

# Teorie algoritmů — 3. týden

Marie Demlová

<http://math.fel.cvut.cz/en/people/demlova>

3. 3. 2026

## Amortizovaná složitost

**Amortizovaná složitost** jedné operace v posloupnosti  $n$  opakování instrukce je průměrná doba připadající na jednu instrukci v čase potřebném pro  $n$  instrukcí v nejhorším případě.

Je-li  $T(n)$  čas potřebný pro  $n$  opakování instrukce, pak

$$\text{amortizovaná složitost je } \frac{T(n)}{n}$$

Existují tři způsoby výpočtu amortizované složitosti

- ▶ agregační metoda
- ▶ účetní metoda
- ▶ potenciálová metoda

## Amortizovaná složitost

- ▶ Agregáčnící metoda — přímý výpočet  $T(n)$  pro všech  $n$  instrukcích, pak amortizovaná složitost jedné instrukce je  $T(n) / n$ .
- ▶ Účetní metoda — provedení každé instrukce má jistý kredit  $\hat{c}_i$ . Je-li  $c_i$  skutečná cena  $i$ -té instrukce, pak  $\hat{c}_i$  je nutno volit tak, aby pro všechna  $k \leq n$

$$\sum_{i=1}^k \hat{c}_i \geq \sum_{i=1}^k c_i.$$

- ▶ Potenciálová metoda — nechť  $D_i$  je stav dat. struktury po  $i$ -té instrukci. Dále  $c_i$  je skutečná cena  $i$ -té instrukce,  $\Phi(D_i)$  potenciál. Amortizovaná cena  $\hat{c}_i$  je

$$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}).$$

# Amortizovaná složitost

## Potenciálová metoda — pokračování

Pak platí (pro každé  $k \leq n$ )

$$\sum_{i=1}^k \hat{c}_i = \sum_{i=1}^k c_i + \Phi(D_k) - \Phi(D_0).$$

Proto z  $\Phi(D_k) \geq \Phi(D_0)$  dostáváme  $\sum_{i=1}^k c_i \leq \sum_{i=1}^k \hat{c}_i$ .

# Amortizovaná složitost

## Příklad.

Určeme amortizovanou složitost následujícího pseudokódu

INCREMENT( $A$ )

1.  $i = 0$
2. if  $i < A.length$  a  $A[i] = 1$
3.      $A[i] := 0$
4.      $i := i + 1$
5. else if  $i < A.length$
6.      $A[i] := 1$

# Časová složitost algoritmů

## Euklidův algoritmus.

Rekurzivní verze Euklidova algoritmu:

`EUCLID( $a, b$ )`

1. `if  $b = 0$`

2.     `return  $a$`

3. `else return EUCLID( $b, a \bmod b$ )`

# Časová složitost algoritmů

## Horní odhad složitosti

**Lemma 1.** Označme  $x_k$  a  $y_k$  dvojici čísel  $x_k > y_k$  po  $k$ -tém rekurzivním volání. Pak  $y_{k+2} < \frac{y_k}{2}$ .

## Dolní odhad složitosti

**Lemma 2.** Je-li  $a > b \geq 1$  a algoritmus  $\text{EUKLID}(a, b)$  potřebuje  $k$  rekurzivních volání, pak  $a \geq F(k+2)$  a  $b \geq F(k+1)$ , kde  $F(i)$  je  $i$ -tý člen Fibonacciho posloupnosti.

Připomeňme, že Fibonacciho posloupnost je posloupnost:

$$F(0) = 0, F(1) = 1, F(n) = F(n-1) + F(n-2) \text{ pro } n \geq 2.$$

# Časová složitost algoritmů

**Lemma 3.**  $\text{EUKLID}(F(k+2), F(k+1))$  potřebuje  $k$  rekurzivních volání.

**Lemma 4.** Pro každé  $n \geq 0$  platí  $F(n+2) \geq \left(\frac{3}{2}\right)^n$ .

## Věta.

Algoritmus  $\text{EUKLID}(a, b)$  vyžaduje  $\Theta(\lg b)$  rekurzivních volání. Proto časová složitost je lineární vzhledem k velikosti vstupu (který je úměrný  $\lg(a+b)$ ).

# Průměrná časová složitost

## Příklad.

Spočítejme průměrnou časovou složitost QuickSortu.

## Tvrzení.

Pro každé  $n \geq 2$  je  $T_{aver}(n) \leq k \cdot n \ln n$ , pro  
 $k = 2c + T_{aver}(0) + T_{aver}(1)$ .

## Důsledek.

Pro průměrnou složitost  $T_{aver}(n)$  algoritmu QuickSort platí

$$T_{aver}(n) \in \mathcal{O}(n \lg n).$$

# Správnost algoritmů

## Variant.

**Variant** je hodnota daná přirozeným číslem, která se během práce algoritmu snižuje (nebo zvětšuje) až nabude nejmenší (nebo největší) možnou hodnotu.

Zaručuje ukončení algoritmu po konečně mnoha krocích.

## Invariant.

**Invariant**, také **podmíněná správnost algoritmu**, je takové tvrzení, že

- ▶ platí před provedením prvního cyklu algoritmu nebo po vykonání prvního cyklu,
- ▶ platí-li před vykonáním cyklu, platí i po jeho vykonání,
- ▶ při ukončení práce algoritmu zaručuje správnost řešení.

# Správnost algoritmů

## Správnost Euklidova algoritmu.

**Variant** — zbytek při dělení, tj. číslo  $a \pmod{b}$ . Protože posloupnost zbytků je klesající posloupnost přirozených, po konečně mnoha krocích musí zbytek být 0.

### **Invariant: Tvrzení.**

Dvojice  $a, b$  a dvojice  $b, a \pmod{b}$  mají stejné společné dělitele; tedy i největšího společného dělitele.