Seminár z algoritmizácie a programovania 1

Martin Bobák
Ústav informatiky
Slovenská akadémia vied

Obsah prednášky

1. **Algoritmy triedenia**

Spätná väzba:
https://forms.gle/iKbuLdF6xDtNSEDP8

Algoritmy

Algoritmus

- predpis, metóda alebo technika, ktorá špecifikuje postup úkonov potrebných na dosiahnutie riešenia nejakej úlohy
  - napr. usporiadanie zoznamu mien podľa abecedy
  - napr. recept na koláč

https://visualgo.net/en

- v informatike: je jednoznačná, presná a konečná postupnosť operácií, ktoré sú aplikovateľné na množinu objektov alebo symbolov (čísiel, šachových figúrok, surovín na koláč)
  - počiatočný stav týchto objektov je vstupom
  - ich koncový stav je výstupom
  - počet operácií, vstupy a výstupy sú konečné (aj keď počítame napr. s iracionálnym číslom π)

Efektívnosť algoritmov

Časová zložitosť:

- Závislosť časových nárokov algoritmu na veľkosti riešeného problému
- Vyjadrená počtom elementárnych operácií (aritmetických, logických, porovnaní, …)

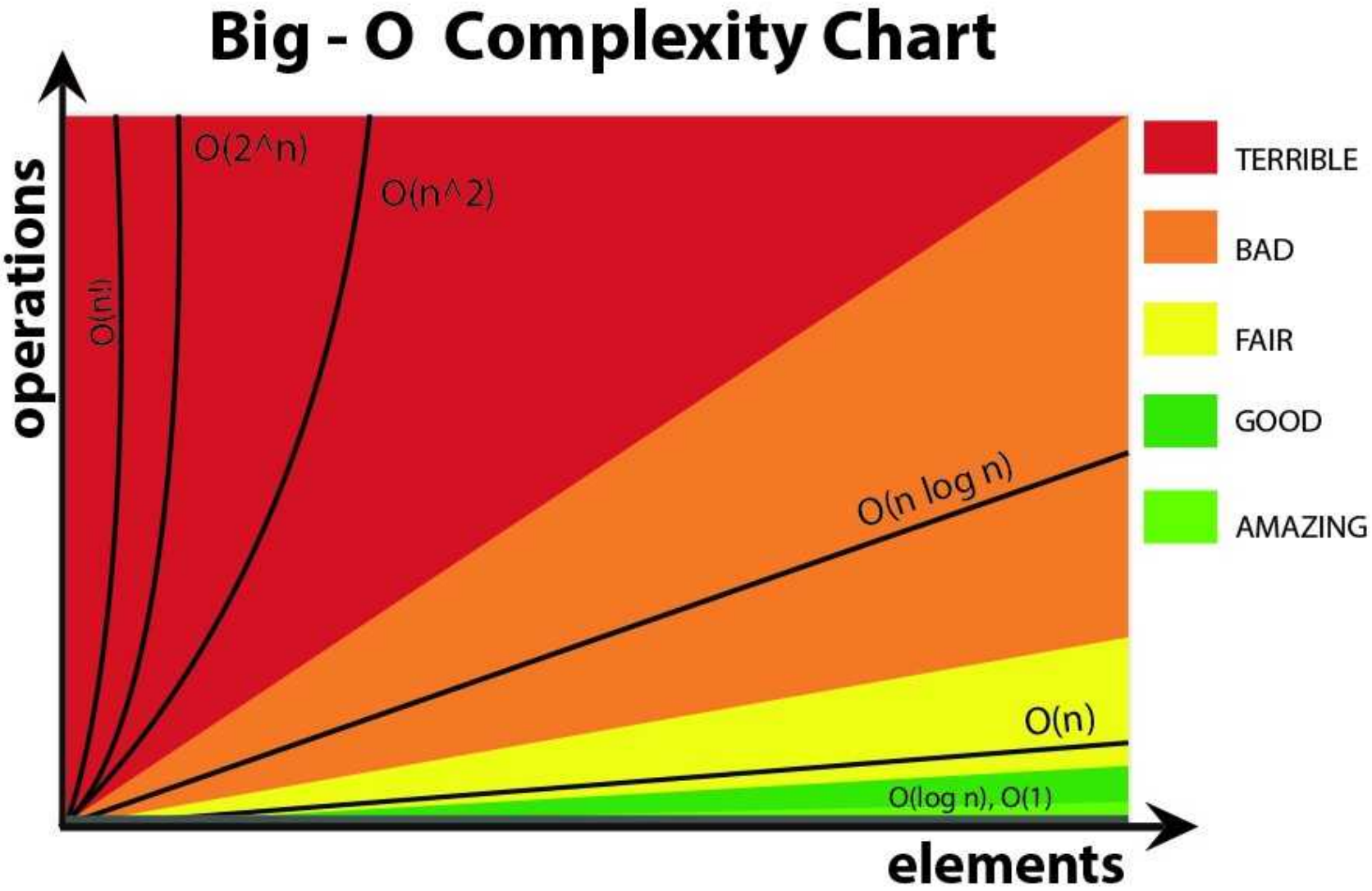
Pamäťová zložitosť:

- Závislosť pamäťových nárokov algoritmu na veľkosti riešeného problému
- Vyjadrená počtom premenných (pamäťových miest), ktoré algoritmus pri výpočte so zvolenými vstupnými údajmi potrebuje

Časová zložitosť

Označenie	Názov	Príklad
O(1)	konštantná	prístup k i-temu prvku v poli (algoritmus nezávisí od veľkosti vstupu)
O(log n)	logaritmická	hľadanie prvku v utriedenom poli (binárne vyhľadávanie)
O(n)	lineárna	hľadanie prvku v neutriedenom poli
O(n log n)	"linearitmická"	triedenie poľa
$O(n^2)$	kvadratická	"dva vnorené for cykly" napr. bublinkové triedenie
$O(2^n)$	exponenciálna	prehľadávanie s návratom
O(n!)	faktoriálová	vygenerovanie všetkých permutácií

Časová zložitosť



https://amitshahi.dev/static/756663638e3de206cc41988bfa13b7db/2d017/bigocomplexitychart.jpg

Triedenie

Problém

Vstup: pole a[1...n] celých čísiel

Výstup: permutácia b[1...n] prvkov pola a taká, že

b[1] <= b[2]<=...<=b[n]

a = {12, 3, 67, 24, 9} -> b = {3, 9, 12, 24, 67}

Ako najefektívnejšie vieme usporiadať pole?

Motivácia

Zrejmé aplikácie:

- organizácia priečinka
- udržiavanie slovníka

Zjednodušenie problémov:

- hľadanie mediána
- binárne vyhľadávanie, hľadanie štatistických "mimoňov"

Pokročilé aplikácie:

- Kompresia dát: hľadanie duplikátov
- Počítačová grafika: renderovanie scény

Parametre triediacich algoritmov

- počet porovnaní/výmen
- využitie pomocnej pamäte
- rekurzia
- stabilita
  - relatívna poloha rovnakých prvkov zostane po triedení zachovaná (napr. pole už bolo utriedené podľa iného kritéria)
- vnútorné/vonkajšie triedenie
  - využitie RAM alebo disku počas triedenia

Bubble sort

- porovnávanie hodnôt dvoch susedných buniek poľa
  - do bunky s nižším indexom menšie z nich
  - do bunky s vyšším indexom väčšie z nich
- po jednom prechode poľom sa určite maximálny prvok dostane na koniec poľa, potom ostáva usporiadať N-1 prvkov poľa (posledný je už na svojom mieste)
- najmenej efektívne in-place triedenie

$$O(n^2)$$

Bubble sort

```
void bubblesort(int a[], int n)
{
 int i,j;

 for (i = n; i > 1; i--)
    for (j = 1; j < i; j++)
       if (a[j-1] > a[j])
          vymen(&a[j-1], &a[j]);
}
```

v úseku 0...i "vybublá" najväčší prvok nakoniec

https://www.youtube.com/watch?v=Cq7SMsQBEUw

```
void vymen(int *x,
int *y)
{
    int pom = *x;
    *x = *y;
    *y = pom;
}
```

Selection sort (Min sort, **Max sort**)

- nájdi v úseku maximálny prvok, vymeň ho s posledným prvkom, skráť usporiadavané pole o 1- pokiaľ nie je jednoprvkové (in-place)
- porovnanie s BubbleSort-om:
  - počet porovnávaní: **O(n²)**
  - počet výmen: **O(n)**

Selection sort (Min sort, Max sort)

```
void maxsort(int a[], int n)
{
    int i, j, max;

    for (i = n-1; i > 0; i--) {
      max = 0;
      for (j = 1; j <= i; j++)
        if (a[j] > a[max])
            max = j;
      if (i != max)
        vymen(&a[max], &a[i])
    }
}
```

https://www.youtube.com/watch?v=92BfuxHn2XE **(Min sort)**

v úseku 0...i nájdeme maximum

ak maximum z úseku 0...i nie i-ty prvok, vymeníme ich

Insert sort

- časť poľa je usporiadaná a vsunie sa do nej prvok tak, aby pole zostalo usporiadané (in-place)
- V najhoršom prípade **O(n²)**, na skoro utriedenom poli O(n)
- Efektívnejšie ako selection sort

Insert sort

```
void insertsort(int a[], int n)
{
    int i, j, pom;

    for (i = 1; i < n; i++) {
        pom = a[i];
        j = i-1;
        while (j >= 0   && a[j] > pom) {
            a[j+1] = a[j];
            j--;
        }
        a[j+1] = pom;
    }
}
```

úsek `0..i-1` je usporiadaný, vsunieme `a[i]` tak, aby zostal usporiadaný

https://www.youtube.com/watch?v=8oJS1BMKE64

hľadanie vhodného miesta pre prvok `a[i]`

Merge sort

- Založený na prístupe rozdeľ a panuj:
  1. jednoprvkové pole je utriedené
  2. rekurzívne utrieď polia a[1...n/2] a a[n/2+1...n]
  3. zlúč dve usporiadané časti polia
- Zložitosť je **O(n log n)**
  - rekurzia sa zavolá log n kráť
  - zlúčenie dvoch polí trvá O(n)

Zlúčenie dvoch utriedených polí

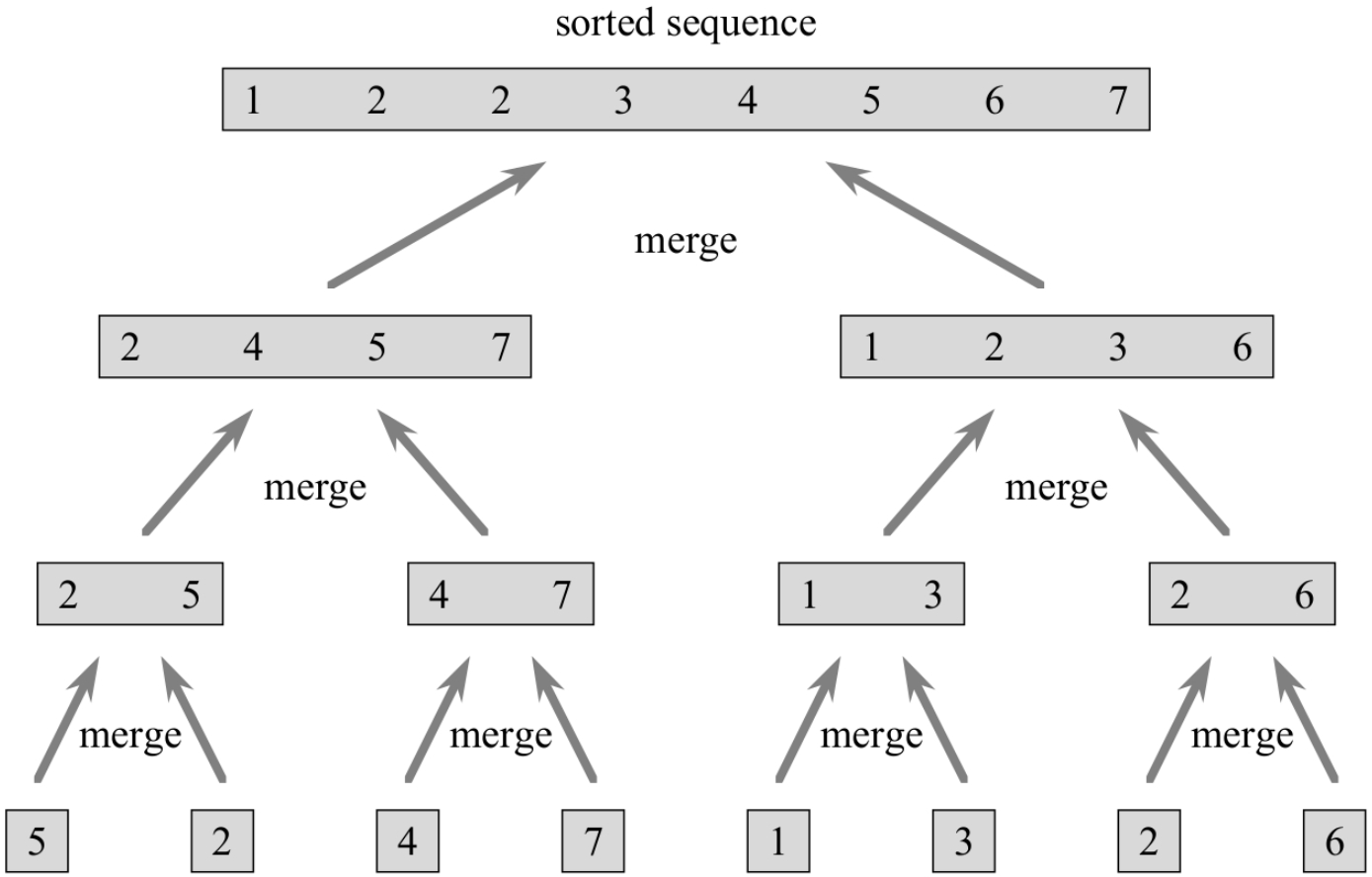
L	R	
[**3**, 6, 7, 9]	[**1**, 5, 8]	[]
[**3**, 6, 7, 9]	[**5**, 8]	[1]
[**6**, 7, 9]	[**5**, 8]	[1, 3]
[**6**, 7, 9]	[**8**]	[1, 3, 5]
[**7**, 9]	[**8**]	[1, 3, 5, 6]
[**9**]	[**8**]	[1, 3, 5, 6, 7]
[**9**]	[]	[1, 3, 5, 6, 7, 8]
[**9**]	[]	[1, 3, 5, 6, 7, 8, 9]

Zlúčenie

```
void merge(int arr[], int l,
int m, int r) {
    int i, j, k;
    int n1 = m - l + 1;
    int n2 = r - m;

    int L[n1], R[n2];
    for (i = 0; i < n1; i++)
        L[i] = arr[l + i];
    for (j = 0; j < n2; j++)
        R[j] = arr[m + 1 + j];

    i = 0;
    j = 0;
    k = l;
```

```
while (i < n1 && j < n2)
{
    if (L[i] <= R[j]) {
        arr[k] = L[i];
        i++;
    } else {
        arr[k] = R[j];
        j++;}
    k++;}
while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;}
while (j < n2) {
    arr[k] = R[j];
    j++;
    k++;} }
```

Merge sort



Zdroj: Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms. MIT press.

a = {5, 2, 4, 7, 1, 3, 2, 6}

Merge sort

```
void mergeSort(int arr[], int l,
int r)
{
    if (l < r) {
        int m = l + (r - l) / 2;

        mergeSort(arr, l, m);
        mergeSort(arr, m + 1, r);

        merge(arr, l, m, r);
    }
}
```

index stredného prvku
(l+r)/2

https://
www.youtube.com/
watch?
v=ZRPoEKHXTJg

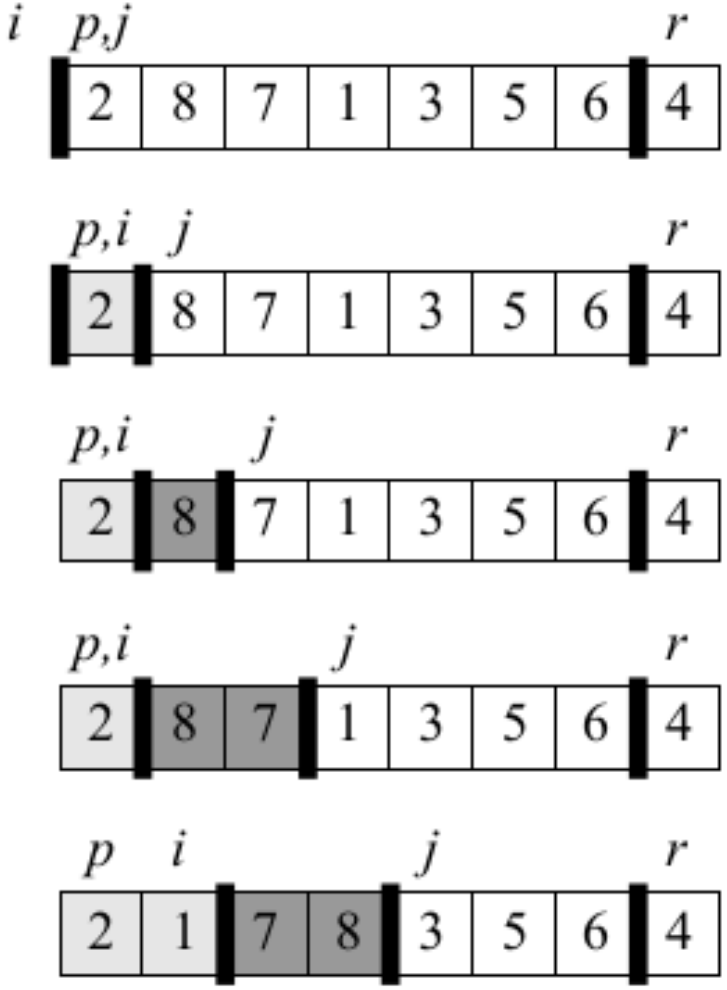
Iteratívny merge sort

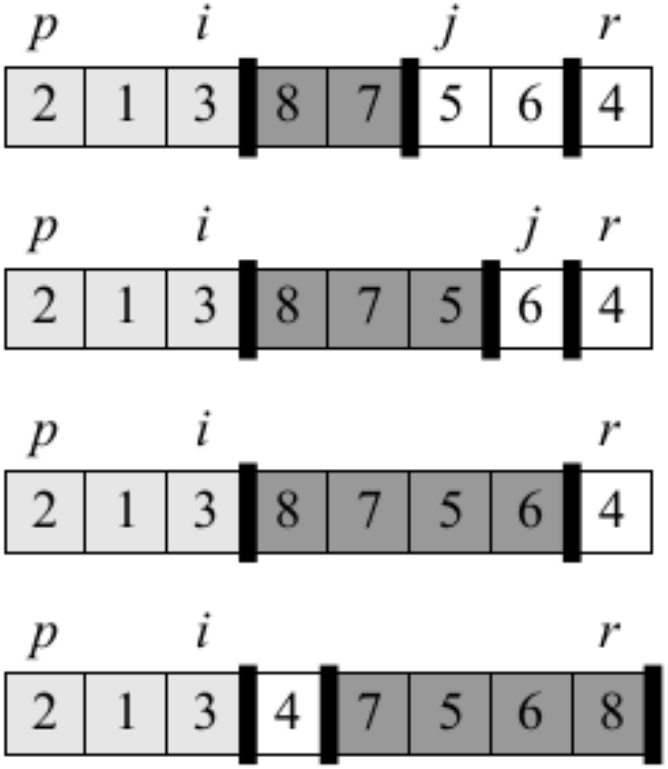
```
void mergeSort(int arr(), int n) {

   for (int i=1; i<=n-1; i = 2*i) {
      for (int l=0; l<n-1; l += 2*i) {

         int m = min(l+i-1, n-1);
         int r = min(l+2*i-1, n-1);
         merge(arr, l, m, r);
        }}}
```

Quick sort

- založený na prístupe rozdeľuj a panuj:
  1. z poľa sa vyberie pivot (napr. prvý prvok poľa)
  2. podľa pivota rozdelíme vstupné pole na tri časti:
     - čísla menšie a rovné ako pivot
     - pivot
     - väčšie ako pivot
  - potom rovnakým spôsobom usporadúvame prvú a tretiu časť
- Najhorší prípad má kvadratickú zložitosť, očakávaný O(n log n)
  - pivot nerozdelí pole na dve časti, ale jedna z nich je prázdna (t.j. pivot je maximum, alebo minimum)
  - rôzne stratégie výberu pivota
- Efektívny pre veľké polia
  - "administratíva" okolo rekurzie a manipulácia s pivotom sú zanedbateľné
  - v praxi hybridné riešenie: kým je pole veľké quick sort, na malých častiach využijeme "neefektívne" triedenie (v mnohých prípadoch je to insert sort)

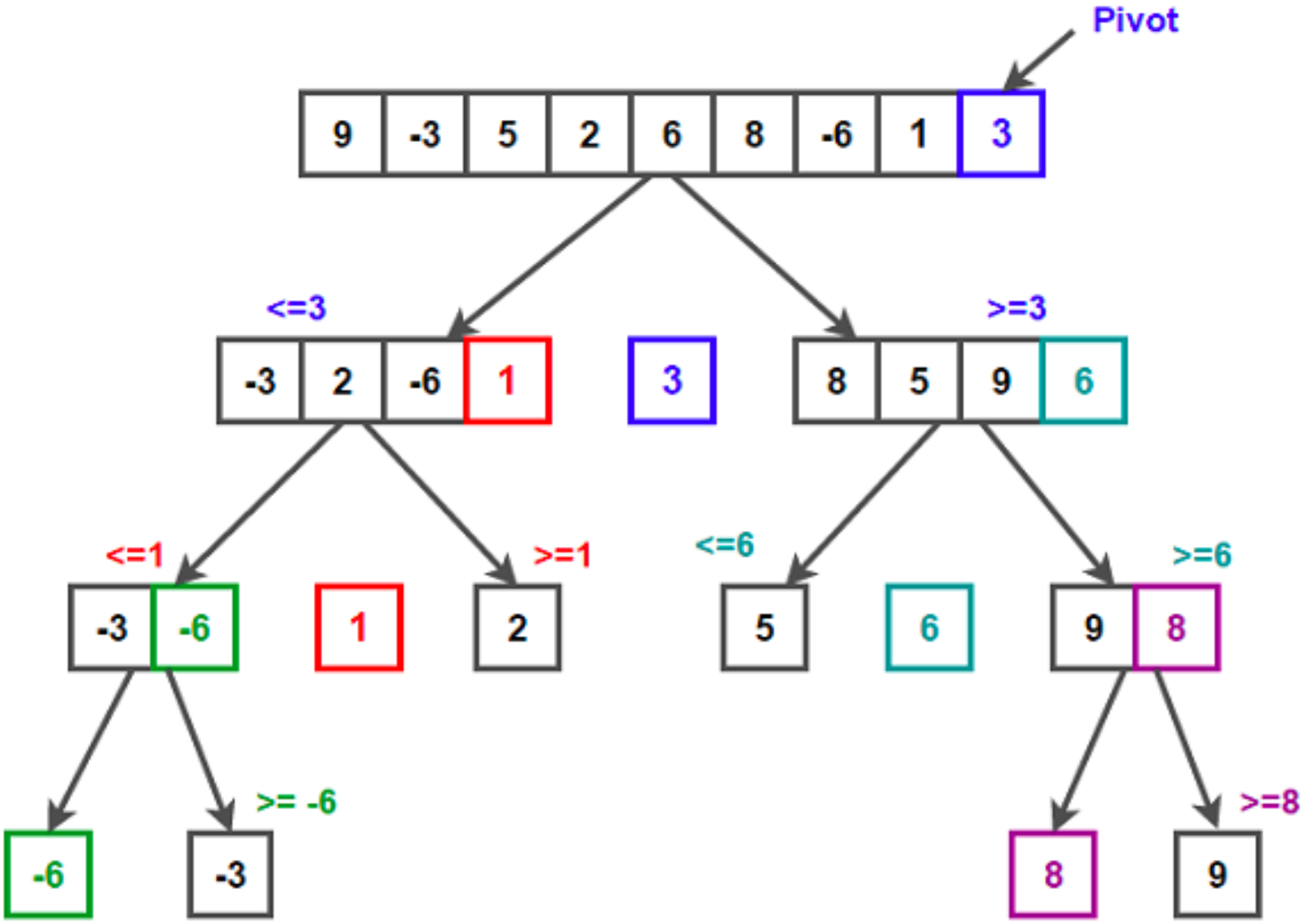
Quick sort





Zdroj: Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms. MIT press.

Quick sort



https://afteracademy.com/blog/quick-sort

Quick sort

```
int rozdel(int a[], int l, int r) {
   int i, j;
   int pivot = a[l];

   i = l; j = r+1;
   do {
      do ++i; while (a[i] <= pivot && i <= r);
      do --j; while (a[j] > pivot);
   if (i < j)
      vymen(&a[i], &a[j]);
   } while (i < j);
   vymen(&a[l], &a[j]);
   return j;
}
```

```
void quickSort(int a[], int l,
int r) {
   int j;

   if (l < r) {
      j = rozdel(a, l, r);
      quickSort(a, l, j-1);
      quickSort(a, j+1, r);
   }
}
```

https://www.youtube.com/watch?v=8hEyhs3OV1w

Quick sort v jazyku C

```
#include <stdlib.h>
```

```
qsort(array, length, sizeof(type), compFunc);
```

veľkosť poľa

veľkosť prvku v bajtoch

ukazovateľ na porovnávaciu funkciu

```
int compFunc(const void *a, const void b*){
    return(*(int *)a - *(int *)b);}
```

záporné číslo znamená výmenu b za a

kladné číslo znamená výmenu a za b

0 znamená, že nie je potrebná výmena

Quick sort v jazyku C

usporiadame osoby vzostupne podľa veku

```
struct Osoba
{
    int vek;
    char meno[20];
};
```

```
int compFunc(const void *a, const void b*){
    int l = ((struct Osoba *)a)->vek;
    int r = ((struct Osoba *)b)->vek;
    return (l - r);
}
```

```
qsort((void*)pole, length, sizeof(pole[0]), compFunc);
```

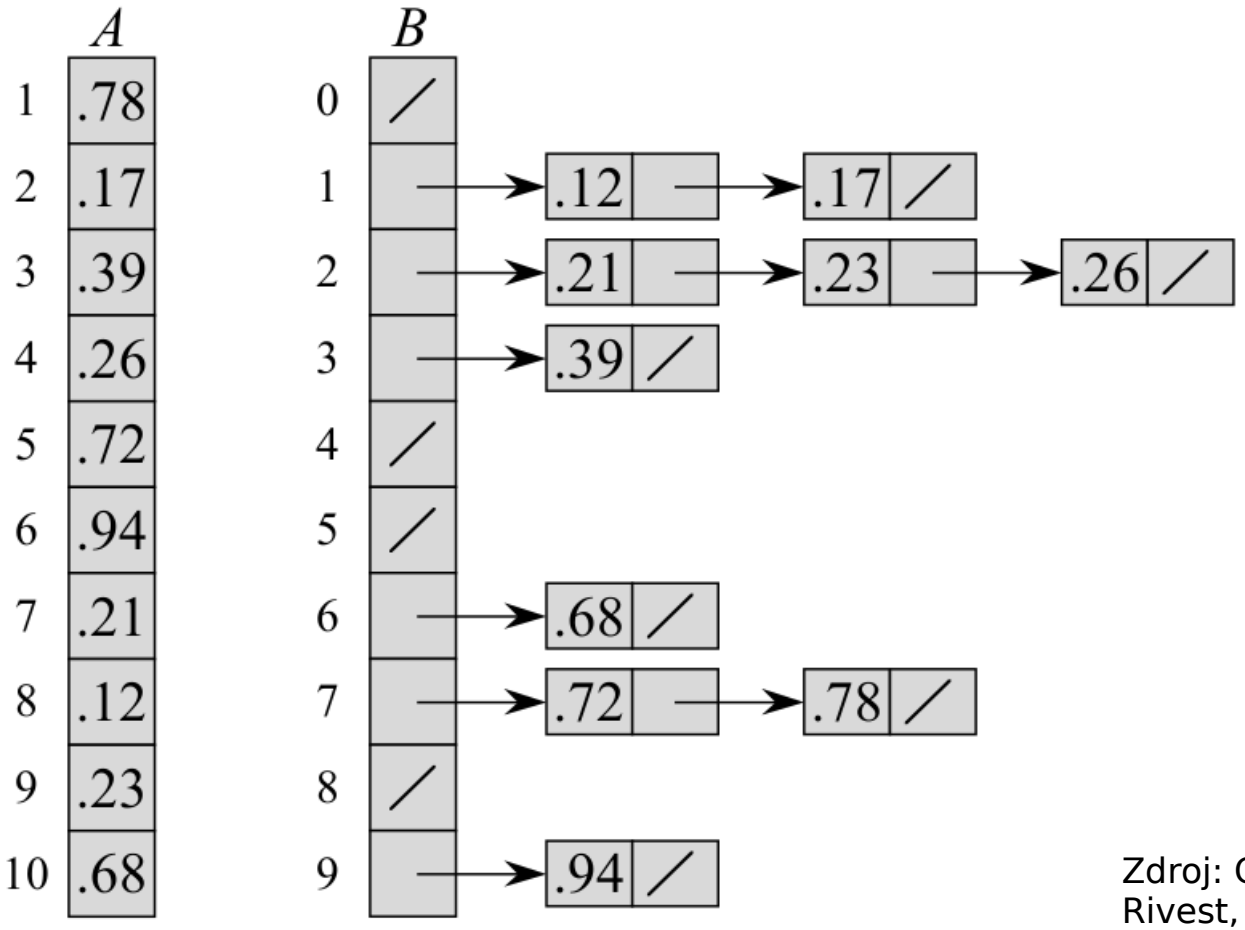
http://www.cplusplus.com/reference/cstdlib/qsort/

Triedenie v O(n)?

- zatiaľ najlepšie triedenie malo zložitosť O(n log n)
- triedenia založené na porovnaniach nemôžu byť rýchlejšie
  - spor s porovnaním všetkých prvkov

- dodatočná informácia
  - poznáme rozsah hodnôt

Bucket sort



Zdroj: Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). Introduction to algorithms. MIT press.

Triedenia

Triedenie v čase $O(n^2)$:

- Bubble sort
- Insert sort
- Selection sort

Triedenie v čase O(n log n):

- Merge sort
- Quick sort
-

Triedenie v čase O(n):

- Bucket sort

Ďakujem vám za pozornosť!

Spätná väzba:
https://forms.gle/iKbuLdF6xDtNSEDP8