Složitost algoritmů

DELTA - Střední škola informatiky a ekonomie, s.r.o.

Ing. Luboš Zápotočný

02.02.2026

Složitost algoritmů

Motivační příklad

Představte si: Máte databázi s 1 000 000 studentů a potřebujete najít jednoho konkrétního studenta

Motivační příklad

Představte si: Máte databázi s 1 000 000 studentů a potřebujete najít jednoho konkrétního studenta

Přístup A: Kontrolujete každého studenta postupně (lineární vyhledávání)

- Počet kontrol:

Motivační příklad

Představte si: Máte databázi s 1 000 000 studentů a potřebujete najít jednoho konkrétního studenta

Přístup A: Kontrolujete každého studenta postupně (lineární vyhledávání)

- Počet kontrol: 1 000 000

Motivační příklad

Představte si: Máte databázi s 1 000 000 studentů a potřebujete najít jednoho konkrétního studenta

Přístup A: Kontrolujete každého studenta postupně (lineární vyhledávání)

- Počet kontrol: 1 000 000

Přístup B: Chytré vyhledávání (binární vyhledávání)

- Počet kontrol:

Motivační příklad

Představte si: Máte databázi s 1 000 000 studentů a potřebujete najít jednoho konkrétního studenta

Přístup A: Kontrolujete každého studenta postupně (lineární vyhledávání)

- Počet kontrol: 1 000 000

Přístup B: Chytré vyhledávání (binární vyhledávání)

- Počet kontrol: 20

Motivační příklad

Představte si: Máte databázi s 1 000 000 studentů a potřebujete najít jednoho konkrétního studenta

Přístup A: Kontrolujete každého studenta postupně (lineární vyhledávání)

- Počet kontrol: 1 000 000

Přístup B: Chytré vyhledávání (binární vyhledávání)

- Počet kontrol: 20

Otázka: Kolik času to zabere?

Reálný dopad

1 operace = (1 μs) =

Reálný dopad

$$1 \text{ operace} = (1\ \mu\text{s}) = 10^{-6}\ \text{s} =$$

Reálný dopad

$$1 \text{ operace} = (1\ \mu s) = 10^{-6}\ s = 0.000001\ s$$

Reálný dopad

$$1 \text{ operace} = (1\ \mu\text{s}) = 10^{-6}\ \text{s} = 0.000001\ \text{s}$$

	Lineární	**Binární**

Reálný dopad

$$1 \text{ operace} = (1\ \mu\text{s}) = 10^{-6}\ \text{s} = 0.000001\ \text{s}$$

	Lineární	Binární
1 000 000 operací		

Reálný dopad

$$1 \text{ operace} = (1\ \mu\text{s}) = 10^{-6}\ \text{s} = 0.000001\ \text{s}$$

	Lineární	**Binární**
1 000 000 operací	1 s	

Reálný dopad

$$1 \text{ operace} = (1\ \mu\text{s}) = 10^{-6}\ \text{s} = 0.000001\ \text{s}$$

	Lineární	Binární
1 000 000 operací	1 s	0,00002 s

Reálný dopad

$$1 \text{ operace} = (1\ \mu s) = 10^{-6}\ s = 0.000001\ s$$

	Lineární	**Binární**
1 000 000 operací	1 s	0,00002 s
1 000 000 000 operací		

Reálný dopad

$$1 \text{ operace} = (1\ \mu s) = 10^{-6}\ s = 0.000001\ s$$

	Lineární	**Binární**
1 000 000 operací	1 s	0,00002 s
1 000 000 000 operací	16 min	

Reálný dopad

$$1 \text{ operace} = (1\ \mu\text{s}) = 10^{-6}\ \text{s} = 0.000001\ \text{s}$$

	Lineární	**Binární**
1 000 000 operací	1 s	0,00002 s
1 000 000 000 operací	16 min	0,00003 s

Reálný dopad

$$1 \text{ operace} = (1\ \mu\text{s}) = 10^{-6}\ \text{s} = 0.000001\ \text{s}$$

	Lineární	Binární
1 000 000 operací	1 s	0,00002 s
1 000 000 000 operací	16 min	0,00003 s
10 000 000 000 operací		

Reálný dopad

$$1 \text{ operace} = (1\ \mu s) = 10^{-6}\ s = 0.000001\ s$$

	Lineární	**Binární**
1 000 000 operací	1 s	0,00002 s
1 000 000 000 operací	16 min	0,00003 s
10 000 000 000 operací	2h 46 min	

Reálný dopad

$$1 \text{ operace} = (1\ \mu s) = 10^{-6}\ s = 0.000001\ s$$

	Lineární	**Binární**
1 000 000 operací	1 s	0,00002 s
1 000 000 000 operací	16 min	0,00003 s
10 000 000 000 operací	2h 46 min	0,000033 s

Fibonacciho posloupnost

Definice

Fibonacciho posloupnost je definována rekurzivně:

Definice

Fibonacciho posloupnost je definována rekurzivně:

$$\mathrm{F}(n) = \begin{cases} 0 & \text{pro } n = 0 \\ 1 & \text{pro } n = 1 \\ \mathrm{F}(n-1) + \mathrm{F}(n-2) & \text{pro } n \geq 2 \end{cases}$$

Hodnoty Fibonacciho posloupnosti

n	0	1	2	3	4	5	6	7	8	9
F(n)										

Hodnoty Fibonacciho posloupnosti

n	0	1	2	3	4	5	6	7	8	9
F(n)	0									

Hodnoty Fibonacciho posloupnosti

n	0	1	2	3	4	5	6	7	8	9
F(n)	0	1								

Hodnoty Fibonacciho posloupnosti

n	0	1	2	3	4	5	6	7	8	9
F(n)	0	1	1							

Hodnoty Fibonacciho posloupnosti

n	0	1	2	3	4	5	6	7	8	9
F(n)	0	1	1	2						

Hodnoty Fibonacciho posloupnosti

n	0	1	2	3	4	5	6	7	8	9
F(n)	0	1	1	2	3					

Hodnoty Fibonacciho posloupnosti

n	0	1	2	3	4	5	6	7	8	9
F(n)	0	1	1	2	3	5				

Hodnoty Fibonacciho posloupnosti

n	0	1	2	3	4	5	6	7	8	9
F(n)	0	1	1	2	3	5	8			

Hodnoty Fibonacciho posloupnosti

n	0	1	2	3	4	5	6	7	8	9
F(n)	0	1	1	2	3	5	8	13		

Hodnoty Fibonacciho posloupnosti

n	0	1	2	3	4	5	6	7	8	9
F(n)	0	1	1	2	3	5	8	13	21	

Hodnoty Fibonacciho posloupnosti

n	0	1	2	3	4	5	6	7	8	9
F(n)	0	1	1	2	3	5	8	13	21	34

Hodnoty Fibonacciho posloupnosti

n	0	1	2	3	4	5	6	7	8	9
F(n)	0	1	1	2	3	5	8	13	21	34

n	10	11	12	13	14	15	16	17	18	19
F(n)	55	89	144	233	377	610	987	1597	2584	4181

Větší hodnoty

n	F(n)

Větší hodnoty

n	F(n)
30	832 040

Větší hodnoty

n	F(n)
30	832 040
40	102 334 155

Větší hodnoty

n	F(n)
30	832 040
40	102 334 155
50	12 586 269 025

Větší hodnoty

n	F(n)
30	832 040
40	102 334 155
50	12 586 269 025
60	1 548 008 755 920

Větší hodnoty

n	F(n)
30	832 040
40	102 334 155
50	12 586 269 025
60	1 548 008 755 920

Hodnoty rostou

Větší hodnoty

n	F(n)
30	832 040
40	102 334 155
50	12 586 269 025
60	1 548 008 755 920

Hodnoty rostou **exponenciálně**

Tři přístupy k výpočtu

Jak můžeme spočítat F(n)?

Tři přístupy k výpočtu

Jak můžeme spočítat F(n)?

1. **Rekurzivní** (naivní) přístup

Tři přístupy k výpočtu

Jak můžeme spočítat F(n)?

1. **Rekurzivní** (naivní) přístup
2. **Iterativní** přístup

Tři přístupy k výpočtu

Jak můžeme spočítat F(n)?

1. **Rekurzivní** (naivní) přístup
2. **Iterativní** přístup
3. **Explicitní vzorec** (Binetův vzorec)

Rekurzivní přístup

Rekurzivní přístup

```c
int fib_recursive(int n) {
    if (n <= 1) return n;
    return fib_recursive(n - 1) + fib_recursive(n - 2);
}
```

Rekurzivní přístup

```
int fib_recursive(int n) {
    if (n <= 1) return n;
    return fib_recursive(n - 1) + fib_recursive(n - 2);
}
```

Složitost:

Rekurzivní přístup

```
int fib_recursive(int n) {
    if (n <= 1) return n;
    return fib_recursive(n - 1) + fib_recursive(n - 2);
}
```

Složitost: $\mathcal{O}(2^n)$

Rekurzivní přístup

```c
int fib_recursive(int n) {
    if (n <= 1) return n;
    return fib_recursive(n - 1) + fib_recursive(n - 2);
}
```

Složitost: $\mathcal{O}(2^n)$ - VELMI POMALÉ

Rekurzivní přístup

```c
int fib_recursive(int n) {
    if (n <= 1) return n;
    return fib_recursive(n - 1) + fib_recursive(n - 2);
}
```

Složitost: $\mathcal{O}(2^n)$ - VELMI POMALÉ

Proč?

Rekurzivní přístup

```c
int fib_recursive(int n) {
    if (n <= 1) return n;
    return fib_recursive(n - 1) + fib_recursive(n - 2);
}
```

Složitost: $\mathcal{O}(2^n)$ - VELMI POMALÉ

Proč? Každé volání vytváří

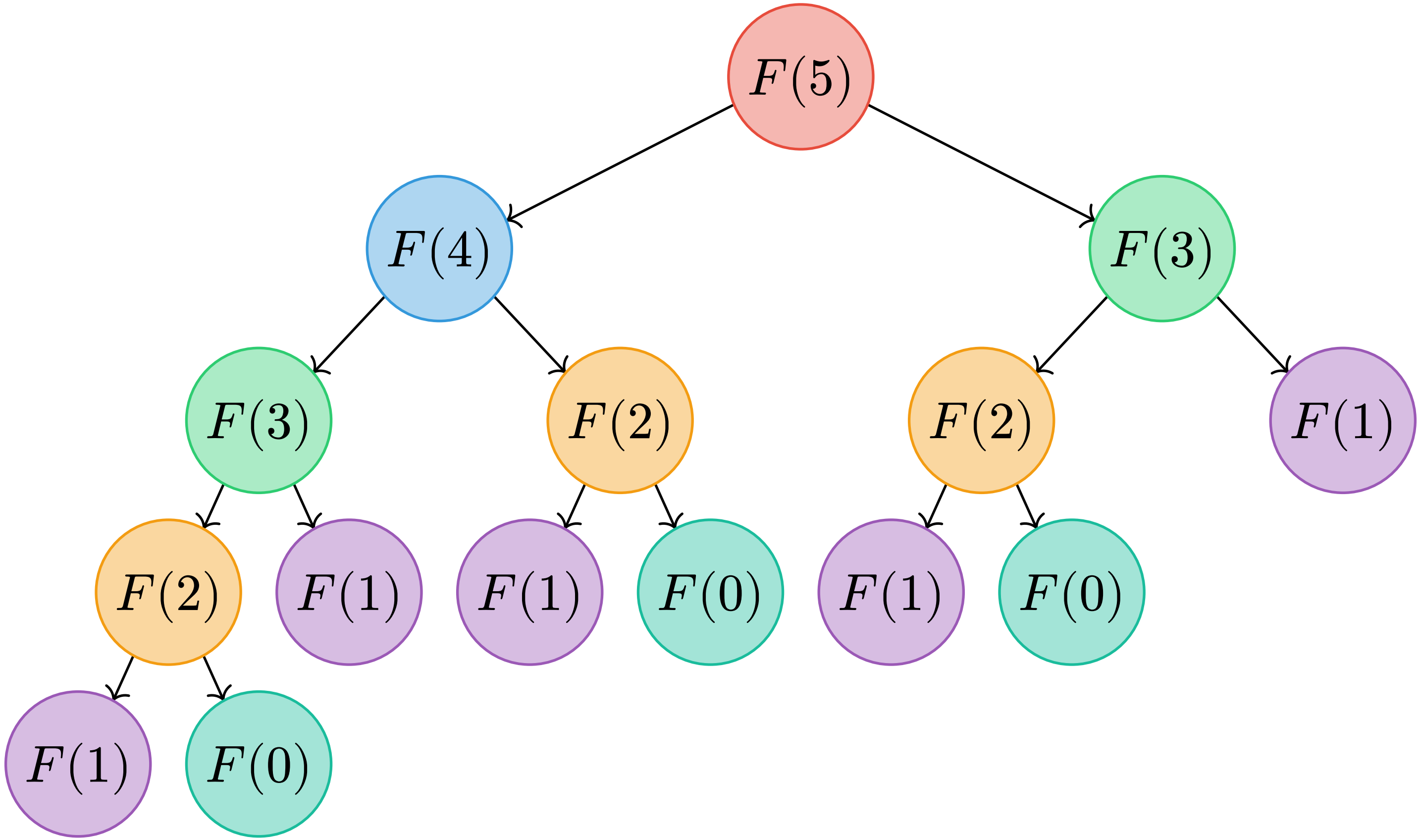
Rekurzivní přístup

```c
int fib_recursive(int n) {
    if (n <= 1) return n;
    return fib_recursive(n - 1) + fib_recursive(n - 2);
}
```

Složitost: $\mathcal{O}(2^n)$ - VELMI POMALÉ

Proč? Každé volání vytváří dvě další volání

Rekurzivní přístup



Iterativní přístup

Iterativní přístup

```c
int fib_iterative(int n) {
    if (n <= 1) return n;
    int a = 0, b = 1;
    for (int i = 2; i <= n; i++) {
        int temp = a + b;
        a = b; b = temp;
    }
    return b;
}
```

Iterativní přístup

```c
int fib_iterative(int n) {
    if (n <= 1) return n;
    int a = 0, b = 1;
    for (int i = 2; i <= n; i++) {
        int temp = a + b;
        a = b; b = temp;
    }
    return b;
}
```

Složitost:

Iterativní přístup

```c
int fib_iterative(int n) {
    if (n <= 1) return n;
    int a = 0, b = 1;
    for (int i = 2; i <= n; i++) {
        int temp = a + b;
        a = b; b = temp;
    }
    return b;
}
```

Složitost: $\mathcal{O}(n)$

Iterativní přístup

```c
int fib_iterative(int n) {
    if (n <= 1) return n;
    int a = 0, b = 1;
    for (int i = 2; i <= n; i++) {
        int temp = a + b;
        a = b; b = temp;
    }
    return b;
}
```

Složitost: $\mathcal{O}(n)$ - Mnohem lepší

Iterativní přístup

```c
int fib_iterative(int n) {
    if (n <= 1) return n;
    int a = 0, b = 1;
    for (int i = 2; i <= n; i++) {
        int temp = a + b;
        a = b; b = temp;
    }
    return b;
}
```

Složitost: $\mathcal{O}(n)$ - Mnohem lepší

Proč?

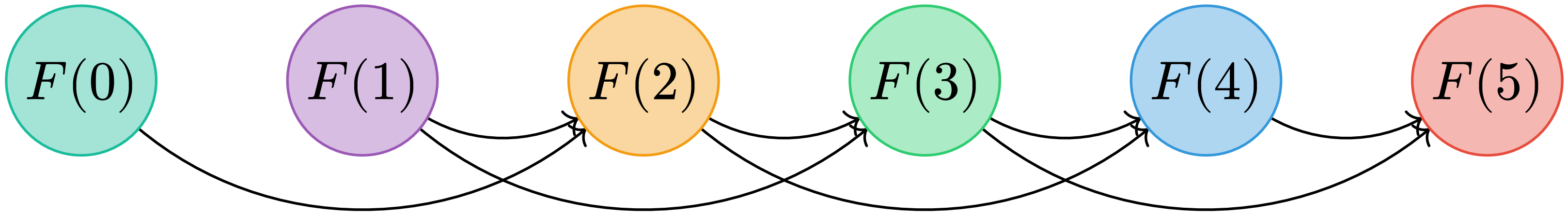
Iterativní přístup

```
int fib_iterative(int n) {
    if (n <= 1) return n;
    int a = 0, b = 1;
    for (int i = 2; i <= n; i++) {
        int temp = a + b;
        a = b; b = temp;
    }
    return b;
}
```

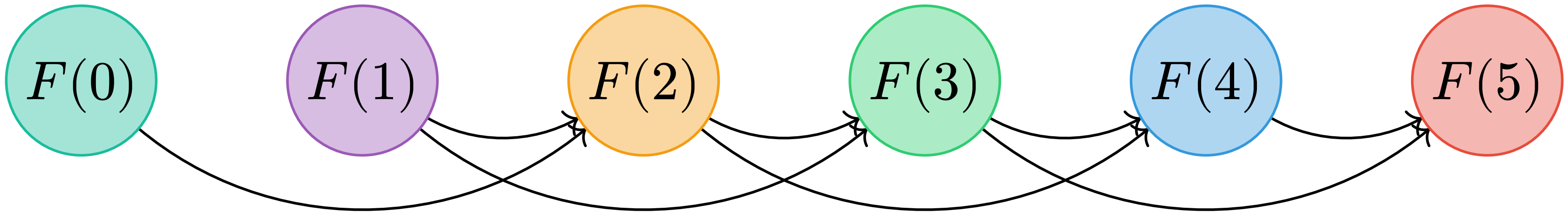
Složitost: $\mathcal{O}(n)$ - Mnohem lepší

Proč? Procházíme čísla od 2 do n pouze jednou

Iterativní přístup



Iterativní přístup



Každý uzel potřebuje pouze **2 předchůdce** → **lineární složitost**

Explicitní vzorec (Binetův)

Explicitní vzorec (Binetův)

$$\mathrm{F}(n) = \frac{\varphi^n - \psi^n}{\sqrt{5}}$$

Explicitní vzorec (Binetův)

$$\mathrm{F}(n) = \frac{\varphi^n - \psi^n}{\sqrt{5}}$$

kde

Explicitní vzorec (Binetův)

$$\mathrm{F}(n) = \frac{\varphi^n - \psi^n}{\sqrt{5}}$$

kde $\varphi = \frac{1+\sqrt{5}}{2}$ (zlatý řez)

Explicitní vzorec (Binetův)

$$\mathrm{F}(n) = \frac{\varphi^n - \psi^n}{\sqrt{5}}$$

kde $\varphi = \frac{1+\sqrt{5}}{2}$ (zlatý řez) a $\psi = \frac{1-\sqrt{5}}{2}$

Explicitní vzorec (Binetův)

$$\mathrm{F}(n) = \frac{\varphi^n - \psi^n}{\sqrt{5}}$$

kde $\varphi = \frac{1+\sqrt{5}}{2}$ (zlatý řez) a $\psi = \frac{1-\sqrt{5}}{2}$

Složitost:

Explicitní vzorec (Binetův)

$$\mathrm{F}(n) = \frac{\varphi^n - \psi^n}{\sqrt{5}}$$

kde $\varphi = \frac{1+\sqrt{5}}{2}$ (zlatý řez) a $\psi = \frac{1-\sqrt{5}}{2}$

Složitost: $\mathcal{O}(1)$

Explicitní vzorec (Binetův)

$$\mathrm{F}(n) = \frac{\varphi^n - \psi^n}{\sqrt{5}}$$

kde $\varphi = \frac{1+\sqrt{5}}{2}$ (zlatý řez) a $\psi = \frac{1-\sqrt{5}}{2}$

Složitost: $\mathcal{O}(1)$ - Konstantní čas

Explicitní vzorec (Binetův)

$$F(n) = \frac{\varphi^n - \psi^n}{\sqrt{5}}$$

kde $\varphi = \frac{1+\sqrt{5}}{2}$ (zlatý řez) a $\psi = \frac{1-\sqrt{5}}{2}$

Složitost: $\mathcal{O}(1)$ - Konstantní čas

Proč?

Explicitní vzorec (Binetův)

$$\mathrm{F}(n) = \frac{\varphi^n - \psi^n}{\sqrt{5}}$$

kde $\varphi = \frac{1+\sqrt{5}}{2}$ (zlatý řez) a $\psi = \frac{1-\sqrt{5}}{2}$

Složitost: $\mathcal{O}(1)$ - Konstantní čas

Proč? Pouze aritmetické operace, nezávisle na velikosti n

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10			

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10	100 operací		

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10	100 operací	10 operací	

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10	100 operací	10 operací	1 operace

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10	100 operací	10 operací	1 operace
20			

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10	100 operací	10 operací	1 operace
20	10 000 operací		

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10	100 operací	10 operací	1 operace
20	10 000 operací	20 operací	

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10	100 operací	10 operací	1 operace
20	10 000 operací	20 operací	1 operace

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10	100 operací	10 operací	1 operace
20	10 000 operací	20 operací	1 operace
30			

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10	100 operací	10 operací	1 operace
20	10 000 operací	20 operací	1 operace
30	1 000 000 operací		

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10	100 operací	10 operací	1 operace
20	10 000 operací	20 operací	1 operace
30	1 000 000 operací	30 operací	

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10	100 operací	10 operací	1 operace
20	10 000 operací	20 operací	1 operace
30	1 000 000 operací	30 operací	1 operace

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10	100 operací	10 operací	1 operace
20	10 000 operací	20 operací	1 operace
30	1 000 000 operací	30 operací	1 operace
40			

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10	100 operací	10 operací	1 operace
20	10 000 operací	20 operací	1 operace
30	1 000 000 operací	30 operací	1 operace
40	1 000 000 000 operací		

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10	100 operací	10 operací	1 operace
20	10 000 operací	20 operací	1 operace
30	1 000 000 operací	30 operací	1 operace
40	1 000 000 000 operací	40 operací	

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10	100 operací	10 operací	1 operace
20	10 000 operací	20 operací	1 operace
30	1 000 000 operací	30 operací	1 operace
40	1 000 000 000 operací	40 operací	1 operace

Srovnání přístupů

n	Rekurzivní	Iterativní	Explicitní
10	100 operací	10 operací	1 operace
20	10 000 operací	20 operací	1 operace
30	1 000 000 operací	30 operací	1 operace
40	1 000 000 000 operací	40 operací	1 operace

Pozn.: Rekurzivní hodnoty jsou horní odhad ($\mathcal{O}(2^n)$),
skutečný počet volání roste jako φ^n kde $\varphi \approx 1.618$

Analýza implementací výpočtu F(n) v C

Asymptotická složitost

Intuitivní vysvětlení

Asymptotická složitost říká, jak rychle roste čas nebo paměť v závislosti na velikosti vstupu

Intuitivní vysvětlení

Asymptotická složitost říká, jak rychle roste čas nebo paměť v závislosti na velikosti vstupu

Analogie:

Intuitivní vysvětlení

Asymptotická složitost říká, jak rychle roste čas nebo paměť v závislosti na velikosti vstupu

Analogie: Hledáte slovo ve slovníku

Intuitivní vysvětlení

Asymptotická složitost říká, jak rychle roste čas nebo paměť v závislosti na velikosti vstupu

Analogie: Hledáte slovo ve slovníku

- Nezajímá nás, kolik času trvá vyhledávání ve slovníku

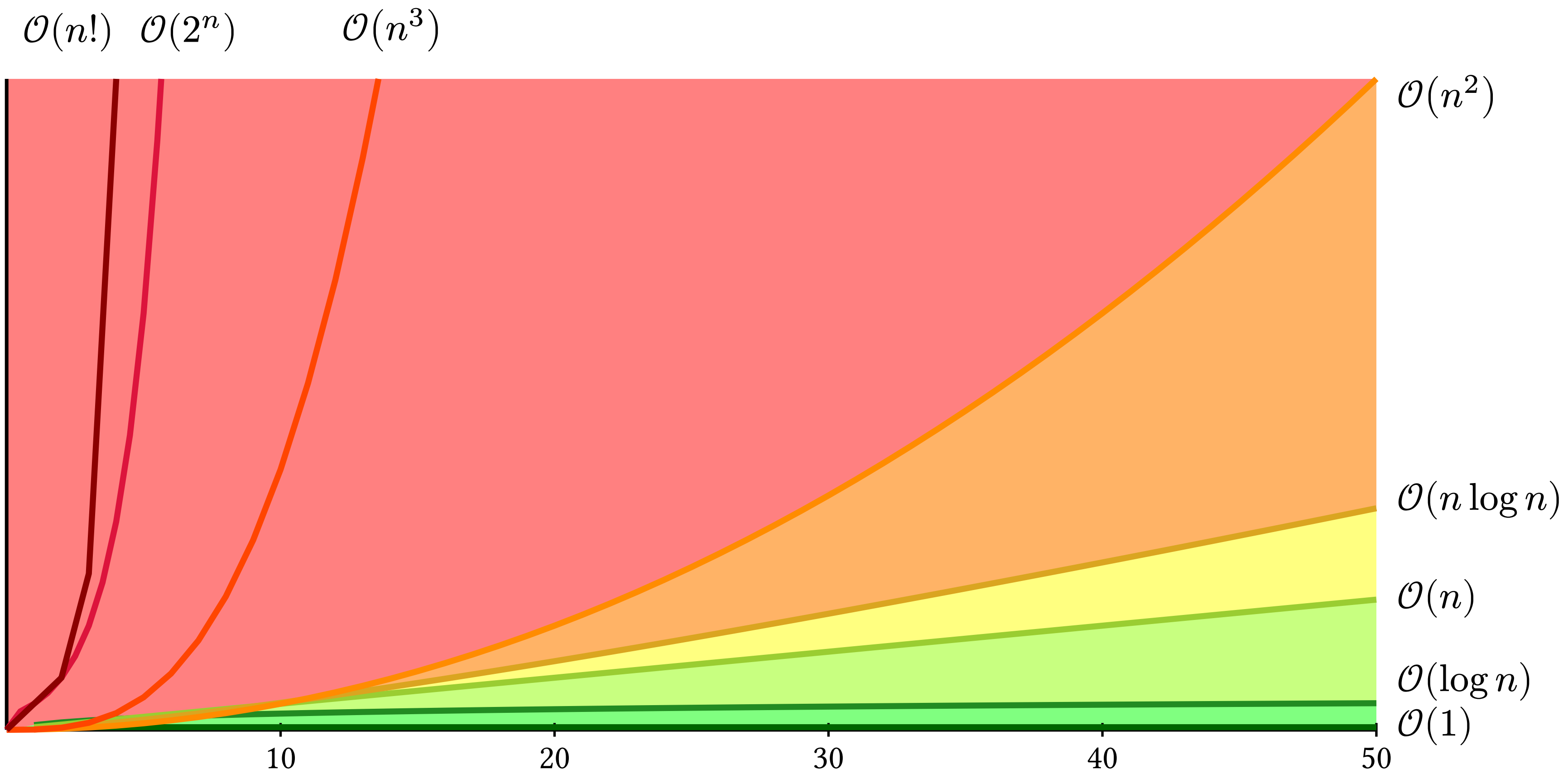
Intuitivní vysvětlení

Asymptotická složitost říká, jak rychle roste čas nebo paměť v závislosti na velikosti vstupu

Analogie: Hledáte slovo ve slovníku

- Nezajímá nás, kolik času trvá vyhledávání ve slovníku
- Pokud se ale slovník zdvojnásobí, trvá to dvakrát déle nebo jen o chvíli déle?

Intuitivní vysvětlení



Očekávané časy

- n = 1 000 000
- 1 operace = 1 mikrosekunda (1 μs)

Očekávané časy

- n = 1 000 000
- 1 operace = 1 mikrosekunda (1 μs)

Složitost	Počet operací	Čas

Očekávané časy

- n = 1 000 000
- 1 operace = 1 mikrosekunda (1 μs)

Složitost	Počet operací	Čas
$\mathcal{O}(1)$		

Očekávané časy

- n = 1 000 000
- 1 operace = 1 mikrosekunda (1 µs)

Složitost	Počet operací	Čas
$\mathcal{O}(1)$	jednotky	

Očekávané časy

- n = 1 000 000
- 1 operace = 1 mikrosekunda (1 μs)

Složitost	Počet operací	Čas
$\mathcal{O}(1)$	jednotky	0.000001 s

Očekávané časy

- n = 1 000 000
- 1 operace = 1 mikrosekunda (1 µs)

Složitost	Počet operací	Čas
$\mathcal{O}(1)$	jednotky	0.000001 s
$\mathcal{O}(\log n)$		

Očekávané časy

- n = 1 000 000
- 1 operace = 1 mikrosekunda (1 μs)

Složitost	Počet operací	Čas
$\mathcal{O}(1)$	jednotky	0.000001 s
$\mathcal{O}(\log n)$	20	

Očekávané časy

- n = 1 000 000
- 1 operace = 1 mikrosekunda (1 µs)

Složitost	Počet operací	Čas
$\mathcal{O}(1)$	jednotky	0.000001 s
$\mathcal{O}(\log n)$	20	0.00002 s

Očekávané časy

- n = 1 000 000
- 1 operace = 1 mikrosekunda (1 μs)

Složitost	Počet operací	Čas
$\mathcal{O}(1)$	jednotky	0.000001 s
$\mathcal{O}(\log n)$	20	0.00002 s
$\mathcal{O}(n)$		

Očekávané časy

- n = 1 000 000
- 1 operace = 1 mikrosekunda (1 µs)

Složitost	Počet operací	Čas
$\mathcal{O}(1)$	jednotky	0.000001 s
$\mathcal{O}(\log n)$	20	0.00002 s
$\mathcal{O}(n)$	1 000 000	

Očekávané časy

- n = 1 000 000
- 1 operace = 1 mikrosekunda (1 µs)

Složitost	Počet operací	Čas
$\mathcal{O}(1)$	jednotky	0.000001 s
$\mathcal{O}(\log n)$	20	0.00002 s
$\mathcal{O}(n)$	1 000 000	1 s

Očekávané časy

- n = 1 000 000
- 1 operace = 1 mikrosekunda (1 µs)

Složitost	Počet operací	Čas
$\mathcal{O}(1)$	jednotky	0.000001 s
$\mathcal{O}(\log n)$	20	0.00002 s
$\mathcal{O}(n)$	1 000 000	1 s
$\mathcal{O}(n \log n)$		

Očekávané časy

- n = 1 000 000
- 1 operace = 1 mikrosekunda (1 μs)

Složitost	Počet operací	Čas
$\mathcal{O}(1)$	jednotky	0.000001 s
$\mathcal{O}(\log n)$	20	0.00002 s
$\mathcal{O}(n)$	1 000 000	1 s
$\mathcal{O}(n \log n)$	20 000 000	

Očekávané časy

- n = 1 000 000
- 1 operace = 1 mikrosekunda (1 µs)

Složitost	Počet operací	Čas
$\mathcal{O}(1)$	jednotky	0.000001 s
$\mathcal{O}(\log n)$	20	0.00002 s
$\mathcal{O}(n)$	1 000 000	1 s
$\mathcal{O}(n \log n)$	20 000 000	20 s

Očekávané časy

Složitost	Počet operací	Čas
$\mathcal{O}(n^2)$		

Očekávané časy

Složitost	Počet operací	Čas
$\mathcal{O}(n^2)$	1 000 000 000 000	

Očekávané časy

Složitost	Počet operací	Čas
$\mathcal{O}(n^2)$	1 000 000 000 000	11.5 dní

Očekávané časy

Složitost	Počet operací	Čas
$\mathcal{O}(n^2)$	1 000 000 000 000	11.5 dní
$\mathcal{O}(n^3)$		

Očekávané časy

Složitost	Počet operací	Čas
$\mathcal{O}(n^2)$	1 000 000 000 000	11.5 dní
$\mathcal{O}(n^3)$	10^{18}	

Očekávané časy

Složitost	Počet operací	Čas
$\mathcal{O}(n^2)$	1 000 000 000 000	11.5 dní
$\mathcal{O}(n^3)$	10^{18}	31700 let

Očekávané časy

Složitost	Počet operací	Čas
$\mathcal{O}(n^2)$	1 000 000 000 000	11.5 dní
$\mathcal{O}(n^3)$	10^{18}	31700 let
$\mathcal{O}(2^n)$		

Očekávané časy

Složitost	Počet operací	Čas
$\mathcal{O}(n^2)$	1 000 000 000 000	11.5 dní
$\mathcal{O}(n^3)$	10^{18}	31700 let
$\mathcal{O}(2^n)$	$2^{1000000}$	

Očekávané časy

Složitost	Počet operací	Čas
$\mathcal{O}(n^2)$	1 000 000 000 000	11.5 dní
$\mathcal{O}(n^3)$	10^{18}	31700 let
$\mathcal{O}(2^n)$	$2^{1000000}$ … (301030 číslic)	

Očekávané časy

Složitost	Počet operací	Čas
$\mathcal{O}(n^2)$	1 000 000 000 000	11.5 dní
$\mathcal{O}(n^3)$	10^{18}	31700 let
$\mathcal{O}(2^n)$	$2^{1000000}$ … (301030 číslic)	$\approx 10^{301000} \times$ trvání vesmíru

Očekávané časy

Klíčový poznatek:

Očekávané časy

Klíčový poznatek:

Efektivita algoritmů se projevuje hlavně na velkých datech

Asymptotická horní mez - Big O

Intuitivní úvod

Big O říká:

Intuitivní úvod

Big O říká: „Můj algoritmus nebude pomalejší než…“

Intuitivní úvod

Big O říká: „Můj algoritmus nebude pomalejší než…“

Je to jako říct:

Intuitivní úvod

Big O říká: „Můj algoritmus nebude pomalejší než…“

Je to jako říct: „Dojedu tam maximálně za 2 hodiny“

Intuitivní úvod

Big O říká: „Můj algoritmus nebude pomalejší než…“

Je to jako říct: „Dojedu tam maximálně za 2 hodiny“

- Možná dojedu dřív

Intuitivní úvod

Big O říká: „Můj algoritmus nebude pomalejší než…“

Je to jako říct: „Dojedu tam maximálně za 2 hodiny“

- Možná dojedu dřív
- Ale určitě ne později

Formální definice

$$f(n) \in \mathcal{O}(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \leq c \cdot g(n)$$

Formální definice

$$f(n) \in \mathcal{O}(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \leq c \cdot g(n)$$

- $\exists c \in \mathbb{R}^+$:

Formální definice

$$f(n) \in \mathcal{O}(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \leq c \cdot g(n)$$

- $\exists c \in \mathbb{R}^+$: Existuje nějaká kladná konstanta c

Formální definice

$$f(n) \in \mathcal{O}(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \leq c \cdot g(n)$$

- $\exists c \in \mathbb{R}^+$: Existuje nějaká kladná konstanta c
- $\exists n_0 \in \mathbb{N}^+$:

Formální definice

$$f(n) \in \mathcal{O}(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \leq c \cdot g(n)$$

- $\exists c \in \mathbb{R}^+$: Existuje nějaká kladná konstanta c
- $\exists n_0 \in \mathbb{N}^+$: Existuje nějaké počáteční n_0

Formální definice

$$f(n) \in \mathcal{O}(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \leq c \cdot g(n)$$

- $\exists c \in \mathbb{R}^+$: Existuje nějaká kladná konstanta c
- $\exists n_0 \in \mathbb{N}^+$: Existuje nějaké počáteční n_0
- $\forall n \geq n_0$:

Formální definice

$$f(n) \in \mathcal{O}(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \leq c \cdot g(n)$$

- $\exists c \in \mathbb{R}^+$: Existuje nějaká kladná konstanta c
- $\exists n_0 \in \mathbb{N}^+$: Existuje nějaké počáteční n_0
- $\forall n \geq n_0$: Pro všechna n větší než n_0

Formální definice

$$f(n) \in \mathcal{O}(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \leq c \cdot g(n)$$

- $\exists c \in \mathbb{R}^+$: Existuje nějaká kladná konstanta c
- $\exists n_0 \in \mathbb{N}^+$: Existuje nějaké počáteční n_0
- $\forall n \geq n_0$: Pro všechna n větší než n_0
- $f(n) \leq c \cdot g(n)$:

Formální definice

$$f(n) \in \mathcal{O}(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \leq c \cdot g(n)$$

- $\exists c \in \mathbb{R}^+$: Existuje nějaká kladná konstanta c
- $\exists n_0 \in \mathbb{N}^+$: Existuje nějaké počáteční n_0
- $\forall n \geq n_0$: Pro všechna n větší než n_0
- $f(n) \leq c \cdot g(n)$: Naše funkce f je menší než c-násobek g

Příklad 1

Otázka: Je $n \in \mathcal{O}(n^2)$?

Příklad 1

Otázka: Je $n \in \mathcal{O}(n^2)$?

Odpověď:

Příklad 1

Otázka: Je $n \in \mathcal{O}(n^2)$?

Odpověď: Ano

Příklad 1

Otázka: Je $n \in \mathcal{O}(n^2)$?

Odpověď: Ano (c=1, n_0=1)

Příklad 1

Otázka: Je $n \in \mathcal{O}(n^2)$?

Odpověď: Ano (c=1, n_0=1)

Vysvětlení:

Příklad 1

Otázka: Je $n \in \mathcal{O}(n^2)$?

Odpověď: Ano (c=1, n_0=1)

Vysvětlení: n je vždy menší než n^2 pro $n \geq 1$

Příklad 1

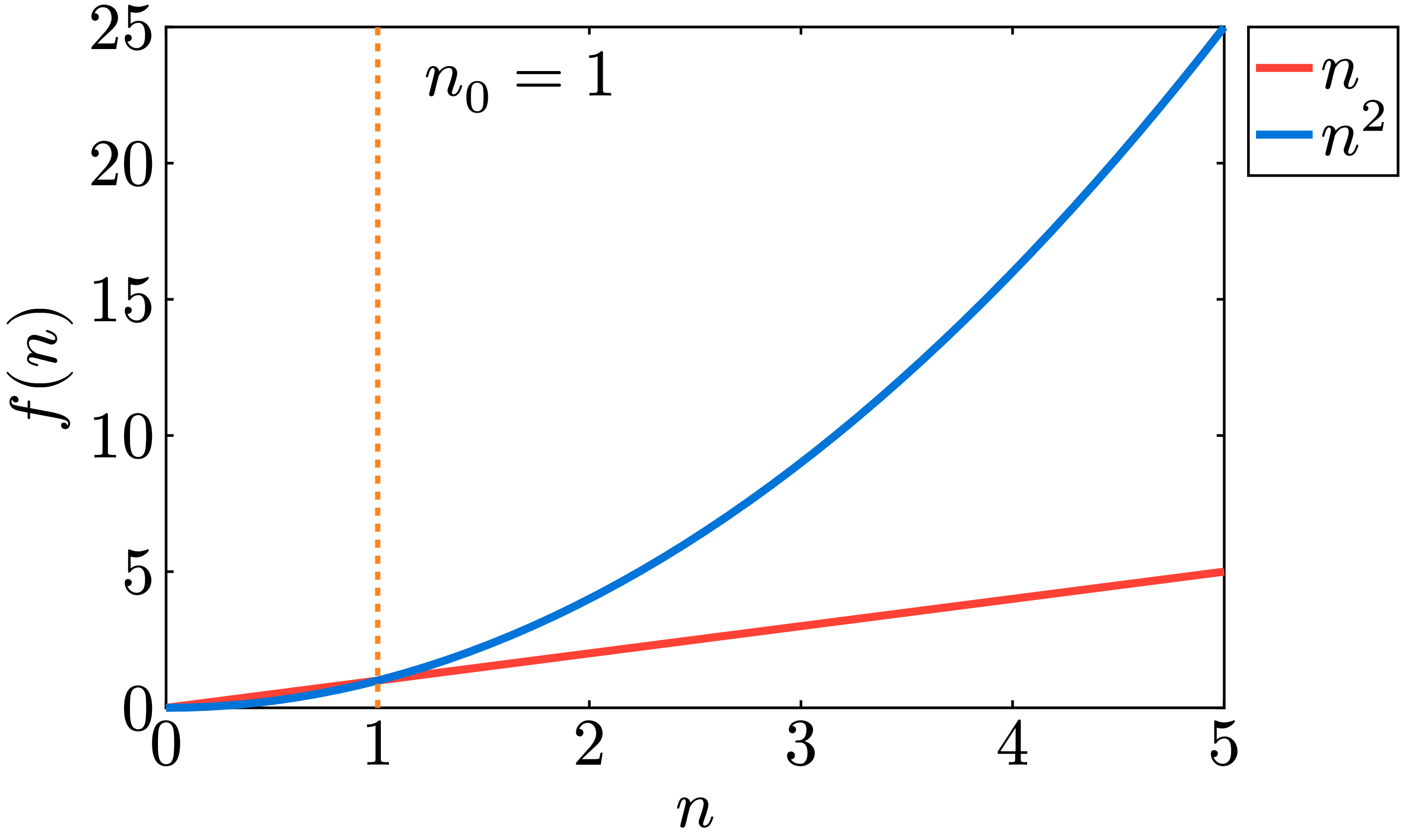
Otázka: Je $n \in \mathcal{O}(n^2)$?

Odpověď: Ano (c=1, n_0=1)

Vysvětlení: n je vždy menší než n^2 pro $n \geq 1$

$$n \leq 1 \cdot n^2 \quad \text{pro všechna } n \geq 1$$

Příklad 1



Příklad 2

Otázka: Je $n \in \mathcal{O}(0.5n)$?

Příklad 2

Otázka: Je $n \in \mathcal{O}(0.5n)$?

Odpověď:

Příklad 2

Otázka: Je $n \in \mathcal{O}(0.5n)$?

Odpověď: Ano

Příklad 2

Otázka: Je $n \in \mathcal{O}(0.5n)$?

Odpověď: Ano (c=3, n_0=1)

Příklad 2

Otázka: Je $n \in \mathcal{O}(0.5n)$?

Odpověď: Ano (c=3, n_0=1)

Vysvětlení:

Příklad 2

Otázka: Je $n \in \mathcal{O}(0.5n)$?

Odpověď: Ano (c=3, n_0=1)

Vysvětlení:

$$n \leq 3 \cdot 0.5n = 1.5n \quad \text{pro všechna } n \geq 1$$

Příklad 2

Otázka: Je $n \in \mathcal{O}(0.5n)$?

Odpověď: Ano (c=3, n_0=1)

Vysvětlení:

$$n \leq 3 \cdot 0.5n = 1.5n \quad \text{pro všechna } n \geq 1$$

Klíčový poznatek: Konstanty nerozhodují! $3 \cdot 0.5n = 1.5n > n$

Příklad 3

Otázka: Je $n^2 + 1 \in \mathcal{O}(n^2)$?

Příklad 3

Otázka: Je $n^2 + 1 \in \mathcal{O}(n^2)$?

Odpověď:

Příklad 3

Otázka: Je $n^2 + 1 \in \mathcal{O}(n^2)$?

Odpověď: Ano

Příklad 3

Otázka: Je $n^2 + 1 \in \mathcal{O}(n^2)$?

Odpověď: Ano (c=2, n_0=1)

Příklad 3

Otázka: Je $n^2 + 1 \in \mathcal{O}(n^2)$?

Odpověď: Ano (c=2, n_0=1)

Vysvětlení:

Příklad 3

Otázka: Je $n^2 + 1 \in \mathcal{O}(n^2)$?

Odpověď: Ano (c=2, n_0=1)

Vysvětlení:

Pro $n \geq 1$ platí:

Příklad 3

Otázka: Je $n^2 + 1 \in \mathcal{O}(n^2)$?

Odpověď: Ano (c=2, n_0=1)

Vysvětlení:

Pro $n \geq 1$ platí:

$$n^2 + 1 \leq n^2 + n^2 = 2n^2$$

Příklad 3

Otázka: Je $n^2 + 1 \in \mathcal{O}(n^2)$?

Odpověď: Ano (c=2, n_0=1)

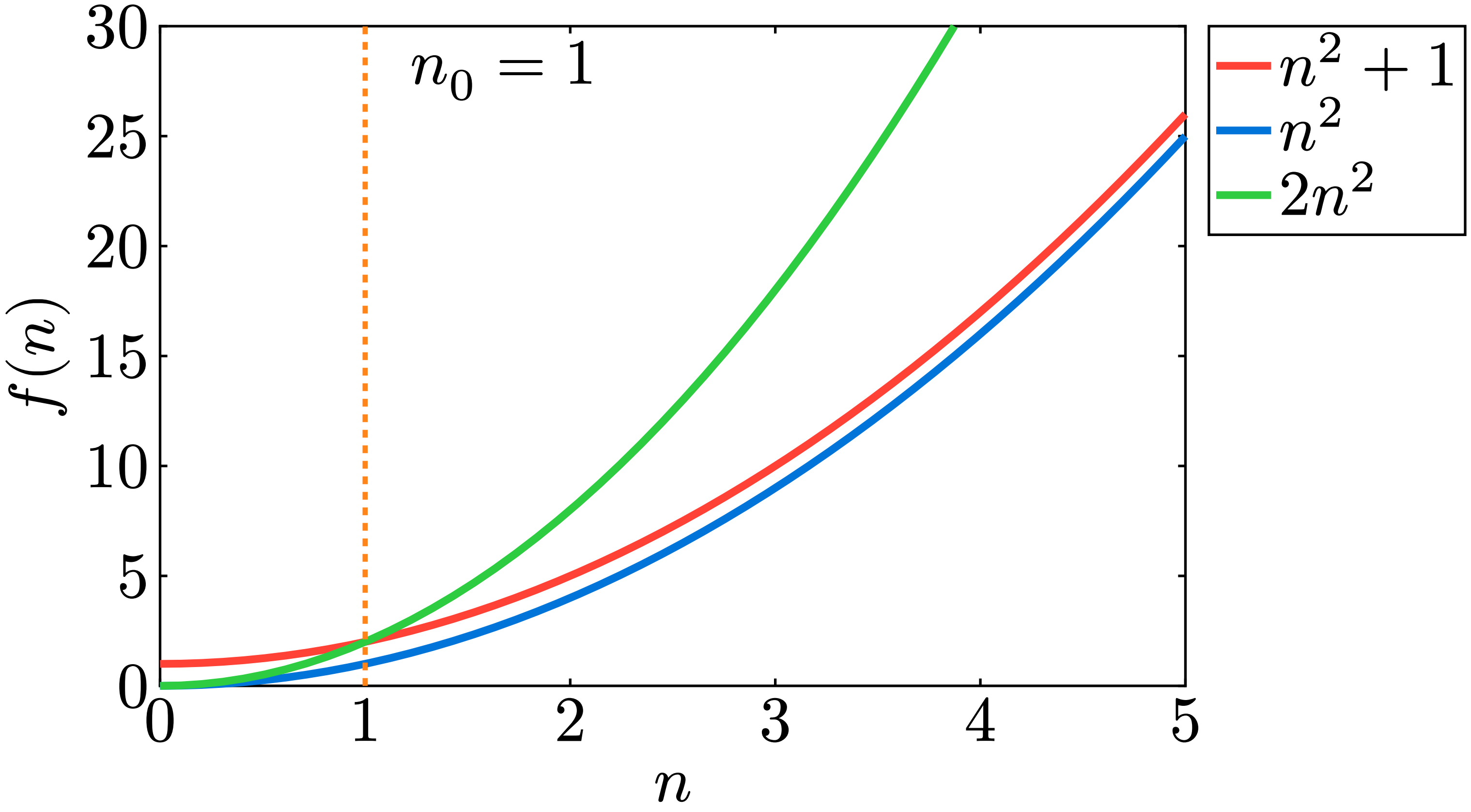
Vysvětlení:

Pro $n \geq 1$ platí:

$$n^2 + 1 \leq n^2 + n^2 = 2n^2$$

Tedy $n^2 + 1 \leq 2 \cdot n^2$ pro všechna $n \geq 1$

Příklad 3



Příklad 4

Otázka: Je $2^{n+5} \in \mathcal{O}(2^n)$?

Příklad 4

Otázka: Je $2^{n+5} \in \mathcal{O}(2^n)$?

Úprava:

Příklad 4

Otázka: Je $2^{n+5} \in \mathcal{O}(2^n)$?

Úprava:

$$2^{n+5} = 2^n \cdot 2^5 = 32 \cdot 2^n$$

Příklad 4

Otázka: Je $2^{n+5} \in \mathcal{O}(2^n)$?

Úprava:

$$2^{n+5} = 2^n \cdot 2^5 = 32 \cdot 2^n$$

Odpověď:

Příklad 4

Otázka: Je $2^{n+5} \in \mathcal{O}(2^n)$?

Úprava:

$$2^{n+5} = 2^n \cdot 2^5 = 32 \cdot 2^n$$

Odpověď: Ano

Příklad 4

Otázka: Je $2^{n+5} \in \mathcal{O}(2^n)$?

Úprava:

$$2^{n+5} = 2^n \cdot 2^5 = 32 \cdot 2^n$$

Odpověď: Ano (c=32, n_0=1)

Příklad 4

Otázka: Je $2^{n+5} \in \mathcal{O}(2^n)$?

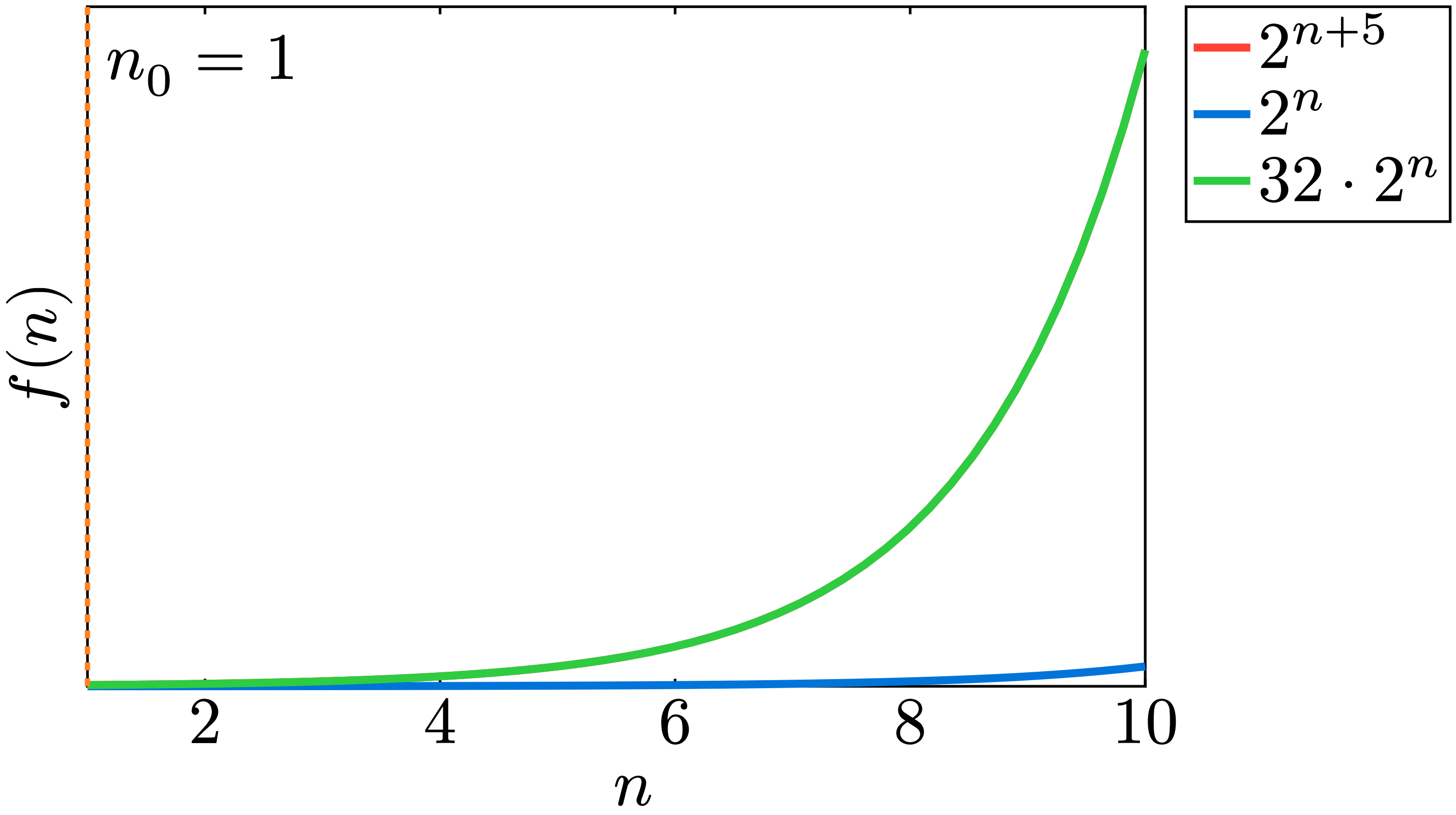
Úprava:

$$2^{n+5} = 2^n \cdot 2^5 = 32 \cdot 2^n$$

Odpověď: Ano (c=32, n_0=1)

Vzor: Konstanty v exponentu lze vytknout jako multiplikátor

Příklad 4



Příklad 5

Otázka: Je $1.5n + 2.2n^2 + 0.001n^3 \in \mathcal{O}(n^3)$?

Příklad 5

Otázka: Je $1.5n + 2.2n^2 + 0.001n^3 \in \mathcal{O}(n^3)$?

Odpověď:

Příklad 5

Otázka: Je $1.5n + 2.2n^2 + 0.001n^3 \in \mathcal{O}(n^3)$?

Odpověď: Ano

Příklad 5

Otázka: Je $1.5n + 2.2n^2 + 0.001n^3 \in \mathcal{O}(n^3)$?

Odpověď: Ano (c=4, n_0=10)

Příklad 5

Otázka: Je $1.5n + 2.2n^2 + 0.001n^3 \in \mathcal{O}(n^3)$?

Odpověď: Ano (c=4, n_0=10)

Vysvětlení:

Pro dostatečně velké n dominuje člen n^3:

Příklad 5

Otázka: Je $1.5n + 2.2n^2 + 0.001n^3 \in \mathcal{O}(n^3)$?

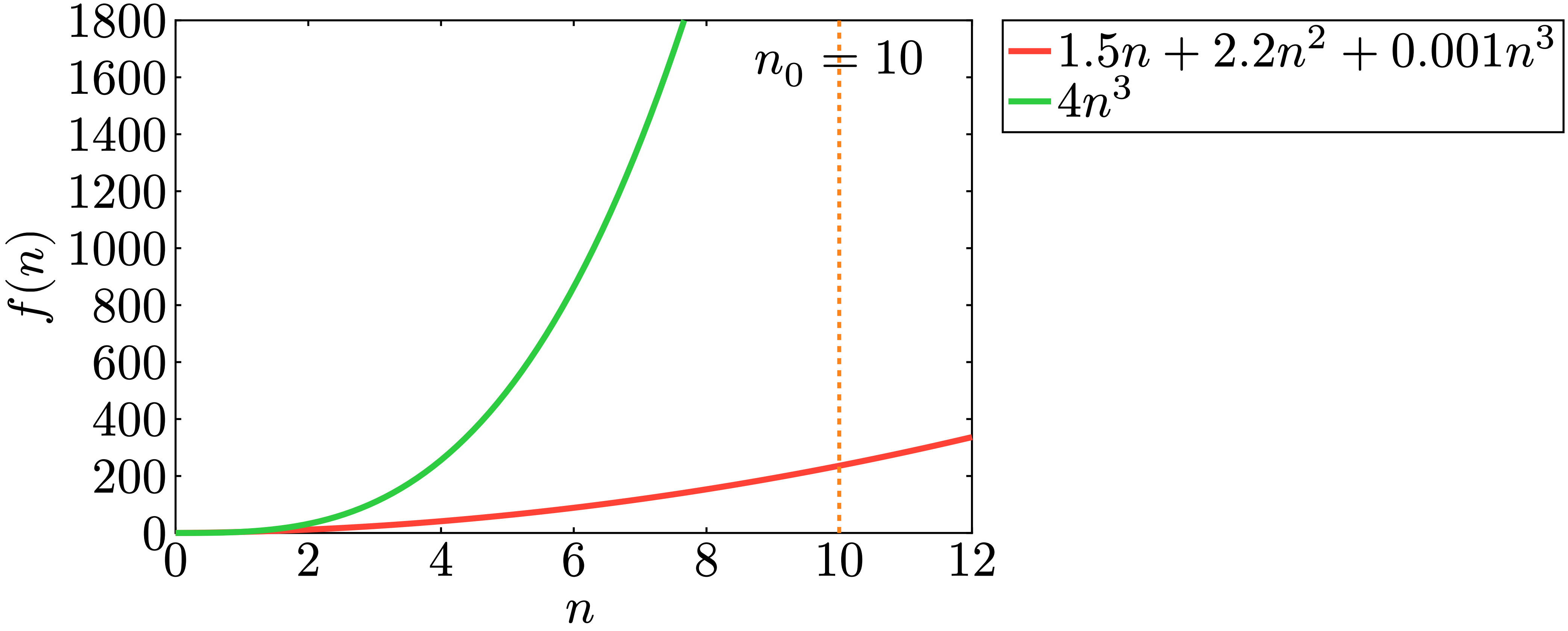
Odpověď: Ano (c=4, n_0=10)

Vysvětlení:

Pro dostatečně velké n dominuje člen n^3:

$$1.5n + 2.2n^2 + 0.001n^3 \leq 4n^3 \quad \text{pro } n \geq 10$$

Příklad 5



$n_0 = 10$
$1.5n + 2.2n^2 + 0.001n^3$
$4n^3$
$f(n)$
n
0
200
400
600
800
1000
1200
1400
1600
1800
0
2
4
6
8
10
12

Příklad 6

Otázka: Je $n^2 \in \mathcal{O}(n)$?

Příklad 6

Otázka: Je $n^2 \in \mathcal{O}(n)$?

Odpověď:

Příklad 6

Otázka: Je $n^2 \in \mathcal{O}(n)$?

Odpověď: Ne

Příklad 6

Otázka: Je $n^2 \in \mathcal{O}(n)$?

Odpověď: Ne

Důkaz

Příklad 6

Otázka: Je $n^2 \in \mathcal{O}(n)$?

Odpověď: Ne

Důkaz sporem

Důkaz sporem

Důkaz sporem

Předpokládejme, že $n^2 \in \mathcal{O}(n)$

Důkaz sporem

Předpokládejme, že $n^2 \in \mathcal{O}(n)$

Pak by existovaly konstanty $c > 0$ a $n_0 \in \mathbb{N}$ takové, že:

Důkaz sporem

Předpokládejme, že $n^2 \in \mathcal{O}(n)$

Pak by existovaly konstanty $c > 0$ a $n_0 \in \mathbb{N}$ takové, že:

$$n^2 \leq c \cdot n \quad \text{pro všechna } n \geq n_0$$

Důkaz sporem

Předpokládejme, že $n^2 \in \mathcal{O}(n)$

Pak by existovaly konstanty $c > 0$ a $n_0 \in \mathbb{N}$ takové, že:

$$n^2 \leq c \cdot n \quad \text{pro všechna } n \geq n_0$$

Vydělíme-li obě strany n (pro $n > 0$):

Důkaz sporem

Předpokládejme, že $n^2 \in \mathcal{O}(n)$

Pak by existovaly konstanty $c > 0$ a $n_0 \in \mathbb{N}$ takové, že:

$$n^2 \le c \cdot n \quad \text{pro všechna } n \ge n_0$$

Vydělíme-li obě strany n (pro $n > 0$): $n \le c$

Důkaz sporem

Předpokládejme, že $n^2 \in \mathcal{O}(n)$

Pak by existovaly konstanty $c > 0$ a $n_0 \in \mathbb{N}$ takové, že:

$$n^2 \leq c \cdot n \quad \text{pro všechna } n \geq n_0$$

Vydělíme-li obě strany n (pro $n > 0$): $n \leq c$

To znamená, že n je omezené konstantou c, což je **spor**, protože n může být libovolně velké

Důkaz sporem

Předpokládejme, že $n^2 \in \mathcal{O}(n)$

Pak by existovaly konstanty $c > 0$ a $n_0 \in \mathbb{N}$ takové, že:

$$n^2 \leq c \cdot n \quad \text{pro všechna } n \geq n_0$$

Vydělíme-li obě strany n (pro $n > 0$): $n \leq c$

To znamená, že n je omezené konstantou c, což je **spor**, protože n může být libovolně velké

Závěr: $n^2 \notin \mathcal{O}(n)$

Neformální pravidlo

Jak rychle určit Big O?

Neformální pravidlo

Jak rychle určit Big O?

1. V polynomu najdi **nejrychleji rostoucí člen**

Neformální pravidlo

Jak rychle určit Big O?

1. V polynomu najdi **nejrychleji rostoucí člen**
2. **Ostatní členy ignoruj**

Neformální pravidlo

Jak rychle určit Big O?

1. V polynomu najdi **nejrychleji rostoucí člen**
2. **Ostatní členy ignoruj**
3. **Ignoruj multiplikativní konstanty**

Neformální pravidlo

Jak rychle určit Big O?

1. V polynomu najdi **nejrychleji rostoucí člen**
2. **Ostatní členy ignoruj**
3. **Ignoruj multiplikativní konstanty**

Neformální pravidlo

Jak rychle určit Big O?

1. V polynomu najdi **nejrychleji rostoucí člen**
2. **Ostatní členy ignoruj**
3. **Ignoruj multiplikativní konstanty**

Příklad:

$$5n^3 + 100n^2 + 50n + 1000 \to$$

Neformální pravidlo

Jak rychle určit Big O?

1. V polynomu najdi **nejrychleji rostoucí člen**
2. **Ostatní členy ignoruj**
3. **Ignoruj multiplikativní konstanty**

Příklad:

$$5n^3 + 100n^2 + 50n + 1000 \to \mathcal{O}(n^3)$$

Cvičení

Určete Big O pro tyto funkce:

Cvičení

Určete Big O pro tyto funkce:

1. $5n + 100$

Cvičení

Určete Big O pro tyto funkce:

1. $5n + 100 \rightarrow \mathcal{O}(n)$

Cvičení

Určete Big O pro tyto funkce:

1. $5n + 100 \to \mathcal{O}(n)$, ale také $\mathcal{O}(n^2)$

Cvičení

Určete Big O pro tyto funkce:

1. $5n + 100 \rightarrow \mathcal{O}(n)$, ale také $\mathcal{O}(n^2)$ nebo $\mathcal{O}(n^3)$

Cvičení

Určete Big O pro tyto funkce:

1. $5n + 100 \to \mathcal{O}(n)$, ale také $\mathcal{O}\left(n^2\right)$ nebo $\mathcal{O}\left(n^3\right)$ atd.

Cvičení

Určete Big O pro tyto funkce:

1. $5n + 100 \rightarrow \mathcal{O}(n)$, ale také $\mathcal{O}(n^2)$ nebo $\mathcal{O}(n^3)$ atd.
2. $3n^2 + 2n + 1$

Cvičení

Určete Big O pro tyto funkce:

1. $5n + 100 \to \mathcal{O}(n)$, ale také $\mathcal{O}(n^2)$ nebo $\mathcal{O}(n^3)$ atd.
2. $3n^2 + 2n + 1 \to \mathcal{O}(n^2)$

Cvičení

Určete Big O pro tyto funkce:

1. $5n + 100 \to \mathcal{O}(n)$, ale také $\mathcal{O}(n^2)$ nebo $\mathcal{O}(n^3)$ atd.
2. $3n^2 + 2n + 1 \to \mathcal{O}(n^2)$
3. $n^3 + 1000n^2$

Cvičení

Určete Big O pro tyto funkce:

1. $5n + 100 \to \mathcal{O}(n)$, ale také $\mathcal{O}(n^2)$ nebo $\mathcal{O}(n^3)$ atd.
2. $3n^2 + 2n + 1 \to \mathcal{O}(n^2)$
3. $n^3 + 1000n^2 \to \mathcal{O}(n^3)$

Cvičení

Určete Big O pro tyto funkce:

1. $5n + 100 \rightarrow \mathcal{O}(n)$, ale také $\mathcal{O}(n^2)$ nebo $\mathcal{O}(n^3)$ atd.
2. $3n^2 + 2n + 1 \rightarrow \mathcal{O}(n^2)$
3. $n^3 + 1000n^2 \rightarrow \mathcal{O}(n^3)$
4. $2^n + n^{100}$

Cvičení

Určete Big O pro tyto funkce:

1. $5n + 100 \rightarrow \mathcal{O}(n)$, ale také $\mathcal{O}(n^2)$ nebo $\mathcal{O}(n^3)$ atd.
2. $3n^2 + 2n + 1 \rightarrow \mathcal{O}(n^2)$
3. $n^3 + 1000n^2 \rightarrow \mathcal{O}(n^3)$
4. $2^n + n^{100} \rightarrow \mathcal{O}(2^n)$

Cvičení

Určete Big O pro tyto funkce:

1. $5n + 100 \to \mathcal{O}(n)$, ale také $\mathcal{O}(n^2)$ nebo $\mathcal{O}(n^3)$ atd.
2. $3n^2 + 2n + 1 \to \mathcal{O}(n^2)$
3. $n^3 + 1000n^2 \to \mathcal{O}(n^3)$
4. $2^n + n^{100} \to \mathcal{O}(2^n)$

Pokud určujete Big O, tak ale většinou určujte „nejměnší“ možné

Asymptotická spodní mez - Omega

Intuitivní úvod

Omega říká:

Intuitivní úvod

Omega říká: „Můj algoritmus nebude rychlejší než…“

Intuitivní úvod

Omega říká: „Můj algoritmus nebude rychlejší než...“

Je to jako říct:

Intuitivní úvod

Omega říká: „Můj algoritmus nebude rychlejší než…“

Je to jako říct: „Dojedu tam minimálně za 30 minut“

Intuitivní úvod

Omega říká: „Můj algoritmus nebude rychlejší než...“

Je to jako říct: „Dojedu tam minimálně za 30 minut“

- Možná dojedu později

Intuitivní úvod

Omega říká: „Můj algoritmus nebude rychlejší než…“

Je to jako říct: „Dojedu tam minimálně za 30 minut“

- Možná dojedu později
- Ale určitě ne dřív

Intuitivní úvod

Omega říká: „Můj algoritmus nebude rychlejší než…“

Je to jako říct: „Dojedu tam minimálně za 30 minut“

- Možná dojedu později
- Ale určitě ne dřív

Je to opak Big O!

Formální definice

$$f(n) \in \Omega(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \geq c \cdot g(n)$$

Formální definice

$$f(n) \in \Omega(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \geq c \cdot g(n)$$

- $\exists c \in \mathbb{R}^+$:

Formální definice

$$f(n) \in \Omega(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \geq c \cdot g(n)$$

- $\exists c \in \mathbb{R}^+$: Existuje nějaká kladná konstanta c

Formální definice

$$f(n) \in \Omega(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \geq c \cdot g(n)$$

- $\exists c \in \mathbb{R}^+$: Existuje nějaká kladná konstanta c
- $\exists n_0 \in \mathbb{N}^+$:

Formální definice

$$f(n) \in \Omega(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \geq c \cdot g(n)$$

- $\exists c \in \mathbb{R}^+$: Existuje nějaká kladná konstanta c
- $\exists n_0 \in \mathbb{N}^+$: Existuje nějaké počáteční n_0

Formální definice

$$f(n) \in \Omega(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \geq c \cdot g(n)$$

- $\exists c \in \mathbb{R}^+$: Existuje nějaká kladná konstanta c
- $\exists n_0 \in \mathbb{N}^+$: Existuje nějaké počáteční n_0
- $\forall n \geq n_0$:

Formální definice

$$f(n) \in \Omega(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \geq c \cdot g(n)$$

- $\exists c \in \mathbb{R}^+$: Existuje nějaká kladná konstanta c
- $\exists n_0 \in \mathbb{N}^+$: Existuje nějaké počáteční n_0
- $\forall n \geq n_0$: Pro všechna n větší než n_0

Formální definice

$$f(n) \in \Omega(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \geq c \cdot g(n)$$

- $\exists c \in \mathbb{R}^+$: Existuje nějaká kladná konstanta c
- $\exists n_0 \in \mathbb{N}^+$: Existuje nějaké počáteční n_0
- $\forall n \geq n_0$: Pro všechna n větší než n_0
- $f(n) \geq c \cdot g(n)$:

Formální definice

$$f(n) \in \Omega(g(n)) \iff$$
$$(\exists c \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) : f(n) \geq c \cdot g(n)$$

- $\exists c \in \mathbb{R}^+$: Existuje nějaká kladná konstanta c
- $\exists n_0 \in \mathbb{N}^+$: Existuje nějaké počáteční n_0
- $\forall n \geq n_0$: Pro všechna n větší než n_0
- $f(n) \geq c \cdot g(n)$: Naše funkce f je větší než c-násobek g

Klíčový rozdíl

Big O vs. Omega:

Klíčový rozdíl

Big O vs. Omega:

- **Big O:**

Klíčový rozdíl

Big O vs. Omega:

- **Big O:** $f(n) \leq c \cdot g(n)$

Klíčový rozdíl

Big O vs. Omega:

- **Big O:** $f(n) \leq c \cdot g(n)$ - horní mez

Klíčový rozdíl

Big O vs. Omega:

- **Big O:** $f(n) \leq c \cdot g(n)$ - horní mez
- **Omega:**

Klíčový rozdíl

Big O vs. Omega:

- **Big O:** $f(n) \leq c \cdot g(n)$ - horní mez
- **Omega:** $f(n) \geq c \cdot g(n)$

Klíčový rozdíl

Big O vs. Omega:

- **Big O:** $f(n) \leq c \cdot g(n)$ - horní mez
- **Omega:** $f(n) \geq c \cdot g(n)$ - spodní mez

Příklad 1

Otázka: Je $n^2 \in \Omega(n)$?

Příklad 1

Otázka: Je $n^2 \in \Omega(n)$?

Odpověď:

Příklad 1

Otázka: Je $n^2 \in \Omega(n)$?

Odpověď: Ano

Příklad 1

Otázka: Je $n^2 \in \Omega(n)$?

Odpověď: Ano (c=1, n_0=1)

Příklad 1

Otázka: Je $n^2 \in \Omega(n)$?

Odpověď: Ano (c=1, n_0=1)

Vysvětlení:

Příklad 1

Otázka: Je $n^2 \in \Omega(n)$?

Odpověď: Ano (c=1, n_0=1)

Vysvětlení: n^2 roste rychleji než n, takže n je spodní mez

Příklad 1

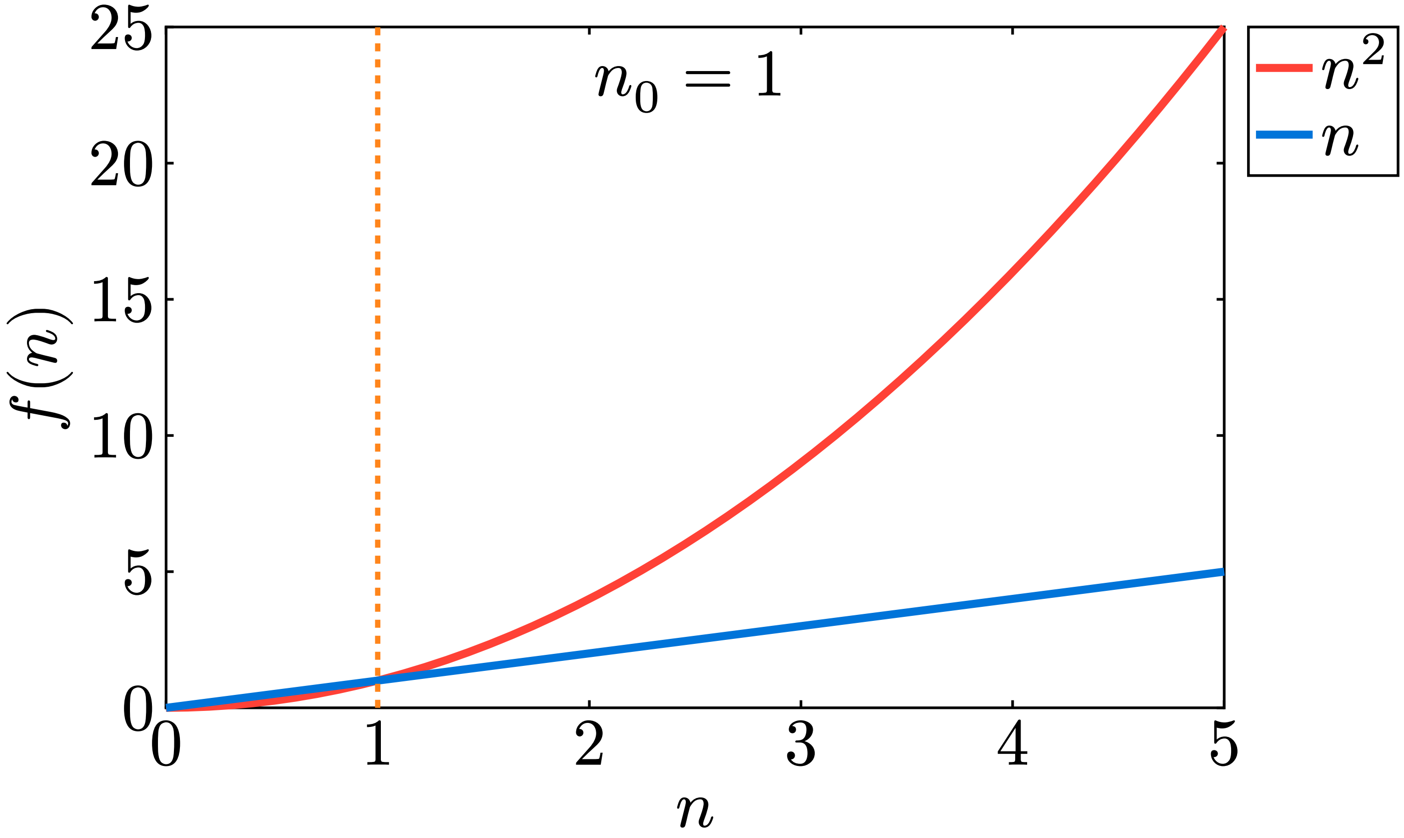
Otázka: Je $n^2 \in \Omega(n)$?

Odpověď: Ano (c=1, n_0=1)

Vysvětlení: n^2 roste rychleji než n, takže n je spodní mez

$$n^2 \geq 1 \cdot n \quad \text{pro všechna } n \geq 1$$

Příklad 1



Příklad 2

Otázka: Je $n^3 \in \Omega\left(1.5n + 2.2n^2 + 0.001n^3\right)$?

Příklad 2

Otázka: Je $n^3 \in \Omega(1.5n + 2.2n^2 + 0.001n^3)$?

Odpověď:

Příklad 2

Otázka: Je $n^3 \in \Omega\left(1.5n + 2.2n^2 + 0.001n^3\right)$?

Odpověď: Ano

Příklad 2

Otázka: Je $n^3 \in \Omega\big(1.5n + 2.2n^2 + 0.001n^3\big)$?

Odpověď: Ano (c=0.001, n_0=1)

Příklad 2

Otázka: Je $n^3 \in \Omega(1.5n + 2.2n^2 + 0.001n^3)$?

Odpověď: Ano (c=0.001, n_0=1)

Vysvětlení: Potřebujeme ukázat, že $n^3 \geq c \cdot (1.5n + 2.2n^2 + 0.001n^3)$ pro nějaké $c > 0$ a dostatečně velké n

Příklad 2

Pro $n \geq 1$ platí:

Příklad 2

Pro $n \geq 1$ platí:

$$n < n^3$$
$$n^2 < n^3$$
$$n^3 = n^3$$

Příklad 2

Tedy:

$$1.5n + 2.2n^2 + 0.001n^3 \leq$$

Příklad 2

Tedy:

$$1.5n + 2.2n^2 + 0.001n^3 \leq 1.5n^3 + 2.2n^3 + 0.001n^3$$

Příklad 2

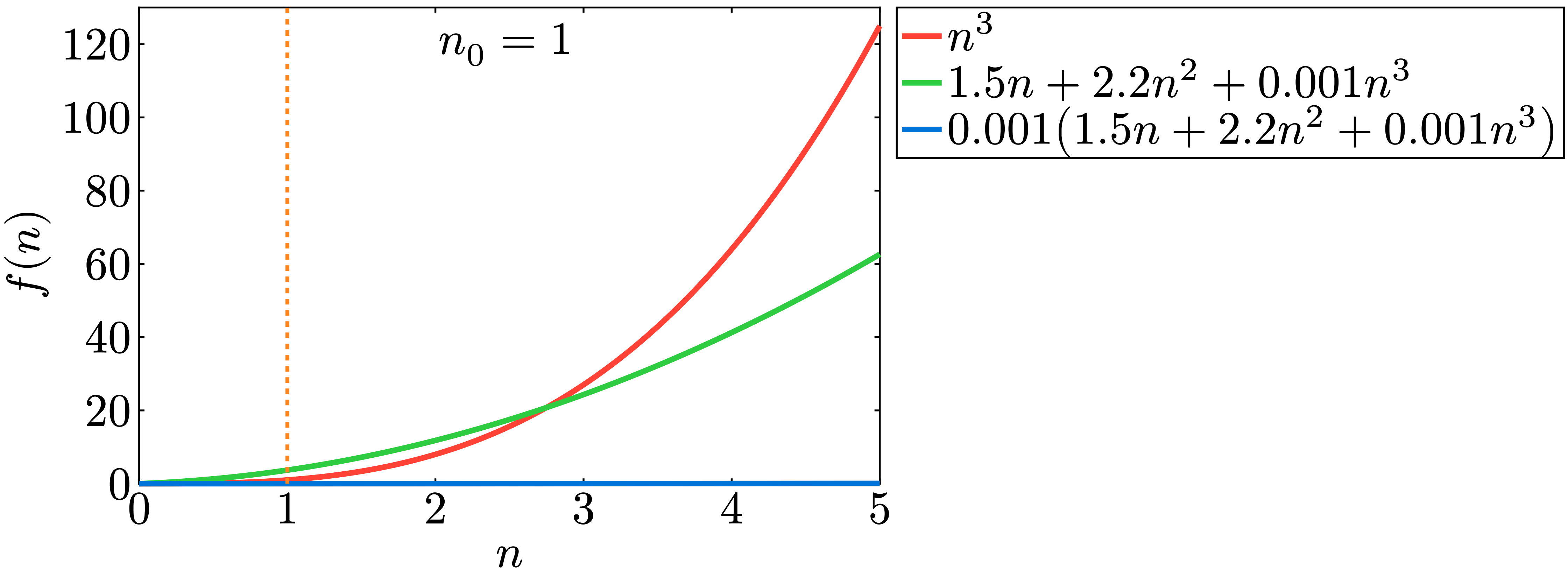
Tedy:

$$1.5n + 2.2n^2 + 0.001n^3 \leq 1.5n^3 + 2.2n^3 + 0.001n^3$$

$$\begin{aligned} n^3 &\geq 0.001 \cdot \left(1.5n + 2.2n^2 + 0.001n^3\right) \\ &\geq 0.0015n + 0.0022n^2 + 0.000001n^3 \\ &\geq 0.0015n^3 + 0.0022n^3 + 0.000001n^3 \\ &\geq 0.003701n^3 \end{aligned}$$

pro všechna $n \geq 1$

Příklad 2



$n_0 = 1$
n^3
$1.5n + 2.2n^2 + 0.001n^3$
$0.001(1.5n + 2.2n^2 + 0.001n^3)$
120
100
80
60
40
20
0
0
1
2
3
4
5
n
$f(n)$

Příklad 3

Otázka: Je $n \in \Omega(n^2)$?

Příklad 3

Otázka: Je $n \in \Omega(n^2)$?

Odpověď:

Příklad 3

Otázka: Je $n \in \Omega(n^2)$?

Odpověď: Ne

Příklad 3

Otázka: Je $n \in \Omega(n^2)$?

Odpověď: Ne

Důkaz

Příklad 3

Otázka: Je $n \in \Omega(n^2)$?

Odpověď: Ne

Důkaz sporem

Důkaz sporem

Důkaz sporem

Předpokládejme, že $n \in \Omega(n^2)$

Důkaz sporem

Předpokládejme, že $n \in \Omega(n^2)$

Pak by existovaly konstanty $c > 0$ a $n_0 \in \mathbb{N}$ takové, že:

Důkaz sporem

Předpokládejme, že $n \in \Omega(n^2)$

Pak by existovaly konstanty $c > 0$ a $n_0 \in \mathbb{N}$ takové, že:

$$n \geq c \cdot n^2 \quad \text{pro všechna } n \geq n_0$$

Důkaz sporem

Předpokládejme, že $n \in \Omega(n^2)$

Pak by existovaly konstanty $c > 0$ a $n_0 \in \mathbb{N}$ takové, že:

$$n \geq c \cdot n^2 \quad \text{pro všechna } n \geq n_0$$

Vydělíme-li obě strany n (pro $n > 0$):

Důkaz sporem

Předpokládejme, že $n \in \Omega(n^2)$

Pak by existovaly konstanty $c > 0$ a $n_0 \in \mathbb{N}$ takové, že:

$$n \geq c \cdot n^2 \quad \text{pro všechna } n \geq n_0$$

Vydělíme-li obě strany n (pro $n > 0$): $1 \geq c \cdot n$

Důkaz sporem

Předpokládejme, že $n \in \Omega(n^2)$

Pak by existovaly konstanty $c > 0$ a $n_0 \in \mathbb{N}$ takové, že:

$$n \geq c \cdot n^2 \quad \text{pro všechna } n \geq n_0$$

Vydělíme-li obě strany n (pro $n > 0$): $1 \geq c \cdot n$

To znamená, že $n \leq \frac{1}{c}$, tedy n je omezené konstantou, což je **spor**, protože n může být libovolně velké

Důkaz sporem

Předpokládejme, že $n \in \Omega(n^2)$

Pak by existovaly konstanty $c > 0$ a $n_0 \in \mathbb{N}$ takové, že:

$$n \geq c \cdot n^2 \quad \text{pro všechna } n \geq n_0$$

Vydělíme-li obě strany n (pro $n > 0$): $1 \geq c \cdot n$

To znamená, že $n \leq \frac{1}{c}$, tedy n je omezené konstantou, což je **spor**, protože n může být libovolně velké

Závěr: $n \notin \Omega(n^2)$

Praktický význam

Omega nám říká, že algoritmus nemůže být rychlejší

Praktický význam

Omega nám říká, že algoritmus nemůže být rychlejší

Příklad: Hledání v neseřazeném poli je $\Omega(n)$

Praktický význam

Omega nám říká, že algoritmus nemůže být rychlejší

Příklad: Hledání v neseřazeném poli je $\Omega(n)$

Proč?

Praktický význam

Omega nám říká, že algoritmus nemůže být rychlejší

Příklad: Hledání v neseřazeném poli je $\Omega(n)$

Proč? Musíme zkontrolovat každý prvek (v nejhorším případě)

Praktický význam

Omega nám říká, že algoritmus nemůže být rychlejší

Příklad: Hledání v neseřazeném poli je $\Omega(n)$

Proč? Musíme zkontrolovat každý prvek (v nejhorším případě)

Nemůžeme to udělat rychleji než $\Omega(n)$

Asymptotická těsná mez - Theta

Intuitivní úvod

Theta říká:

Intuitivní úvod

Theta říká: „Můj algoritmus bude trvat přesně...“

Intuitivní úvod

Theta říká: „Můj algoritmus bude trvat přesně…“

Je to když **Big O a Omega jsou**

Intuitivní úvod

Theta říká: „Můj algoritmus bude trvat přesně…“

Je to když **Big O a Omega jsou stejné**

Intuitivní úvod

Theta říká: „Můj algoritmus bude trvat přesně...“

Je to když **Big O a Omega jsou stejné**

Nejpřesnější popis složitosti

Vizuální diagram

$\mathcal{O}$ **(Big O)** – horní mez

$$\downarrow$$

Θ **(Theta)** – těsná mez

$$\uparrow$$

Ω **(Omega)** – spodní mez

Formální definice

$$f(n) \in \Theta(g(n)) \iff$$

$$(\exists c_1, c_2 \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) :$$

$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

Formální definice

$$f(n) \in \Theta(g(n)) \iff$$
$$(\exists c_1, c_2 \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) :$$
$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

- $\exists c_1, c_2 \in \mathbb{R}^+$:

Formální definice

$$f(n) \in \Theta(g(n)) \iff$$
$$(\exists c_1, c_2 \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) :$$
$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

- $\exists c_1, c_2 \in \mathbb{R}^+$: Existují dvě kladné konstanty c_1 a c_2

Formální definice

$$f(n) \in \Theta(g(n)) \iff$$
$$(\exists c_1, c_2 \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) :$$
$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

- $\exists c_1, c_2 \in \mathbb{R}^+$: Existují dvě kladné konstanty c_1 a c_2
- $\exists n_0 \in \mathbb{N}^+$:

Formální definice

$$f(n) \in \Theta(g(n)) \iff$$
$$(\exists c_1, c_2 \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) :$$
$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

- $\exists c_1, c_2 \in \mathbb{R}^+$: Existují dvě kladné konstanty c_1 a c_2
- $\exists n_0 \in \mathbb{N}^+$: Existuje nějaké počáteční n_0

Formální definice

$$f(n) \in \Theta(g(n)) \iff$$
$$(\exists c_1, c_2 \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) :$$
$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

- $\exists c_1, c_2 \in \mathbb{R}^+$: Existují dvě kladné konstanty c_1 a c_2
- $\exists n_0 \in \mathbb{N}^+$: Existuje nějaké počáteční n_0
- $\forall n \geq n_0$:

Formální definice

$$f(n) \in \Theta(g(n)) \iff$$
$$(\exists c_1, c_2 \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) :$$
$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

- $\exists c_1, c_2 \in \mathbb{R}^+$: Existují dvě kladné konstanty c_1 a c_2
- $\exists n_0 \in \mathbb{N}^+$: Existuje nějaké počáteční n_0
- $\forall n \geq n_0$: Pro všechna n větší než n_0

Formální definice

$$f(n) \in \Theta(g(n)) \iff$$
$$(\exists c_1, c_2 \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) :$$
$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

- $\exists c_1, c_2 \in \mathbb{R}^+$: Existují dvě kladné konstanty c_1 a c_2
- $\exists n_0 \in \mathbb{N}^+$: Existuje nějaké počáteční n_0
- $\forall n \geq n_0$: Pro všechna n větší než n_0
- $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$:

Formální definice

$$f(n) \in \Theta(g(n)) \iff$$
$$(\exists c_1, c_2 \in \mathbb{R}^+)(\exists n_0 \in \mathbb{N}^+)(\forall n \geq n_0) :$$
$$c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$$

- $\exists c_1, c_2 \in \mathbb{R}^+$: Existují dvě kladné konstanty c_1 a c_2
- $\exists n_0 \in \mathbb{N}^+$: Existuje nějaké počáteční n_0
- $\forall n \geq n_0$: Pro všechna n větší než n_0
- $c_1 \cdot g(n) \leq f(n) \leq c_2 \cdot g(n)$: Funkce f je „sevřená“ mezi $c_1 \cdot g$ a $c_2 \cdot g$

Příklad

Otázka: Je $\ln\left(\frac{n}{2}\right) \in \Theta(\ln(n))$?

Příklad

Otázka: Je $\ln\left(\frac{n}{2}\right) \in \Theta(\ln(n))$?

Úprava:

Příklad

Otázka: Je $\ln\left(\frac{n}{2}\right) \in \Theta(\ln(n))$?

Úprava:

$$\ln\left(\frac{n}{2}\right) = \ln(n) - \ln(2)$$

Příklad

Otázka: Je $\ln(\frac{n}{2}) \in \Theta(\ln(n))$?

Úprava:

$$\ln\left(\frac{n}{2}\right) = \ln(n) - \ln(2)$$

Pro velké n je $\ln(2)$ zanedbatelné

Příklad

Otázka: Je $\ln(\frac{n}{2}) \in \Theta(\ln(n))$?

Úprava:

$$\ln\left(\frac{n}{2}\right) = \ln(n) - \ln(2)$$

Pro velké n je $\ln(2)$ zanedbatelné

Odpověď:

Příklad

Otázka: Je $\ln\left(\frac{n}{2}\right) \in \Theta(\ln(n))$?

Úprava:

$$\ln\left(\frac{n}{2}\right) = \ln(n) - \ln(2)$$

Pro velké n je $\ln(2)$ zanedbatelné

Odpověď: Ano

Příklad

Otázka: Je $\ln(\frac{n}{2}) \in \Theta(\ln(n))$?

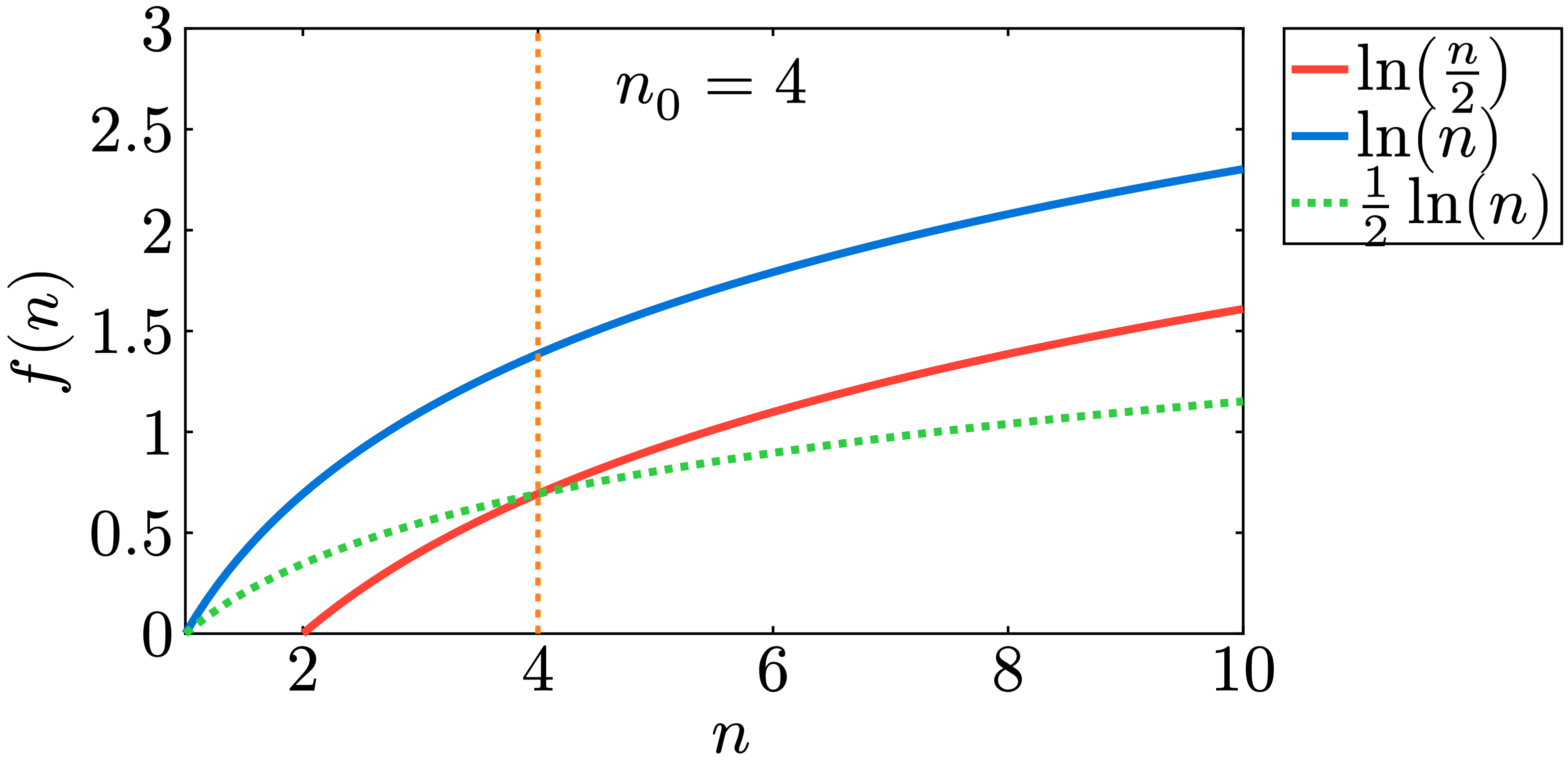
Úprava:

$$\ln\left(\frac{n}{2}\right) = \ln(n) - \ln(2)$$

Pro velké n je $\ln(2)$ zanedbatelné

Odpověď: Ano (c_1=1/2, c_2=1, n_0=4)

Příklad



Praktický příklad

Praktický příklad

```c
// This algorithm is Θ(n)
for (int i = 0; i < n; i++) {
    printf("%d\n", i);
}
```

Praktický příklad

```c
// This algorithm is Θ(n)
for (int i = 0; i < n; i++) {
    printf("%d\n", i);
}
```

- Best case:

Praktický příklad

```c
// This algorithm is Θ(n)
for (int i = 0; i < n; i++) {
    printf("%d\n", i);
}
```

- Best case: $\Theta(n)$ - musíme projít všechny prvky

Praktický příklad

```c
// This algorithm is Θ(n)
for (int i = 0; i < n; i++) {
    printf("%d\n", i);
}
```

- Best case: $\Theta(n)$ - musíme projít všechny prvky
- Worst case:

Praktický příklad

```c
// This algorithm is Θ(n)
for (int i = 0; i < n; i++) {
    printf("%d\n", i);
}
```

- Best case: $\Theta(n)$ - musíme projít všechny prvky
- Worst case: $\Theta(n)$ - musíme projít všechny prvky

Praktický příklad

```c
// This algorithm is Θ(n)
for (int i = 0; i < n; i++) {
    printf("%d\n", i);
}
```

- Best case: $\Theta(n)$ - musíme projít všechny prvky
- Worst case: $\Theta(n)$ - musíme projít všechny prvky
- Average case:

Praktický příklad

```c
// This algorithm is Θ(n)
for (int i = 0; i < n; i++) {
    printf("%d\n", i);
}
```

- Best case: $\Theta(n)$ - musíme projít všechny prvky
- Worst case: $\Theta(n)$ - musíme projít všechny prvky
- Average case: $\Theta(n)$ - musíme projít všechny prvky

Praktický příklad

```c
// This algorithm is Θ(n)
for (int i = 0; i < n; i++) {
    printf("%d\n", i);
}
```

- Best case: $\Theta(n)$ - musíme projít všechny prvky
- Worst case: $\Theta(n)$ - musíme projít všechny prvky
- Average case: $\Theta(n)$ - musíme projít všechny prvky

Vždy stejné! $\rightarrow$ Theta

Srovnání notací

Shrnutí

Notace	Význam	Analogie

Shrnutí

Notace	Význam	Analogie
$\mathcal{O}(g(n))$		

Shrnutí

Notace	Význam	Analogie
$\mathcal{O}(g(n))$	Horní mez	

Shrnutí

Notace	Význam	Analogie
$\mathcal{O}(g(n))$	Horní mez	„Maximálně tak pomalé“

Shrnutí

Notace	Význam	Analogie
$\mathcal{O}(g(n))$	Horní mez	„Maximálně tak pomalé“
$\Omega(g(n))$		

Shrnutí

Notace	Význam	Analogie
$\mathcal{O}(g(n))$	Horní mez	„Maximálně tak pomalé“
$\Omega(g(n))$	Spodní mez	

Shrnutí

Notace	Význam	Analogie
$\mathcal{O}(g(n))$	Horní mez	„Maximálně tak pomalé“
$\Omega(g(n))$	Spodní mez	„Minimálně tak rychlé“

Shrnutí

Notace	Význam	Analogie
$\mathcal{O}(g(n))$	Horní mez	„Maximálně tak pomalé“
$\Omega(g(n))$	Spodní mez	„Minimálně tak rychlé“
$\Theta(g(n))$		

Shrnutí

Notace	Význam	Analogie
$\mathcal{O}(g(n))$	Horní mez	„Maximálně tak pomalé“
$\Omega(g(n))$	Spodní mez	„Minimálně tak rychlé“
$\Theta(g(n))$	Těsná mez	

Shrnutí

Notace	Význam	Analogie
$\mathcal{O}(g(n))$	Horní mez	„Maximálně tak pomalé“
$\Omega(g(n))$	Spodní mez	„Minimálně tak rychlé“
$\Theta(g(n))$	Těsná mez	„Přesně tak rychlé“

Vztahy mezi notacemi

Pokud $f \in \Theta(g)$,

Vztahy mezi notacemi

Pokud $f \in \Theta(g)$, **pak** $f \in \mathcal{O}(g)$ **a zároveň** $f \in \Omega(g)$

Vztahy mezi notacemi

Pokud $f \in \Theta(g)$, **pak** $f \in \mathcal{O}(g)$ **a zároveň** $f \in \Omega(g)$

Ale $f \in \mathcal{O}(g)$ **neznamená** $f \in \Theta(g)$

Vztahy mezi notacemi

Pokud $f \in \Theta(g)$, **pak** $f \in \mathcal{O}(g)$ **a zároveň** $f \in \Omega(g)$

Ale $f \in \mathcal{O}(g)$ **neznamená** $f \in \Theta(g)$

Příklad:

Vztahy mezi notacemi

Pokud $f \in \Theta(g)$, **pak** $f \in \mathcal{O}(g)$ **a zároveň** $f \in \Omega(g)$

Ale $f \in \mathcal{O}(g)$ **neznamená** $f \in \Theta(g)$

Příklad:

- $n \in \mathcal{O}(n^2)$

Vztahy mezi notacemi

Pokud $f \in \Theta(g)$, **pak** $f \in \mathcal{O}(g)$ **a zároveň** $f \in \Omega(g)$

Ale $f \in \mathcal{O}(g)$ **neznamená** $f \in \Theta(g)$

Příklad:

- $n \in \mathcal{O}(n^2)$ ✓ (pravda, ale není těsné)

Vztahy mezi notacemi

Pokud $f \in \Theta(g)$, **pak** $f \in \mathcal{O}(g)$ **a zároveň** $f \in \Omega(g)$

Ale $f \in \mathcal{O}(g)$ **neznamená** $f \in \Theta(g)$

Příklad:

- $n \in \mathcal{O}(n^2)$ ✓ (pravda, ale není těsné)
- $n \in \Theta(n^2)$

Vztahy mezi notacemi

Pokud $f \in \Theta(g)$, **pak** $f \in \mathcal{O}(g)$ **a zároveň** $f \in \Omega(g)$

Ale $f \in \mathcal{O}(g)$ **neznamená** $f \in \Theta(g)$

Příklad:

- $n \in \mathcal{O}(n^2)$ ✓ (pravda, ale není těsné)
- $n \in \Theta(n^2)$ ✕ (nepravda, není dostatečně těsné)

Vztahy mezi notacemi

Pokud $f \in \Theta(g)$, **pak** $f \in \mathcal{O}(g)$ **a zároveň** $f \in \Omega(g)$

Ale $f \in \mathcal{O}(g)$ **neznamená** $f \in \Theta(g)$

Příklad:

- $n \in \mathcal{O}(n^2)$ ✓ (pravda, ale není těsné)
- $n \in \Theta(n^2)$ ✗ (nepravda, není dostatečně těsné)
- $n \in \Theta(n)$

Vztahy mezi notacemi

Pokud $f \in \Theta(g)$**, pak** $f \in \mathcal{O}(g)$ **a zároveň** $f \in \Omega(g)$

Ale $f \in \mathcal{O}(g)$ **neznamená** $f \in \Theta(g)$

Příklad:

- $n \in \mathcal{O}(n^2)$ ✓ (pravda, ale není těsné)
- $n \in \Theta(n^2)$ ✕ (nepravda, není dostatečně těsné)
- $n \in \Theta(n)$ ✓ (pravda)

Časová vs. Prostorová složitost

Dva rozměry složitosti

Časová složitost (Time Complexity):

Dva rozměry složitosti

Časová složitost (Time Complexity):

- Kolik času (operací) algoritmus potřebuje

Dva rozměry složitosti

Časová složitost (Time Complexity):

- Kolik času (operací) algoritmus potřebuje
- Měříme počet základních operací

Dva rozměry složitosti

Časová složitost (Time Complexity):

- Kolik času (operací) algoritmus potřebuje
- Měříme počet základních operací
- Nejčastěji analyzovaná

Dva rozměry složitosti

Časová složitost (Time Complexity):

- Kolik času (operací) algoritmus potřebuje
- Měříme počet základních operací
- Nejčastěji analyzovaná

Prostorová složitost (Space Complexity):

Dva rozměry složitosti

Časová složitost (Time Complexity):

- Kolik času (operací) algoritmus potřebuje
- Měříme počet základních operací
- Nejčastěji analyzovaná

Prostorová složitost (Space Complexity):

- Kolik paměti algoritmus potřebuje

Dva rozměry složitosti

Časová složitost (Time Complexity):

- Kolik času (operací) algoritmus potřebuje
- Měříme počet základních operací
- Nejčastěji analyzovaná

Prostorová složitost (Space Complexity):

- Kolik paměti algoritmus potřebuje
- Měříme velikost použité paměti

Dva rozměry složitosti

Časová složitost (Time Complexity):

- Kolik času (operací) algoritmus potřebuje
- Měříme počet základních operací
- Nejčastěji analyzovaná

Prostorová složitost (Space Complexity):

- Kolik paměti algoritmus potřebuje
- Měříme velikost použité paměti
- Důležitá pro embedded systémy, velká data

Příklad 1: Součet pole

Příklad 1: Součet pole

```c
int sum(int arr[], int n) {
    int total = 0;
    for (int i = 0; i < n; i++) {
        total += arr[i];
    }
    return total;
}
```

Příklad 1: Součet pole

```c
int sum(int arr[], int n) {
    int total = 0;
    for (int i = 0; i < n; i++) {
        total += arr[i];
    }
    return total;
}
```

- **Časová:**

Příklad 1: Součet pole

```
int sum(int arr[], int n) {
    int total = 0;
    for (int i = 0; i < n; i++) {
        total += arr[i];
    }
    return total;
}
```

- **Časová:** $\mathcal{O}(n)$ - procházíme všechny prvky

Příklad 1: Součet pole

```
int sum(int arr[], int n) {
    int total = 0;
    for (int i = 0; i < n; i++) {
        total += arr[i];
    }
    return total;
}
```

- **Časová:** $\mathcal{O}(n)$ - procházíme všechny prvky
- **Prostorová:**

Příklad 1: Součet pole

```
int sum(int arr[], int n) {
    int total = 0;
    for (int i = 0; i < n; i++) {
        total += arr[i];
    }
    return total;
}
```

- **Časová:** $\mathcal{O}(n)$ - procházíme všechny prvky
- **Prostorová:** $\mathcal{O}(1)$ - používáme jen konstantní paměť (total, i)

Příklad 2: Kopírování pole

Příklad 2: Kopírování pole

```c
int* copy_array(int arr[], int n) {
    int* new_arr = malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        new_arr[i] = arr[i];
    }
    return new_arr;
}
```

Příklad 2: Kopírování pole

```
int* copy_array(int arr[], int n) {
    int* new_arr = malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        new_arr[i] = arr[i];
    }
    return new_arr;
}
```

- **Časová:**

Příklad 2: Kopírování pole

```
int* copy_array(int arr[], int n) {
    int* new_arr = malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        new_arr[i] = arr[i];
    }
    return new_arr;
}
```

- **Časová:** $\mathcal{O}(n)$ - kopírujeme všechny prvky

Příklad 2: Kopírování pole

```
int* copy_array(int arr[], int n) {
    int* new_arr = malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        new_arr[i] = arr[i];
    }
    return new_arr;
}
```

- **Časová:** $\mathcal{O}(n)$ - kopírujeme všechny prvky
- **Prostorová:**

Příklad 2: Kopírování pole

```
int* copy_array(int arr[], int n) {
    int* new_arr = malloc(n * sizeof(int));
    for (int i = 0; i < n; i++) {
        new_arr[i] = arr[i];
    }
    return new_arr;
}
```

- **Časová:** $\mathcal{O}(n)$ - kopírujeme všechny prvky
- **Prostorová:** $\mathcal{O}(n)$ - alokujeme nové pole velikosti n

Trade-off

Někdy můžeme **vyměnit čas za paměť** (nebo naopak)

Trade-off

Někdy můžeme **vyměnit čas za paměť** (nebo naopak)

Příklad: Memoizace

Trade-off

Někdy můžeme **vyměnit čas za paměť** (nebo naopak)

Příklad: Memoizace

- Použijeme více paměti k uložení výsledků

Trade-off

Někdy můžeme **vyměnit čas za paměť** (nebo naopak)

Příklad: Memoizace

- Použijeme více paměti k uložení výsledků
- Ušetříme čas tím, že je nemusíme počítat znovu

Trade-off

Někdy můžeme **vyměnit čas za paměť** (nebo naopak)

Příklad: Memoizace

- Použijeme více paměti k uložení výsledků
- Ušetříme čas tím, že je nemusíme počítat znovu

Fibonacci s memoizací:

Trade-off

Někdy můžeme **vyměnit čas za paměť** (nebo naopak)

Příklad: Memoizace

- Použijeme více paměti k uložení výsledků
- Ušetříme čas tím, že je nemusíme počítat znovu

Fibonacci s memoizací:

- Časová: $\mathcal{O}(2^n) \to \mathcal{O}(n)$

Trade-off

Někdy můžeme **vyměnit čas za paměť** (nebo naopak)

Příklad: Memoizace

- Použijeme více paměti k uložení výsledků
- Ušetříme čas tím, že je nemusíme počítat znovu

Fibonacci s memoizací:

- Časová: $\mathcal{O}(2^n) \to \mathcal{O}(n)$
- Prostorová: $\mathcal{O}(1) \to \mathcal{O}(n)$

Praktické příklady v C

Příklad 1: Konstantní čas

Příklad 1: Konstantní čas

```c
int get_first(int arr[], int n) {
    return arr[0];  // Always 1 operation
}
```

Příklad 1: Konstantní čas

```c
int get_first(int arr[], int n) {
    return arr[0];  // Always 1 operation
}
```

- Nezáleží na velikosti pole

Příklad 1: Konstantní čas

```c
int get_first(int arr[], int n) {
    return arr[0];  // Always 1 operation
}
```

- Nezáleží na velikosti pole
- Vždy stejně rychlé

Příklad 1: Konstantní čas

```c
int get_first(int arr[], int n) {
    return arr[0];  // Always 1 operation
}
```

- Nezáleží na velikosti pole
- Vždy stejně rychlé
- **Složitost:**

Příklad 1: Konstantní čas

```c
int get_first(int arr[], int n) {
    return arr[0];  // Always 1 operation
}
```

- Nezáleží na velikosti pole
- Vždy stejně rychlé
- **Složitost:** $\mathcal{O}(1)$

Příklad 2: Lineární čas

Příklad 2: Lineární čas

```c
int find_max(int arr[], int n) {
    int max = arr[0];
    for (int i = 1; i < n; i++) {  // n-1 iterations
        if (arr[i] > max) max = arr[i];
    }
    return max;
}
```

Příklad 2: Lineární čas

```c
int find_max(int arr[], int n) {
    int max = arr[0];
    for (int i = 1; i < n; i++) {  // n-1 iterations
        if (arr[i] > max) max = arr[i];
    }
    return max;
}
```

- Musíme projít všechny prvky

Příklad 2: Lineární čas

```
int find_max(int arr[], int n) {
    int max = arr[0];
    for (int i = 1; i < n; i++) {  // n-1 iterations
        if (arr[i] > max) max = arr[i];
    }
    return max;
}
```

- Musíme projít všechny prvky
- **Složitost:** $\mathcal{O}(n)$

Příklad 3: Kvadratický čas

Příklad 3: Kvadratický čas

```c
// Find all pairs of elements
void print_all_pairs(int arr[], int n) {
    for (int i = 0; i < n; i++) {        // n iterations
        for (int j = 0; j < n; j++) {    // n iterations
            printf("(%d, %d)\n", arr[i], arr[j]);
        }
    }
}
```

Příklad 3: Kvadratický čas

```c
// Find all pairs of elements
void print_all_pairs(int arr[], int n) {
    for (int i = 0; i < n; i++) {        // n iterations
        for (int j = 0; j < n; j++) {    // n iterations
            printf("(%d, %d)\n", arr[i], arr[j]);
        }
    }
}
```

- Vnořené cykly: $n \cdot n = n^2$

Příklad 3: Kvadratický čas

```c
// Find all pairs of elements
void print_all_pairs(int arr[], int n) {
    for (int i = 0; i < n; i++) {        // n iterations
        for (int j = 0; j < n; j++) {    // n iterations
            printf("(%d, %d)\n", arr[i], arr[j]);
        }
    }
}
```

- Vnořené cykly: $n \cdot n = n^2$
- **Složitost:** $\mathcal{O}(n^2)$

Příklad 4: Logaritmický čas

Příklad 4: Logaritmický čas

```
int binary_search(int arr[], int n, int x) {
    int left = 0, right = n - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == x) return mid;
        if (arr[mid] < x) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}
```

Příklad 4: Logaritmický čas

```
int binary_search(int arr[], int n, int x) {
    int left = 0, right = n - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == x) return mid;
        if (arr[mid] < x) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}
```

Každá iterace půlí prostor hledání

Příklad 4: Logaritmický čas

```
int binary_search(int arr[], int n, int x) {
    int left = 0, right = n - 1;
    while (left <= right) {
        int mid = left + (right - left) / 2;
        if (arr[mid] == x) return mid;
        if (arr[mid] < x) left = mid + 1;
        else right = mid - 1;
    }
    return -1;
}
```

Každá iterace půlí prostor hledání - **Složitost:** $\mathcal{O}(\log n)$

Příklad 5: Exponenciální čas

Příklad 5: Exponenciální čas

```c
int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);  // Two recursive calls
}
```

Příklad 5: Exponenciální čas

```
int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);  // Two recursive calls
}
```

- Každé volání vytváří 2 další volání

Příklad 5: Exponenciální čas

```c
int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);  // Two recursive calls
}
```

- Každé volání vytváří 2 další volání
- Exponenciální růst: 2, 4, 8, 16, 32, …

Příklad 5: Exponenciální čas

```
int fib(int n) {
    if (n <= 1) return n;
    return fib(n-1) + fib(n-2);  // Two recursive calls
}
```

- Každé volání vytváří 2 další volání
- Exponenciální růst: 2, 4, 8, 16, 32, …
- **Složitost:** $\mathcal{O}(2^n)$

Srovnání

Algoritmus	Složitost	Pro $n = 1000$

Srovnání

Algoritmus	Složitost	Pro $n = 1000$
`get_first`		

Srovnání

Algoritmus	Složitost	Pro $n = 1000$
get_first	$\mathcal{O}(1)$	

Srovnání

Algoritmus	Složitost	Pro $n = 1000$
`get_first`	$\mathcal{O}(1)$	1 operace

Srovnání

Algoritmus	Složitost	Pro $n = 1000$
`get_first`	$\mathcal{O}(1)$	1 operace
`find_max`		

Srovnání

Algoritmus	Složitost	Pro $n = 1000$
`get_first`	$\mathcal{O}(1)$	1 operace
`find_max`	$\mathcal{O}(n)$	

Srovnání

Algoritmus	Složitost	Pro $n = 1000$
`get_first`	$\mathcal{O}(1)$	1 operace
`find_max`	$\mathcal{O}(n)$	1 000 operací

Srovnání

Algoritmus	Složitost	Pro $n = 1000$
`get_first`	$\mathcal{O}(1)$	1 operace
`find_max`	$\mathcal{O}(n)$	1 000 operací
`binary_search`		

Srovnání

Algoritmus	Složitost	Pro $n = 1000$
`get_first`	$\mathcal{O}(1)$	1 operace
`find_max`	$\mathcal{O}(n)$	1 000 operací
`binary_search`	$\mathcal{O}(\log n)$	

Srovnání

Algoritmus	Složitost	Pro $n = 1000$
`get_first`	$\mathcal{O}(1)$	1 operace
`find_max`	$\mathcal{O}(n)$	1 000 operací
`binary_search`	$\mathcal{O}(\log n)$	10 operací

Srovnání

Algoritmus	Složitost	Pro $n = 1000$
`get_first`	$\mathcal{O}(1)$	1 operace
`find_max`	$\mathcal{O}(n)$	1 000 operací
`binary_search`	$\mathcal{O}(\log n)$	10 operací
`print_all_pairs`		

Srovnání

Algoritmus	Složitost	Pro $n = 1000$
`get_first`	$\mathcal{O}(1)$	1 operace
`find_max`	$\mathcal{O}(n)$	1 000 operací
`binary_search`	$\mathcal{O}(\log n)$	10 operací
`print_all_pairs`	$\mathcal{O}(n^2)$	

Srovnání

Algoritmus	Složitost	Pro $n = 1000$
`get_first`	$\mathcal{O}(1)$	1 operace
`find_max`	$\mathcal{O}(n)$	1 000 operací
`binary_search`	$\mathcal{O}(\log n)$	10 operací
`print_all_pairs`	$\mathcal{O}(n^2)$	1 000 000 operací

Srovnání

Algoritmus	Složitost	Pro $n = 1000$
`get_first`	$\mathcal{O}(1)$	1 operace
`find_max`	$\mathcal{O}(n)$	1 000 operací
`binary_search`	$\mathcal{O}(\log n)$	10 operací
`print_all_pairs`	$\mathcal{O}(n^2)$	1 000 000 operací
`fib` (recursive)		

Srovnání

Algoritmus	Složitost	Pro $n = 1000$
`get_first`	$\mathcal{O}(1)$	1 operace
`find_max`	$\mathcal{O}(n)$	1 000 operací
`binary_search`	$\mathcal{O}(\log n)$	10 operací
`print_all_pairs`	$\mathcal{O}(n^2)$	1 000 000 operací
`fib` (recursive)	$\mathcal{O}(2^n)$	

Srovnání

Algoritmus	Složitost	Pro $n = 1000$
`get_first`	$\mathcal{O}(1)$	1 operace
`find_max`	$\mathcal{O}(n)$	1 000 operací
`binary_search`	$\mathcal{O}(\log n)$	10 operací
`print_all_pairs`	$\mathcal{O}(n^2)$	1 000 000 operací
`fib` (recursive)	$\mathcal{O}(2^n)$	Číslo s 302 dekadickými číslicemi

Jak určit složitost algoritmu?

Praktická pravidla

Praktická pravidla

1. Žádné cykly:

Praktická pravidla

1. **Žádné cykly:** $\mathcal{O}(1)$

Praktická pravidla

1. Žádné cykly: $\mathcal{O}(1)$

```
int x = arr[0] + arr[1];
```

Praktická pravidla

1. Žádné cykly: $\mathcal{O}(1)$

```
int x = arr[0] + arr[1];
```

2. Jeden cyklus:

Praktická pravidla

1. Žádné cykly: $\mathcal{O}(1)$

```
int x = arr[0] + arr[1];
```

2. Jeden cyklus: $\mathcal{O}(n)$

Praktická pravidla

1. Žádné cykly: $\mathcal{O}(1)$

```
int x = arr[0] + arr[1];
```

2. Jeden cyklus: $\mathcal{O}(n)$

```
for (int i = 0; i < n; i++) { ... }
```

Praktická pravidla

1. Žádné cykly: $\mathcal{O}(1)$

```
int x = arr[0] + arr[1];
```

2. Jeden cyklus: $\mathcal{O}(n)$

```
for (int i = 0; i < n; i++) { ... }
```

3. Dva vnořené cykly:

Praktická pravidla

1. Žádné cykly: $\mathcal{O}(1)$

```
int x = arr[0] + arr[1];
```

2. Jeden cyklus: $\mathcal{O}(n)$

```
for (int i = 0; i < n; i++) { ... }
```

3. Dva vnořené cykly: $\mathcal{O}(n^2)$

Praktická pravidla

1. Žádné cykly: $\mathcal{O}(1)$

```
int x = arr[0] + arr[1];
```

2. Jeden cyklus: $\mathcal{O}(n)$

```
for (int i = 0; i < n; i++) { ... }
```

3. Dva vnořené cykly: $\mathcal{O}(n^2)$

```
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) { ... }
}
```

Praktická pravidla

4. Půlení prostoru:

Praktická pravidla

4. Půlení prostoru: $\mathcal{O}(\log n)$

Praktická pravidla

4. Půlení prostoru: $\mathcal{O}(\log n)$

```
while (n > 1) {
    n = n / 2;
}
```

Praktická pravidla

4. Půlení prostoru: $\mathcal{O}(\log n)$

```
while (n > 1) {
    n = n / 2;
}
```

5. Cyklus + půlení:

Praktická pravidla

4. Půlení prostoru: $\mathcal{O}(\log n)$

```
while (n > 1) {
    n = n / 2;
}
```

5. Cyklus + půlení: $\mathcal{O}(n \log n)$

Praktická pravidla

4. Půlení prostoru: $\mathcal{O}(\log n)$

```
while (n > 1) {
    n = n / 2;
}
```

5. Cyklus + půlení: $\mathcal{O}(n \log n)$

```
for (int i = 0; i < n; i++) {
    int x = n;
    while (x > 1) { x = x / 2; }
}
```

Praktická pravidla

6. Rekurze s větvením:

Praktická pravidla

6. Rekurze s větvením: Často $\mathcal{O}(2^n)$ nebo horší

Praktická pravidla

6. Rekurze s větvením: Často $\mathcal{O}(2^n)$ nebo horší

```
if (n <= 1) return n;
return func(n-1) + func(n-2);  // Two branches
```

Cvičení

Určete složitost těchto algoritmů:

Cvičení

Určete složitost těchto algoritmů:

```c
// 1.
for (int i = 0; i < n; i++) {
    printf("%d\n", arr[i]);
}
```

Cvičení

Určete složitost těchto algoritmů:

```c
// 1.
for (int i = 0; i < n; i++) {
    printf("%d\n", arr[i]);
}
```

Složitost:

Cvičení

Určete složitost těchto algoritmů:

```c
// 1.
for (int i = 0; i < n; i++) {
    printf("%d\n", arr[i]);
}
```

Složitost: $\mathcal{O}(n)$

Cvičení

```c
// 2.
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        for (int k = 0; k < n; k++) {
            printf("%d\n", arr[i] + arr[j] + arr[k]);
        }
    }
}
```

Cvičení

```c
// 2.
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        for (int k = 0; k < n; k++) {
            printf("%d\n", arr[i] + arr[j] + arr[k]);
        }
    }
}
```

Složitost:

Cvičení

```c
// 2.
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        for (int k = 0; k < n; k++) {
            printf("%d\n", arr[i] + arr[j] + arr[k]);
        }
    }
}
```

Složitost: $\mathcal{O}(n^3)$

Cvičení

```c
// 3.
int i = n;
while (i > 0) {
    printf("%d\n", i);
    i = i / 2;
}
```

Cvičení

```c
// 3.
int i = n;
while (i > 0) {
    printf("%d\n", i);
    i = i / 2;
}
```

Složitost:

Cvičení

```c
// 3.
int i = n;
while (i > 0) {
    printf("%d\n", i);
    i = i / 2;
}
```

Složitost: $\mathcal{O}(\log n)$

Cvičení

```c
// 4.
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        printf("%d %d\n", i, j);
    }
}
```

Cvičení

```c
// 4.
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        printf("%d %d\n", i, j);
    }
}
```

Složitost:

Cvičení

```
// 4.
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        printf("%d %d\n", i, j);
    }
}
```

Složitost: $\mathcal{O}(n^2)$

Cvičení

```c
// 4.
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        printf("%d %d\n", i, j);
    }
}
```

Složitost: $\mathcal{O}(n^2)$

Vysvětlení:

Cvičení

```c
// 4.
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        printf("%d %d\n", i, j);
    }
}
```

Složitost: $\mathcal{O}(n^2)$

Vysvětlení: Vnitřní cyklus běží od `i` do `n`. Celkový počet iterací: $n + (n-1) + (n-2) + \ldots + 1 = \frac{n(n+1)}{2} \approx \frac{n^2}{2}$

Cvičení

```c
// 4.
for (int i = 0; i < n; i++) {
    for (int j = i; j < n; j++) {
        printf("%d %d\n", i, j);
    }
}
```

Složitost: $\mathcal{O}(n^2)$

Vysvětlení: Vnitřní cyklus běží od `i` do `n`. Celkový počet iterací: $n + (n-1) + (n-2) + \ldots + 1 = \frac{n(n+1)}{2} \approx \frac{n^2}{2} \to$ Konstanty ignorujeme $\to \mathcal{O}(n^2)$

Cvičení

```c
// 5.
for (int i = 1; i < n; i *= 2) {
    for (int j = 0; j < n; j++) {
        printf("%d\n", arr[j]);
    }
}
```

Cvičení

```c
// 5.
for (int i = 1; i < n; i *= 2) {
    for (int j = 0; j < n; j++) {
        printf("%d\n", arr[j]);
    }
}
```

Složitost:

Cvičení

```c
// 5.
for (int i = 1; i < n; i *= 2) {
    for (int j = 0; j < n; j++) {
        printf("%d\n", arr[j]);
    }
}
```

Složitost: $\mathcal{O}(n \log n)$

Cvičení

```c
// 5.
for (int i = 1; i < n; i *= 2) {
    for (int j = 0; j < n; j++) {
        printf("%d\n", arr[j]);
    }
}
```

Složitost: $\mathcal{O}(n \log n)$

Vysvětlení:

Cvičení

```c
// 5.
for (int i = 1; i < n; i *= 2) {
    for (int j = 0; j < n; j++) {
        printf("%d\n", arr[j]);
    }
}
```

Složitost: $\mathcal{O}(n \log n)$

Vysvětlení: Vnější cyklus: `i` se zdvojnásobuje $(1, 2, 4, 8, \ldots)$

Cvičení

```c
// 5.
for (int i = 1; i < n; i *= 2) {
    for (int j = 0; j < n; j++) {
        printf("%d\n", arr[j]);
    }
}
```

Složitost: $\mathcal{O}(n \log n)$

Vysvětlení: Vnější cyklus: `i` se zdvojnásobuje $(1, 2, 4, 8, \ldots) \rightarrow \mathcal{O}(\log n)$ iterací.

Cvičení

```c
// 5.
for (int i = 1; i < n; i *= 2) {
    for (int j = 0; j < n; j++) {
        printf("%d\n", arr[j]);
    }
}
```

Složitost: $\mathcal{O}(n \log n)$

Vysvětlení: Vnější cyklus: `i` se zdvojnásobuje $(1, 2, 4, 8, \ldots) \to \mathcal{O}(\log n)$ iterací. Vnitřní cyklus: vždy zpracuje celé pole

Cvičení

```c
// 5.
for (int i = 1; i < n; i *= 2) {
    for (int j = 0; j < n; j++) {
        printf("%d\n", arr[j]);
    }
}
```

Složitost: $\mathcal{O}(n \log n)$

Vysvětlení: Vnější cyklus: `i` se zdvojnásobuje $(1, 2, 4, 8, \ldots) \to \mathcal{O}(\log n)$ iterací. Vnitřní cyklus: vždy zpracuje celé pole $\to \mathcal{O}(n)$ operací.

Cvičení

```c
// 5.
for (int i = 1; i < n; i *= 2) {
    for (int j = 0; j < n; j++) {
        printf("%d\n", arr[j]);
    }
}
```

Složitost: $\mathcal{O}(n \log n)$

Vysvětlení: Vnější cyklus: `i` se zdvojnásobuje $(1, 2, 4, 8, \ldots) \to \mathcal{O}(\log n)$ iterací. Vnitřní cyklus: vždy zpracuje celé pole $\to \mathcal{O}(n)$ operací.
Celkem: $\mathcal{O}(\log n) \times \mathcal{O}(n) = \mathcal{O}(n \log n)$

Cvičení

```c
// 6.
for (int i = 1; i < n; i++) {
    if (i * i > n) {
        break;
    }
    printf("%d\n", i);
}
```

Cvičení

```c
// 6.
for (int i = 1; i < n; i++) {
    if (i * i > n) {
        break;
    }
    printf("%d\n", i);
}
```

Složitost:

Cvičení

```c
// 6.
for (int i = 1; i < n; i++) {
    if (i * i > n) {
        break;
    }
    printf("%d\n", i);
}
```

Složitost: $\mathcal{O}(\sqrt{n})$

Cvičení

```c
// 6.
for (int i = 1; i < n; i++) {
    if (i * i > n) {
        break;
    }
    printf("%d\n", i);
}
```

Složitost: $\mathcal{O}(\sqrt{n})$

Vysvětlení:

Cvičení

```c
// 6.
for (int i = 1; i < n; i++) {
    if (i * i > n) {
        break;
    }
    printf("%d\n", i);
}
```

Složitost: $\mathcal{O}(\sqrt{n})$

Vysvětlení: Cyklus vypadá jako $\mathcal{O}(n)$, ale podmínka `i * i > n` ho ukončí, když

Cvičení

```c
// 6.
for (int i = 1; i < n; i++) {
    if (i * i > n) {
        break;
    }
    printf("%d\n", i);
}
```

Složitost: $\mathcal{O}(\sqrt{n})$

Vysvětlení: Cyklus vypadá jako $\mathcal{O}(n)$, ale podmínka `i * i > n` ho ukončí, když $i^2 > n$

Cvičení

```c
// 6.
for (int i = 1; i < n; i++) {
    if (i * i > n) {
        break;
    }
    printf("%d\n", i);
}
```

Složitost: $\mathcal{O}(\sqrt{n})$

Vysvětlení: Cyklus vypadá jako $\mathcal{O}(n)$, ale podmínka `i * i > n` ho ukončí, když $i^2 > n \Leftrightarrow i > \sqrt{n}$.

Cvičení

```
// 6.
for (int i = 1; i < n; i++) {
    if (i * i > n) {
        break;
    }
    printf("%d\n", i);
}
```

Složitost: $\mathcal{O}(\sqrt{n})$

Vysvětlení: Cyklus vypadá jako $\mathcal{O}(n)$, ale podmínka `i * i > n` ho ukončí, když $i^2 > n \Leftrightarrow i > \sqrt{n}$. Cyklus tedy běží pouze do $\sqrt{n}$

Cvičení

```c
// 7.
int i = n;
while (i > 1) {
    printf("%d\n", i);
    i = i / 2;
}
int j = n;
while (j > 1) {
    printf("%d\n", j);
    j = j / 2;
}
```

Cvičení

Složitost:

Cvičení

Složitost: $\mathcal{O}(\log n)$

Cvičení

Složitost: $\mathcal{O}(\log n)$

Vysvětlení:

Cvičení

Složitost: $\mathcal{O}(\log n)$

Vysvětlení: První cyklus: $\mathcal{O}(\log n)$, druhý cyklus: $\mathcal{O}(\log n)$.

Cvičení

Složitost: $\mathcal{O}(\log n)$

Vysvětlení: První cyklus: $\mathcal{O}(\log n)$, druhý cyklus: $\mathcal{O}(\log n)$.
Cykly běží **sekvenčně** (ne vnořeně), takže

Cvičení

Složitost: $\mathcal{O}(\log n)$

Vysvětlení: První cyklus: $\mathcal{O}(\log n)$, druhý cyklus: $\mathcal{O}(\log n)$.
Cykly běží **sekvenčně** (ne vnořeně), takže $\mathcal{O}(\log n) + \mathcal{O}(\log n) = \mathcal{O}(\log n)$.

Cvičení

Složitost: $\mathcal{O}(\log n)$

Vysvětlení: První cyklus: $\mathcal{O}(\log n)$, druhý cyklus: $\mathcal{O}(\log n)$.
Cykly běží **sekvenčně** (ne vnořeně), takže $\mathcal{O}(\log n) + \mathcal{O}(\log n) = \mathcal{O}(\log n)$.
Při sčítání stejných řádů dominuje nejvyšší člen.

Časté chyby a mýty

Chyba 1: Konstanty

Chybné tvrzení: $\mathcal{O}(2n)$ je jiné než $\mathcal{O}(n)$

Chyba 1: Konstanty

Chybné tvrzení: $\mathcal{O}(2n)$ je jiné než $\mathcal{O}(n)$

✕ **Špatně!** Konstanty ignorujeme

Chyba 1: Konstanty

Chybné tvrzení: $\mathcal{O}(2n)$ je jiné než $\mathcal{O}(n)$

✗ **Špatně!** Konstanty ignorujeme

✓ **Správně:** $\mathcal{O}(2n) = \mathcal{O}(n)$

Chyba 1: Konstanty

Chybné tvrzení: $\mathcal{O}(2n)$ je jiné než $\mathcal{O}(n)$

✗ **Špatně!** Konstanty ignorujeme

✓ **Správně:** $\mathcal{O}(2n) = \mathcal{O}(n)$

Proč?

Chyba 1: Konstanty

Chybné tvrzení: $\mathcal{O}(2n)$ je jiné než $\mathcal{O}(n)$

✕ **Špatně!** Konstanty ignorujeme

✓ **Správně:** $\mathcal{O}(2n) = \mathcal{O}(n)$

Proč? Asymptotická notace se zajímá o **růst**, ne absolutní hodnoty

Chyba 2: Konstantní členy

Chybné tvrzení: $\mathcal{O}(n + 100)$ je jiné než $\mathcal{O}(n)$

Chyba 2: Konstantní členy

Chybné tvrzení: $\mathcal{O}(n + 100)$ je jiné než $\mathcal{O}(n)$

✕ **Špatně!** Konstantní členy ignorujeme

Chyba 2: Konstantní členy

Chybné tvrzení: $\mathcal{O}(n+100)$ je jiné než $\mathcal{O}(n)$

✗ **Špatně!** Konstantní členy ignorujeme

✓ **Správně:** $\mathcal{O}(n+100) = \mathcal{O}(n)$

Chyba 2: Konstantní členy

Chybné tvrzení: $\mathcal{O}(n + 100)$ je jiné než $\mathcal{O}(n)$

✕ **Špatně!** Konstantní členy ignorujeme

✓ **Správně:** $\mathcal{O}(n + 100) = \mathcal{O}(n)$

Proč?

Chyba 2: Konstantní členy

Chybné tvrzení: $\mathcal{O}(n + 100)$ je jiné než $\mathcal{O}(n)$

✕ **Špatně!** Konstantní členy ignorujeme

✓ **Správně:** $\mathcal{O}(n + 100) = \mathcal{O}(n)$

Proč? Pro velké n je +100 zanedbatelné

Chyba 3: Rychlejší počítač

Tvrzení: „Rychlejší počítač změní složitost“

Chyba 3: Rychlejší počítač

Tvrzení: „Rychlejší počítač změní složitost"

× **Špatně!** Složitost je o růstu, ne absolutním čase

Chyba 3: Rychlejší počítač

Tvrzení: „Rychlejší počítač změní složitost“

✕ **Špatně!** Složitost je o růstu, ne absolutním čase

✓ **Správně:** Rychlejší počítač zrychlí všechno stejně, ale $\mathcal{O}(n^2)$ zůstane $\mathcal{O}(n^2)$

Chyba 3: Rychlejší počítač

Tvrzení: „Rychlejší počítač změní složitost“

× **Špatně!** Složitost je o růstu, ne absolutním čase

✓ **Správně:** Rychlejší počítač zrychlí všechno stejně, ale $\mathcal{O}(n^2)$ zůstane $\mathcal{O}(n^2)$

- Starý počítač: $\mathcal{O}(n^2)$ trvá 1 hodinu

Chyba 3: Rychlejší počítač

Tvrzení: „Rychlejší počítač změní složitost"

✕ **Špatně!** Složitost je o růstu, ne absolutním čase

✓ **Správně:** Rychlejší počítač zrychlí všechno stejně, ale $\mathcal{O}(n^2)$ zůstane $\mathcal{O}(n^2)$

- Starý počítač: $\mathcal{O}(n^2)$ trvá 1 hodinu
- Nový počítač (10× rychlejší): $\mathcal{O}(n^2)$ trvá 6 minut

Chyba 3: Rychlejší počítač

Tvrzení: „Rychlejší počítač změní složitost"

✗ **Špatně!** Složitost je o růstu, ne absolutním čase

✓ **Správně:** Rychlejší počítač zrychlí všechno stejně, ale $\mathcal{O}(n^2)$ zůstane $\mathcal{O}(n^2)$

- Starý počítač: $\mathcal{O}(n^2)$ trvá 1 hodinu
- Nový počítač (10× rychlejší): $\mathcal{O}(n^2)$ trvá 6 minut
- Ale pořád $\mathcal{O}(n^2)$!

Chyba 4: Big O a rozpad případů

Big O může mluvit také o nejlepším a průměrném případu

Chyba 4: Big O a rozpad případů

Big O může mluvit také o nejlepším a průměrném případu

- **Best case** $\mathcal{O}(\ldots)$

Chyba 4: Big O a rozpad případů

Big O může mluvit také o nejlepším a průměrném případu

- **Best case** $\mathcal{O}(\ldots)$
- **Average case** $\mathcal{O}(\ldots)$

Chyba 4: Big O a rozpad případů

Big O může mluvit také o nejlepším a průměrném případu

- **Best case** $\mathcal{O}(...)$
- **Average case** $\mathcal{O}(...)$
- **Worst case** $\mathcal{O}(...)$ - standardní Big O

Chyba 4: Big O a rozpad případů

Big O může mluvit také o nejlepším a průměrném případu

- **Best case** $\mathcal{O}(\dots)$
- **Average case** $\mathcal{O}(\dots)$
- **Worst case** $\mathcal{O}(\dots)$ - standardní Big O

Příklad: Lineární vyhledávání

Chyba 4: Big O a rozpad případů

Big O může mluvit také o nejlepším a průměrném případu

- **Best case** $\mathcal{O}(...)$
- **Average case** $\mathcal{O}(...)$
- **Worst case** $\mathcal{O}(...)$ - standardní Big O

Příklad: Lineární vyhledávání

- Best case: $\mathcal{O}(1)$ - prvek je na začátku

Chyba 4: Big O a rozpad případů

Big O může mluvit také o nejlepším a průměrném případu

- **Best case** $\mathcal{O}(...)$
- **Average case** $\mathcal{O}(...)$
- **Worst case** $\mathcal{O}(...)$ - standardní Big O

Příklad: Lineární vyhledávání

- Best case: $\mathcal{O}(1)$ - prvek je na začátku
- Average case: $\mathcal{O}(n)$ - prvek je uprostřed

Chyba 4: Big O a rozpad případů

Big O může mluvit také o nejlepším a průměrném případu

- **Best case** $\mathcal{O}(...)$
- **Average case** $\mathcal{O}(...)$
- **Worst case** $\mathcal{O}(...)$ - standardní Big O

Příklad: Lineární vyhledávání

- Best case: $\mathcal{O}(1)$ - prvek je na začátku
- Average case: $\mathcal{O}(n)$ - prvek je uprostřed
- Worst case: $\mathcal{O}(n)$ - prvek je na konci nebo není tam

Shrnutí

Klíčové poznatky

1. **Složitost měří růst**, ne absolutní čas

Klíčové poznatky

1. **Složitost měří růst**, ne absolutní čas
2. **Tři hlavní notace:**

Klíčové poznatky

1. **Složitost měří růst**, ne absolutní čas
2. **Tři hlavní notace:**

- $\mathcal{O}(g)$ - horní mez („maximálně tak pomalé“)

Klíčové poznatky

1. **Složitost měří růst**, ne absolutní čas

2. **Tři hlavní notace:**

- $\mathcal{O}(g)$ - horní mez („maximálně tak pomalé“)
- $\Omega(g)$ - spodní mez („minimálně tak rychlé“)

Klíčové poznatky

1. **Složitost měří růst**, ne absolutní čas
2. **Tři hlavní notace:**

- $\mathcal{O}(g)$ - horní mez („maximálně tak pomalé“)
- $\Omega(g)$ - spodní mez („minimálně tak rychlé“)
- $\Theta(g)$ - těsná mez („přesně tak rychlé“)

Klíčové poznatky

1. **Složitost měří růst**, ne absolutní čas

2. **Tři hlavní notace:**

- $\mathcal{O}(g)$ - horní mez („maximálně tak pomalé“)
- $\Omega(g)$ - spodní mez („minimálně tak rychlé“)
- $\Theta(g)$ - těsná mez („přesně tak rychlé“)

3. **Ignorujeme konstanty** a nižší řády

Klíčové poznatky

4. **Běžné složitosti** (od nejlepší k nejhorší):

Klíčové poznatky

4. **Běžné složitosti** (od nejlepší k nejhorší):

$$\mathcal{O}(1) < \mathcal{O}(\log n) < \mathcal{O}(n) < \mathcal{O}(n \log n) < \mathcal{O}(n^2) < \mathcal{O}(2^n) < \mathcal{O}(n!)$$

Klíčové poznatky

4. **Běžné složitosti** (od nejlepší k nejhorší):

   $$\mathcal{O}(1) < \mathcal{O}(\log n) < \mathcal{O}(n) < \mathcal{O}(n \log n) < \mathcal{O}(n^2) < \mathcal{O}(2^n) < \mathcal{O}(n!)$$

5. **Volba algoritmu je důležitá!**

Klíčové poznatky

4. **Běžné složitosti** (od nejlepší k nejhorší):

   $$\mathcal{O}(1) < \mathcal{O}(\log n) < \mathcal{O}(n) < \mathcal{O}(n \log n) < \mathcal{O}(n^2) < \mathcal{O}(2^n) < \mathcal{O}(n!)$$

5. **Volba algoritmu je důležitá!**

Rozdíl mezi $\mathcal{O}(n)$ a $\mathcal{O}(n^2)$ může být rozdíl mezi sekundami a dny