Složitost algoritmů

Složitost algoritmu

- Počítače pracují rychle, ale ne nekonečně rychle. Provedení každé instrukce trvá nějakou (i když velmi krátkou) dobu.
- Stejný problém může řešit více různých algoritmů a doba výpočtu (daná hlavně počtem provedených instrukcí) může být pro různé algoritmy různá.
- Algoritmy bychom chtěli mezi sebou porovnávat a zvolit si ten lepší.
- Algoritmy můžeme naprogramovat a změřit čas výpočtu. Tím zjistíme jak dlouho trvá výpočet na konkrétních datech, na kterých algoritmus testujeme.
- Chtěli bychom mít i nějakou přesnější představu o tom, jak dlouho bude trvat výpočet na všech možných vstupních datech.

Časová složitost

Problém „Vyhledávání“

Vstup: Celé číslo x a sekvence celých čísel $a_1, a_2, \ldots, a_n$ (kde $a_i \neq 0$) ukončená 0.

Výstup: Pokud $a_i = x$, je výstupem i (pokud jich je takových i více, tak nejmenší z nich), jinak je výstupem 0.

```
start:  READ
        STORE  3
        LOAD   =1
cyklus: STORE  2
        READ
        JZERO  vypis
        SUB    3
        JZERO  nasel
        LOAD   2
        ADD    =1
        JUMP   cyklus
nasel:  LOAD   2
vypis:  WRITE
        HALT
```

Časová složitost

```
start:  READ
        STORE  3
        LOAD   =1
cyklus: STORE  2
        READ
        JZERO  vypis
        SUB    3
        JZERO  nasel
        LOAD   2
        ADD    =1
        JUMP   cyklus
nasel:  LOAD   2
vypis:  WRITE
        HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 0
Buňka 1: 0
Buňka 2: 0
Buňka 3: 0
Buňka 4: 0
⋮

Výstup:

Instrukcí: 0

Časová složitost

```
start:   READ
         STORE   3
         LOAD    =1
cyklus:  STORE   2
         READ
         JZERO   vypis
         SUB     3
         JZERO   nasel
         LOAD    2
         ADD     =1
         JUMP    cyklus
nasel:   LOAD    2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 0
Buňka 1: 0
Buňka 2: 0
Buňka 3: 0
Buňka 4: 0
⋮

Výstup:

Instrukcí: 0

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 9
Buňka 1: 0
Buňka 2: 0
Buňka 3: 0
Buňka 4: 0
⋮

Výstup:

Instrukcí: 1

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 9
Buňka 1: 0
Buňka 2: 0
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 2

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 1
Buňka 1: 0
Buňka 2: 0
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 3

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 1
Buňka 1: 0
Buňka 2: 1
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 4

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 13
Buňka 1: 0
Buňka 2: 1
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 5

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 13
Buňka 1: 0
Buňka 2: 1
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 6

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 4
Buňka 1: 0
Buňka 2: 1
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 7

Časová složitost

```
start:   READ
         STORE   3
         LOAD    =1
cyklus:  STORE   2
         READ
         JZERO   vypis
         SUB     3
         JZERO   nasel
         LOAD    2
         ADD     =1
         JUMP    cyklus
nasel:   LOAD    2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 4
Buňka 1: 0
Buňka 2: 1
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 8

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 1
Buňka 1: 0
Buňka 2: 1
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 9

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 2
Buňka 1: 0
Buňka 2: 1
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 10

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 2
Buňka 1: 0
Buňka 2: 1
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 11

Časová složitost

```
 start:   READ
          STORE   3
          LOAD    =1
cyklus:   STORE   2
          READ
          JZERO   vypis
          SUB     3
          JZERO   nasel
          LOAD    2
          ADD     =1
          JUMP    cyklus
 nasel:   LOAD    2
 vypis:   WRITE
          HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 2
Buňka 1: 0
Buňka 2: 2
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 12

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 5
Buňka 1: 0
Buňka 2: 2
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 13

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 5
Buňka 1: 0
Buňka 2: 2
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 14

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: -4
Buňka 1: 0
Buňka 2: 2
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 15

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: -4
Buňka 1: 0
Buňka 2: 2
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 16

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 2
Buňka 1: 0
Buňka 2: 2
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 17

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 3
Buňka 1: 0
Buňka 2: 2
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 18

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 3
Buňka 1: 0
Buňka 2: 2
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 19

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 3
Buňka 1: 0
Buňka 2: 3
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 20

Časová složitost

```
start:   READ
         STORE   3
         LOAD    =1
cyklus:  STORE   2
         READ
         JZERO   vypis
         SUB     3
         JZERO   nasel
         LOAD    2
         ADD     =1
         JUMP    cyklus
nasel:   LOAD    2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 9
Buňka 1: 0
Buňka 2: 3
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 21

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 9
Buňka 1: 0
Buňka 2: 3
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 22

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 0
Buňka 1: 0
Buňka 2: 3
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 23

Časová složitost

```
start:   READ
         STORE  3
         LOAD   =1
cyklus:  STORE  2
         READ
         JZERO  vypis
         SUB    3
         JZERO  nasel
         LOAD   2
         ADD    =1
         JUMP   cyklus
nasel:   LOAD   2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 0
Buňka 1: 0
Buňka 2: 3
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 24

Časová složitost

```
 start:   READ
          STORE   3
          LOAD    =1
cyklus:   STORE   2
          READ
          JZERO   vypis
          SUB     3
          JZERO   nasel
          LOAD    2
          ADD     =1
          JUMP    cyklus
 nasel:   LOAD    2
 vypis:   WRITE
          HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 3
Buňka 1: 0
Buňka 2: 3
Buňka 3: 9
Buňka 4: 0
⋮

Výstup:

Instrukcí: 25

Časová složitost

```
start:   READ
         STORE   3
         LOAD    =1
cyklus:  STORE   2
         READ
         JZERO   vypis
         SUB     3
         JZERO   nasel
         LOAD    2
         ADD     =1
         JUMP    cyklus
nasel:   LOAD    2
vypis:   WRITE
         HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 3
Buňka 1: 0
Buňka 2: 3
Buňka 3: 9
Buňka 4: 0
⋮

Výstup: 3

Instrukcí: 26

Časová složitost

```
 start:   READ
          STORE   3
          LOAD    =1
cyklus:   STORE   2
          READ
          JZERO   vypis
          SUB     3
          JZERO   nasel
          LOAD    2
          ADD     =1
          JUMP    cyklus
 nasel:   LOAD    2
 vypis:   WRITE
          HALT
```

Vstup:
$9, 13, 5, 9, 7, 2, 0$

Buňka 0: 3
Buňka 1: 0
Buňka 2: 3
Buňka 3: 9
Buňka 4: 0
⋮

Výstup: 3

Instrukcí: 27

Časová složitost

- V uvedeném příkladě, kdy vstup byl

$$9, 13, 5, 9, 7, 2, 0$$

bylo provedeno 27 instrukcí.

Časová složitost

- V uvedeném příkladě, kdy vstup byl

  $$9, 13, 5, 9, 7, 2, 0$$

  bylo provedeno 27 instrukcí.

Rozeberme nyní, kolik instrukcí se provede obecně pro vstup

$$x, a_1, a_2, \ldots, a_n, 0$$

Časová složitost

- V uvedeném příkladě, kdy vstup byl

$$9, 13, 5, 9, 7, 2, 0$$

bylo provedeno 27 instrukcí.

Rozeberme nyní, kolik instrukcí se provede obecně pro vstup

$$x, a_1, a_2, \ldots, a_n, 0$$

- Pokud pro všechna i (kde $1 \leq i \leq n$) platí $x \neq a_i$:

$$3 + 8 \cdot n + 3 + 2 = 8n + 8$$

Časová složitost

- V uvedeném příkladě, kdy vstup byl

$$9, 13, 5, 9, 7, 2, 0$$

bylo provedeno 27 instrukcí.

Rozeberme nyní, kolik instrukcí se provede obecně pro vstup

$$x, a_1, a_2, \ldots, a_n, 0$$

- Pokud pro všechna i (kde $1 \leq i \leq n$) platí $x \neq a_i$:

$$3 + 8 \cdot n + 3 + 2 = 8n + 8$$

- Pokud pro nějaké i (kde $1 \leq i \leq n$) platí $x = a_i$:

$$3 + 8 \cdot (i - 1) + 5 + 3 = 8i + 3$$

Velikost vstupu

Pro různé vstupy provede program různý počet instrukcí.

Pokud chceme počet provedených instrukcí nějak analyzovat, je vhodné si zavést pojem **velikost vstupu**.

Typicky je velikost vstupu číslo, které udává, jak je daná instance „velká“ (čím větší číslo, tím větší instance).

Příklad: Pro problém „Vyhledávání“, kde jsou vstupy tvaru

$$x, a_1, a_2, \ldots, a_n, 0$$

můžeme jako velikost vstupu zvolit například hodnotu n.

Velikost vstupu $9, 13, 5, 9, 7, 2, 0$ je tedy potom 5.

Velikost vstupu

Poznámka: Velikost vstupu si v daném konkrétním případě můžeme definovat, jak chceme a jak je to pro další analýzu výhodné.

Co přesně zvolíme jako velikost vstupu není předem dáno, ale z podstaty zadaného problému většinou nějak přirozeně vyplývá, co za velikost vstupu zvolit.

Příklady:

- Pro problém „Třídění“, kde vstupem je sekvence čísel $a_1, a_2, \ldots, a_n$ a výstupem jsou tato čísla setříděná, můžeme vzít jako velikost vstupu hodnotu n.
- Pro problém „Prvočíselnost“, kde vstupem je přirozené číslo x, a kde se ptáme, zda x je prvočíslo, můžeme vzít jako velikost vstupu počet bitů čísla x.

  (Jinou možností by bylo vzít jako velikost vstupu přímo hodnotu x.)

Velikost vstupu

Někdy je vhodné popsat velikost vstupu pomocí více čísel.

Například u problémů, kde vstupem je graf, můžeme definovat velikost vstupu jako dvojici čísel n, m, kde:

- n – počet vrcholů grafu
- m – počet hran grafu

Poznámka: Jinou možností by bylo definovat velikost vstupu jako jediné číslo $n + m$.

Velikost vstupu

Obecně můžeme pro libovolný problém definovat velikost vstupu následovně:

- Pokud je vstupem slovo w z nějaké obecedy Σ:
  délka slova w
- Pokud je vstupem sekvence bitů (tj. slovo z abecedy $\{0, 1\}$):
  počet bitů v této sekvenci
- Pokud je vstupem přirozené číslo x:
  počet bitů nutných k zápisu čísla x

Časová složitost

Chceme analyzovat konkrétní algoritmus (jeho konkrétní implementaci).

Zajímá nás, kolik instrukcí se provede, pokud algoritmus dostane vstup velikosti $1, 2, 3, 4, \ldots$.

Je zřejmé, že i pro vstupy, které mají stejnou velikost, může být počet provedených instrukcí různý.

Předpokládejme, že X je množina všech možných vstupů daného problému a $g(x)$ je počet instrukcí provedených algoritmem pro vstup $x \in X$.

Označme si velikost vstupu $x \in X$ jako $|x|$.

Nyní definujme následující funkci $f : \mathbb{N} \to \mathbb{N}$ takovou, že pro $n \in \mathbb{N}$ je

$$f(n) = \max\{g(x) \mid x \in X, |x| = n\}$$

Časová složitost v nejhorším případě

Takto definované funkci $f(n)$ (tj. funkci, která pro daný algoritmus a danou definici velikosti vstupu přiřazuje každému přirozenému číslu n maximální počet instrukcí, které algoritmus provede, pokud dostane vstup velikosti n) se říká **časová složitost algorimu v nejhorším případě**.

Časová složitost v nejhorším případě

Takto definované funkci $f(n)$ (tj. funkci, která pro daný algoritmus a danou definici velikosti vstupu přiřazuje každému přirozenému číslu n maximální počet instrukcí, které algoritmus provede, pokud dostane vstup velikosti n) se říká **časová složitost algorimu v nejhorším případě**.

Příklad: Jak jsme zjistili, dříve popsaný algorimus řešící problém „Vyhledávání“ provede pro vstup $x, a_1, a_2, \ldots, a_n, 0$ následující počty instrukcí:

- Pokud pro všechna i platí $x \neq a_i$, algorimus provede $8n + 8$ instrukcí.
- Pokud pro nějaké i platí $x = a_i$, algorimus provede $8i + 3$ instrukcí.

Vzhledem k tomu, že ve druhém případě je vždy $i \leq n$, je časová složitost tohoto algorimu v nejhorším případě $f(n) = 8n + 8$.

Časová složitost v průměrném případě

Kromě časové složitosti v nejhorším případě má smysl zkoumat i časovou složitost **v průměrném případě**.

V tomto případě $f(n)$ nedefinujeme jako maximum, ale jako aritmetický průměr z hodnot

$$\{g(x) \mid x \in X, |x| = n\}$$

- Určit časovou složitost v průměrném případě je většinou těžší než určit časovou složitost v nejhorším případě.
- Často se tyto dvě funkce příliš neliší, někdy je ale rozdíl významný.

Poznámka: Zkoumat složitost v nejlepším případě většinou moc smysl nemá.

Rychlost růstu funkcí

Program zpracovává vstup velikosti n.
Předpokládejme, že pro vstup velikosti n provede $f(n)$ operací, a že provedení jedné operace trvá $1\,\mu\mathrm{s}$ ($10^{-6}\,\mathrm{s}$).

	n							
$f(n)$	20	40	60	80	100	200	500	1000
n	$20\,\mu$s	$40\,\mu$s	$60\,\mu$s	$80\,\mu$s	0.1 ms	0.2 ms	0.5 ms	1 ms
$n \log n$	$86\,\mu$s	0.213 ms	0.354 ms	0.506 ms	0.664 ms	1.528 ms	4.48 ms	9.96 ms
n^2	0.4 ms	1.6 ms	3.6 ms	6.4 ms	10 ms	40 ms	0.25 s	1 s
n^3	8 ms	64 ms	0.216 s	0.512 s	1 s	8 s	125 s	16.7 min.
n^4	0.16 s	2.56 s	12.96 s	42 s	100 s	26.6 min.	17.36 hod.	11.57 dní
2^n	1.05 s	12.75 dní	36560 let	$38.3 \cdot 10^9$ let	$40.1 \cdot 10^{15}$ let	$50 \cdot 10^{45}$ let	$10.4 \cdot 10^{136}$ let	–
$n!$	77147 let	$2.59 \cdot 10^{34}$ let	$2.64 \cdot 10^{68}$ let	$2.27 \cdot 10^{105}$ let	$2.96 \cdot 10^{144}$ let	–	–	–

Rychlost růstu funkcí

Uvažujme 3 algoritmy se složitostmi $t_1(n) = n, t_2(n) = n^3, t_3(n) = 2^n$. Náš počítač zvládne v reálném čase (kolik jsme ochotni počkat) 10^{12} kroků.

Složitost	Velikost vstupu
$t_1(n) = n$	10^{12}
$t_2(n) = n^3$	10^4
$t_3(n) = 2^n$	40

Rychlost růstu funkcí

Uvažujme 3 algoritmy se složitostmi $t_1(n) = n, t_2(n) = n^3, t_3(n) = 2^n$. Náš počítač zvládne v reálném čase (kolik jsme ochotni počkat) 10^{12} kroků.

Složitost	Velikost vstupu
$t_1(n) = n$	10^{12}
$t_2(n) = n^3$	10^4
$t_3(n) = 2^n$	40

Nyní počítač 1000 násobně zrychlíme. Zvládne tedy 10^{15} kroků.

Složitost	Velikost vstupu	Nárust
$t_1(n) = n$	10^{15}	$1000\times$
$t_2(n) = n^3$	10^5	$10\times$
$t_3(n) = 2^n$	50	$+10$

Asymptotická notace

- Přesnou složitost bývá problém vyjádřit.
- Přesná složitost je silně závislá na konkrétním zvoleném modelu a konkrétní implementaci (na detailech této implementace).
- Složitost nás většinou zajímá hlavně pro velké vstupy. Pro malé vstupy obvykle i neefektivní algoritmus proběhne rychle.
- Ve většině případů nepotřebujeme znát přesný počet provedených instrukcí, ale spokojíme se s odhadem toho, jak rychle tento počet narůstá se zvětšováním velikosti vstupu.
- Proto zavádíme tzv. **asymptotickou notaci**, která nám umožní zanedbat méně důležité detaily a odhadnout přibližně, jak rychle daná funkce roste, a která analýzu podstatným způsobem zjednodušuje.

Asymptotická notace

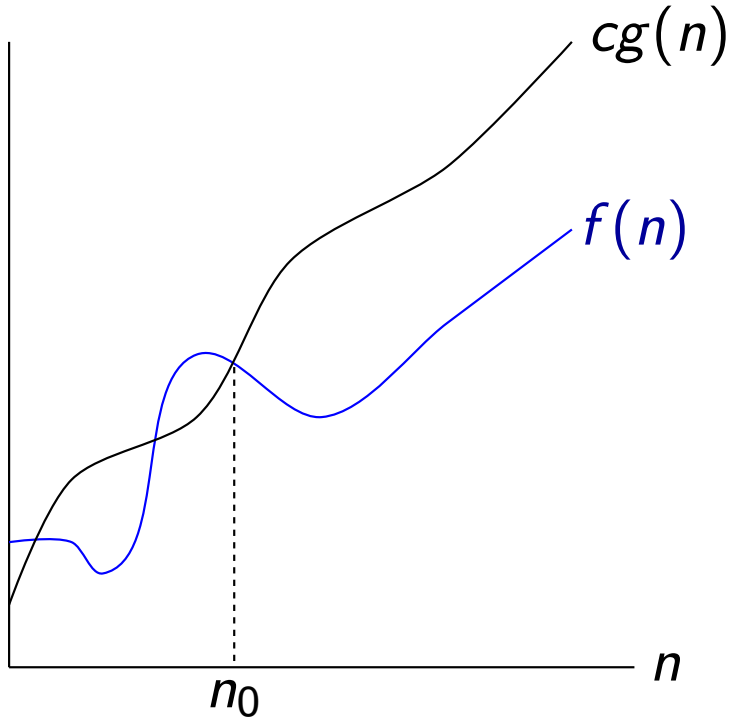
Vezměme si libovolnou funkci $g : \mathbb{N} \to \mathbb{N}$. Zápisy $O(g)$, $\Omega(g)$ a $\Theta(g)$ označují **množiny funkcí** typu $\mathbb{N} \to \mathbb{N}$, kde:

- $O(g)$ – množina všech funkcí, které rostou nejvýše tak rychle jako g
- $\Omega(g)$ – množina všech funkcí, které rostou alespoň tak rychle jako g
- $\Theta(g)$ – množina všech funkcí, které rostou stejně rychle jako g

Poznámka: Toto nejsou definice! Ty následují na následujících slidech.

- O – velké „O“
- Ω – velké řecké písmeno „omega“
- Θ – velké řecké písmeno „theta“

Asymptotická notace – symbol O

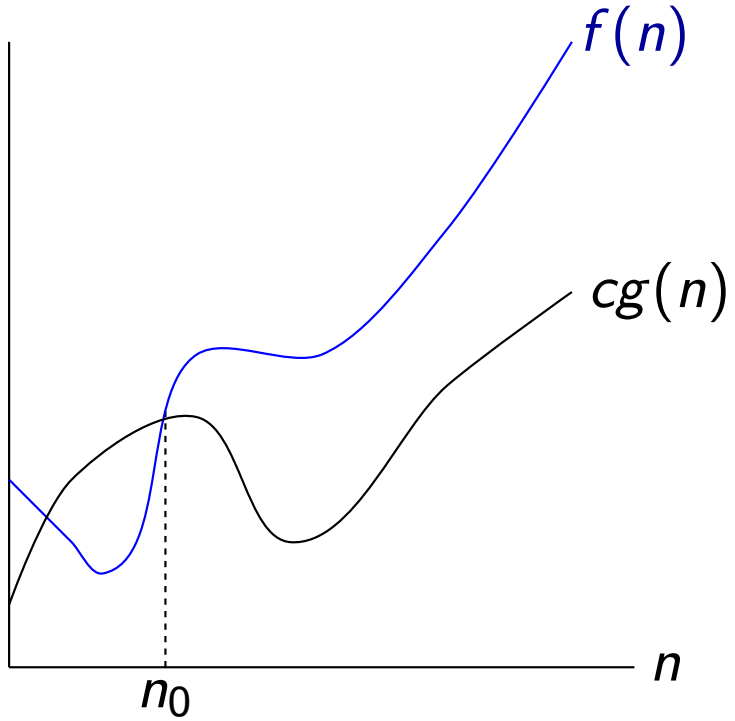


Definice

Vezměme si libovolnou funkci $g : \mathbb{N} \to \mathbb{N}$. Pro funkci $f : \mathbb{N} \to \mathbb{N}$ platí $f \in O(g)$ právě tehdy, když

$$(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0) : f(n) \leq c\, g(n).$$

Asymptotická notace – symbol Ω

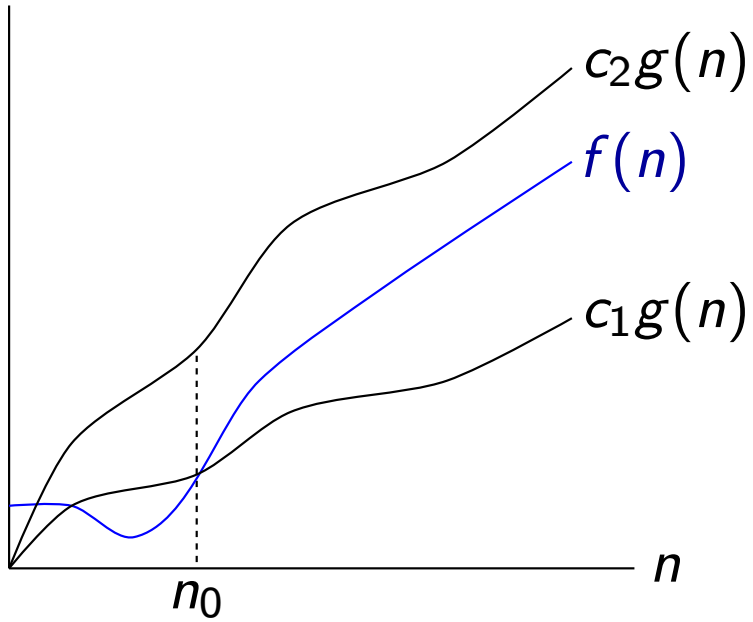


Definice

Vezměme si libovolnou funkci $g : \mathbb{N} \to \mathbb{N}$. Pro funkci $f : \mathbb{N} \to \mathbb{N}$ platí $f \in \Omega(g)$ právě tehdy, když

$$(\exists c > 0)(\exists n_0 \geq 0)(\forall n \geq n_0) : c\, g(n) \leq f(n).$$

Asymptotická notace – symbol Θ



Definice

Vezměme si libovolnou funkci $g : \mathbb{N} \to \mathbb{N}$. Pro funkci $f : \mathbb{N} \to \mathbb{N}$ platí $f \in \Theta(g)$ právě tehdy, když

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

Asymptotická notace

Pro jednoduchost uvažujeme v předchozích definicích pouze funkce typu $\mathbb{N} \to \mathbb{N}$.

Ve skutečnosti by se tyto definice daly rozšířit na všechny **asymptoticky nezáporné** funkce typu $\mathbb{R}_+ \to \mathbb{R}$, které navíc mohou být na nějakém konečné podintervalu svého definičního nedefinované.

Funkce $f : \mathbb{R}_+ \to \mathbb{R}$ je **asymptoticky nezáporná** pokud pro ni platí:

$$(\exists n_0 \geq 0)(\forall n \geq n_0)(f(n) \geq 0)$$

Poznámka: Pro $n < n_0$, může být hodnota $f(n)$ nedefinovaná.

$\mathbb{R}_+ = \{x \in \mathbb{R} \mid x \geq 0\}$

Asymptotická notace

Příklady:

$n \in O(n^2)$
$1000n \in O(n)$
$2^{\log_2 n} \in \Theta(n)$
$n^3 \notin O(n^2)$
$n^2 \notin O(n)$
$n^3 + 2^n \notin O(n^2)$

$n^3 \in O(n^4)$
$0.00001n^2 - 10^{10}n \in \Theta(10^{10}n^2)$
$n^3 - n^2 \log_2^3 n + 1000n - 10^{100} \in \Theta(n^3)$
$n^3 + 1000n - 10^{100} \in O(n^3)$
$n^3 + n^2 \notin \Theta(n^2)$
$n! \notin O(2^n)$

Asymptotická notace

- Pro libovolnou funkci $g : \mathbb{N} \to \mathbb{N}$ platí:

$$g \in O(g) \qquad g \in \Omega(g) \qquad g \in \Theta(g)$$

- Pro libovolné dvě funkce $f, g : \mathbb{N} \to \mathbb{N}$ platí:
  - $f \in O(g)$ právě tehdy, když $g \in \Omega(f)$
  - $f \in \Theta(g)$ právě tehdy, když $g \in \Theta(f)$
  - $f \in \Theta(g)$ právě tehdy, když $f \in O(g)$ a $f \in \Omega(g)$

- Pro libovolné tři funkce $f, g, h : \mathbb{N} \to \mathbb{N}$ platí:
  - jestliže $f \in O(g)$ a $g \in O(h)$, pak $f \in O(h)$
  - jestliže $f \in \Omega(g)$ a $g \in \Omega(h)$, pak $f \in \Omega(h)$
  - jestliže $f \in \Theta(g)$ a $g \in \Theta(h)$, pak $f \in \Theta(h)$

Asymptotická notace

- Existují dvojice funkcí $f, g : \mathbb{N} \to \mathbb{N}$ takové, že

$$f \notin O(g) \qquad \text{a} \qquad g \notin O(f),$$

  například

$$f(n) = n \qquad\qquad g(n) = n^{1+\sin(n)}.$$

- $O(1)$ označuje množinu všech **omezených** funkcí, tj. funkcí jejichž funkční hodnoty jsou shora omezeny nějakou konstantou.

Asymptotická notace

- Pro libovolné dvě funkce $f, g : \mathbb{N} \to \mathbb{N}$ platí:
  - $\max(f, g) \in \Theta(f + g)$
  - pokud $f \in O(g)$, pak $f + g \in \Theta(g)$

- Pro libovolné čtyři funkce $f_1, f_2, g_1, g_2 : \mathbb{N} \to \mathbb{N}$ platí:
  - pokud $f_1 \in O(f_2)$ a $g_1 \in O(g_2)$, pak $f_1 + g_1 \in O(f_2 + g_2)$ a $f_1 \cdot g_1 \in O(f_2 \cdot g_2)$
  - pokud $f_1 \in \Theta(f_2)$ a $g_1 \in \Theta(g_2)$, pak $f_1 + g_1 \in \Theta(f_2 + g_2)$ a $f_1 \cdot g_1 \in \Theta(f_2 \cdot g_2)$

Asymptotická notace

- O funkci f řekneme, že je:

  logaritmická, pokud $f(n) \in \Theta(\log n)$
  lineární, pokud $f(n) \in \Theta(n)$
  kvadratická, pokud $f(n) \in \Theta(n^2)$
  kubická, pokud $f(n) \in \Theta(n^3)$
  polynomiální, pokud $f(n) \in O(n^k)$ pro nějaké $k > 0$
  exponenciální, pokud $f(n) \in O(c^{n^k})$ pro nějaké $c > 1$ a $k > 0$

- Exponenciální funkce se v asymptotické notaci často uvádí ve tvaru $2^{O(n^k)}$, protože potom již nemusíme uvažovat různé základy mocniny.

Asymptotická notace

- Pro libovolná $k, \ell > 0$ taková, že $k < \ell$, a libovolné $c > 1$ platí:

$$n^k \in O(n^\ell) \qquad n^\ell \notin O(n^k)$$

$$\log_c^k n \in O(\log_c^\ell n) \qquad \log_c^\ell n \notin O(\log_c^k n)$$

$$c^{n^k} \in O(c^{n^\ell}) \qquad c^{n^\ell} \notin O(c^{n^k})$$

- Pro libovolná $k, \ell > 0$ a libovolné $c > 1$ platí:

$$\log_c^k n \in O(n^\ell) \qquad n^\ell \notin O(\log_c^k n)$$

$$n^k \in O(c^{n^\ell}) \qquad c^{n^\ell} \notin O(n^k)$$

Poznámka: $\log_c^k n$ je stručnější zápis pro $(\log_c n)^k$.

Asymptotická notace

Tvrzení

Pro libovolná $a, b > 1$ a libovolné $n > 0$ platí

$$\log_a n = \frac{\log_b n}{\log_b a}$$

Důkaz: Z $n = a^{\log_a n}$ plyne $\log_b n = \log_b(a^{\log_a n})$.
Protože $\log_b(a^{\log_a n}) = \log_a n \cdot \log_b a$, dostáváme $\log_b n = \log_a n \cdot \log_b n$,
z čehož plyne výše uvedený závěr. □

Pro libovolné konstanty $a, b > 1$ tedy platí $\log_a n \in \Theta(\log_b n)$.

Z toho důvodu se při použití asymptotické notace základ logaritmu obvykle vynechává: například místo $\Theta(n \log_2 n)$ můžeme napsat $\Theta(n \log n)$.

Asymptotická notace

Jak bylo uvedeno, výrazy $O(g)$, $\Omega(g)$ a $\Theta(g)$ označují určité množiny funkcí.

V odborných textech se však někdy používají tyto výrazy i v poněkud odlišném významu:

- zápis $O(g)$, $\Omega(g)$ nebo $\Theta(g)$ nereprezentuje danou množinu funkcí, ale **nějakou** funkci z dané množiny.

Tato konvence se používá zejména v zápisu rovnic nebo nerovnic.

Příklad: $3n^3 + 5n^2 - 11n + 2 = 3n^3 + O(n^2)$

Při použití této konvence je tedy možné například psát $f = O(g)$ místo $f \in O(g)$.

Asymptotická notace

- Asymptotická notace se dá přímočaře zobecnit pro funkce více argumentů:

  Uvažujme například funkci $g : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$. Pro funkci $f : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$ platí $f(n, m) \in O(g(n, m))$ právě tehdy, když

  $$(\exists c > 0)(\exists n_0 \geq 0)(\exists m_0 \geq 0)(\forall n \geq n_0)(\forall m \geq m_0)(f(n, m) \leq c\, g(n, m))$$

- Kromě výrazů $O(g)$, $\Omega(g)$ a $\Theta(g)$ se používají ještě výrazy $o(g)$ a $\omega(g)$.
  - $o(g)$ – množina všech funkcí, které rostou pomaleji než funkce g
  - $\omega(g)$ – množina všech funkcí, které rostou rychleji než funkce g

  Jejich přesné definice zde nebudeme uvádět a nebudeme se jimi dále zabývat.

Složitost algoritmů

Řekněme, že bychom chtěli analyzovat časovou složitost $t(n)$ nějakého algoritmu, který se skládá z instrukcí $I_1, I_2, \ldots, I_k$:

- Pokud $m_1, m_2, \ldots, m_k$ jsou počty provedení jednotlivých instrukcí pro nějaký vstup x (tj. pro vstup x se instrukce I_i provede m_i krát), tak celkový počet instrukcí provedených pro vstup x je
$$m_1 + m_2 + \cdots + m_k.$$
- Vezměme si funkce $f_1, f_2, \ldots, f_k$, kde $f_i : \mathbb{N} \to \mathbb{N}$, přičemž $f_i(n)$ je maximum z počtu provedení instrukce I_i pro všechny vstupy velikosti n.
- Zjevně platí, že pro libovolnou funkci f_i je $t \in \Omega(f_i)$.
- Zjevně také platí $t \in O(f_1 + f_2 + \cdots + f_k)$.

Složitost algoritmů

- Připomeňme si, že pokud $f \in O(g)$, pak $f + g \in O(g)$.
- Pokud tedy pro některou fukci f_i platí, že pro všechny f_j, kde $j \neq i$, je $f_j \in O(f_i)$, pak

$$t \in O(f_i).$$

- Často se tedy při analýze celkové časové složitosti $t(n)$ můžeme omezit pouze na analýzu počtu provedení nejčastěji prováděné instrukce (pro vstup velikosti n je provedena maximálně $f_i(n)$ krát), protože platí

$$t \in \Theta(f_i).$$

Složitost algoritmů

Příklad: Při analýze složitosti vyhledávání čísla v posloupnosti jsme zjistili, že časová složitost daného algoritmu v nejhorším případě je

$$f(n) = 8n + 8\,.$$

Kdybychom to nechtěli takto podrobně zjišťovat a spokojili se s hrubším odhadem, mohli jsme určit, že časová složitost tohoto algoritmu je $\Theta(n)$, protože:

- Algoritmus obsahuje jediný cyklus, který se provede nejvýše n krát, přičemž pro některé vstupy se skutečně může až n krát provést.
- V rámci jednoho průchodu cyklem se provede několik instrukcí, jejichž počet je shora i zdola omezen nějakými konstantami nezávislými na velikosti vstupu.
- Ostatní instrukce se provedou maximálně jednou. K celkové složitosti tak přispívají přičtením nějaké konstanty.

Složitost algoritmů

Pokusme se analyzovat časovou složitost následujícího algoritmu:

INSERTION-SORT(A, n):

for $j := 2$ **to** n **do**
  $x := A[j]$
  $i := j - 1$
  while $i > 0$ and $A[i] > x$ **do**
    $A[i+1] := A[i]$
    $i := i - 1$
  end while
  $A[i+1] := x$
end for

Tj. chceme najít funkci $t(n)$ takovou, že časová složitost algoritmu INSERTION-SORT v nejhorším případě je v $\Theta(t(n))$.

Složitost algoritmů

Uvažujme vstupy velikosti n:

- Vnější cyklus **for** se provede $n-1$ krát.
- Vnitřní cyklus **while** se pro danou hodnotu j provede maximálně $(j-1)$ krát.
- Existují vstupy, pro které platí že pro každou hodnotu j od 2 do n se vnitřní cyklus **while** provede právě $(j-1)$ krát.
- V nejhorším případě se tedy cyklus **while** provede celkem m krát, kde

$$m = 1 + 2 + \cdots + (n-1) = (1 + (n-1)) \cdot \tfrac{n-1}{2} = \tfrac{1}{2}n^2 - \tfrac{1}{2}n$$

- Celková časová složitost algoritmu Insertion-Sort v nejhorším případě je tedy $\Theta(n^2)$.

Složitost algoritmů

V předchozím případě jsme přesně spočítali celkový počet průchodů cyklem **while**.

Obecně to není vždy možné spočítat takto přesně nebo to může být hodně komplikované. Pokud nás zajímá jen asymptotický odhad, tak to často ani není nutné.

Složitost algoritmů

Pokud bychom například neuměli spočítat součet aritmetické posloupnosti, mohli bychom provést analýzu následovně:

- Vnější cyklus **for** se neprovede více než n krát, vnitřní cyklus **while** se při každé iteraci vnějšího cyklu provede maximálně n krát. Celkově se tedy vnitřní cyklus provede maximálně n^2 krát.

  Platí tedy $t \in O(n^2)$.

- Pro některé vstupy se při posledních $\lfloor n/2 \rfloor$ průchodech cyklem **for** provede cyklus **while** alespoň $\lceil n/2 \rceil$ krát.

  Pro některé vstupy se tedy cyklus **while** provede alespoň $\lfloor n/2 \rfloor \cdot \lceil n/2 \rceil$ krát.

  $\lfloor n/2 \rfloor \cdot \lceil n/2 \rceil \geq (n/2 - 1) \cdot (n/2) = \frac{1}{4}n^2 - \frac{1}{2}n$

  Platí tedy $t \in \Omega(n^2)$.

Složitost algoritmů

Při používání asymptotických odhadů časové složitosti algoritmů bychom si měli být vědomi některých úskalí:

- Asyptotické odhady se týkají pouze toho, jak roste čas s rostoucí velikostí vstupu.
- Neříkají nic o konkrétní době výpočtu. V asymptotické notaci mohou být skryty velké konstanty.
- Algoritmus, který má lepší asymptotickou časovou složitost než nějaký jiný algoritmus, může být ve skutečnosti rychlejší až pro nějaké hodně velké vstupy.
- Většinou analyzujeme složitost v nejhorším případě. Pro některé algoritmy může být doba výpočtu v nejhorším případě mnohem větší než doba výpočtu na „typických“ instancích.

Složitost algoritmů

- Můžeme si to ilustrovat na algoritmech pro třídění.

Algoritmus	Nejhorší	Průměrný
Bubblesort	$\Theta(n^2)$	$\Theta(n^2)$
Heapsort	$\Theta(n \log n)$	$\Theta(n \log n)$
Quicksort	$\Theta(n^2)$	$\Theta(n \log n)$

- Quicksort má horší asymptotickou složitost v nejhorším případě než Heapsort, stejnou asymptotickou složitost v průměrném případě a přesto je v praxi nejrychlejší.

Složitost algoritmů

- Zatím jsme uvažovali, že provedení všech instrukcí trvá stejně dlouho.
- V praxi mohou být některé instrukce časově náročnější.
- Pokud známe časy provedení jednotlivých instrukcí a můžeme si spočítat počty jejich provedení, čas běhu potom dostaneme jako

$$\sum_i m_i t_i$$

kde m_i je počet provedení instrukce i a t_i je čas potřebný k vykonání této instrukce.

Složitost algoritmů

Předpokládejme, že analyzujeme časovou složitost nějakého algoritmu realizovaného strojem RAM.

- Zatím jsme při analýze počítali jen počet provedených instrukcí. Tato se označuje jako použití tzv. **jednotkové míry**.

  Odhady časové složitosti v jednotkové míře odpovídají době běhu na skutečných počítačích za předpokladu, že operace, které provádí stroj RAM, může skutečný počítač provést v konstantním čase.

  To platí, pokud čísla, se kterými algoritmus pracuje, jsou malá (vejdou se např. do 32 nebo 64 bitů).

- Pokud by stroj RAM pracoval s „velkými“ čísly (např. 1000 bitovými), bude odhad časové složitosti v jednotkové míře nerealistický v tom smyslu, že výpočet na skutečném počítači bude trvat mnohem déle.

Složitost algoritmů

- Proto se při analýze časové složitosti algoritmů, u kterých se předpokládá práce s velkými čísly, používá tzv. **logaritmická míra**, kdy doba trvání jedné instrukce stroje RAM není 1, ale je úměrná počtu **bitových operací**, které je třeba pro provedení dané instrukce provést.
- Doba trvání instrukce je tedy závislá na aktuálních hodnotách jejích operandů.
- Například doba provádění instrukcí `ADD` a `SUB` je rovna součtu počtů bitů jejich operandů.
- Doba provádění instrukcí `MUL` a `DIV` je rovna součinu počtů bitů jejich operandů.

Prostorová (paměťová) složitost algoritmů

- Zatím jsme se zajímali o čas, který potřebujeme k výpočtu
- Někdy bývá kritickou velikost paměti potřebné k provedení výpočtu.

Množstvím paměti stroje RAM $\mathcal{M}$ použitým pro vstup x rozumíme počet buněk paměti, které stroj $\mathcal{M}$ během svého výpočtu nad vstupem x použije.

Definice

Prostorová složitost stroje RAM $\mathcal{M}$ (v nejhorším případě) je funkce $s : \mathbb{N} \to \mathbb{N}$, kde $s(n)$ udává maximální množství paměti použité strojem $\mathcal{M}$ pro vstupy délky n.

Prostorová (paměťová) složitost algoritmů

- Pro konkrétní problém můžeme mít dva algoritmy takové, že jeden má menší prostorovou složitost a druhý zase časovou složitost.
- Je-li časová složitost algoritmu v $O(f(n))$ je i prostorová v $O(f(n))$ (počet buněk navštívených RAMem nemůže být větší než počet kroků, protože v každém kroku použije kromě akumulátoru maximálně jednu další buňku).