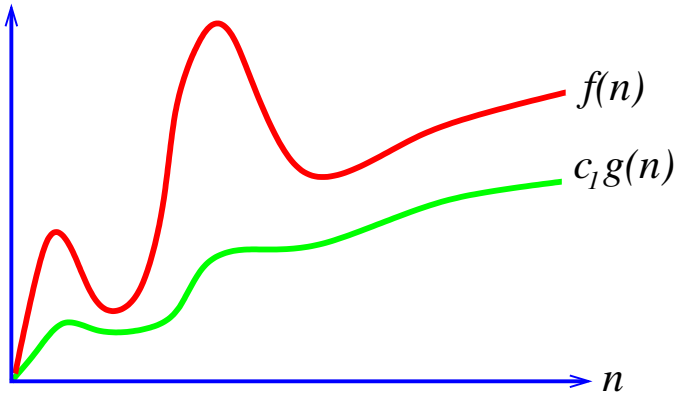
Alternativní definice.

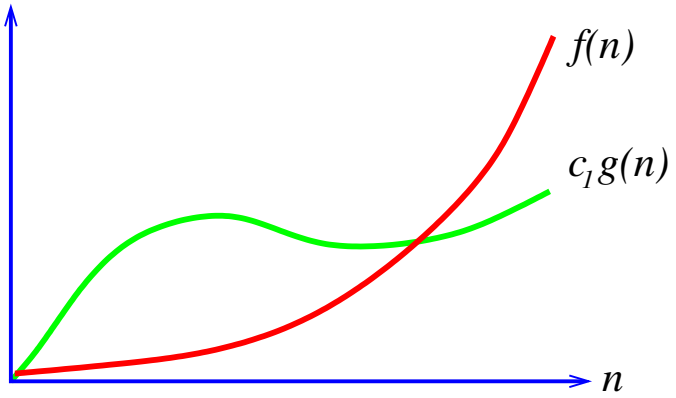
Definice

(Množinová) Je-li dána funkce $g(n)$, pak $\Omega(g(n))$ je definována jako **(nekonečná) množina funkcí**
$\{f(n) : \exists c_1 \in \Re^+ \quad \exists n_0 \in \mathcal{N}^+; \quad \forall n \geq n_0 : \quad c_1.g(n) \leq f(n)\}.$

- Ω-notaci používáme pro vyjádření **dolní meze** funkce až na **multiplikativní** konstantu.

Asymptotická dolní mez : Ω-notace (pokr.)





- $f(n) = \Theta(g(n))$ implikuje $f(n) = \Omega(g(n))$
  (Θ je silnější podmínka než Ω neboli $\Theta(g(n)) \subset \Omega(g(n))$).
- Zápisem $f(n) = \Omega(g(n))$ vyjadřujeme, že $g(n)$ je pro $f(n)$ asymp. dolní mezí **stejného nebo nižšího** řádu.
- Ω-výrazy používáme pro **odhady** složitostí algoritmů v **nejlepších** případech.

Příklad

- $2n \log n + 3\sqrt{n} + 1 = \Omega(n \log n)$, ale také
  $2n \log n + 3\sqrt{n} + 1 = \Omega(n)$.

Striktně větší horní mez : o-notace

Definice

(Relační) Jsou-li dány funkce $f(n)$ a $g(n)$, pak řekneme, že $f(n)$ je **striktně nižšího řádu než** *$g(n)$, psáno $f(n) = o(g(n))$, jestliže*
$\forall c_2 \in \Re^+ \quad \exists n_0 \in \mathcal{N}^+ \quad \forall n \geq n_0 : \quad f(n) < c_2.g(n).$

Alternativní definice.

Definice

(Množinová) Je-li dána funkce $g(n)$, pak $o(g(n))$ je definována jako **(nekonečná) množina funkcí**
$\{f(n) : \forall c_2 \in \Re^+ \quad \exists n_0 \in \mathcal{N}^+ \quad \forall n \geq n_0 : \quad f(n) < c_2.g(n)\}.$

- Zápisy $f(n) \in o(g(n))$ a $f(n) = o(g(n))$ jsou opět rovnocenné.
- Vyjadřují, že $g(n)$ je pro $f(n)$ asymp. horní mezí **vyššího** řádu.
- Pak platí $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$.

Striktně menší dolní mez : ω-notace

Definice

(Relační) Jsou-li dány funkce $f(n)$ a $g(n)$, pak řekneme, že $f(n)$ je **striktně vyššího řádu** *než $g(n)$, psáno $f(n) = \omega(g(n))$, jestliže*
$\forall c_1 \in \Re^+ \quad \exists n_0 \in \mathcal{N}^+; \quad \forall n \geq n_0 : \quad c_1.g(n) < f(n).$

Alternativní definice.

Definice

(Množinová) Je-li dána funkce $g(n)$, pak $\omega(g(n))$ je definována jako **(nekonečná) množina funkcí**
$\{f(n) : \forall c_1 \in \Re^+ \quad \exists n_0 \in \mathcal{N}^+; \quad \forall n \geq n_0 : \quad c_1.g(n) < f(n)\}.$

- Zápis $f(n) = \omega(g(n))$ vyjadřuje, že $g(n)$ je pro $f(n)$ asymp. dolní mezí **nižšího** řádu.
- Pak platí $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$.
- Žádná multiplikativní konstanta neumožní, aby $g(n)$ dostihlo $f(n)$.

Použití asymptotické notace

- Asymptotická notace nám umožňuje zjednodušovat výrazy zanedbáním nepodstatných částí.
- Co znamená zápis $f(n) = 4n^3 + \Theta(n^2)$?
  Výraz $\Theta(n)$ zastupuje anonymní funkci z množiny $\Theta(n^2)$ (nějaká kvadratická funkce), jejíž konkrétní podobu pro zatím zanedbáváme.
- Co znamenají zápisy $\Theta(1)$, $O(1)$, $\Omega(1)$?
  Přesněji neurčené **konstantní** meze.
- Co znamená zápis $f(n) = O(n^{O(1)})$?
  $f(n)$ je shora omezena nějakou polynomiální funkcí n^c, kde c sice přesně neznáme, ale víme, že to je konstanta.

Zákony asymptotické aritmetika

Transitivita:	$f(n) = O(g(n)) \ \wedge \ g(n) = O(h(n)) \Rightarrow f(n) = O(h(n))$. Podobně pro $\Omega, \Theta, o, \omega$.
Reflexivita:	$f(n) = O(f(n))$. Podobně pro Ω, Θ.
Symetrie:	$f(n) = \Theta(g(n)) \Leftrightarrow g(n) = \Theta(f(n))$.
Transpoziční symetrie:	$f(n) = O(g(n)) \Leftrightarrow g(n) = \Omega(f(n))$, $f(n) = o(g(n)) \Leftrightarrow g(n) = \omega(f(n))$.
Inkluze:	$f(n) = o(g(n)) \Rightarrow f(n) = O(g(n))$, $f(n) = \Theta(g(n)) \Rightarrow f(n) = O(g(n))$, $f(n) = \omega(g(n)) \Rightarrow f(n) = \Omega(g(n))$. $f(n) = \Theta(g(n)) \Rightarrow f(n) = \Omega(g(n))$,

Všimněte si následující analogie s porovnáváním čísel:

O	Ω	Θ	o	ω
		$\approx$		
$\leq$	$\geq$	$=$	$<$	$>$

Základní funkce jedné proměnné

- dolní celá část $\lfloor x \rfloor$, horní celá část $\lceil x \rceil$,
- polynomiální
  - exponent > 1, např. n^2,
  - lineární,
  - exponent < 1, např. $\sqrt[4]{n}$,
- exponenciální, např. 2^n,
- polylogaritmické, např. $\log^3 n$,
- faktoriální $n!$,
- logaritmická iterace $\log^* n$,
- Fibonacciho čísla $F(n)$.
- ....

Porovnání řádu funkcí

Příklad

1. Dokažte, že $\log n = o(n^c)$ pro libovolnou konstantu $0 < c < 1$.
2. Seřaďte následující funkce dle jejich asymptotické složitosti.

n^3	$\lceil \log n \rceil !$	e^n
$2^{\sqrt{n}}$	n	$\log(n!)$
$\log \log n$	1.5^n	$n2^n$
$n!$	$n^{\log n}$	$\log n$
$\log^2 n$	2^{2^n}	$n^2 \log n$
2^n	$n/\log n$	$2^{\log^2 n}$
$n^{1/\log n}$	$\sqrt{\log n}$	$n \log n$

Srovnání časových složitostí

Předpokládejme, že 1 operace trvá čas $1\mu s$ a že počet operací je dán funkcí v prvním sloupci. Pak doba výpočtu bude pro různé hodnoty n následující: (Připomeňme, že počet atomů ve vesmíru se odhaduje na 10^{80} a stáří vesmíru na 14×10^9 let.)

n	10	20	40	80	500	1000
$\log n$	$3,3\ \mu$s	$4,3\mu s$	$5\mu s$	$5,8\mu s$	$9\mu s$	$10\mu s$
n	$10\mu s$	$20\mu s$	$40\mu s$	$60\mu s$	$0,5ms$	$1ms$
$n \log n$	$33\mu s$	$86\mu s$	$0,2ms$	$0,35ms$	$4,5ms$	$10ms$
n^2	$0,1ms$	$0,4ms$	$1,6\ ms$	$3,6ms$	$0,25s$	$1s$
n^3	$1ms$	$8ms$	$64ms$	$0,2s$	$125s$	$17min$
n^4	$10ms$	$160ms$	$2,56s$	$13s$	17 h	$11,6$ dní
2^n	$1ms$	$1s$	$12,7$dní	3600 let	10^{137} let	10^{287} let
$n!$	$3,6s$	77100 let	10^{34} let	10^{105} let	10^{1110} let	10^{2554} let

Základní diskrétní posloupnosti a řady

- **Aritmetická posloupnost:**
  Základní: $1 + 2 + \cdots + n = \sum_{i=1}^{n} i = \frac{1}{2}n(n+1)$
  Obecná: $a_1 + \cdots + a_n = \frac{1}{2}n(a_1 + a_n)$.
- **Geometrická posloupnost:** Pro $x \neq 1$:
  $1 + x + x^2 + \cdots + x^n = \sum_{i=0}^{n} x^i = \frac{x^{n+1}-1}{x-1}$.
- **Obecná posloupnost:** Je-li $f(n)$ **monotonně rostoucí** funkce, lze součet řady $\sum_{k=m}^{n} f(k)$ aproximovat určitým integrálem:

$$\int_{m-1}^{n} f(x)dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m}^{n+1} f(x)dx$$

Aplikace:

- Příklad 1: **Harmonická posloupnost:**
  $H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \frac{1}{n} = \sum_{k=1}^{n} \frac{1}{k} = \ln n + O(1)$.
- Příklad 2: **Součet kvadrátů:**
  $1 + 2^2 + 3^2 + \cdots + n^2 = \sum_{k=1}^{n} k^2 \doteq \frac{n^3}{3} + \frac{n^2}{2}$.

Základní kombinatorika

- **Permutace:** (prvky se neopakují, záleží na pořadí)
  Počet k-permutací z n prvků je $\frac{n!}{(n-k)!} = n(n-1)\dots(n-k+1)$.
  Počet permutací z n prvků je $n!$.
- **Permutace s opakováním:** (prvky se opakují, záleží na pořadí)

$$\frac{n!}{k_1!k_2!\dots k_d!},$$

  kde $\sum_{i=1}^{d} k_i = n$.
- **Kombinace:** (prvky se neopakují, nezáleží na pořadí).
  Počet k-kombinací z n prvků je $\binom{n}{k} = \frac{n!}{(n-k)!k!}$.
  $=$ Počet k-podmnožin z n-prvkové množiny.
  $=$ Počet n-bitových řetězců s k bity 1.
  $=$ Permutace s opakováním pro $d = 2$.
- **Variace** (prvky se opakují libovolně, záleží na pořadí):
  Počet k-variací z n-prvkové množiny je n^k.