# Algoritmická složitost

## KIV/ADT – 2. přednáška

Miloslav Konopík

1. března 2024

# Složitost algoritmů

1. Motivace
2. Algoritmická složitost
3. Řád růstu funkce.
4. Praktické příklady na složitost.

# Motivace

# Rychlost algoritmů

Proč měřit rychlost algoritmů:

- Jak velká data jsme schopni zpracovat?
- Který algoritmus je lepší?

Jak měřit rychlost algoritmů:

- Změřit dobu běhu (benchmark) – měříme algoritmus, programovací jazyk, hardware, ... .
- Najít formální metriku závislou pouze na algoritmu.

**Příklad: Problém vyhledávání prvků v seřazeném poli.**

- Sekvenční vyhledávání.
- Binární vyhledávání (půlení intervalů).

# Příklad – sekvenční vyhledávání

```
1  def sequential_search(arr: List[int], target: int) -> int:
2      for i in range(len(arr)):
3          if arr[i] == target:
4              return i
5      return None
```

# Příklad – binární vyhledávání

```
1  def binary_search(arr: List[int], target: int) -> int:
2      left: int = 0
3      right: int = len(arr) - 1

4      while left <= right:
5          mid: int = (left + right) // 2
6          if arr[mid] == target:
7              return mid
8          elif arr[mid] < target:
9              left = mid + 1
10         else:
11             right = mid - 1

12     return None
```

# Porovnání rychlostí

| Počet dat | Lineární vyhledávání nejlepší | Lineární vyhledávání průměrný | Lineární vyhledávání nejhorší | Binární vyhledávání | Poměr průměr VS nejhorší |
|---|---|---|---|---|---|
| 5 | 1 | 3 | 5 | 3 | 1x |
| 10 | 1 | 5 | 10 | 4 | 1x |
| 500 | 1 | 250 | 500 | 9 | 28x |
| 1000 | 1 | 500 | 1000 | 10 | 50x |
| 5000 | 1 | 2500 | 5000 | 13 | 192x |
| 10000 | 1 | 5000 | 10000 | 14 | 3 57x |
| 50000 | 1 | 25000 | 50000 | 16 | 1 563x |
| 100000 | 1 | 50000 | 100000 | 17 | 2 941x |
| 500000 | 1 | 250000 | 500000 | 19 | 13 158x |

# Algoritmická složitost

# Algoritmická složitost

- Vykonání algoritmů je různě náročné na zdroje počítače.
- Různé algoritmy, které řeší stejný problém, mohou mít různé nároky (různou složitost).
- Existují dvě přirozené míry pro porovnání:
  - Časová složitost: Doba výpočtu podle daného algoritmu potřebná pro zpracování daného množství dat.
  - Paměťová složitost: Maximum alokované paměti v průběhu výpočtu.
- Pro určení složitosti používáme abstraktní model počítače.
  - Časová složitost: Počítáme nejčastější operace, např. počet porovnání.
  - Paměťová složitost: Sčítáme alokace paměti, kdy základním datovým typům přiřadíme velikost 1.
- V praxi většinou důležité pro „velká“ data.

# Algoritmická složitost

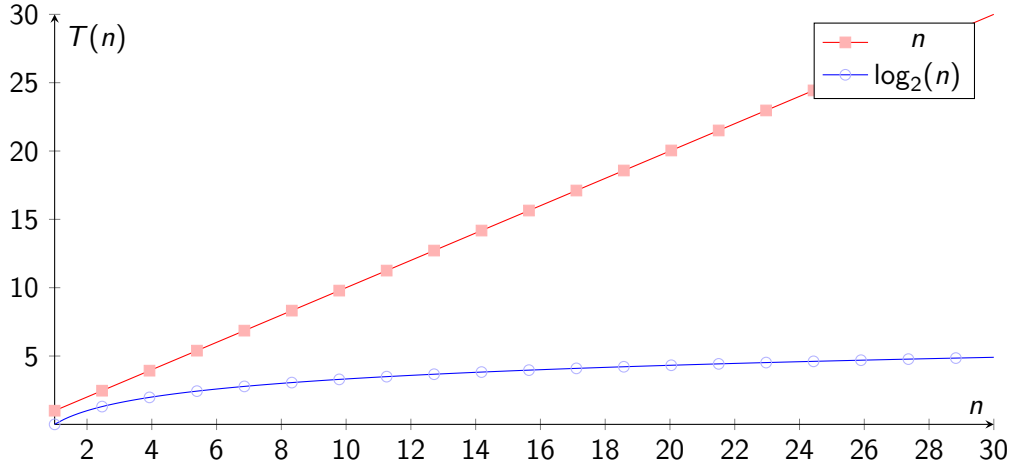
- Vykonání algoritmů je různě náročné na zdroje počítače.
- Různé algoritmy, které řeší stejný problém, mohou mít různé nároky (různou složitost).
- Existují dvě přirozené míry pro porovnání:
  - **Časová složitost: Doba výpočtu podle daného algoritmu potřebná pro zpracování daného množství dat.**
  - Paměťová složitost: Maximum alokované paměti v průběhu výpočtu.
- Pro určení složitosti používáme abstraktní model počítače.
  - **Časová složitost: Počítáme nejčastější operace, např. počet porovnání.**
  - Paměťová složitost: Sčítáme alokace paměti, kdy základním datovým typům přiřadíme velikost 1.
- V praxi většinou důležité pro „velká“ data.

# Funkce T

- Určujeme funkci $T(n) : \mathbb{N} \to \mathbb{R}_0^+$.
- $n$ je velikost problému.
  - Délka (počet) vstupních dat
  - $n$ může být vícerozměrné (více parametrů, například šířka, výška obrázku).
- Zajímá nás rychlost růstu funkce $T(n)$ v závislosti na růstu parametru $n$.
- Porovnáním funkcí $T_1(n)$ a $T_2(n)$ pro dva různé algoritmy můžeme najít vhodné řešení daného problému.

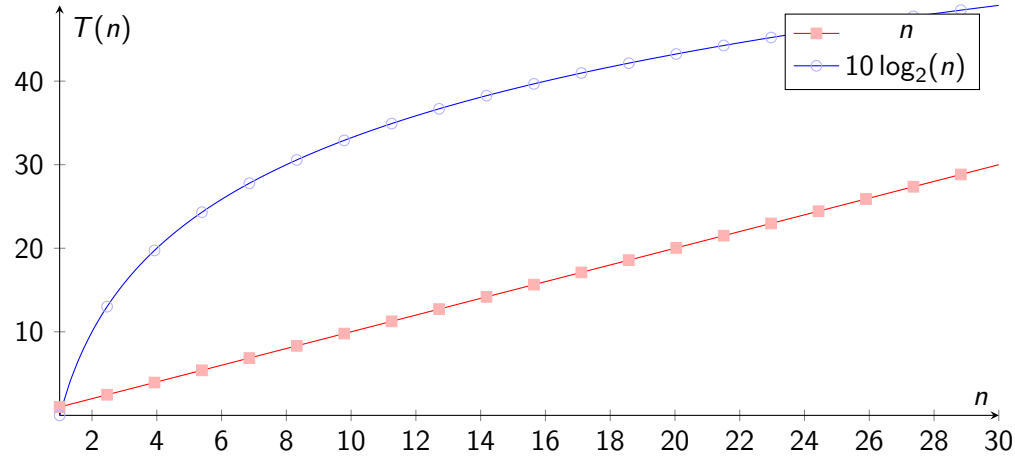
# Příklad řazení I

Graf pro příklad s řazením:



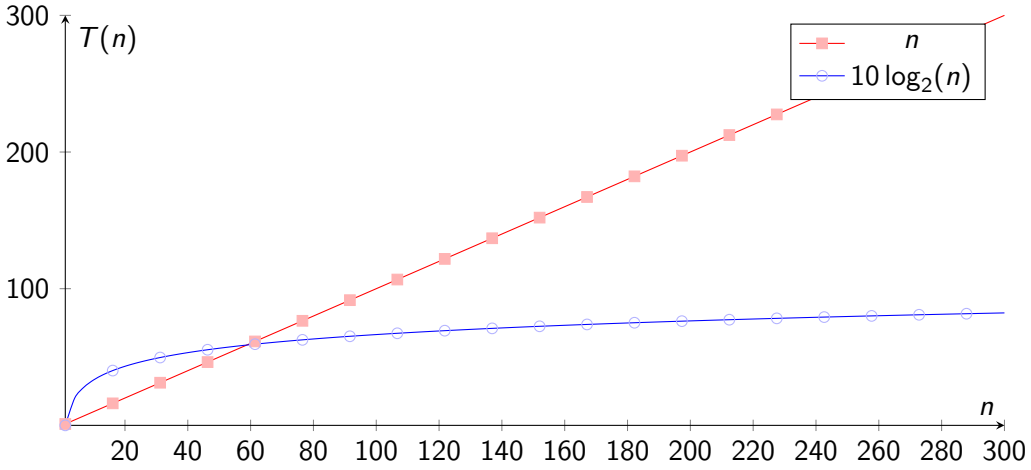
# Příklad řazení II

Co když budeme počítat náročnost operací:



# Příklad řazení III

Náročné operace s více daty:



# Řád růstu funkce.

# Asymptotická složitost

- Přesný vzorec $T(n)$ je pro netriviální algoritmy obtížné určit.
- Rozdíly algoritmů se projeví až pro velká $n \to +\infty$.
- V těchto limitních stavech (nebo pro velká $n$) lze méně rostoucí komponenty $T(n)$ zanedbat.
- Užíváme pojem asymptotická složitost $\to$ asymptoticky se blíží k této hodnotě.

## Příklad pro polynomy

$T_1(n) = n^3$ a $T_2(n) = cn^2$, chceme ukázat, že $T_1(n) > T_2(n)$ pro $n \to \infty$ ($c$ je konstanta). Ukážeme, že $\lim_{n\to\infty} \frac{T_1(n)}{T_2(n)} = \infty$.

$$\lim_{n\to\infty} \frac{T_1(n)}{T_2(n)} = \lim_{n\to\infty} \frac{n^3}{cn^2} = \lim_{n\to\infty} \frac{n}{c} = \infty$$

# $\mathcal{O}$-notace I

## Příklad pro polynomy

$T_1(n) = n^3$ a $T_2(n) = cn^2$, chceme ukázat, že $T_1(n) > T_2(n)$ pro $n \to \infty$ ($c$ je konstanta). Ukážeme, že $\lim_{n\to\infty} \frac{T_1(n)}{T_2(n)} = \infty$.

$$\lim_{n\to\infty} \frac{T_1(n)}{T_2(n)} = \lim_{n\to\infty} \frac{n^3}{cn^2} = \lim_{n\to\infty} \frac{n}{c} = \infty$$

Definice limity jdoucí do nekonečna nám říká, pro každé reálné číslo $M > 0$ najdeme číslo $N$, tak že $\frac{T_1(n)}{T_2(n)} > M$ pro každné $n > N$. Vidíme, že vždy najdeme dostatečně velké N, aby platilo $T_1(n) > T_2(n)$.

## Definice ($\mathcal{O}$-notace)

$T(n)$ patří do $\mathcal{O}(g(n))$ pokud existují konstanty $c > 0$ a $n_0 > 0$ takové, že $T(n) \leq cg(n)$ pro všechna $n \geq n_0$.

# $\mathcal{O}$-notace

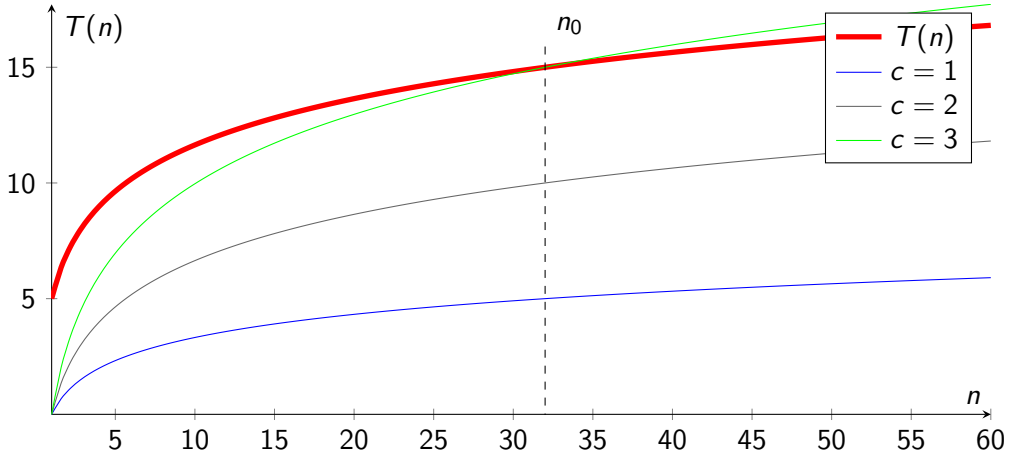
**Definice ($\mathcal{O}$-notace)**

$T(n)$ patří do $\mathcal{O}(g(n))$ pokud existují konstanty $c > 0$ a $n_0 > 0$ takové, že $T(n) \leq cg(n)$ pro všechna $n \geq n_0$.

- $\mathcal{O}(g(n))$ je *asymptotická horní mez* (horní odhad).
- Uvádíme nejlepší známý odhad.

# Vizualizace $\mathcal{O}$-notace

$T(n) = 2\log_2(n) + 5,\ g(n) = \log_2(n)$



# Třídy složitostí

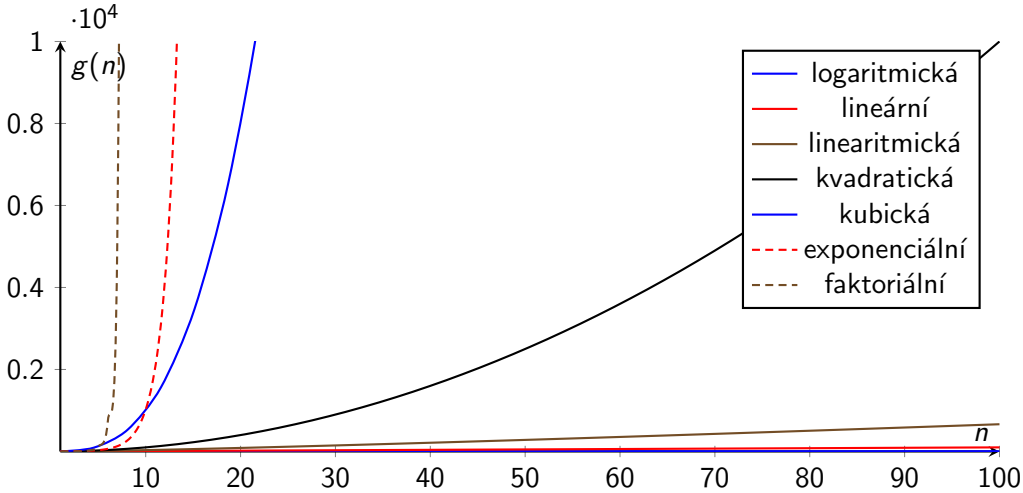
Běžné používané třídy složitosti:

- $\mathcal{O}(1)$ – konstantní složitost
- $\mathcal{O}(\log n)$ – logaritmická složitost
- $\mathcal{O}(n)$ – lineární složitost
- $\mathcal{O}(n \log n)$ – linearitmická složitost
- $\mathcal{O}(n^2)$ – kvadratická složitost
- $\mathcal{O}(n^3)$ – kubická složitost
- $\mathcal{O}(n^k)$ – polynomiální složitost
- $\mathcal{O}(2^n)$ – exponenciální složitost
- $\mathcal{O}(n!)$ – faktoriální složitost

Poznámky:

- Můžeme používat i jiné funkce. Některé dávají smysl, jiné moc ne. Například:
- $\mathcal{O}(n^2 \log n)$, $\mathcal{O}(3^n)$ mohout být opodstatněné.
- $\mathcal{O}(3n^2 + 5)$ je nevhodná volba $(\mathcal{O}(3n^2 + 5) = \mathcal{O}(n^2))$.

# Srovnání složitostí



# Doba výpočtu

Modelový příklad pro ilustraci doby výpočtu v závislosti na složitosti algoritmu a velikosti vstupních dat. Předpokládáme trvání jedné operace 1 ns.

| $\Theta(n)$ | Velikost vstupních dat / Doba výpočtu | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | 10 | 20 | 50 | 100 | 1 000 | 1 000 000 | $1 \times 10^9$ | $1 \times 10^{20}$ |
| 1 | 1 ns | 1 ns | 1 ns | 1 ns | 1 ns | 1 ns | 1 ns | 1 ns |
| $\log(n)$ | 4 ns | 5 ns | 6 ns | 7 ns | 10 ns | 20 ns | 30 ns | 67 ns |
| $n$ | 10 ns | 20 ns | 50 ns | 100 ns | 1 µs | 1 ms | 1 s | 3 171 let |
| $n\log(n)$ | 34 ns | 87 ns | 283 ns | 665 ns | 10 µs | 20 ms | 30 s | 210 675 let |
| $n^2$ | 100 ns | 400 ns | 3 µs | 10 µs | 1 ms | 17 min | 32 let | $3 \times 10^{23}$ let |
| $n^3$ | 1 µs | 8 µs | 125 µs | 1 ms | 1 s | 32 let | | |
| $2^n$ | 1 µs | 1 ms | 13 dní | $4 \times 10^{13}$ let | | | | |
| $n!$ | 4 ms | 77 let | $9 \times 10^{47}$ let | | | | | |

Poznámka: stáří vesmíru je $1.37 \times 10^{10}$ let.

# Dominantní výraz

Při určování třídy složitosti z funkce $T(n)$ odstraníme:

- všechny konstanty,
- sčítance nižších tříd.

## Příklad

$$T(n) = \frac{n^2}{8} + 12\log(n)$$

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

## Poznámka

Hledáme nejlepší třídu složitosti. Téměř vždy můžeme říci, že funkce má složitost $\mathcal{O}(n!)$ a bude to správně, ale pro praktické srovnání algoritmů nám to neposlouží.

# Rozšiřující poznatky k $\mathcal{O}$-notaci

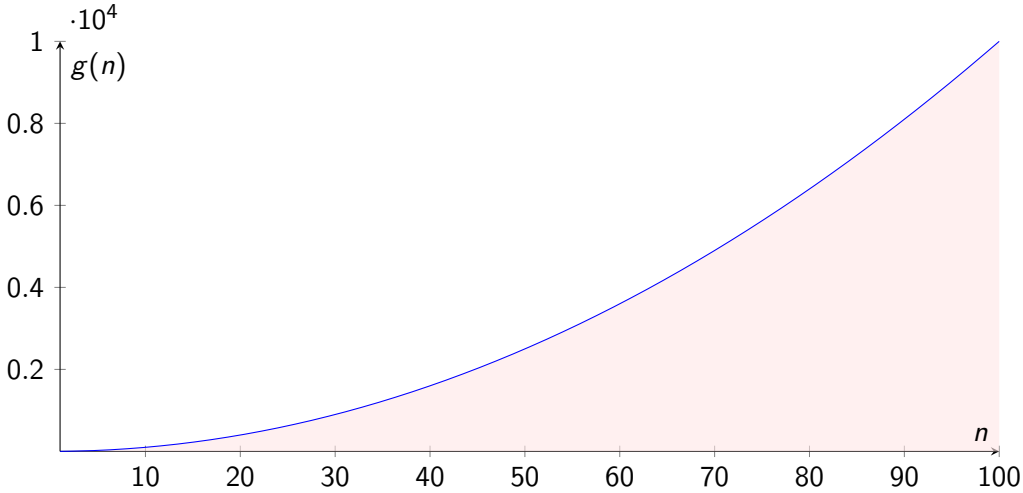
$\mathcal{O}$ omezuje funkci shora. Existují další odhady:

- $\Omega$ notace – asymptotická dolní mez (dolní odhad).
- $\Theta$ notace – asymptotická těsná mez ($\mathcal{O}$ a $\Theta$ zároveň).

Je nutné zvažovat, jak je $T(n)$ stanovena:

- **Nejhorší případ.**
- Průměrný případ (očekávaná hodnota pro náhodnou proměnnou).
- Amortizovaná složitost (ignoruje lokální odchylky).

# Vizualizace $\mathcal{O}$-notace



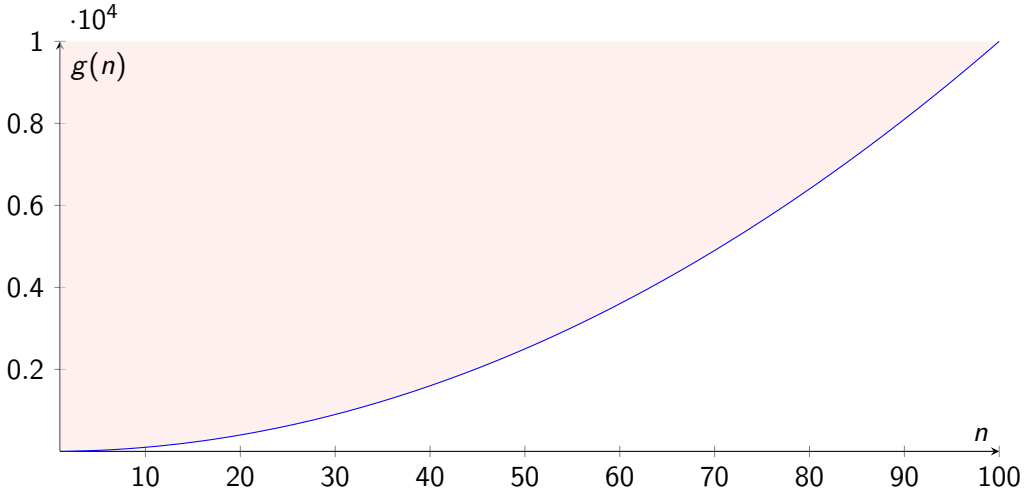
# Ω-notace

**Definice (Ω-notace)**

$T(n)$ patří do $\Omega(g(n))$ pokud existují konstanty $c > 0$ a $n_0 > 0$ takové, že $T(n) \geq cg(n)$ pro všechna $n \geq n_0$.

- $\Omega(g(n))$ je *asymptotická dolní mez*.
- Opět uvádíme co nejbližší známý odhad (tedy funkci s největším růstem).

# Vizualizace Ω-notace



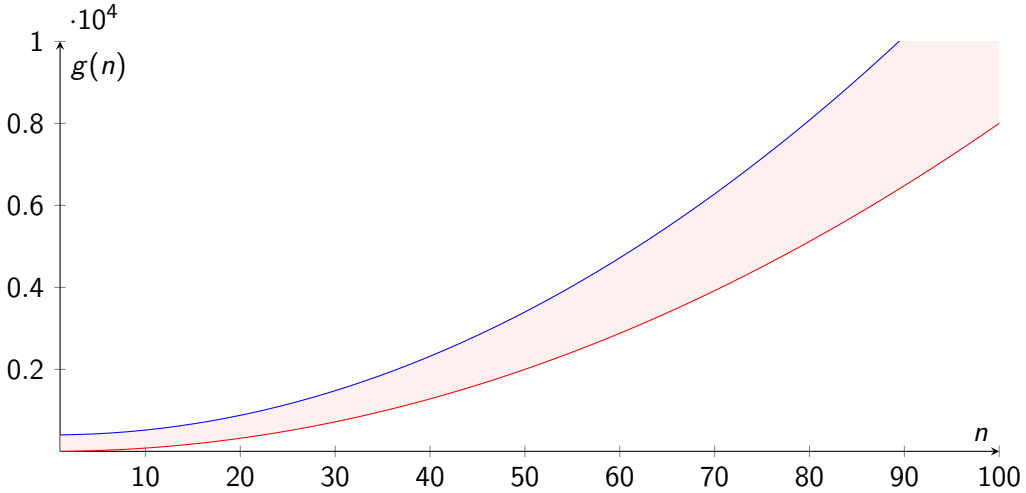
# Θ-notace

**Definice (Θ-notace)**

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

- $\Theta(g(n))$ je *asymptotická těsná mez*.
- Asymptoticky omezuje funkci zároveň shora i zdola.
- Nemusí vždy existovat ($g_1 \neq g_2$ pro $\mathcal{O}(g_1(n))$ a $\Omega(g_2(n))$).
- V praxi se často používá $\mathcal{O}$-notace ve smyslu Θ-notace.
- V přednáškách budeme užívat Θ-notaci vždy, když to bude možné.

# Vizualizace Θ-notace



# Kvíz I

# Praktické příklady na složitost.

# Výrazy

| T(n) funkce | Třída složitosti |
|---|---|
| $5n^2 + 3n + 2$ | |
| $3n\log(n) + 2n + 7$ | |
| $2n + \log(n) + 1$ | |
| $7n^3 + 4n^2\log(n) + 2n\log(n) + 3n + 1$ | |
| $100n + 5$ | |
| $2^n + 100n^2$ | |
| $10n\log(n) + 100n$ | |
| $3n^2 + 10n\log(n) + 50n + 20$ | |

# Výrazy

| **T(n) funkce** | **Třída složitosti** |
|---|---|
| $5n^2 + 3n + 2$ | $\Theta(n^2)$ |
| $3n\log(n) + 2n + 7$ | |
| $2n + \log(n) + 1$ | |
| $7n^3 + 4n^2\log(n) + 2n\log(n) + 3n + 1$ | |
| $100n + 5$ | |
| $2^n + 100n^2$ | |
| $10n\log(n) + 100n$ | |
| $3n^2 + 10n\log(n) + 50n + 20$ | |

# Výrazy

| **T(n) funkce** | **Třída složitosti** |
|---|---|
| $5n^2 + 3n + 2$ | $\Theta(n^2)$ |
| $3n\log(n) + 2n + 7$ | $\Theta(n\log(n))$ |
| $2n + \log(n) + 1$ | |
| $7n^3 + 4n^2\log(n) + 2n\log(n) + 3n + 1$ | |
| $100n + 5$ | |
| $2^n + 100n^2$ | |
| $10n\log(n) + 100n$ | |
| $3n^2 + 10n\log(n) + 50n + 20$ | |

# Výrazy

| **T(n) funkce** | **Třída složitosti** |
|---|---|
| $5n^2 + 3n + 2$ | $\Theta(n^2)$ |
| $3n\log(n) + 2n + 7$ | $\Theta(n\log(n))$ |
| $2n + \log(n) + 1$ | $\Theta(n)$ |
| $7n^3 + 4n^2\log(n) + 2n\log(n) + 3n + 1$ | |
| $100n + 5$ | |
| $2^n + 100n^2$ | |
| $10n\log(n) + 100n$ | |
| $3n^2 + 10n\log(n) + 50n + 20$ | |

# Výrazy

| **T(n) funkce** | **Třída složitosti** |
|---|---|
| $5n^2 + 3n + 2$ | $\Theta(n^2)$ |
| $3n\log(n) + 2n + 7$ | $\Theta(n\log(n))$ |
| $2n + \log(n) + 1$ | $\Theta(n)$ |
| $7n^3 + 4n^2\log(n) + 2n\log(n) + 3n + 1$ | $\Theta(n^3)$ |
| $100n + 5$ | |
| $2^n + 100n^2$ | |
| $10n\log(n) + 100n$ | |
| $3n^2 + 10n\log(n) + 50n + 20$ | |

# Výrazy

| **T(n) funkce** | **Třída složitosti** |
|---|---|
| $5n^2 + 3n + 2$ | $\Theta(n^2)$ |
| $3n\log(n) + 2n + 7$ | $\Theta(n\log(n))$ |
| $2n + \log(n) + 1$ | $\Theta(n)$ |
| $7n^3 + 4n^2\log(n) + 2n\log(n) + 3n + 1$ | $\Theta(n^3)$ |
| $100n + 5$ | $\Theta(n)$ |
| $2^n + 100n^2$ | |
| $10n\log(n) + 100n$ | |
| $3n^2 + 10n\log(n) + 50n + 20$ | |

# Výrazy

| **T(n) funkce** | **Třída složitosti** |
|---|---|
| $5n^2 + 3n + 2$ | $\Theta(n^2)$ |
| $3n\log(n) + 2n + 7$ | $\Theta(n\log(n))$ |
| $2n + \log(n) + 1$ | $\Theta(n)$ |
| $7n^3 + 4n^2\log(n) + 2n\log(n) + 3n + 1$ | $\Theta(n^3)$ |
| $100n + 5$ | $\Theta(n)$ |
| $2^n + 100n^2$ | $\Theta(2^n)$ |
| $10n\log(n) + 100n$ | $\Theta(n\log(n))$ |
| $3n^2 + 10n\log(n) + 50n + 20$ | $\Theta(n^2)$ |

# Program I

Součet čísel od 1 do $n$.

```
1  def summation(n: int) -> int:
2      result: int = 0
3      for i in range(1, n+1):
4          result += i
5      return result
6  summation(5)
```

# Program I

Součet čísel od 1 do $n$.

```
1  def summation(n: int) -> int:
2      result: int = 0
3      for i in range(1, n+1):
4          result += i
5      return result
6  summation(5)
```

Složitost: $\Theta(n)$

# Program II

Součet čísel od 1 do $n$ lépe.

```
1  def summation(n):
2      return n * (n + 1) // 2

3  summation(5)
```

# Program II

Součet čísel od 1 do $n$ lépe.

```
1  def summation(n):
2      return n * (n + 1) // 2

3  summation(5)
```

Složitost: $\Theta(1)$

# Program III

Nalezení největšího prvku v seznamu.

```
1  def find_max(arr: List[int]) -> int:
2      max_val: int = arr[0]
3      for val in arr:
4          if val > max_val:
5              max_val = val
6      return max_val
7  array: List[int] = [2, 5, 1, 8, 4]
8  max_val = find_max(array)
```

# Program III

Nalezení největšího prvku v seznamu.

```
1  def find_max(arr: List[int]) -> int:
2      max_val: int = arr[0]
3      for val in arr:
4          if val > max_val:
5              max_val = val
6      return max_val
7  array: List[int] = [2, 5, 1, 8, 4]
8  max_val = find_max(array)
```

Složitost: $\Theta(n)$

# Program IV

Tisk trojúhleníků z hvězdiček.

```
1  def print_star_triangle(num_lines: int) -> None:
2      for i in range(num_lines):
3          for j in range(i+1):
4              print('*', end='')
5          print()

6  print_star_triangle(10)
```

# Program IV

Tisk trojúhleníků z hvězdiček.

```
1  def print_star_triangle(num_lines: int) -> None:
2      for i in range(num_lines):
3          for j in range(i+1):
4              print('*', end='')
5          print()

6  print_star_triangle(10)
```

Složitost: $\Theta(n^2)$

# Program V

Tisk obdélníku z hvězdiček.

```
1  def print_star_rectangle(height: int, width: int) -> None:
2      for i in range(height):
3          for j in range(width):
4              if i == 0 or i == height - 1 or j == 0 or j == width - 1:
5                  print('*', end='')
6              else:
7                  print(' ', end='')
8          print()

9  print_star_rectangle(10,5)
```

# Program V

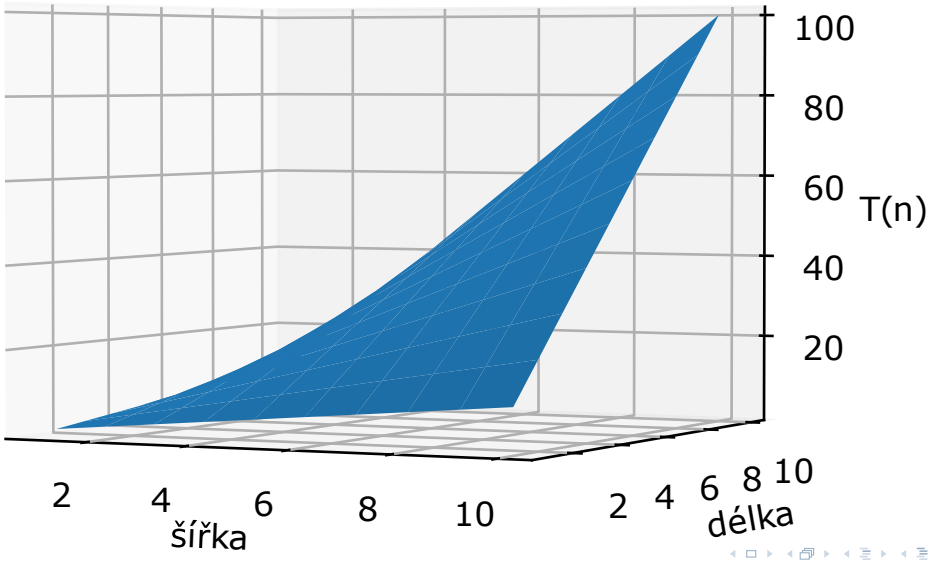
Tisk obdélníku z hvězdiček.

```
1  def print_star_rectangle(height: int, width: int) -> None:
2      for i in range(height):
3          for j in range(width):
4              if i == 0 or i == height - 1 or j == 0 or j == width - 1:
5                  print('*', end='')
6              else:
7                  print(' ', end='')
8          print()

9  print_star_rectangle(10,5)
```

Složitost: $\Theta(height \cdot width)$

# Program V – Vizualizace



# Program VI

Generování všech variací barev.

```
1  def generate_color_variations(colors: List[str], length: int = -1) ->
   ↪  List[List[str]]:

2      if length == -1:
3          length = len(colors)
4      elif length == 0:
5          return [[]]

6      combinations = []
7      for color in colors:
8          for sub_combination in generate_color_variations(colors, length-1):
9              combinations.append([color] + sub_combination)

10     return combinations
```

# Kvíz II

# Program VI – Složitost

Spuštění:

```
1 colors = ['červená', 'modrá', 'zelená']

2 combinations = generate_color_variations(colors)
```

Složitost generování všech variací barev:

$$\Theta(n^n)$$

# Složitost v praxi

Praktická poznámka na závěr:

- "Potlačené"části $T(n)$ (konstanty, slabší polynomy, etc.) hrají roli!
- Benchmarking je důležitý.
- Algoritmus $T_1 = n^2$ bude 10x rychlejší než $T_2 = 10n^2$