

# Vyhledávání, řazení, složitost

IB111 Programování a algoritmizace

2011

# Otrávené studny

- 8 studen, jedna z nich je otrávená
- laboratorní rozbor
  - dokáže rozpoznat přítomnost jedu ve vodě
  - je drahý
  - (je časově náročný)
- kolik rozborů (času) potřebujeme?
- jak určit otrávenou studnu?

# Otrávené studny: řešení

studna	kód	studna	kód
A	000	E	100
B	001	F	101
C	010	G	110
D	011	H	111

test	přidělené studny
1	B, D, F, H
2	C, D, G, H
3	E, F, G, H

# Vyhledávání: hra

- Myslím si přirozené číslo  $X$  mezi 1 a 1000.
- Povolená otázka: „Je  $X$  menší než  $N$ ?“
- Kolik otázek potřebujete na odhalení čísla?
- Kolik *předem formulovaných* otázek potřebujete?
- Mezi kolika čísla jste schopni odhalit skryté číslo na  $K$  otázek?

# Vyhledávání: řešení

- „dynamické otázky“: půlení intervalu
- „předem formulované otázky“: dotazy na bity v bitovém zápisu (stejně jako u studen)
- $N$  čísel: potřebujeme  $\log_2 N$  otázek
- $K$  otázek: rozlišíme mezi  $2^K$  čísla

# Vyhledávání: problém

mnoho různých variant, základní verze:

- vstup: seřazená posloupnost čísel + dotaz (číslo)  
např. 2, 3, 7, 8, 9, 14 + dotaz 8
- výstup: index hledaného čísla v posloupnosti (případně -1  
pokud tam není)  
např. 3

# Vyhledávání a logaritmus

- naivní metoda = průchod seznamu
  - **lineární** vyhledávání,  $O(n)$
  - pomalé (viz např. databáze s milióny záznamů)
  - jen velmi krátké seznamy
- základní rozumná metoda = půlení intervalu
  - analogicky hře s hádáním čísel
  - podívám se na prostřední člen
  - podle jeho hodnoty pokračuji v levém/pravém intervalu
  - **logaritmický** počet kroků (vzhledem k délce seznamu),  $O(\log(n))$

# Vyhledávání půlením intervalu (rekurzivně)

## Algorithm 6.2 BINARYSEARCHREC

**Input:** An array  $A[1..n]$  of  $n$  elements sorted in nondecreasing order and an element  $x$ .

**Output:**  $j$  if  $x = A[j]$ ,  $1 \leq j \leq n$ , and 0 otherwise.

1.  $\text{binarysearch}(1, n)$

### Procedure $\text{binarysearch}(low, high)$

1. if  $low > high$  then return 0
2. else
3.      $mid \leftarrow \lfloor (low + high)/2 \rfloor$
4.     if  $x = A[mid]$  then return  $mid$
5.     else if  $x < A[mid]$  then return  $\text{binarysearch}(low, mid - 1)$
6.     else return  $\text{binarysearch}(mid + 1, high)$
7. end if

M. H. Alsuwaiyel: Algorithms, Design Techniques and Analysis.



# Vyhledávání půlením intervalu (iterativně)

## Algorithm 1.2 BINARYSEARCH

**Input:** An array  $A[1..n]$  of  $n$  elements sorted in nondecreasing order and an element  $x$ .

**Output:**  $j$  if  $x = A[j]$ ,  $1 \leq j \leq n$ , and 0 otherwise.

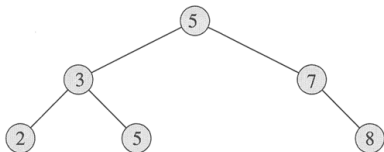
1.  $low \leftarrow 1$ ;  $high \leftarrow n$ ;  $j \leftarrow 0$
2. **while** ( $low \leq high$ ) **and** ( $j = 0$ )
3.      $mid \leftarrow \lfloor (low + high)/2 \rfloor$
4.     **if**  $x = A[mid]$  **then**  $j \leftarrow mid$
5.     **else if**  $x < A[mid]$  **then**  $high \leftarrow mid - 1$
6.     **else**  $low \leftarrow mid + 1$
7. **end while**
8. **return**  $j$

M. H. Alsuwaiyel: Algorithms, Design Techniques and Analysis.

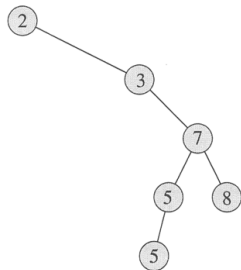
# Upozornění k uvedeným algoritmům

- předchozí (a následující) pseudokódy nejsou přímo v Pythonu
- hlavní rozdíl: indexování od 0 vs od 1
- záměr:
  - abyste trénovali čtení různých notací
  - budeme dělat na cvičení, tak abyste nemohli jen tak přepsat kód ze slidů

# Vyhledávací stromy



(a)



(b)

T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms.

(více později)

# Řadicí algoritmy: terminologická poznámka

- anglicky „sorting algorithm“
- česky používáno: řadicí algoritmy nebo třídící algoritmy
- řadicí obecně (ale nikoliv jednoznačně) považováno za „správnější“

# Řadící algoritmy: komentář

- mnoho různých algoritmů pro stejný účel
- většina programovacích jazyků má vestavěnou podporu (funkce `sort()`)

*Proč se tím tedy zabýváme?*

*Proč se tím tedy zabýváme?*

- 1 ilustrace algoritmického myšlení, technik návrhu algoritmů
- 2 ilustrace drobné změny algoritmu s velkým dopadem na rychlost programu
- 3 tradice, hezky se to vizualizuje a vysvětluje

# Doporučený zdroj

<http://www.sorting-algorithms.com/>

- animace
- kódy
- vizualizace

Více podobných: Google → sorting algorithms

# Řadicí algoritmy: problém

- vstup: posloupnost (přirozených) čísel  
např. 8, 2, 14, 3, 7, 9
- výstup: seřazená posloupnost  
např. 2, 3, 7, 8, 9, 14



# Co vy na to?

- zkuste vymyslet
  - řadicí algoritmus
  - co nejvíce různých principů
  - co nejefektivnější algoritmus
- možná inspirace: jak řadíte karty?

$n$  – délka vstupní posloupnosti

	počet operací
jednoduché algoritmy	$O(n^2)$ , „kvadratická“
složitější algoritmy	$O(n \log(n))$

# Bublňkové řazení (Bubble sort)

- „proublávání“ vyšších hodnot nahoru
- srovnávání a prohazování sousedů

```
for i in range(n):  
    print a  
    for j in range(n-i-1):  
        if a[j] > a[j+1]:  
            tmp = a[j]  
            a[j] = a[j+1]  
            a[j+1] = tmp
```

invariant cyklu:  $a[n-i-1..n-1]$  ve finální pozici

# Bublinkové řazení: příklad běhu

[8, 2, 7, 14, 3, 1]

[2, 7, 8, 3, 1, 14]

[2, 7, 3, 1, 8, 14]

[2, 3, 1, 7, 8, 14]

[2, 1, 3, 7, 8, 14]

[1, 2, 3, 7, 8, 14]

# Řazení výběrem (Select sort)

- řazení výběrem
- projdeme dosud neseřazenou část pole a vybereme nejmenší prvek
- nejmenší prvek zařadíme na aktuální pozici (výměnou)

# Řazení výběrem (Select sort)

**Algorithm 1.4** SELECTIONSORT

**Input:** An array  $A[1..n]$  of  $n$  elements.

**Output:**  $A[1..n]$  sorted in nondecreasing order.

1. **for**  $i \leftarrow 1$  **to**  $n - 1$
2.      $k \leftarrow i$
3.     **for**  $j \leftarrow i + 1$  **to**  $n$      {Find the  $i$ th smallest element.}
4.         **if**  $A[j] < A[k]$  **then**  $k \leftarrow j$
5.     **end for**
6.     **if**  $k \neq i$  **then** interchange  $A[i]$  and  $A[k]$
7. **end for**

M. H. Alsuwaiyel: Algorithms, Design Techniques and Analysis.

# Řazení vkládáním (Insert sort)

- podobně jako „řazení karet“
- prefix pole udržujeme seřazený
- každou další hodnotu zařadíme tam, kam patří

# Řazení vkládáním (Insert sort)

**Algorithm 1.5** INSERTIONSORT

**Input:** An array  $A[1..n]$  of  $n$  elements.

**Output:**  $A[1..n]$  sorted in nondecreasing order.

1. **for**  $i \leftarrow 2$  **to**  $n$
2.      $x \leftarrow A[i]$
3.      $j \leftarrow i - 1$
4.     **while**  $(j > 0)$  **and**  $(A[j] > x)$
5.          $A[j + 1] \leftarrow A[j]$
6.          $j \leftarrow j - 1$
7.     **end while**
8.      $A[j + 1] \leftarrow x$
9. **end for**

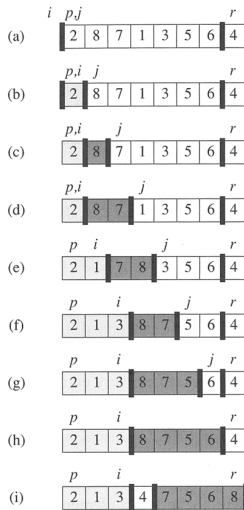
M. H. Alsuwaiyel: Algorithms, Design Techniques and Analysis.



# Quicksort

- rekurzivní algoritmus
- vybereme „pivota“ a pole rozdělíme na dvě části:
  - větší než pivot
  - menší než pivot
- obě části pak nezávisle seřadíme (rekurzivně pomocí quicksortu)

# Quick sort ilustrace



# Quick sort

- pokud máme smůlu při výběru pivota, tak je stejně pomalý jako předchozí
- v průměrném případě je rychlý – *quick*  
 $O(n \log(n))$

# Řazení slučováním (Merge sort)

- rekurzivní algoritmus
- pole rozdělíme na dvě poloviny a ty seřadíme (pomocí Merge sort)
- ze seřazených polovin vyrobíme jedno seřazené pole – „zipování“

# Merge sort

**Algorithm 6.3** MERGESORT

**Input:** An array  $A[1..n]$  of  $n$  elements.

**Output:**  $A[1..n]$  sorted in nondecreasing order.

1.  $\text{mergesort}(A, 1, n)$

**Procedure**  $\text{mergesort}(low, high)$

1. **if**  $low < high$  **then**
2.      $mid \leftarrow \lfloor (low + high)/2 \rfloor$
3.      $\text{mergesort}(A, low, mid)$
4.      $\text{mergesort}(A, mid + 1, high)$
5.     MERGE ( $A, low, mid, high$ )
6. **end if**

M. H. Alsuwaiyel: Algorithms, Design Techniques and Analysis.

# Operace Merge

## Algorithm 1.3 MERGE

**Input:** An array  $A[1..m]$  of elements and three indices  $p, q$  and  $r$ , with  $1 \leq p \leq q < r \leq m$ , such that both the subarrays  $A[p..q]$  and  $A[q + 1..r]$  are sorted individually in nondecreasing order.

**Output:**  $A[p..r]$  contains the result of merging the two subarrays  $A[p..q]$  and  $A[q + 1..r]$ .

1. **comment:**  $B[p..r]$  is an auxiliary array.
2.  $s \leftarrow p$ ;  $t \leftarrow q + 1$ ;  $k \leftarrow p$
3. **while**  $s \leq q$  **and**  $t \leq r$
4.     **if**  $A[s] \leq A[t]$  **then**
5.          $B[k] \leftarrow A[s]$
6.          $s \leftarrow s + 1$
7.     **else**
8.          $B[k] \leftarrow A[t]$
9.          $t \leftarrow t + 1$
10.     **end if**
11.      $k \leftarrow k + 1$
12. **end while**
13. **if**  $s = q + 1$  **then**  $B[k..r] \leftarrow A[t..r]$
14. **else**  $B[k..r] \leftarrow A[s..q]$
15. **end if**
16.  $A[p..r] \leftarrow B[p..r]$

# Radix sort

- předchozí algoritmy využívají pouze operaci porovnání dvou hodnot
- aplikovatelné na cokoliv, co lze porovnávat, žádné další předpoklady
- s doplňujícími předpoklady můžeme dostat nové algoritmy (obecný princip)

# Radix sort

- aplikovatelné na (krátká) čísla
- postupujeme od nejméně významné cifry k nejvýznamnější
- seřadíme pole podle dané cifry = rozdělení do 10 „kyblíčků“ (jednoduché, rychlé)



# Radix sort ilustrace

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms.

# Složitost trochu podrobněji

- složitost algoritmu – jak je algoritmus výpočetně náročný
- časová, prostorová
- měříme počet **operací** nikoliv čas na konkrétním stroji
- vyjadřujeme jako funkci délky vstupu
- $O$  notace – zanedbáváme konstanty
- např.  $O(n)$ ,  $O(n \log(n))$ ,  $O(n^2)$

# Ilustrace rozdílů v složitosti

$n$	$\log n$	$n$	$n \log n$	$n^2$	$n^3$	$2^n$
8	3 nsec	0.01 $\mu$	0.02 $\mu$	0.06 $\mu$	0.51 $\mu$	0.26 $\mu$
16	4 nsec	0.02 $\mu$	0.06 $\mu$	0.26 $\mu$	4.10 $\mu$	65.5 $\mu$
32	5 nsec	0.03 $\mu$	0.16 $\mu$	1.02 $\mu$	32.7 $\mu$	4.29 sec
64	6 nsec	0.06 $\mu$	0.38 $\mu$	4.10 $\mu$	262 $\mu$	5.85 cent
128	0.01 $\mu$	0.13 $\mu$	0.90 $\mu$	16.38 $\mu$	0.01 sec	$10^{20}$ cent
256	0.01 $\mu$	0.26 $\mu$	2.05 $\mu$	65.54 $\mu$	0.02 sec	$10^{58}$ cent
512	0.01 $\mu$	0.51 $\mu$	4.61 $\mu$	262.14 $\mu$	0.13 sec	$10^{135}$ cent
2048	0.01 $\mu$	2.05 $\mu$	22.53 $\mu$	0.01 sec	1.07 sec	$10^{598}$ cent
4096	0.01 $\mu$	4.10 $\mu$	49.15 $\mu$	0.02 sec	8.40 sec	$10^{1214}$ cent
8192	0.01 $\mu$	8.19 $\mu$	106.50 $\mu$	0.07 sec	1.15 min	$10^{2447}$ cent
16384	0.01 $\mu$	16.38 $\mu$	229.38 $\mu$	0.27 sec	1.22 hrs	$10^{4913}$ cent
32768	0.02 $\mu$	32.77 $\mu$	491.52 $\mu$	1.07 sec	9.77 hrs	$10^{9845}$ cent
65536	0.02 $\mu$	65.54 $\mu$	1048.6 $\mu$	0.07 min	3.3 days	$10^{19709}$ cent
131072	0.02 $\mu$	131.07 $\mu$	2228.2 $\mu$	0.29 min	26 days	$10^{39438}$ cent
262144	0.02 $\mu$	262.14 $\mu$	4718.6 $\mu$	1.15 min	7 mnths	$10^{78894}$ cent
524288	0.02 $\mu$	524.29 $\mu$	9961.5 $\mu$	4.58 min	4.6 years	$10^{157808}$ cent
1048576	0.02 $\mu$	1048.60 $\mu$	20972 $\mu$	18.3 min	37 years	$10^{315634}$ cent

Table 1.1 Running times for different sizes of input. “nsec” stands for nanoseconds, “ $\mu$ ” is one microsecond and “cent” stands for centuries.

# Složitost řadících algoritmů

$n$  – délka vstupní posloupnosti

	počet operací
jednoduché algoritmy	$O(n^2)$ , „kvadratická“
složitější algoritmy	$O(n \log(n))$

- vyhledávání: půlení intervalu, rekurze
- řadící algoritmy:
  - jednoduché (kvadratické): bubble, selection, insertion
  - složitější ( $n \log n$ , rekurzivní): quick sort, merge sort