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 více, tak nejmenší), 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

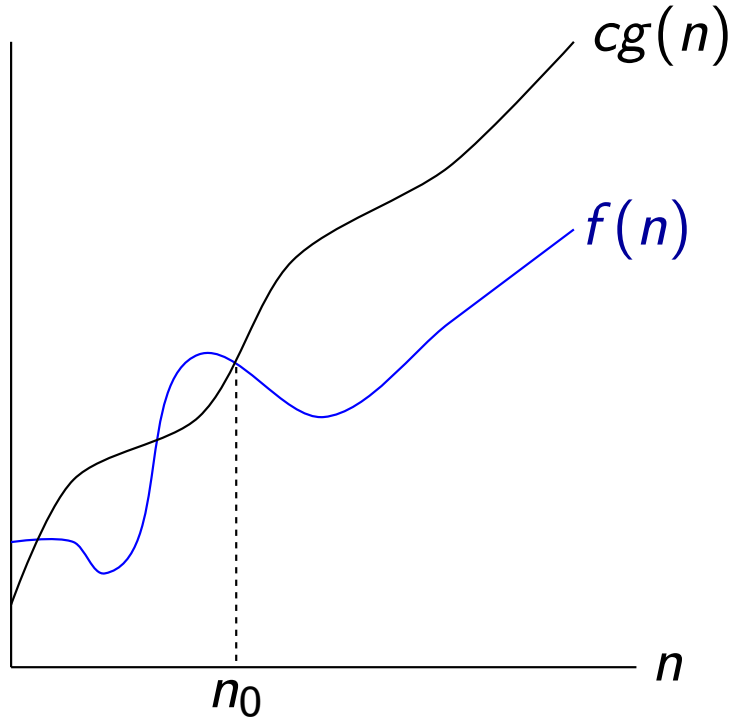
Nyní počítač 1000 násobně zrychlíme. Zvládne tedy 10^{15} kroků.

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.

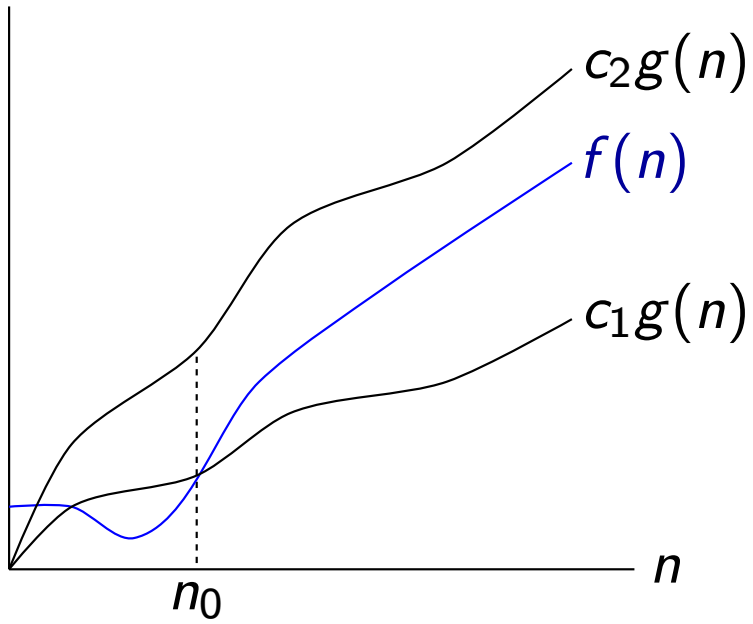
Asymptotická notace



Definice

Pro libovolné dvě funkce $f, g : \mathbb{N} \to \mathbb{N}$ 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



Definice

Pro libovolné dvě funkce $f, g : \mathbb{N} \to \mathbb{N}$ platí $f \in \Theta(g)$ právě tehdy, když $(\exists c_1 > 0)(\exists c_2 > 0)(\exists n_0 \geq 0)(\forall n \geq n_0) : 0 \leq c_1\, g(n) \leq f(n) \leq c_2\, g(n)$.

Asymptotická notace

Poznámka: Často se píše $f(n) = O(g(n))$ místo $f(n) \in O(g(n))$

Příklad: Při analýze složitosti vyhledávání čísla v posloupnosti jsme dostali $f(n) = 8n + 8$. Platí $f \in O(n)$. Takže můžeme říct, že složitost našeho algoritmu je v $O(n)$.

Příklady:

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

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

Složitost algoritmů

- Ne vždy je algoritmus s lepší asymptotickou složitostí výhodnější v praxi.
- Můžeme si to ilustrovat na algoritmech pro třídění.

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

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

Složitost algoritmů

- O algoritmu s časovou složitostí $t(n)$ řekneme, že je:

  logaritmický, pokud $t(n) \in \Theta(\log n)$
  lineární, pokud $t(n) \in \Theta(n)$
  kvadratický, pokud $t(n) \in \Theta(n^2)$
  kubický, pokud $t(n) \in \Theta(n^3)$
  polynomiální, pokud $t(n) \in \Theta(n^k)$ pro nějaké $k \in \mathbb{N}$
  exponenciální, pokud $t(n) \in \Theta(k^n)$ pro nějaké $k \in \mathbb{N}$

- Pro exponenciální složitost se používá také $2^{\Theta(n)}$, protože potom již nemusíme uvažovat různé základy mocniny.

- Při asymptotických odhadech není u logaritmů důležitý základ logaritmu, protože $\log_k n \in \Theta(\log_l n)$ pro všechna $k, l > 0$.

Složitost algoritmů

- Zatím jsme uvažovali, že provedení všech instrukcí trvá stejně dlouho.
- V praxi mohou být některé instrukce časově náročnější.
- Místo počtu instrukcí v nejhorším případě nás může zajímat počet provedení jedné konkrétní instrukce v nejhorším případě.
- Pokud známe časy provedení jednotlivých instrukcí a můžeme si spočítat počty jejich provedení, čas běhu potom dostaneme jako

$$\sum_i n_i t_i$$

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

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

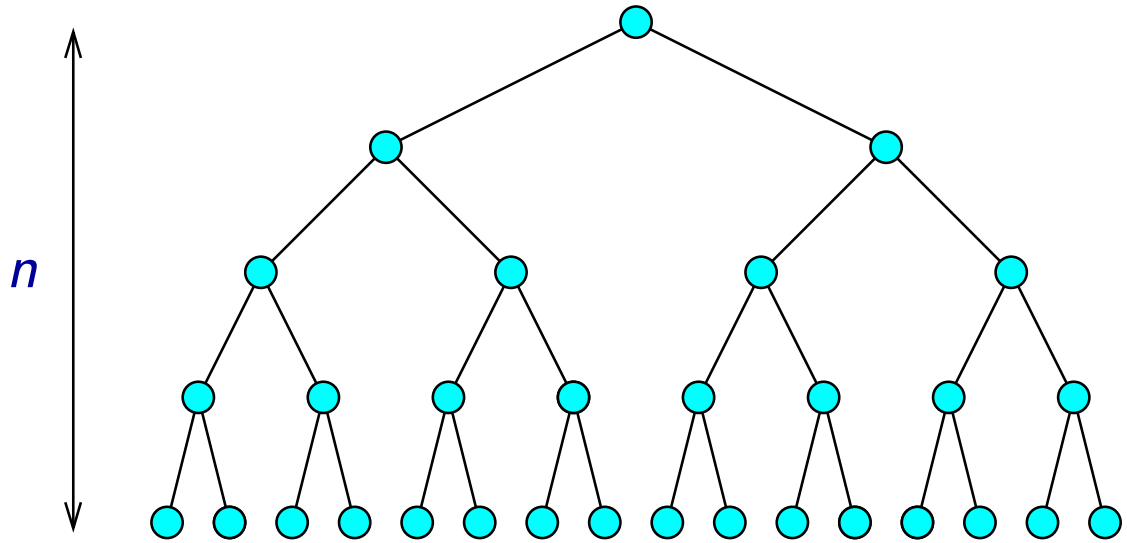
- Pro konkrétní problém můžeme mít dva algoritmy takové, že jeden má menší prostorovou složitost a druhý zase časovou složitost.
- Prostorová složitost je někdy kritičtější než časová. Když dojde paměť, ve výpočtu se nedá pokračovat.
- Je-li časová složitost algoritmu v $O(f(n))$ je i prostorová v $O(f(n))$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

Rekurentní vztahy

Někdy je vhodné vyjadřovat hodnotu funkce pomocí **rekurentních vztahů**.

Rekurentní vztah je rovnice nebo nerovnice, kde je hodnota funkce pro větší argumenty vyjádřena pomocí hodnot, kterých funkce nabývá pro menší argumenty.

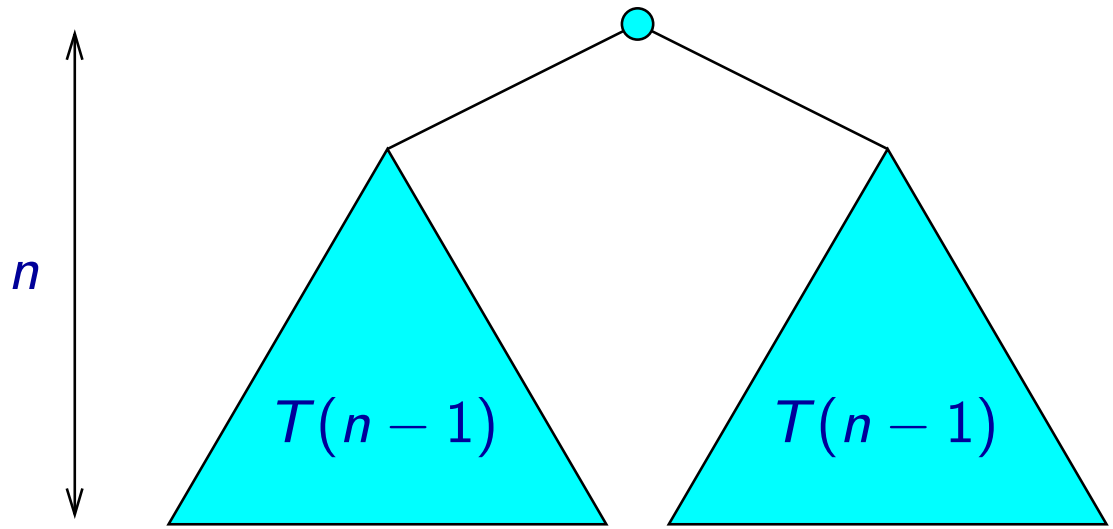
Příklad: Kolik vrcholů má úplný binární strom výšky n?



Rekurentní vztahy

$T(n)$ – počet vrcholů úplného binárního stromu výšky n

$$T(n) = \begin{cases} 1 & \text{pro } n = 0 \\ 2 \cdot T(n-1) + 1 & \text{pro } n > 0 \end{cases}$$



Rekurentní vztahy

$$T(n) = \begin{cases} 1 & \text{pro } n = 0 \\ 2 \cdot T(n-1) + 1 & \text{pro } n > 0 \end{cases}$$

$$T(0) = 1$$
$$T(1) = 2 \cdot T(0) + 1 = 2 \cdot 1 + 1 = 3$$
$$T(2) = 2 \cdot T(1) + 1 = 2 \cdot 3 + 1 = 7$$
$$T(3) = 2 \cdot T(2) + 1 = 2 \cdot 7 + 1 = 15$$
$$T(4) = 2 \cdot T(3) + 1 = 2 \cdot 15 + 1 = 31$$
$$T(5) = 2 \cdot T(4) + 1 = 2 \cdot 31 + 1 = 63$$
$$T(6) = 2 \cdot T(5) + 1 = 2 \cdot 63 + 1 = 127$$
$$T(7) = 2 \cdot T(6) + 1 = 2 \cdot 127 + 1 = 255$$
$$T(8) = 2 \cdot T(7) + 1 = 2 \cdot 255 + 1 = 511$$
$$T(9) = 2 \cdot T(8) + 1 = 2 \cdot 511 + 1 = 1023$$
$$T(10) = 2 \cdot T(9) + 1 = 2 \cdot 1023 + 1 = 2047$$
$$\ldots$$

Rekurentní vztahy

Pokud chceme vyjádřit hodnotu $T(n)$ nerekurentně, jednou z možností je použít tzv. **substituční metodu**:

1. Uhodnout správné řešení.
2. Ověřit jeho správnost pomocí matematické indukce.

Rekurentní vztahy

Příklad:

$$T(n) = \begin{cases} 1 & \text{pro } n = 0 \\ 2 \cdot T(n-1) + 1 & \text{pro } n > 0 \end{cases}$$

Uhodneme, že $T(n) = 2^{n+1} - 1$.

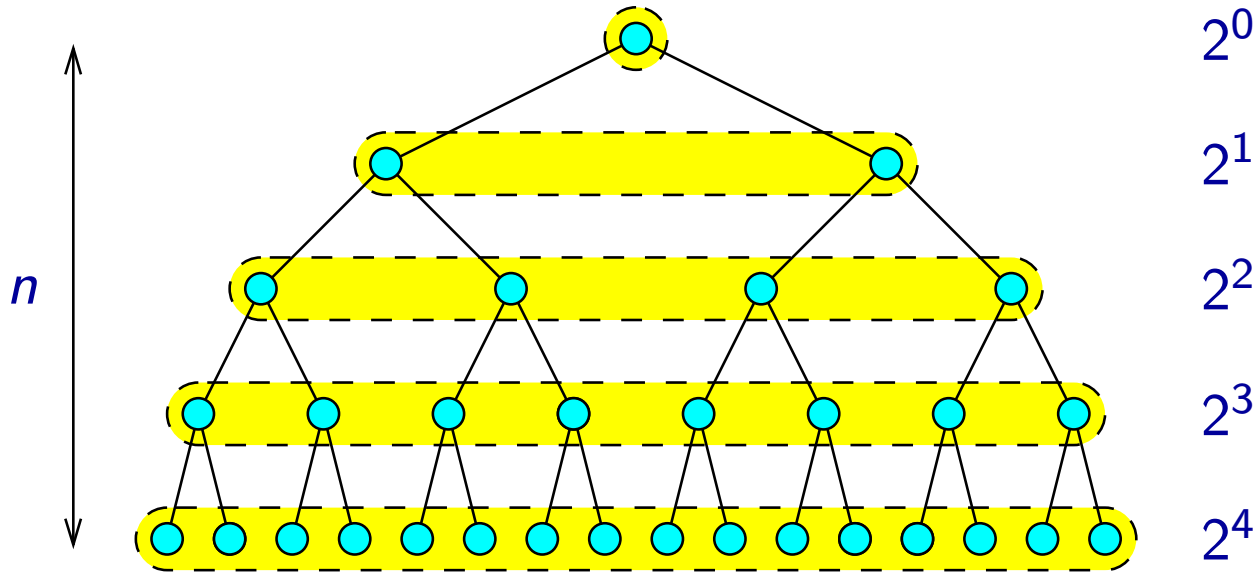
Indukcí ověříme, že tomu tak skutečně je:

- Báze ($i = 0$): $T(0) = 2^{0+1} - 1 = 1$
- Indukční krok ($i > 0$):

$$\begin{aligned} T(i) &= 2 \cdot T(i-1) + 1 \\ &= 2 \cdot (2^{(i-1)+1} - 1) + 1 \\ &= 2 \cdot 2^i - 2 + 1 \\ &= 2^{i+1} - 1 \end{aligned}$$

Rekurentní vztahy

Poznámka: Alternativní možností je spočítat počty vrcholů v jednotlivých „vrstvách“ stromu.



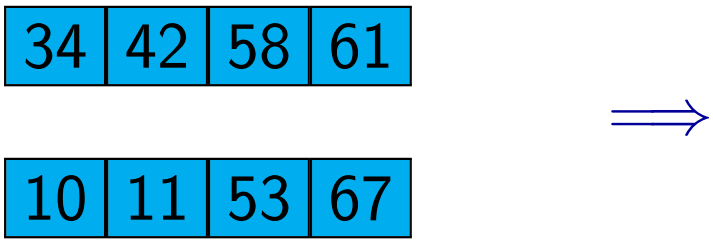
$$T(n) = \sum_{i=0}^{n} 2^i = \frac{2^{n+1} - 1}{2 - 1} = 2^{n+1} - 1$$

Rekurentní vztahy

Rekurentní vztahy se často používají při zkoumání časové složitosti rekurzivních algoritmů.

Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.

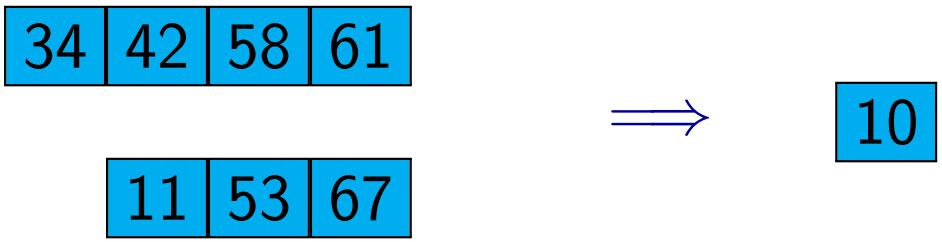


Rekurentní vztahy

Rekurentní vztahy se často používají při zkoumání časové složitosti rekurzivních algoritmů.

Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.

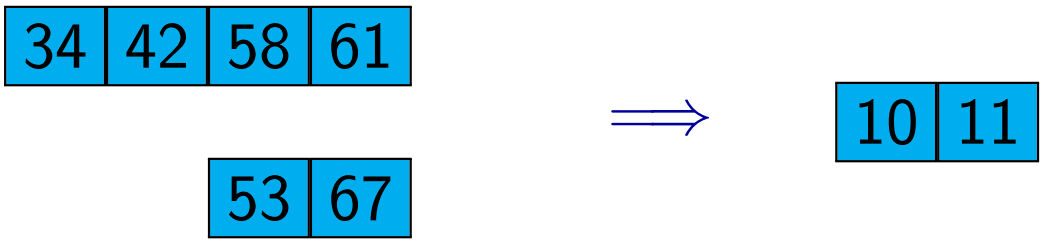


Rekurentní vztahy

Rekurentní vztahy se často používají při zkoumání časové složitosti rekurzivních algoritmů.

Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.

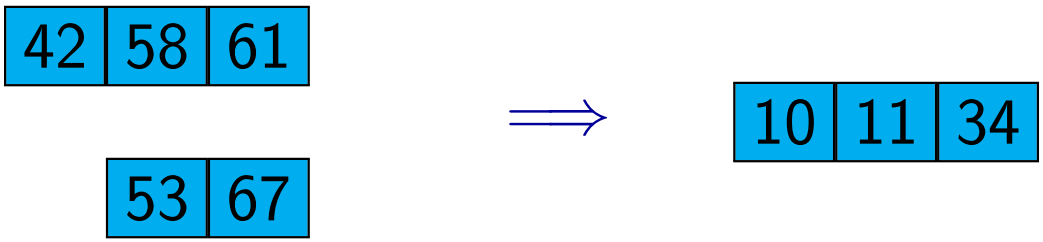


Rekurentní vztahy

Rekurentní vztahy se často používají při zkoumání časové složitosti rekurzivních algoritmů.

Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.

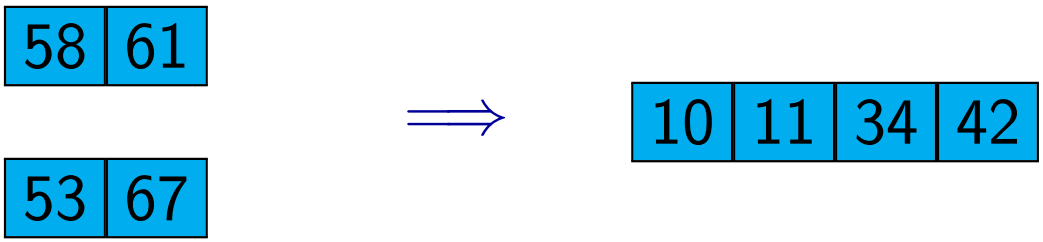


Rekurentní vztahy

Rekurentní vztahy se často používají při zkoumání časové složitosti rekurzivních algoritmů.

Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.

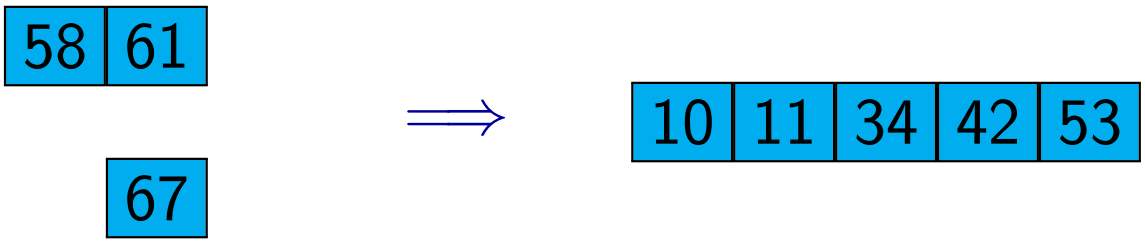


Rekurentní vztahy

Rekurentní vztahy se často používají při zkoumání časové složitosti rekurzivních algoritmů.

Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.

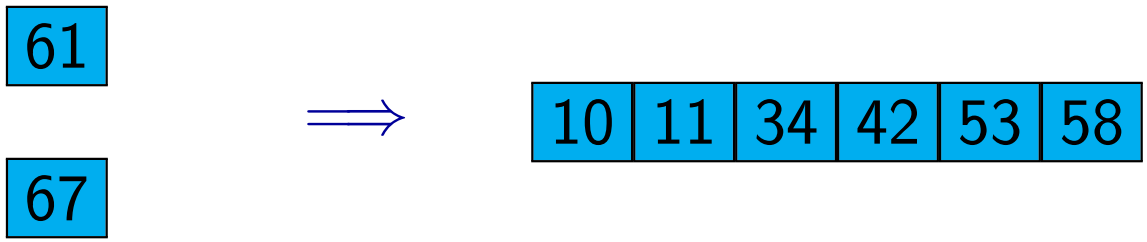


Rekurentní vztahy

Rekurentní vztahy se často používají při zkoumání časové složitosti rekurzivních algoritmů.

Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.

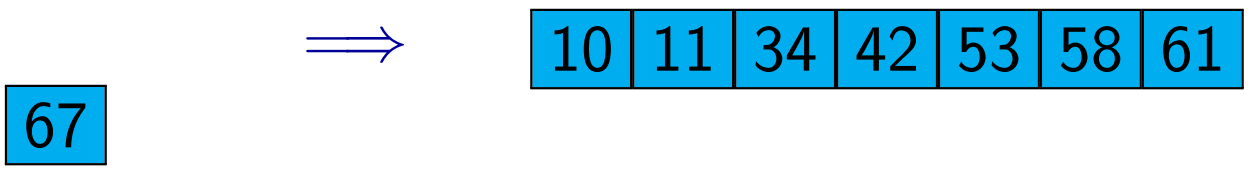


Rekurentní vztahy

Rekurentní vztahy se často používají při zkoumání časové složitosti rekurzivních algoritmů.

Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.



Rekurentní vztahy

Rekurentní vztahy se často používají při zkoumání časové složitosti rekurzivních algoritmů.

Příklad: Algoritmus MERGE-SORT.

Hlavní myšlenka algoritmu: Dvě setříděné posloupnosti snadno spojíme do jediné setříděné posloupnosti.
Pokud mají obě posloupnosti dohromady n prvků, vyžaduje tato operace n kroků.

Rekurentní vztahy

MERGE-SORT(A, p, r)

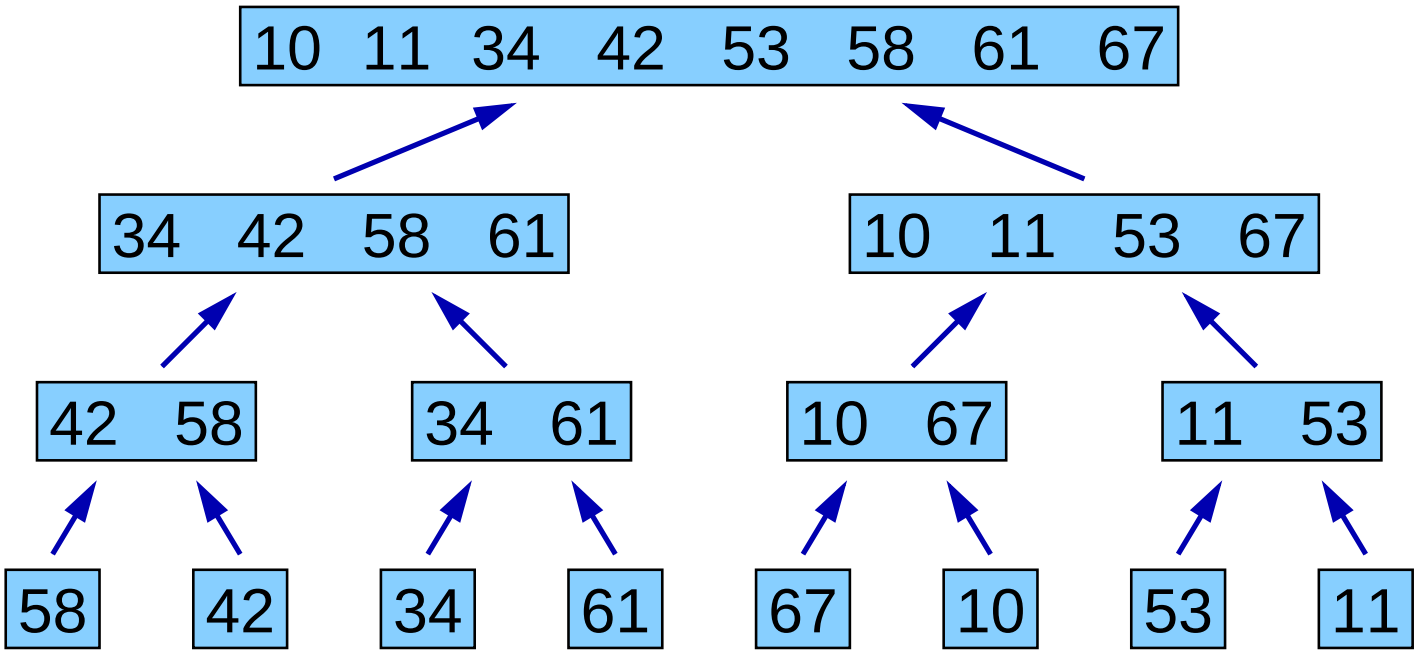
```
1  if r − p > 1
2      then q ← ⌊(p + r)/2⌋
3           MERGE-SORT(A, p, q)
4           MERGE-SORT(A, q, r)
5           MERGE(A, p, q, r)
```

Pro setřídění pole A, které obsahuje prvky $A[0], A[1], \cdots, A[n-1]$, zavoláme MERGE-SORT$(A, 0, n)$.

Poznámka: Procedura MERGE(A, p, q, r) spojí setříděné posloupnosti uložené v $A[p \,..\, q-1]$ a $A[q \,..\, r-1]$ do jedné posloupnosti uložené v $A[p \,..\, r-1]$.

Rekurentní vztahy

Vstup: 58, 42, 34, 61, 67, 10, 53, 11



Rekurentní vztahy

Dobu výpočtu $T(n)$ algoritmu MERGE-SORT pro vstup velikosti n můžeme vyjádřit jako:

$$T(n) = \begin{cases} \Theta(1) & \text{pro } n = 1 \\ 2T(n/2) + \Theta(n) & \text{pro } n > 1 \end{cases}$$

Poznámka: $\Theta(1)$ označuje množinu všech funkcí jejichž hodnota je omezena konstantou.

Poznámka: Přesněji můžeme $T(n)$ vyjádřit jako

$$T(n) = \begin{cases} \Theta(1) & \text{pro } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + \Theta(n) & \text{pro } n > 1 \end{cases}$$

Rekurentní vztahy

Pokud bychom znali hodnoty všech konstant a přesné hodnoty funkcí, mohli bychom spočítat přesnou hodnotu $T(n)$ pro libovolné n.

Příklad:

$$T(n) = \begin{cases} 4 & \text{pro } n = 1 \\ T(\lceil n/2 \rceil) + T(\lfloor n/2 \rfloor) + 5n + 11 & \text{pro } n > 1 \end{cases}$$

$$\begin{aligned}
T(1) &= 4 \\
T(2) &= T(\lceil 2/2 \rceil) + T(\lfloor 2/2 \rfloor) + 5 \cdot 2 + 11 = T(1) + T(1) + 21 = 29 \\
T(3) &= T(\lceil 3/2 \rceil) + T(\lfloor 3/2 \rfloor) + 5 \cdot 3 + 11 = T(2) + T(1) + 26 = 59 \\
T(4) &= T(\lceil 4/2 \rceil) + T(\lfloor 4/2 \rfloor) + 5 \cdot 4 + 11 = T(2) + T(2) + 31 = 89 \\
T(5) &= T(\lceil 5/2 \rceil) + T(\lfloor 5/2 \rfloor) + 5 \cdot 5 + 11 = T(3) + T(2) + 36 = 124 \\
T(6) &= T(\lceil 6/2 \rceil) + T(\lfloor 6/2 \rfloor) + 5 \cdot 6 + 11 = T(3) + T(3) + 41 = 159 \\
T(7) &= T(\lceil 7/2 \rceil) + T(\lfloor 7/2 \rfloor) + 5 \cdot 7 + 11 = T(4) + T(3) + 46 = 194 \\
&\ldots
\end{aligned}$$

Rekurentní vztahy

- Přesné hodnoty funkcí většinou neznáme, místo nich známe jen asymptotické odhady.
- V tom případě bude výsledkem také jen asymptotický odhad.
- I když všechny přesné hodnoty známe, může nám stačit jen asymptotický odhad, pokud by přesné odvození bylo příliš komplikované.

Rekurentní vztahy

Příklad: $T(n) = \begin{cases} \Theta(1) & \text{pro } n = 1 \\ 2T(\lfloor n/2 \rfloor) + n & \text{pro } n > 1 \end{cases}$

Uhodneme, že řešením je $T(n) = O(n \log n)$, a dokážeme, že $T(n) \leq cn \lg n$ pro nějakou vhodně zvolenou konstantu c:

$$\begin{aligned} T(n) &\leq 2(c\lfloor n/2 \rfloor \lg(\lfloor n/2 \rfloor)) + n \\ &\leq cn \lg(n/2) + n \\ &= cn \lg n - cn \lg 2 + n \\ &= cn \lg n - cn + n \\ &\leq cn \lg n \end{aligned}$$

kde poslední nerovnost platí pro libovolné $c \geq 1$.

Poznámka: $\lg n$ označuje $\log_2 n$.

Rekurentní vztahy

Zbývá ověřit, že platí báze indukce.

Předpokládejme na chvíli pro jednoduchost, že $T(1) = 1$.

Všimněte si, že vztah $T(n) \leq cn \lg n$ neplatí pro $n = 1$, ať už zvolíme jakoukoliv hodnotu $c \geq 1$ (protože $\lg 1 = 0$), a báze indukce tedy neplatí!

Řešení: Pokud chceme odvodit pouze asymptotický odhad, stačí ukázat, že $T(n) \leq cn \lg n$ platí pro $n \geq n_0$, kde n_0 je nějaká vhodně zvolená konstanta.

Zvolíme-li například $n_0 = 2$, stačí ukázat, že pro nějaké c platí pro libovolné $n \geq 2$ vztah $T(n) \leq cn \lg n$:

- Báze: $T(2) \leq c2 \lg 2$, $T(3) \leq c3 \lg 3$
  Platí pro $c = 2$, neboť $T(2) = 4$ a $T(3) = 5$
- Indukční krok: pro $n \geq 4$ hodnota $T(n)$ nezávisí (přímo) na $T(1)$.