Složitost

BI-PA1 – Programování a Algoritmizace I.

Ladislav Vagner

Katedra teoretické informatiky
Fakulta informačních technologií
ČVUT v Praze
xvagner@fit.cvut.cz

27. listopadu 2013

Obsah

- Základy (součty posloupností, řád růstu funkce).
- Složitost jednoduchých cyklů.
- Složitější příklady (vnořené cykly).
- Příklady algoritmů a jejich složitosti.

Základy - součet aritmetické posloupnosti

- Aritmetická posloupnost:

$$a_i = a_{i-1} + d$$

- Součet aritmetické posloupnosti:

$$s_n = a_0 + a_1 + \cdots + a_n$$

- Pro n liché je počet členů posloupnosti sudý:

$$s_n = (a_0+a_n)+(a_1+a_{n-1})+\cdots+(a_{\lfloor \frac{n}{2} \rfloor}+a_{\lceil \frac{n}{2} \rceil}) = (a_0+a_n)\frac{n+1}{2}$$

- Pro n sudé je počet členů posloupnosti lichý:

$$s_n = (a_0 + a_n) + (a_1 + a_{n-1}) + \cdots + (a_{\frac{n}{2}}) = (a_0 + a_n)\frac{n+1}{2}$$

Základy - součet geometrické posloupnosti

- Geometrická posloupnost:

$$a_i = q a_{i-1}$$

- Součet geometrické posloupnosti:

$$s_n = a_0 + a_1 + \cdots + a_n =$$

$$a_0 + a_0 q + a_0 q^2 + \cdots + a_0 q^n$$

- Trik pro součet:

$$s_n(q-1) = q s_n - s_n = a_0 q + a_0 q^2 + \cdots + a_0 q^{n+1} -$$

$$a_0 - a_0 q - a_0 q^2 - \ldots - a_0 q^n = a_0 q^{n+1} - a_0$$

$$s_n = a_0 \frac{q^{n+1} - 1}{q - 1}$$

Řád růstu funkce

- Při hodnocení efektivity algoritmu nás zajímá, kolik operací se při výpočtu musí provést.
- Nezajímá nás konkrétní číslo, důležitější je nalézt časovou složitost algoritmu (= závislost mezi velikostí vstupních dat a počtem provedených operací):

$$T(n) = f(n)$$

- Přesné určení funkce $T(n)$ bývá téměř nemožné. Spokojíme se s nalezením řádu růstu funkce $T(n)$, tedy s funkcí, která $T(n)$ pro velká n aproximuje. Zapisujeme např.:

$$T(n) = \mathcal{O}(n^3)$$

obecně

$$T(n) = \mathcal{O}(f(n))$$

Řád růstu funkce

Co znamenají zápisy $\mathcal{O}$, Ω a Θ:

- $T(n) = \mathcal{O}(f(n))$:

$$\exists_{c \in R^+, n_0 \in N} \forall_{n \in N, n > n_0} : T(n) \leq cf(n)$$

- $T(n) = \Omega(g(n))$:

$$\exists_{c \in R^+, n_0 \in N} \forall_{n \in N, n > n_0} : T(n) \geq cg(n)$$

- $T(n) = \Theta(h(n))$:

$$\exists_{c_1, c_2 \in R^+, n_0 \in N} \forall_{n \in N, n > n_0} : c_1 h(n) \leq T(n) \leq c_2 h(n)$$

Řád růstu funkce

Předpokládejme, že pro náš algoritmus je $T(n) = 5n^2 + 6n + 1015$.
Je $T(n) = \mathcal{O}(n^2)$?

Řád růstu funkce

Předpokládejme, že pro náš algoritmus je $T(n) = 5n^2 + 6n + 1015$.
Je $T(n) = \mathcal{O}(n^2)$?
Zvolme $c = 6$. Pak musí platit: $6n^2 \geq 5n^2 + 6n + 1015$ pro všechna n od nějaké meze n_0.

$$n^2 - 6n - 1015 \geq 0$$

$$n \in (-\infty; -29\rangle \cup \langle 35; \infty)$$

Závěr: platí, že $T(n) = 5n^2 + 6n + 1015 = \mathcal{O}(n^2)$ (z definice, pro $c = 6$ a $n_0 = 35$).

Řád růstu funkce

Předpokládejme, že pro náš algoritmus je $T(n) = 5n^2 + 6n + 1015$.
Je $T(n) = \mathcal{O}(n^2)$?

Řád růstu funkce

Předpokládejme, že pro náš algoritmus je $T(n) = 5n^2 + 6n + 1015$.
Je $T(n) = \mathcal{O}(n^2)$?
Zvolme $n_0 = 10$. Pak musí platit: $cn^2 \geq 5n^2 + 6n + 1015$ pro nějaké kladné c a všechna $n \geq 10$. Řešíme tedy kvadratickou nerovnici s parametrem c:

$$(c - 5)n^2 - 6n - 1015 \geq 0$$

Pro hodnoty parametru $c \leq 5$ bude řešením pouze interval čísel n, tedy takové řešení nás nezajímá. Pro hodnotu $n_0 = 10$ lze najít parametr $c = 15.75$, aby nerovnice platila. Protože $15.75 > 5$, bude v tomto případě $c = 15.75$ hledaným parametrem.
Závěr: platí, že $T(n) = 5n^2 + 6n + 1015 = \mathcal{O}(n^2)$ (z definice, pro $c = 15.75$ a $n_0 = 10$).

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	$= \mathcal{O}(n^4)$
$12(n+2)^3 - 3n + 100$	

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	$= \mathcal{O}(n^4)$
$12(n+2)^3 - 3n + 100$	$= \mathcal{O}(n^3)$
$\log_2(3n^4 + 6n)$	

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	$= \mathcal{O}(n^4)$
$12(n+2)^3 - 3n + 100$	$= \mathcal{O}(n^3)$
$\log_2(3n^4 + 6n)$	$= \mathcal{O}(\log n)$
$0.00001n + 100000 \log n$	

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	$= \mathcal{O}(n^4)$
$12(n+2)^3 - 3n + 100$	$= \mathcal{O}(n^3)$
$\log_2(3n^4 + 6n)$	$= \mathcal{O}(\log n)$
$0.00001n + 100000 \log n$	$= \mathcal{O}(n)$
$10 \log n + 0.1 \log^2 n$	

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	$= \mathcal{O}(n^4)$
$12(n+2)^3 - 3n + 100$	$= \mathcal{O}(n^3)$
$\log_2(3n^4 + 6n)$	$= \mathcal{O}(\log n)$
$0.00001n + 100000 \log n$	$= \mathcal{O}(n)$
$10 \log n + 0.1 \log^2 n$	$= \mathcal{O}(\log^2 n)$
$\sqrt{n} + \log^{20} n$	

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	$= \mathcal{O}(n^4)$
$12(n+2)^3 - 3n + 100$	$= \mathcal{O}(n^3)$
$\log_2(3n^4 + 6n)$	$= \mathcal{O}(\log n)$
$0.00001n + 100000 \log n$	$= \mathcal{O}(n)$
$10 \log n + 0.1 \log^2 n$	$= \mathcal{O}(\log^2 n)$
$\sqrt{n} + \log^{20} n$	$= \mathcal{O}(\sqrt{n})$
$\log \log n + \log n$	

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	$= \mathcal{O}(n^4)$
$12(n+2)^3 - 3n + 100$	$= \mathcal{O}(n^3)$
$\log_2(3n^4 + 6n)$	$= \mathcal{O}(\log n)$
$0.00001n + 100000 \log n$	$= \mathcal{O}(n)$
$10 \log n + 0.1 \log^2 n$	$= \mathcal{O}(\log^2 n)$
$\sqrt{n} + \log^{20} n$	$= \mathcal{O}(\sqrt{n})$
$\log \log n + \log n$	$= \mathcal{O}(\log n)$
$n + n \log n$	

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	$= \mathcal{O}(n^4)$
$12(n+2)^3 - 3n + 100$	$= \mathcal{O}(n^3)$
$\log_2(3n^4 + 6n)$	$= \mathcal{O}(\log n)$
$0.00001n + 100000 \log n$	$= \mathcal{O}(n)$
$10 \log n + 0.1 \log^2 n$	$= \mathcal{O}(\log^2 n)$
$\sqrt{n} + \log^{20} n$	$= \mathcal{O}(\sqrt{n})$
$\log \log n + \log n$	$= \mathcal{O}(\log n)$
$n + n \log n$	$= \mathcal{O}(n \log n)$
$n + n \log \log n$	

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	$= \mathcal{O}(n^4)$
$12(n+2)^3 - 3n + 100$	$= \mathcal{O}(n^3)$
$\log_2(3n^4 + 6n)$	$= \mathcal{O}(\log n)$
$0.00001n + 100000 \log n$	$= \mathcal{O}(n)$
$10 \log n + 0.1 \log^2 n$	$= \mathcal{O}(\log^2 n)$
$\sqrt{n} + \log^{20} n$	$= \mathcal{O}(\sqrt{n})$
$\log \log n + \log n$	$= \mathcal{O}(\log n)$
$n + n \log n$	$= \mathcal{O}(n \log n)$
$n + n \log \log n$	$= \mathcal{O}(n \log \log n)$
$n^{1.00001} + n \log n$	

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	$= \mathcal{O}(n^4)$
$12(n+2)^3 - 3n + 100$	$= \mathcal{O}(n^3)$
$\log_2(3n^4 + 6n)$	$= \mathcal{O}(\log n)$
$0.00001n + 100000 \log n$	$= \mathcal{O}(n)$
$10 \log n + 0.1 \log^2 n$	$= \mathcal{O}(\log^2 n)$
$\sqrt{n} + \log^{20} n$	$= \mathcal{O}(\sqrt{n})$
$\log \log n + \log n$	$= \mathcal{O}(\log n)$
$n + n \log n$	$= \mathcal{O}(n \log n)$
$n + n \log \log n$	$= \mathcal{O}(n \log \log n)$
$n^{1.00001} + n \log n$	$= \mathcal{O}(n^{1.00001})$
$n^{100} + 0.0001 \cdot 2^n$	

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	$= \mathcal{O}(n^4)$
$12(n+2)^3 - 3n + 100$	$= \mathcal{O}(n^3)$
$\log_2(3n^4 + 6n)$	$= \mathcal{O}(\log n)$
$0.00001n + 100000 \log n$	$= \mathcal{O}(n)$
$10 \log n + 0.1 \log^2 n$	$= \mathcal{O}(\log^2 n)$
$\sqrt{n} + \log^{20} n$	$= \mathcal{O}(\sqrt{n})$
$\log \log n + \log n$	$= \mathcal{O}(\log n)$
$n + n \log n$	$= \mathcal{O}(n \log n)$
$n + n \log \log n$	$= \mathcal{O}(n \log \log n)$
$n^{1.00001} + n \log n$	$= \mathcal{O}(n^{1.00001})$
$n^{100} + 0.0001 \cdot 2^n$	$= \mathcal{O}(2^n)$
$3^n + 2^n$	

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	$= \mathcal{O}(n^4)$
$12(n+2)^3 - 3n + 100$	$= \mathcal{O}(n^3)$
$\log_2(3n^4 + 6n)$	$= \mathcal{O}(\log n)$
$0.00001n + 100000 \log n$	$= \mathcal{O}(n)$
$10 \log n + 0.1 \log^2 n$	$= \mathcal{O}(\log^2 n)$
$\sqrt{n} + \log^{20} n$	$= \mathcal{O}(\sqrt{n})$
$\log \log n + \log n$	$= \mathcal{O}(\log n)$
$n + n \log n$	$= \mathcal{O}(n \log n)$
$n + n \log \log n$	$= \mathcal{O}(n \log \log n)$
$n^{1.00001} + n \log n$	$= \mathcal{O}(n^{1.00001})$
$n^{100} + 0.0001 \cdot 2^n$	$= \mathcal{O}(2^n)$
$3^n + 2^n$	$= \mathcal{O}(3^n)$
$3^n + 2^{2n}$	

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	$= \mathcal{O}(n^4)$
$12(n+2)^3 - 3n + 100$	$= \mathcal{O}(n^3)$
$\log_2(3n^4 + 6n)$	$= \mathcal{O}(\log n)$
$0.00001n + 100000 \log n$	$= \mathcal{O}(n)$
$10 \log n + 0.1 \log^2 n$	$= \mathcal{O}(\log^2 n)$
$\sqrt{n} + \log^{20} n$	$= \mathcal{O}(\sqrt{n})$
$\log \log n + \log n$	$= \mathcal{O}(\log n)$
$n + n \log n$	$= \mathcal{O}(n \log n)$
$n + n \log \log n$	$= \mathcal{O}(n \log \log n)$
$n^{1.00001} + n \log n$	$= \mathcal{O}(n^{1.00001})$
$n^{100} + 0.0001 \cdot 2^n$	$= \mathcal{O}(2^n)$
$3^n + 2^n$	$= \mathcal{O}(3^n)$
$3^n + 2^{2n}$	$= \mathcal{O}(2^{2n}) = \mathcal{O}(4^n)$
$3^{-n} + 10$	

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	$= \mathcal{O}(n^4)$
$12(n+2)^3 - 3n + 100$	$= \mathcal{O}(n^3)$
$\log_2(3n^4 + 6n)$	$= \mathcal{O}(\log n)$
$0.00001n + 100000 \log n$	$= \mathcal{O}(n)$
$10 \log n + 0.1 \log^2 n$	$= \mathcal{O}(\log^2 n)$
$\sqrt{n} + \log^{20} n$	$= \mathcal{O}(\sqrt{n})$
$\log \log n + \log n$	$= \mathcal{O}(\log n)$
$n + n \log n$	$= \mathcal{O}(n \log n)$
$n + n \log \log n$	$= \mathcal{O}(n \log \log n)$
$n^{1.00001} + n \log n$	$= \mathcal{O}(n^{1.00001})$
$n^{100} + 0.0001 \cdot 2^n$	$= \mathcal{O}(2^n)$
$3^n + 2^n$	$= \mathcal{O}(3^n)$
$3^n + 2^{2n}$	$= \mathcal{O}(2^{2n}) = \mathcal{O}(4^n)$
$3^{-n} + 10$	$= \mathcal{O}(1)$
$n^2 + n!$	

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	$= \mathcal{O}(n^4)$
$12(n+2)^3 - 3n + 100$	$= \mathcal{O}(n^3)$
$\log_2(3n^4 + 6n)$	$= \mathcal{O}(\log n)$
$0.00001n + 100000 \log n$	$= \mathcal{O}(n)$
$10 \log n + 0.1 \log^2 n$	$= \mathcal{O}(\log^2 n)$
$\sqrt{n} + \log^{20} n$	$= \mathcal{O}(\sqrt{n})$
$\log \log n + \log n$	$= \mathcal{O}(\log n)$
$n + n \log n$	$= \mathcal{O}(n \log n)$
$n + n \log \log n$	$= \mathcal{O}(n \log \log n)$
$n^{1.00001} + n \log n$	$= \mathcal{O}(n^{1.00001})$
$n^{100} + 0.0001 \cdot 2^n$	$= \mathcal{O}(2^n)$
$3^n + 2^n$	$= \mathcal{O}(3^n)$
$3^n + 2^{2n}$	$= \mathcal{O}(2^{2n}) = \mathcal{O}(4^n)$
$3^{-n} + 10$	$= \mathcal{O}(1)$
$n^2 + n!$	$= \mathcal{O}(n!)$
$2^n + n!$	

Řád růstu funkce

$T(n)$	$\mathcal{O}(f(n))$
$12n^4 - 5n^3 + 4n - 8$	$= \mathcal{O}(n^4)$
$12(n+2)^3 - 3n + 100$	$= \mathcal{O}(n^3)$
$\log_2(3n^4 + 6n)$	$= \mathcal{O}(\log n)$
$0.00001n + 100000 \log n$	$= \mathcal{O}(n)$
$10 \log n + 0.1 \log^2 n$	$= \mathcal{O}(\log^2 n)$
$\sqrt{n} + \log^{20} n$	$= \mathcal{O}(\sqrt{n})$
$\log \log n + \log n$	$= \mathcal{O}(\log n)$
$n + n \log n$	$= \mathcal{O}(n \log n)$
$n + n \log \log n$	$= \mathcal{O}(n \log \log n)$
$n^{1.00001} + n \log n$	$= \mathcal{O}(n^{1.00001})$
$n^{100} + 0.0001 \cdot 2^n$	$= \mathcal{O}(2^n)$
$3^n + 2^n$	$= \mathcal{O}(3^n)$
$3^n + 2^{2n}$	$= \mathcal{O}(2^{2n}) = \mathcal{O}(4^n)$
$3^{-n} + 10$	$= \mathcal{O}(1)$
$n^2 + n!$	$= \mathcal{O}(n!)$
$2^n + n!$	$= \mathcal{O}(n!)$

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

$$n = \mathcal{O}(n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

$$n = \mathcal{O}(n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

$$T(n) = n = \mathcal{O}(n)$$

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 2 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 2 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\approx \frac{n}{2} = \mathcal{O}(n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 2 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\approx \frac{n}{2} = \mathcal{O}(n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

$$T(n) = n = \mathcal{O}(n)$$

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 5 == 1 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 5 == 1 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\approx \frac{n}{5} = \mathcal{O}(n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 5 == 1 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\approx \frac{n}{5} = \mathcal{O}(n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

$$T(n) = n = \mathcal{O}(n)$$

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 5 == 1 && i % 3 == 1 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 5 == 1 && i % 3 == 1 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\frac{n+13}{15} \approx \frac{n}{15} = \mathcal{O}(n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 5 == 1 && i % 3 == 1 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\frac{n+13}{15} \approx \frac{n}{15} = \mathcal{O}(n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

$$T(n) = n = \mathcal{O}(n)$$

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 5 == 1 || i % 3 == 1 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 5 == 1 || i % 3 == 1 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\approx \frac{7n}{15} = \mathcal{O}(n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 5 == 1 || i % 3 == 1 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\approx \frac{7n}{15} = \mathcal{O}(n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

$$T(n) = n = \mathcal{O}(n)$$

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 5 != 1 && i % 3 != 1 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 5 != 1 && i % 3 != 1 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\approx \frac{8n}{15} = \mathcal{O}(n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 5 != 1 && i % 3 != 1 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\approx \frac{8n}{15} = \mathcal{O}(n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

$$T(n) = n = \mathcal{O}(n)$$

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 5 == i % 3 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 5 == i % 3 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\approx \frac{3n}{15} = \mathcal{O}(n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

Časová složitost – jednoduché cykly

```
for ( i = 0; i < n; i ++ )
 if ( i % 5 == i % 3 )
  doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\approx \frac{3n}{15} = \mathcal{O}(n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

$$T(n) = n = \mathcal{O}(n)$$

Časová složitost – jednoduché cykly

```
for ( i = 1; i < n; i *= 2 )
 doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – jednoduché cykly

```
for ( i = 1; i < n; i *= 2 )
 doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\log_2 n = \mathcal{O}(\log n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

Časová složitost – jednoduché cykly

```
for ( i = 1; i < n; i *= 2 )
 doSomething ();
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\log_2 n = \mathcal{O}(\log n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

$$T(n) = \log_2 n = \mathcal{O}(\log n)$$

Časová složitost – jednoduché cykly

```
i = n;
while ( i ) {
 doSomething ();
 i /= 2;
}
```

- Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – jednoduché cykly

```
i = n;
while ( i ) {
 doSomething ();
 i /= 2;
}
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\log_2 n = \mathcal{O}(\log n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

Časová složitost – jednoduché cykly

```
i = n;
while ( i ) {
 doSomething ();
 i /= 2;
}
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\log_2 n = \mathcal{O}(\log n)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

$$T(n) = \log_2 n = \mathcal{O}(\log n)$$

Časová složitost – jednoduché cykly

```
j = 1;
for ( i = 0; i < n; i += j ) {
 doSomething ();
 j += 2;
}
```

- Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – jednoduché cykly

```
j = 1;
for ( i = 0; i < n; i += j ) {
 doSomething ();
 j += 2;
}
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\sqrt{n} = \mathcal{O}(\sqrt{n})$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

Časová složitost – jednoduché cykly

```
j = 1;
for ( i = 0; i < n; i += j ) {
 doSomething ();
 j += 2;
}
```

- Kolikrát se zavolá funkce `doSomething()`?

$$\sqrt{n} = \mathcal{O}(\sqrt{n})$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

$$T(n) = \sqrt{n} = \mathcal{O}(\sqrt{n})$$

Časová složitost – jednoduché cykly

```
for ( i = j = 0; i < n; j ++ ) {
 doSomething ();
 i += j / n;
 j %= n;
}
```

- Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – jednoduché cykly

```
for ( i = j = 0; i < n; j ++ ) {
 doSomething ();
 i += j / n;
 j %= n;
}
```

- Kolikrát se zavolá funkce `doSomething()`?

$$n^2 = \mathcal{O}(n^2)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

Časová složitost – jednoduché cykly

```
for ( i = j = 0; i < n; j ++ ) {
 doSomething ();
 i += j / n;
 j %= n;
}
```

- Kolikrát se zavolá funkce `doSomething()`?

$$n^2 = \mathcal{O}(n^2)$$

- Jaká je časová složitost tohoto algoritmu (předpokládáme, že samotná funkce `doSomething()` má složitost $\mathcal{O}(1)$)?

$$T(n) = n^2 = \mathcal{O}(n^2)$$

Časová složitost – vnořené cykly

```
for (i = 0; i < n; i++) {
  j = n;
  while ( j > 0 ) {
    doSomething ();
    j = j / 2;
  }
}
```

Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – vnořené cykly

```
for (i = 0; i < n; i++) {
  j = n;
  while ( j > 0 ) {
    doSomething ();
    j = j / 2;
  }
}
```

Kolikrát se zavolá funkce `doSomething()`?

$$n \log_2 n = \mathcal{O}(n \log n)$$

Časová složitost – vnořené cykly

```
for (i = 0; i < n; i++) {
  j = 1;
  while ( j < i ) {
    doSomething ();
    j = j * 2;
  }
}
```

Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – vnořené cykly

```
for (i = 0; i < n; i++) {
  j = 1;
  while ( j < i ) {
    doSomething ();
    j = j * 2;
  }
}
```

Kolikrát se zavolá funkce `doSomething()`?

$$\approx n \log_2 n = \mathcal{O}(n \log n)$$

Časová složitost – vnořené cykly

```
for (i = 0; i < 2*n; i++) {
  if ( i < n )
    for (j = i; j < 2 * n; j ++ ) doSomething ();
  else
    for (j = 0; j < i; j ++ ) doSomething ();
}
```

Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – vnořené cykly

```
for (i = 0; i < 2*n; i++) {
  if ( i < n )
    for (j = i; j < 2 * n; j ++ ) doSomething ();
  else
    for (j = 0; j < i; j ++ ) doSomething ();
}
```

Kolikrát se zavolá funkce `doSomething()`?

$$3n^2 = \mathcal{O}(n^2)$$

Časová složitost – vnořené cykly

```
for (i = 0; i < 2*n; i++) {
  if ( i < n )
    for (j = 0; j < 2 * i; j ++ ) doSomething ();
  else
    for (j = 0; j < 2 * n; j ++ ) doSomething ();
}
```

Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – vnořené cykly

```
for (i = 0; i < 2*n; i++) {
  if ( i < n )
    for (j = 0; j < 2 * i; j ++ ) doSomething ();
  else
    for (j = 0; j < 2 * n; j ++ ) doSomething ();
}
```

Kolikrát se zavolá funkce `doSomething()`?

$$n^2 - n + 2n^2 = 3n^2 - n = \mathcal{O}(n^2)$$

Časová složitost – vnořené cykly

```
for (i = 0; i < 2*n; i++) {
  if ( i > n )
    for (j = i; j < 2 * i; j ++ ) doSomething();
  else
    for (j = n; j < 2 * n; j ++ ) doSomething();
}
```

Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – vnořené cykly

```
for (i = 0; i < 2*n; i++) {
  if ( i > n )
    for (j = i; j < 2 * i; j ++ ) doSomething();
  else
    for (j = n; j < 2 * n; j ++ ) doSomething();
}
```

Kolikrát se zavolá funkce `doSomething()`?

$$\frac{3n^2 - 3n}{2} + n^2 + n = \mathcal{O}(n^2)$$

Časová složitost – vnořené cykly

```
for (i = 0; i < 2*n; i += 2) {
  if ( i > n )
    for (j = i; j < 2 * i; j ++ ) doSomething();
  else
    for (j = n; j < 2 * n; j ++ ) doSomething();
}
```

Kolikrát se zavolá funkce `doSomething()`?

Časová složitost – vnořené cykly

```
for (i = 0; i < 2*n; i += 2) {
  if ( i > n )
    for (j = i; j < 2 * i; j ++ ) doSomething();
  else
    for (j = n; j < 2 * n; j ++ ) doSomething();
}
```

Kolikrát se zavolá funkce `doSomething()`?

$$\frac{3n^2 - 3n}{4} + \frac{n^2 + n}{2} = \mathcal{O}(n^2)$$

Příklady algoritmů a jejich složitostí

Ve městě je celkem n křižovatek. Křižovatky jsou spojené ulicemi, dvojice křižovatek je spojena nejvýše jednou ulicí. Ulice mohou být jednosměrné. Pokud lze projet z křižovatky i do křižovatky j, pak v matici křižovatek bude `cross[i][j] = 1`. Na vstupu máte zadanou vyplněnou matici křižovatek `cross`. Nalezněte algoritmus pro určení počtu jednosměrných ulic. Určete časovou složitost tohoto algoritmu.

Příklady algoritmů a jejich složitostí

Ve městě je celkem n křižovatek. Křižovatky jsou spojené ulicemi, dvojice křižovatek je spojena nejvýše jednou ulicí. Ulice mohou být jednosměrné. Pokud lze projet z křižovatky i do křižovatky j, pak v matici křižovatek bude `cross[i][j] = 1`. Na vstupu máte zadanou vyplněnou matici křižovatek `cross`. Nalezněte algoritmus pro určení počtu jednosměrných ulic. Určete časovou složitost tohoto algoritmu.

```
jednosmerek = 0
pro každou křižovatku I
 pro každou křižovatku J
  pokud I != J a cross[I][J] == 1 a cross[J][I] == 0
   jednosmerek ++;
```

Příklady algoritmů a jejich složitostí

Ve městě je celkem n křižovatek. Křižovatky jsou spojené ulicemi, dvojice křižovatek je spojena nejvýše jednou ulicí. Ulice mohou být jednosměrné. Pokud lze projet z křižovatky i do křižovatky j, pak v matici křižovatek bude `cross[i][j] = 1`. Na vstupu máte zadanou vyplněnou matici křižovatek `cross`. Nalezněte algoritmus pro určení počtu jednosměrných ulic. Určete časovou složitost tohoto algoritmu.

```
jednosmerek = 0
pro každou křižovatku I
 pro každou křižovatku J
  pokud I != J a cross[I][J] == 1 a cross[J][I] == 0
   jednosmerek ++;
```

Časová složitost: $\mathcal{O}(n^2)$

Příklady algoritmů a jejich složitostí

Ve městě je celkem n křižovatek. Křižovatky jsou spojené ulicemi, dvojice křižovatek je spojena nejvýše jednou ulicí. Ulice mohou být jednosměrné. Pokud lze projet z křižovatky i do křižovatky j, pak v matici křižovatek bude `cross[i][j] = 1`. Na vstupu máte zadanou vyplněnou matici křižovatek `cross`. Nalezněte algoritmus pro určení počtu jednosměrných ulic. Určete časovou složitost tohoto algoritmu.

```
jednosmerek = 0
pro každou křižovatku I
 pro každou křižovatku J
  pokud I != J a cross[I][J] == 1 a cross[J][I] == 0
   jednosmerek ++;
```

Časová složitost: $\mathcal{O}(n^2)$
Rychlejší algoritmus nelze najít, protože jednosměrek může být až $n^2 - n = \mathcal{O}(n^2)$

Příklady algoritmů a jejich složitostí

Předpokládáme konečnou stanici autobusu. Ráno před zahájením provozu přijede na stanici z garáží x autobusů. Podle jízdního řádu z konečné autobusy odjíždějí spoje a zároveň další autobusy přijíždějí (z opačného směru). Předpokládáme, že žádný další autobus z garáží nepřijede (musí být přistaven již ráno). Pokud znáte časy příjezdu a odjezdu autobusů, určete, kolik autobusů musí být ráno přistaveno, aby bylo možné dodržet jízdní řád. Předpokládáme, že jízdní řády mají podobu neseřazených polí.

Příklady algoritmů a jejich složitostí

Předpokládáme konečnou stanici autobusu. Ráno před zahájením provozu přijede na stanici z garáží x autobusů. Podle jízdního řádu z konečné autobusy odjíždějí spoje a zároveň další autobusy přijíždějí (z opačného směru). Předpokládáme, že žádný další autobus z garáží nepřijede (musí být přistaven již ráno). Pokud znáte časy příjezdu a odjezdu autobusů, určete, kolik autobusů musí být ráno přistaveno, aby bylo možné dodržet jízdní řád. Předpokládáme, že jízdní řády mají podobu neseřazených polí.

```
1. pro každý spoj A z jízdního řádu příjezdů
2.  najdi spoj B v jízdním řádu odjezdů, jehož čas
     odjezdu je nejblíže vyšší
3.  pokud spoj B existuje, odstraň jej ze seznamu odjezdů
4.  pokud spoj B neexistuje, nedělej nic
5. spočti počet zbývajících odjezdů, to je potřebný počet
  ráno přistavených autobusů
```

Příklady algoritmů a jejich složitostí

```
1. pro každý spoj A z jízdního řádu příjezdů
2.  najdi spoj B v jízdním řádu odjezdů, jehož čas
     odjezdu je nejblíže vyšší
3.  pokud spoj B existuje, odstraň jej ze seznamu odjezdů
4.  pokud spoj B neexistuje, nedělej nic
5. spočti počet zbývajících odjezdů, to je potřebný počet
  ráno přistavených autobusů
```

Příklady algoritmů a jejich složitostí

```
1. pro každý spoj A z jízdního řádu příjezdů
2.  najdi spoj B v jízdním řádu odjezdů, jehož čas
     odjezdu je nejblíže vyšší
3.  pokud spoj B existuje, odstraň jej ze seznamu odjezdů
4.  pokud spoj B neexistuje, nedělej nic
5. spočti počet zbývajících odjezdů, to je potřebný počet
  ráno přistavených autobusů
```

Časová složitost:

- nalezení spoje (bod 2): $\mathcal{O}(n)$,
- odstranění z pole (bod 3): $\mathcal{O}(1)$,
- body 2,3,4 se opakují celkem $\mathcal{O}(m)$ krát,
- bod 5 trvá nejvýše $\mathcal{O}(n)$.

Celkem:

$$T = \mathcal{O}(m) * (\mathcal{O}(n) + \mathcal{O}(1)) + \mathcal{O}(n) = \mathcal{O}(nm)$$

Příklady algoritmů a jejich složitostí

Lepší řešení:

1. vytvoř pole velikosti počet odjezdů + počet příjezdů
2. zkopíruj do pole hodnoty z jízdních řádů, ke každému prvku pole přidej příznak příjezd nebo odjezd
3. seřaď pole podle času vzestupně,
4. nastav akumulátor na hodnotu 0
5. projdi seřazené pole
6. za každý příjezd přičti k akumulátoru 1, za odjezd -1
7. nejvyšší hodnota akumulátoru během výpočtu udává potřebný počet ráno přistavených autobusů

Příklady algoritmů a jejich složitostí

Lepší řešení:

```
1. vytvoř pole velikosti počet odjezdů + počet příjezdů
2. zkopíruj do pole hodnoty z jízdních řádů, ke každému
    prvku pole přidej příznak příjezd nebo odjezd
3. seřaď pole podle času vzestupně,
4. nastav akumulátor na hodnotu 0
5. projdi seřazené pole
6.  za každý příjezd přičti k akumulátoru 1, za odjezd -1
7. nejvyšší hodnota akumulátoru během výpočtu udává
    potřebný počet ráno přistavených autobusů
```

Časová složitost:

- příprava a kopírování pole (body 1 a 2): $\mathcal{O}(n+m)$,
- řazení pole (bod 3): $\mathcal{O}((n+m)\log(n+m))$,
- procházení pole, hledání maxima (4,5,6 a 7): $\mathcal{O}(n+m)$.

Celkem:

$$T = \mathcal{O}(n+m) + \mathcal{O}((n+m)\log(n+m)) = \mathcal{O}((n+m)\log(n+m))$$

Příklady algoritmů a jejich složitostí

Pokud by již na vstupu byly jízdní řády seřazené:

Příklady algoritmů a jejich složitostí

Pokud by již na vstupu byly jízdní řády seřazené:

1. vytvoř pole velikosti počet odjezdů + počet příjezdů
2. sluč do pole hodnoty z jízdních řádů, ke každému prvku pole přidej příznak příjezd nebo odjezd, ke sloučení použij proceduru merge, podobně jako v merge sortu
3. nastav akumulátor na hodnotu 0
4. projdi seřazené pole
5. za každý příjezd přičti k akumulátoru 1, za odjezd -1
6. nejvyšší hodnota akumulátoru během výpočtu udává potřebný počet ráno přistavených autobusů

Příklady algoritmů a jejich složitostí

Pokud by již na vstupu byly jízdní řády seřazené:

```
1. vytvoř pole velikosti počet odjezdů + počet příjezdů
2. sluč do pole hodnoty z jízdních řádů, ke každému prvku
    pole přidej příznak příjezd nebo odjezd, ke sloučení
    použij proceduru merge, podobně jako v merge sortu
3. nastav akumulátor na hodnotu 0
4. projdi seřazené pole
5.  za každý příjezd přičti k akumulátoru 1, za odjezd -1
6. nejvyšší hodnota akumulátoru během výpočtu udává
    potřebný počet ráno přistavených autobusů
```

Časová složitost:

- slučování pole (body 1 a 2): $\mathcal{O}(n + m)$,
- procházení pole, hledání maxima (3,4,5 a 6): $\mathcal{O}(n + m)$.

Celkem:

$$T = \mathcal{O}(n + m) + \mathcal{O}(n + m) = \mathcal{O}(n + m)$$

Dotazy ...

Děkuji za pozornost.