VŠB TECHNICKÁ UNIVERZITA OSTRAVA | HORNICKO GEOLOGICKÁ FAKULTA | KATEDRA GEOINFORMATIKY

Programování v GIS 1
7 - Algoritmy pro třídění a vyhledávání

Michal Kačmařík

A924, tel.: 5512

e-mail: michal.kacmarik@vsb.cz

https://www.hgf.vsb.cz/548/cs/

Základní úlohy

Velmi často využívanými algoritmy jsou algoritmy pro úlohy

- třídění,
- vyhledávání,
- indexace.

Složitost algoritmů

- Stejný problém může být běžně řešen různými přístupy = různými algoritmy
- Rychlost provedení programu záleží mimo jiné na použitém algoritmu (počet provedených operací, typ operace – například operace přiřazení či sčítání/odčítání je rychlejší než operace násobení/dělení a ty jsou zase rychlejší než operace s (od)mocninami)

Složitost algoritmů

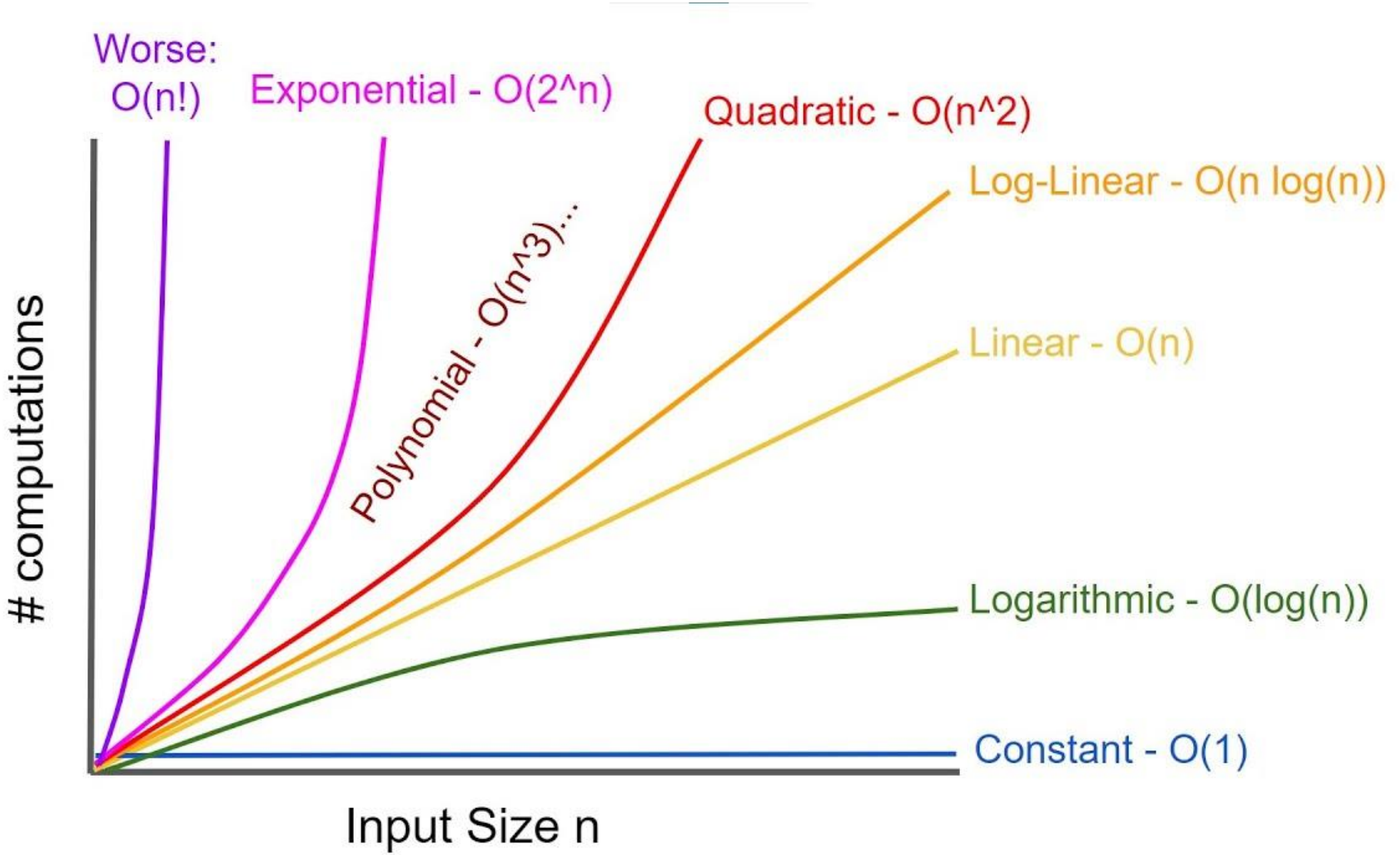
- Složitost algoritmu udává, **jak je daný algoritmus rychlý** (kolik provede elementárních operací) v závislosti na množství vstupních dat.
- Ke klasifikaci algoritmů se obvykle používá tzv. **časová (asymptotická) složitost**,
- což je rozdělení algoritmů do tříd složitostí, u kterých platí, že od určité velikosti dat (*N*), je algoritmus dané třídy vždy pomalejší než algoritmus třídy předchozí, bez ohledu na to, jestli je některý z počítačů c-násobně výkonnější (c je konstanta).

Složitost algoritmů

Složitost	Vyjádření	Úlohy
Konstantní	1	indexace prvků v poli, nezávisí na vstupních datech
Logaritmická	log(N)	prohledávání binárního stromu, doba běhu se mírně zvyšuje v závislosti na N.
Lineární	N	sekvenční vyhledávání, doba běhu roste lineárně s N
Lineárnělogaritmická	N*log(N)	třídění s použitím Rozděl a panuj, doba běhu roste téměř lineárně s N
Kvadratická	N^2	vnořený cyklus, zpracování maticových dat
Kubická	N^3	vnořený cyklus ve vnořeném cyklu
Exponenciální	2^N	problém obchodního cestujícího, exponenciální doba běhu

$$1 \ll \log(n) \ll n \ll n \cdot \log(n) \ll n^k \ll k^n \ll n! \ll n^n$$

Složitost algoritmů



Třídění

- Třídící (řadící) algoritmy slouží k setřízení prvků ze vstupního souboru podle určitého kritéria (například dle velikosti) vzestupně nebo sestupně
- důvod třídění – uspořádání dat, efektivnější vyhledávání
- kritéria výběru vhodného algoritmu jsou dána vlastnostmi algoritmu:
  - složitost algoritmu (paměťová, časová),
  - charakter vstupních dat,
  - stabilita algoritmu,
  - chování na částečně seřazených datech.

Třídění – vstupní data

- Vstupní data (jejich velikost) – ovlivňují volbu metody
- *Vnitřní algoritmy* – data jsou uložena v operační paměti

  InsertSort, BubbleSort, SelectSort – $O(N^2)$
- *Vnější algoritmy* – rozsáhlá data se načítají z disku

  QuickSort, MergeSort – O(N log N)

Třídění – stabilita algoritmu

- Třídící algoritmy pracují se strukturovanými prvky, porovnávacím parametrem třídění je ***klíč***.
- *Stabilní algoritmy* – uchovávají pořadí prvků podle klíče
- *Nestabilní algoritmy* – pořadí se během zpracování může měnit
- Snahou je volit strukturu dat tak, aby bylo možné použít stabilní algoritmy.

Třídění – stabilita algoritmu

- Stabilita algoritmu je užitečná při třídění podle dvou (či více) parametrů.
- Řadíme-li například osoby napřed dle křestního jména a poté dle příjmení, pak výsledek stabilního algoritmu odpovídá očekávání (první je Karel Novák, následuje Václav Novák).
- Pokud by algoritmus nebyl stabilní, tak tento postup nebude fungovat, protože druhé řazení by mohlo zpřeházet výsledek prvního (Václav Novák by mohl být před Karlem Novákem).

Třídění – seřazení dat

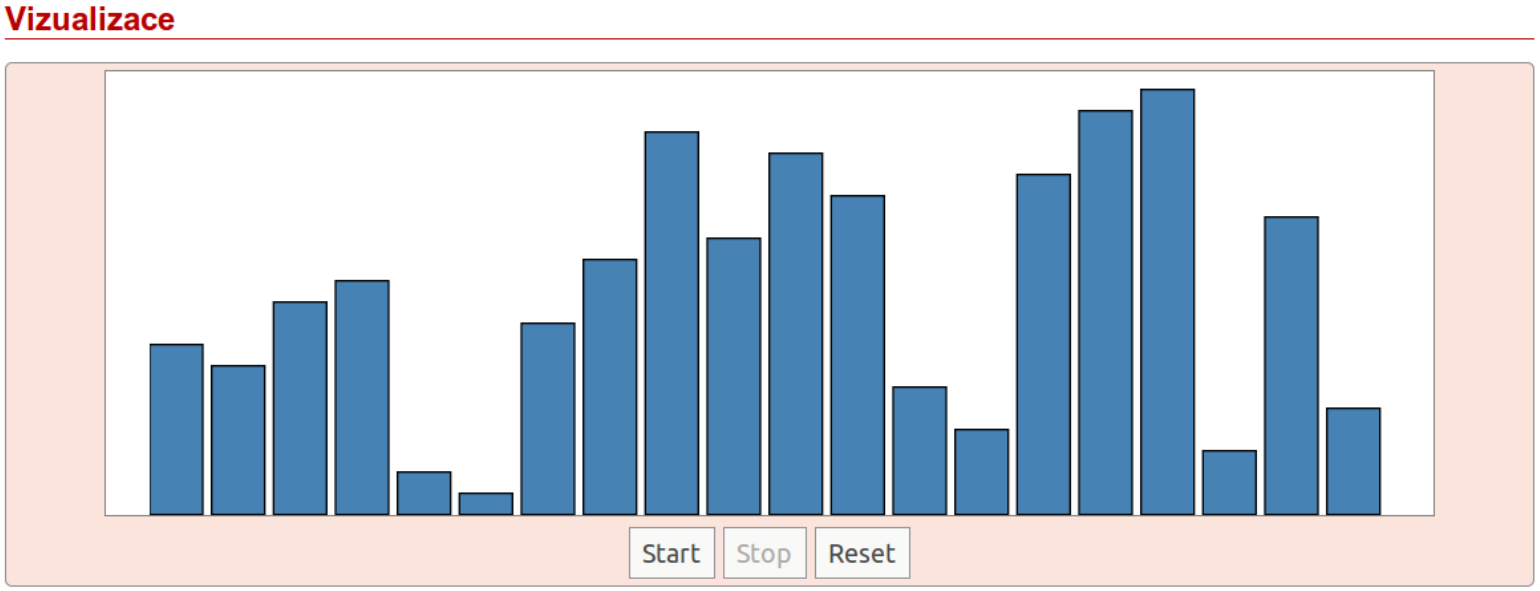
- Vstupní data mohou obsahovat část dat již uspořádaných podle kritéria, pak:
  - *Přirozené* algoritmy – vykazují rychlejší průběh.
  - *Nepřirozené* – uspořádanost vstupních dat nemá vliv na rychlost algoritmu.

Třídění – Bubble Sort

- **Metoda bublinkového třídění**
- Složitost $O(N^2)$
- Prvky probublávají tak dlouho, až jsou ve výsledku data setříděna.
- Popis Bubble Sort:
  1. *Porovnáváme sousední prvky.*
  2. *V případě splnění podmínky je vyměníme mezi sebou.*
  3. *Opakujeme, dokud průchod celou sadou je bez výměny sousedních prvků.*

Třídění – Bubble Sort

Ukázka bublinkového třídění na stránkách
https://www.algoritmy.net/article/3/Bubble-sort



Třídění – Bubble Sort

```
BubbleSort(pole prvek)
  FOR i=1, prvek.delka - 1 DO{
     FOR j=1, prvek.delka - i - 1 DO{
        IF prvek[j] < prvek[j+1] THEN
          PROHOD(prvek[j], prvek[j+1])
     }
  }
```

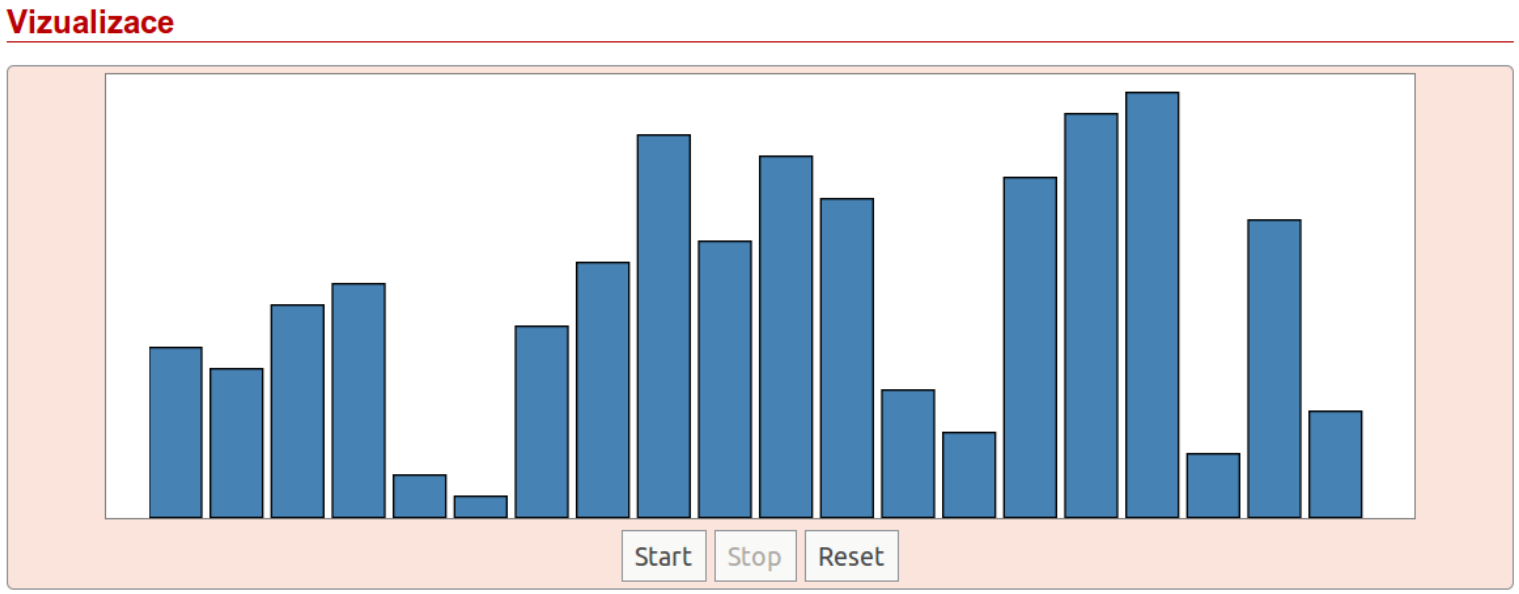
Třídění – Insert Sort

- **Metoda přímého vkládání**
- Složitost $O(N^2)$
- Data rozdělíme na "setříděnou" a "nesetříděnou" část. Prvky z nesetříděné části vložíme na odpovídající místo do setříděné části.
- Popis Insert Sort:

1. *První prvek z nesetříděné části vložíme na pozici dle velikosti.*
2. *Prvky ze setříděné části ležící za vloženým prvkem posuneme o jednu pozici dozadu.*
3. *Opakujeme, dokud je v nesetříděné části alespoň jeden prvek.*

Třídění – Insert Sort

Ukázka přímého třídění na stránkách
https://www.algoritmy.net/article/8/Insertion-sort



Třídění – Insert Sort

```
InsertSort(pole prvek)
  FOR i=0, prvek.delka - 2 DO{
    j=i+1
    tmp = a[j]
    WHILE j>0 AND tmp>a[j-1] DO{
      a[j] = a[j-1]
      j = j -1
    a[j] = temp
    }
  }
```

Třídění – Select Sort

- **Metoda třídění výběrem**
- Složitost $O(N^2)$
- Princip je postaven na výběru prvku a jeho uložení na odpovídající pozici (v případě sestupného třídění největší prvek na první pozici).
- Popis Select Sort:

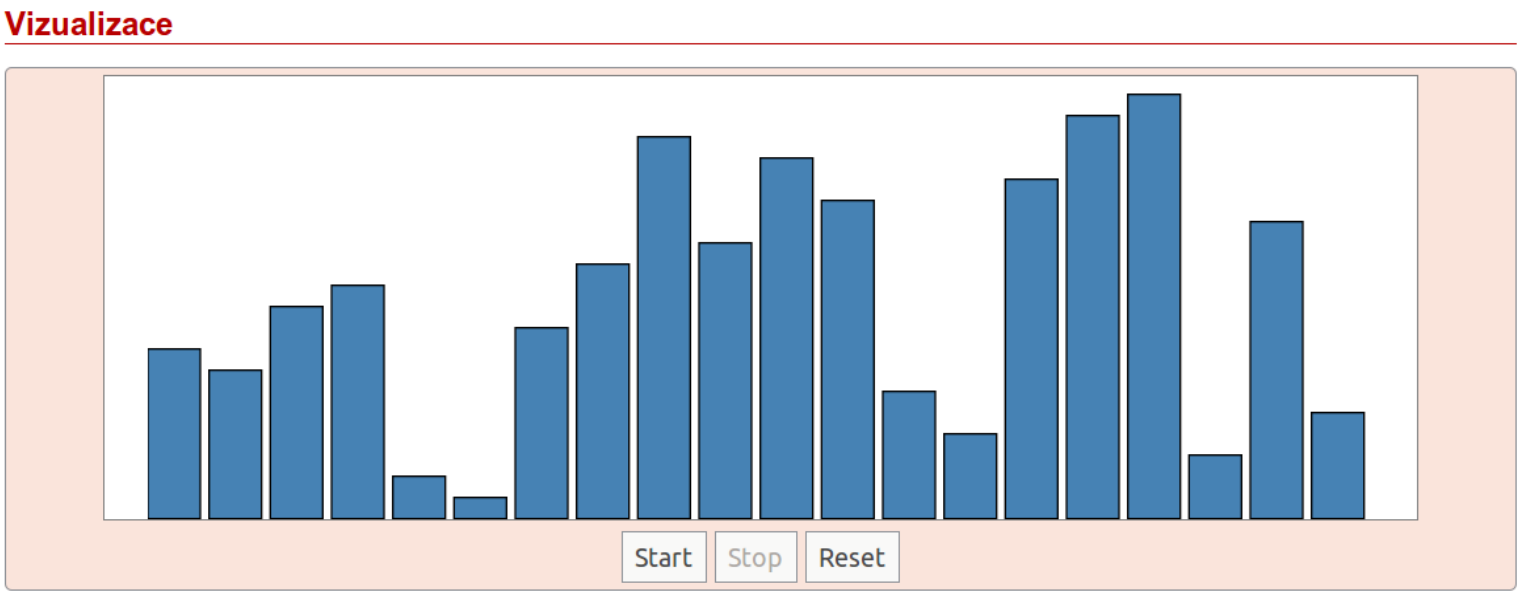
  1. Posloupnost prvků rozdělíme na setříděnou a nesetříděnou část.

  2. V nesetříděné části vyhledáme největší prvek a uložíme jej na poslední pozici v setříděné části.

  3. Proces opakujeme, doku nesetříděná část obsahuje alespoň jeden prvek.

Třídění – Select Sort

Ukázka třídění výběrem na stránkách
https://www.algoritmy.net/article/4/Selection-sort



Třídění – Select Sort

```
SelectSort(pole prvek)
  FOR i=0, prvek.delka - 2 DO{
    maxIndex = i
    FOR j=i+1, prvek.delka - 1 DO{
      if a[j] > maxIndex
        maxIndex = j
    PROHOD(prvek[j], prvek[j+1])
    }
  }
```

Třídění – Quick Sort

- **Metoda rychlého třídění**
- Složitost O(N log N), nejhorší případ $O(N^2)$
- Využívá princip rozděl a panuj (Divide & Conquere), implementace často s rekurzí
- Popis Quick Sort:

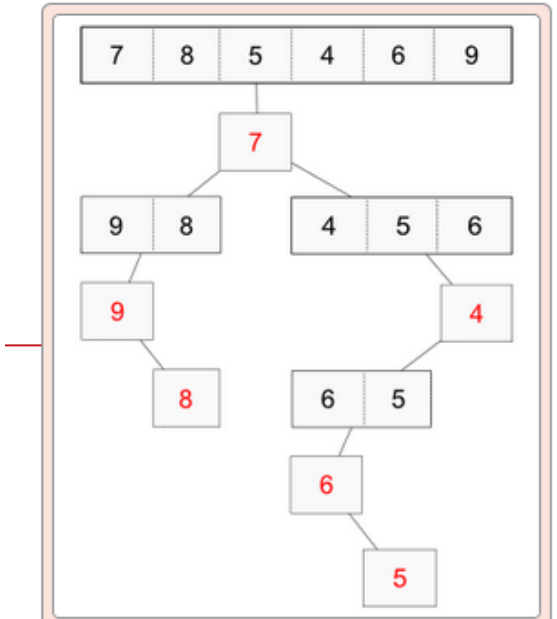
  1. Volíme prvek – pivota.

  2. Podle pivota rozdělíme posloupnost prvků do dvou částí, v jedné jsou menší a v druhé větší než pivot.

  3. Opakujeme princip na každou vzniklou část, dokud bude obsahovat více než jeden prvek.

Třídění – Quick Sort

Ukázka rychlého třídění na stránkách
https://www.algoritmy.net/article/10/Quicksort



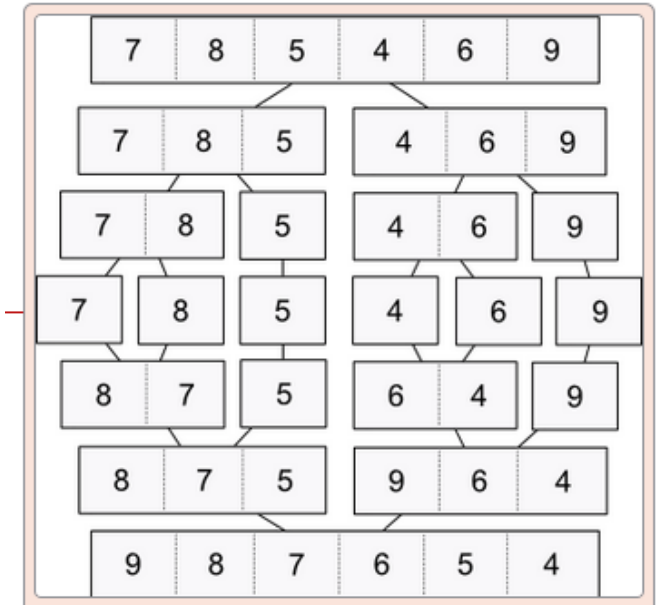
Quicksort - pivotem je první prvek

Třídění – Merge Sort

- **Metoda slévání**
- složitost O(N log N)
- Principem je slévání dvou setříděných posloupností prvků do jedné výsledné. Využívá principu rozděl a panuj (Divide & Conquere), implementace využívá pomocné pole.
- Popis Merge Sort:
  1. *Vstupní posloupnost prvků rozdělíme na dvě stejně velké části.*
  2. *Dělení (rekurze) skončí při jednotkové velikosti pole – triviálně setříděné.*
  3. *Sléváme dvě vedlejší pole tak, že do výstupního vkládáme prvky podle výsledku porovnání.*
  4. *Opakujeme, až je výstupem slévání celá posloupnost prvků.*

Třídění – Merge Sort

Ukázka třídění sléváním na stránkách
https://www.algoritmy.net/article/13/Merge-sort



Merge sort - vizualizace

Třídění – výběr algoritmu

- Výběr vhodného algoritmu ovlivní řada kritérií.
  - Struktura vstupních dat.
  - Velikost operační paměti.
  - Znalost/náročnost implementace.

- Který algoritmus je nejlepší?

Vyhledávání

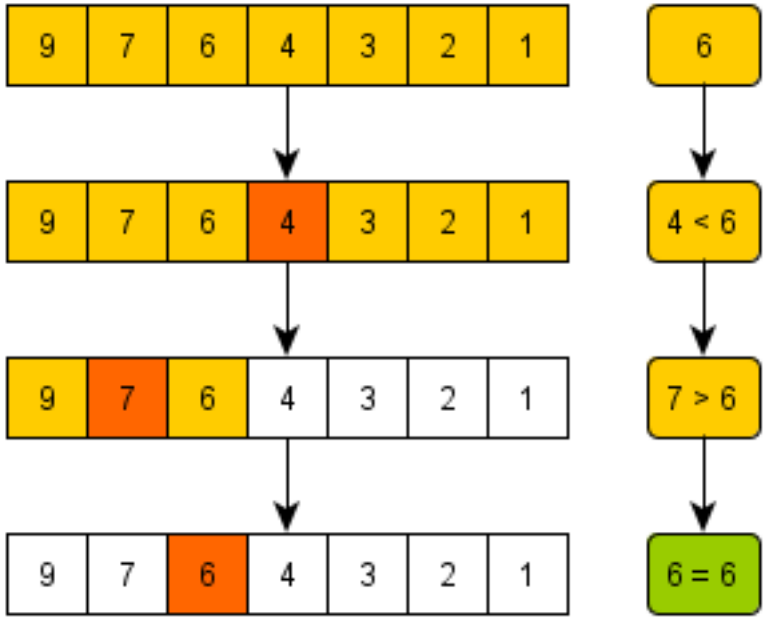
- Základní úlohou je najít prvek při shodě *klíče* – vyhledávací kritérium.

Sekvenční vyhledávání

- Lineární vyhledávání
- Postupně procházíme všechny prvky prostoru S a porovnáváme je s klíčem, dokud nenajdeme hledaný prvek.
- Na vstupu jsou neuspořádaná data.
- Složitost O(N).

Binární vyhledávání

- Lze najít i pod označením vyhledávání pomocí půlení intervalu.
- Data na vstupu musí být uspořádána.
- Používá se rekurze.
- Složitost O(log N).



Příklad v Pythonu

- Tvorba funkce pro setřídění seznamu čísel s využitím algoritmu Bubble sort

Příklad v Pythonu

```
def fSetrid_BS(seznam, razeni): #vytvarim vlastni funkci prikazem def, musim specifikovat jeji vstupni parametry v zavorce
    from sys import exit #importuji funkci exit pro okamzite ukonceni behu programu

    for i in range(0, len(seznam)-1):
        for j in range(0, len(seznam)-i-1): #vnoreny cyklus slouzi k jednomu projiti seznamu a
        #vzajemnemu prohazovani dvou sousednich prvku pokud je to potreba
            if razeni == 'od_nejvetsi': #pouzije se pokud se maji hodnoty setridit od nejvetsi po nejmensi
                if seznam[j] < seznam[j+1]:
                    temp = seznam[j] #pomuzu si pomocnou promennou temp, abych mohl prohodit dva sousedni prvky
                    seznam[j] = seznam[j+1]
                    seznam[j+1] = temp
            elif razeni == 'od_nejmensi': #pouzije se pokud se maji hodnoty setridit od nejmensi po nejvetsi
                if seznam[j] > seznam[j+1]:
                    temp = seznam[j]
                    seznam[j] = seznam[j+1]
                    seznam[j+1] = temp
            else: #pokud uzivatel spatne vlozi parametr razeni, je o tom informovan a program se ukonci
                print('vlozena hodnota parametru razeni je neplatna, mozne hodnoty jsou od_nejvetsi, od_nejmensi')
                exit()
    return seznam #prikaz return slouzi k vraceni pozadovaneho na vystupu funkce
```

Příklad v Pythonu

Použití funkce:

```
seznam_hodnot = [2,1.1,15,6,55,22,3,8,19,1.2]
seznam_hodnot_setrizeny = fSetrid_BS(seznam_hodnot,'od_nejmensi') #volame vlastni fkci
print(seznam_hodnot_setrizeny)
```

Literatura

- http://www.algoritmy.net/
- Korespondenční seminář z programování, https://ksp.mff.cuni.cz/

Děkuji za pozornost

Michal Kačmařík

michal.kacmarik@vsb.cz

www.vsb.cz