Algoritmy a datové struktury

Petr Švarný 2022

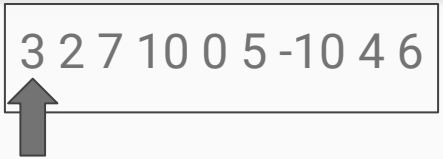
Algoritmy lze psát různě

Úloha: Min/max v poli

MAX: 3

MIN: 3

```
max = pole[0]

min = pole[0]
```



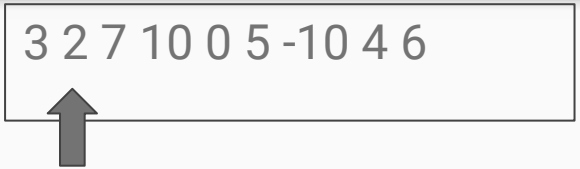

```
for n in pole:

    if n < min: min = n

    if n > max: max = n
```

Úloha: Min/max v poli

MAX: 3

MIN: 3

```
max = pole[0]

min = pole[0]
```




```
for n in pole:

    if n < min: min = n

    if n > max: max = n
```

MAX: 3

MIN: 2

Min/max vyhodnocení

Kolik kroků bude potřebovat náš Min/max algoritmus na zpracování pole o délce n:

A) To nevíme, záleží na HW na kterém to běží.
B) Bude to n kroků.
C) Záleží na tom, kde bude v poli min a max.

Min/max vyhodnocení

Kolik kroků bude potřebovat náš Min/max algoritmus na zpracování pole o délce n:

A) To nevíme, záleží na HW na kterém to běží.
B) Bude to n kroků.
C) Záleží na tom, kde bude v poli min a max.

Úloha: Min/max v poli, ale rychleji

MAX: 3

MIN: 3

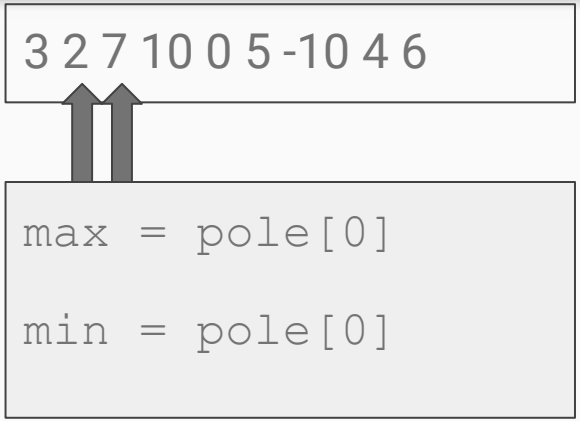
```
max = pole[0]

min = pole[0]
```

3 2 7 10 0 5 -10 4 6

```
?
```

Úloha: Min/max v poli, ale rychleji

```
for i in range(0, len(pole), 2):
    if pole[i] < pole[i+1]:
        if pole[i] < min: min = pole[i]
        if pole[i+1] > max: max = pole[i+1]
    else:
        if pole[i] > max: max = pole[i]
        if pole[i+1] < min: min = pole[i+1]
```



Jak porovnáme dané dva algoritmy?

- Diskuze

Jak porovnáme dané dva algoritmy?

Původní

Srozumitelnější, přímočařejší

“Po dvou”

Kratší běh skrz pole

Složitější struktura

Tři porovnání na 2 čísla

Musí zohlednit sudost pole

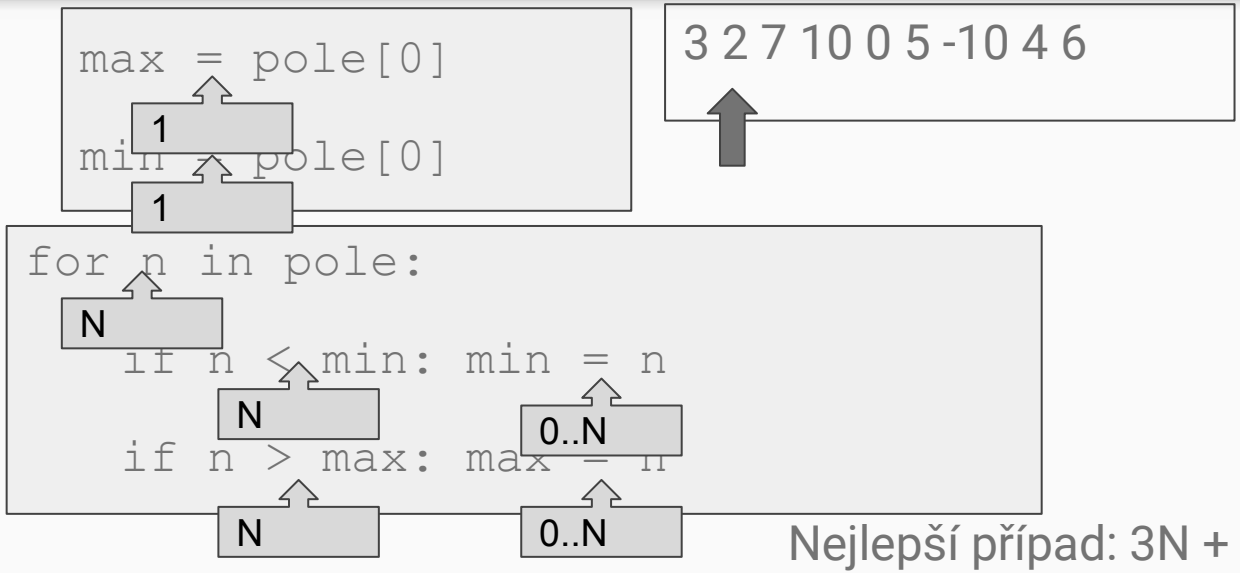
Porovnání

- Nakonec vždy záleží na účelu
- Složitost algoritmu se měří **“elementárními operacemi”**
  - Tedy celkový počet porovnání v datech u min/max a přiřazení, práce s daty ap.
- Pro zjednodušení se berou v potaz jen testy v datech

Složitost: Veškeré operace

MAX: 3

MIN: 3



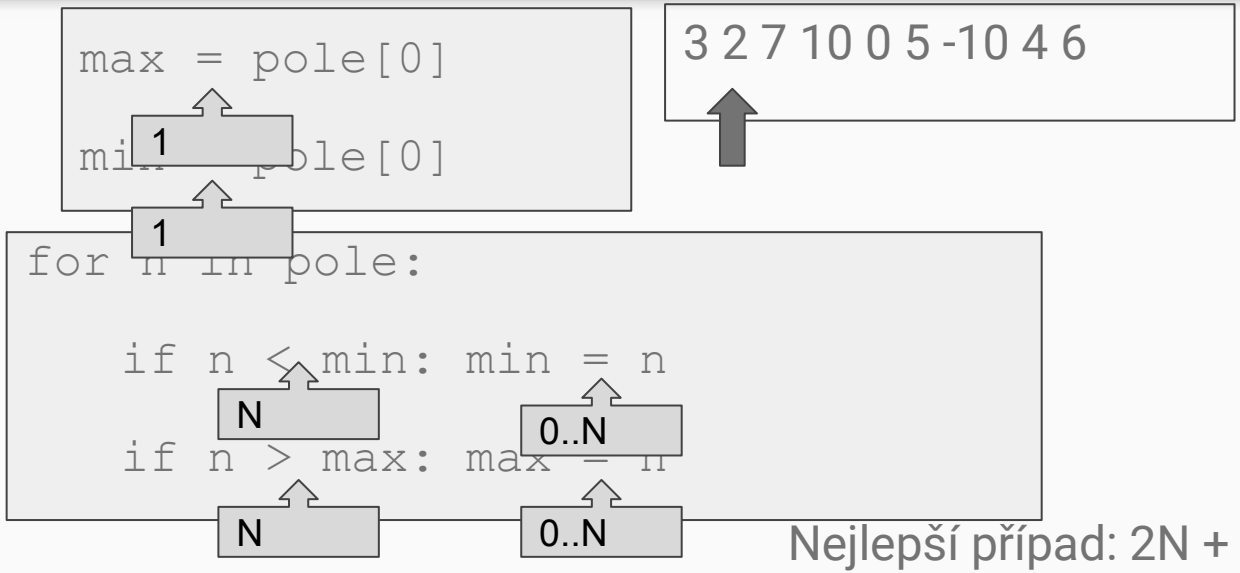
Nejlepší případ: 3N + 2

Nejhorší případ: 5N + 2

Složitost: Operace nad daty

MAX: 3

MIN: 3



Nejlepší případ: 2N + 2

Nejhorší případ: 4N + 2

Složitost: Jen testy v datech

MAX: 3

MIN: 3

```
max = pole[0]

min = pole[0]
```

3 2 7 10 0 5 -10 4 6

```
for n in pole:

    if n < min: min = n
    if n > max: max = n
```

N

N

Nejlepší případ: 2N

Nejhorší případ: 2N

Složitost: Jen testy

```
for i in range(0, len(pole), 2):
    if pole[i] < pole[i+1]:
        if pole[i] < min: min = pole[i]
        if pole[i+1] > max: max = pole[i+1]
    else:
        if pole[i] > max: max = pole[i]
        if pole[i+1] < min: min = pole[i+1]
```

Dvojice = 3 testy
Dvojic v poli: (N-1)/2

3 2 7 10 0 5 -10 4 6

```
max = pole[0]
min = pole[0]
```

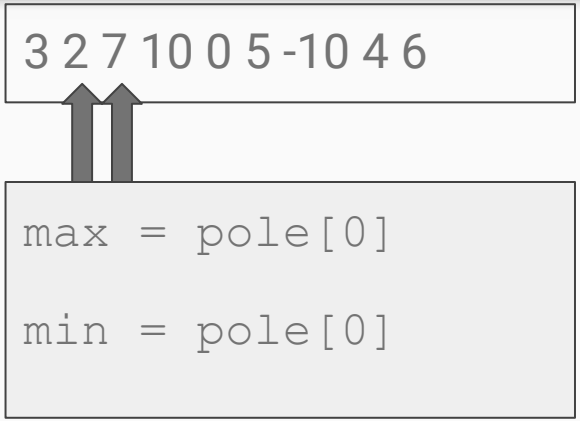
Testů (starý): 2N-2

Testů (nový): (3N-3)/2

Složitost: Jen testy

```
for i in range(0, len(pole), 2):
    if pole[i] < pole[i+1]:
        if pole[i] < min: min = pole[i]
        if pole[i+1] > max: max = pole[i+1]
    else:
        if pole[i] > max: max = pole[i]
        if pole[i+1] < min: min = pole[i+1]
```

Dvojice = 3 testy
Dvojic v poli: N/2



Testů (starý): 2N

Testů (nový): 3N/2

Motivační příklad - hledání v poli

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

Lineárně:

Najdi 363: 1 test

Najdi 993: 18 testů (obecně N)

Jak to lze jednoduše urychlit?

Motivační příklad - hledání v poli

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

Půlením intervalu (binary search):

Půlíme zkoumaný interval (1. bude 836), je hledané číslo < či >?

Najdi 363: 836, 603, 369, 363 - 4 testy

Najdi 993: 836, 939, 968, 983, 993 - 5 testů

Obecně: $\log_2(N)$

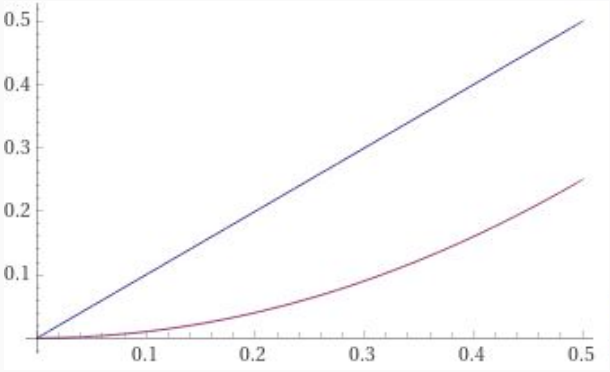
Na čem to záleží?

- Měli jsme seřazené pole čísel, co kdyby nebylo seřazené?
- Kde vůbec vezmu seřazené pole?

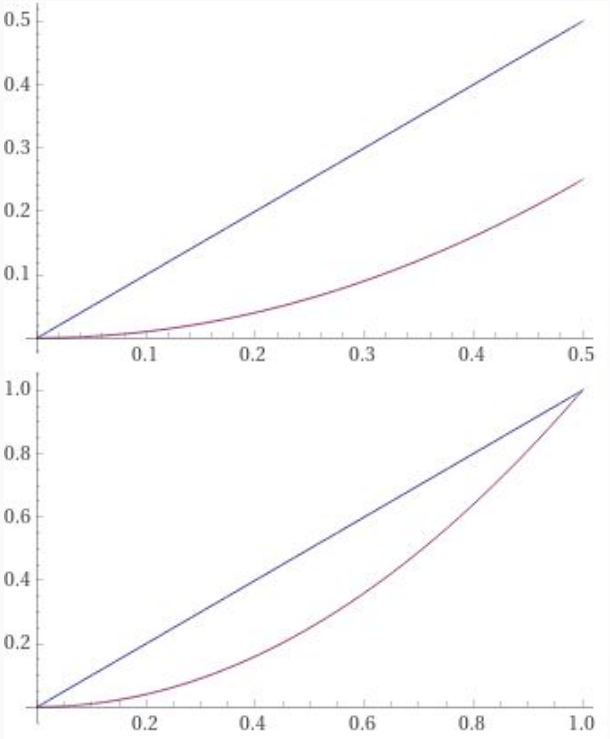
Alias toto jsou otázky, co vedou k diskuzi o různém ukládání dat a nejen o psaní algoritmů samotných, o hledání (search) nebo řazení dat (sorting) a také o efektivitě algoritmů v nejlepším či nejhorším případě.

Asymptotická složitost

Porovnání náročnosti funkcí



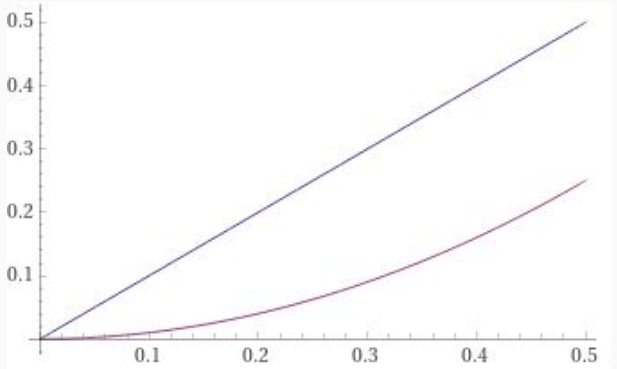
Porovnání náročnosti funkcí



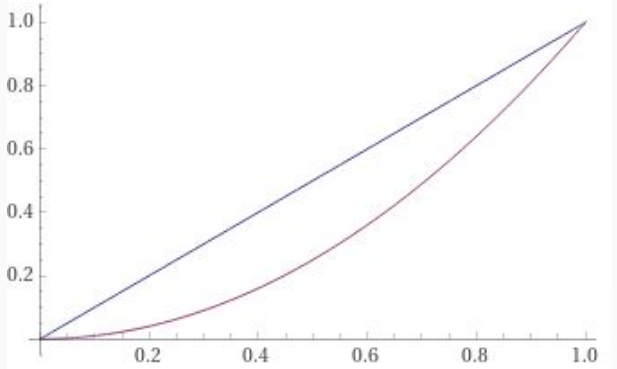
Porovnání náročnosti funkcí

Potřebujeme odstup

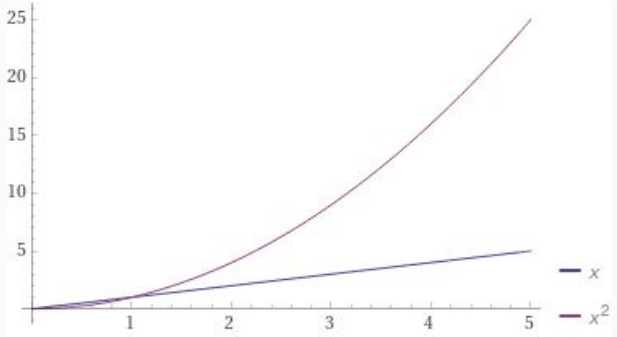
0.5



1



5



Růst funkcí

Pokud 1 operace zabere 1 µs

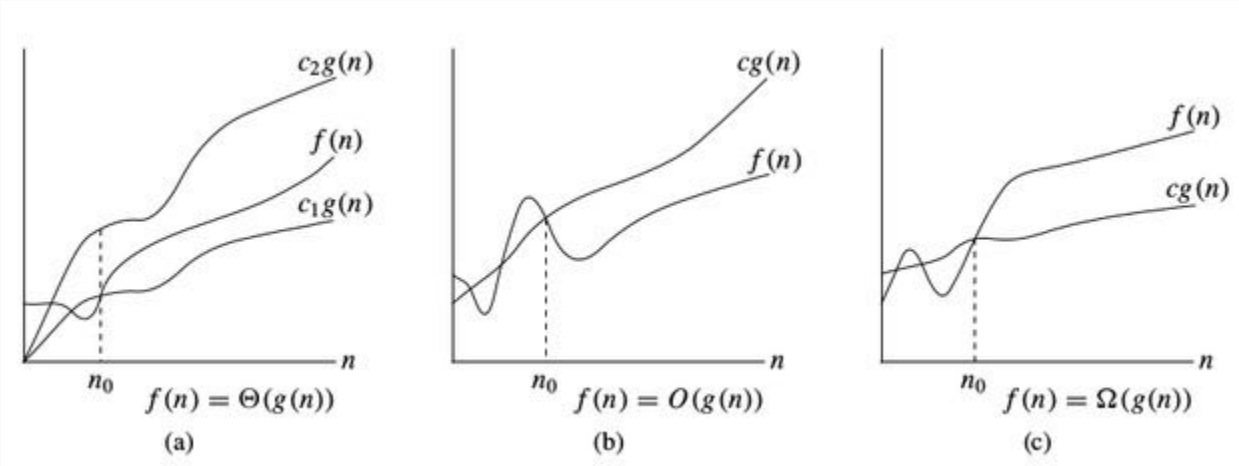
	10	20	40	60	500	1000
$\log_2 n$	3,3 µs	4,3 µs	5 µs	5,8 µs	9 µs	10 µs
n	10 µs	20 µs	40 µs	60 µs	500 µs	**1 ms**
n^2	0,1 ms	0,4 ms	1,6 ms	3,6 ms	0,25 s	**1 s**
2^n	**1 ms**	**1 s**	12,7 d	36000 y	10^{137} y	10^{287} y
n!	3,6 s	77000 y	10^{34} y	10^{68} y	10^{1110} y	10^{2554} y

Složitosti podle druhu

- Prostorová složitost
  - alias kolik to zabere v paměti
  - např. iterativní vs. rekurzivní metoda
- Časová složitost
  - alias jak dlouho to poběží

Často mohou jít proti sobě, kdy si můžete ušetřit opakovaný výpočet tím, že si nějaké hodnoty uložíte do paměti.

Asymptotické porovnání



- Průměr
- Horní
- Dolní

https://www.hackerearth.com/practice/basic-programming/complexity-analysis/time-and-space-complexity/tutorial/

Dolní odhad Ω

“Omega”

- Zkoumaná funkce f je ohraničena funkcí g zdola (až na konstantu)
- f(n) $\in \Omega$(g(n))

$$(\exists c > 0)(\exists n_0)(\forall n > n_0) : c \cdot g(n) \leq f(n)$$

Horní odhad $\mathbf{O}$

“Omikron”, “Big O notation”

- f je asymptoticky ohraničena shora
- f(n) $\in \mathbf{O}$(g(n))

$$(\exists c > 0)(\exists n_0)(\forall n > n_0) : f(n) \leq c \cdot g(n)$$

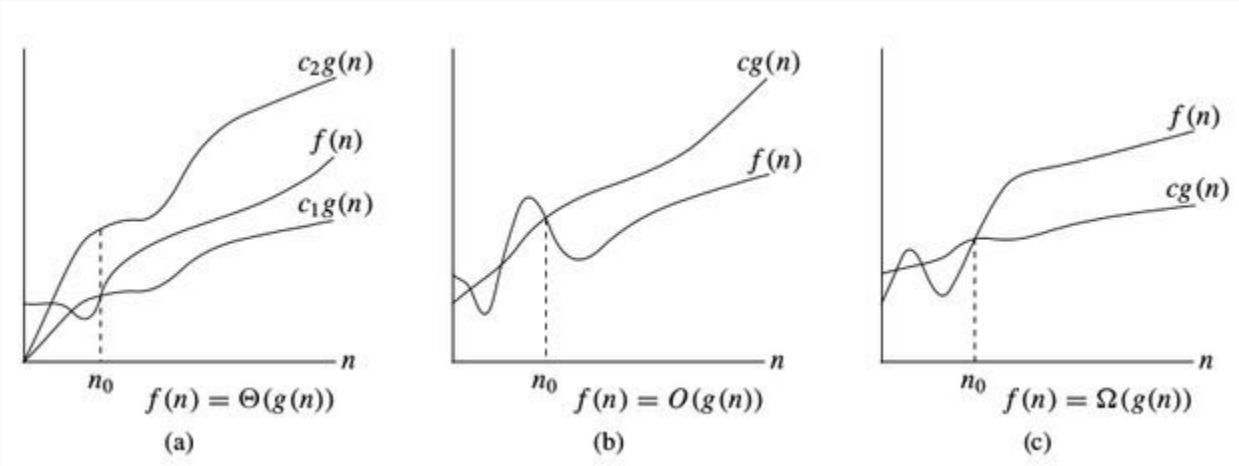
“Průměrný” odhad Θ

“Theta”

- f je asymptoticky ohraničena g z obou stran (až na konstantu)

$$(\exists c_1, c_2 > 0)(\exists n_0)(\forall n > n_0) : c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

Asymptotické porovnání



- Průměr
- Horní
- Dolní

https://www.hackerearth.com/practice/basic-programming/complexity-analysis/time-and-space-complexity/tutorial/

Asymptotické odhady

Logaritmický	$f(n) \in \Theta(\log(n))$ (jedno jaký základ)
Lineární	$f(n) \in \Theta(n)$
Kvadratický	$f(n) \in \Theta(n^2)$
Kubický	$f(n) \in \Theta(n^3)$
Polynomiální	$f(n) \in \Theta(n^k)$
Exponenciální	$f(n) \in \Theta(k^n)$

Asymptotické odhady

Logaritmický	$f(n) \in \Theta(\log(n))$ (jedno jaký základ)
Lineární	$f(n) \in \Theta(n)$
Kvadratický	$f(n) \in \Theta(n^2)$
Kubický	$f(n) \in \Theta(n^3)$
Polynomiální	$f(n) \in \Theta(n^k)$
Exponenciální	$f(n) \in \Theta(k^n)$

Proč se soustředíme na Theta odhad?

Prashant 7 @prashant3302 · 55s

Alternative Big O notations:

O(1) = O(yeah)
O(log n) = O(nice)
O(nlogn) = O(k-ish)
O(n) = O(ok)
$O(n^2)$ = O(my)
$O(2^n)$ = O(no)
O(n^n) = O(f**k)
O(n!) = O(mg!)
#programminghumor

Vlastnosti asymptotických odhadů

$$
\begin{aligned}
n^m &\in O(n^{m'}) \text{ pokud } m \leq m' \\
f(n) &\in O(f(n)) \\
c \cdot O(f(n)) &= O(c \cdot f(n)) = O(f(n)) \\
O(O(f(n))) &= O(f(n)) \\
O(f(n)) + O(g(n)) &= O(\max\{f(n), g(n)\}) \\
O(f(n)) \cdot O(g(n)) &= O(f(n) \cdot g(n)) \\
O(f(n) \cdot g(n)) &= f(n) \cdot O(g(n))
\end{aligned}
$$

https://cw.fel.cvut.cz/old/courses/b4b33alg/start

Vlastnosti asymptotických odhadů

$$
\begin{aligned}
n^m &\in O(n^{m'}) \text{ pokud } m \leq m' \\
f(n) &\in O(f(n)) \\
c \cdot O(f(n)) &= O(c \cdot f(n)) = O(f(n)) \\
O(O(f(n))) &= O(f(n)) \\
O(f(n)) + O(g(n)) &= O(\max\{f(n), g(n)\}) \\
O(f(n)) \cdot O(g(n)) &= O(f(n) \cdot g(n)) \\
O(f(n) \cdot g(n)) &= f(n) \cdot O(g(n))
\end{aligned}
$$

Jaká je asymptotická složitost funkce dané polynomem:

$n^4 + 4n^3 + 3n^2 + 2n + 1$

https://cw.fel.cvut.cz/old/courses/b4b33alg/start

Vlastnosti asymptotických odhadů

- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = 0$, pak $f(n) \in O(g(n))$, ale **neplatí** $f(n) \in \Theta(g(n))$
- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = a$, kde $0 < a < \infty$, pak $f(n) \in \Theta(g(n))$
- $\lim_{n\to\infty} \frac{f(n)}{g(n)} = \infty$, pak $g(n) \in O(f(n))$, ale **neplatí** $g(n) \in \Theta(f(n))$

https://cw.fel.cvut.cz/old/courses/b4b33alg/start

Algoritmus není funkce

Když roste funkce **rychleji**,

tak to znamená, že algoritmus je **náročnější** (=**pomalejší** či zabírá víc prostoru).

DU

1) Popište přesně algoritmus pro sčítání a algoritmus pro násobení dvou čísel.
2) Vyhodnoťte jejich složitost.

DU

Najděte zajímavý příklad pro **každý** případ asymptotického odhadu.

Ukázka:

- $x^2 \in \Omega(x)$ ...
- $x^2 \in \mathbf{O}(2^x)$ ...
- $\ln x \in \Theta(\log_{10}(x))$ ...

Zkuste i nějaké netriviální příklady.