Složitost algoritmů

Karel Richta a kol.

Přednášky byly připraveny s pomocí materiálů, které vyrobili Marko Berezovský, Petr Felkel, Josef Kolář, Michal Píše a Pavel Tvrdík

Katedra počítačů
Fakulta elektrotechnická
České vysoké učení technické v Praze

© Karel Richta a kol., 2018

Datové struktury a algoritmy, B6B36DSA
01/2018, Lekce 3

https://cw.fel.cvut.cz/wiki/courses/b6b36dsa/start

Složitost algoritmů

- Algoritmy mají různou složitost.
- V praxi je důležité umět mezi sebou porovnat více algoritmů, které řeší stejný problém, abychom mohli rozhodnout, kdy který použít.
- Existují dvě přirozené míry pro porovnání:
  - doba výpočtu podle daného algoritmu potřebná pro zpracování daného objemu dat (časová složitost) a
  - velikost paměti využívané při výpočtu (paměťová složitost).
- Pro určení složitosti zpravidla používáme intuitivní abstraktní model RAM počítače, který pracuje s celými čísly a má paměť adresovanou celými čísly, do které lze ukládat celá čísla.
- Některé metody určení složitosti rekurzivních algoritmů:
  - strom rekurze,
  - substituční metoda,
  - mistrovská metoda.

Asymptotická složitost

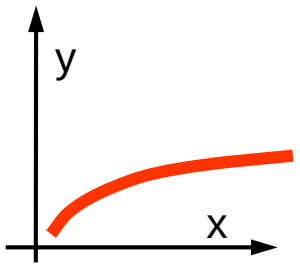
- Skutečná složitost výpočtu závisí na implementaci algoritmu, na konkrétním počítači kde je prováděn - vlastně se nedá v obecném případě přesně spočítat. Abychom mohli spočítat aspoň něco, začaly se používat odhady složitosti až na multiplikativní konstantu. Tyto odhady popisují růst složitosti vzhledem ke zvětšujícím se vstupům, ale neurčují konkrétní funkci (číslo).
- 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 ($\rightarrow \infty$) instance problému.
- Typicky nám stačí vyjádřit řád růstu funkce složitosti se zanedbáním příspěvků nižších řádů.
- Říkáme tomu proto asymptotická složitost $\rightarrow$ asymptoticky se blíží k této hodnotě.

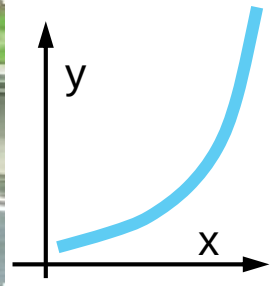
Rychlost...

Jeden algoritmus (program, postup, metoda…)
je rychlejší než druhý.

Co ta věta znamená ??

Asymptotická složitost





Každému algoritmu lze jednoznačně přiřadit

neklesající funkci

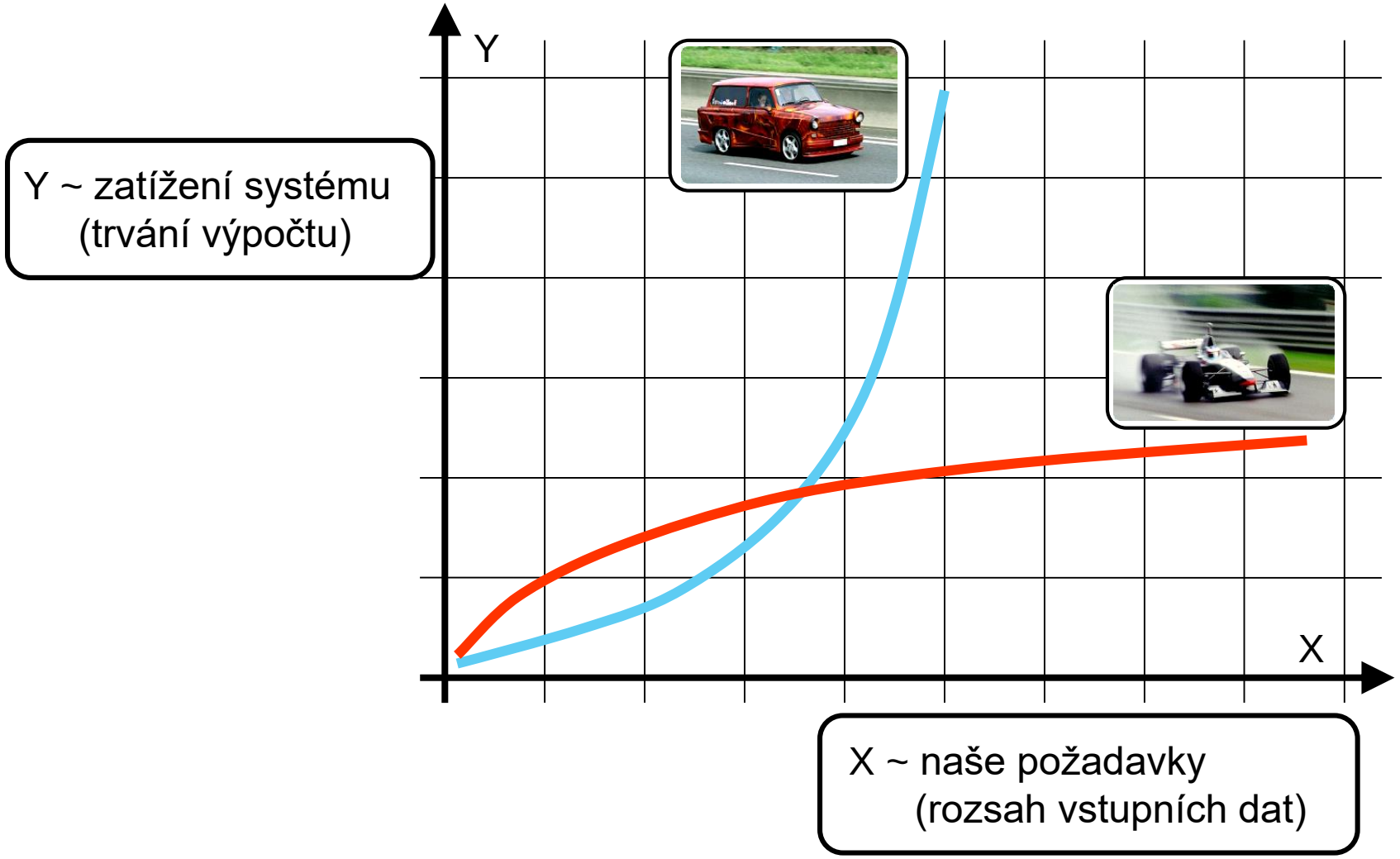
zvanou

asymptotická (časová) složitost,

která charakterizuje počet operací algoritmu
v závislosti na rostoucím rozsahu vstupních dat.

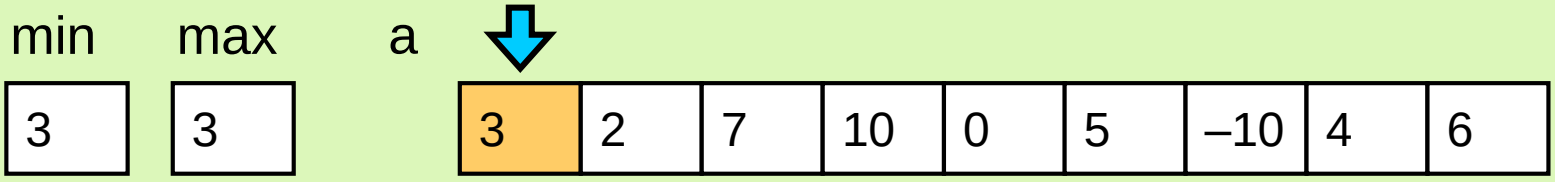
Čím pomaleji tato funkce roste, tím je algoritmus rychlejší.

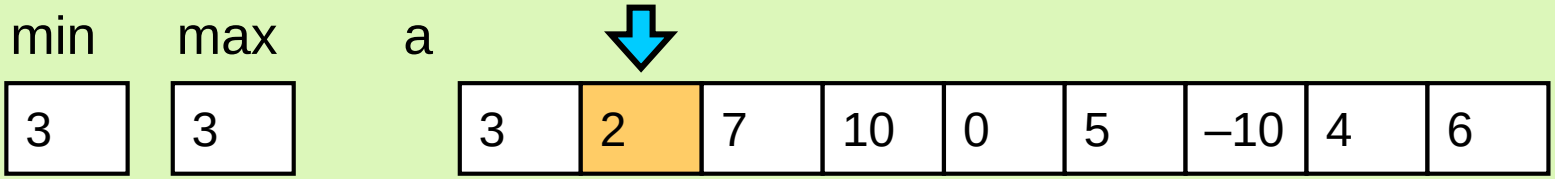
Asymptotická složitost



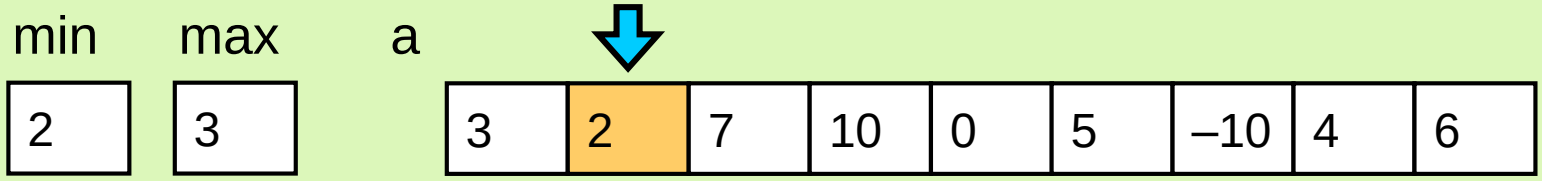
Příklady

Najdi min and max hodnotu v poli — STANDARD



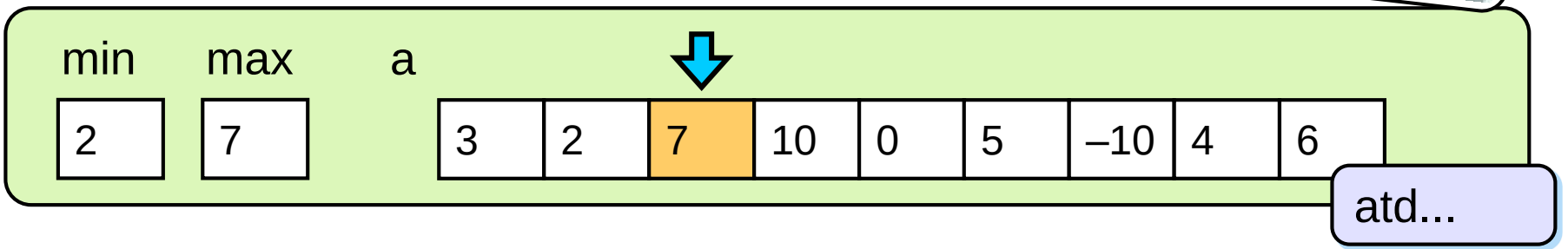


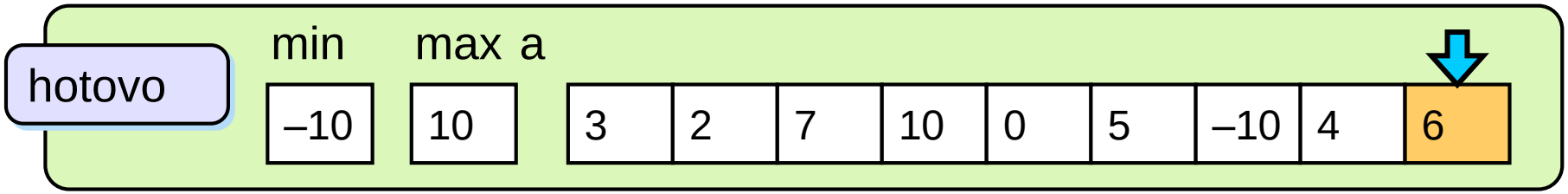

```
if (a[i] < min) min = a[i];
if (a[i] > max) max = a[i];
```



Příklady

Najdi min and max hodnotu v poli — STANDARD

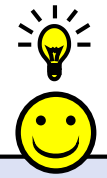




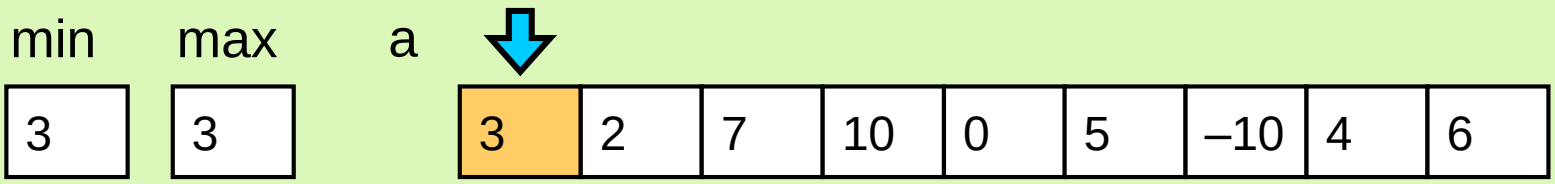
kód

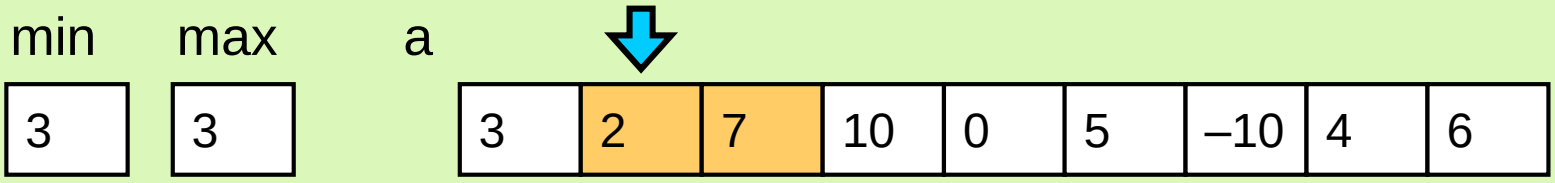
```
min = a[0]; max = a[0];
for ( i = 1; i < a.length; i++ ) {
   if (a[i] < min) min = a[i];
   if (a[i] > max) max = a[i]; }
```

Příklady

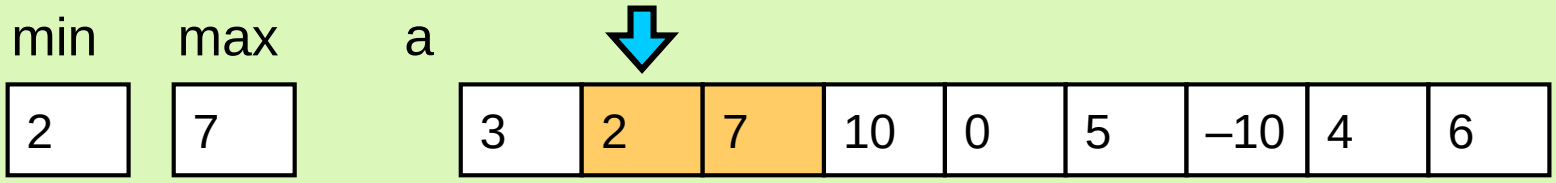


Najdi min and max hodnotu v poli — RYCHLEJI!



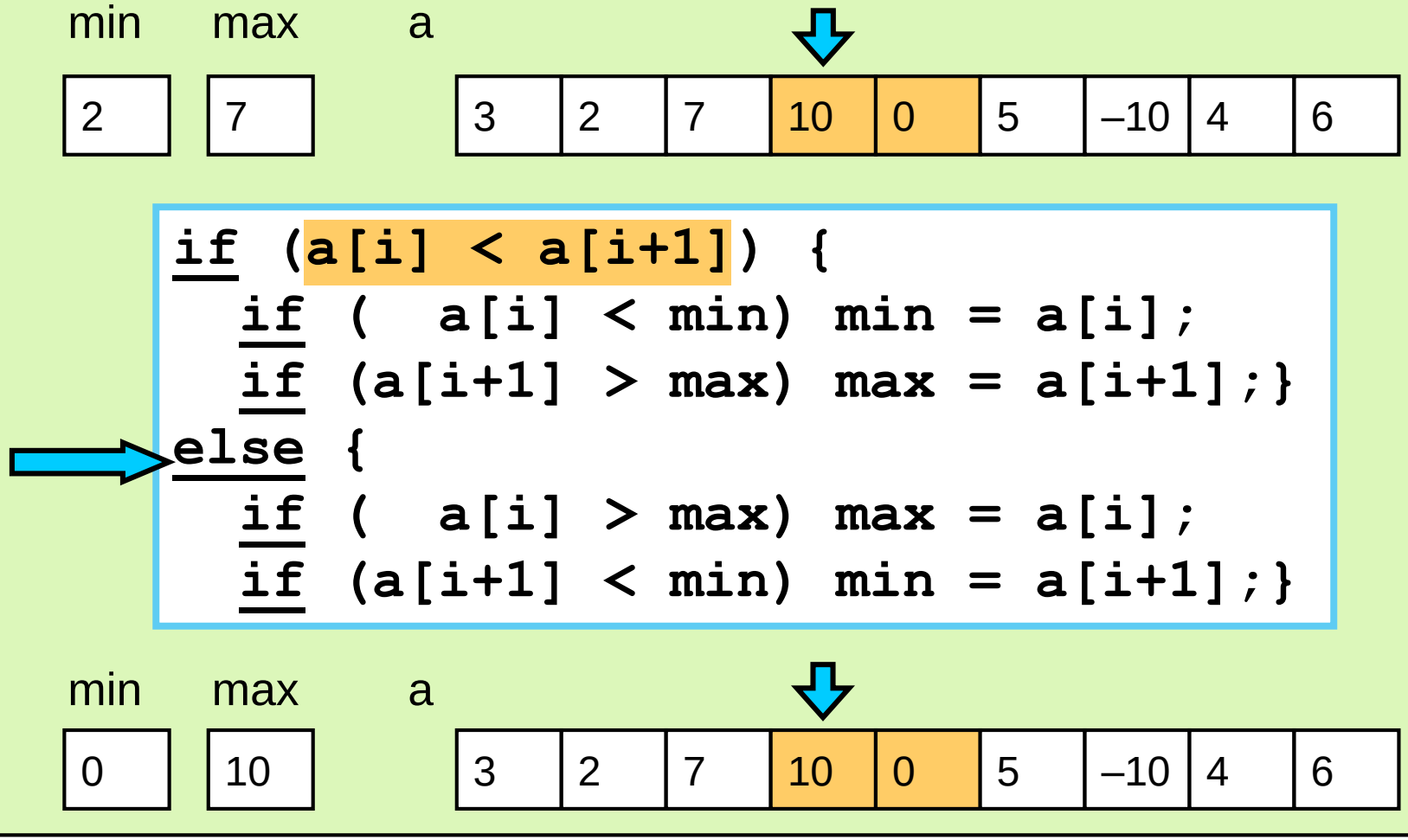



```
if (a[i] < a[i+1]) {
   if (  a[i] < min) min = a[i];
   if (a[i+1] > max) max = a[i+1];}
```



Příklady

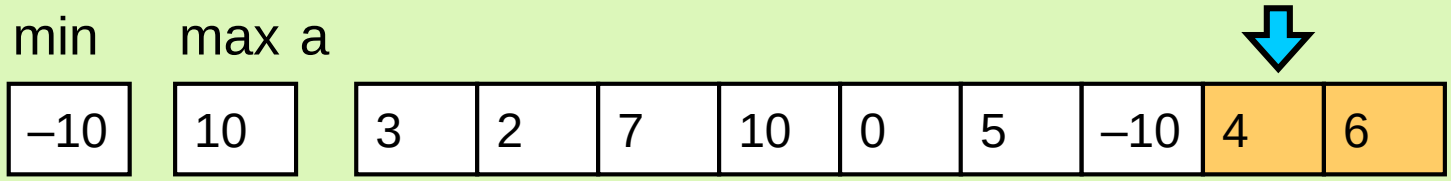
Najdi min and max hodnotu v poli — RYCHLEJI!



Příklady

Najdi min and max hodnotu v poli — RYCHLEJI!

hotovo



kód

```
min = a[0]; max = a[0];
for (i=1; i < a.length-1; i=i+2 ) {
  if (a[i] < a[i+1]) {
    if (  a[i] < min) min = a[i];
    if (a[i+1] > max) max = a[i+1];}
  else {
    if (  a[i] > max) max = a[i];
    if (a[i+1] < min) min = a[i+1];}}
```

Počítání složitosti

Elementární operace

- aritmetická operace
- porovnání dvou čísel
- přesun čísla v paměti

Složitost (A)

celkový počet elementárních operací

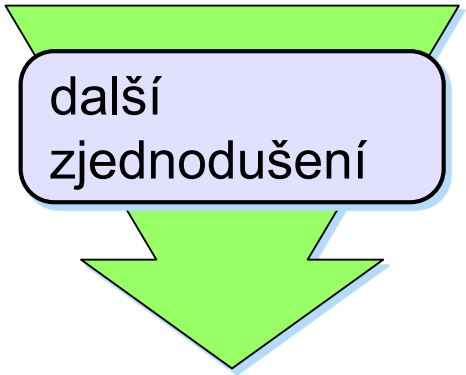
zjednodušení

Složitost (B)

celkový počet elementárních operací nad daty

Počítání složitosti

Složitost B — celkový počet elementárních operací nad daty



Složitost C — celkový počet porovnání čísel (znaků) v datech

Takto složitost mnohdy počítáme

Počítání složitosti

Najdi min and max hodnotu v poli — STANDARD

A

Složitost

Všechny operace

a.length = N

```
min = a[0]; max = a[0];
for ( i = 1; i < a.length; i++){
  if (a[i] < min) min = a[i];
  if (a[i] > max) max = a[i]; }
```

1, 1, 1, N, N–1, N–1, N–1, 0...N–1

exkluzive!

Případ

nejlepší: 1 + 1 + 1 + N + N–1 + N–1 + 0 + N–1 + 0 = 4N

nejhorší: 1 + 1 + 1 + N + N–1 + N–1 + N–1 + N–1 = 5N–1

Počítání složitosti

Najdi min and max hodnotu v poli — STANDARD

složitost B

Operace nad daty

a.length = N

```
min = a[0]; max = a[0];
for ( i = 1; i < a.length; i++){
  if (a[i] < min) min = a[i];
  if (a[i] > max) max = a[i]; }
```

1 1 N–1 0...N–1 N–1 0...N–1

Případ

nejlepší: 1 + 1 + N–1 + 0 + N–1 + 0 = 2N

nejhorší: 1 + 1 + N–1 + N–1 + N–1 + N–1 = 4N–2

Počítání složitosti

Najdi min and max hodnotu v poli — STANDARD

C

složitost

pouze testy v datech

a.length = N

```
min = a[0]; max = a[0];

for ( i = 1; i < a.length; i++){

  if (a[i] < min) min = a[i];

  if (a[i] > max) max = a[i]; }
```

N–1

N–1

vždy

N–1 + N–1 = 2N–2 testů

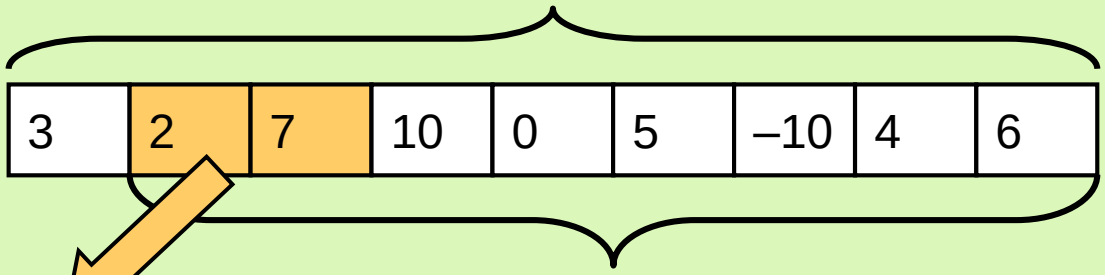
Počítání složitosti

Najdi min and max hodnotu v poli — RYCHLEJI!

složitost C

pouze testy v datech

a.length = N



Jedna dvojice — 3 testy

(N–1)/2 dvojic

vždy 3(N–1)/2 = (3N – 3)/2 testů

Počítání složitosti

Velikost pole **N**	Počet testů STANDARDNÍ **2 (N – 1)**	Počet testů RYCHLEJŠÍ **(3N – 3)/2**	poměr STD./RYCHL.
11	20	15	1.33
21	40	30	1.33
51	100	75	1.33
101	200	150	1.33
201	400	300	1.33
501	1 000	750	1.33
1 001	2 000	1 500	1.33
2 001	4 000	3 000	1.33
5 001	10 000	7 500	1.33
1 000 001	2 000 000	1 500 000	1.33

Tab. 1

Příklady

data

pole a:

1	–1	0	–2	5	1	0

pole b:

4	2	4	3	4	2	7

úloha

Kolik prvků pole b je rovno součtu prvků pole a?

řešení

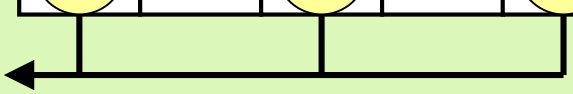
pole a:

1	–1	0	–2	5	1	0

součet = 4

pole b:

4	2	4	3	4	2	7



výsledek = 3

Příklady

funkce

```
int sumArr(int[] a) { /*snadné*/ }
```

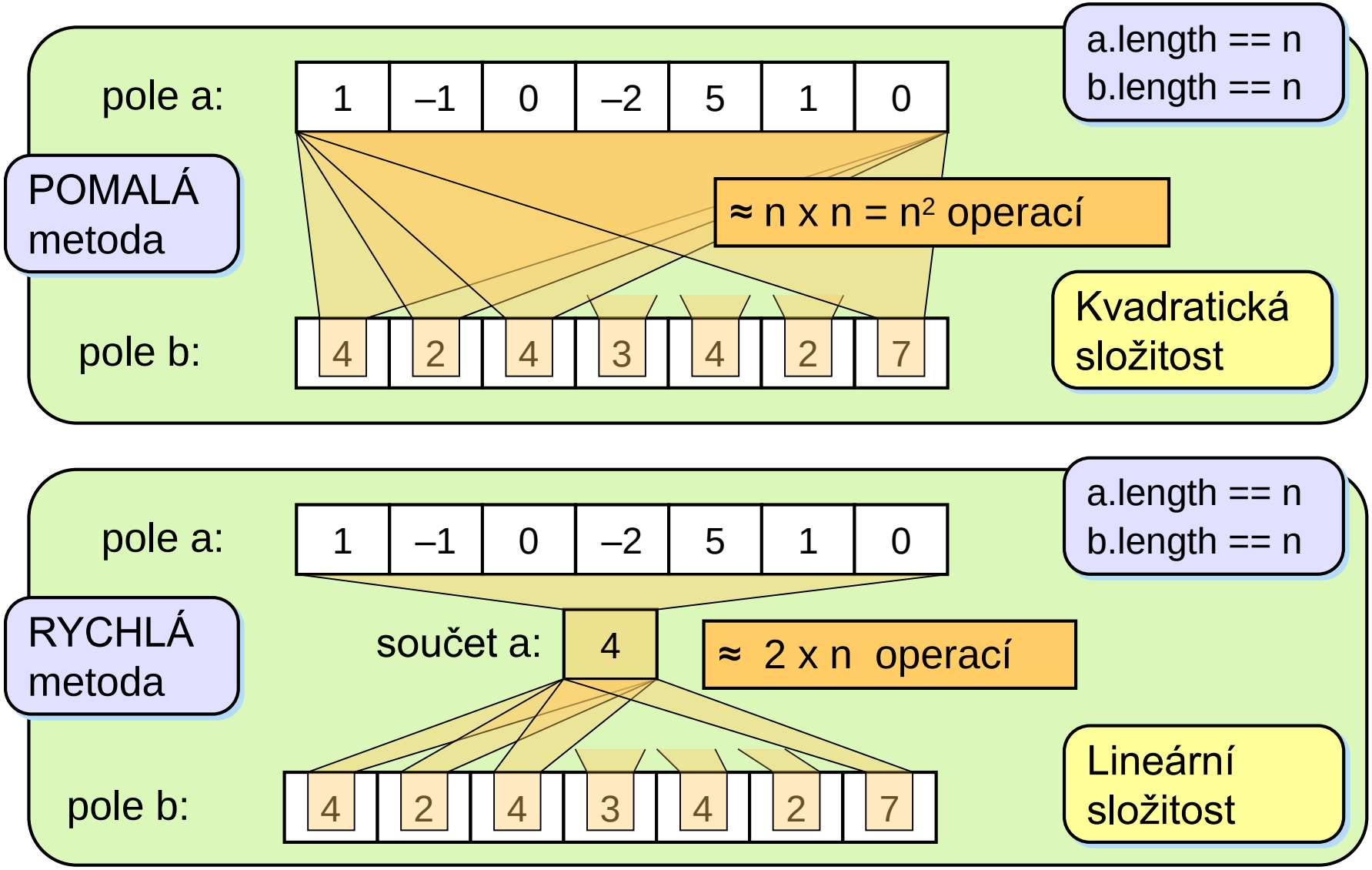
POMALÁ metoda

```
count = 0;
for (int i = 0; i < b.length; i++)
  if (b[i]==sumArr(a)) count++;
return count;
```

RYCHLÁ metoda

```
count = 0;
sumOf_b = sumArr(a);
for (int i = 0; i < b.length; i++)
  if (b[i]==sumOf_b) count++;
return count;
```

Počítání složitosti



Počítání složitosti

Velikost pole **N**	POMALÁ metoda operací **N^2**	RYCHLÁ metoda operací **2N**	poměr POMALÁ/RYCHLÁ
11	121	22	5.5
21	441	42	10 .5
51	2 601	102	25 .5
101	10 201	202	50 .5
201	40 401	402	100 .5
501	251 001	1 002	250 .5
1 001	1 002 001	2 002	500 .5
2 001	4 004 001	4 002	1 000 .5
5 001	25 010 001	10 002	2 500 .5
1 000 001	1 000 002 000 001	2 000 002	500 000 .5

Tab. 2

Počítání složitosti

Velikost pole **N**	Poměr rychlostí řešení 1. úlohy	Poměr rychlostí řešení 2. úlohy
11	1.33	5.5
21	1.33	10 .5
51	1.33	25 .5
101	1.33	50 .5
201	1.33	100 .5
501	1.33	250 .5
1 001	1.33	500 .5
2 001	1.33	1 000 .5
5 001	1.33	2 500 .5
1 000 001	1.33	500 000 .5

Tab. 3

Příklady

Hledání v seřazeném poli — lineární, POMALÉ

dané pole

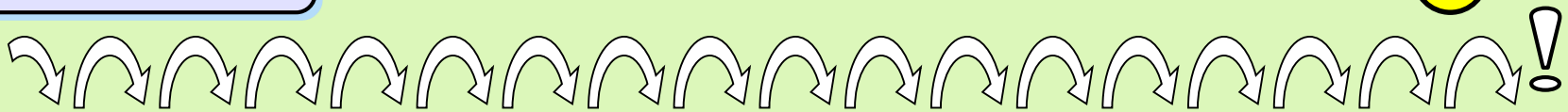
Seřazené pole: →

velikost = N

363	369	388	603	638	693	803	833	836	839	860	863	938	939	966	968	983	993

najdi 993 !

testů: N



363	369	388	603	638	693	803	833	836	839	860	863	938	939	966	968	983	993

najdi 363 !

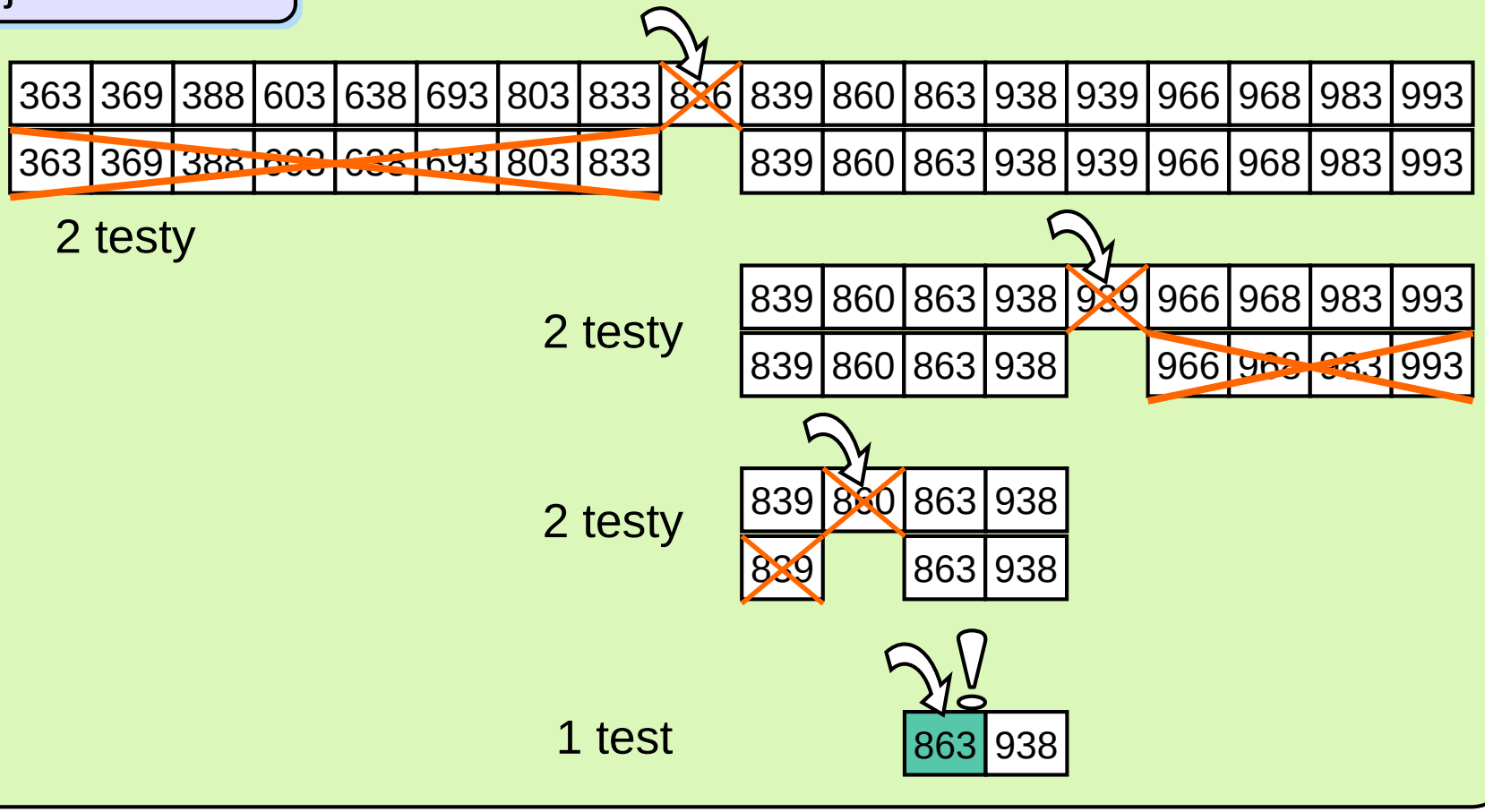
testů: 1

363	369	388	603	638	693	803	833	836	839	860	863	938	939	966	968	983	993

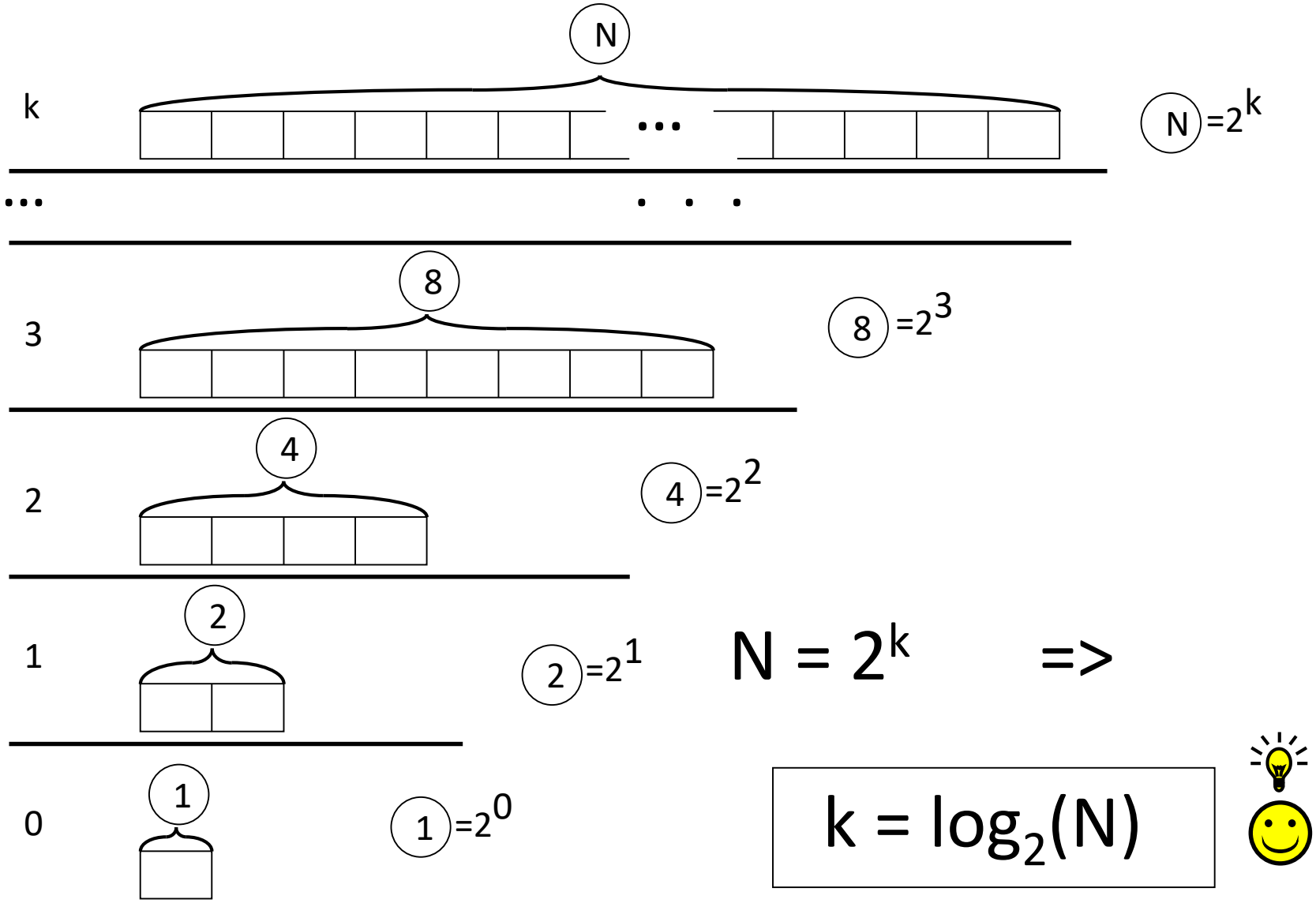
Příklady

Hledání v seřazeném poli — binární, RYCHLÉ

najdi 863 !



Exponent, logarimus a půlení intervalu



Počítání složitosti

Velikost pole	Počet testů: lineární hledání — případ nejlepší	lineární hledání — případ nejhorší	lineární hledání — případ průměrný	binární hledání nejhorší případ	poměr
5	1	5	3	5	0.6
10	1	10	5.5	7	0.79
20	1	20	10.5	9	1.17
50	1	50	25.5	11	2.32
100	1	100	50.5	13	3.88
200	1	200	100.5	15	6.70
500	1	500	250.5	17	14.74
1 000	1	1000	500.5	19	26.34
2 000	1	2000	1000.5	21	47.64
5 000	1	5000	2500.5	25	100.02
1 000 000	1	1 000 000	500 000.5	59	8 474.58

Tab. 4

Počítání složitosti

Doba výpočtu
pro různé časové složitosti
s předpokladem, že 1 operace trvá 1 μs (10^{-6} sec)

složitost	Počet operací					
	10	20	40	60	500	1000
$log_2\, n$	3,3 μs	4,3 μs	5 μs	5,8 μs	9 μs	10 μs
n	10 μs	20 μs	40 μs	60 μs	0,5 ms	1 ms
$n\, log_2\, n$	33 μs	86 μs	0,2 ms	0,35 ms	4,5 ms	10 ms
n^2	0,1 ms	0,4 ms	1,6 ms	3,6 ms	0,25 s	1 s
n^3	1 ms	8 ms	64 ms	0,2 s	125 s	17 min
n^4	10 ms	160 ms	2,56 s	13 s	17 hod	11,6 dnů
2^n	1 ms	1 s	12,7 dnů	36000 let	10^{137} let	10^{287} let
$n!$	3,6 s	77000 let	10^{34} let	10^{68} let	10^{1110} let	10^{2554} let

Tab. 5

Druhy složitosti

- Mějme problém *P* a algoritmus *A*, který ho řeší. Řekneme, že algoritmus *A* vyžaduje čas $f(n)$, když pro každé přirozené číslo $n > n_0$ platí, že $f(n)$ je maximum doby běhu algoritmu *A* pro všechny instance problému *P* velikosti nejvýše *n*. Funkci *f* nazýváme složitost v nejhorším případě, tj. hranici, kterou algoritmus *A* nikdy nepřekročí. Číslo n_0 slouží pro odstínění malých dat, kdy se může algoritmus chovat odlišně.
- Horní odhad – funkci *f* není snadné spočítat, proto někdy hledáme horní odhad, tj. funkci *g* takovou, že ji *f* nikdy nedosáhne: $f = \mathrm{O}(g)$.
- Dolní odhad – někdy naopak hledáme dolní odhad.
- Očekávaná složitost – dobu běhu uvažujeme jako náhodnou veličinu, očekávaná složitost je pak její očekávaná hodnota. Není to přesnější odhad, ale lze tak získat realističtější odhad pro velký objem dat.
- Amortizovaná složitost – je to jistý kompromis mezi složitostí v nejhorším případě a očekávanou složitostí. Snaží se odhadnout maximální počet kroků pro provedení posloupnosti operací.

Odhady složitosti

- Nechť f a g jsou funkce na přirozených číslech. Značíme:
  - $f = O(g)$ když $\exists c > 0\ \forall n : f(n) \leq cg(n)$ … (asymptotická horní mez – funkce f nepřesáhne v limitě hodnoty funkce g až na konstantu – g představuje horní mez, kam až f může dosáhnout – odhad složitosti v nejhorším případě)
  - $f = \Omega(g)$ když $\exists c > 0\ \exists$ nekonečně mnoho $n : f(n) > cg(n)$ … (asymptotická dolní mez – funkce f nedosáhne v limitě hodnoty funkce g až na konstantu – g představuje dolní mez, kam už f nemůže dosáhnout – odhad složitosti v nejlepším případě)
  - $f = \Theta(g)$ když $\exists c,d > 0\ \forall n : dg(n) \leq f(n) \leq cg(n)$ … (asymptotická těsná mez – funkce f a g jsou až na konstantu stejné – odhad složitosti v průměrném případě)
- Převážně budeme používat O, protože chceme hlavně horní odhady, kdežto dolní odhady bývá obvykle těžší zjistit.
- U složitostí se bavíme o funkcích z N do N. Reálné funkce konvertujeme na funkce $N \rightarrow N$ nejčastěji pomocí $\lfloor\ \rfloor$ a $\lceil\ \rceil$.

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.

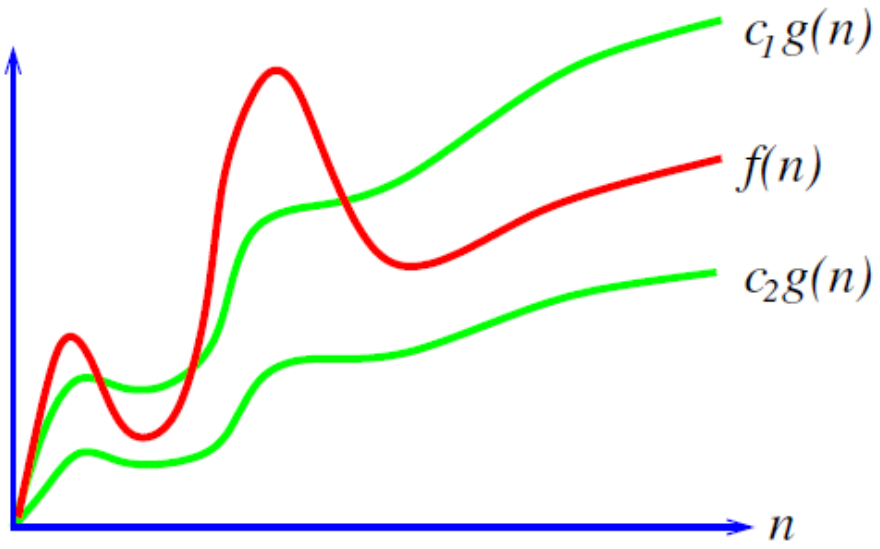
Definice

(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.

Příklad: Ověření složitosti dle definice

- Ověřme, že: $\frac{1}{2}n^2 - 3n = \Theta(n^2)$
- Z definice: $c_1 n^2 \leq \frac{1}{2}n^2 - 3n \leq c_2 n^2$ po úpravě: $c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$
- A stačí tedy (pro $n \geq n_0 = 7$): $c_1 \leq 1/14$ $c_2 \geq 1/2$
- Ověřme, že: $6n^3 \neq \Theta(n^2)$
- Z definice: $6n^3 \leq c_2 n^2$ po úpravě: $n \leq c_2/6$
- A to pro všechna libovolně velká n jistě neplatí.

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

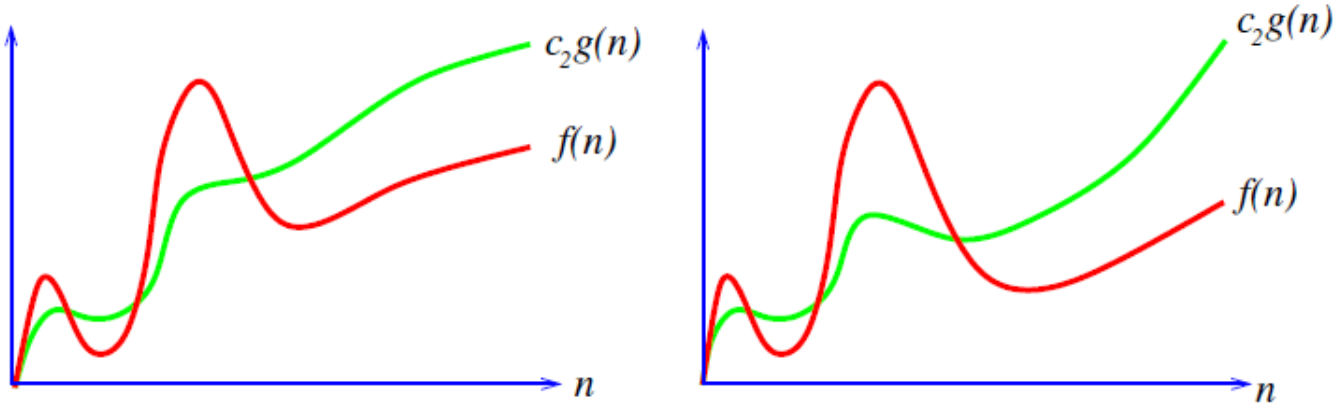
Alternativní definice.

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

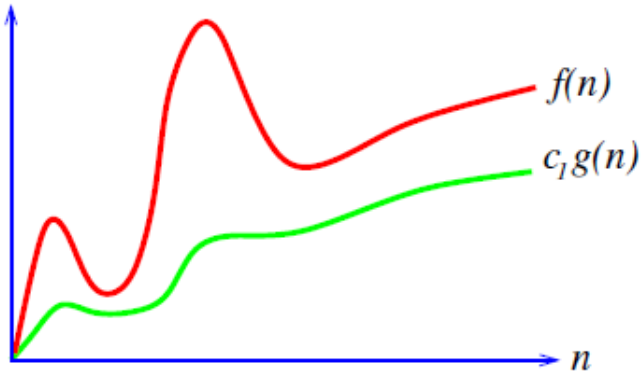
Alternativní definice.

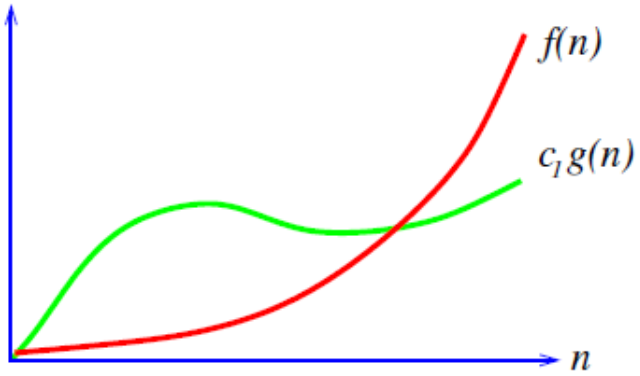
Definice

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

Striktní odhady složitosti

- Nechť f a g jsou funkce na přirozených číslech. Značíme:
  - $f = o(g)$ když $\exists n_0 \in \text{Nat}\ \forall n \geq n_0 : f(n) < c \cdot g(n)$ … (striktní asymptotická horní mez – funkce f nepřesáhne v limitě hodnoty funkce g až na konstantu – g představuje horní mez, kam až f může dosáhnout – odhad složitosti v nejhorším případě)
  - $f = \omega(g)$ když $\exists n_0 \in \text{Nat}\ \forall n \geq n_0 : c \cdot g(n) < f(n)$ … (striktní asymptotická dolní mez – funkce g je pro f asymptotickou mezí – žádná multiplikativní konstanta neumožní, aby g(n) dostihlo f(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 $f(n)$.

Zákony asymptotické aritmetiky

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
		$\approx$		
$\leq$	$\geq$	$=$	$<$	$>$

Použití asymptotické notace

- Asymptotická notace nám umožnuje zjednodušovat výrazy zanedbáním nepodstatných částí.
- Co znamená zápis $f(n) = 4n^3 + \Theta(n^2)$?
  - Výraz $\Theta(n^2)$ 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.

Řád růstu funkcí

Porovnání rychlosti růstu funkcí

Funkce f(x) roste asymptoticky rychleji než funkce g(x), když

$f(x) \in \Omega(g(x)) \ \& \ f(x) \notin \Theta(g(x))$

Pozor!

Porovnání rychlosti algoritmů

Algoritmus A je asymptoticky pomalejší než algoritmus B, když

$f_A(n) \in \Omega(f_B(n)) \ \& \ f_A(n) \notin \Theta(f_B(n))$,

kde $f_A(n)$, resp. $f_B(n)$ je funkce určující počet operací,
které provede algoritmus A, resp. B,
při zpracování dat o rozsahu n.

Řád růstu funkcí

Řád růstu funkce

Řád růstu funkce f je taková „co nejjednodušší" funkce g, pro kterou platí
$g(x) \in \Theta(f(x))$

Manipulace

Řád růstu funkce f získáme většinou tak, že zanedbáme

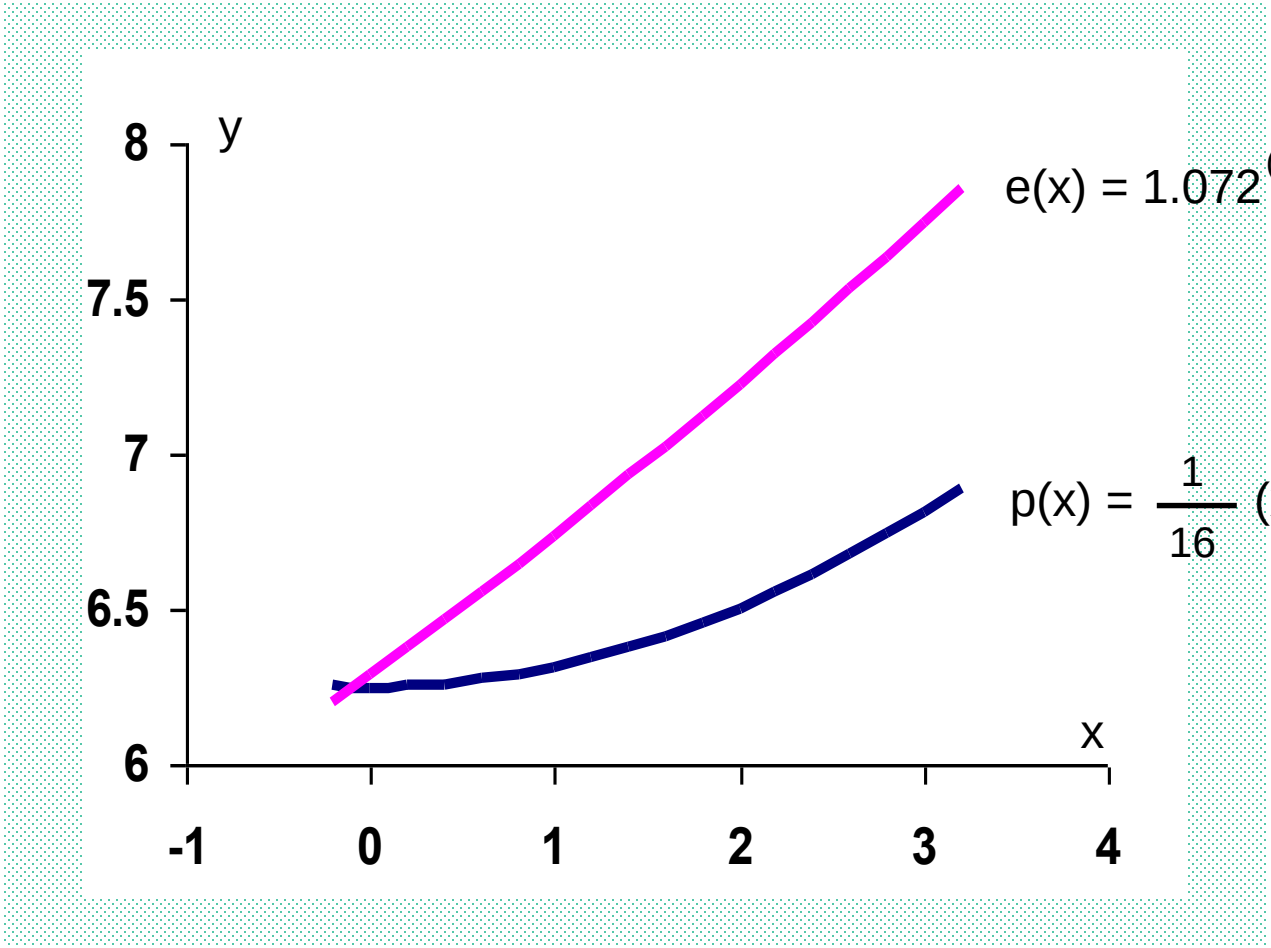
1. Aditivní členy rostoucí pomaleji nebo stejně rychle
2. Multiplikativní konstantu

Příklady

$ff(n) = 4 \cdot 2^n + 3 \cdot 2^{n-1} + 5 \cdot 2^{n/2} \in \Theta(2^n)$ Řád růstu ff(n) je 2^n

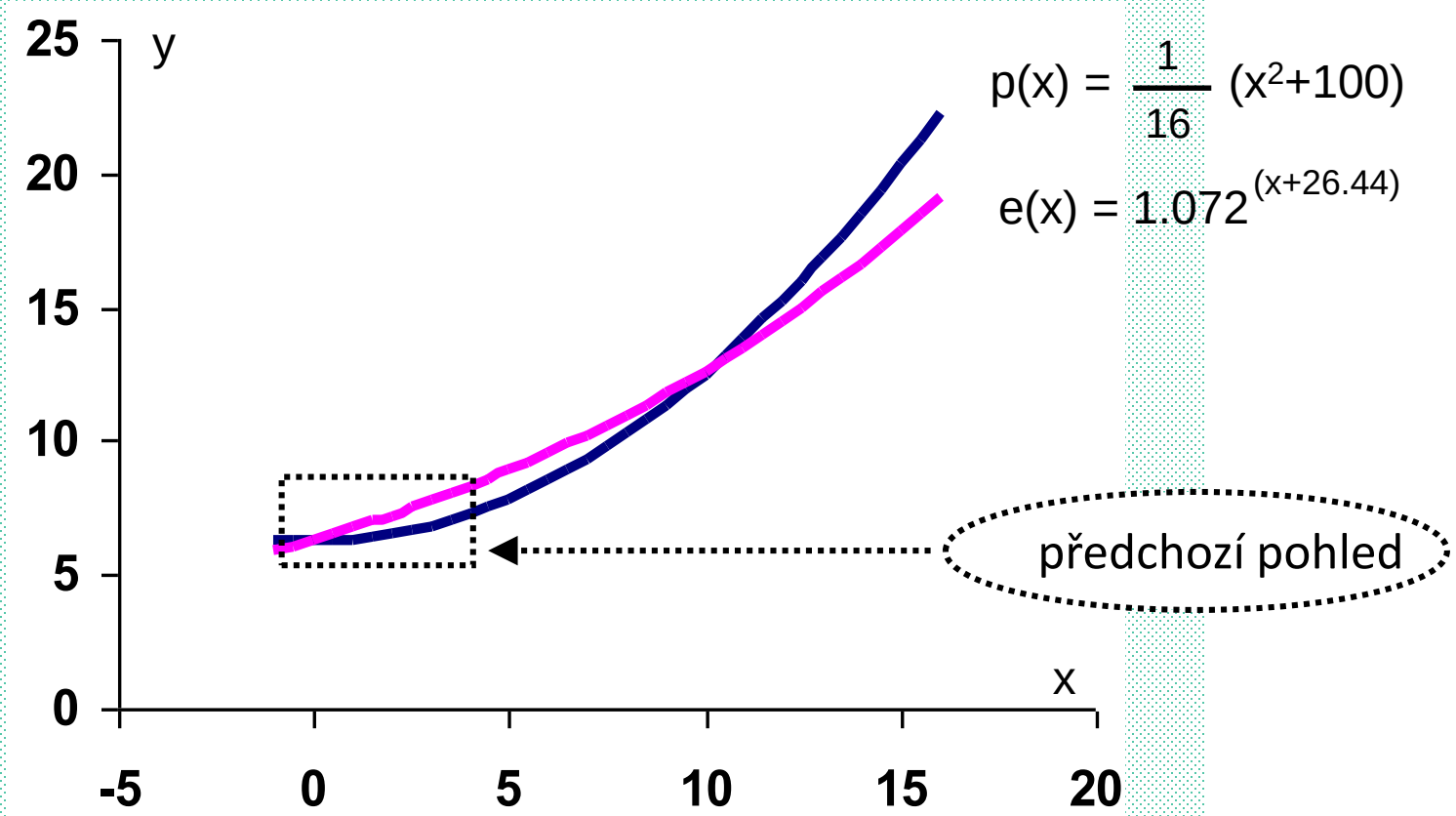
$hh(x) = x + \log_2(x) - \sqrt{x} \in \Theta(x)$ Řád růstu hh(x) je x

Řád růstu funkcí



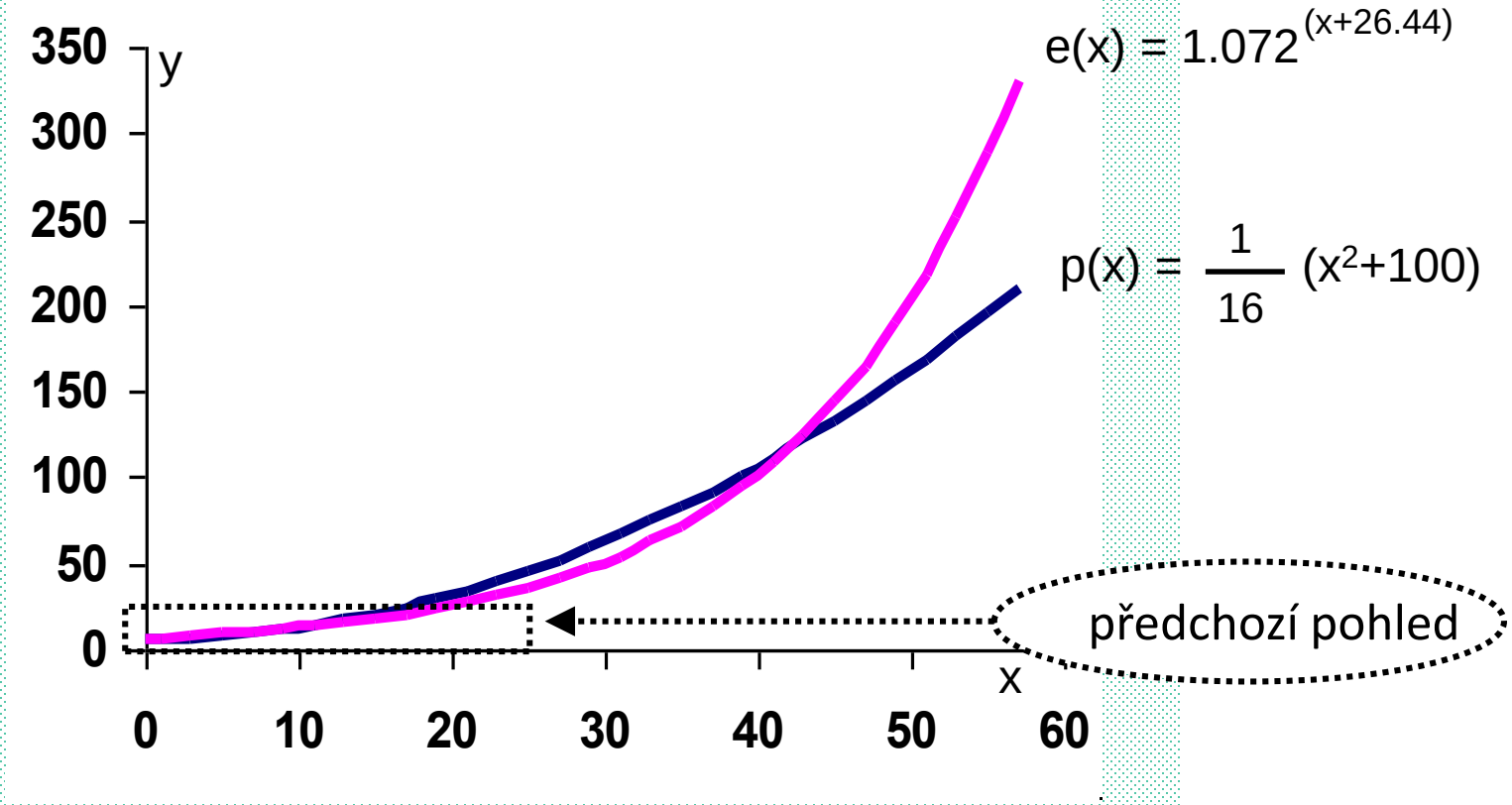
Řád růstu funkcí

Zoom out! :



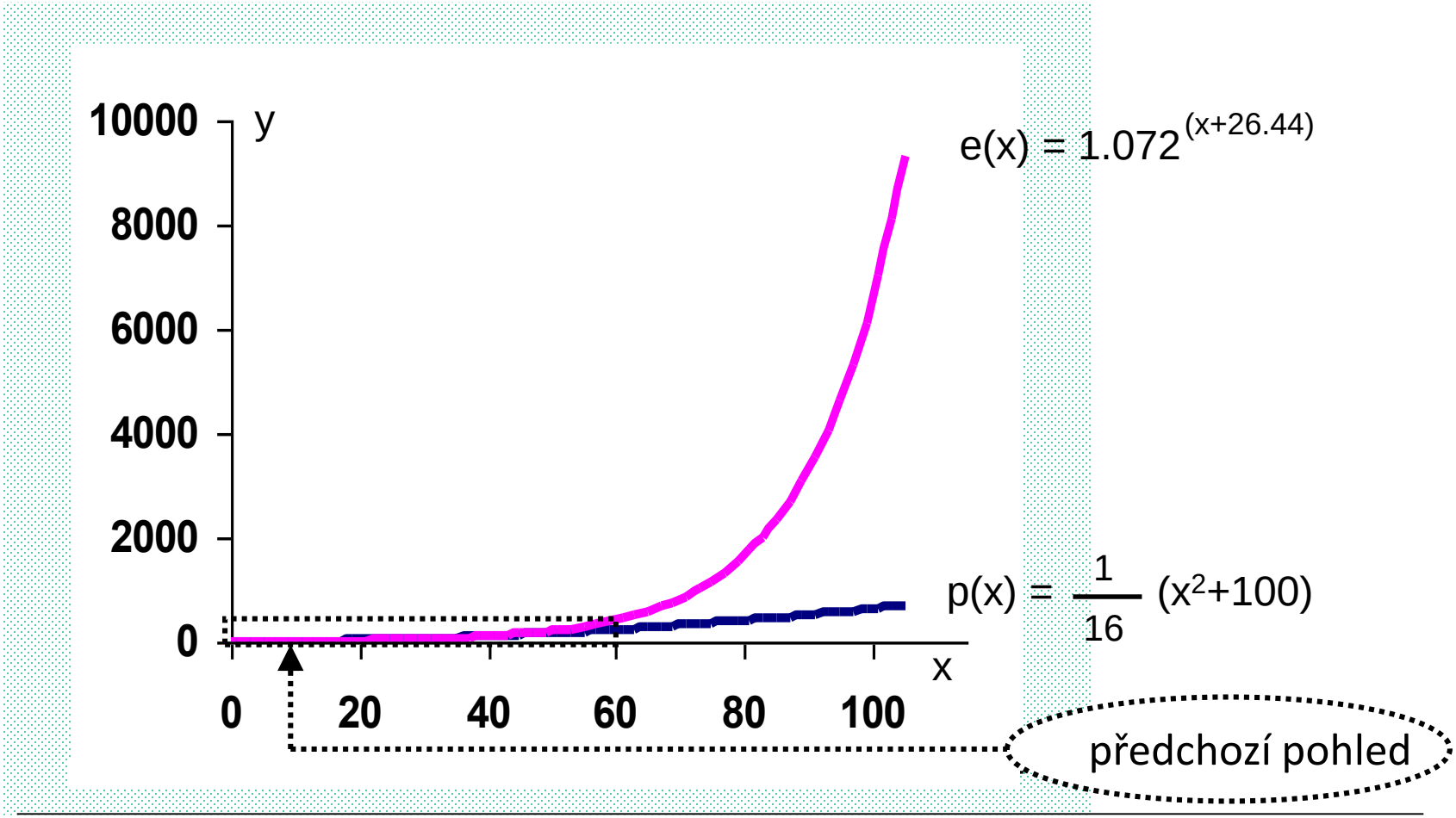
Řád růstu funkcí

Zoom out! :



Řád růstu funkcí

Zoom out! :



atd:… e(1000) = 984318123660540890654762870434 2.9 p(1000) = 62506.25 …

Asymptotická složitost

Asymptotická složitost algoritmu

Asymptotická složitost algoritmu A
je řád růstu funkce f(n), která charakterizuje počet elementárních operací algoritmu A
při zpracování dat o rozsahu n.

(rozsah dat = celkový počet čísel a znaků v nich)

Ve většině případů nehraje roli, zda uvažujeme

A) počet všech elementárních operací
B) počet všech elementárních operací nad daty
C) počet testů nad daty

Asymptotická složitost vychází táž.

Asymptotická složitost

Asymptotická složitost předložených ukázek

Asymptotická složitost hledání minima v poli o n prvcích je n.
V obou uvedených případech.

Asymptotická složitost pomalého zjišťování, kolik čísel v poli je rovno součtu jiného pole, je n^2.

Asymptotická složitost rychlého zjišťování, kolik čísel v poli je rovno součtu jiného pole, je n.

Za předpokladu, že obě pole mají délku n.

Asymptotická složitost lineárního hledání prvku v poli je n.

Asymptotická složitost hledání prvku uspořádaném poli pomocí půlení intervalu je $\log(n)$.

Za předpokladu, že pole má délku n.

Asymptotická složitost

Úmluvy

Zjednodušení

Běžně se zkráceným termínem „složitost algoritmu“
rozumí právě výraz „asymptotická časová složitost algoritmu“

Zmatení

Běžně se v literatuře neříká $f(x)$ náleží do $\Theta(g(x))$,
ale $f(x)$ je $\Theta(g(x))$.
Rovněž se to tak značí: $f(x) = \Theta(g(x))$
namísto $f(x) \in \Theta(g(x))$
(a stejně pro O a Ω).

Chápe se to ale beze změny, v původním smyslu náležení.

Asymptotická složitost

$\in \Theta($ $)$

$\in \Theta($ $)$

$\in \Omega($ $)$

$\in O($ $)$

Složitost rekurzivních algoritmů

T(n) -- asymptotická (časová) složitost algoritmu při vstupu o velikosti n

Př.: **MERGESORT**

Asymptotická složitost vyřešení triviální úlohy

$$T(n) = \begin{cases} \Theta(1) & \text{pro } n = 1 \\ 2T(n/2) + \Theta(n) & \text{pro } n > 1 \end{cases}$$

Jak asymptotická složitost při vstupu o velikosti n závisí na asymptotické složitosti při vstupu o velikosti n/2

Složitost rozdělení problému a spojení dílčích řešení (polovin pole v Merge sortu)

Složitost rekurzivních algoritmů

Co lze zanedbat

Typickou hodnotu n si vhodně zvolíme
(v Merge sortu mocninu 2)

Konkrétní konstanta neovlivní výslednou asymptotickou složitost

$$T(n) = \begin{cases} \Theta(1) & \text{pro } n = 1 \\ 2T(n/2) + \Theta(n) & \text{pro } n > 1 \end{cases}$$

n/2 a obecně n/konst není celé číslo, mysleme si však, že (víceméně) je,
a použijme jej místo správného $\lceil n/2 \rceil$ či $\lfloor n/2 \rfloor$ apod.
Většinou výsledek nebude ovlivněn.

Složitost rekurzivních algoritmů

Příklad

Pro algoritmus A platí

$$T(n) = \begin{cases} \Theta(1) & \text{pro } n = 1 \\ 3T(\lfloor n/4 \rfloor) + \Theta(n^2) & \text{pro } n > 1 \end{cases}$$

Rozdělí data na čtvrtiny. Vyřešení „čtvrtinové“ úlohy trvá $T(\lfloor n/4 \rfloor)$.

Jedna čtvrtina se nezpracovává*) $\Rightarrow$ tři čtvrtiny se zpracují v čase $3T(\lfloor n/4 \rfloor)$.

*) z důvodů zde nepodstatných :-)

Čas potřebný na rozdělení na čtvrtiny
a na spojení „čtvrtinových“ řešení je $\Theta(n^2)$.

Složitost rekurzivních algoritmů

Příklad

$$T(n) = \begin{cases} \Theta(1) & \text{pro } n = 1 \\ 3T(\lfloor n/4 \rfloor) + \Theta(n^2) & \text{pro } n > 1 \end{cases}$$

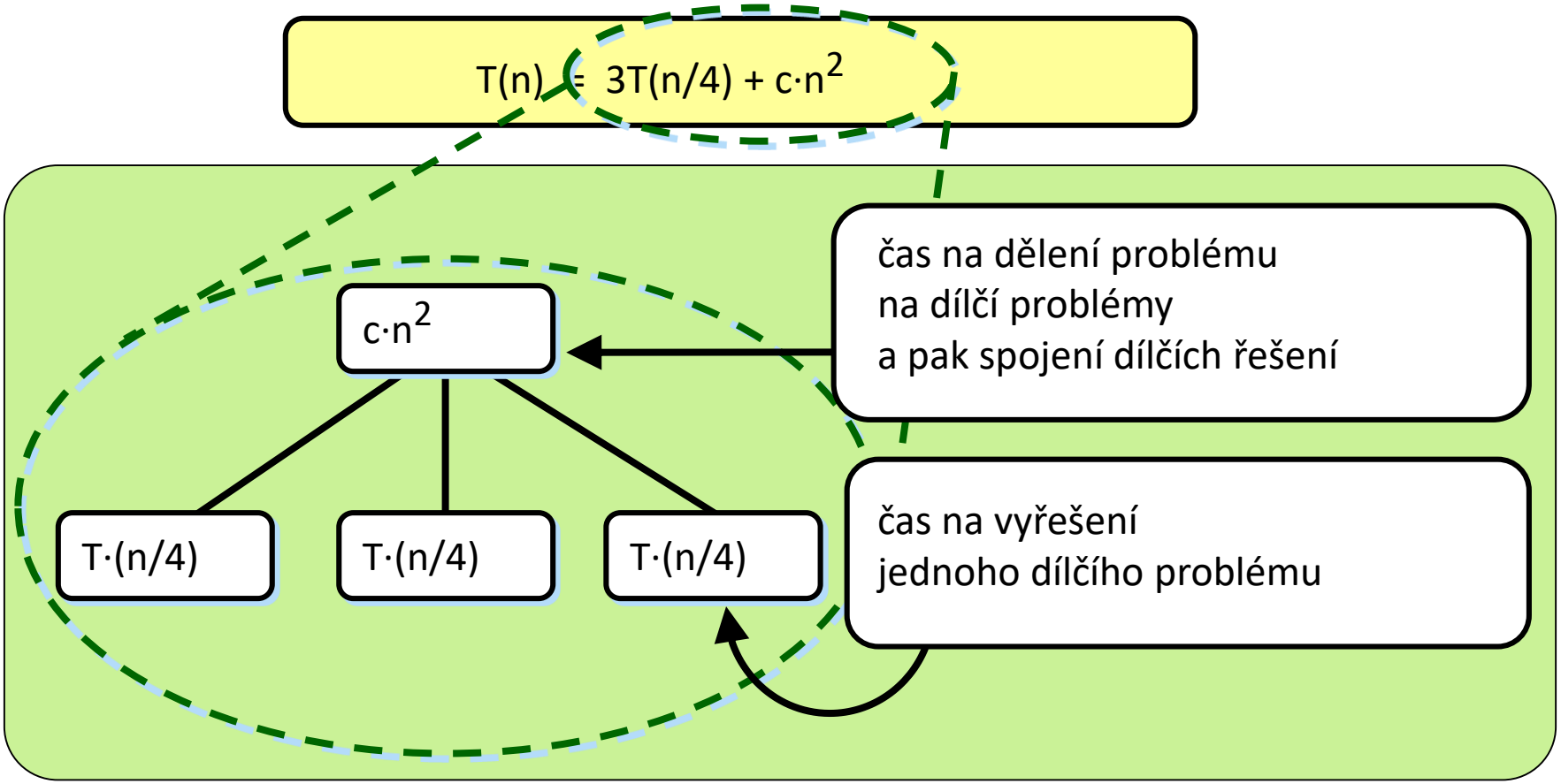
Zanedbáme celé části,
Θ nahradíme úměrou

Vztah pro výpočet

$$T(n) = 3T(n/4) + c \cdot n^2$$

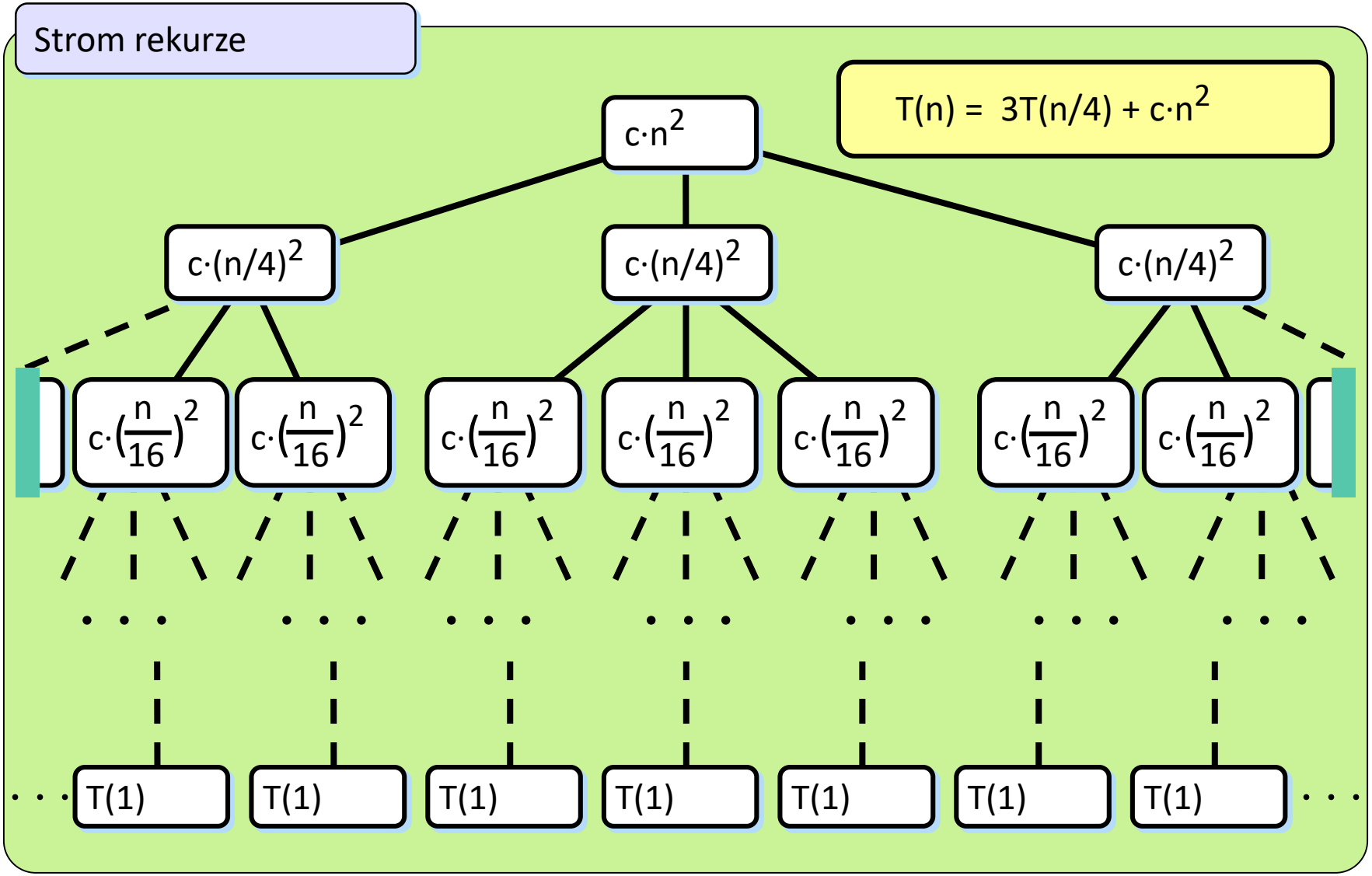
STROM REKURZE

Strom rekurze



Strom rekurze je ale větší ...

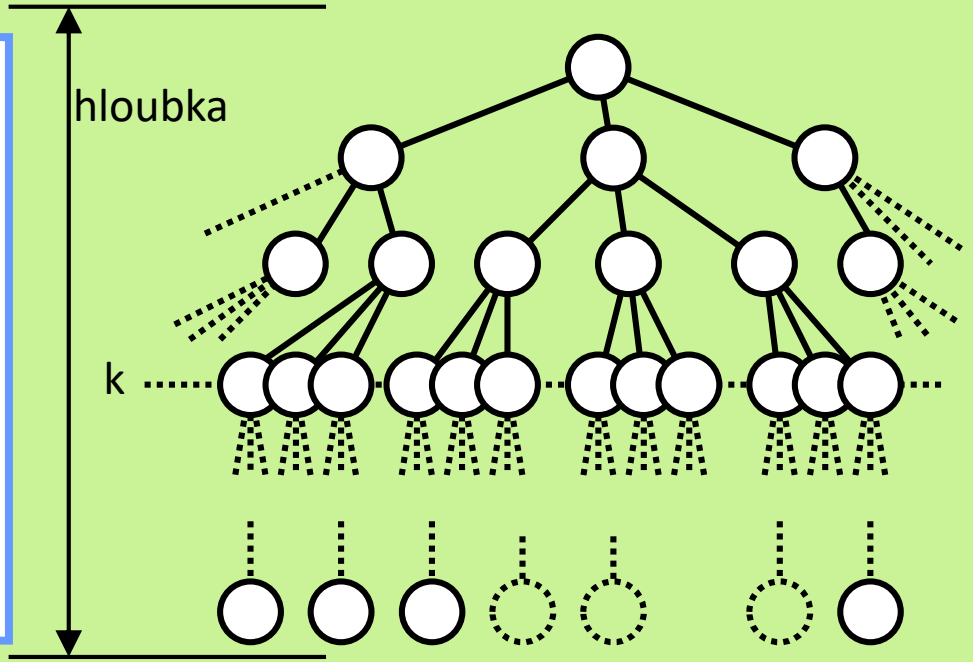
Strom rekurze



Strom rekurze

Průběh výpočtu

1. Nakresli strom rekurze
2. Spočti jeho hloubku
3. Spočti jeho šířku v patře k
4. Spočti cenu uzlu v patře k
5. Sečti ceny uzlů v patře k
6. Sečti ceny všech pater



cena uzlu = asymptotická složitost zpracování podproblému odpovídajícího uzlu ve stromu rekurze.

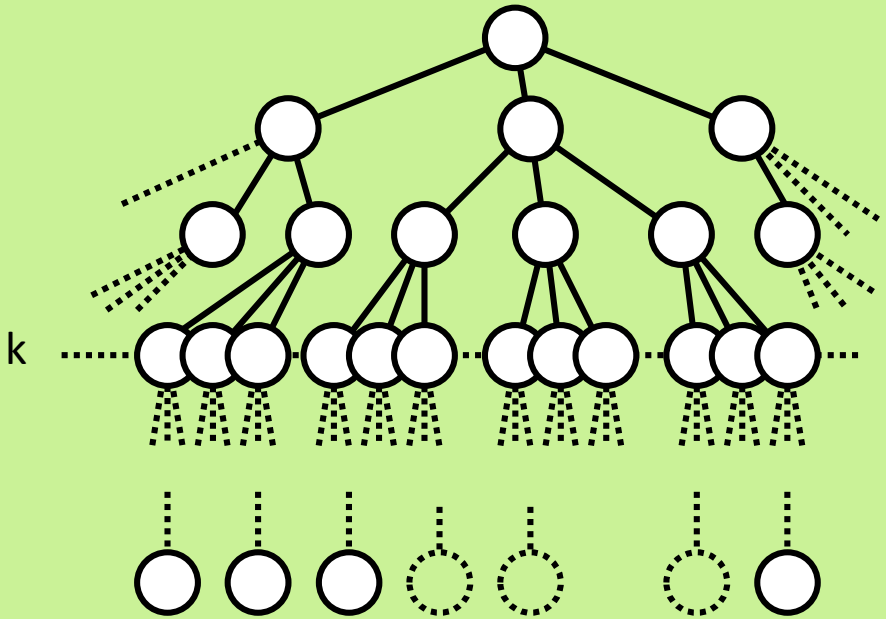
cena stromu = asymptotická složitost zpracování celé úlohy.

Strom rekurze

$T(n) = 3T(n/4) + c \cdot n^2$

Průběh výpočtu

1. Nakresli strom rekurze
2. Spočti jeho hloubku
...



V hloubce k
je velikost podproblému $n/4^k$.

Velikost podproblému je tedy = 1, když $n/4^k = 1$, tj $k = \log_4(n)$.

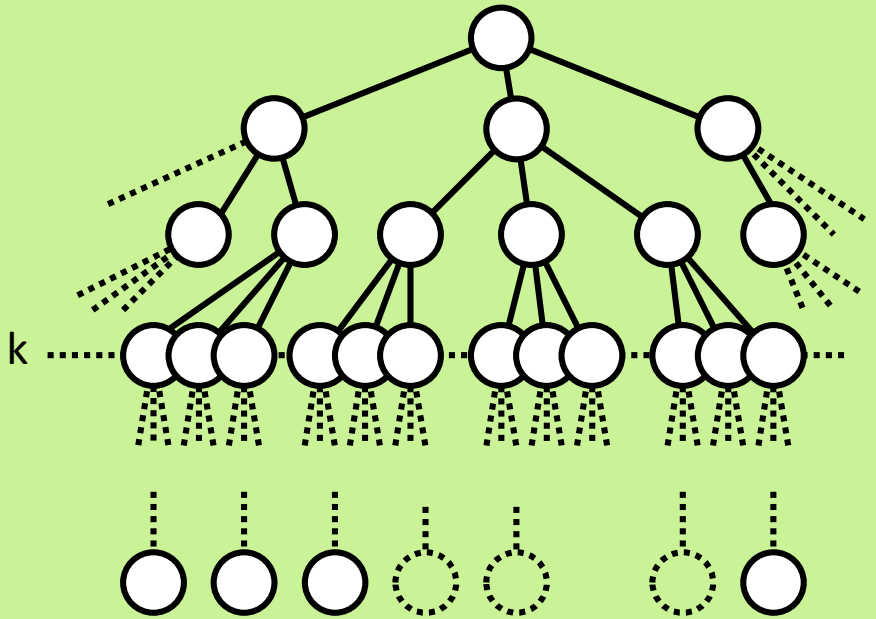
hloubka stromu

Takže strom má $\log_4(n) + 1$ pater (hloubka kořene je k=0).

Strom rekurze

$T(n) = 3T(n/4) + c \cdot n^2$

Průběh výpočtu
...
3. Spočti jeho šířku v patře k
...



0. patro – 1 uzel
1. patro – 3 uzly
2. patro – 3·3 = 9 uzlů
3. patro – 3·3·3 = 27 uzlů
...

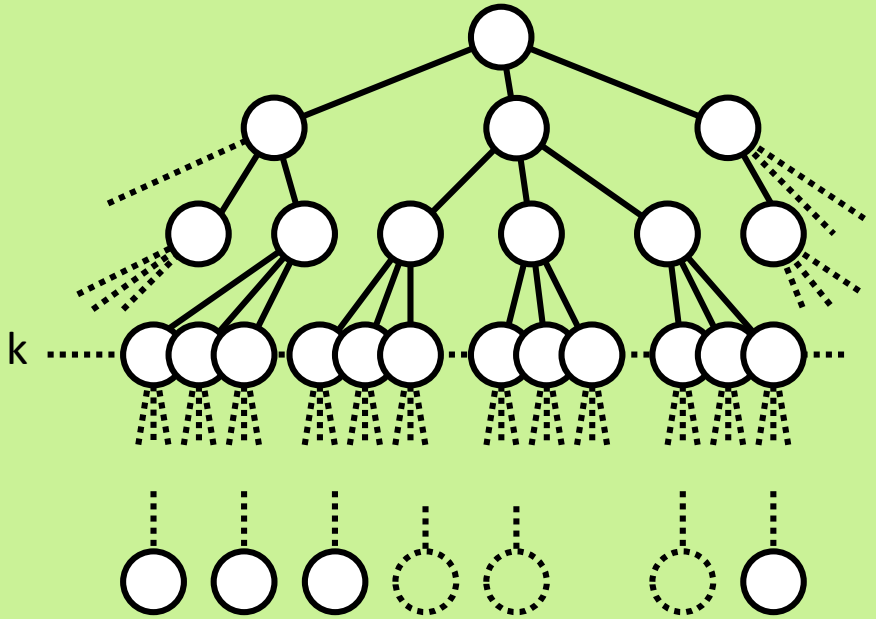
počet uzlů v jednom patře

k. patro – $3 \cdot 3 \cdot \ldots \cdot 3 \cdot 3 = 3^k$ uzlů

Strom rekurze

Průběh výpočtu
...
4. Spočti cenu uzlu v patře k
...

$T(n) = 3T(n/4) + c \cdot n^2$



0. patro – $c \cdot n^2$
1. patro – $c \cdot (n/4)^2$
2. patro – $c \cdot (n/16)^2$
3. patro – $c \cdot (n/64)^2$
...

cena uzlu v patře k

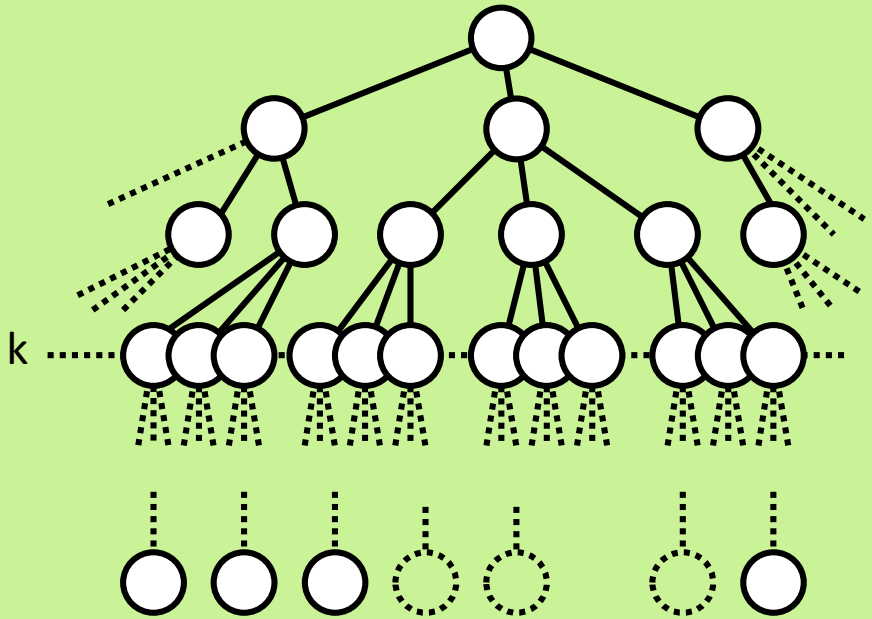
k. patro – $c \cdot (n/4^k)^2$

Strom rekurze

$T(n) = 3T(n/4) + c \cdot n^2$

Průběh výpočtu
...
5. Sečti ceny uzlů v patře k
...

V patře k je 3^k uzlů,
každý má cenu $c \cdot (n/4^k)^2$.



celková cena patra

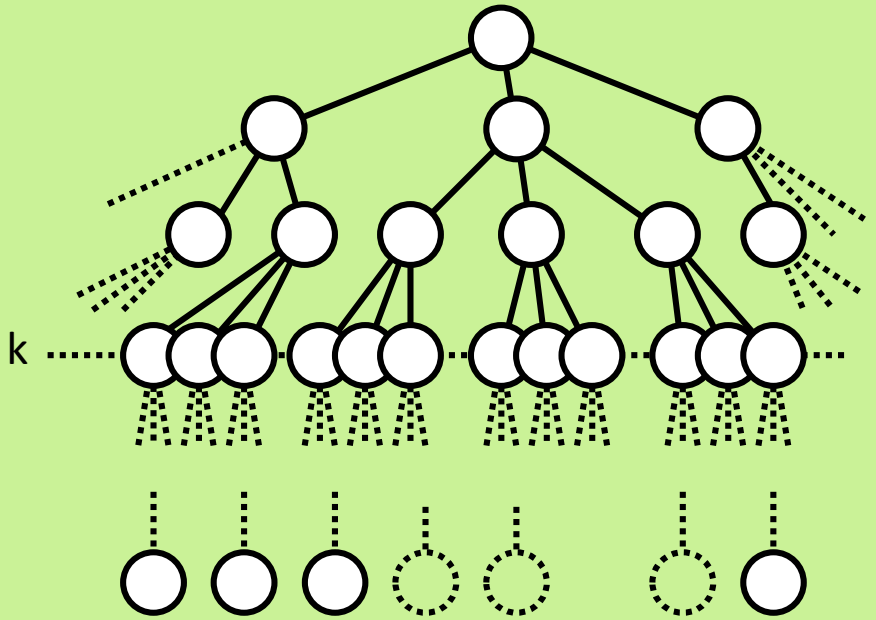
$3^k \cdot c \cdot (n/4^k)^2 = (3/16)^k \cdot c \cdot n^2$

Pozor na poslední patro:

Strom rekurze

$T(n) = 3T(n/4) + c \cdot n^2$

Průběh výpočtu
...
5. Sečti ceny uzlů v patře k
...



Poslední patro je v hloubce $\log_4(n)$ a má tedy

$3^{\log_4(n)} = n^{\log_4(3)}$ uzlů.

cena posledního patra

Každý uzel v posledním patře přispívá konstantní cenou, takže cena posledního patra je

$n^{\log_4(3)} \cdot konst = \Theta(n^{\log_4(3)})$

Strom rekurze

Průběh výpočtu
...
6. Sečti ceny všech pater

$T(n) = 3T(n/4) + c \cdot n^2$

Celková cena =

$cn^2 + 3/16 \cdot cn^2 + (3/16)^2 \cdot cn^2 + \ldots + (3/16)^{\log_4(n-1)} \cdot cn^2 + \Theta(n^{\log_4(3)}) =$

$(1 + 3/16 \cdot + (3/16)^2 + \ldots + (3/16)^{\log_4(n-1)}) \cdot cn^2 + \Theta(n^{\log_4(3)}) \doteq$

Geometrickou posloupnost nahradíme přibližně geometrickou řadou (zbytek řady je zanedbatelný). Získáváme horní odhad součtu.

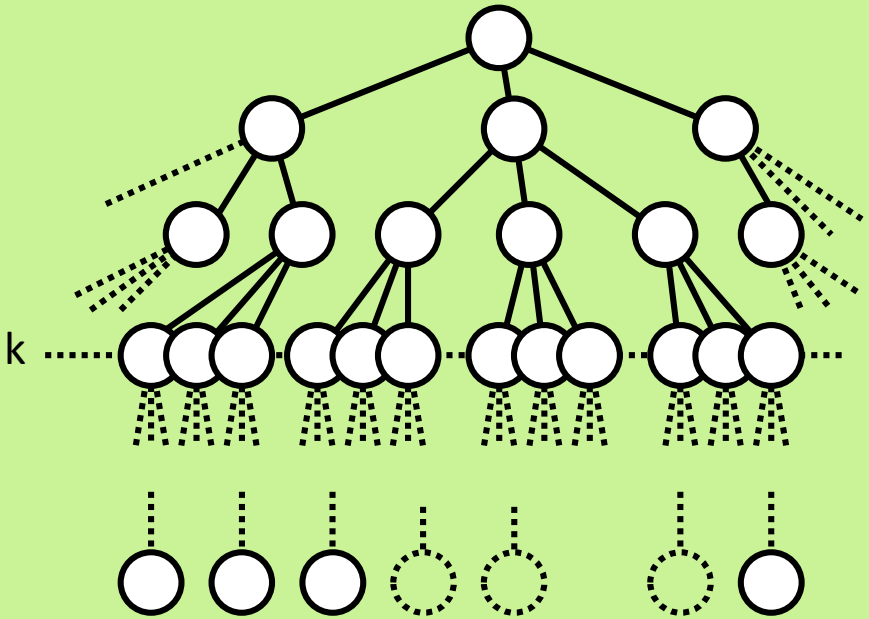
$(1 + 3/16 \cdot + (3/16)^2 + (3/16)^3 + \ldots \text{ ad inf. }) \cdot cn^2 + \Theta(n^{\log_4(3)}) =$

$(1 / (1-3/16)) \cdot cn^2 + \Theta(n^{\log_4(3)}) = 16/13 \cdot cn^2 + \Theta(n^{\log_4(3)}) =$

Strom rekurze

Průběh výpočtu
...
6. Sečti ceny všech pater

$T(n) = 3T(n/4) + c \cdot n^2$



Asymptotická složitost algoritmu A

$2 > \log_4(3) \Rightarrow$

$= 16/13 \cdot cn^2 + \Theta(n^{\log_4(3)}) = \Theta(n^2)$

SUBSTITUČNÍ METODA

Substituční metoda

Příklad

Rekurentní vztah popisující asymptotickou složitost algoritmu B

$$T(n) = 2T(\lfloor n/2 \rfloor) + n$$

Náš odhad složitosti

$$T(n) = O(n \cdot \log_2(n))$$

Zdroj odhadu: zkušenost, podobnost s jinými úlohami
úvaha, intuice ... :-)

Chceme dokázat: Náš odhad platí

Metoda: Běžná matematická indukce, do níž dosadíme (substituujeme) daný rekurentní vztah

Substituční metoda

Chceme dokázat : $T(n) = O(n \cdot \log_2(n))$,
to jest : $T(n) \leq c \cdot n \cdot \log_2(n)$, pro vhodné $c > 0$

Krok II (obecný krok) matematické indukce:

Dokážeme, že pokud nerovnost platí pro $\lfloor n/2 \rfloor$, platí i pro n.

Víme: $T(n) = 2T(\lfloor n/2 \rfloor) + n$

Předpokládáme: $T(\lfloor n/2 \rfloor) \leq c \cdot \lfloor n/2 \rfloor \cdot \log_2(\lfloor n/2 \rfloor)$

Substituce: $T(n) \leq 2 \cdot c \cdot \lfloor n/2 \rfloor \cdot \log_2(\lfloor n/2 \rfloor) + n$

úpravy: $\leq cn \cdot \log_2(n/2) + n = cn \cdot \log_2(n) - cn \cdot \log_2(2) + n$

$= cn \cdot \log_2(n) - cn + n$

$\leq cn \cdot \log_2(n)$, pokud $c \geq 1$

Substituční metoda

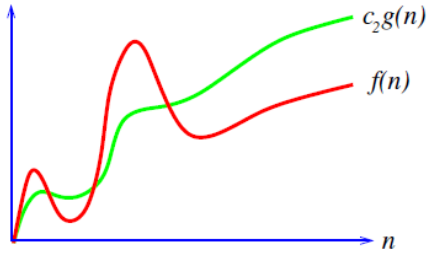
Krok I matematické indukce

Nerovnost $T(n) \leq c \cdot n \cdot \log_2(n)$, platí pro nějaké konkrétní malé n.

Potíž

Nelze dokazovat pro n = 1,

neboť bychom dokazovali $T(1) \leq c \cdot 1 \cdot \log_2(1) = 0$, což neplatí, protože jistě je $T(1) > 0$.



Pozorování

$T(n) = 2T(\lfloor n/2 \rfloor) + n$

Pokud n > 3, v rekurentním vztahu se T(1) neobjeví,
tedy pokud dokážeme indukční krok I pro n = 2 a n = 3,
je důkaz hotov pro všechna $n \geq 2$.

Řešení

Jde nám ale o asymptotickou složitost, tudíž **důkaz pro $n \geq 2$ stačí.**

Substituční metoda

Krok I matematické indukce pro n = 2 a n = 3

Ať T(1) = k.

$T(n) = 2T(\lfloor n/2 \rfloor) + n$

Z rekurentního vztahu plyne
T(2) = 2k+2
T(3) = 2k+3

Chceme mít:
$T(2) \leq c \cdot 2 \cdot \log_2(2)$
$T(3) \leq c \cdot 3 \cdot \log_2(3)$

Stačí tedy volit $c \geq \max \{ (2k+2)/2,\ (2k+3)/(3 \cdot \log_2(3)) \}$.

Tudíž, kdyby k bylo např. 10, stačilo by volit
$c \geq \max \{ (2 \cdot 10 + 2)/2,\ (2 \cdot 10 + 3)/(3 \cdot \log_2(3)) \} = \max \{11, 4.84 \} = 11.$

MISTROVSKÁ METODA (THE MASTER METHOD)

Mistrovská metoda

Věta

Nechť $a \geq 1$ a $b > 1$ jsou konstanty, $f(n)$ je funkce a nechť asymptotická složitost daného algoritmu je definována rekurentním vztahem

$$T(n) = aT(n/b) + f(n),$$

kde podíl n/b lze libovolně interpretovat jako $\lfloor n/b \rfloor$ nebo $\lceil n/b \rceil$.

Potom platí

1. Pokud $f(n) = O(n^{\log_b(a) - e})$ pro nějakou konstantu $e > 0$, pak
   $T(n) = \Theta(n^{\log_b(a)})$.
2. Pokud $f(n) = \Theta(n^{\log_b(a)})$, pak
   $T(n) = \Theta(n^{\log_b(a)} \cdot \log_2(n))$.
3. Pokud $f(n) = \Omega(n^{\log_b(a) + e})$ pro nějakou konstantu $e > 0$, a pokud $a \cdot f(n/b) \leq c \cdot f(n)$ pro pro nějaké $c < 1$ a pro všechna dostatečně velká n, pak
   $T(n) = \Theta(f(n))$.

Mistrovská metoda

Komentáře k podmínkám věty

1. Pokud $f(n) = O(n^{\log_b(a) - e})$ pro nějakou konstantu $e > 0$, pak

$$T(n) = \Theta(n^{\log_b(a)}).$$

/ Podmínka 1. říká, že f(n) musí růst polynomiálně pomaleji
/ než funkce $n^{\log_b(a)}$.
/

3. Pokud $f(n) = \Omega(n^{\log_b(a) + e})$ pro nějakou konstantu $e > 0$,
a pokud $a \cdot f(n/b) \leq c \cdot f(n)$ pro pro nějaké $c < 1$
a pro všechna dostatečně velká n, pak

$$T(n) = \Theta(f(n)).$$

/ Podmínka 3. říká, že f(n) musí růst polynomiálně rychleji
/ než funkce $n^{\log_b(a)}$.
/

Mistrovská metoda

...

1. Pokud $f(n) = O(n^{\log_b(a) - e})$ pro nějakou konstantu $e > 0$, pak

$T(n) = \Theta(n^{\log_b(a)})$.

...

Příklad.

$T(n) = 9T(n/3) + n$

$a = 9, b = 3, f(n) = n.$

Platí $n^{\log_b(a)} = n^{\log_3(9)} = \Theta(n^2)$

$f(n) = O(n^{\log_3(9) - e})$, kde $e = 1$

Celkem tedy $T(n) = \Theta(n^2)$

Mistrovská metoda

...

3. Pokud $f(n) = \Omega(n^{\log_b(a)+e})$ pro nějakou konstantu $e > 0$, a pokud $a \cdot f(n/b) \leq c \cdot f(n)$ pro pro nějaké $c < 1$ a pro všechna dostatečně velká n, pak

$T(n) = \Theta(f(n))$.

Příklad.

$T(n) = 2T(n/2) + n \cdot \log_2(n)$

$a = 2$, $b = 2$, $f(n) = n \cdot \log_2(n)$.

Platí $n^{\log_b(a)} = n$

$f(n) = n \cdot \log_2(n)$ roste asymptoticky rychleji než $n^{\log_b(a)} = n$.

Pozor, neroste ale polynomiálně rychleji. Poměr $n \cdot \log_2(n) / n = \log_2(n)$ roste pomaleji než každá rostoucí polynomiální funkce.

$n \cdot \log2(n) \notin \Omega(n^{1+e})$ pro každé kladné e. Předpoklad 3. není splněn.

Diskuze k mistrovské metodě

- Rovnice *T(n) = aT(n/b) + f(n)* odpovídá rekurzivnímu algoritmu, který dělí problém o velikosti *n* na *a* částí o velikosti *n/b* a na dělení a zpětné kombinování výsledku potřebuje *f(n)* času.
- Ve všech 3 případech se *f(n)* srovnává s $n^{\log_b a}$ a řešení je dáno větším z nich:
  - *f(n)* je polynomiálně menší než $n^{\log_b a}$ která se nakonec prosadí.
  - Obě funkce jsou stejného řádu a proto výsledkem je:
    $$\Theta(f(n)\log n) = \Theta(n^{\log_b a}\log n)$$
  - Opak případu (1) - *f(n)* je polynomiálně větší než $n^{\log_b a}$ prosadí se *f*.
- Tyto 3 případy nepokrývají všechny možnosti.
- Pokud je rozdíl mezi funkcemi menší než polynomiální, nelze tuto metodu použít!!!
- Např. rovnice *t(n) = 2t(n/2) + n log n* není mistrovskou metodou řešitelná.

Příklad: MERGE-SORT

- Rovnice: $T(n) = 2\,T(n/2) + n$

  $a = 2$

  $b = 2$

  $f(n) = n = n^{\lg 2} = \Theta(n)$

- *Jedná se tedy o případ, kdy jsou funkce stejného řádu a dle mistrovské věty platí:*

$$\Theta(f(n)\log n) = \Theta(n^{\log_b a}\log n)$$

$T(n) = \Theta(n \lg n)$

The End