Welcome To BigBlueButton

BigBlueButton is an open source web conferencing system designed for online learning

CHAT

Send public and private messages.

WEBCAMS

Hold visual meetings.

AUDIO

Communicate using high quality audio.

EMOJIS

Express yourself.

BREAKOUT ROOMS

Group users into breakout rooms for team collaboration.

POLLING

Poll your users anytime.

SCREEN SHARING

Share your screen.

MULTI-USER WHITEBOARD

Draw together.

For more information visit bigbluebutton.org →

Algoritmizace

Marko Genyg-Berezovskyj, Daniel Průša

2010 - 2020

Začátek v 11:00

Přednáší: Daniel Průša

prusa@fel.cvut.cz

Přehled

- Merge sort, binární halda, Heap sort
- Postřehová rozcvička
- Radix sort, Counting sort
- Zadání HW_06

Průzkum

Kolik z následujících algoritmů dobře znáte?

Merge sort, Heap sort, Radix sort, Counting sort

A. 4

B. 3

C. 2

D. 1

E. 0

Průzkum

Kolik z následujících algoritmů dobře znáte?

Merge sort, Heap sort, Radix sort, Counting sort

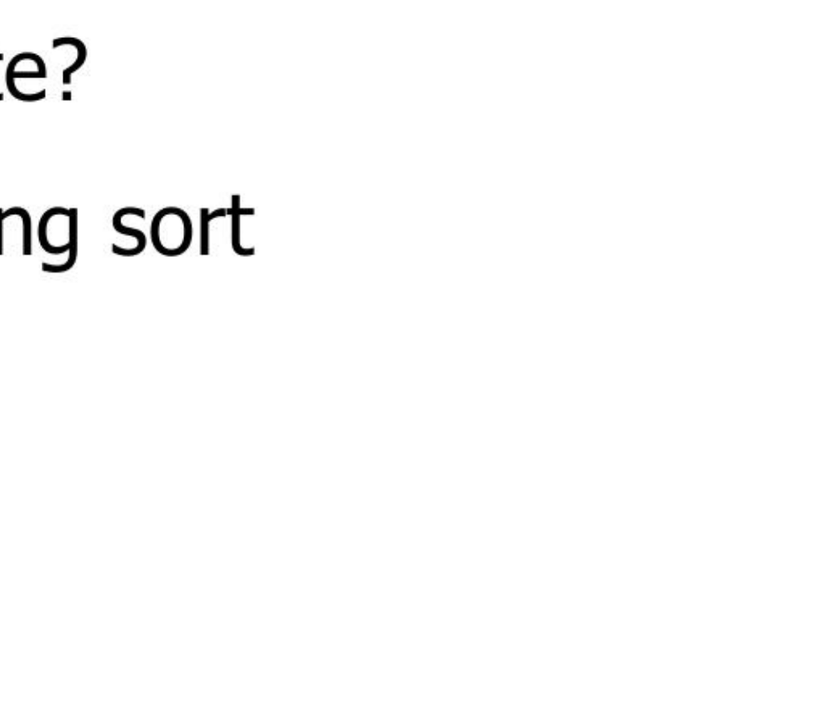
A. 4

B. 3

C. 2

D. 1

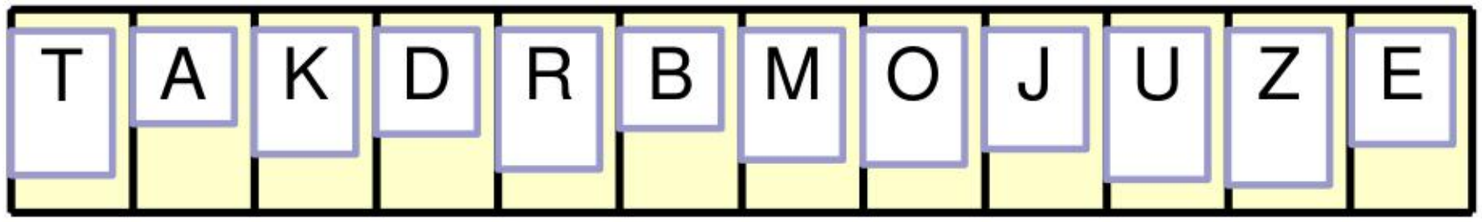
E. 0



Merge sort (řazení slučováním)

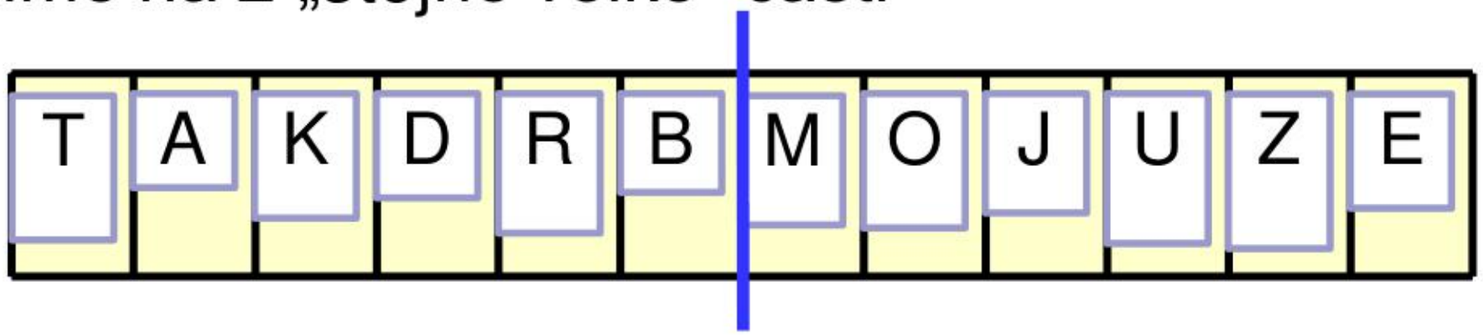
- John von Neumann (1945), technika rozděl a panuj

Vstup



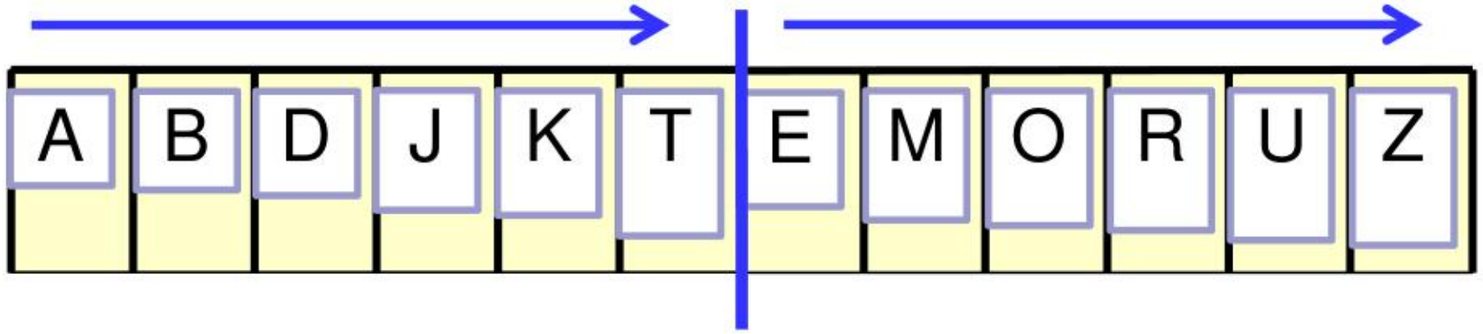
Vstup rozdělíme na 2 „stejně velké“ části

Rozděl



a každou část seřadíme rekurzivně.

2x rekurze

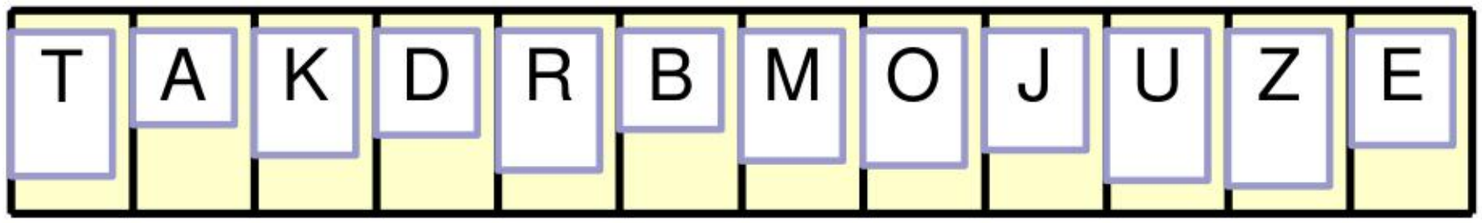


Nakonec obě části sloučíme v lineárním čase.

Merge sort (řazení slučováním)

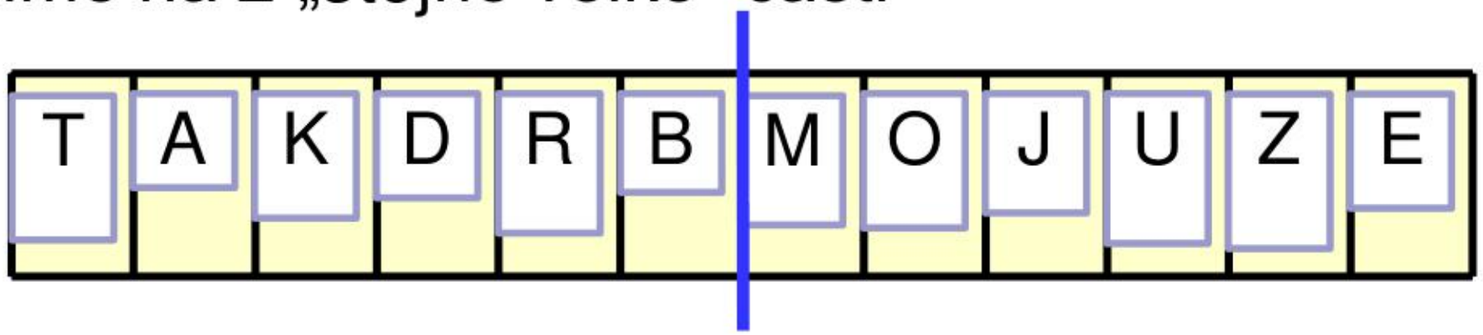
- John von Neumann (1945), technika rozděl a panuj

Vstup



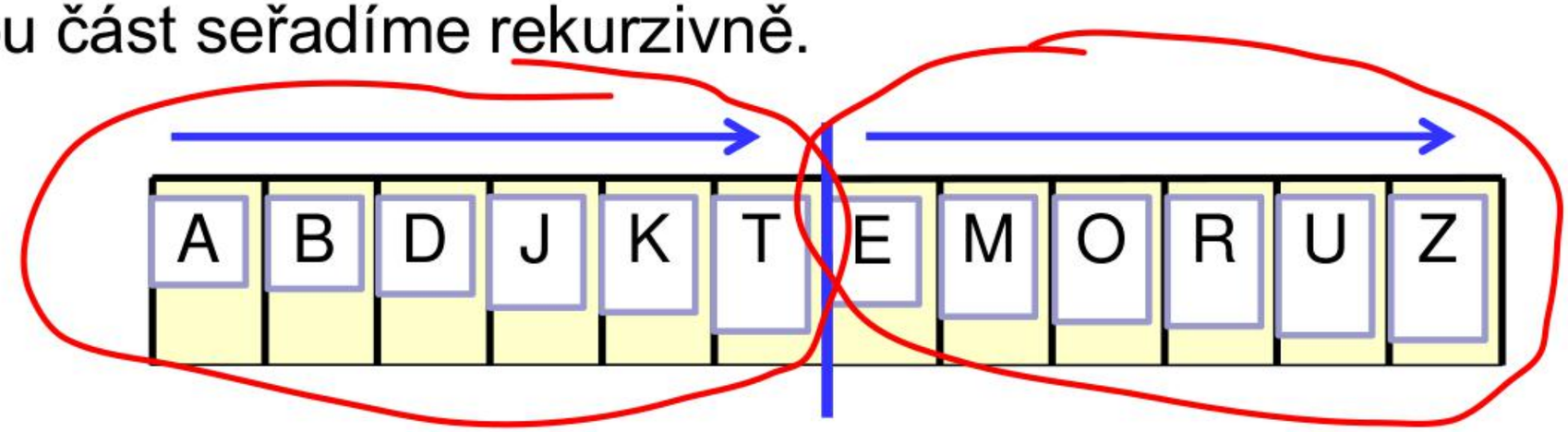
Vstup rozdělíme na 2 „stejně velké“ části

Rozděl



a každou část seřadíme rekurzivně.

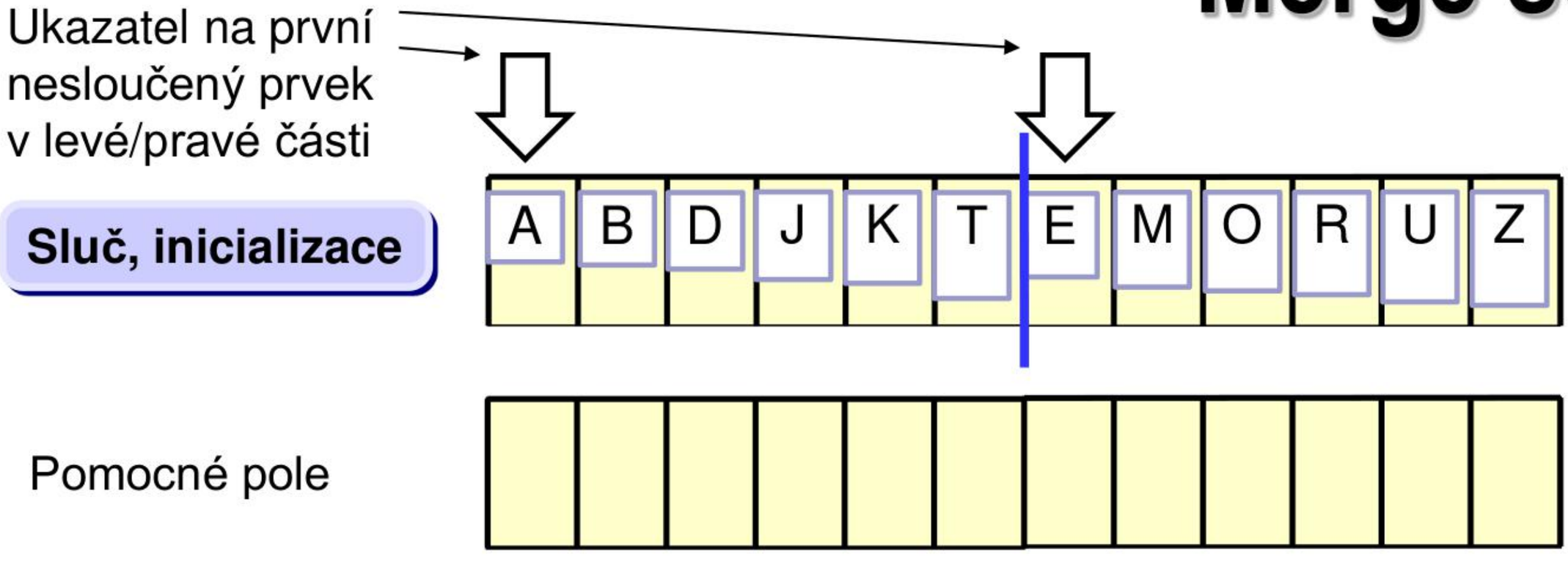
2x rekurze



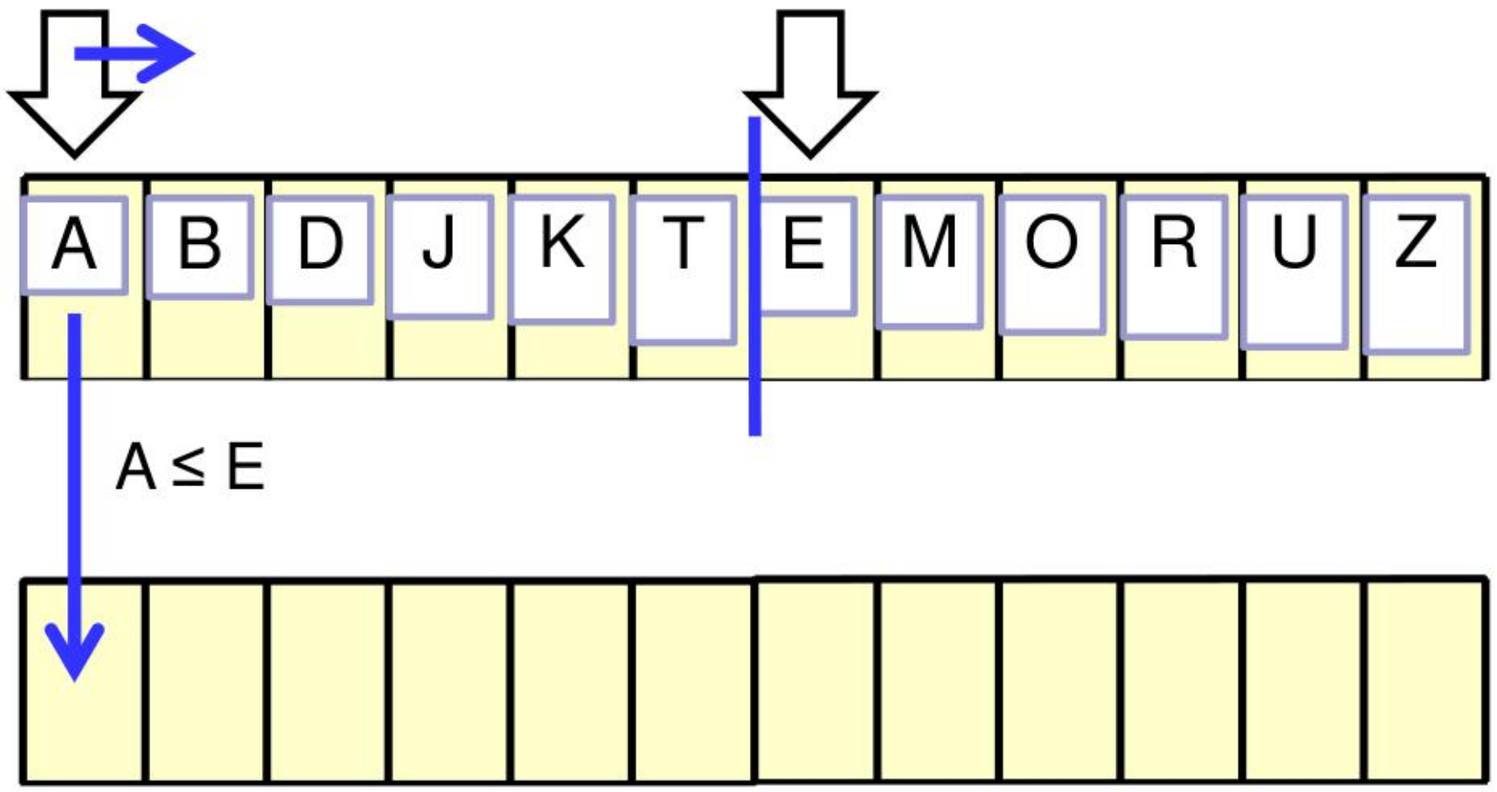
Nakonec obě části sloučíme v lineárním čase.

Merge sort

Sluč, inicializace



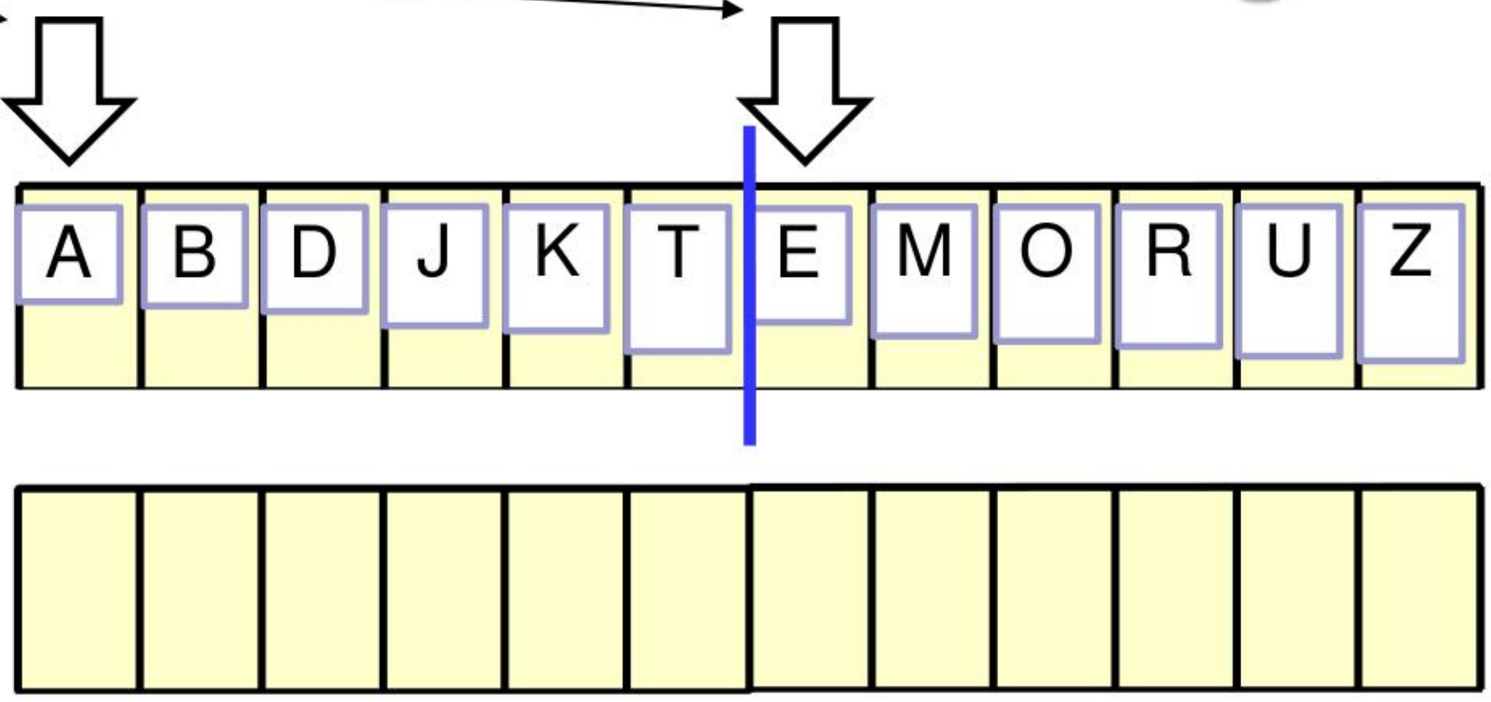
Sluč, 1. krok



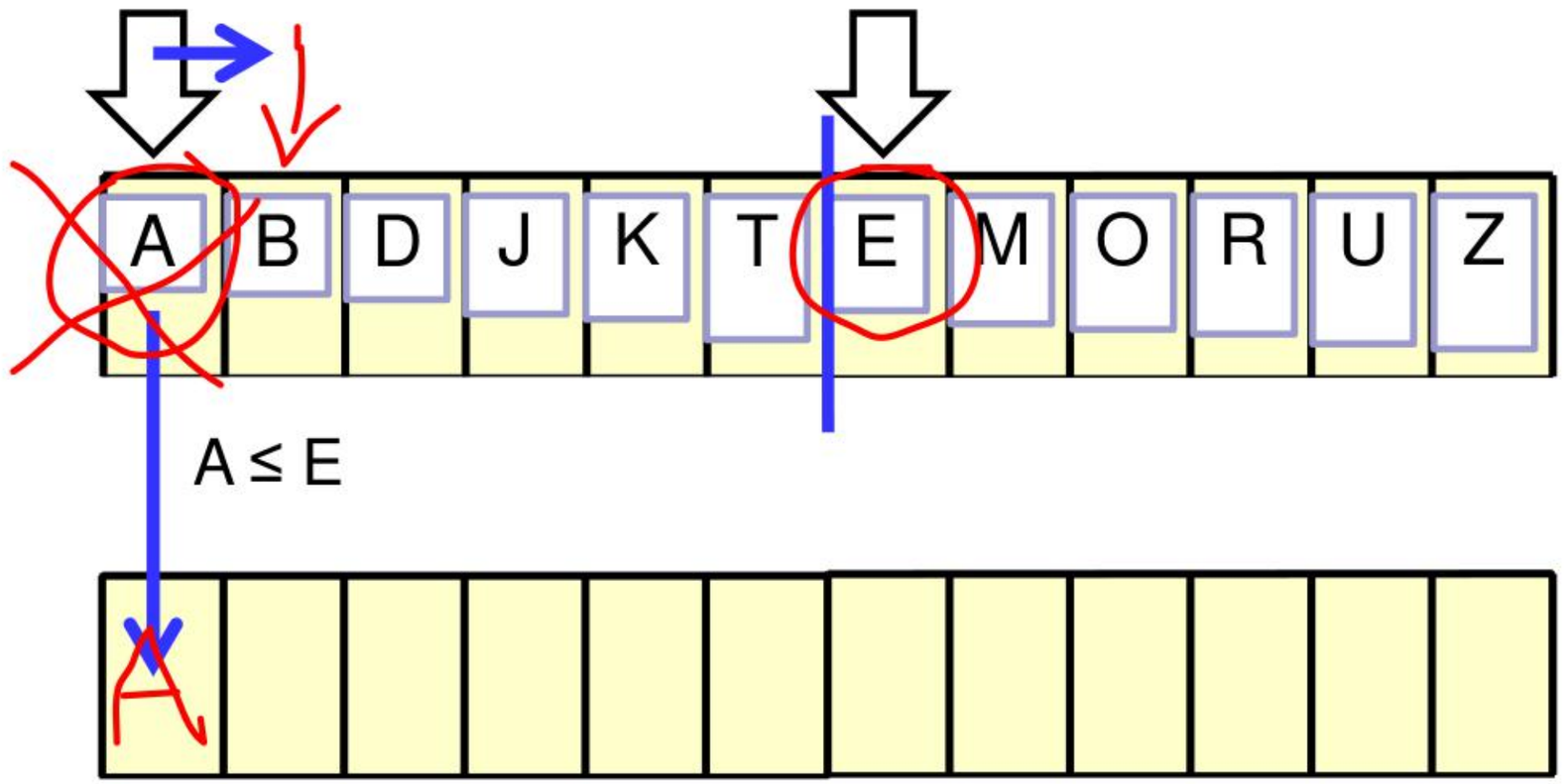
Merge sort

Ukazatel na první nesloučený prvek v levé/pravé části

Sluč, inicializace

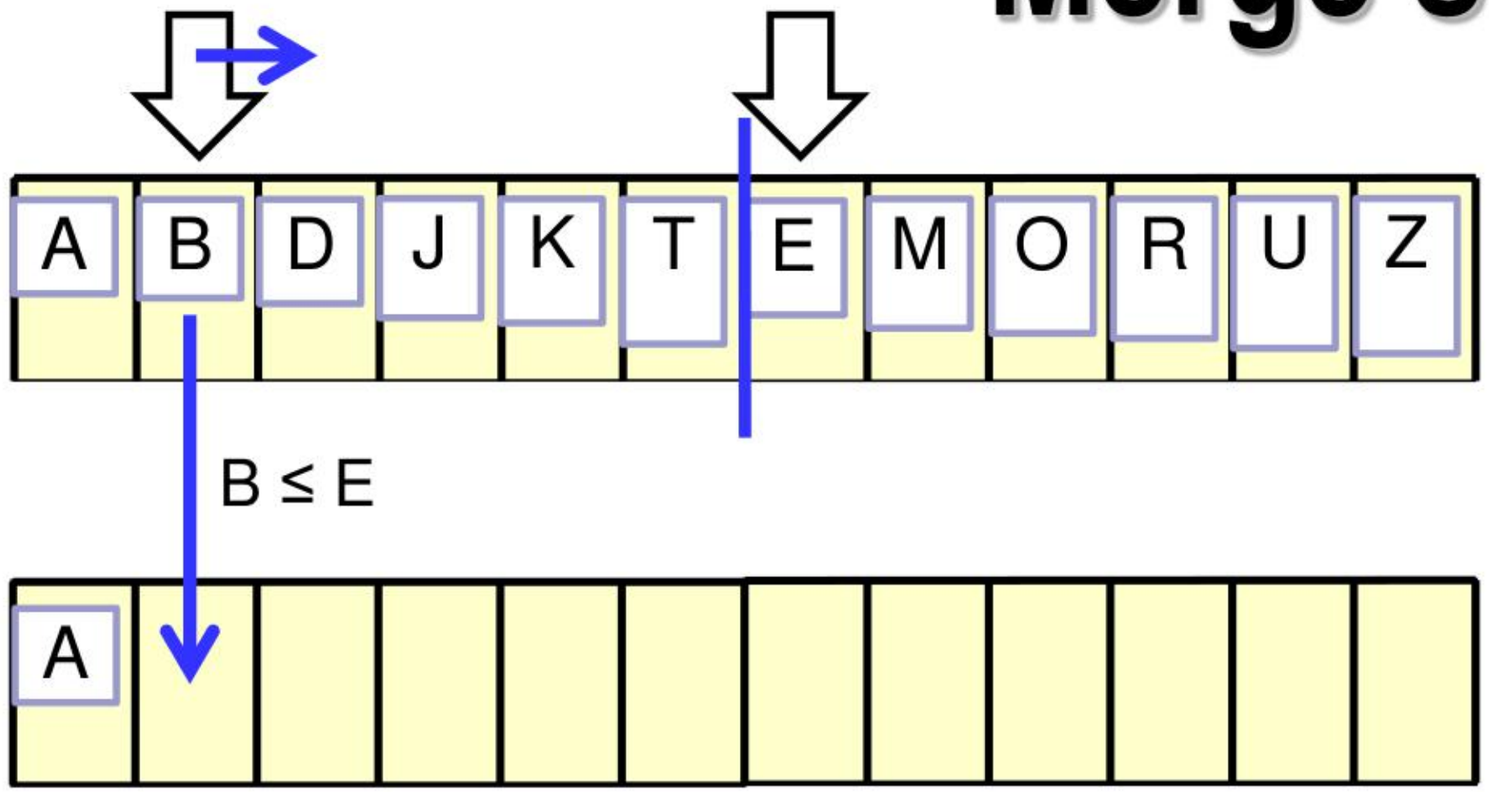


Sluč, 1. krok



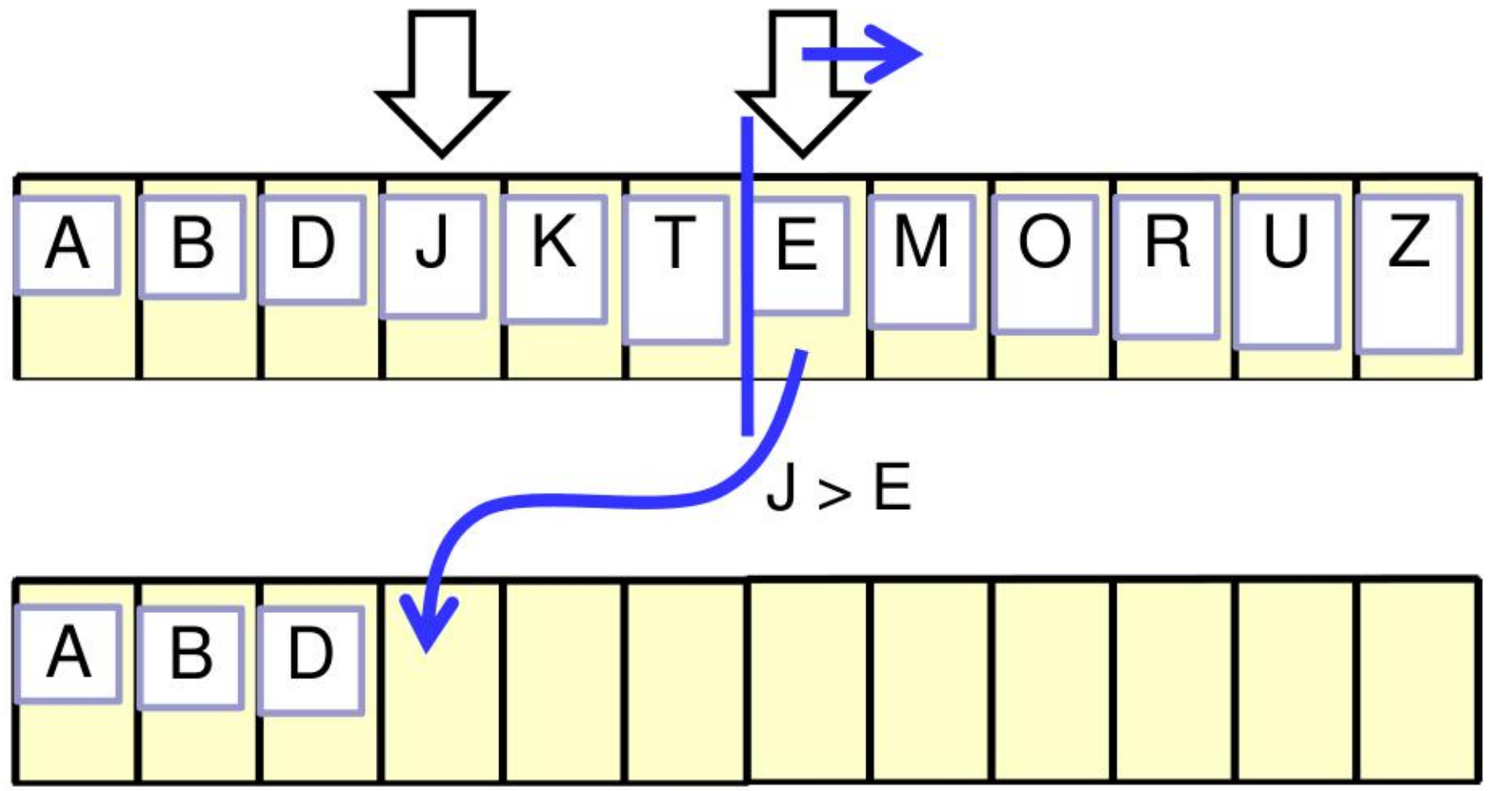
Merge sort

Sluč, 2. krok



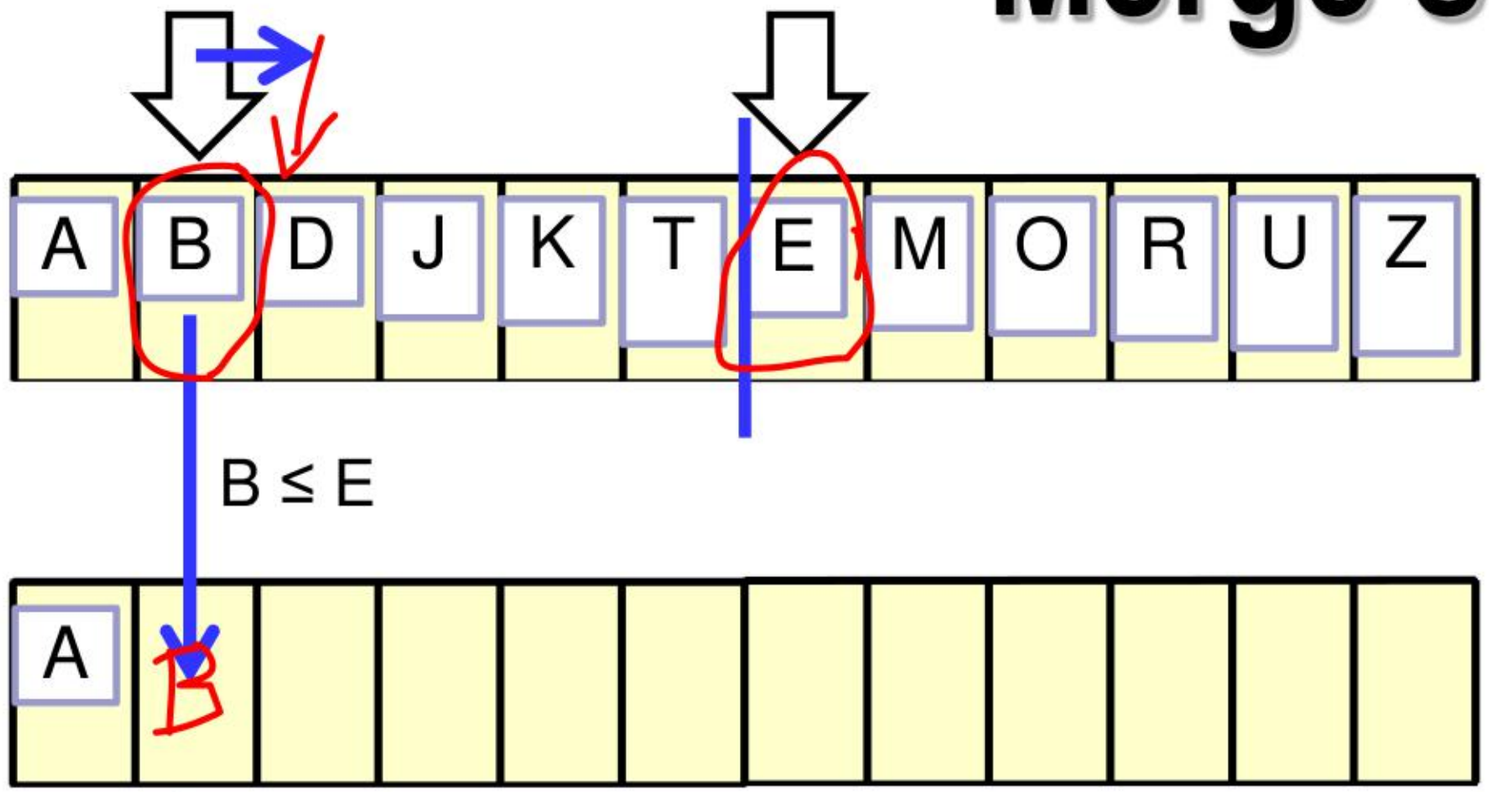
...

Sluč, 4. krok



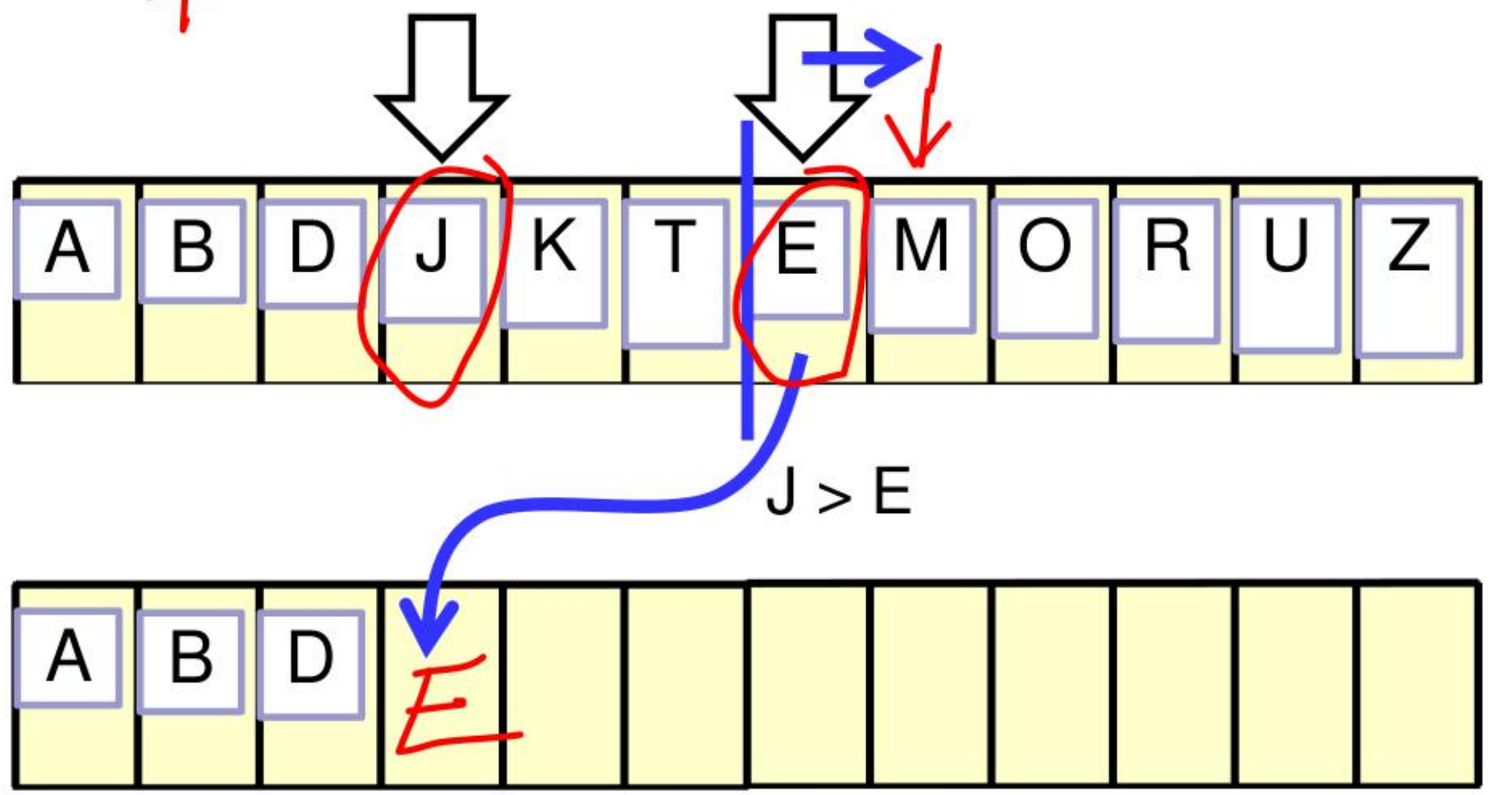
Merge sort

Sluč, 2. krok



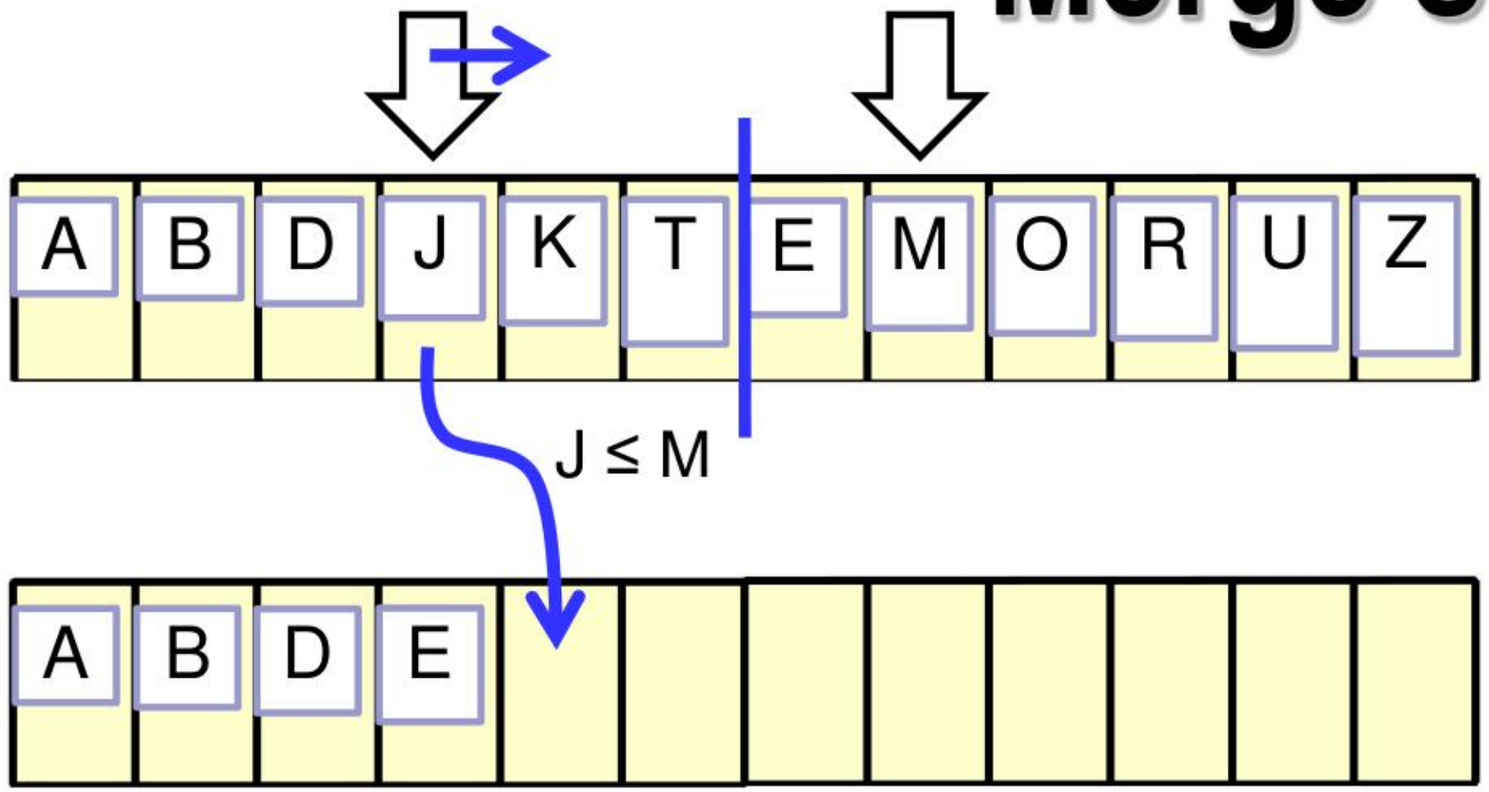
...

Sluč, 4. krok



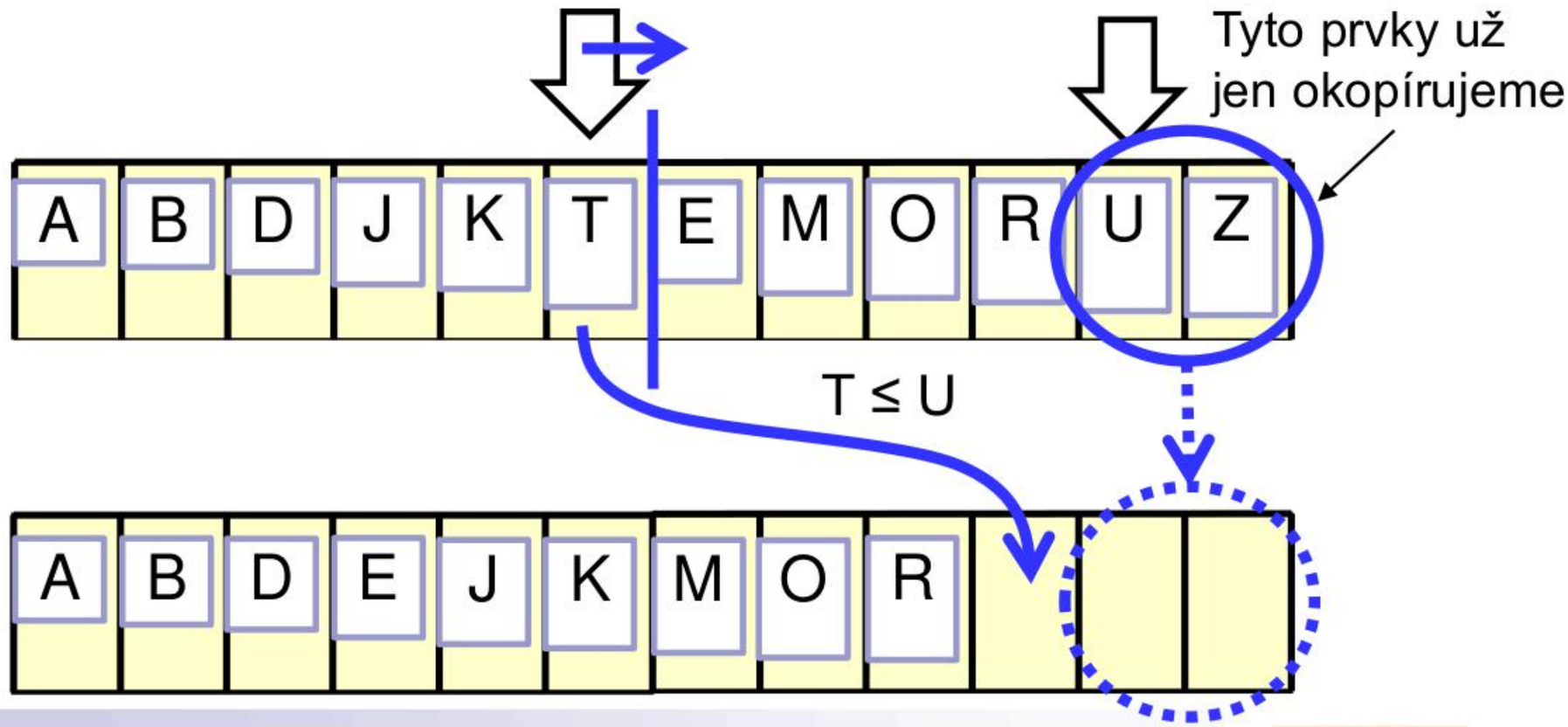
Merge sort

Sluč, 5. krok



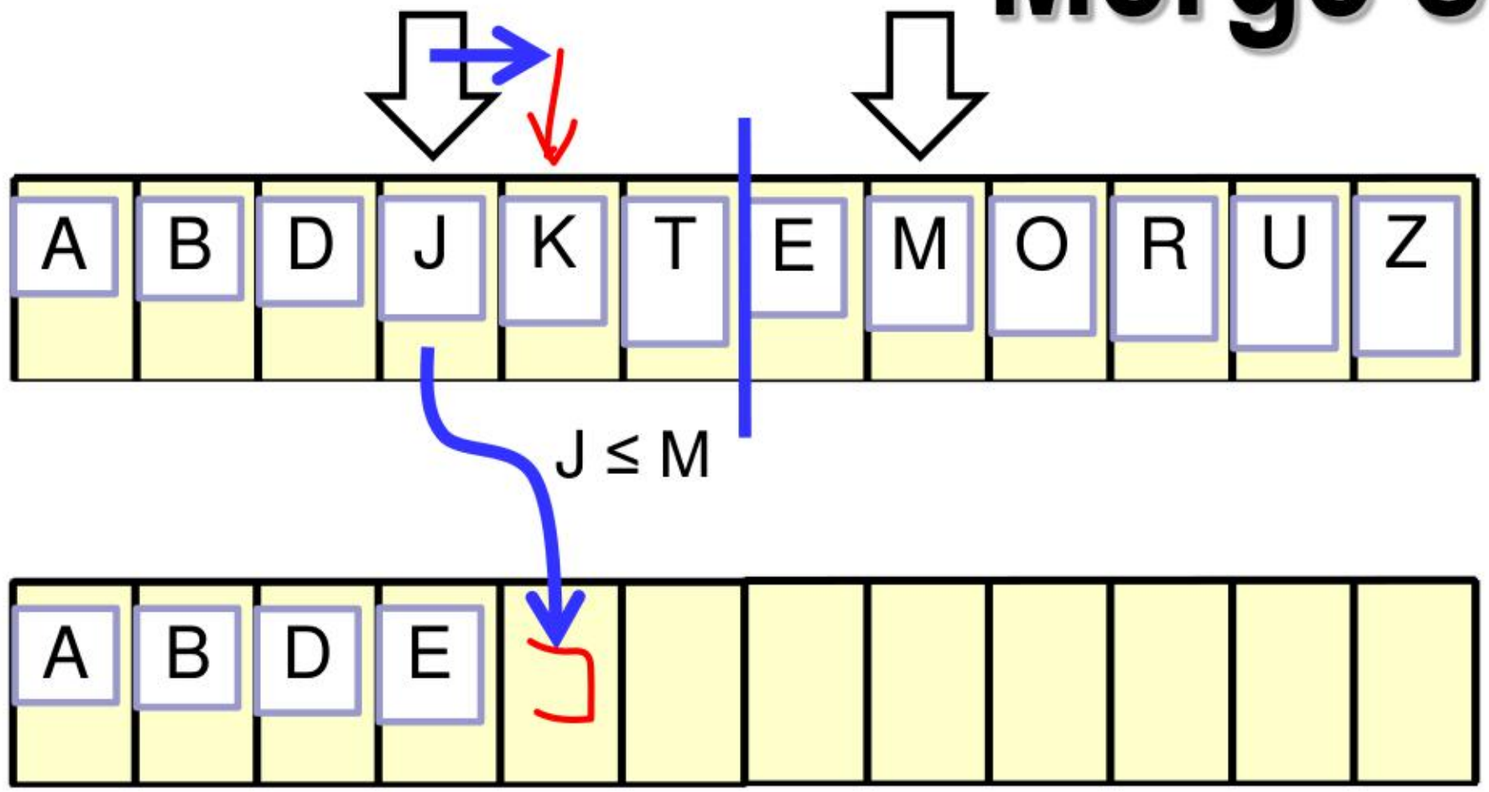
...

Sluč, 10. krok



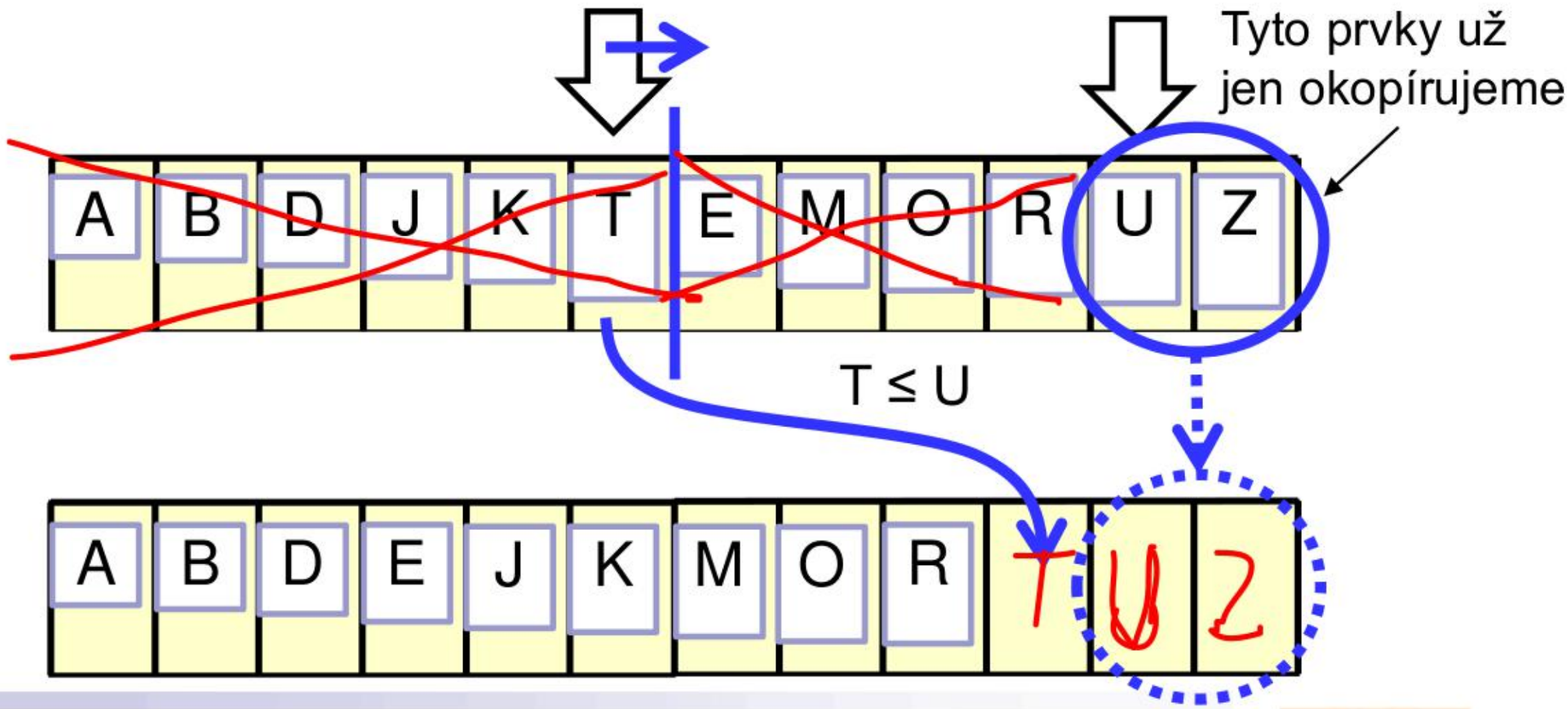
Merge sort

Sluč, 5. krok

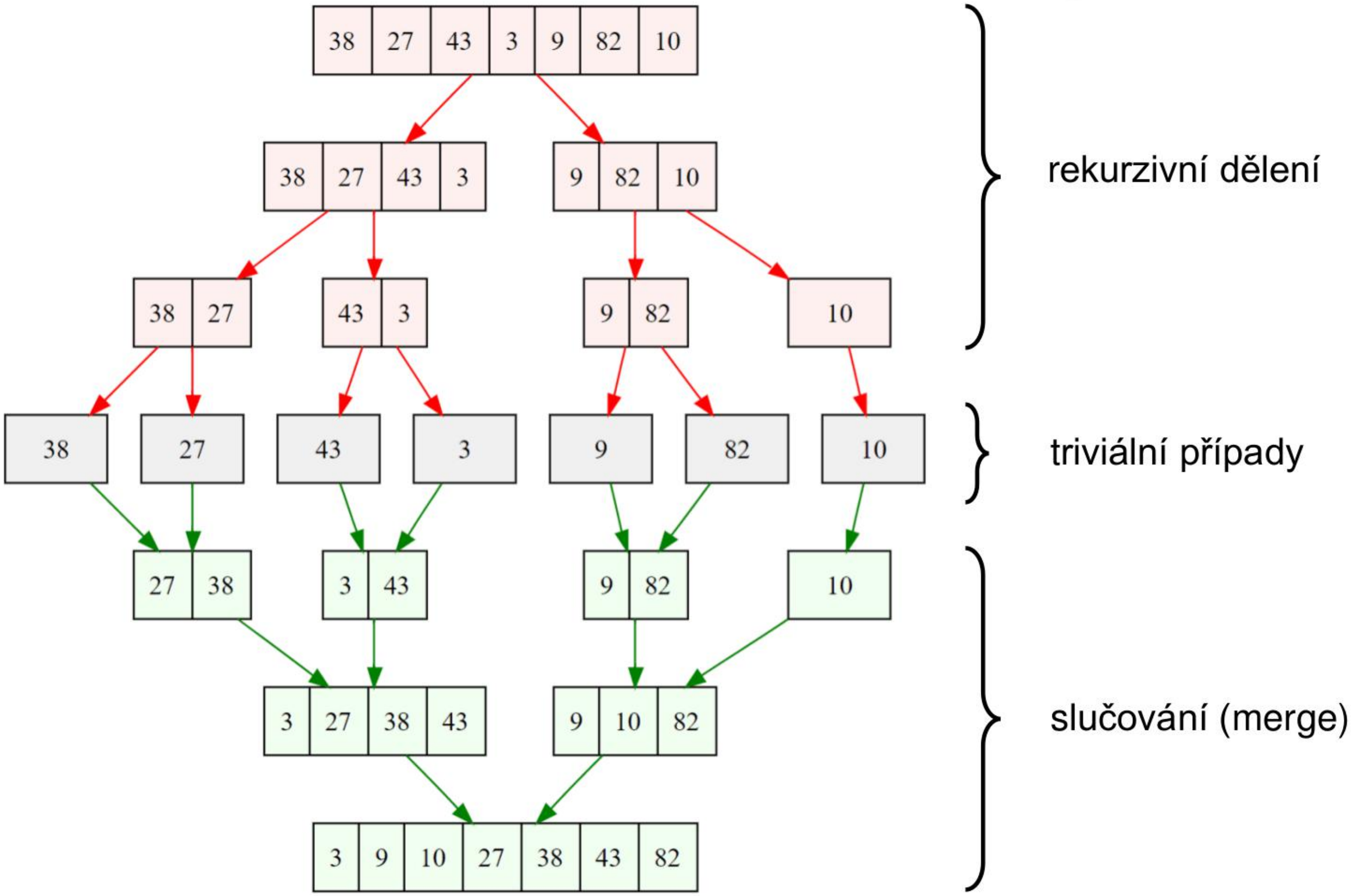


...

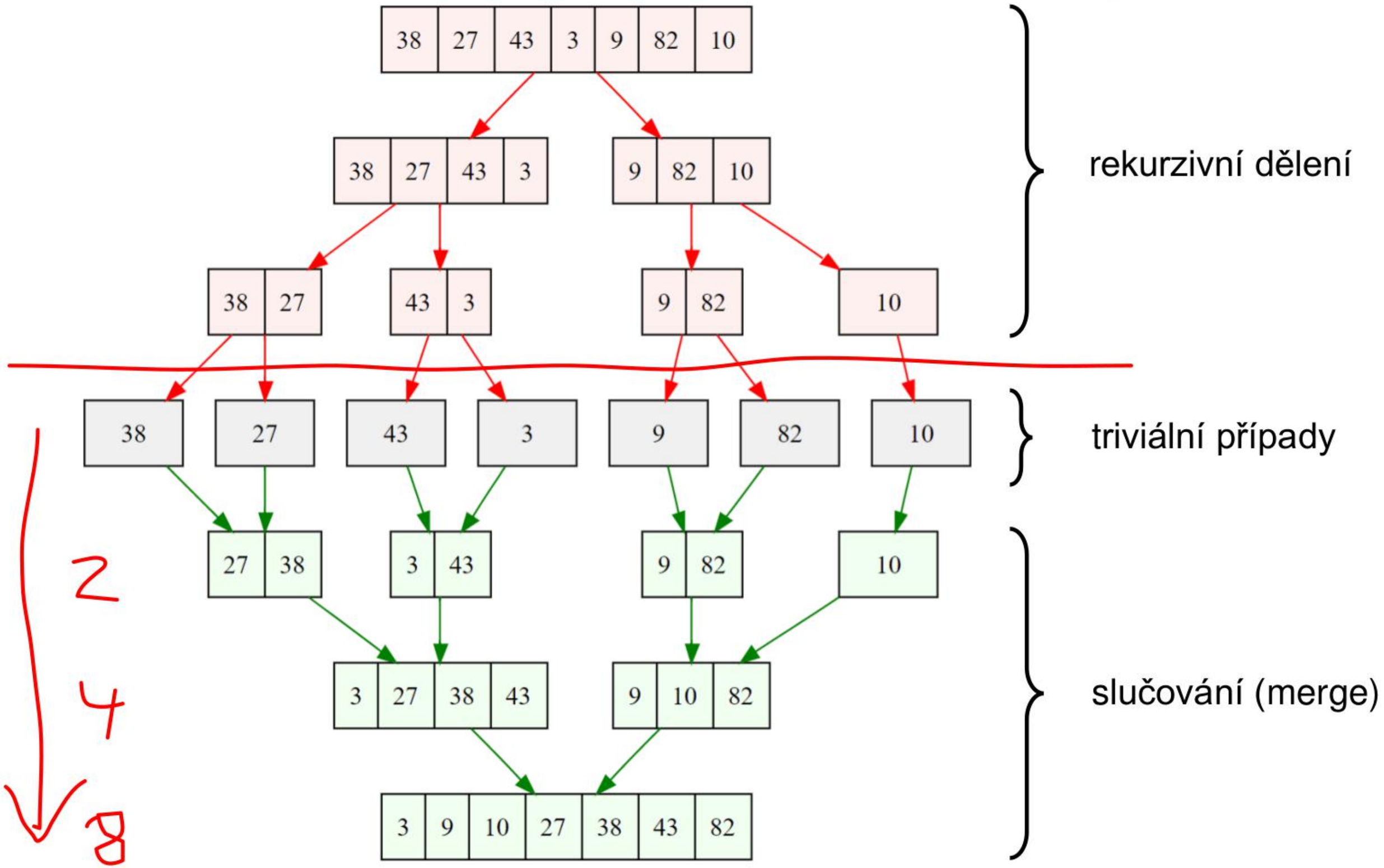
Sluč, 10. krok



Merge sort


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82
rekurzivní dělení
triviální případy
slučování (merge)

Merge sort


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82
2
4
8
rekurzivní dělení
triviální případy
slučování (merge)

Merge sort

```
void merge(int[] in, int[] out, int low, int high) {
  int half = (low+high)/2;
  int i1 = low;
  int i2 = half+1;
  int j = low;

                                    // compare and merge
  while ((i1 <= half) && (i2 <= high)) {
    if (in[i1] <= in[i2]) { out[j] = in[i1]; i1++; }
    else { out[j] = in[i2]; i2++; }
    j++;
  }
                                    // copy the rest
  while (i1 <= half) { out[j] = in[i1]; i1++; j++; }
  while (i2 <= high) { out[j] = in[i2]; i2++; j++; }
}
```

Merge sort

```
void merge(int[] in, int[] out, int low, int high) {
  int half = (low+high)/2;
  int i1 = low;
  int i2 = half+1;
  int j = low;

                                        // compare and merge
  while ((i1 <= half) && (i2 <= high)) {
    if (in[i1] <= in[i2]) { out[j] = in[i1]; i1++; }
    else { out[j] = in[i2]; i2++; }
    j++;
  }
                                        // copy the rest
  while (i1 <= half) { out[j] = in[i1]; i1++; j++; }
  while (i2 <= high) { out[j] = in[i2]; i2++; j++; }
}
```

Merge sort

```
void mergeSort(int[] a) {
  int[] aux = Arrays.copyOf(a, a.length);
  mergeSort(a, aux, 0, a.length-1, 0);
}

void mergeSort(int[] a, int[] aux,
               int low, int high, int depthMod2) {
  int half = (low+high)/2;
  if (low >= high) return;

  mergeSort(a, aux,     low, half, 1-depthMod2);
  mergeSort(a, aux, half+1, high, 1-depthMod2);

                // note the exchange of a and aux
  if (depthMod2 == 0) merge(aux, a, low, high);
  else                merge(a, aux, low, high);
}
```

Merge sort

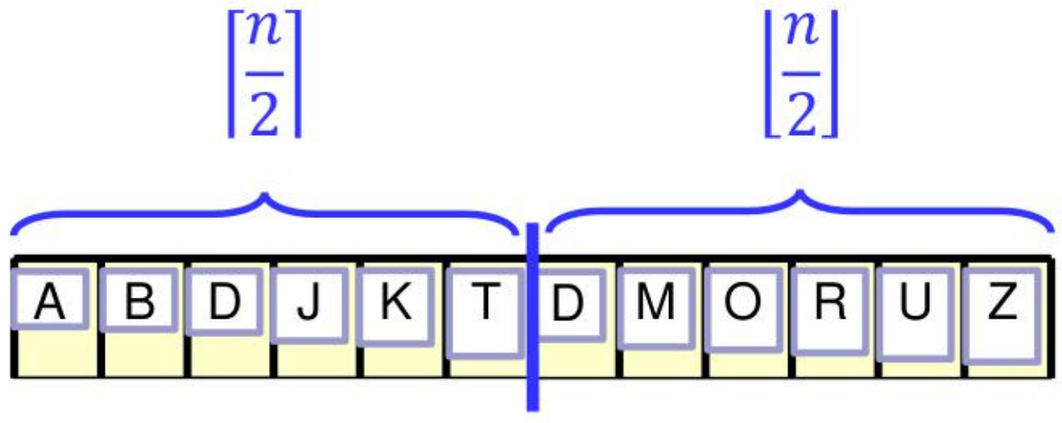
```
void mergeSort(int[] a) {
  int[] aux = Arrays.copyOf(a, a.length);
  mergeSort(a, aux, 0, a.length-1, 0);
}

void mergeSort(int[] a, int[] aux,
               int low, int high, int depthMod2) {
  int half = (low+high)/2;
  if (low >= high) return;

  mergeSort(a, aux,    low, half, 1-depthMod2);
  mergeSort(a, aux, half+1, high, 1-depthMod2);

               // note the exchange of a and aux
  if (depthMod2 == 0) merge(aux, a, low, high);
  else                merge(a, aux, low, high);
}
```

Merge sort

- Asymptotická složitost

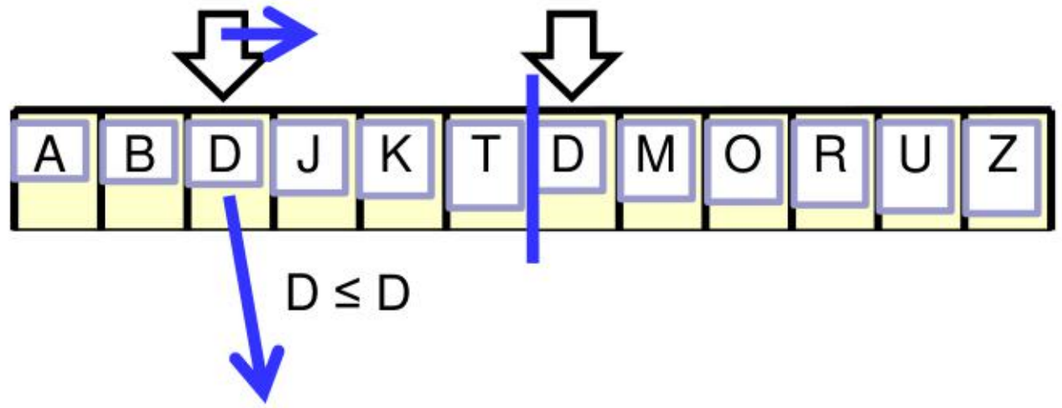
$$T(1) = \Theta(1)$$

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n), \quad n > 1$$



$\Rightarrow T(n) \in \Theta(n \log n)$ ... podle mistrovské věty

- Stabilita ✓
  - prvky se přesunují pouze při slučování
  - v případě rovnosti dvou prvků jejich relativní pořadí zachováváme



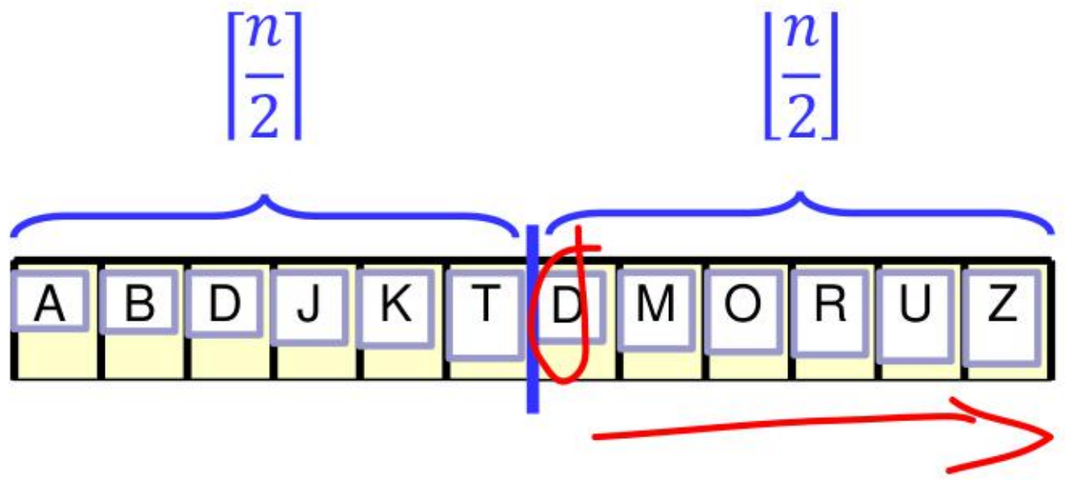
Merge sort

- Asymptotická složitost

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

$$T(n) = 2T\left(\frac{n}{2}\right) + \Theta(n), \quad n > 1$$

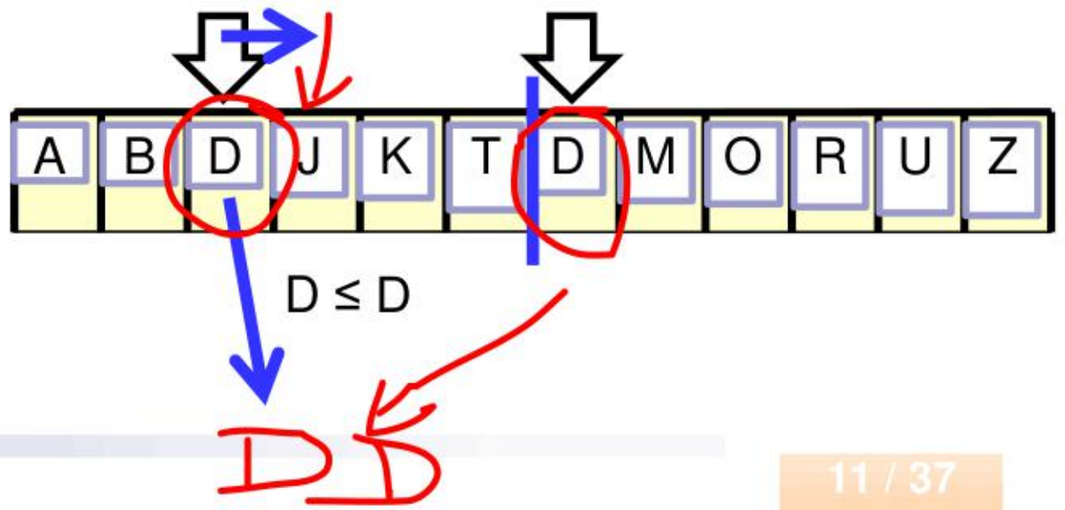
$$\Rightarrow T(n) \in \Theta(n \log n) \quad \text{... podle mistrovské věty}$$



h

2

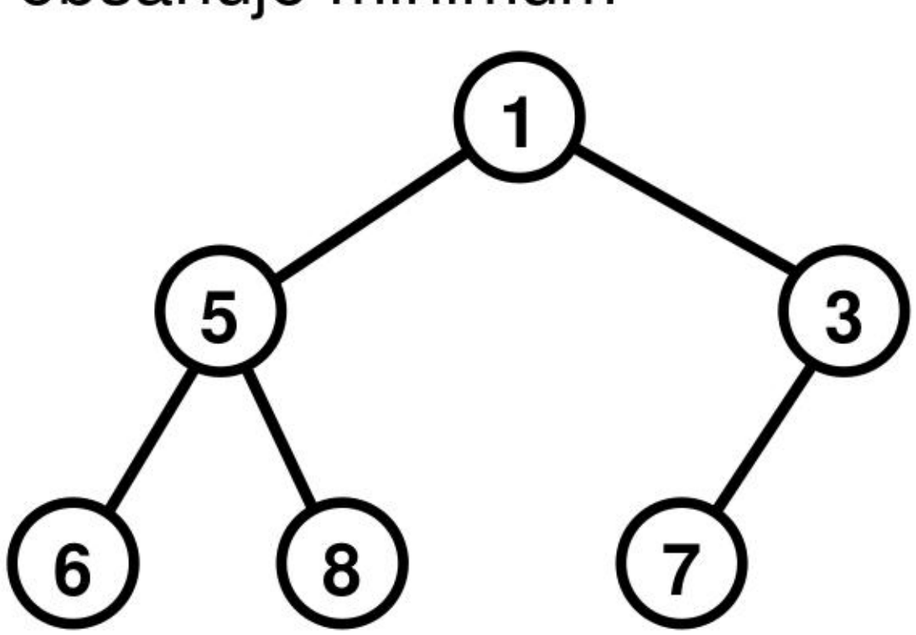
- Stabilita ✓
  - prvky se přesunují pouze při slučování
  - v případě rovnosti dvou prvků jejich relativní pořadí zachováme



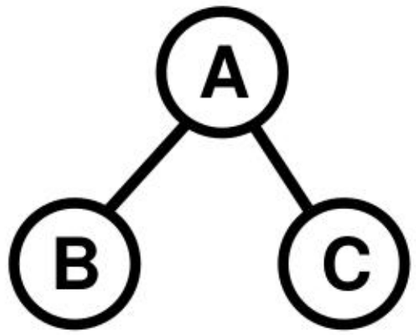
Binární halda

- Vyvážený binární strom
- Operace GetMin, DeleteMin, Insert
- Implementuje prioritní frontu

Vrchol haldy (kořen)
obsahuje minimum



min-halda

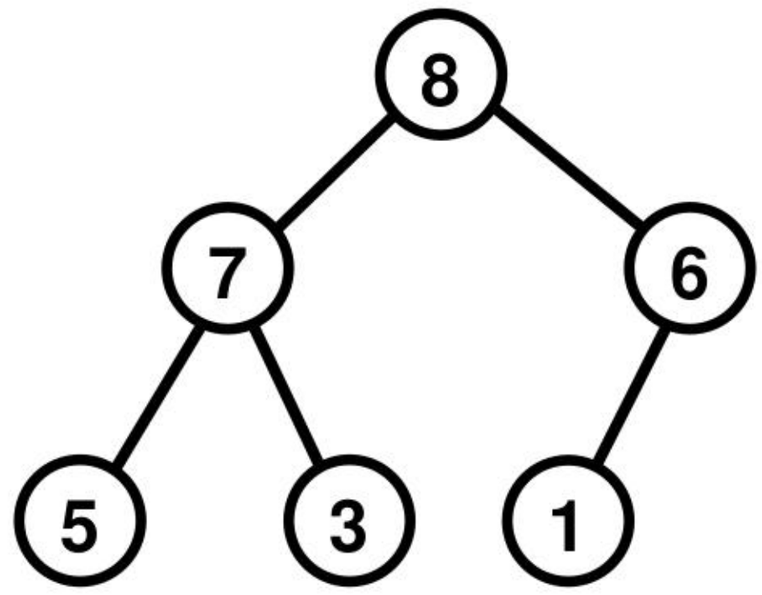


Podmínka min-haldy:

$A \leq B$ a $A \leq C$

(platí pro každý uzel)

alternativa:
max-halda
($A \geq B$ a $A \geq C$)

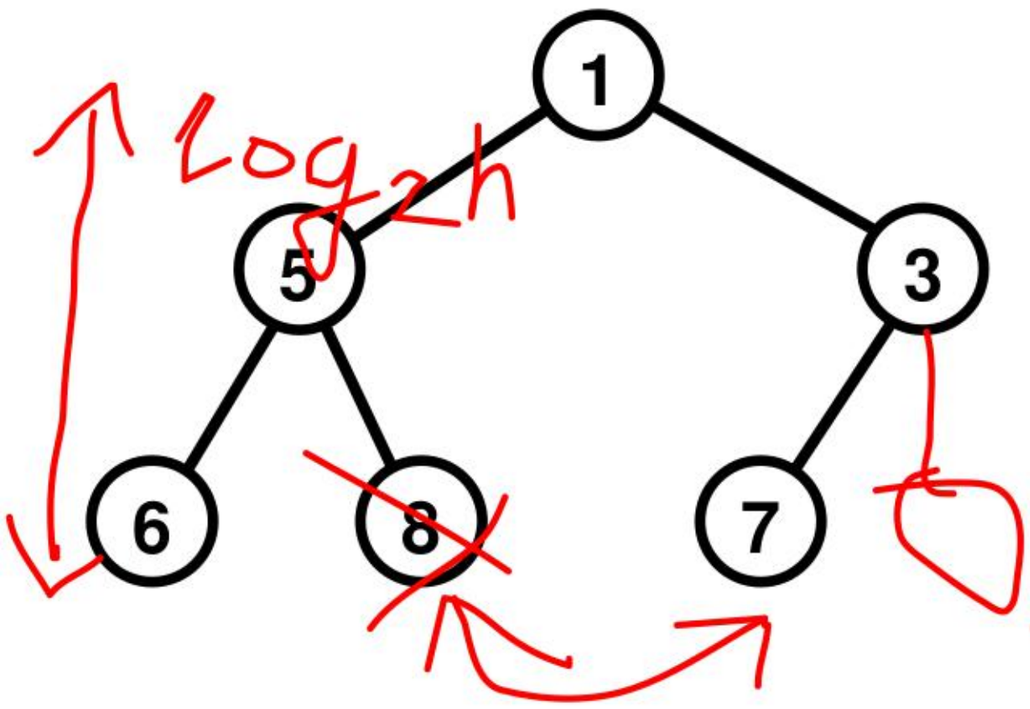


GetMax, DeleteMax

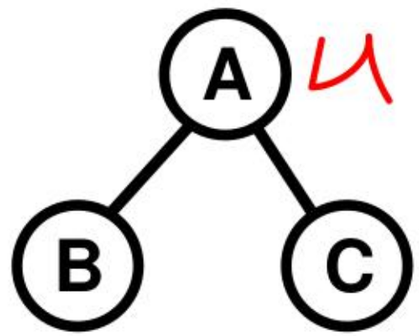
Binární halda

- Vyvážený binární strom
- Operace GetMin, DeleteMin, Insert
- Implementuje prioritní frontu

Vrchol haldy (kořen) obsahuje minimum



min-halda

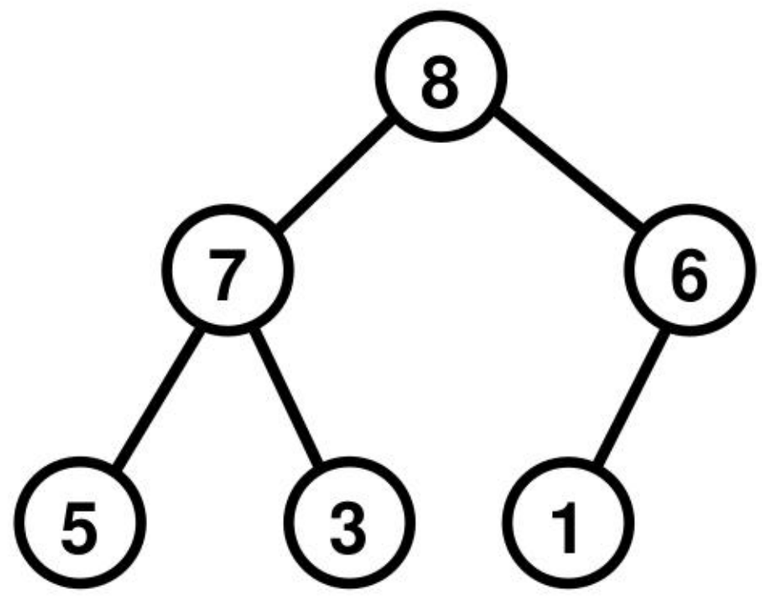


Podmínka min-haldy:

$A \leq B$ a $A \leq C$

(platí pro každý uzel)

alternativa:
max-halda
($A \geq B$ a $A \geq C$)

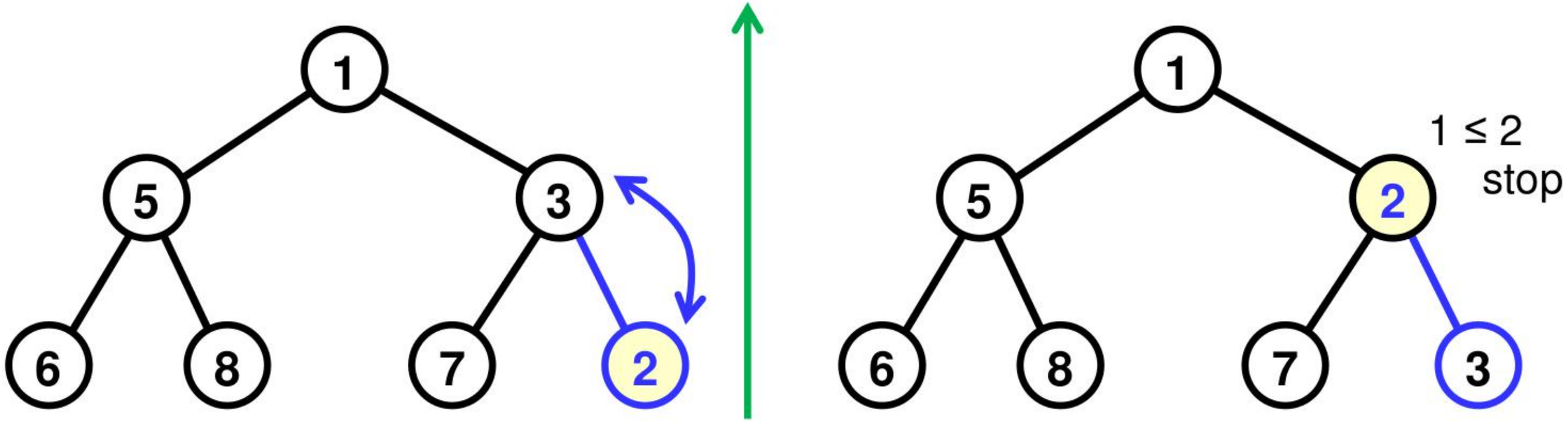


GetMax, DeleteMax

Insert v binární haldě

1. Nový prvek vložíme jako další list v pořadí
2. Na cestě ke kořeni kontrolujeme podmínku haldy a provádíme opravy (prohozením s rodičem)

Insert 2

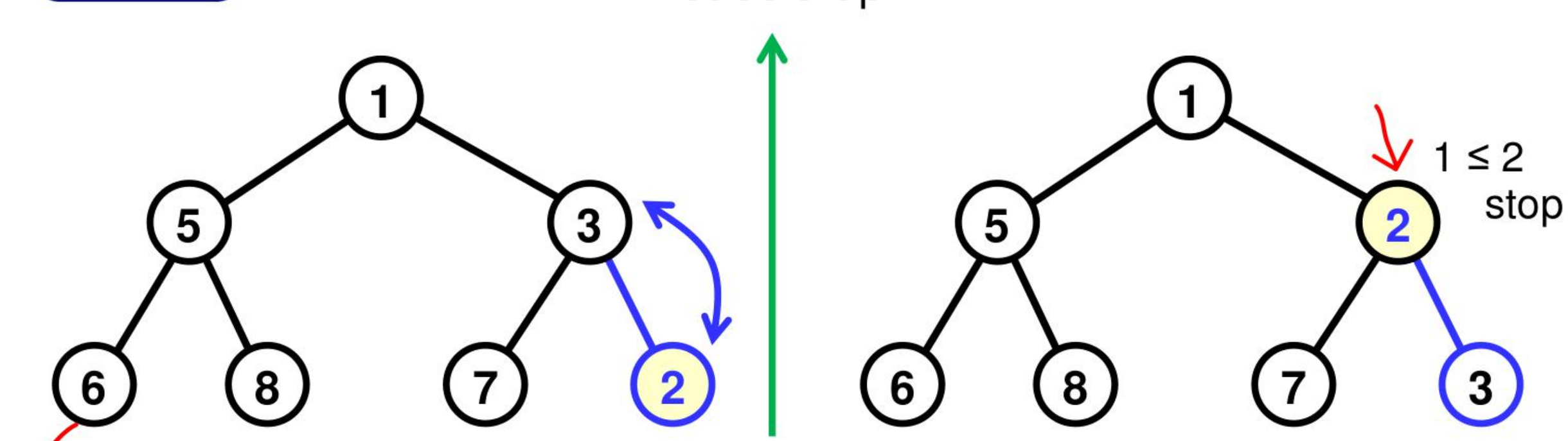


Časová složitost $O(\log n)$

Insert v binární haldě

1. Nový prvek vložíme jako další list v pořadí
2. Na cestě ke kořeni kontrolujeme podmínku haldy a provádíme opravy (prohozením s rodičem)

Insert 2

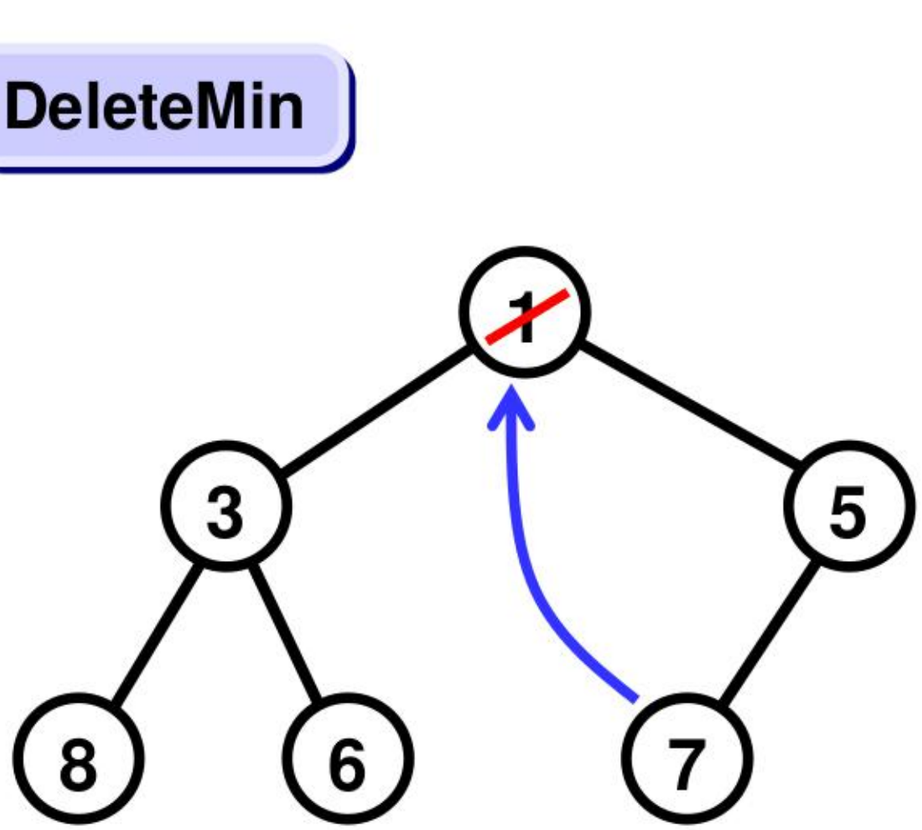


Časová složitost $O(\log n)$

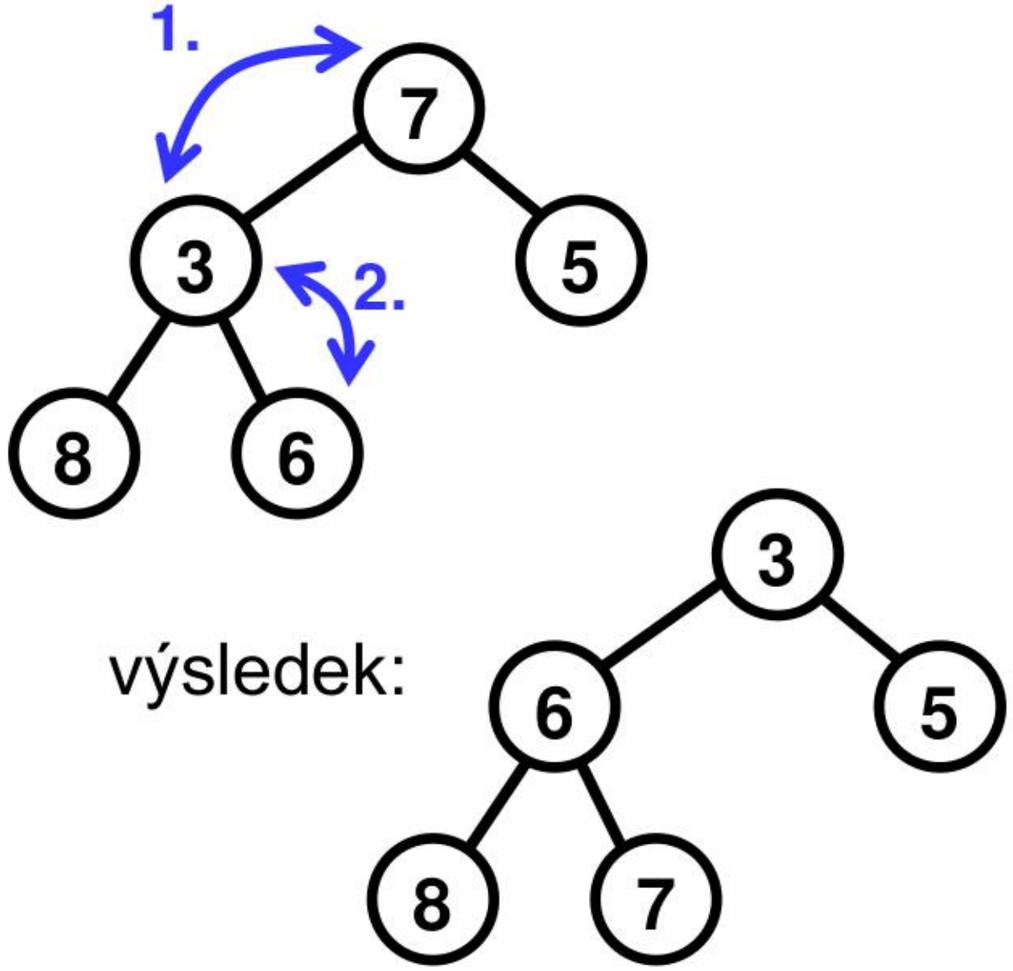
DeleteMin v binární haldě

1. Kořen nahradíme posledním listem
2. Na cestě od kořene k listům kontrolujeme podmínku haldy a provádíme opravy (prohozením s menším potomkem)

DeleteMin



bubble-down

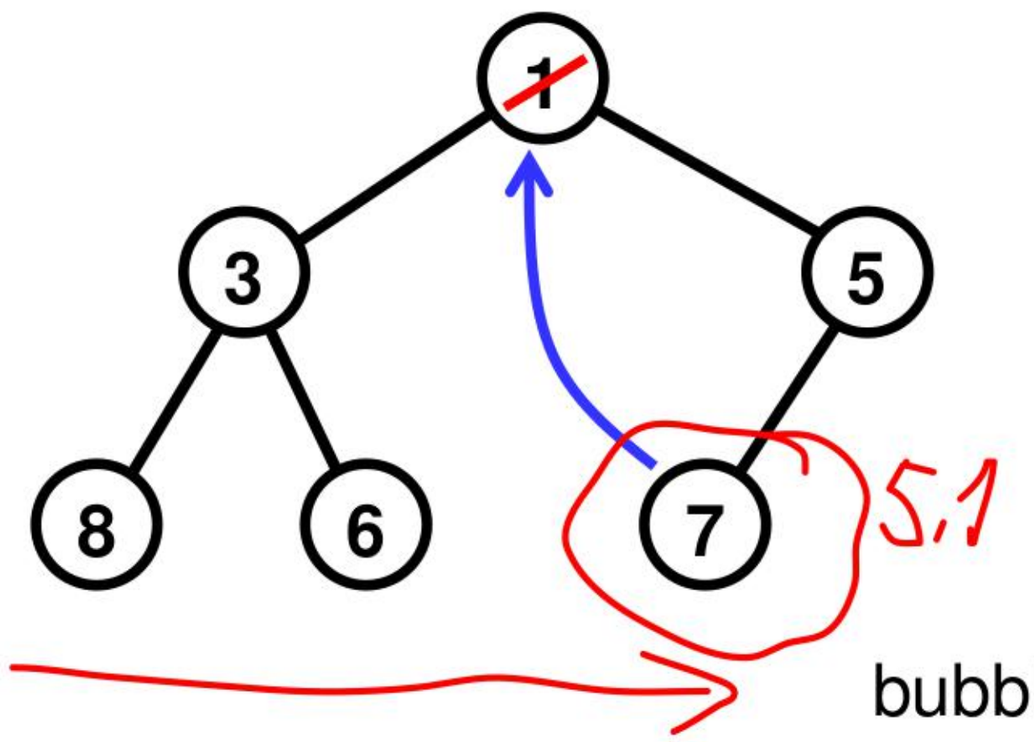


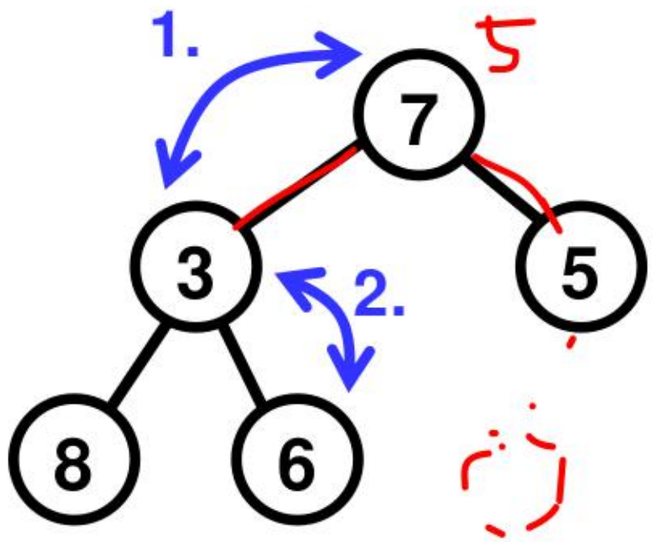
Časová složitost $O(\log n)$

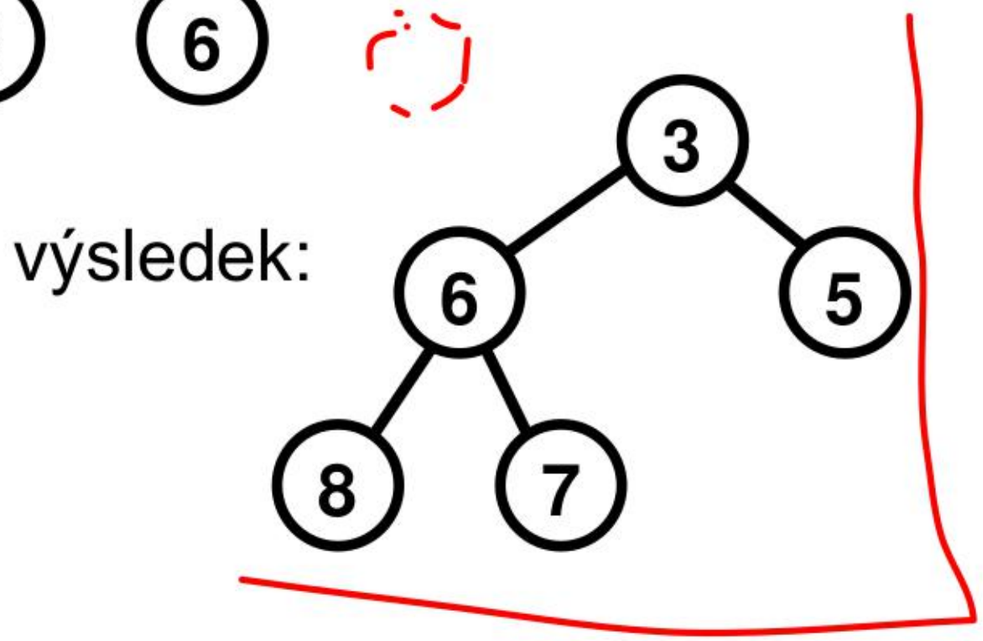
DeleteMin v binární haldě

1. Kořen nahradíme posledním listem
2. Na cestě od kořene k listům kontrolujeme podmínku haldy a provádíme opravy (prohozením s menším potomkem)

DeleteMin

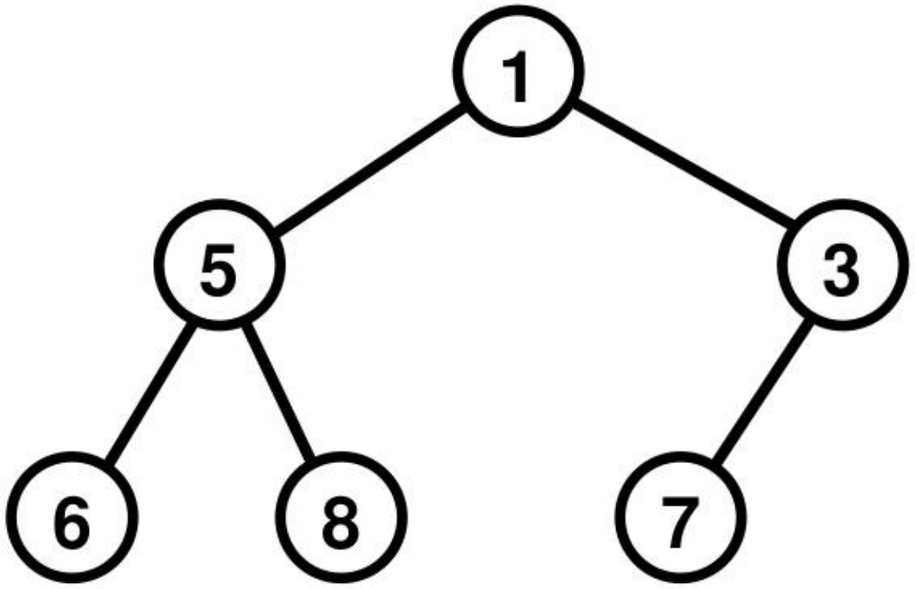


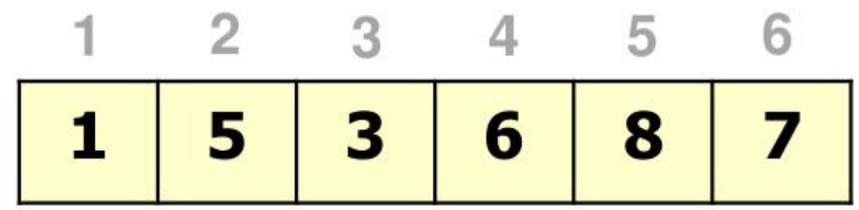




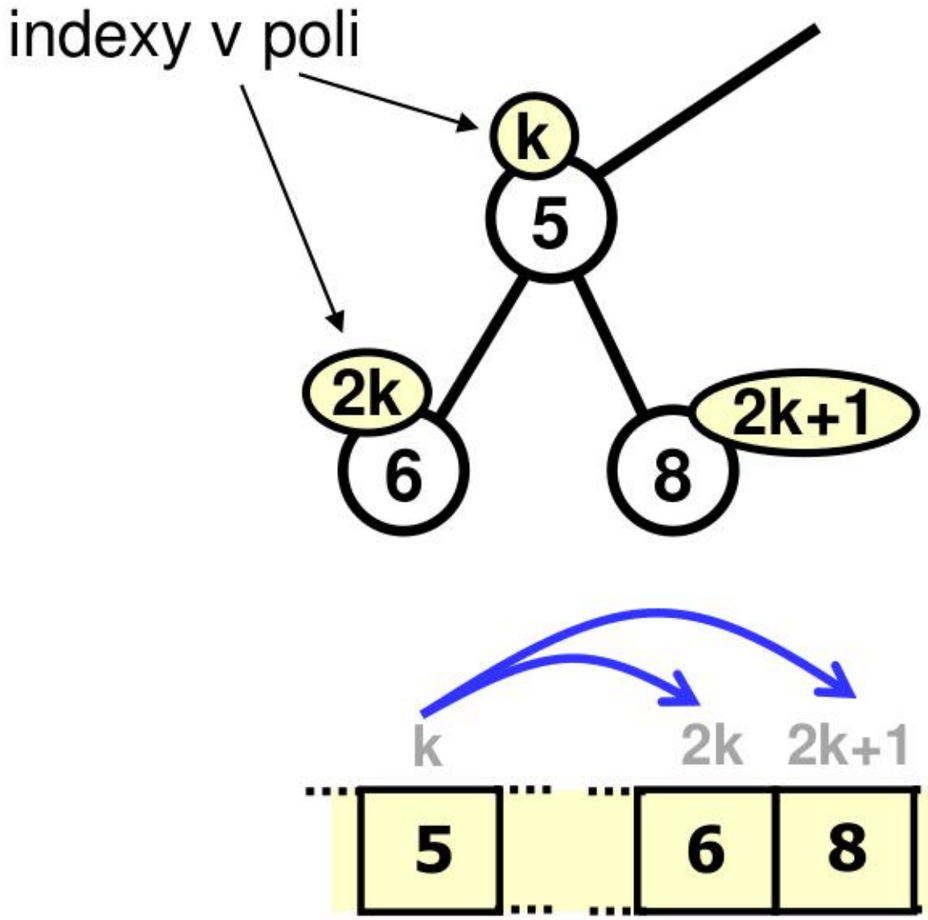
Časová složitost $O(\log n)$

Reprezentace haldy v poli



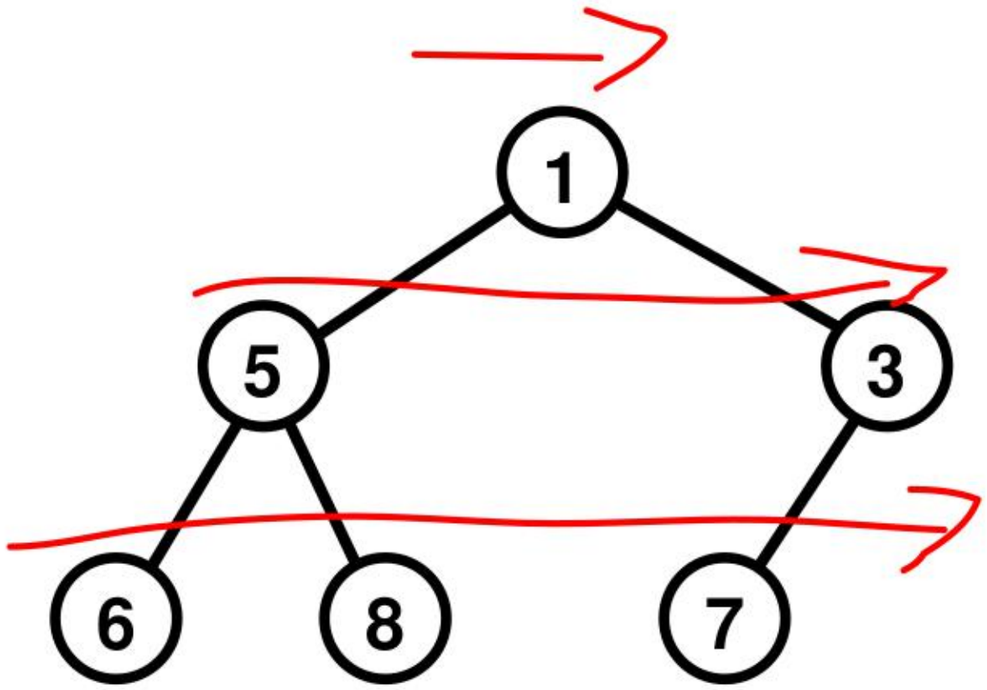


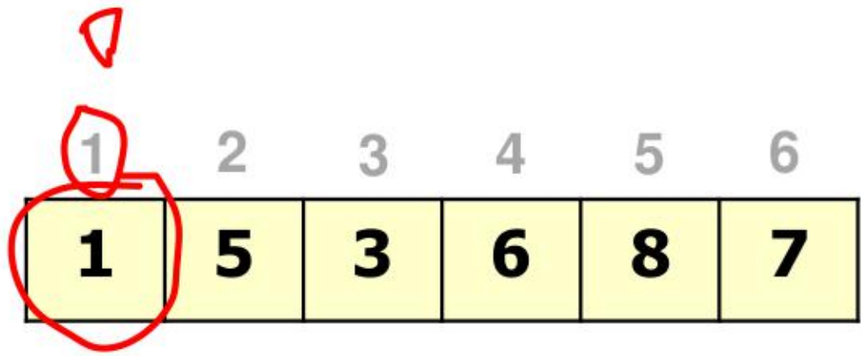
vrchol haldy je prvním prvkem pole



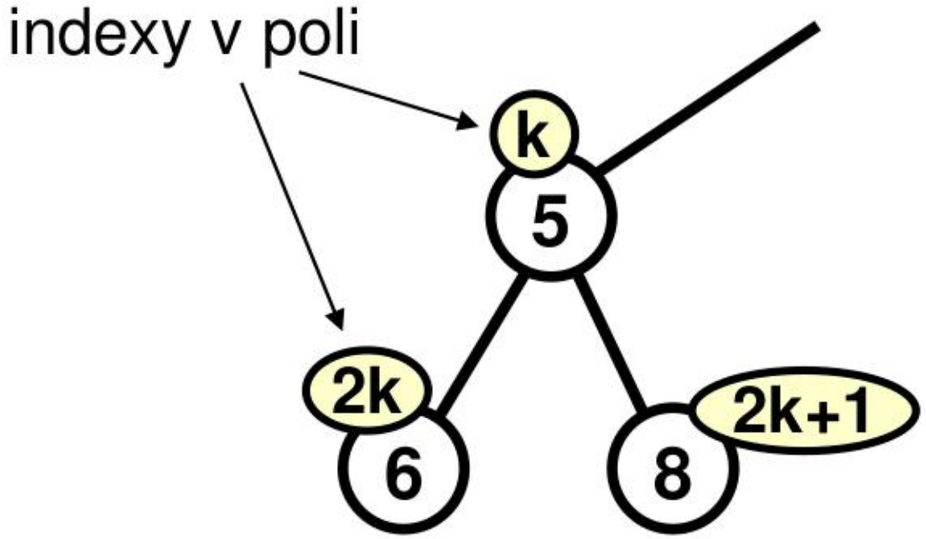
potomci prvku na indexu k jsou
na indexech 2k (levý) a 2k+1 (pravý)

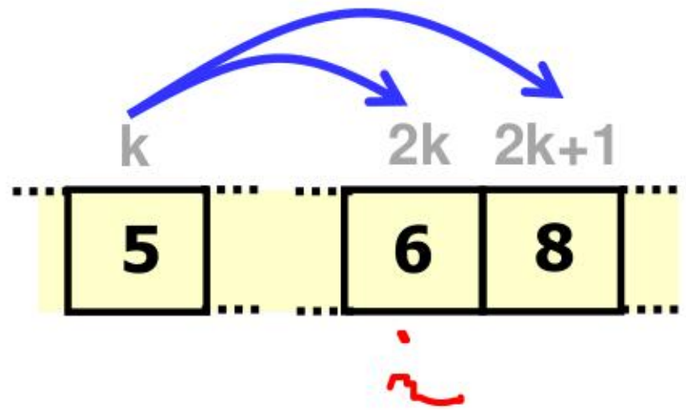
Reprezentace haldy v poli





vrchol haldy je prvním prvkem pole

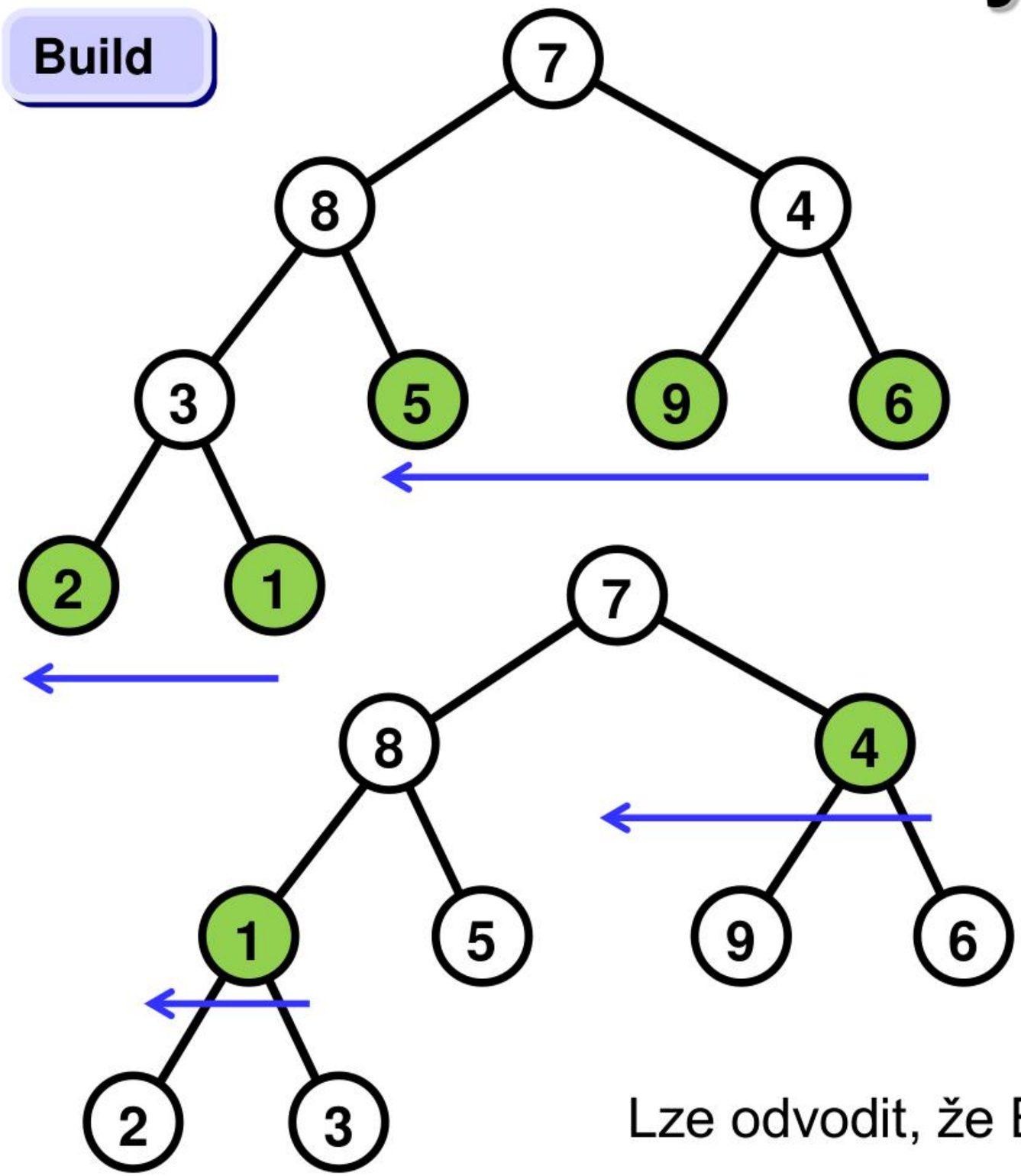




potomci prvku na indexu k jsou
na indexech 2k (levý) a 2k+1 (pravý)

i/2 parent

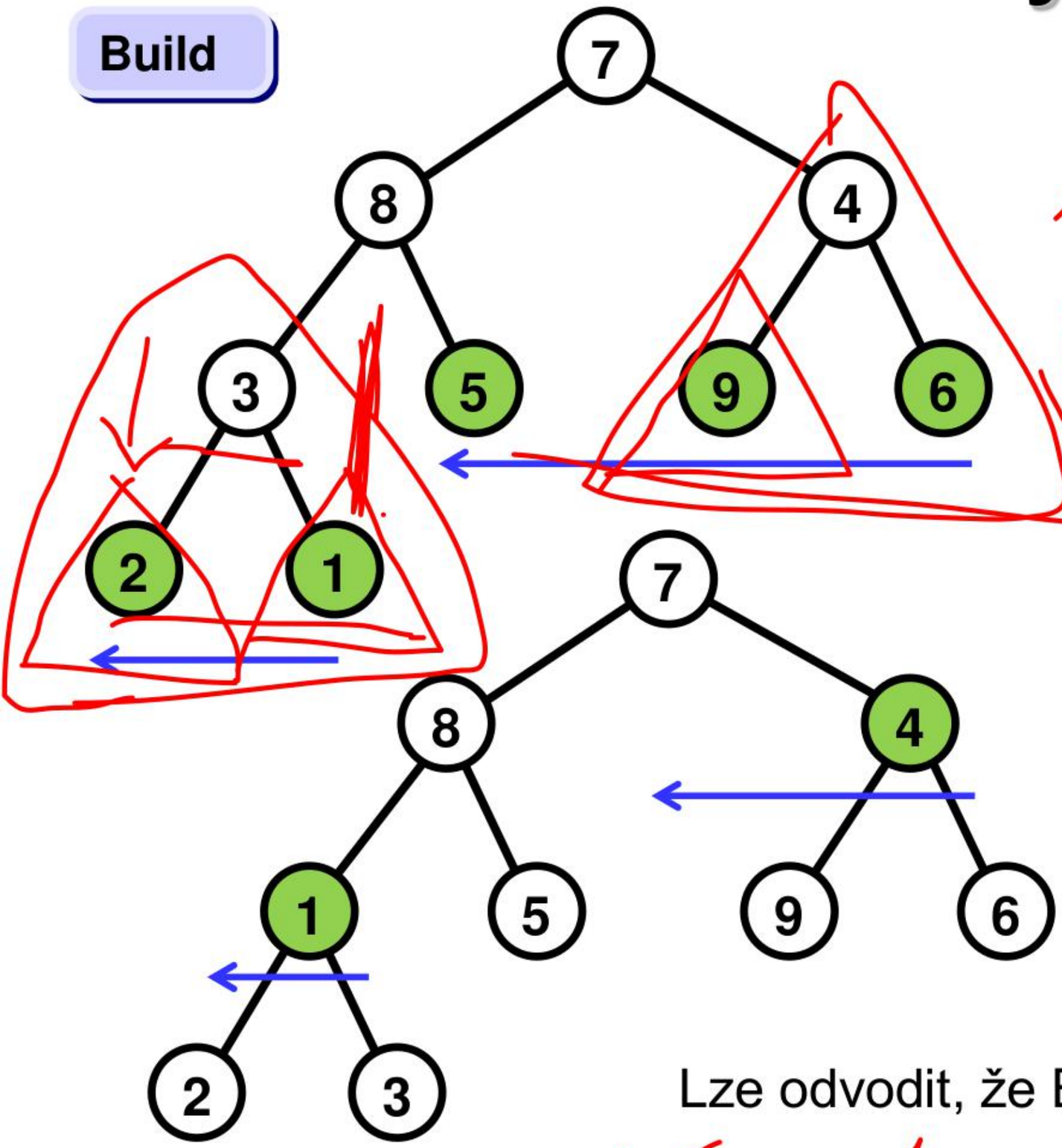
Vybudování haldy



- V poli reprezentujícím haldu postupujeme od posledního prvku k prvnímu.
- Podstrom zakořeněný v listu je halda.
- Pokud v podstromu je pravidlo haldy porušeno jen v jeho kořeni, napravíme to provedením bubble-down.

Lze odvodit, že Build má časovou složitost $\Theta(n)$.

Vybudování haldy

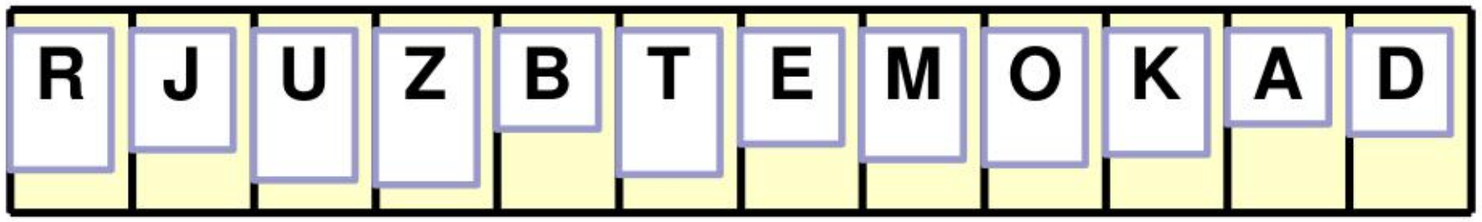


- V poli reprezentujícím haldu postupujeme od posledního prvku k prvnímu.
- Podstrom zakořeněný v listu je halda.
- Pokud v podstromu je pravidlo haldy porušeno jen v jeho kořeni, napravíme to provedením bubble-down.

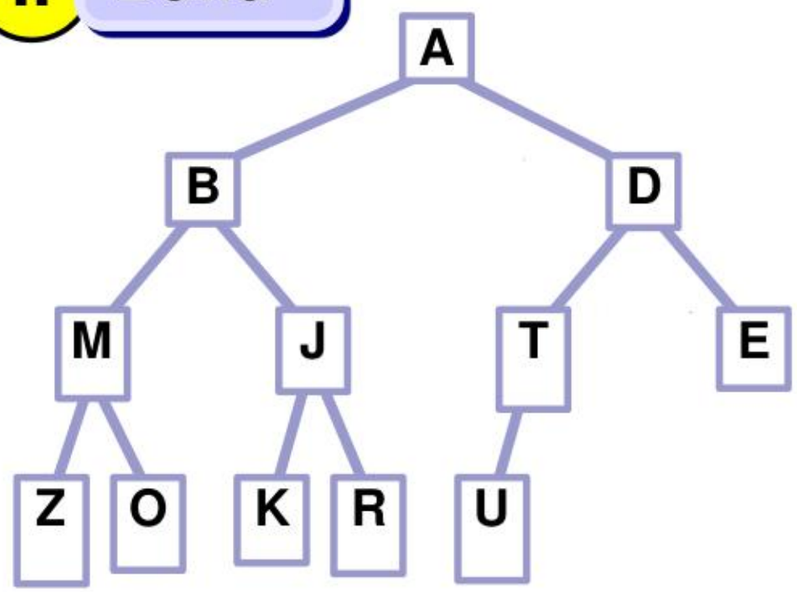
Lze odvodit, že Build má časovou složitost $\Theta(n)$.

Heap sort (řazení haldou)

I Vstup

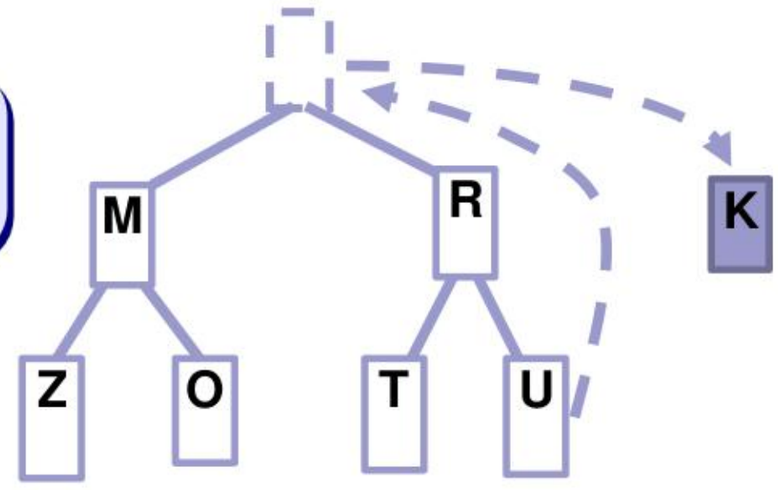


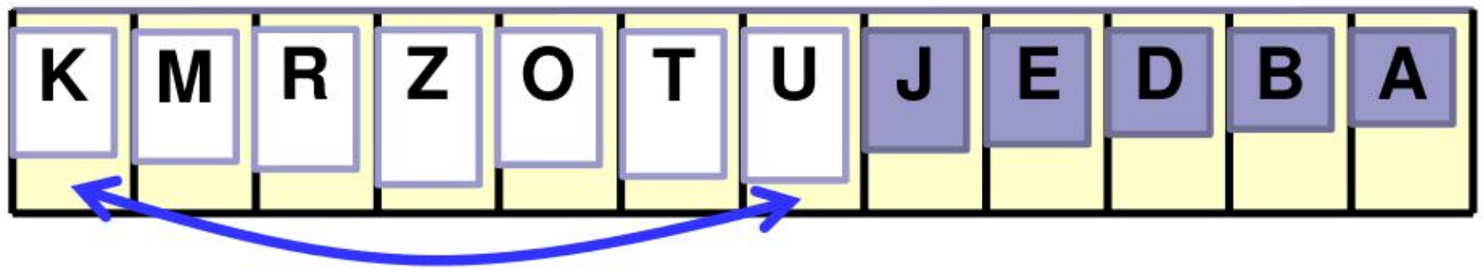
II Build

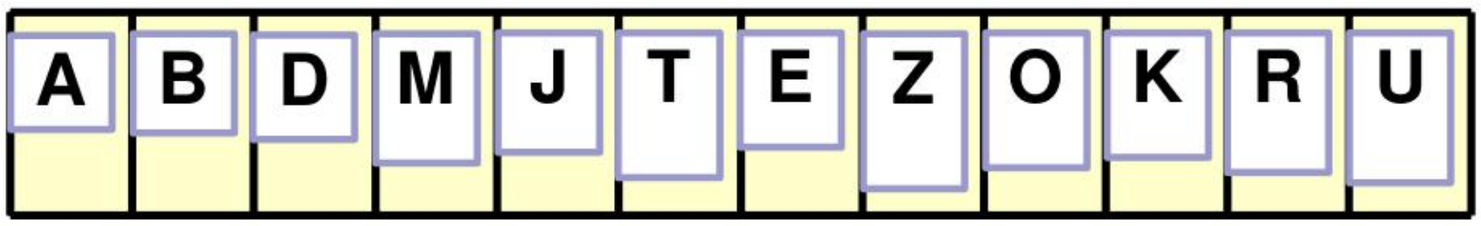


III

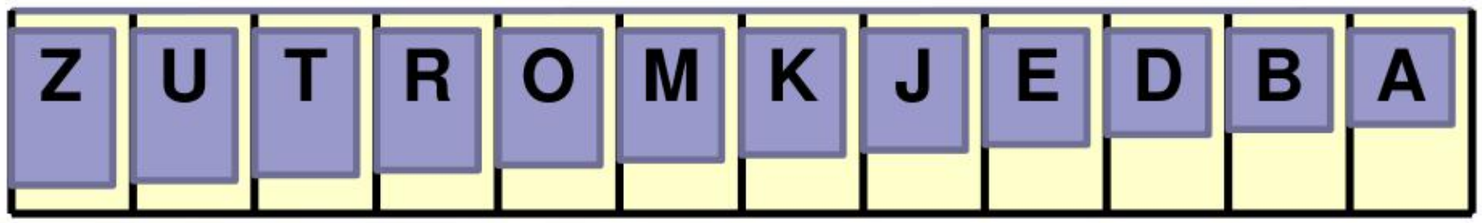
```
for (i = 1; i <= n; i++)
    DeleteMin;
```





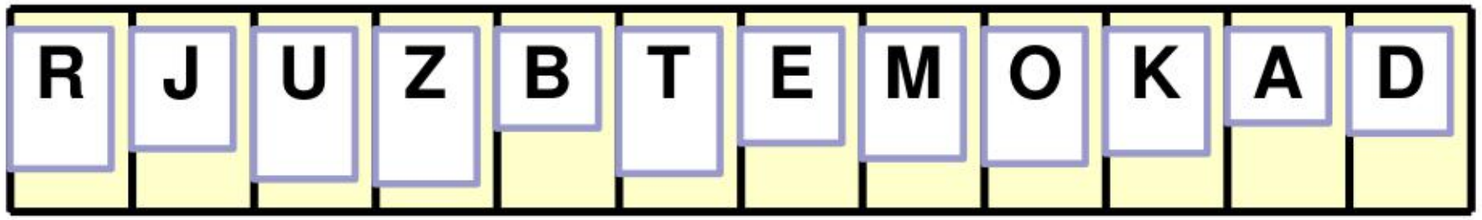


IV Výstup

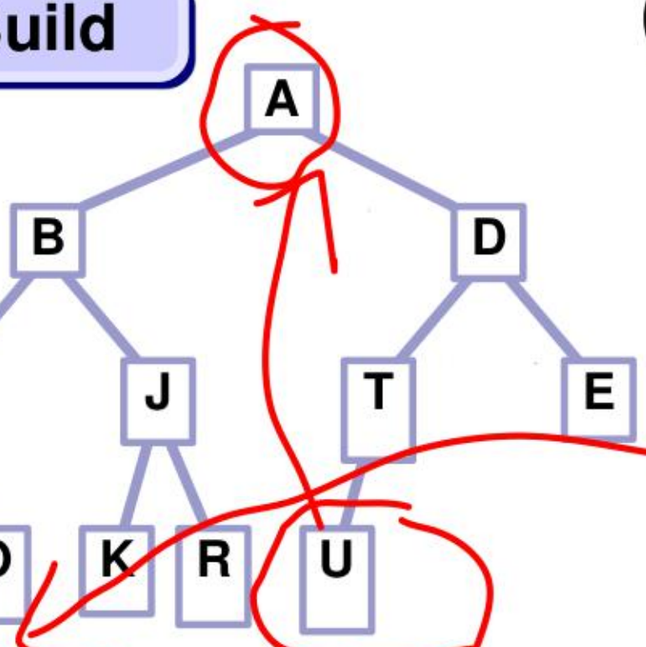


Heap sort (řazení haldou)

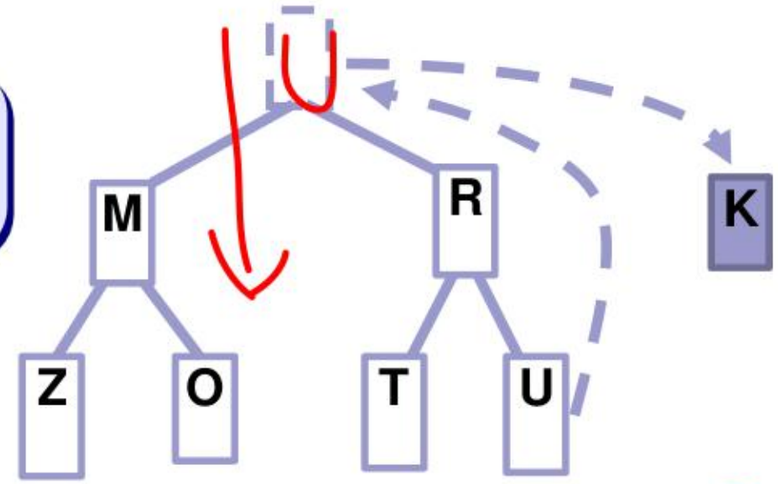
I Vstup

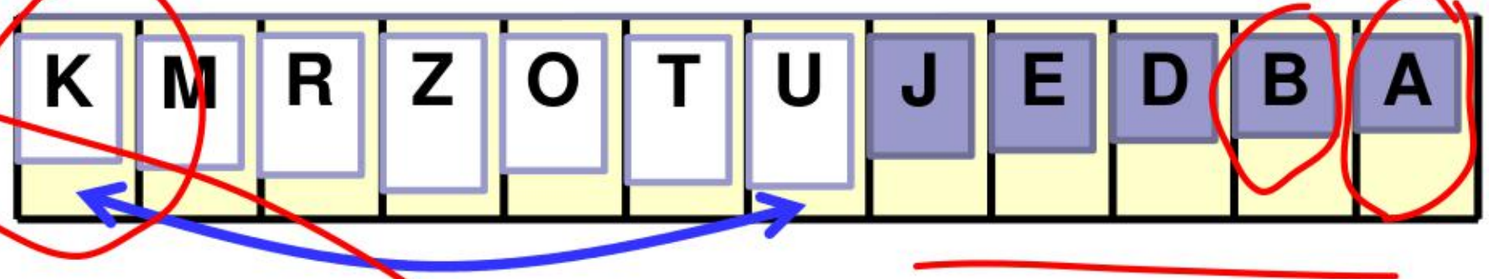


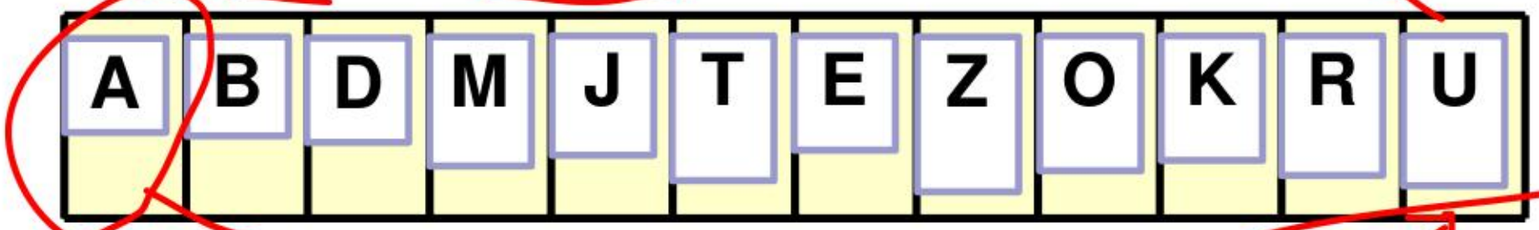
II Build



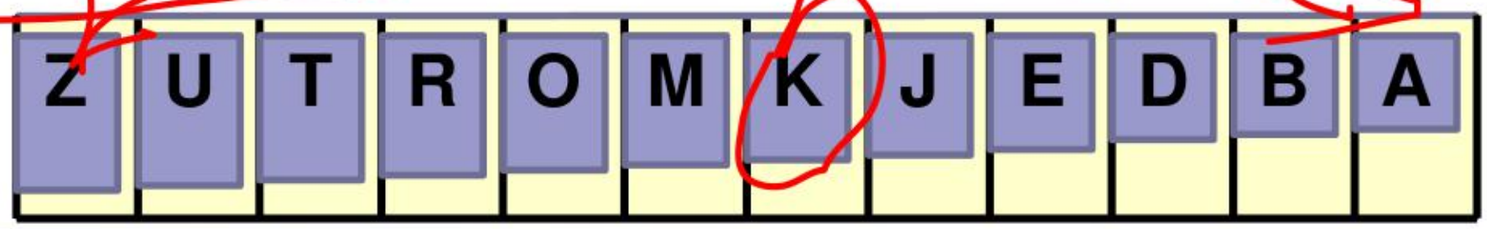
III for (i = 1; i <= n; i++)
DeleteMin;







IV Výstup



Heap sort

```
                                  // array: a[1]...a[n]  !!!!

void heapSort(Item[] a, int n) {

                                            // create a heap
   for (int i = n/2; i > 0; i--)
     repairTop(a, i, n);                    // bubble-down

                                            // sort
   for (int i = n; i > 1; i--) {
     swap(a, 1, i);                         // swap a[1] and a[i]
     repairTop(a, 1, i-1);                  // bubble-down
   }
}
```

II

III

Heap sort

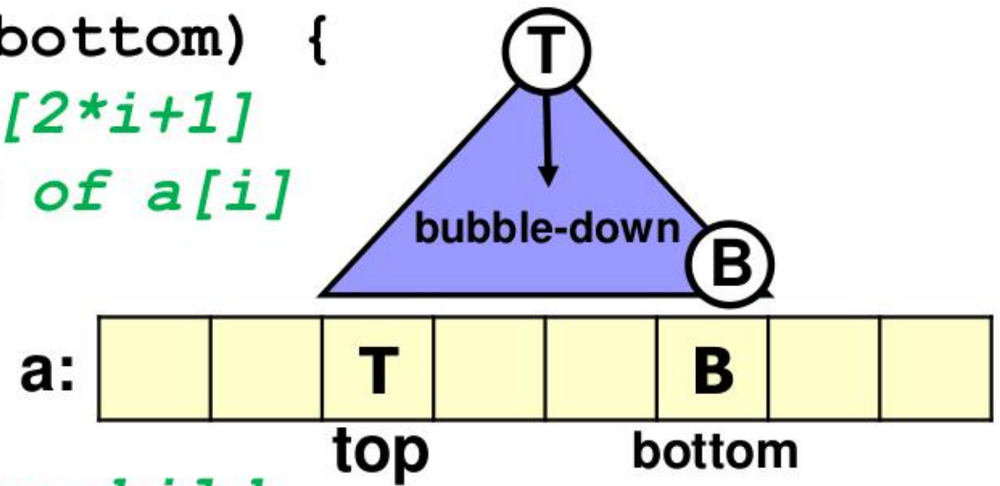
```
                                  // array: a[1]...a[n]  !!!!

void heapSort(Item[] a, int n) {

                                            // create a heap
   for (int i = n/2; i > 0; i--)
     repairTop(a, i, n);                    // bubble-down

                                            // sort
   for (int i = n; i > 1; i--) {
     swap(a, 1, i);                         // swap a[1] and a[i]
     repairTop(a, 1, i-1);                  // bubble-down
   }
}
```

II

III

Heap sort

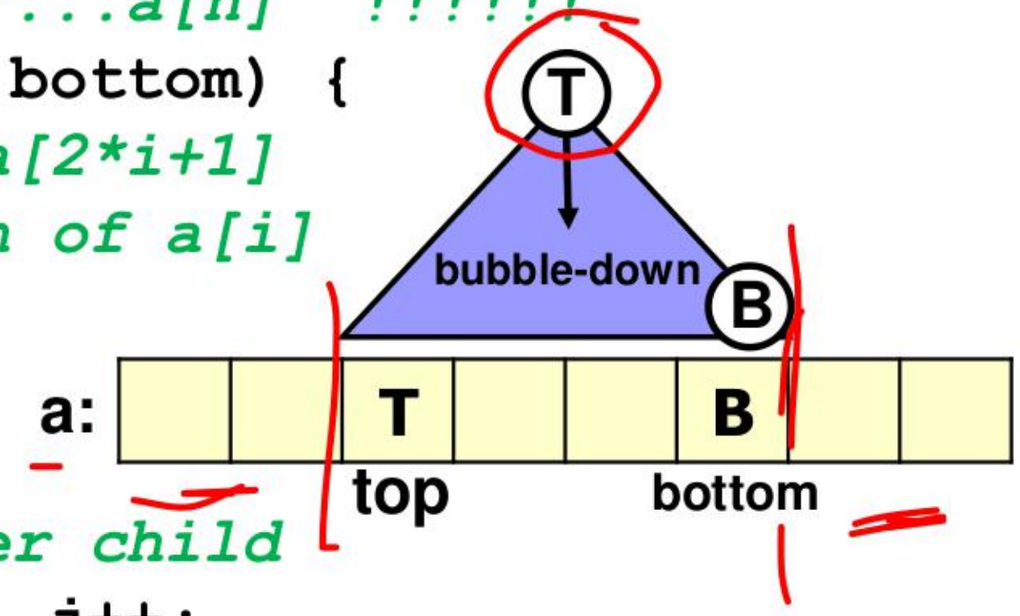
```
                              // array:  a[1]...a[n]  !!!!!!
void repairTop(Item[] a, int top, int bottom) {
  int i = top;                // a[2*i] and a[2*i+1]
  int j = i*2;                // are children of a[i]

  Item topVal = a[top];
```




```
                              // select smaller child
  if ((j < bottom) && (a[j] > a[j+1])) j++;

                              // while (topVal > children)
                              //         move children up
  while ((j <= bottom) && (topVal > a[j])) {
    a[i] = a[j];
    i = j;  j = j*2;  // skip to next child
    if ((j < bottom) && (a[j] > a[j+1])) j++;
  }
  a[i] = topVal;              // put topVal where it belongs
}
```

Heap sort

```
                                  // array:  a[1]...a[n]  !!!!!!
void repairTop(Item[] a, int top, int bottom) {
   int i = top;                   // a[2*i] and a[2*i+1]
   int j = i*2;                   // are children of a[i]

   Item topVal = a[top];

                                  // select smaller child
   if ((j < bottom) && (a[j] > a[j+1])) j++;

                                  // while (topVal > children)
                                  //          move children up
   while ((j <= bottom) && (topVal > a[j])) {
      a[i] = a[j];
      i = j;  j = j*2;  // skip to next child
      if ((j < bottom) && (a[j] > a[j+1])) j++;
   }
   a[i] = topVal;                 // put topVal where it belongs
}
```



Heap sort

- Asymptotická složitost pro vstupní pole délky n

  Build + n-krát DeleteMin

$\Rightarrow \Theta(n) + O(n \log n) = O(n \log n)$

- Heap sort není stabilní

Heap sort

- Asymptotická složitost pro vstupní pole délky n

Build + n-krát DeleteMin

$\Rightarrow \Theta(n) + O(n \log n) = O(n \log n)$

- Heap sort není stabilní

Empirické porovnání

Délka pole	% seř.	Doba běhu v milisekundách, není-li uvedeno jinak					
		Sort					
		Select	Insert	Bubble	Quick	Merge	Heap
10	0%	0.0005	☆ 0.0002	0.0005	0.0004	0.0009	0.0005
10	90%*	0.0004	☆ 0.0001	0.0004	0.0004	0.0007	0.0005
100	0%	0.028	0.016	0.043	0.081	0.014	☆ 0.011
100	90%	0.026	☆ 0.003	0.030	0.010	0.011	0.011
1 000	0%	2.36	1.30	4.45	☆ 0.12	0.19	0.17
1 000	90%	2.31	0.18	2.86	0.16	☆ 0.15	0.16
10 000	0%	228	130	450	☆ 1.57	2.40	2.31
10 000	90%	229	17.5	285	1.93	☆ 1.68	2.11
100 000	0%	22 900	12 800	45 000	☆ 18.7	31.4	31.4
100 000	90%	22 900	1 760	28 500	27.4	☆ 24.6	25.5
1 000 000	0%	38 min	22 min	75 min	☆ 237	385	570
1 000 000	90%	38 min	2.9 min	47.5 min	336	☆ 301	381

* 90% prvků vstupního pole je na správné pozici

Prostředí: Intel(R) 1.8 GHz, Microsoft Windows XP SP3, jdk 1.6.0_16

Empirické porovnání

Délka pole	% seř.	Doba běhu v milisekundách, není-li uvedeno jinak					
		Sort					
		Select	Insert	Bubble	Quick	Merge	Heap
10	0%	0.0005	★ 0.0002	0.0005	0.0004	0.0009	0.0005
10	90%*	0.0004	★ 0.0001	0.0004	0.0004	0.0007	0.0005
100	0%	0.028	0.016	0.043	0.081	0.014	★ 0.011
100	90%	0.026	★ 0.003	0.030	0.010	0.011	0.011
1 000	0%	2.36	1.30	4.45	★ 0.12	0.19	0.17
1 000	90%	2.31	0.18	2.86	0.16	★ 0.15	0.16
10 000	0%	228	130	450	★ 1.57	2.40	2.31
10 000	90%	229	17.5	285	1.93	★ 1.68	2.11
100 000	0%	22 900	12 800	45 000	★ 18.7	31.4	31.4
100 000	90%	22 900	1 760	28 500	27.4	★ 24.6	25.5
1 000 000	0%	38 min	22 min	75 min	★ 237	385	570
1 000 000	90%	38 min	2.9 min	47.5 min	336	★ 301	381

I+Q

*90% prvků vstupního pole je na správné pozici

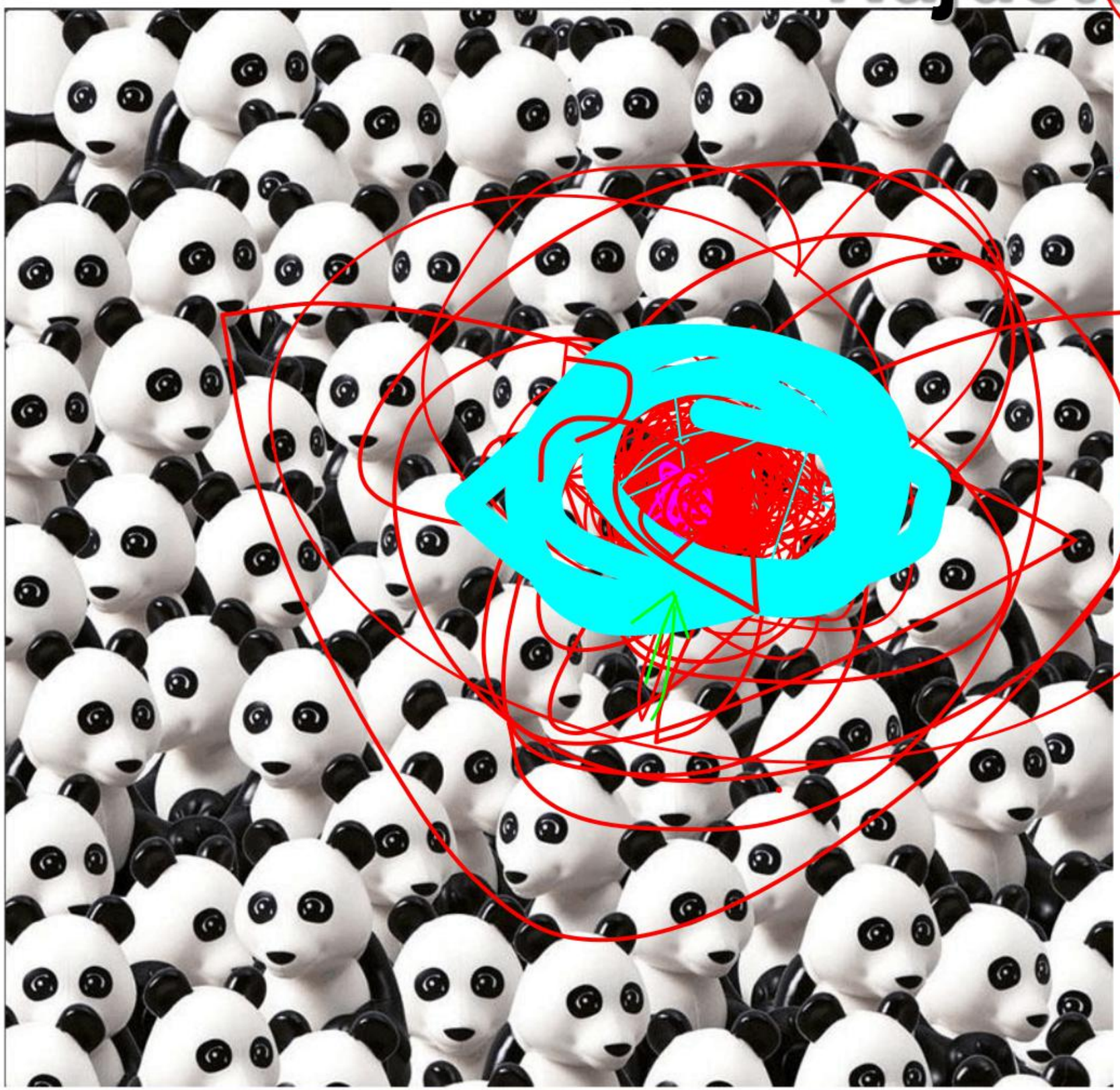
Prostředí: Intel(R) 1.8 GHz, Microsoft Windows XP SP3, jdk 1.6.0_16

Najděte pandu

Najděte pandu

Najděte psa

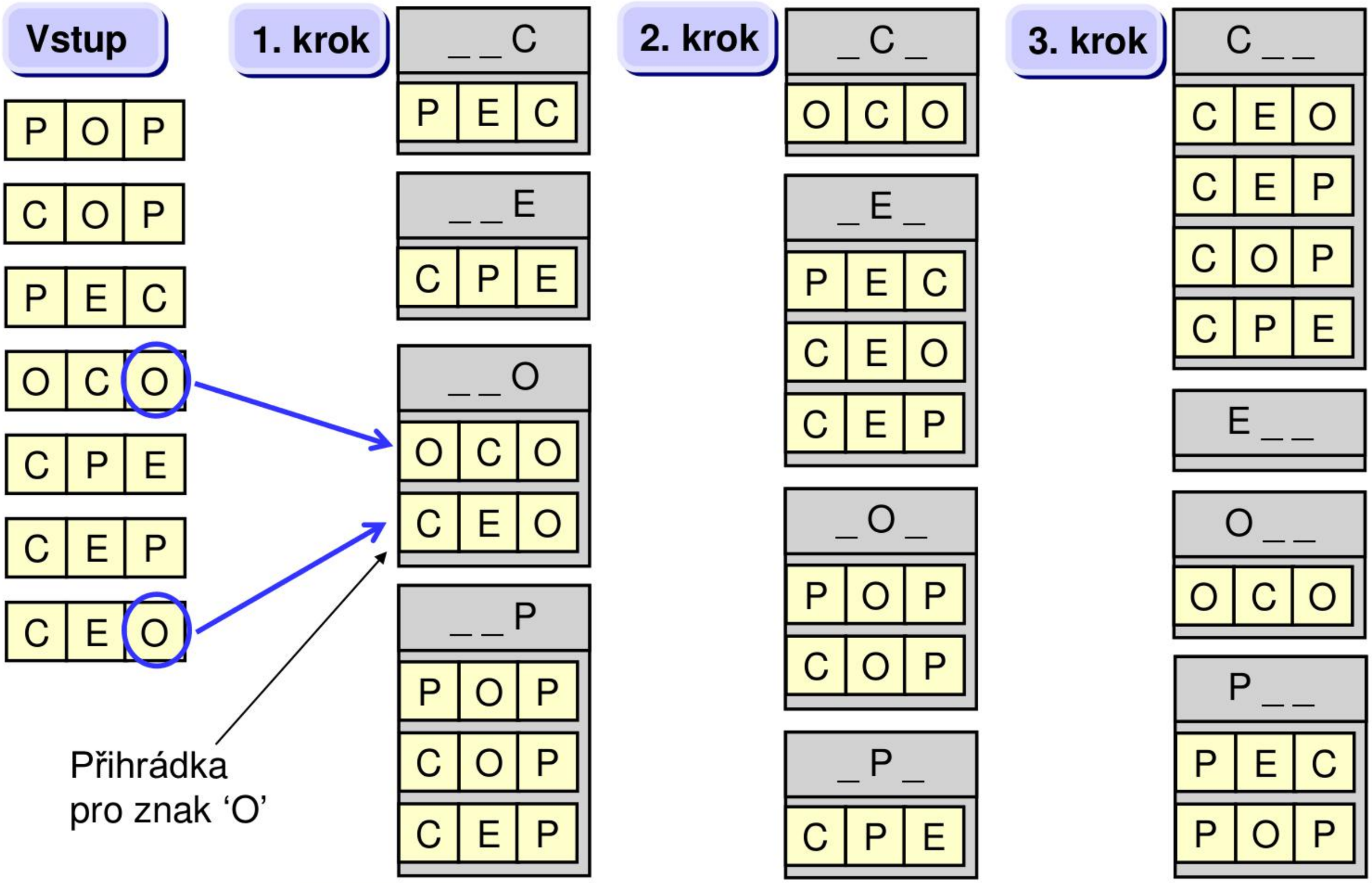
Najděte psa



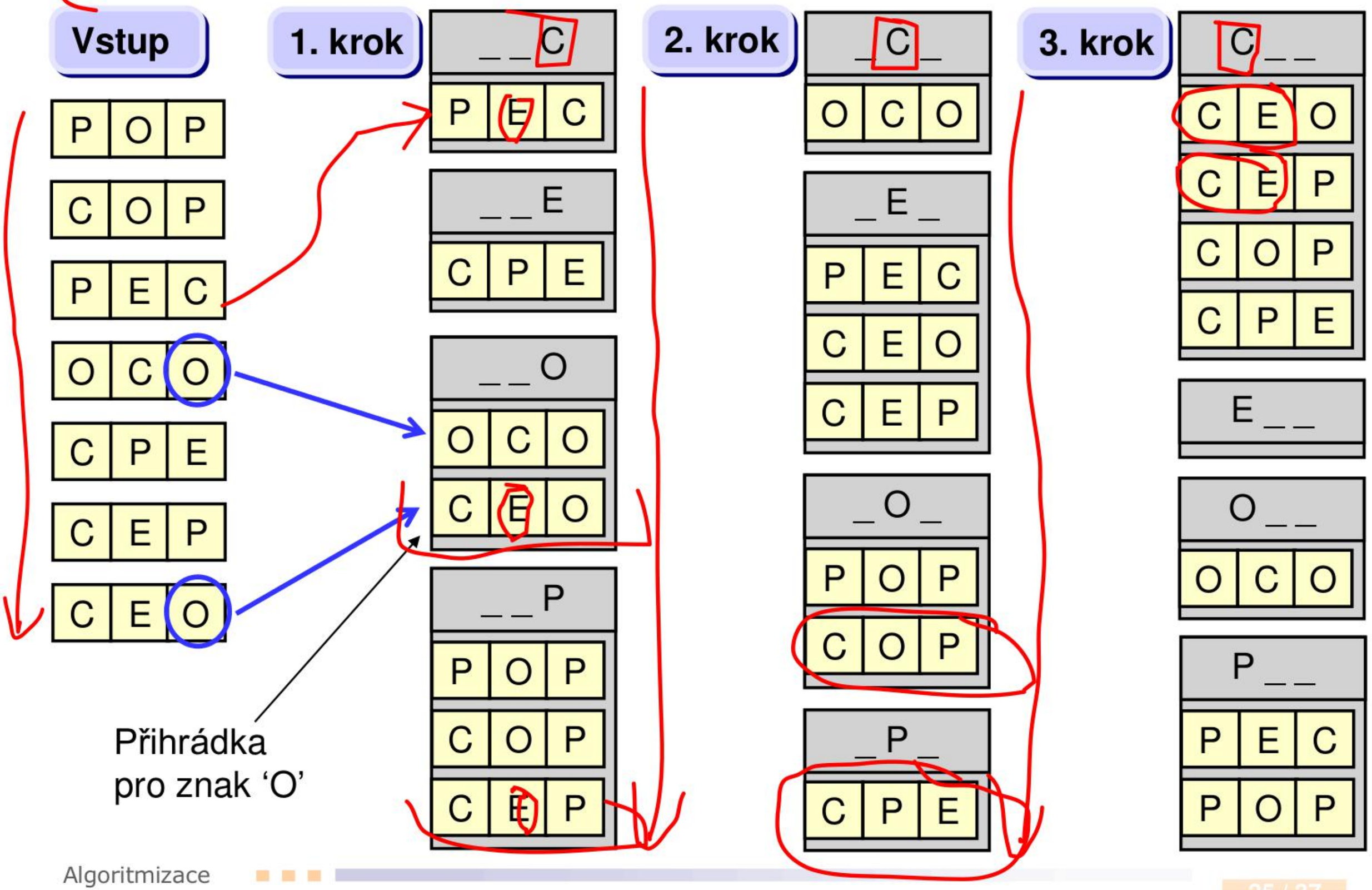
Najděte 10 rozdílů

Najděte 10 rozdílů

Radix sort (přihrádkové řazení)

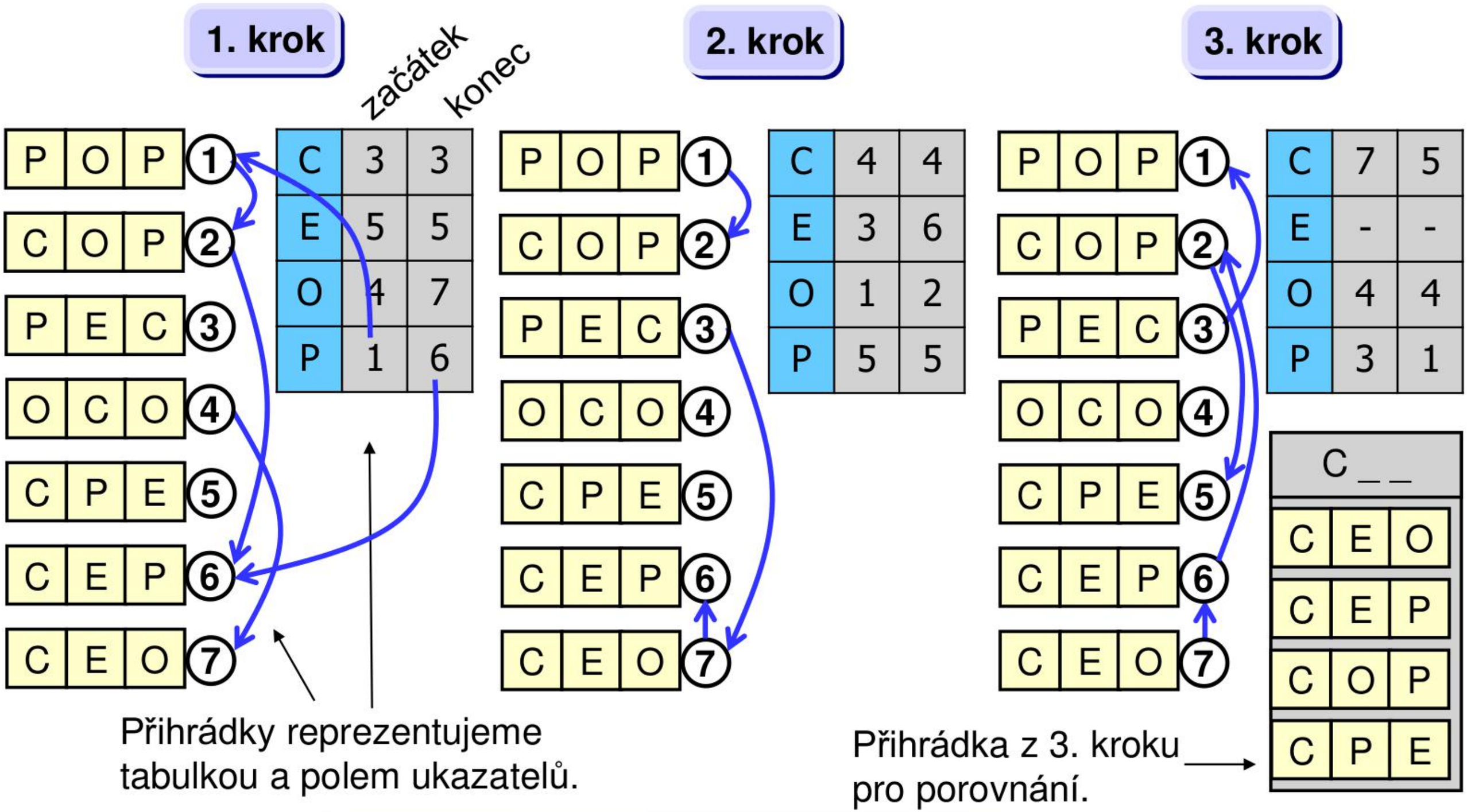


Radix sort (přihrádkové řazení)



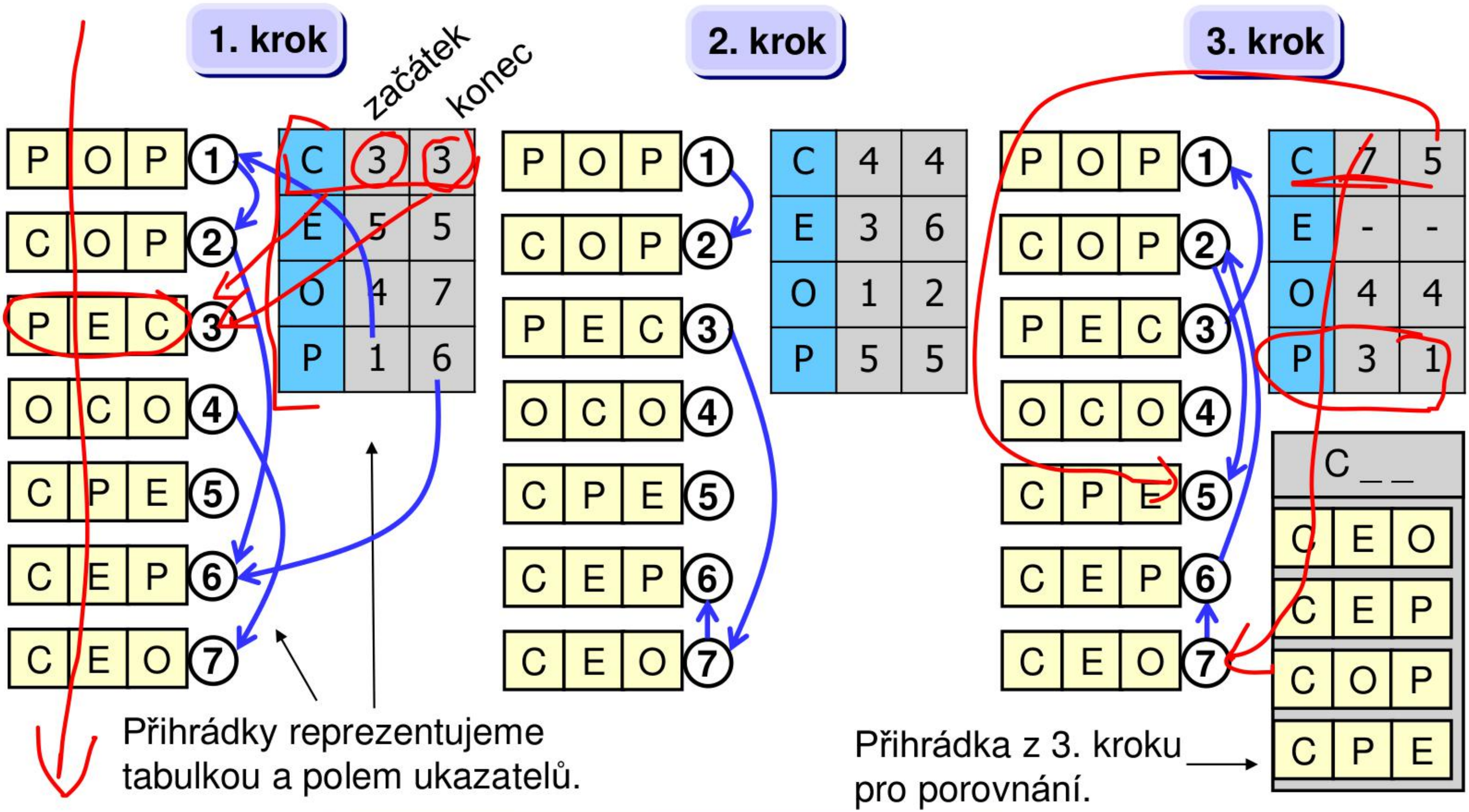
Radix sort

- Implementace bez přesouvání vstupních dat



Radix sort

- Implementace bez přesouvání vstupních dat

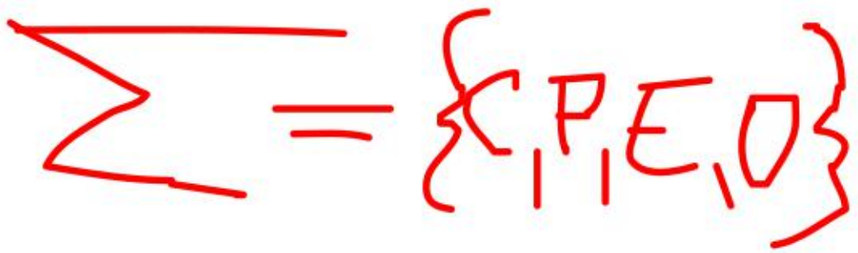


Radix sort

- Řadíme n řetězců délky k nad abecedou Σ.
- Časová složitost:
  - jeden krok … $\Theta(n + |\Sigma|)$
  - k kroků … $\Theta\big(k \cdot (n + |\Sigma|)\big) = \Theta(k \cdot n)$ (pro fixní abecedu)
- Stabilní řazení.

Radix sort

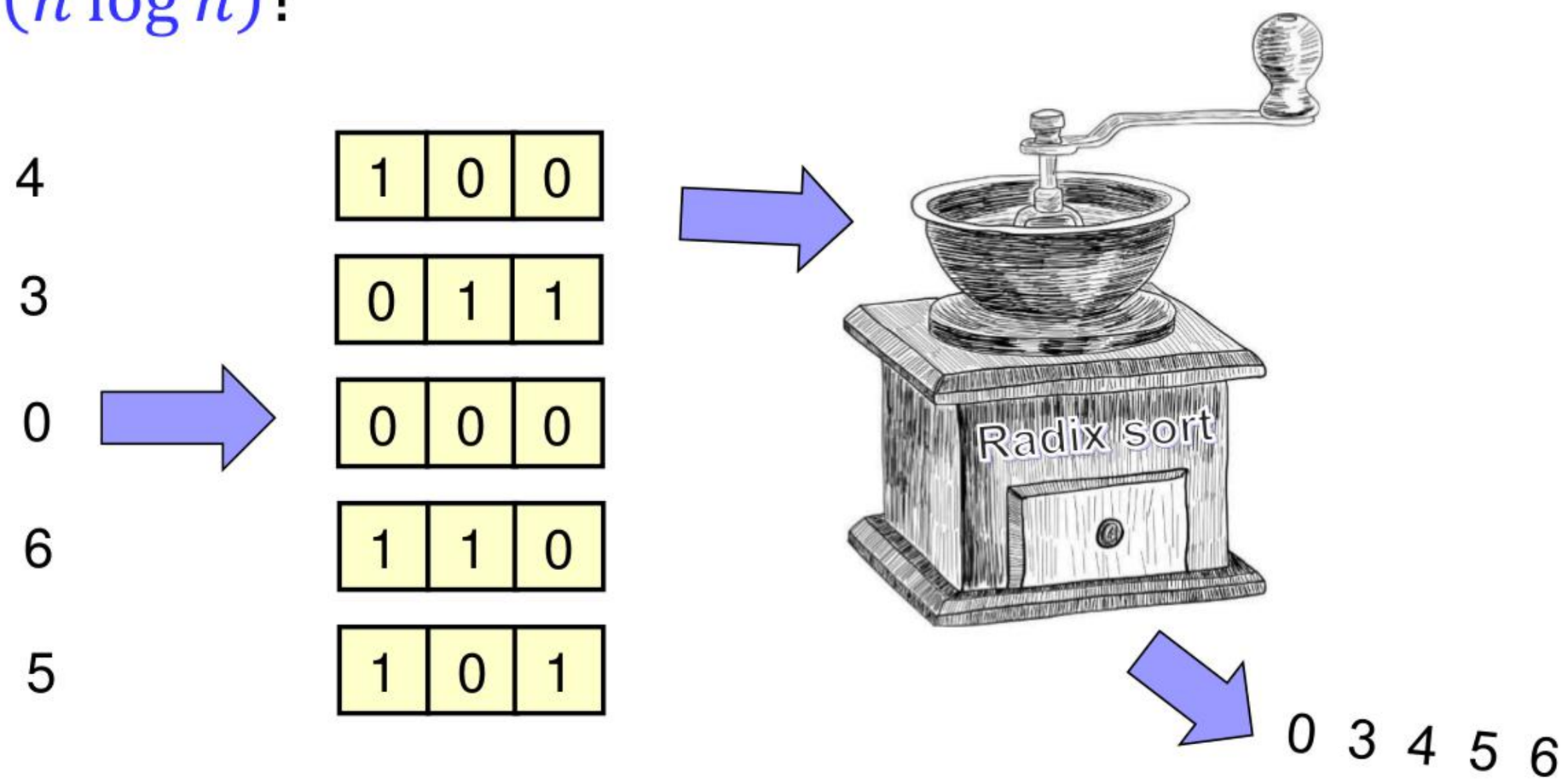
- Řadíme n řetězců délky k nad abecedou Σ.
- Časová složitost:
  - jeden krok … $\Theta(n + |\Sigma|)$
  - k kroků … $\Theta\big(k \cdot (n + |\Sigma|)\big) = \Theta(k \cdot n)$ (pro fixní abecedu)
- Stabilní řazení.



Radix sort – zamyšlení

Každé celé nezáporné číslo můžeme vyjádřit jako binární řetězec (doplněný zleva nulami pro dosažení jednotné délky).

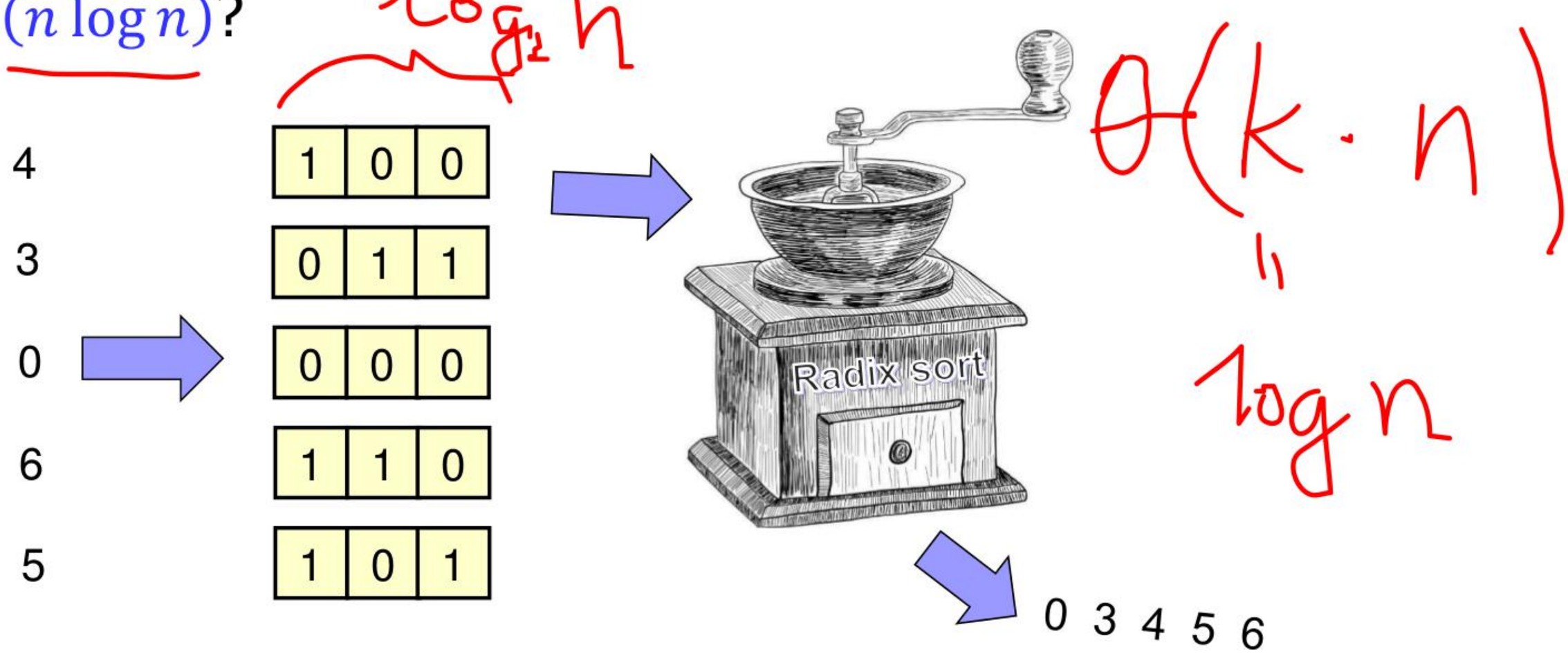
Pokud setřídíme n různých celých nezáporných čísel pomocí Radix sortu, dosáhneme lepší asymptotické složitosti než $O(n \log n)$?



Radix sort – zamyšlení

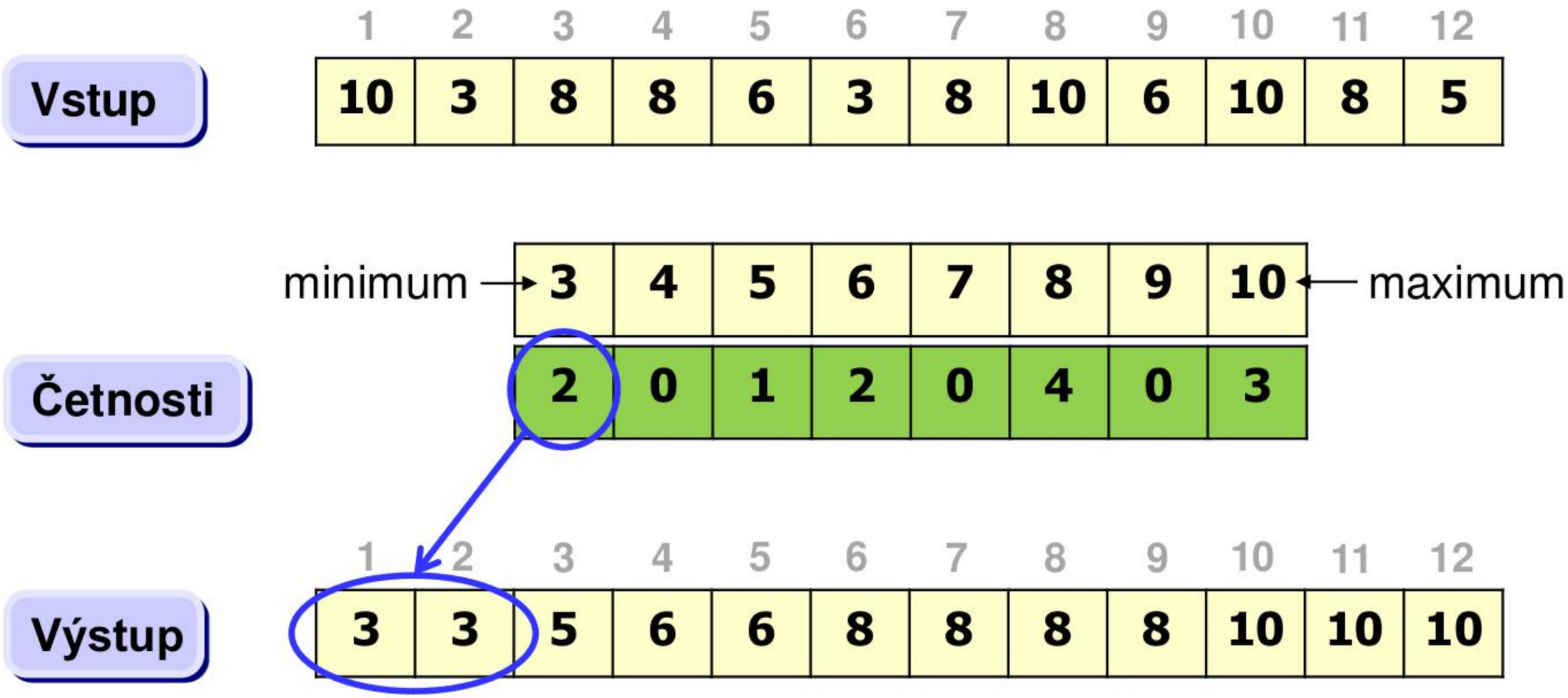
Každé celé nezáporné číslo můžeme vyjádřit jako binární řetězec (doplněný zleva nulami pro dosažení jednotné délky).

Pokud setřídíme n různých celých nezáporných čísel pomocí Radix sortu, dosáhneme lepší asymptotické složitosti než $O(n \log n)$?



Counting sort (řazení počítáním)

- Vhodný pro řazení velkého pole prvků nabývajících jen malého počtu různých diskrétních hodnot.



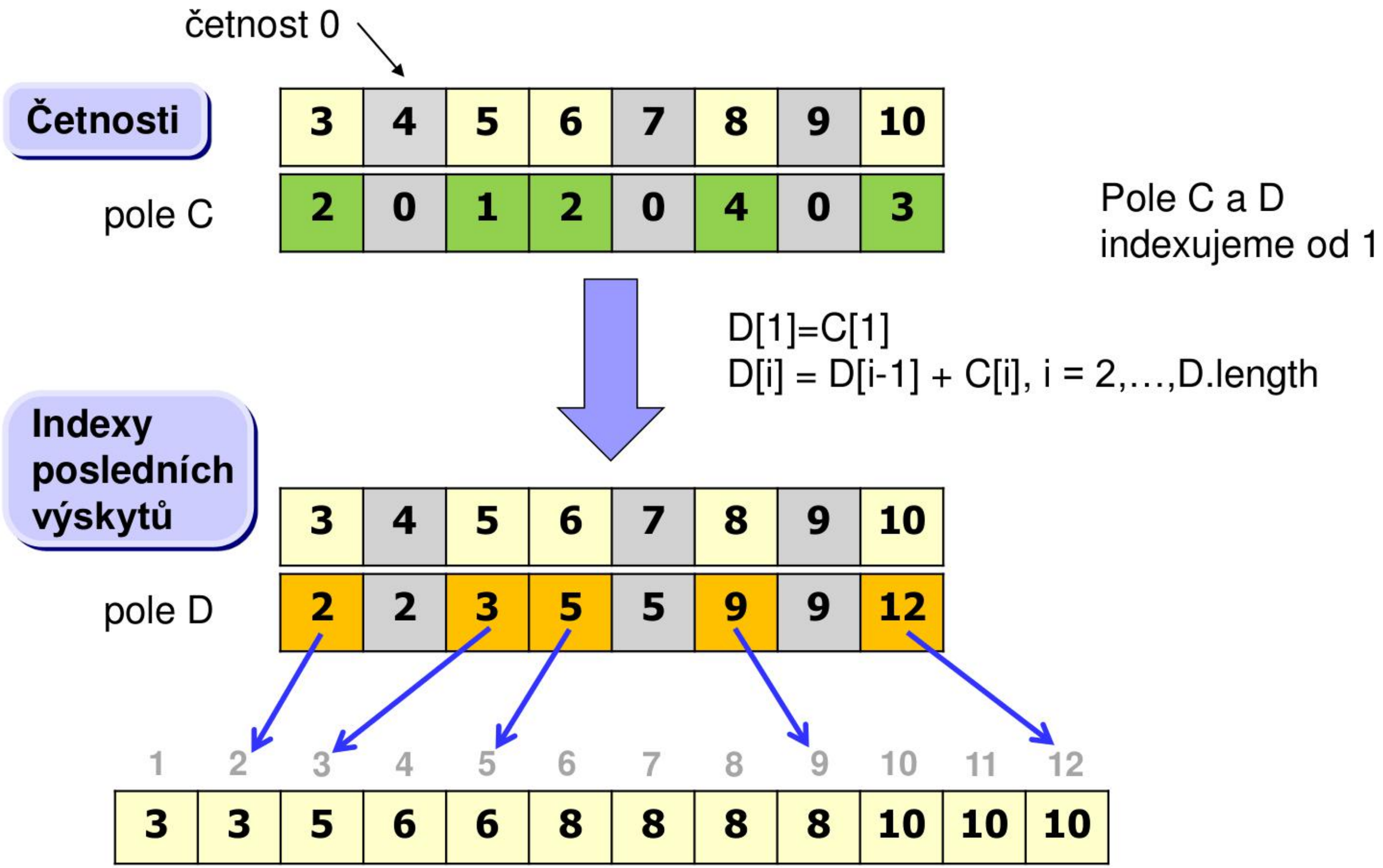
Pokud ale řadíme objekty, musíme postupovat jinak.

Counting sort (řazení počítáním)

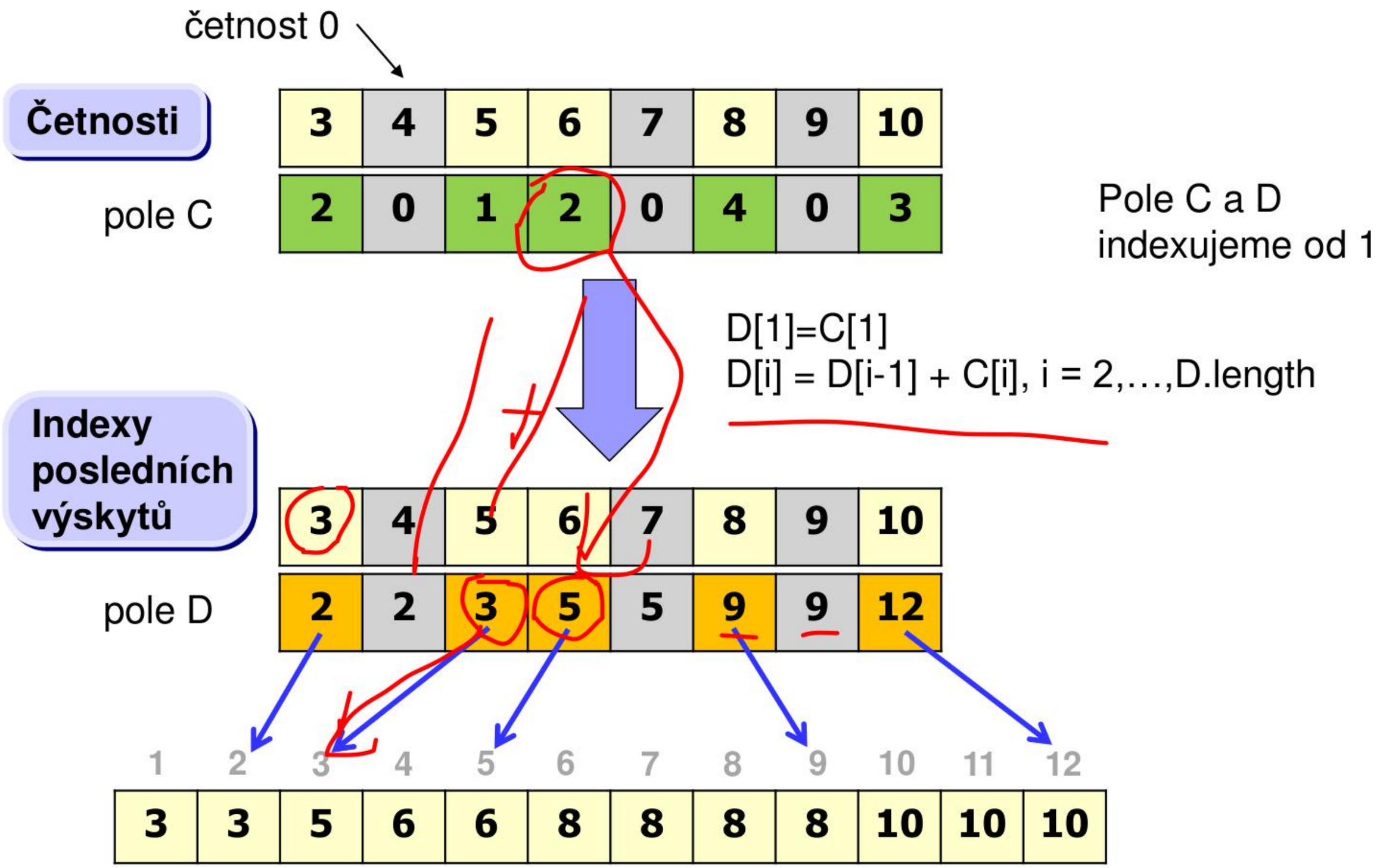
- Vhodný pro řazení velkého pole prvků nabývajících jen malého počtu různých diskrétních hodnot.

Pokud ale řadíme objekty, musíme postupovat jinak.

Counting sort



Counting sort



Counting sort

Vstup

1	2	3	4	5	6	7	8	9	10	11	12
10	3	8	8	6	3	8	10	6	10	8	5

Prvky řadíme na jejich pozici pozpátku

Indexy posledních výskytů

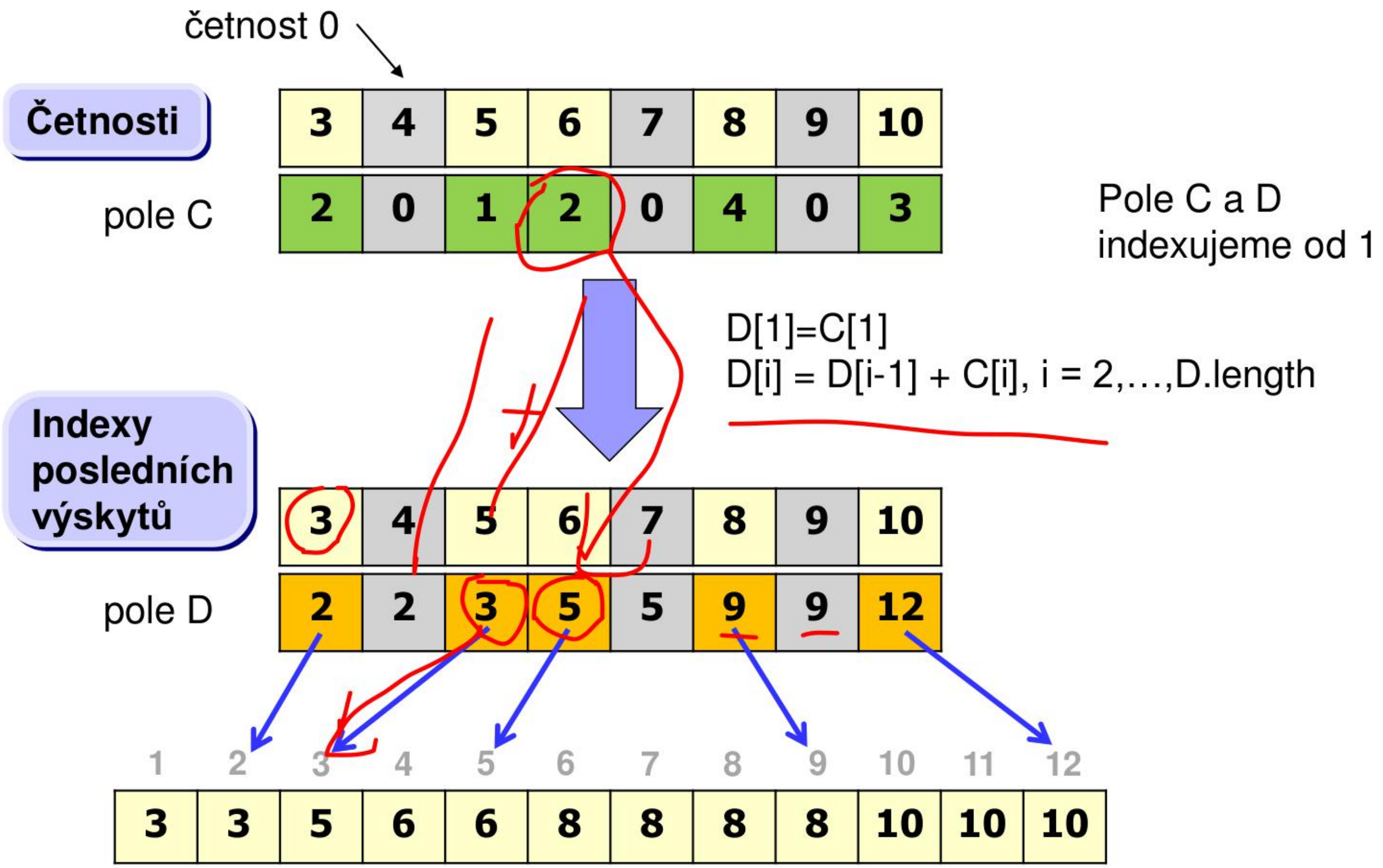
3	4	5	6	7	8	9	10
2	2	3	5	5	9	9	12

Index snížeme o 1

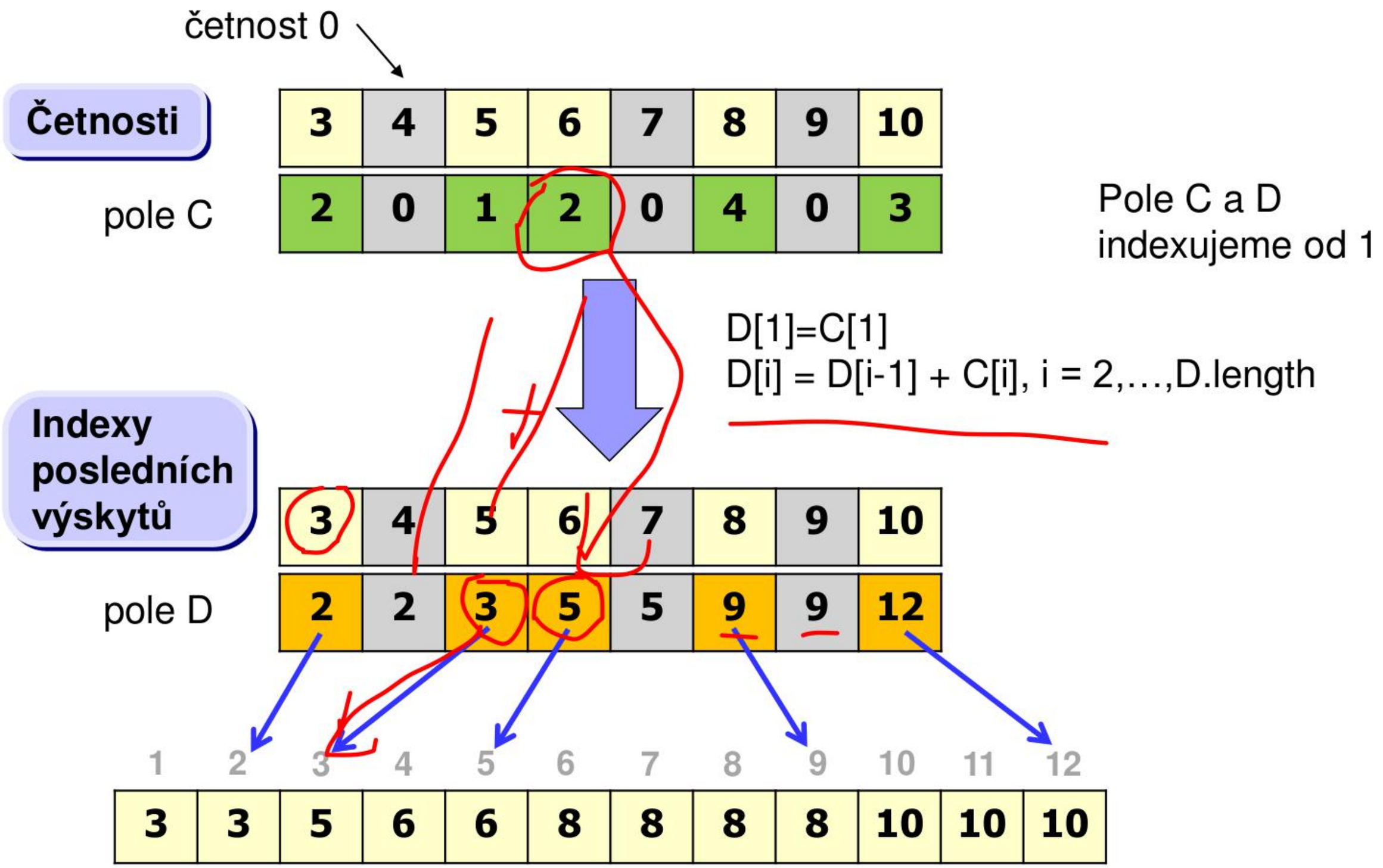
Výstup

1	2	3	4	5	6	7	8	9	10	11	12
		5									

Counting sort



Counting sort



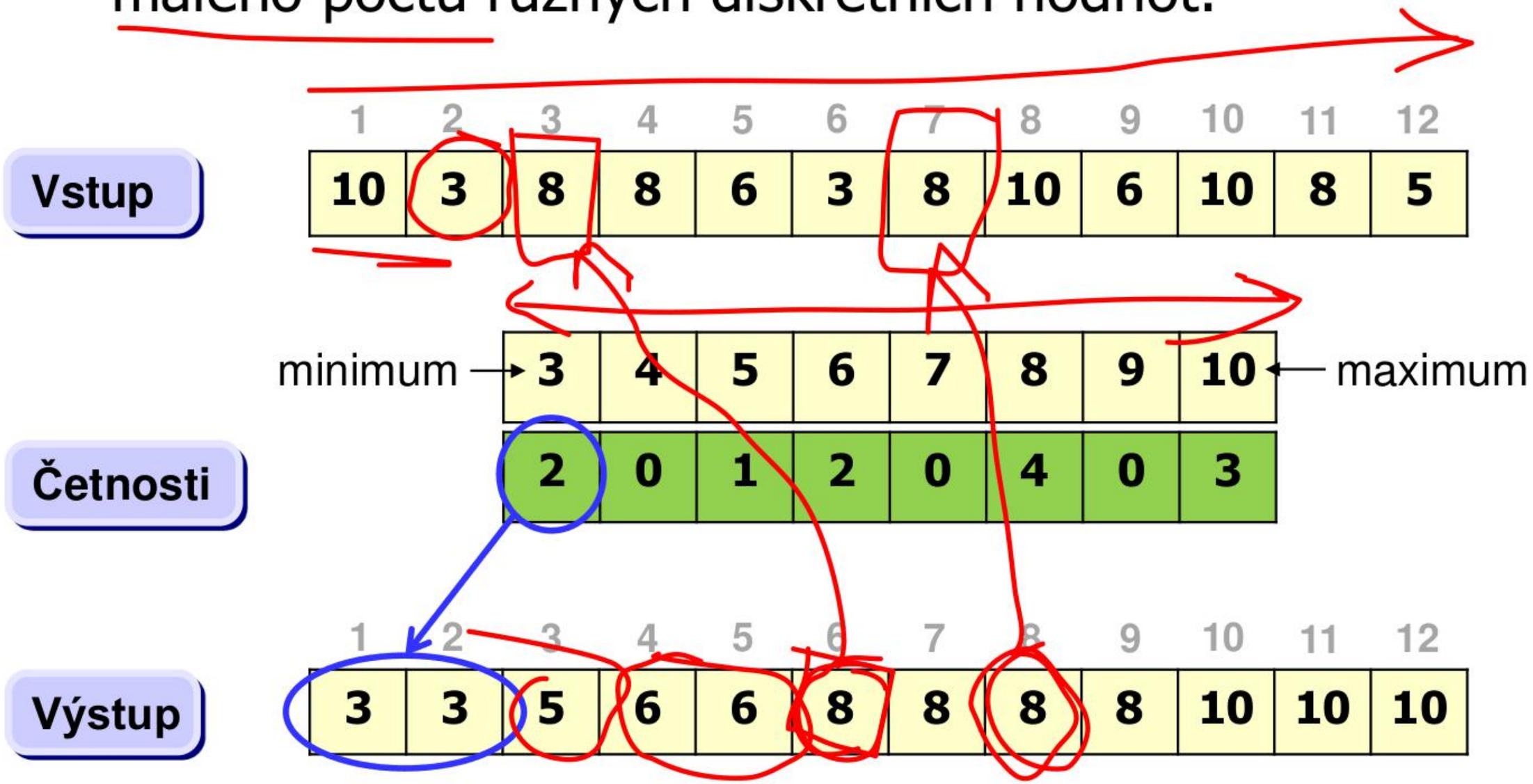
Counting sort (řazení počítáním)

- Vhodný pro řazení velkého pole prvků nabývajících jen malého počtu různých diskrétních hodnot.

Pokud ale řadíme objekty, musíme postupovat jinak.

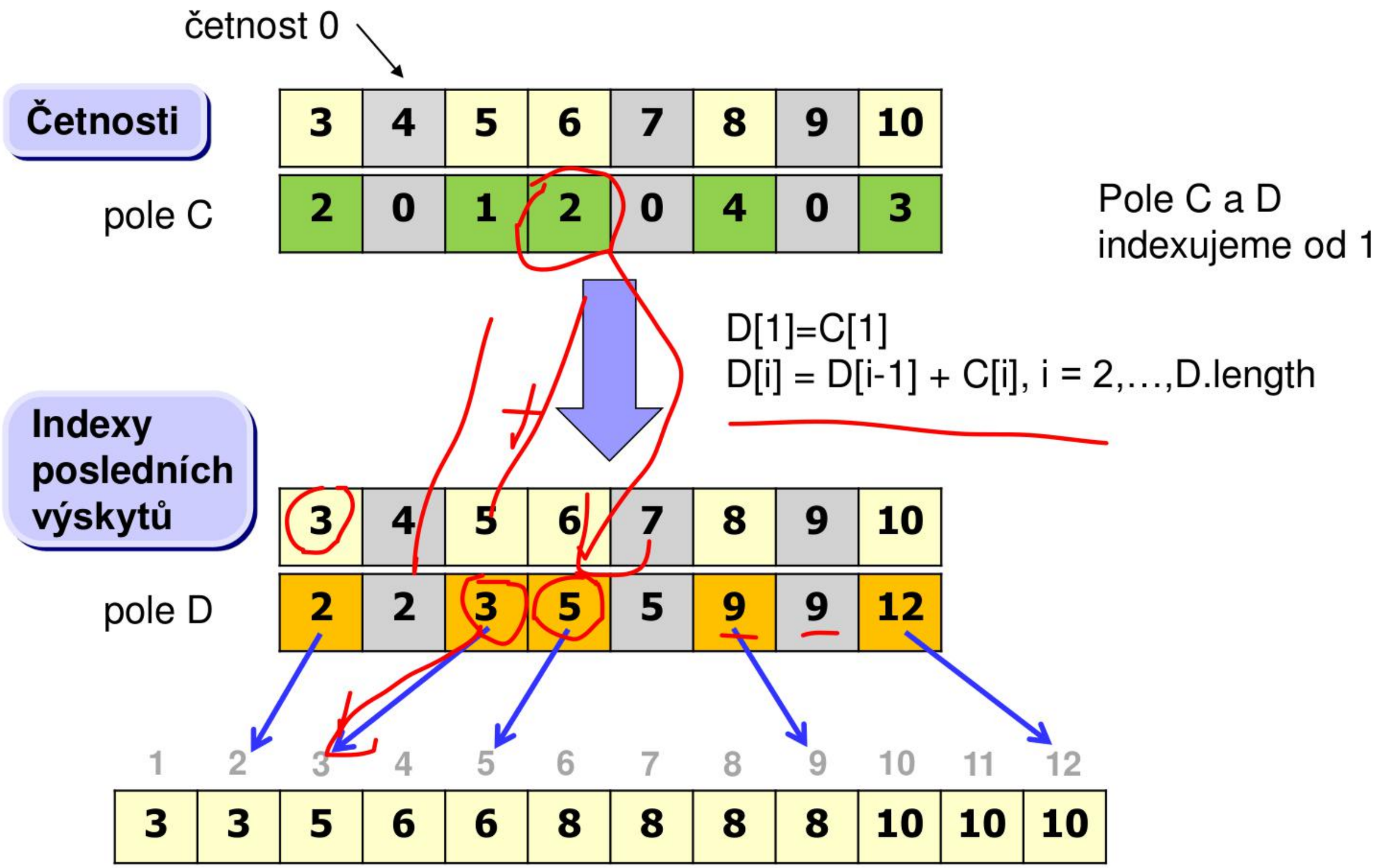
Counting sort (řazení počítáním)

- Vhodný pro řazení velkého pole prvků nabývajících jen malého počtu různých diskrétních hodnot.

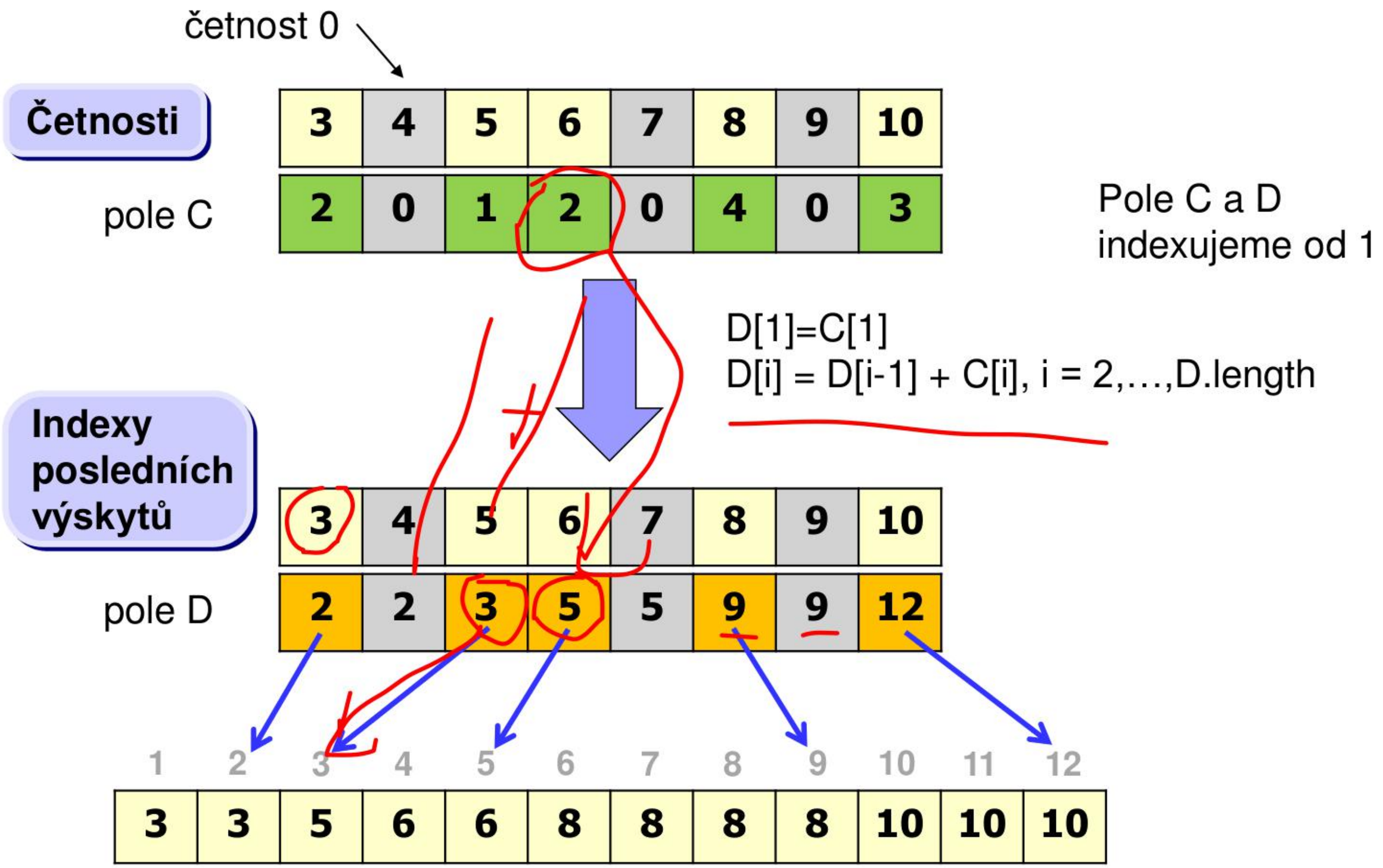


Pokud ale řadíme objekty, musíme postupovat jinak.

Counting sort



Counting sort



Counting sort

Vstup

1	2	3	4	5	6	7	8	9	10	11	12
10	3	8	8	6	3	8	10	6	10	8	5

Prvky řadíme na jejich pozici pozpátku

Indexy posledních výskytů

3	4	5	6	7	8	9	10
2	2	3	5	5	9	9	12

Index snížeme o 1

Výstup

1	2	3	4	5	6	7	8	9	10	11	12
		5									

Counting sort

Vstup

1	2	3	4	5	6	7	8	9	10	11	12
10	3	8	8	6	3	8	10	6	10	8	5

Prvky řadíme na jejich pozici pozpátku

Indexy posledních výskytů

3	4	5	6	7	8	9	10
2	2	3	5	5	9	9	12

Index snížeme o 1

Výstup

1	2	3	4	5	6	7	8	9	10	11	12
		5									

Counting sort

Vstup

1	2	3	4	5	6	7	8	9	10	11	12
10	3	8	8	6	3	8	10	6	10	8	5

Prvky řadíme na jejich pozici pozpátku

Indexy posledních výskytů

3	4	5	6	7	8	9	10
2	2	2	5	5	9	9	12

Index snížeme o 1

Výstup

1	2	3	4	5	6	7	8	9	10	11	12
		5						8			

Counting sort

Vstup

1	2	3	4	5	6	7	8	9	10	11	12
10	3	8	8	6	3	8	10	6	10	8	5

Prvky řadíme na jejich pozici pozpátku

Indexy posledních výskytů

3	4	5	6	7	8	9	10
2	2	2	5	5	9	9	12

Index snížeme o 1

Výstup

1	2	3	4	5	6	7	8	9	10	11	12
		5						8			

Counting sort

Vstup

1	2	3	4	5	6	7	8	9	10	11	12
10	3	8	8	6	3	8	10	6	10	8	5

Prvky řadíme na jejich pozici pozpátku

Indexy posledních výskytů

3	4	5	6	7	8	9	10
2	2	2	5	5	8	9	12

Index snížeme o 1

Výstup

1	2	3	4	5	6	7	8	9	10	11	12
		5						8			10

...

Counting sort

Vstup

1	2	3	4	5	6	7	8	9	10	11	12
10	3	8	8	6	3	8	10	6	10	8	5

Prvky řadíme na jejich pozici pozpátku

Indexy posledních výskytů

3	4	5	6	7	8	9	10
2	2	2	5	5	8	9	12

Index snížeme o 1

Výstup

1	2	3	4	5	6	7	8	9	10	11	12
		5		6				8		10	10

...

Counting sort

- Řadíme n prvků jejichž celočíselné klíče jsou z intervalu délky m.
- Časová složitost je $\Theta(n + m)$.
- Místo pole četností lze použít HashMap.
- Stabilní řazení.

Counting sort

- Řadíme n prvků jejichž celočíselné klíče jsou z intervalu délky m.
- Časová složitost je $\Theta(n + m)$.
- Místo pole četností lze použít HashMap.
- Stabilní řazení.

Šestá domácí úloha – změna

```
   zadání -- odevzdání
1.   25.9. -- 18.10.   ( Asymptotic complexity )
2.   9.10. --  1.11.   ( Recursion/Backtrack )
3.  16.10. --  8.11.   ( Tree search )
4.  30.10. -- 22.11.   ( Graph search )
5.   6.11. -- 29.11.   ( BST processing )
6.  13.11. -- 13.12.   ( AVL processing ) Divide and conquer
7.  27.11. -- 20.12.   ( Dynamic Programming )
8.  11.12. -- 10.1.    ( Dynamic Programming )
```

https://cw.fel.cvut.cz/wiki/courses/b4b33alg/cviceni

Šestá domácí úloha – změna

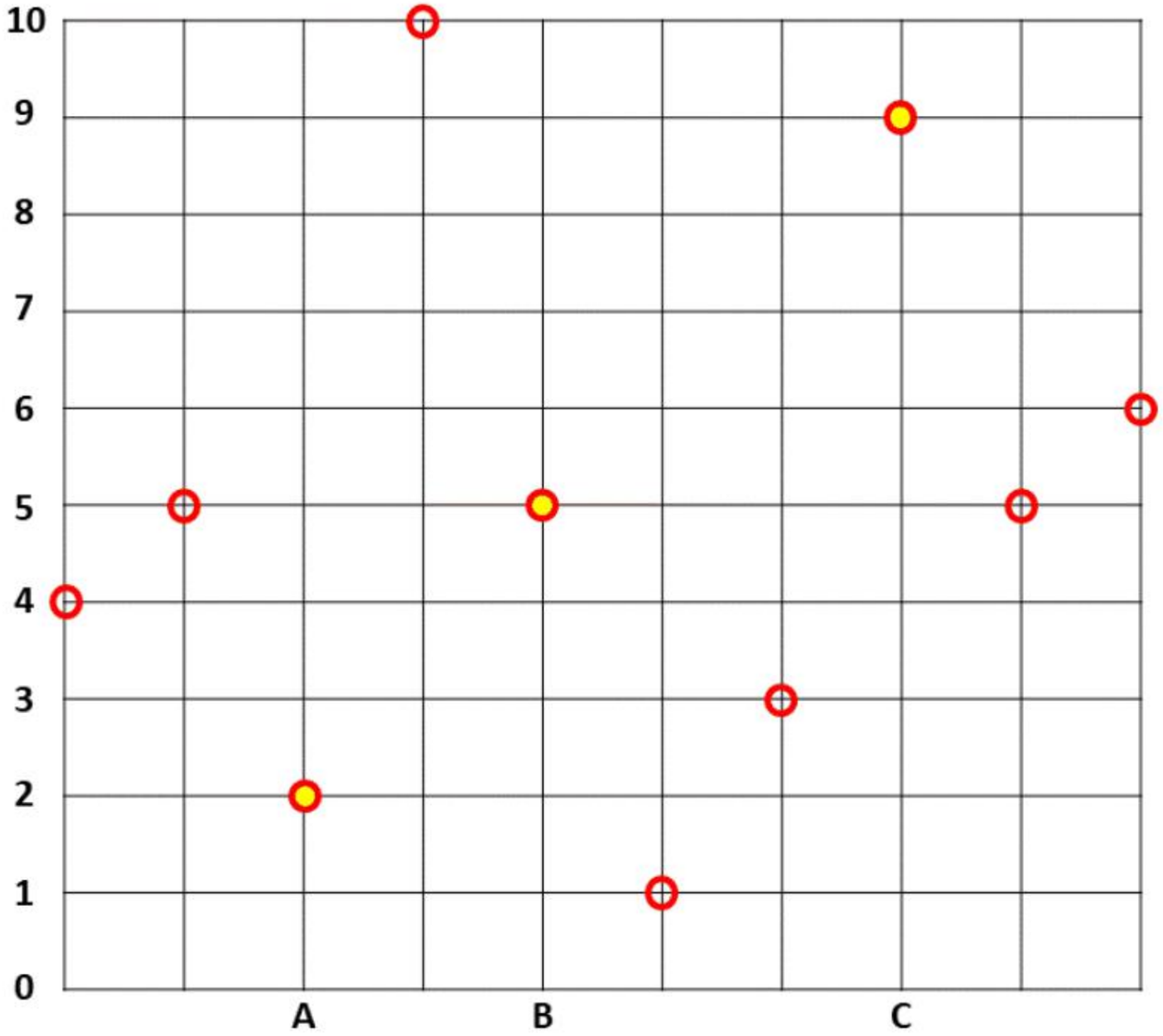
```
   zadání -- odevzdání
1.   25.9. -- 18.10.   ( Asymptotic complexity )
2.   9.10. --  1.11.   ( Recursion/Backtrack )
3.  16.10. --  8.11.   ( Tree search )
4.  30.10. -- 22.11.   ( Graph search )
5.   6.11. -- 29.11.   ( BST processing )
6.  13.11. -- 13.12.   ( AVL processing )
7.  27.11. -- 27.12.   ( Dynamic Programming )
8.  11.12. -- 10.1.    ( Dynamic Programming )
```

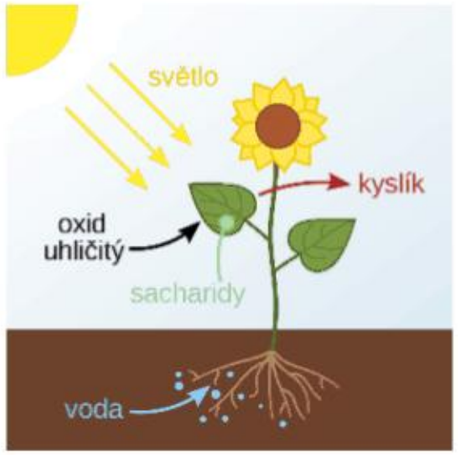
Divide and conquer

https://cw.fel.cvut.cz/wiki/courses/b4b33alg/cviceni

Šestá domácí úloha

- Je dána předpověď teplot $T_1,\ldots,T_N$ pro N následujících dnů.
- Chceme nalézt tři dny A<B<C, které maximalizují výraz $\min\{T_B-T_A, T_C-T_B\}$.



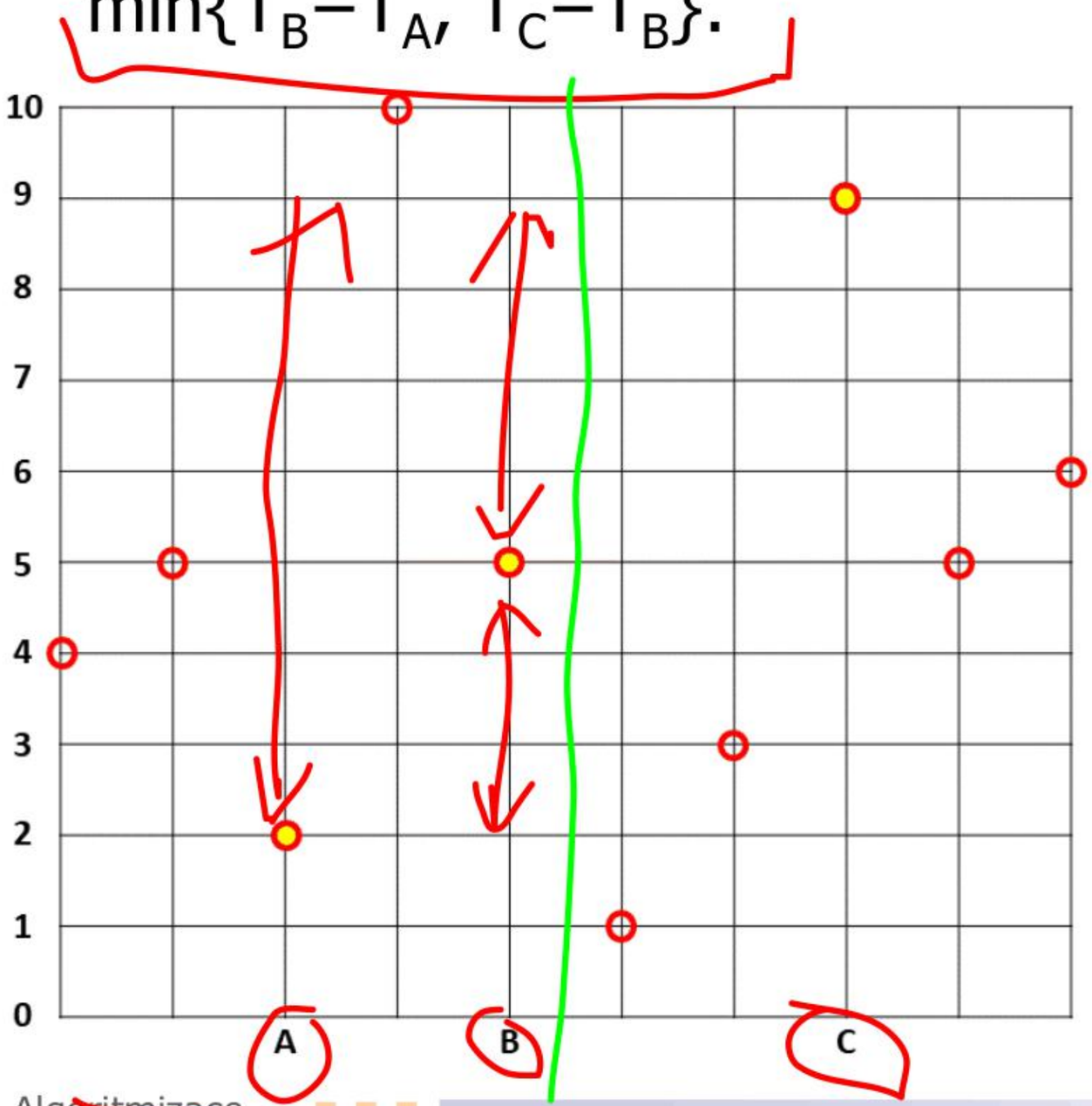


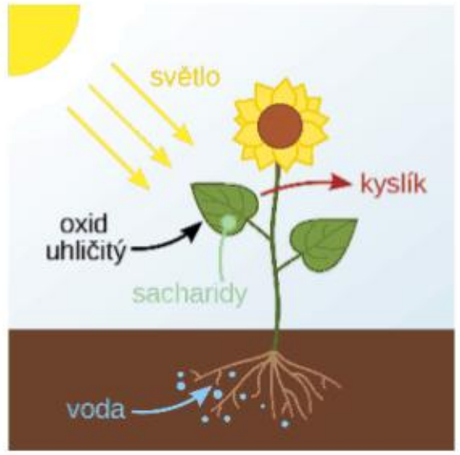
Jak postupovat: Rozděl a panuj, zobecnit příklad s akciemi.

Jak nepostupovat: Hrubá síla, $\Theta(N^3)$ (N je až 400000).

Šestá domácí úloha

- Je dána předpověď teplot $T_1,\ldots,T_N$ pro N následujících dnů.
- Chceme nalézt tři dny A<B<C, které maximalizují výraz $\min\{T_B-T_A, T_C-T_B\}$.

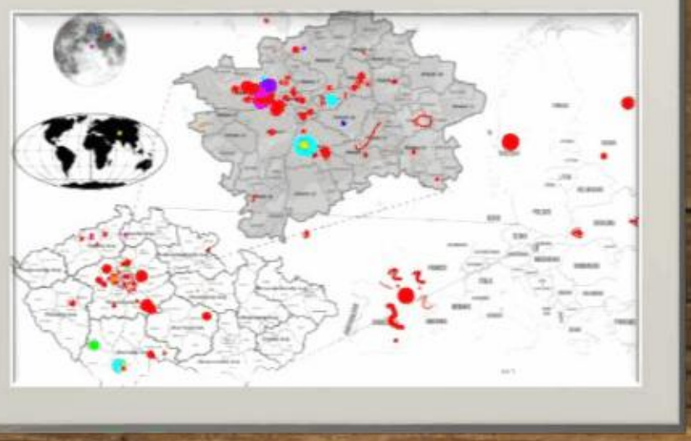


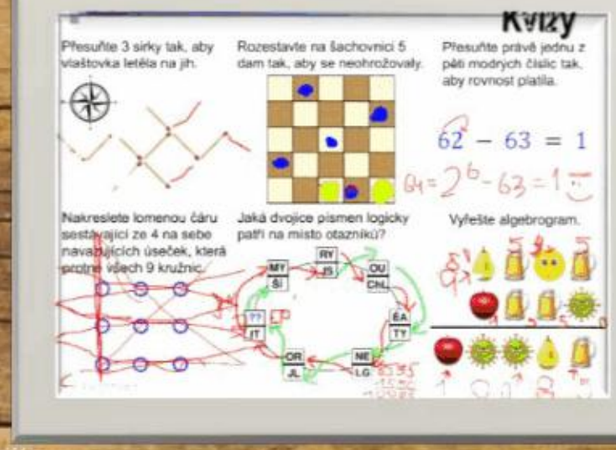


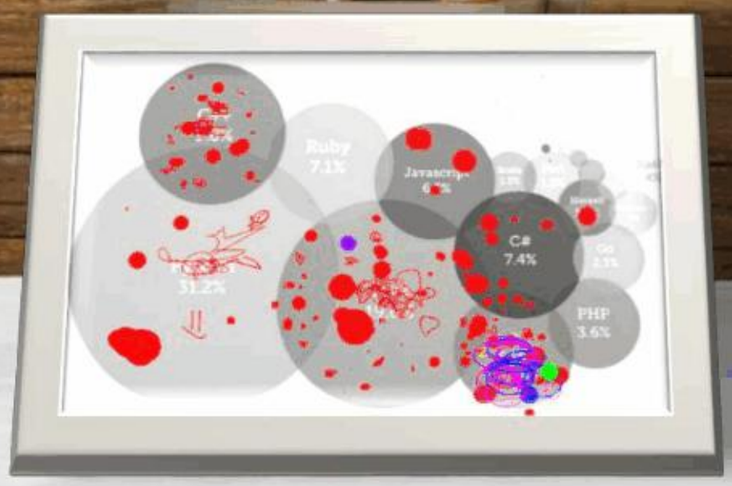
Jak postupovat: Rozděl a panuj, zobecnit příklad s akciemi.

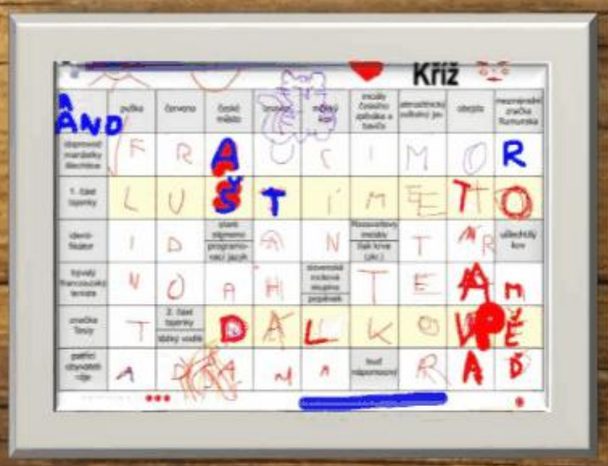
Jak nepostupovat: Hrubá síla, $\Theta(N^3)$ (N je až 400000).


ALG - 2020




Kvízy
Přesuňte 3 sirky tak, aby vlaštovka letěla na jih.
Rozestavte na šachovnici 5 dam tak, aby se neohrožovaly.
Přesuňte právě jednu z pěti modrých číslic tak, aby rovnost platila.
62 − 63 = 1
64 = 2^6 − 63 = 1
Nakreslete lomenou čáru sestávající ze 4 na sebe navazujících úseček, která protne všech 9 kružnic.
Jaká dvojice písmen logicky patří na místo otazníků?
Vyřešte algebrogram.


Ruby
7.1%
C#
7.4%
PHP
3.6%


Kříž

ALG 2020