STU FIIT

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ

Dátové štruktúry a algoritmy

Triedenie – Usporadúvanie

21. 4. 2021

letný semester 2020/2021

prednášajúci: Lukáš Kohútka

Triedenie – Usporadúvanie

- Základná aplikácia počítačov
- Vstup:
  - Postupnosť: $a_1, a_2, a_3 \dots a_n$
    $k(a_i)$ označíme kľúč k_i prvku a_i
  - Usporiadanie kľúčov $<$ (binárna relácia)

    Lineárne usporiadaná množina K (total ordering)
    Pre $k_1, k_2 \in K$ budeme písať, že $k_1 \leq k_2$ akk $k_1 < k_2$ alebo $k_1 = k_2$.
- Výstup:
  - Permutácia π čísel 1, …, n taká, že platí
    $k(a_{\pi(1)}) \leq k(a_{\pi(2)}) \leq \dots \leq k(a_{\pi(n)})$

Triedenie – Usporadúvanie – príklad

- Vstup: **Postupnosť: $a_1, a_2, \ldots, a_n$**
  $k(a_i)$ označíme kľúč k_i prvku a_i
- Výstup: **Permutácia π čísel 1, …, n taká, že platí**
  $k(a_{\pi(1)}) \leq k(a_{\pi(2)}) \leq \ldots \leq k(a_{\pi(n)})$

- Vstup:
  Peter, Jano, Milan, Miro, Filip
- Výstup?
- **$\pi = (5, 2, 3, 4, 1)$**
  Výsledné poradie kľúčov:
  Filip, Jano, Milan, Miro, Peter

Odhady zložitosti algoritmov – opakovanie

- Analýza najhoršieho prípadu
- Použitie O-notácie pre asymptotický horný odhad
- Klasifikujeme algoritmy podľa týchto zložitostí
- Nevýhoda tohto prístupu: **Nemôžeme použiť na predvídanie výkonu alebo porovnanie algoritmov!**
  - Quicksort – počet porovnaní v najhoršom prípade $O(N^2)$
  - Mergesort – počet porovnaní v najhoršom prípade $O(N \log N)$
  - V praxi je však Quicksort zvyčajne dva krát rýchlejší a používa polovičné množstvo pamäti…

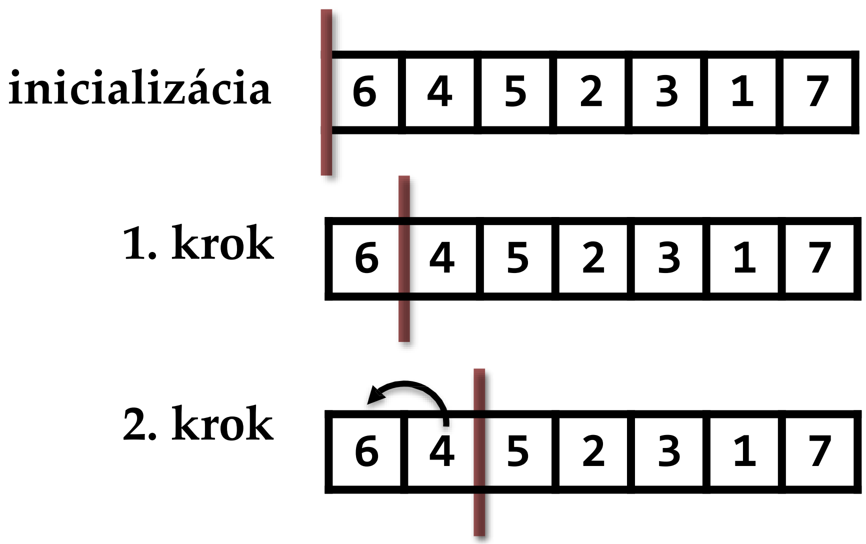
Triedenie priamym vkladaním (Insert sort)

- Insert sort spracúva vstupnú postupnosť postupne tak, že pojednom pridáva prvky na správne miesto do výslednej usporiadanej postupnosti (ktorá je najskôr prázdna a postupne sa rozširuje).

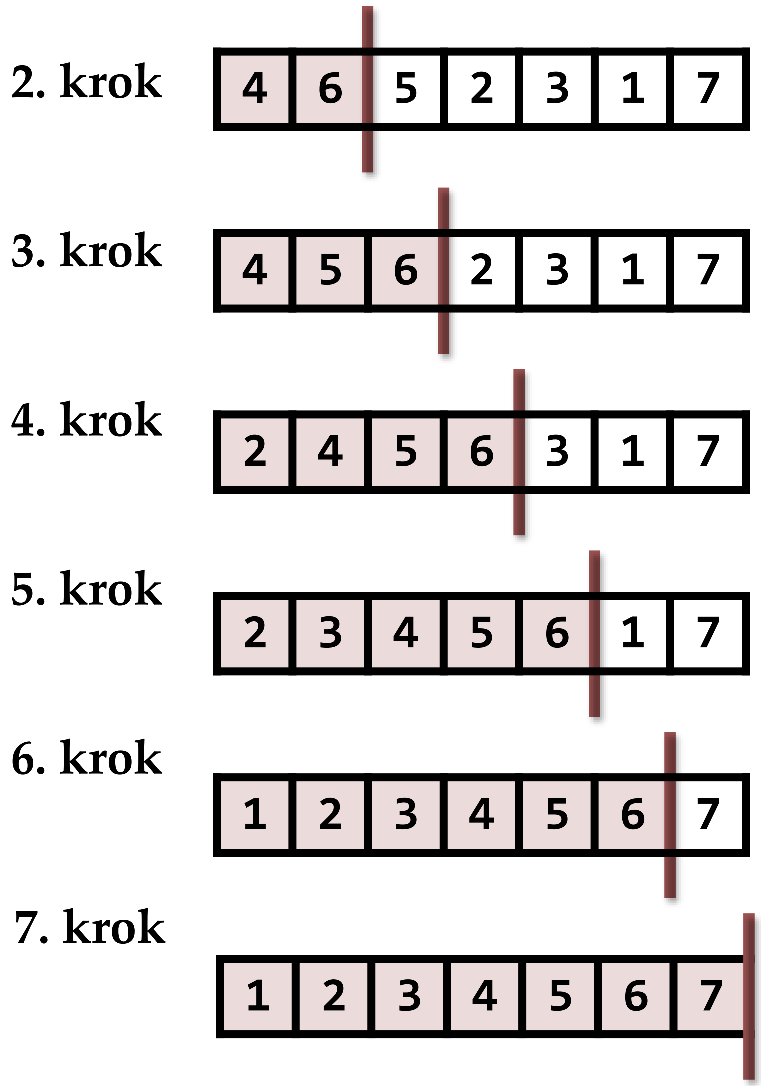
```
int* insert_sort(int *input, int n)
{
  int i, result[n];
  for (i = 0; i < n; i++)
    insert(input[i], result);
  return result;
}
```

Triedenie priamym vkladaním (Insert sort)

- Insert sort spracúva vstupnú postupnosť postupne tak, že po jednom pridáva prvky na správne miesto do výslednej usporiadanej postupnosti (ktorá je najskôr prázdna a postupne sa rozširuje).



Triedenie priamym vkladaním (Insert sort)



Triedenie priamym vkladaním (Insert sort)

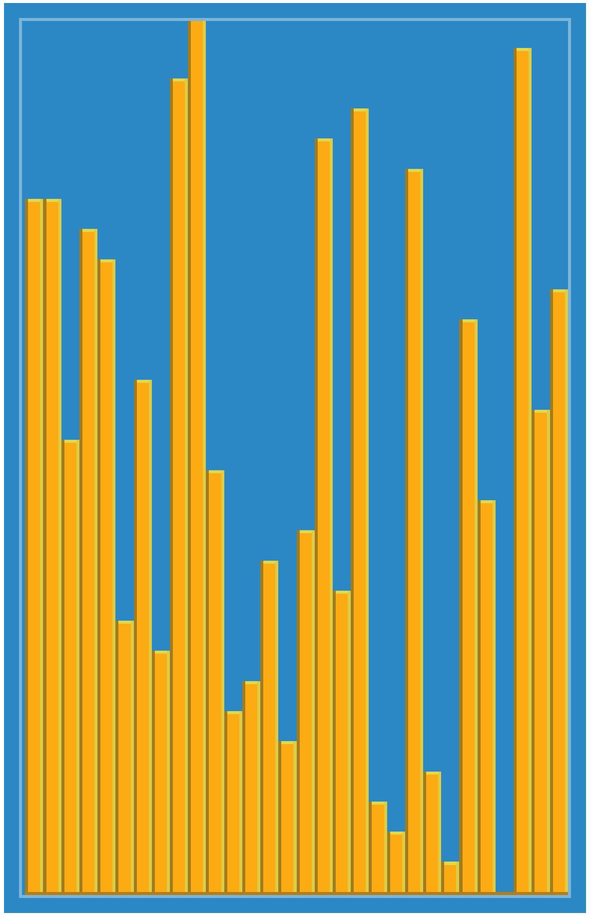
```
int* insert_sort(int *input, int n)
{
  int i, result[n];
  for (i = 0; i < n; i++)
    insert(input[i], result);
  return result;
}
```

- Procedúra **insert**(prvok, pole) pomocou jednoduchého cyklu vloží prvok do poľa (v ktorom sú prvky v usporiadanom poradí) na správne miesto; vyžaduje rádovo L operácií, kde L je dĺžka poľa.

Triedenie priamym vkladaním (Insert sort)

- Animovaná ukážka: https://visualgo.net/bn/sorting

6 5 3 1 8 7 2 4

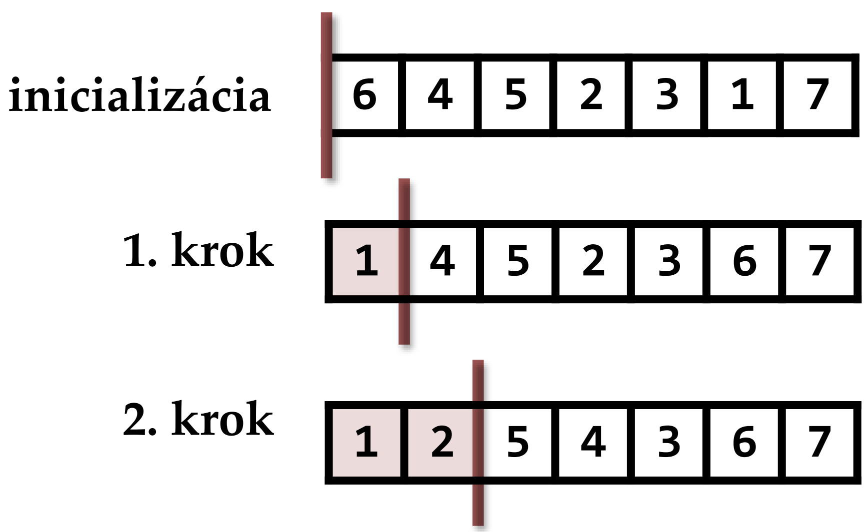


Analýza zložitosti (Insert sort)

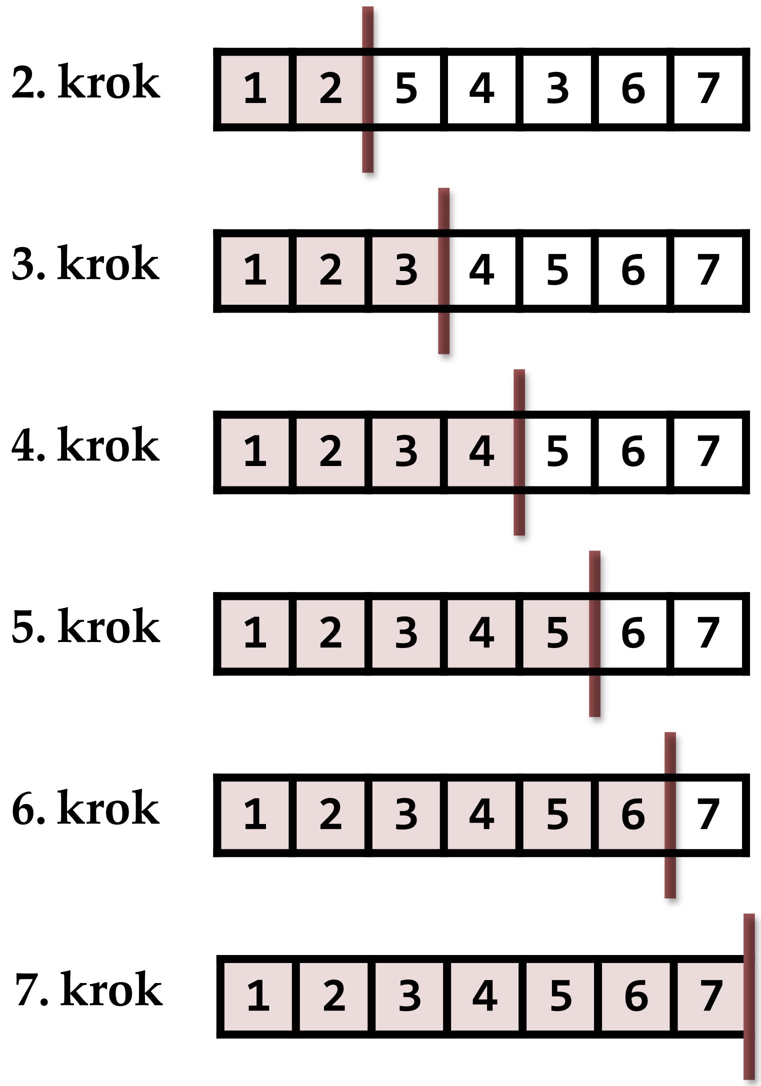
- **Najlepší prípad: prvky sú už usporiadané**
  Procedúra insert vykoná O(1) presunov
  Celkovo – N krát insert O(1) = O(N) operácií
- **Najhorší prípad: prvky sú usporiadané opačne**
  Volania procedúry insert vykonajú koľko presunov?
  0 + 1 + 2 + ... + N-1 = (N-1)*N/2
  Celkovo $O(N^2)$ operácií
- **Priemerný prípad: prvky sú náhodne usporiadané**
  Volania procedúry insert vykonajú koľko presunov?
  Asi polovicu ako pri najhoršom prípade
  Celkovo $O(N^2)$ operácií

Triedenie výberom (Select sort)

- Najjednoduchší-najprirodzenejší algoritmus
- Algoritmus:
  - Najmenší prvok môžeme zaradiť na začiatok vstupného poľa (najmenší vymeníme s prvkom, ktorý je nazačiatku)
  - Najmenší prvok zo zvyšku poľa bude druhý najmenší, atď.
- Označujeme aj MinSort / MaxSort



Triedenie výberom (Select sort)



Najlepší prípad?

Najhorší prípad?

Priemerný prípad?

Miera usporiadanosti vstupnej postupnosti poľa nemá vplyv na časovú zložitosť – vždy sa vykoná maximálny počet porovnaní. Ovplyvniť môžeme len počet výmen, ktorých je ale vždy menej ako porovnaní.

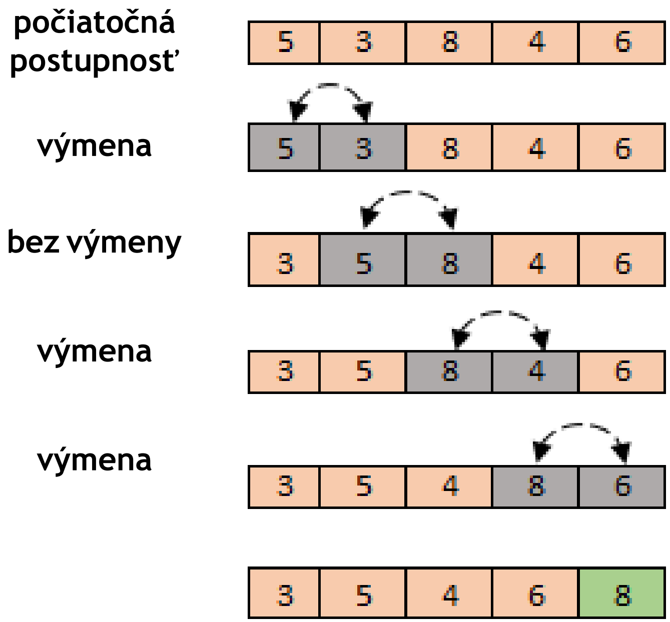
Usporadúvanie výmenami (Bubble sort)

- Pri usporadúvaní porovnáva dva susedné prvky a ak nie sú v správnom poradí, vymenia sa
- Procedúra sa opakuje, až kým nie sú prvky usporiadané (nie sú potrebné ďalšie výmeny)

```
koniec = 0;
while (!koniec)  // opakujeme, kym su neusporiadane
{
  koniec = 1;
  for (i = 0; i < n-1; i++)
    if (a[i] > a[i+1])
    {
      swap(&a[i], &a[i+1]); // vymen prvky i a i+1
      koniec = 0;
    }
}
```

Bubble sort - príklad

- Jeden prechod vnútorného cyklu
  - presun najväčšieho prvku na koniec



- Opakujeme prechody, až kým nie je všetko usporiadané
  - Zistím tak, že spravím prechod pri ktorom nebolo potrebné vymeniť žiadnu dvojicu susedných prvkov

Analýza zložitosti (Bubble sort)

- jeden prechod = presun najväčšieho prvku na koniec
- i-tý prechod: n-i+1 operácií
- Najlepší prípad: 1 prechod :) O(n)
- Najhorší prípad:
  $(n - 1) + (n - 2) + \ldots + 1 = (n - 1) * n / 2 = O(n^2)$
- Implementačne jednoduchý ale výpočtovo neefektívny

Problémy:

- Čo keď najmenší prvok je na konci?
  Až v poslednom prechode bude na začiatku
- Vylepšenia sa snažia vylepšiť tento (a podobné) prípady

Ďalšie sortovacie algoritmy

- Merge sort
- Quick sort
- Heap sort
- Tree sort
- Counting sort
- Radix sort
- Bucket sort
- … a iné …

Triedenie – Usporadúvanie

- Základná aplikácia počítačov
- Vstup:
  - Postupnosť: $a_1, a_2, a_3 \ldots a_n$
    $k(a_i)$ označíme kľúč k_i prvku a_i
  - Usporiadanie kľúčov $<$ (binárna relácia)

    Lineárne usporiadaná množina K (total ordering)
    Pre $k_1, k_2 \in K$ budeme písať, že $k_1 \leq k_2$ akk $k_1 < k_2$ alebo $k_1 = k_2$.
- Výstup:
  - Permutácia π čísel 1, …, n taká, že platí
    $k(a_{\pi(1)}) \leq k(a_{\pi(2)}) \leq \ldots \leq k(a_{\pi(n)})$

Jednoduché triediace algoritmy

- Triedenie priamym vkladaním (Insert sort)
- Triedenie výberom (Select sort)
- Usporadúvanie výmenami (Bubble sort)

- worst-case a average-case $O(N^2)$

Triedenie zlučovaním (Merge sort)

- Merge sort vstupnú postupnosť rozdelí na dve polovice, každú rekurzívne utriedi, no a výslednú usporiadanú postupnosť všetkých prvkov určí zlúčením týchto menších usporiadaných postupností.

```
int* merge_sort(int *input, int left, int right)
{
  int mid = (left+right)/2;
  merge_sort(input,left,mid);
  merge_sort(input,mid+1,right);
  return merge(input,left,mid,right);
}
```

Triedenie zlučovaním (Merge sort)

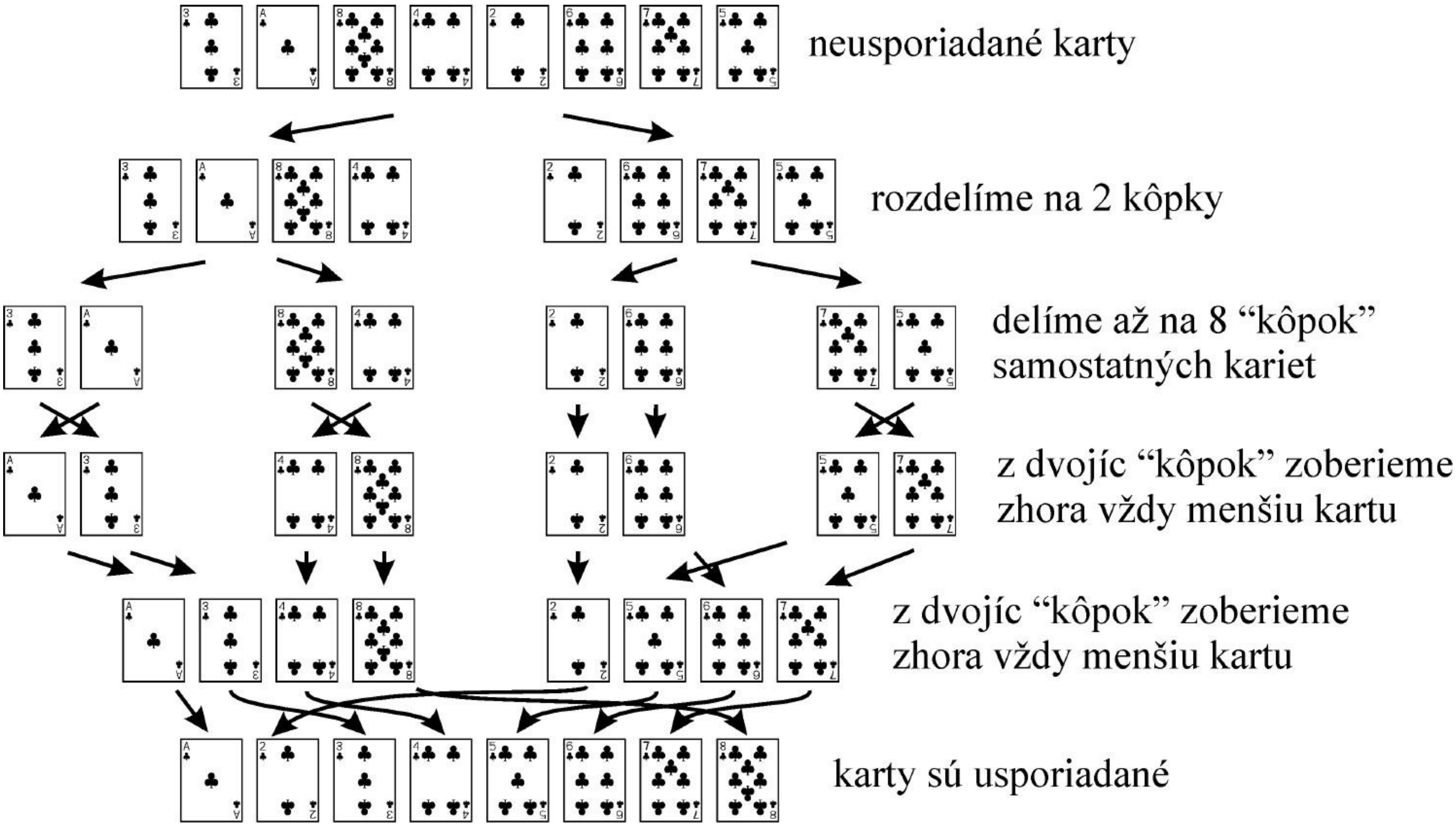
```
int* merge_sort(int *input, int left, int right)
{
  int mid = (left+right)/2;
  merge_sort(input,left,mid);
  merge_sort(input,mid+1,right);
  return merge(input,left,mid,right);
}
```

- Procedúra **merge**(input, left, middle, right) pomocou jednoduchého cyklu spojí usporiadané postupnosti prvkov input[left, …, middle] a input[middle+1, …, right] do jednej usporiadanej postupnosti; vyžaduje rádovo right-left (dĺžka poľa vstupujúceho do operácie) operácií.

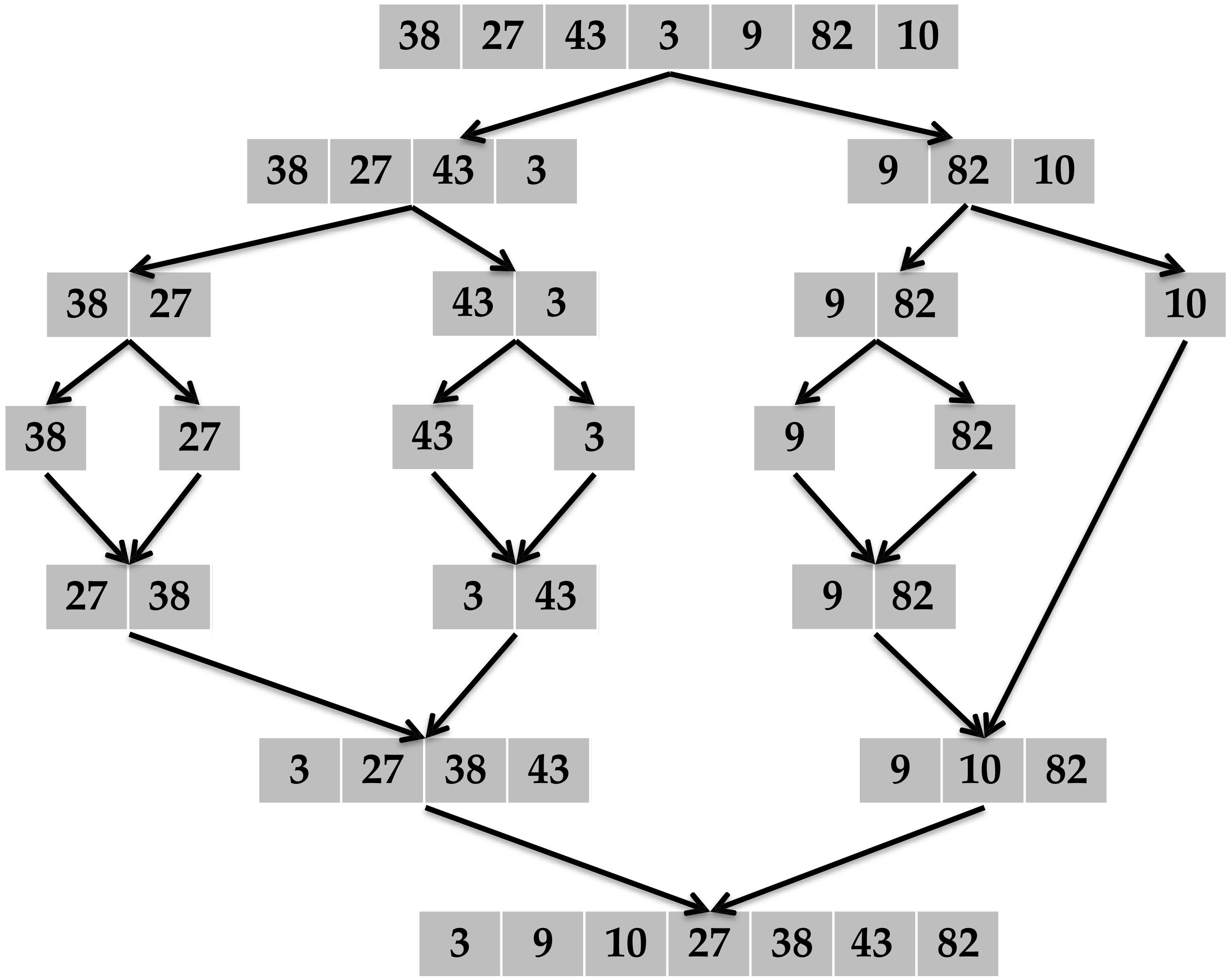
Triedenie zlučovaním (Merge sort)

6 5 3 1 8 7 2 4

Ukážka triedenia zlučovaním hracích kariet



Priebeh rekurzívnych volaní (Merge sort)



Shellsort

- Usporadúvanie vkladaním so zmenšovaním prírastku
- Zovšeobecnenie triedenia vkladaním (Insert sort) a bublinkového (Bubble sort)
- Dobrá implementácia je jedna z najrýchlejších pre usporiadanie kratších postupností (do 1000 prvkov)
- Netriedi naraz celú postupnosť, ale pre prírastok h utriedi Insert sort-om vybranú podpostupnosť prvkov vzdialených h (pre všetky možné začiatky i):

```
for(h = n/2; h > 0; h = h/2) // zmensujuce sa prirastky
  for (i = 0; i < h; i++)
    insert_sort(a[i,i+h,i+2*h,...]);
```

- Postupnosť zmenšujúcich sa prírastkov, posledný h=1

Shellsort - príklad

1. krok, prírastok 4 (n/2),
(vyznačené čísla sa usporiadajú vkladaním)

6 4 5 2 **8** 3 1 7 –> **6** 4 5 2 **8** 3 1 7
6 **4** 5 2 8 **3** 1 7 –> 6 **3** 5 2 8 **4** 1 7
6 3 **5** 2 8 4 **1** 7 –> 6 3 **1** 2 8 4 **5** 7
6 3 1 **2** 8 4 5 **7** –> 6 3 1 **2** 8 4 5 **7**

2. krok, prírastok 2

6 3 **1** 2 **8** 4 **5** 7 –> **1** 3 **5** 2 **6** 4 **8** 7
1 **3** 5 **2** 6 **4** 8 **7** –> 1 **2** 5 **3** 6 **4** 8 **7**

3. krok, prírastok 1

1 2 5 3 6 4 8 7 –> 1 2 3 4 5 6 7 8

Rýchle usporiadanie (Quicksort)

- Quicksort alebo usporadúvanie rozdeľovaním je jeden z najrýchlejších známych algoritmov založených na porovnávaní prvkov
- Priemerná doba výpočtu Quicksort-u je najlepšia zo všetkých podobných algoritmov
- Nevýhodou je, že pri nevhodnom usporiadaní vstupných dát môže byť časová aj pamäťová náročnosť omnoho väčšia
  - Quicksort – počet porovnaní v najhoršom prípade $O(N^2)$
  - Mergesort – počet porovnaní v najhoršom prípade $O(N \log N)$
  - V praxi je však Quicksort zvyčajne dva krát rýchlejší a používa polovičné množstvo pamäti...

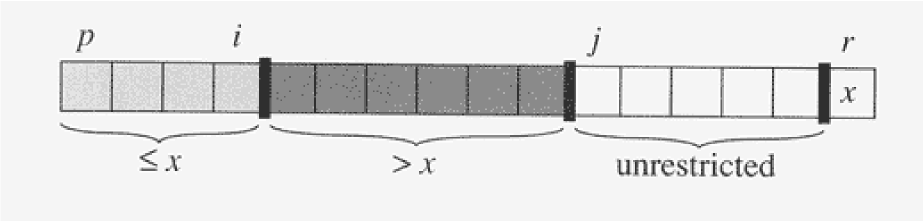
Quicksort - hlavná myšlienka

- Jeden prechod = rozčlenenie prvkov na dve podpostupnosti podľa pivota x: prvky $\leq$ x, prvky > x
- Rekurzívne usporiadať podpostupnosti
- Lomuto schéma

```
QUICKSORT(A, p, r)
if p < r
   then q ← PARTITION(A, p, r)
        QUICKSORT(A, p, q − 1)
        QUICKSORT(A, q + 1, r)
```

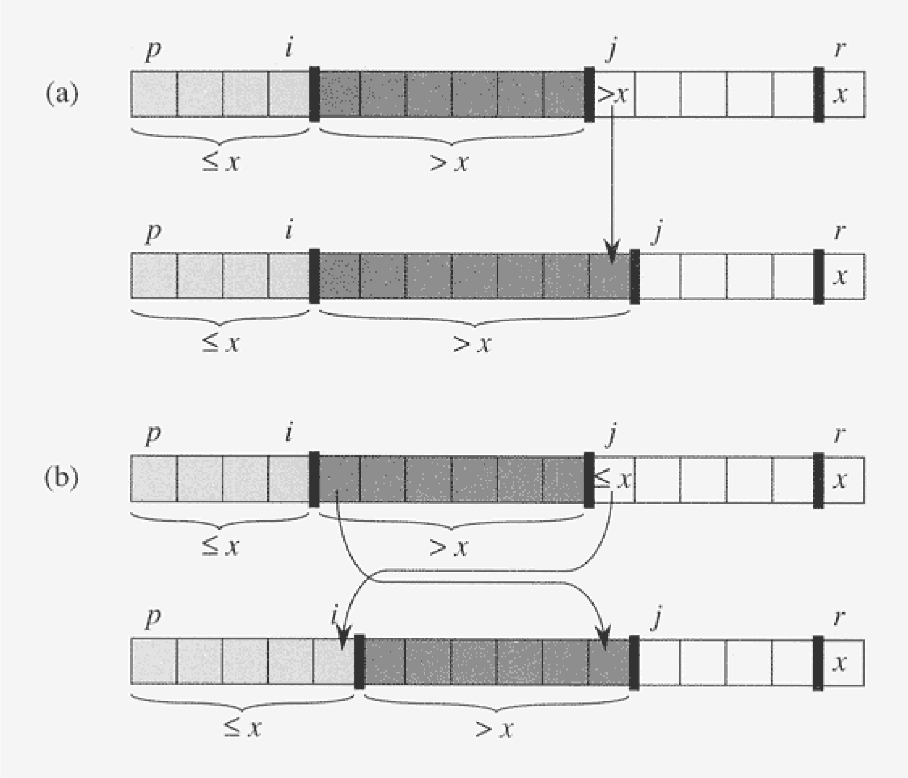
```
PARTITION(A, p, r)
x ← A[r]
i ← p − 1
for j ← p to r − 1
     do if A[j] ≤ x
           then i ← i + 1
                exchange A[i] ↔ A[j]
exchange A[i + 1] ↔ A[r]
return i + 1
```

Quicksort – rozčlenenie

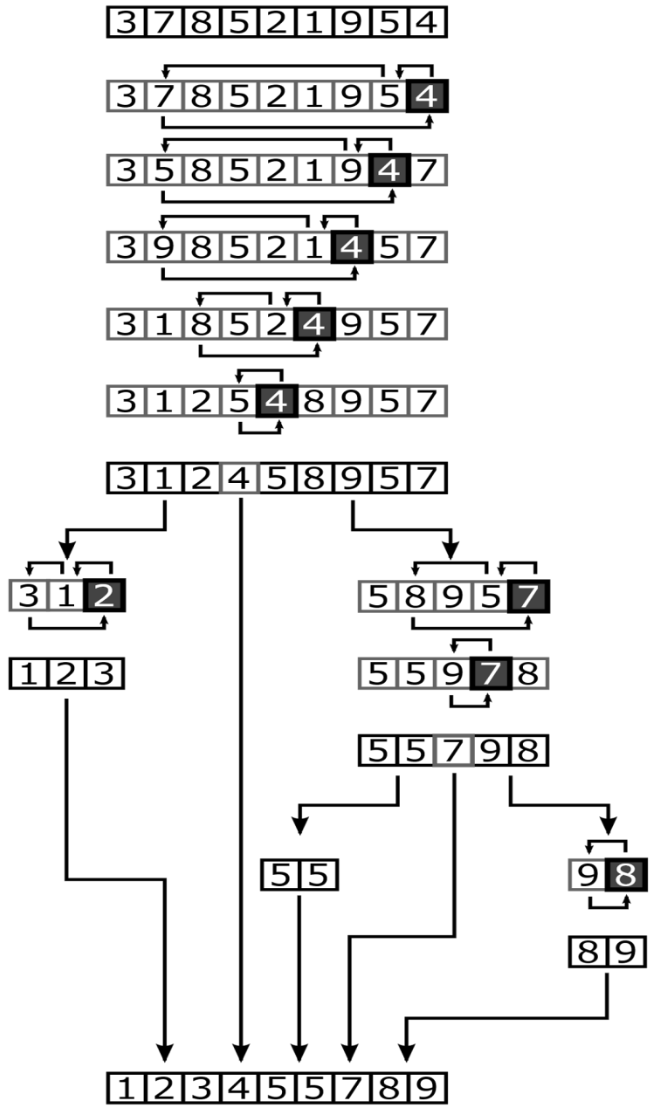


- Prvky A[p..i] sú menšie alebo rovné x
- Prvky A[i+1..j-1] sú väčšie x
- Prvky A[j..r-1] sú ešte nerozčlenené

- Pri rozčleňovaní ďalšieho prvku (j) môžu nastať dva prípady:
  a. A[j] > x, len posuniem j
  b. A[j] ≤ x, presuniem prvok na i-tu pozíciu, posuniem i a j



Quicksort – príklad

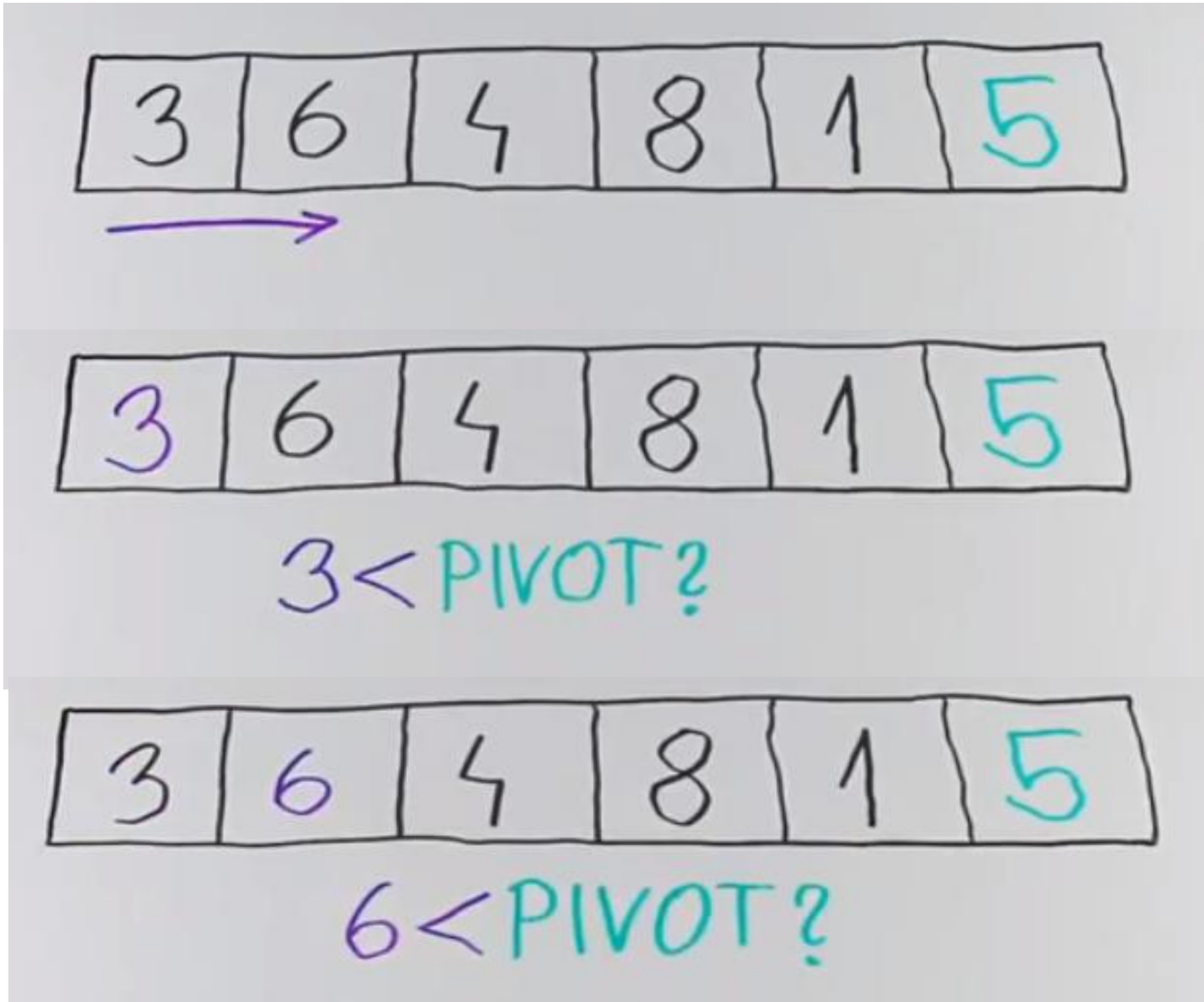


Quicksort - Hoare schéma

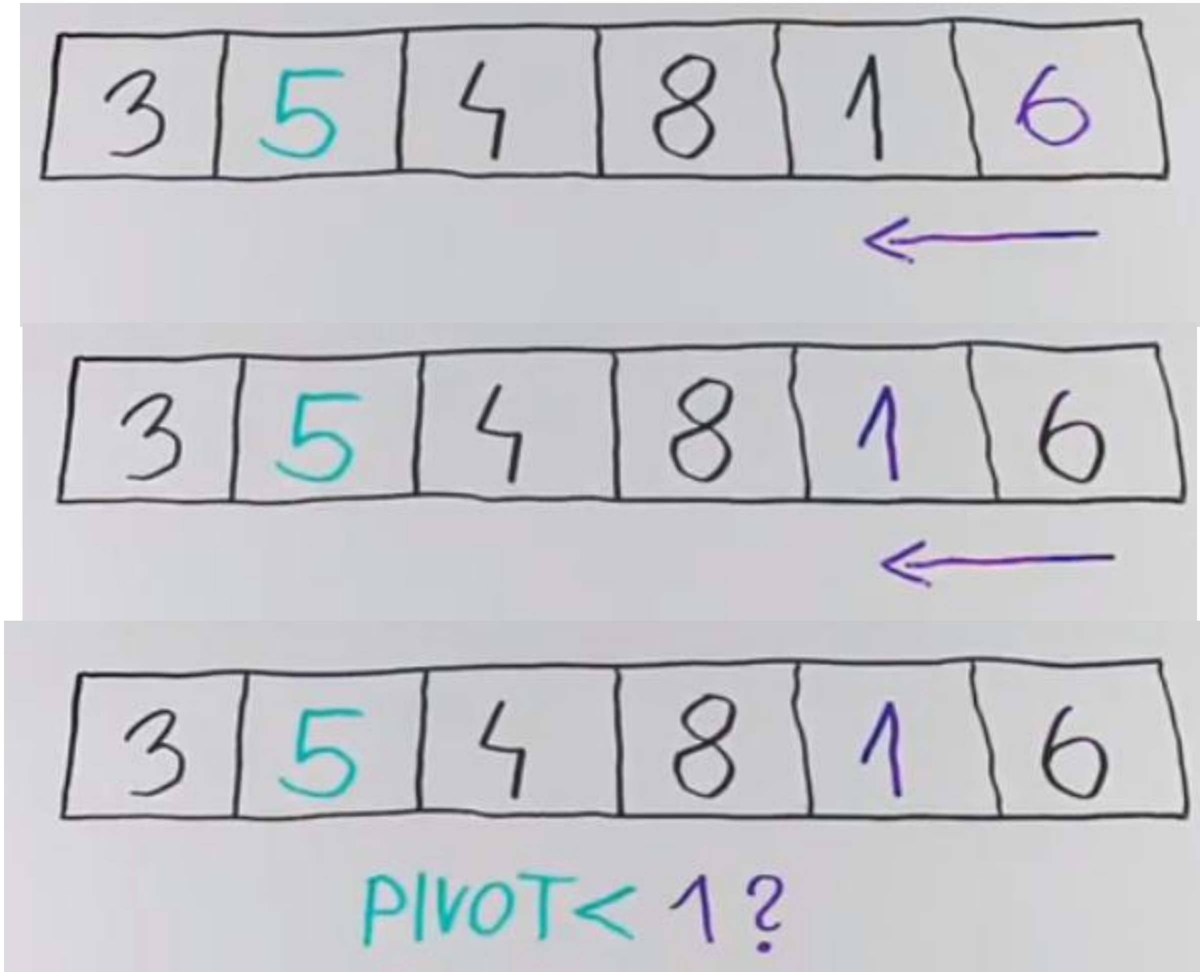
```
algorithm quicksort(A, lo, hi) is
    if lo < hi then
        p := partition(A, lo, hi)
        quicksort(A, lo, p)
        quicksort(A, p + 1, hi)

algorithm partition(A, lo, hi) is
    pivot := A[⌊(hi + lo) / 2⌋]
    i := lo - 1
    j := hi + 1
    loop forever
        do
            i := i + 1
        while A[i] < pivot
        do
            j := j - 1
        while A[j] > pivot
        if i ≥ j then
            return j
        swap A[i] with A[j]
```

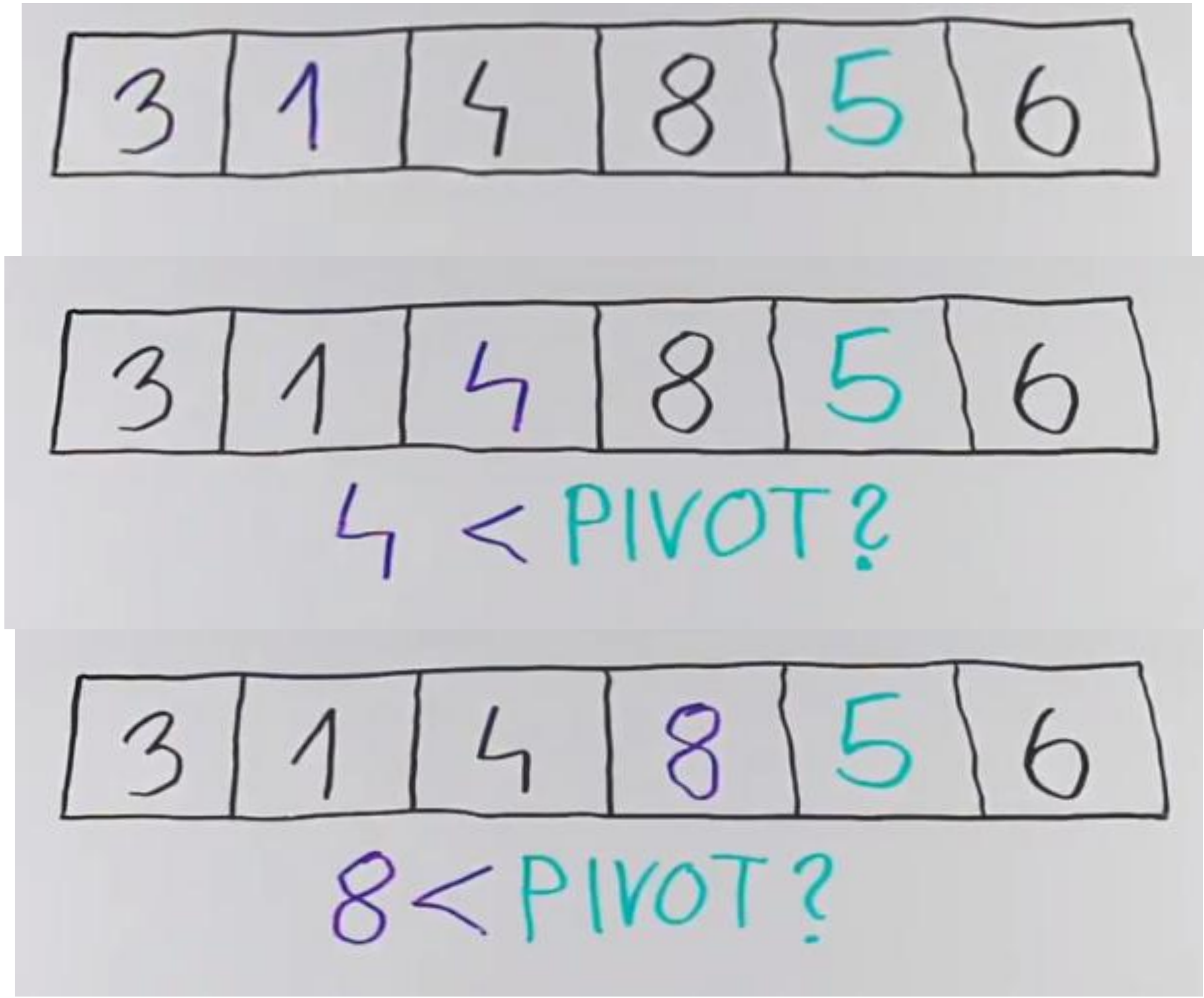
Quicksort – Hoare schéma – príklad



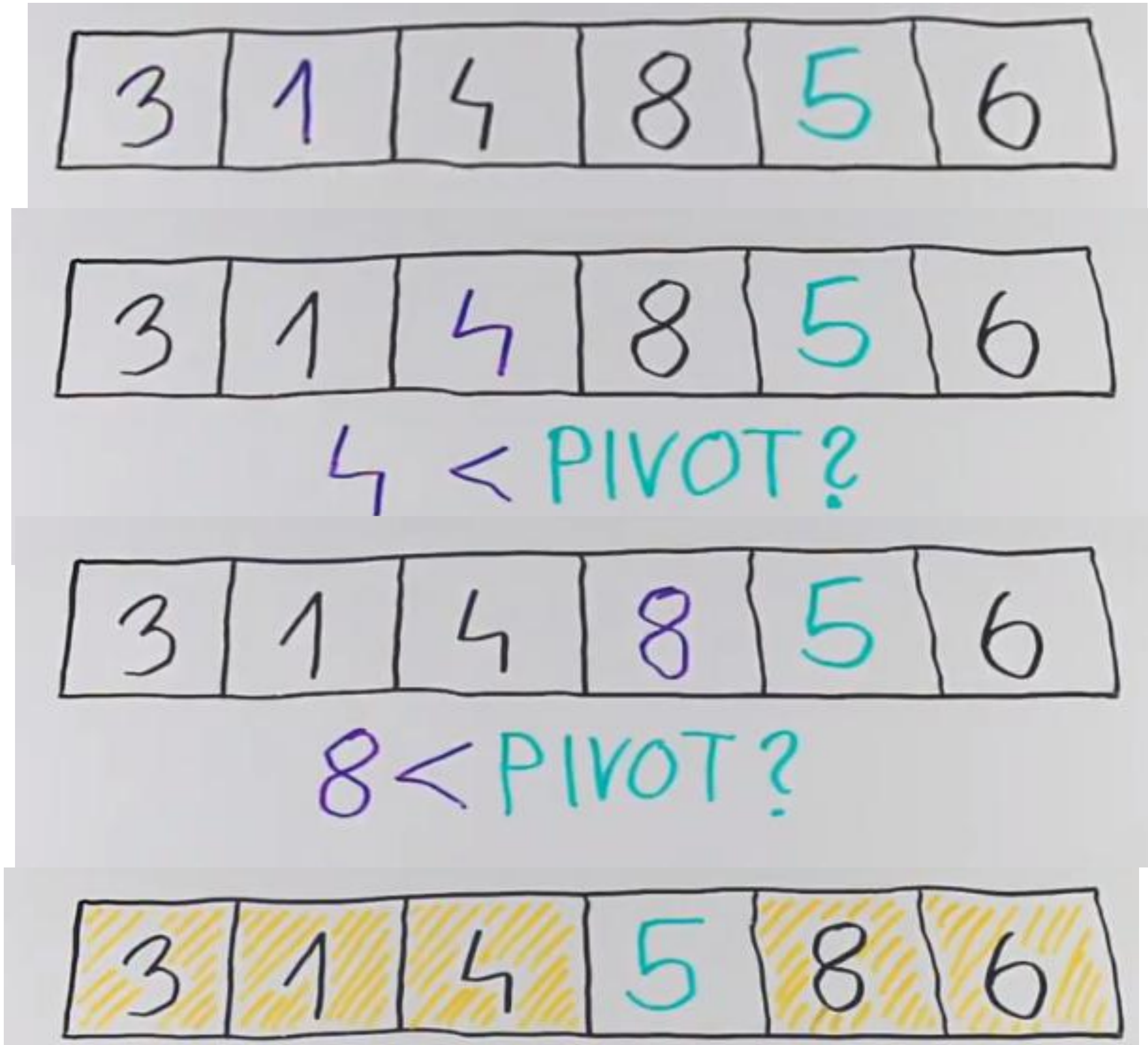
Quicksort – Hoare schéma – príklad



Quicksort – Hoare schéma – príklad



Quicksort – Hoare schéma – príklad



Analýza rýchleho usporiadania

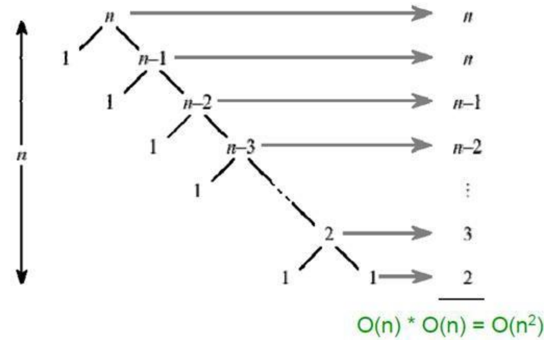
- Najhorší prípad: vždy zle vyvážené rozčlenenie

  Usporiadaná postupnosť :(

  $T(1) = \Theta(1)$

  $T(n) = T(n - 1) + \Theta(n)$

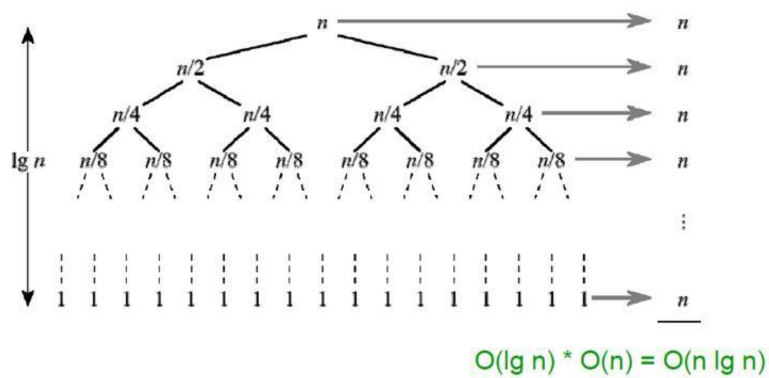
  $T(n) = \Theta(n^2)$



- Najlepší prípad: vždy dokonale vyvážené rozčlenenie

  $T(n) = 2T(n/2) + \Theta(n)$

  $T(n) = \Theta(n \lg n)$



- Priemerný prípad: náhodný

  Napr. aj pre rozčlenenie 9 ku 1

  $T(n) = T(9n/10) + T(n/10) + n$

  $T(n) = \Theta(n \lg n)$

Výber pivota

- Krajný prvok – konštantný čas O(1)
  nemusí byť vhodný
- Ktorý by bol najlepší?
  - Taký, pre ktorý je počet prvkov rozčlenených podpostupností rovnaký (alebo čo najbližšie k sebe)
  - **Medián**
- Dobrý algoritmus: **vybrať náhodný pivot**

```
RANDOMIZED-PARTITION(A, p, r)
i ← RANDOM(p, r)
exchange A[r] ↔ A[i]
return PARTITION(A, p, r)
```

Nájdenie k-teho najmenšieho prvku

- Daná postupnosť A[1..n] (neusporiadaných) čísel a celé číslo k ($1 \leq k \leq n$). Úloha je nájsť k-te najmenšie číslo v A.
- Špeciálne prípady
  - pre k=1 ide o nájdenie najmenšieho prvku,
  - pre k=n ide o nájdenie najväčšieho prvku,
  - ak n je nepárne, k=(n+1)/2 dá medián
  - ak n je párne, podľa dohody je medián niečo medzi prípadmi k=floor((n+1)/2) a k=ceiling((n+1)/2)

Nájdenie k-teho najmenšieho prvku

- Prvý nápad na riešenie:
  usporiadať pole A a vybrať A[k]
  - usporiadať vieme na mieste O(n log n). Výber A[k] je O(1). Spolu O(n log n).
- Dá sa to rýchlejšie?
  - Ak máme už dané k, tak usporiadaním sa urobilo viac práce ako je potrebné na určenie k-teho najmenšieho prvku. Prečo?
  - Ak by k nebolo vopred dané, tak by práve usporiadané pole dávalo k-ty najmenší prvok pre ľubovoľné k.

Nájdenie k-teho najmenšieho prvku

- Druhý nápad:
  nájsť najmenší prvok v poli A a odstrániť ho. Pokračovať nájdením najmenšieho prvku vo zvyšku poľa A, odstránením atď. Opakovať k-krát.
  - nájsť minimum a odstrániť ho vieme O(n).
    Spolu O(k*n).
- Je to rýchlejšie?
  - závisí od porovnania k a log(n).
  - pre veľké k (väčšie ako log n) je lepší prvý nápad
  - pre malé k je druhý nápad lepší

Nájdenie k-teho najmenšieho prvku

- Dá sa to rýchlejšie?
  - všimnime si, že v prvom aj druhom prípade dostaneme na výstupe **k najmenších prvkov usporiadaných**.
  - Ale toto (usporiadanie) týchto k prvkov nepotrebujeme! Robíme robotu navyše.
  - Výzva je určiť LEN k-ty prvok a urobiť to v čase O(n).

- Blum, Floyd, Pratt, Rivest a Tarjan, 1973
  Algoritmus s mediánom mediánov ako pivotom:

```
Select (A, k)
1.  x = median(A) //akurát, že zatiaľ nevieme ako v O(n)
2.  rozčleň A podľa pivota x. Nech je m-1 prvkov takých, že A[i]<x.
    Potom bude A[m]=x a n-m prvkov bude takých, že A[i]>x.
3.  if k=m then return x
                else if k<m then Select (A[1..m-1], k)
                        else Select (A[m+1..n], k-m)
```

Algoritmus medián mediánov – zložitosť

$T_{select}(n) = T_{select}(n/2) + n + T_{\mathrm{median}}(A)$

- predpokladajme, že $T_{\mathrm{median}}(A)$ je $O(n)$ a preto

  $T_{select}(n) = T_{select}(n/2) + n$

- riešenie je n+n/2+n/4+…+1 = 2n-1

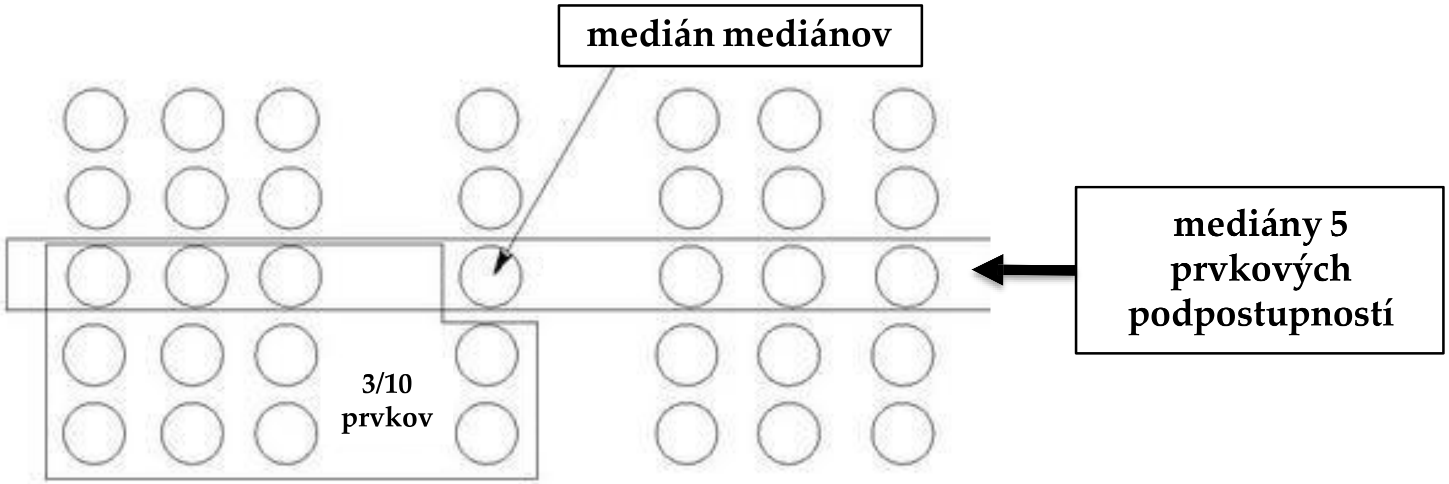
Celkovo $O(n)$

- Predpokladáme, že $T_{\mathrm{median}}(A)$ je $O(n)$ a teda nám stačí približný medián, napr. taký, ktorý je zaručene väčší než 3/10 všetkých prvkov a menší než 3/10 všetkých prvkov. Presnejšie, uvažujme približný medián taký, že je x-tý najmenší zo všetkých prvkov v A a platí

$$3n/10 \leq x \leq 7n/10$$

Medián mediánom z piatich

1. Rozdeľ vstupnú postupnosť n prvkov do skupín po piatich (a možno jednej zvyškovej)
2. Nájdi medián každej skupiny (usporiadaním alebo natvrdo tretí najmenší) – dostaneš n/5 mediánov.
3. Rekurzívne Select(„n/5 mediánov“, n/10)



Zložitosť: $T_{select}(n) = T_{select}(n/5) + T_{select}(7n/10) + n$ – *Celkovo O(n)*

Triedenie binárnym vyhľadávacím stromom

- Štruktúra vrcholov v BVS umožňuje skonštruovať jednoduchý algoritmus usporadúvania tzv. **Treesort**
- In-order prehľadávanie – usporiadaný výpis obsahu BVS
  - Zložitosť O(n), kde n je počet prvkov v strome

- Máme teda porovnávací algoritmus, ktorý dokáže usporiadať n čísel na vstupe rýchlejšie ako O(n log n)?
  - Nie, pretože vytvorenie BVS trvá O(n log n)
  - Všimnime si, že štruktúra prvkov v BVS je „ekvivalentná“ samotnému problému usporiadania, pretože keď už máme BVS, tak dokážeme výsledné poradie získať v O(n)

Ďakujem za pozornosť