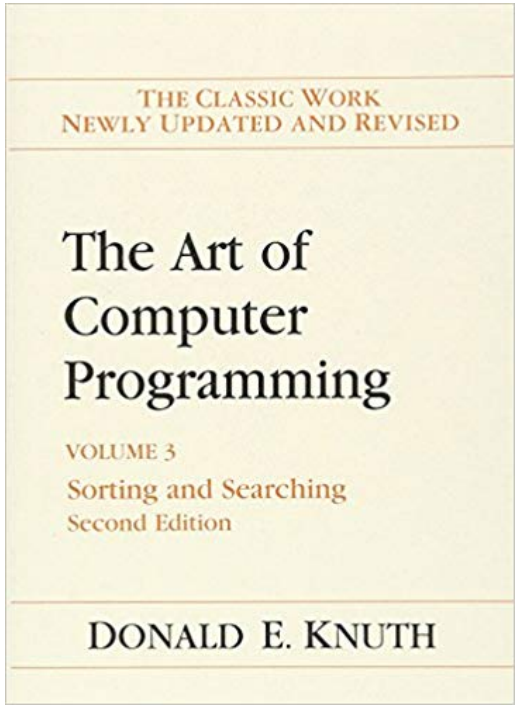
Algoritmizace

Vyhledávání



Osnova

- Vyhledávání v poli
  - sekvenční průchod
  - binární vyhledávání

Datová struktura pole (array)

Posloupnost položek stejného typu, které jsou uloženy za sebou v souvislém bloku paměti

Přístup k položkám v čase O(1) pomocí indexu

- identifikuje pořadí prvku
- nejčastěji přirozené číslo

`prvocisla`

0	1	2	3	4	5	6	7
2	3	5	7	11	13	17	19

pevná délka (např. 4B)

Vestavěný datový typ

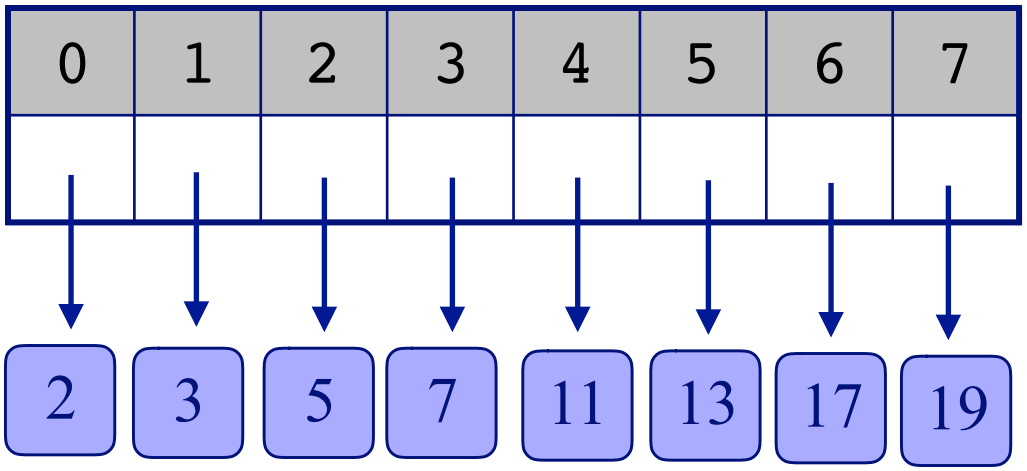
- ve většině programovacích jazyků
- C, C++, C#, Java, …

Datová struktura seznam (list)

Seznam (list) v jazyce Python

- posloupnost prvků libovolného typu
- "heterogenní pole"
- indexovaná 0, 1, 2, …
- přístup k prvku v čase O(1)

`prvocisla`



Vyhledávání v poli

Vstup: pole `a` indexované od 0, prvek `x`

Výstup: pokud se `x` v `a` vyskytuje, index 1. výskytu
`False` jinak

Poznámka: V jazyce Python k dispozici

- operátor `in`
- metody `index()` a `count()`

} čas $\Theta(n)$ v nejhorším případě

```
>>> a = [10, 20, 30, 20]
>>> 20 in a
True
>>> a.index(20)
1
>>> a.count(20)
2
```

Vyhledávání v poli – sekvenční průchod

```
def hledej(a,x):
    for i in range(len(a)):
        if a[i] == x:
            return i
    return False
```

Čas $\Theta(n)$ v nejhorším případě

Vyhledávání v poli – se zarážkou

```
def hledej1(a,x):
    a.append(x)  # vložení zarážky
                 # na konec seznamu
    n = len(a)-1 # index zarážky

    i = 0
    while a[i] != x:
        i += 1
    del a[n]    # odstranění zarážky

    if i == n:
        return False
    else:
        return i
```

zarážka zajistí
že je x vždy nalezeno

Vyhledávání v uspořádaném poli

Položky pole `a` jsou uspořádané

- `a[i] ≤ a[i+1] for i in range(n)`
- kde `n = len(a)-1`

Binární vyhledávání (půlení intervalu)

- je-li `a[n // 2] == x`, jsme hotovi
- je-li `a[n // 2] > x`, prohledáme
  `a[0],a[1],…,a[n // 2 - 1]`
- je-li `a[n // 2] < x`, prohledáme
  `a[n // 2 +1],a[n // 2 +2],…,a[n]`

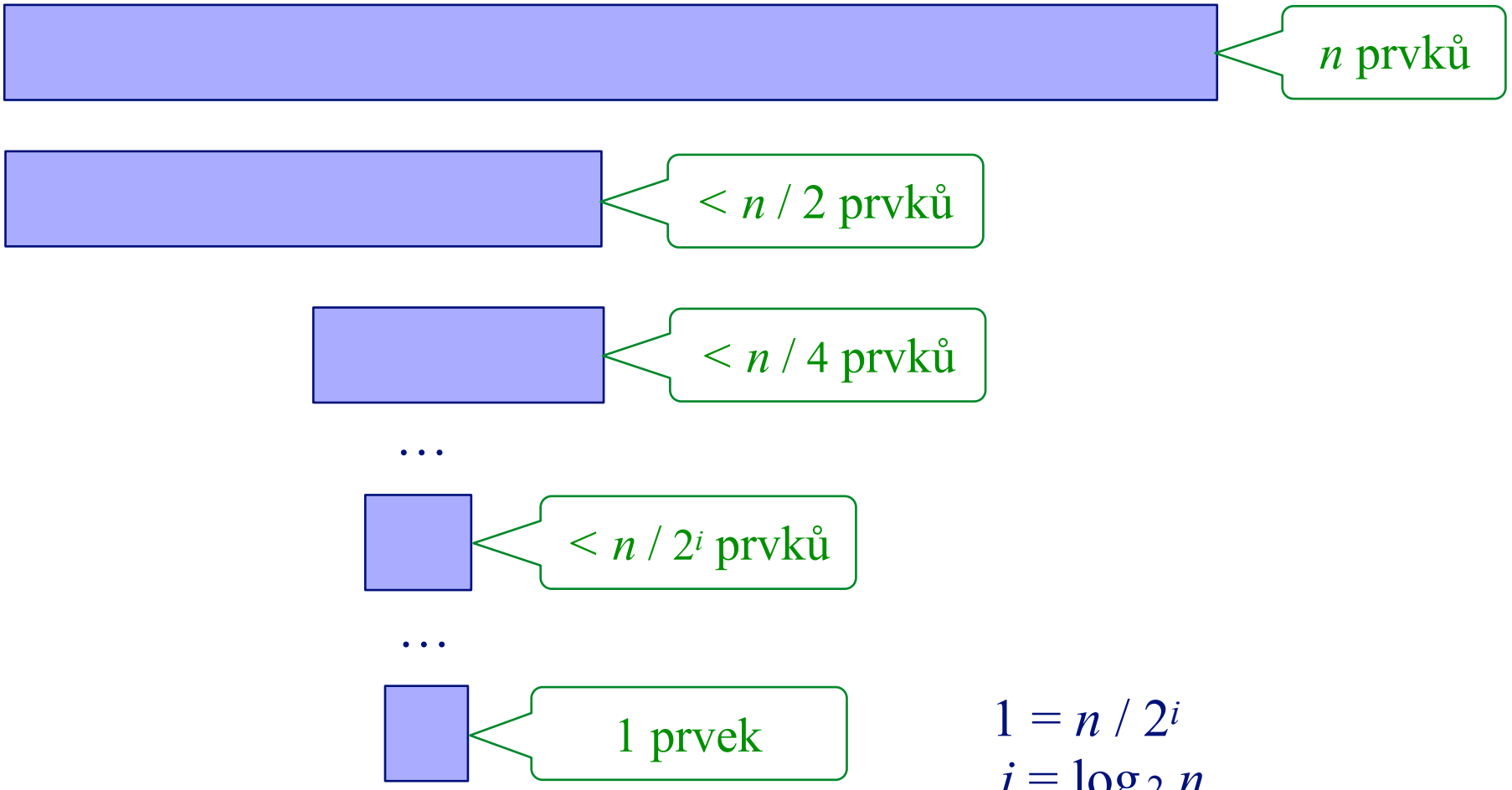
Binární vyhledávání

seznam

```
def binSearch(a,x):
    dolni, horni = -1, len(a)
    stred = (dolni + horni) // 2
    while horni - dolni > 1:
        if a[stred] == x:
            return stred
        elif a[stred] < x:
            dolni = stred
        else:
            horni = stred
        stred = (dolni + horni) // 2
    return False
```

Invariant cyklu: `a[i] != x` pro `i ≤ dolni` a `i ≥ horni`

Binární vyhledávání – složitost



$1 = n / 2^i$

$i = \log_2 n$

Časová složitost $\Theta(\log n)$

Výpočet $\sqrt{n}$ půlením intervalu

n > 1

```
def odmocnina(n):

    eps = 0.01
    dolni,horni = 1.0, n
    stred = (dolni + horni)/2.0
    mocnina = stred**2

    while abs(mocnina - n) >= eps:
        if mocnina < n:
            dolni = stred
        else:
            horni = stred
        stred = (dolni + horni)/2.0
        mocnina = stred**2

    return stred
```

Problémy ✎

① Zobecněte funkci odmocnina(n) tak, aby fungovala pro libovolné kladné číslo n.