Různé algoritmy mají různou složitost

Algoritmus a program není totéž

Stabilní řazení

záznam: | Jméno | Příjmení |

Vstup: seznam seřazen pouze podle jména

Andrew	Cook
Andrew	Amundsen
Andrew	Brown
Barbara	Cook
Barbara	Brown
Barbara	Amundsen
Charles	Amundsen
Charles	Cook
Charles	Brown

stabilní řazení

seřaď záznamy pouze podle příjmení

Výstup: seznam seřazen podle jména i příjmení

Andrew	Amundsen
Barbara	Amundsen
Charles	Amundsen
Andrew	Brown
Barbara	Brown
Charles	Brown
Andrew	Cook
Barbara	Cook
Charles	Cook

Pořadí záznamů se stejným příjmením se nezměnilo

Nestabilní řazení

záznam: | Jméno | Příjmení |

Vstup: seznam seřazen pouze podle jména

Andrew	Cook
Andrew	Amundsen
Andrew	Brown
Barbara	Cook
Barbara	Brown
Barbara	Amundsen
Charles	Amundsen
Charles	Cook
Charles	Brown

QuickSort

seřaď záznamy pouze podle příjmení

Výstup: původní pořadí jmen je ztraceno

seřazeno

Barbara	Amundsen
Andrew	Amundsen
Charles	Amundsen
Barbara	Brown
Charles	Brown
Andrew	Brown
Charles	Cook
Andrew	Cook
Barbara	Cook

Pořadí záznamů se stejným příjmením se změnilo

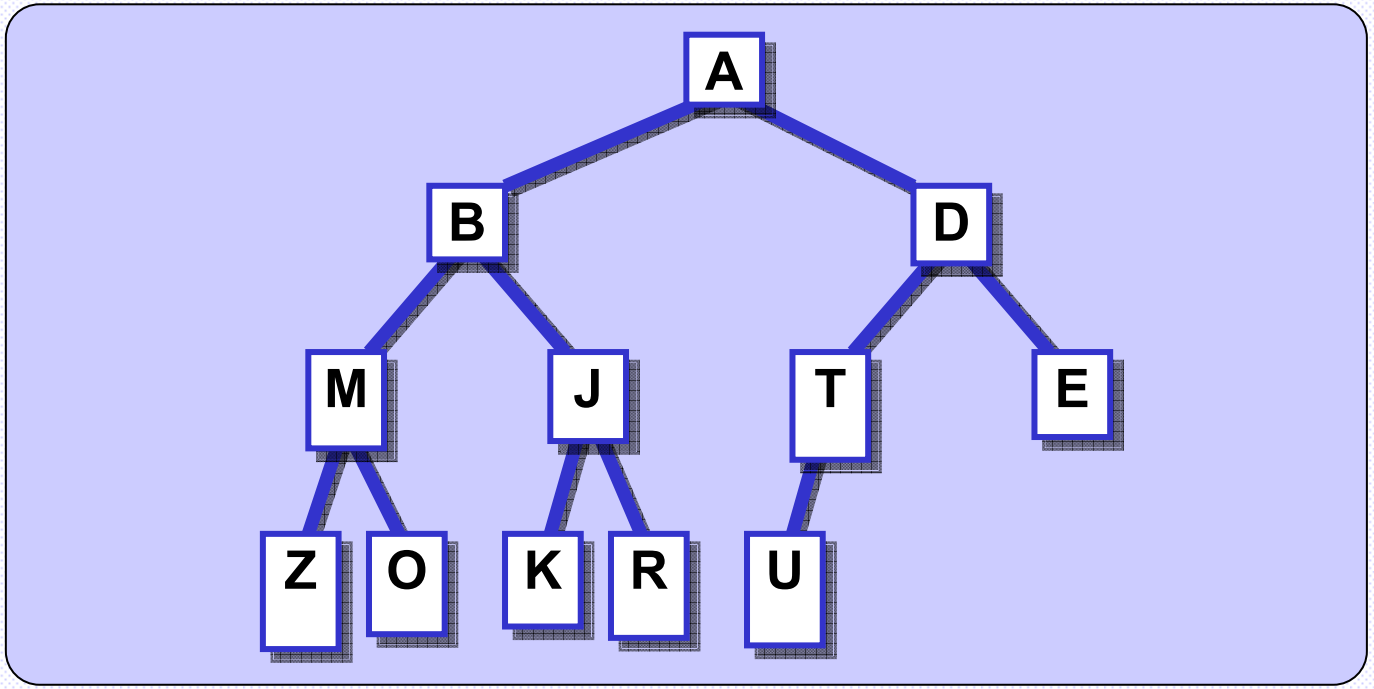
Řazení

Heap sort

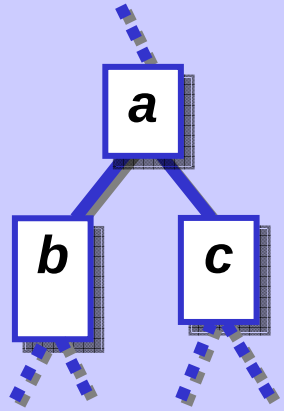
Řazení haldou

Halda

Halda



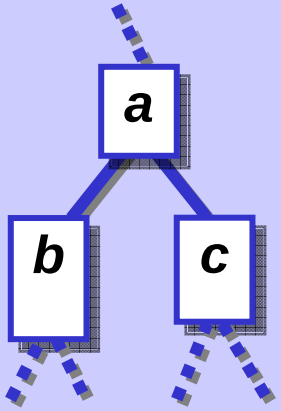
Pravidlo haldy



$$a \leq b \;\&\&\; a \leq c$$

Halda

Terminologie

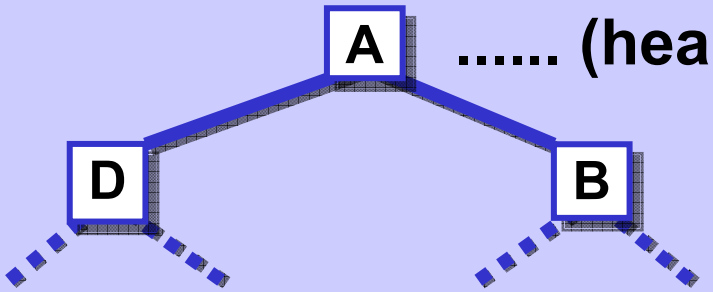


a ..... predecessor, parent of **b** **c**

..... předchůdce, rodič

b , **c** ..... successor, child of **a**

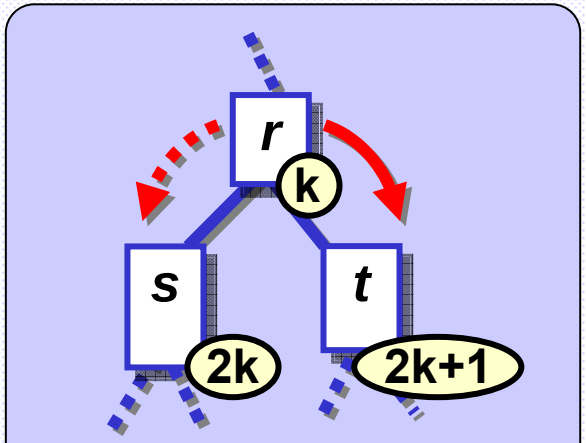
..... následník, potomek



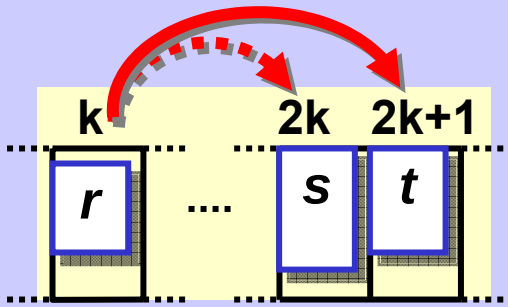
..... (heap) top ..... vrchol (haldy)

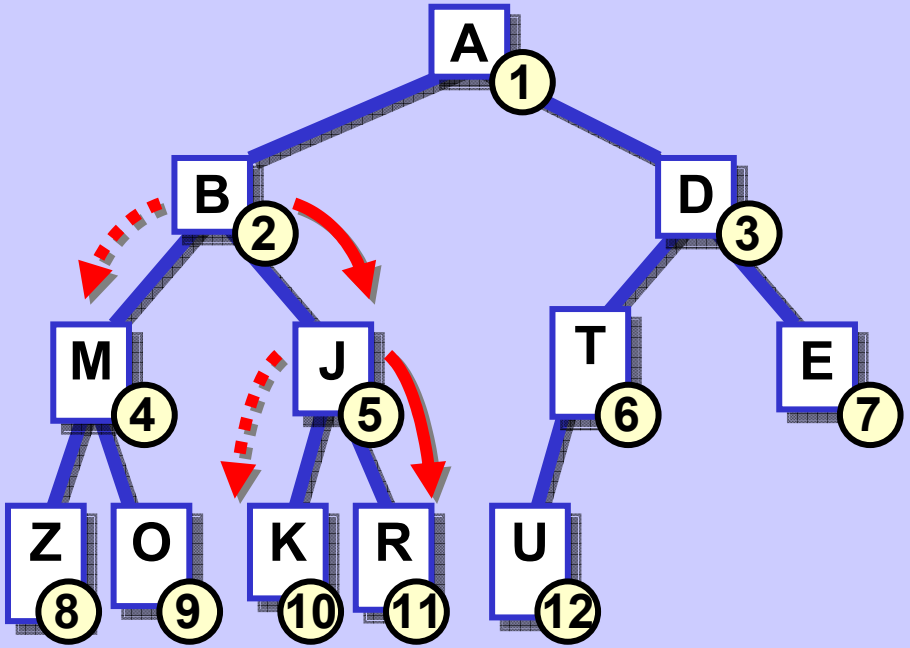
Halda v poli

Halda uložená v poli

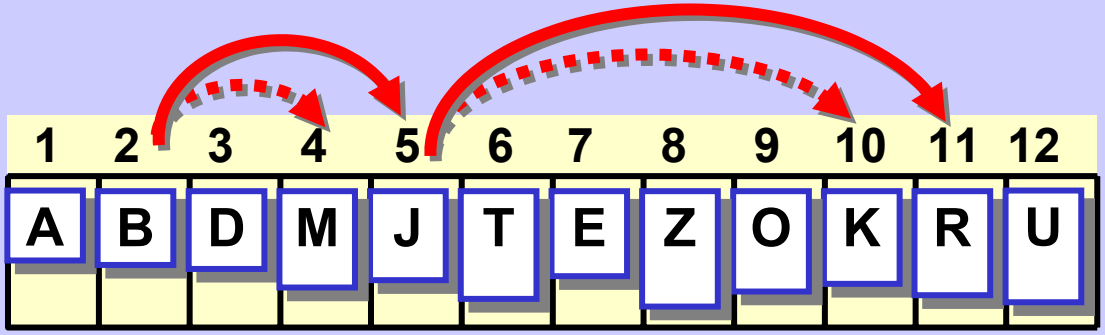


následníci



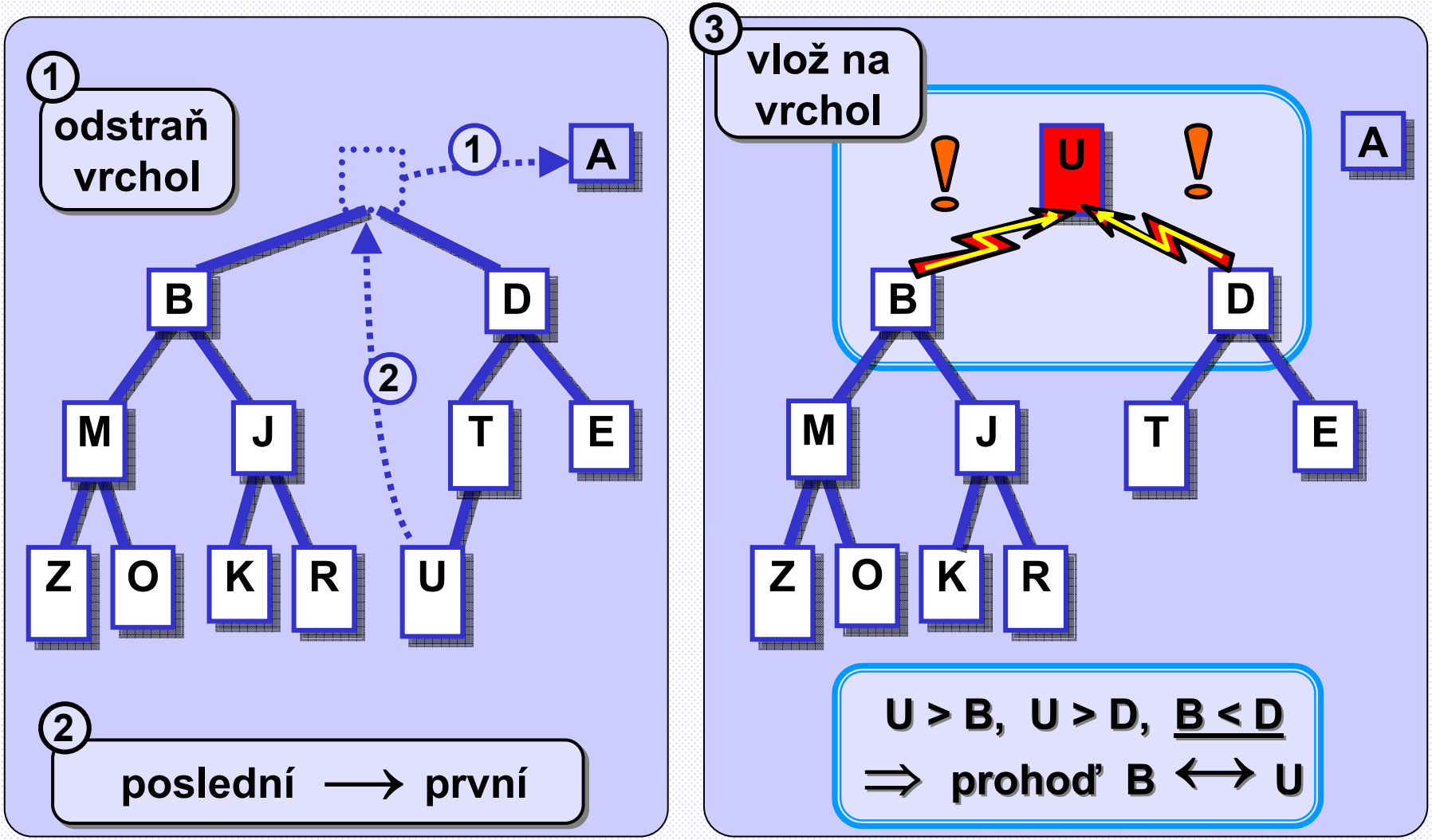


následníci



Oprava haldy

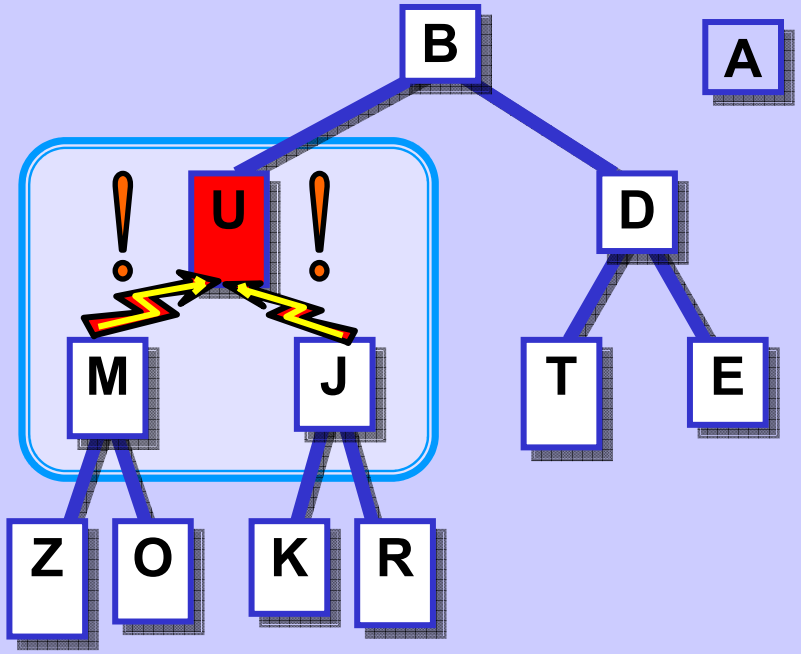
Vrchol odstraněn (1)



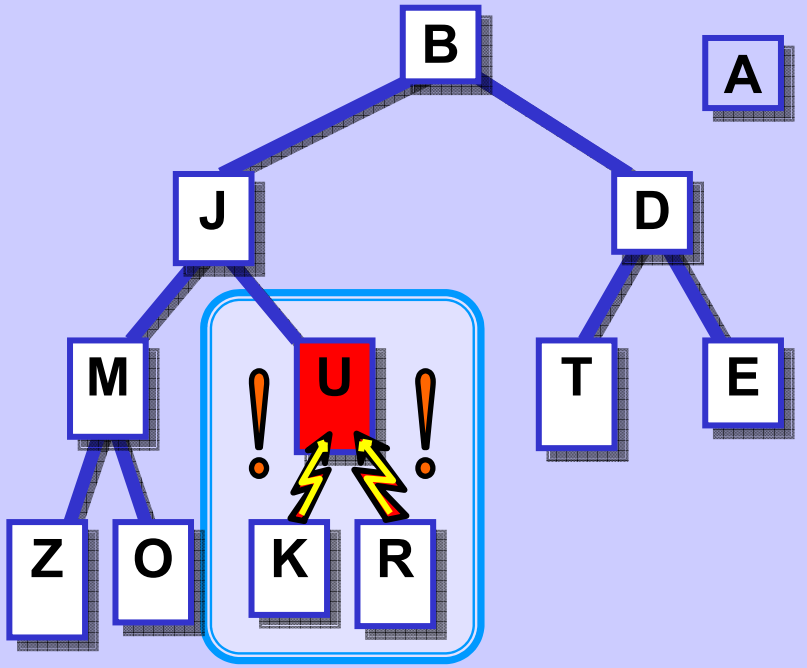
Oprava haldy

Vrchol odstraněn (2)

3 vlož na vrchol - pokračování



U > M, U > J, J < M
⇒ prohoď J ↔ U

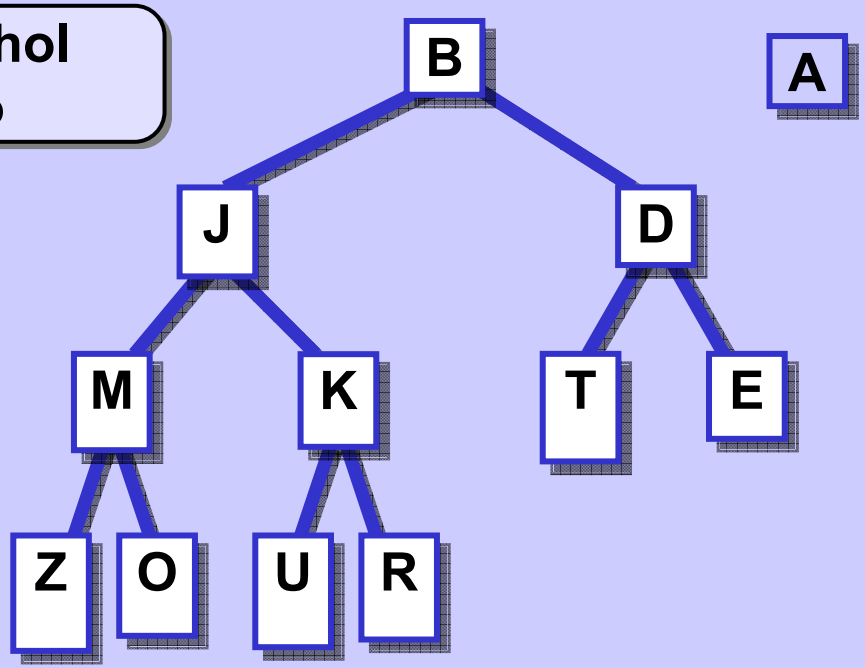


U > K, U > R, K < R
⇒ prohoď K ↔ U

Oprava haldy

Vrchol odstraněn (3)

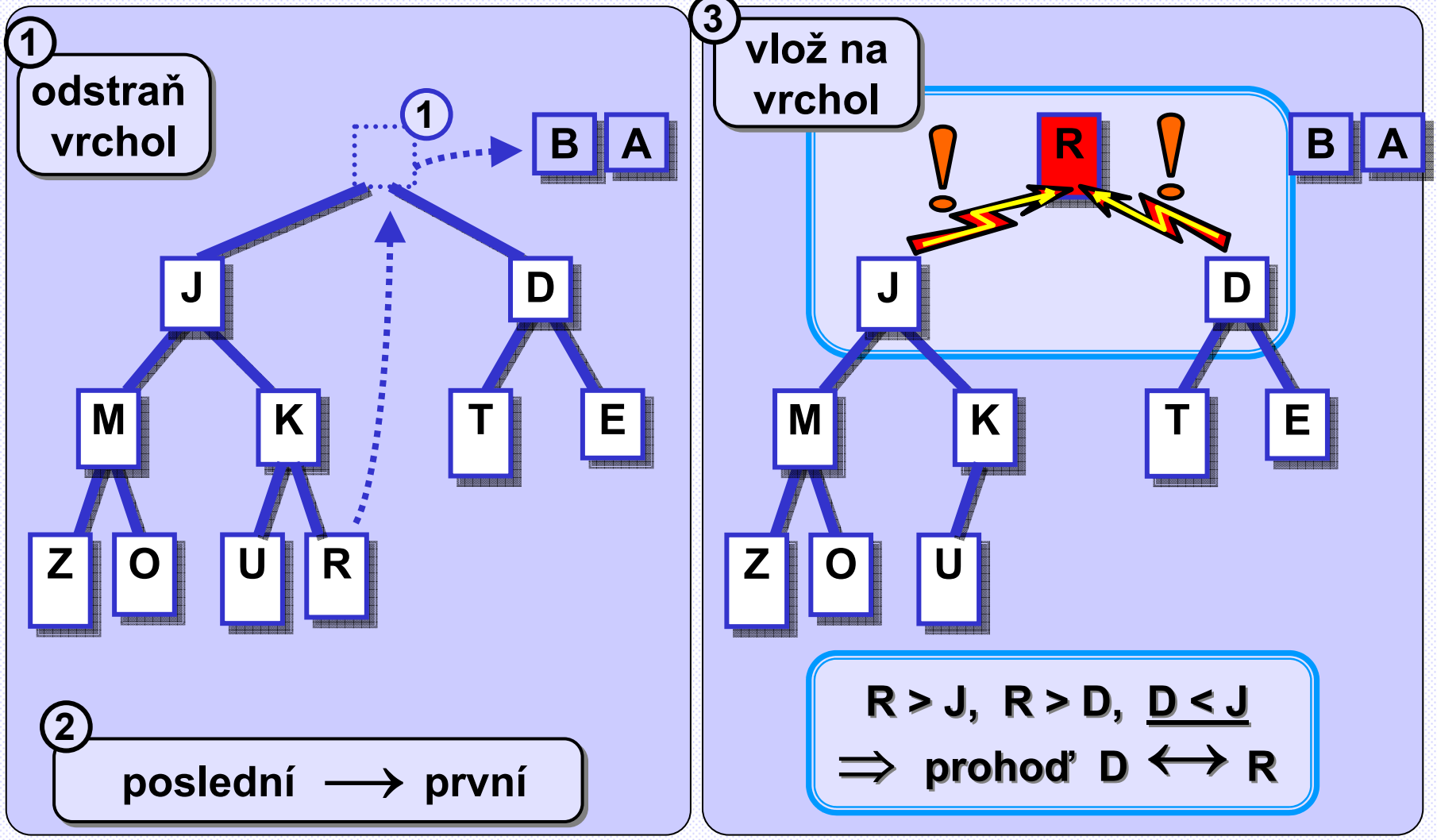
3 vlož na vrchol - hotovo



Nová halda

Oprava haldy

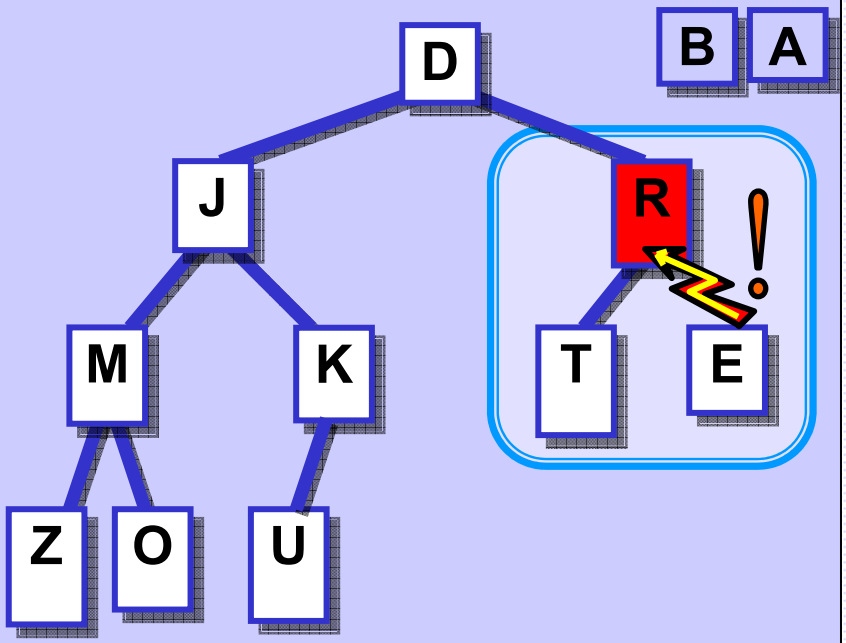
Vrchol odstraněn II (1)



Oprava haldy

Vrchol odstraněn II (2)

3 vlož na vrchol - pokračování

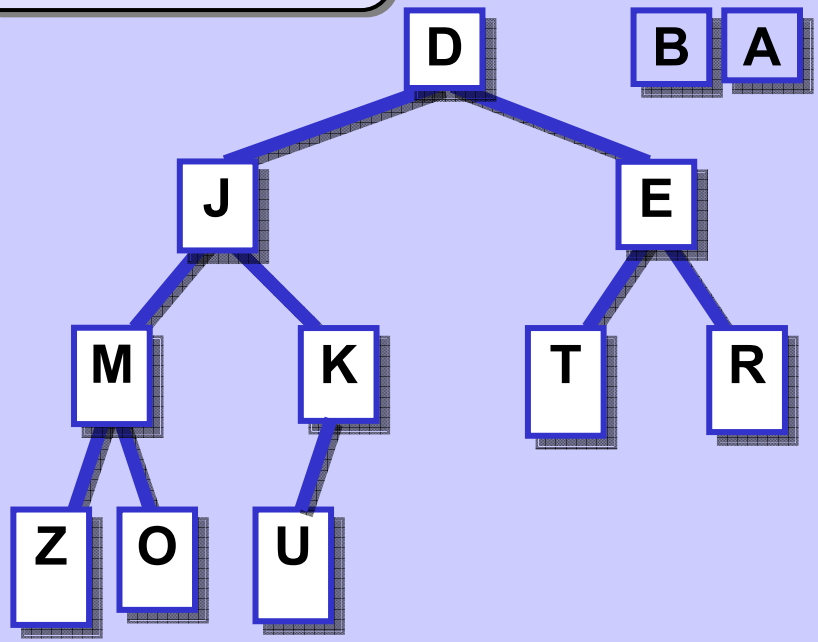


R < T, R > E

⇒ prohoď E ⟷ R

Vrchol odstraněn II (3)

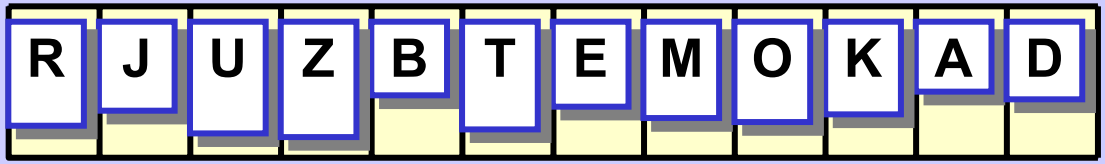
3 vlož na vrchol - hotovo



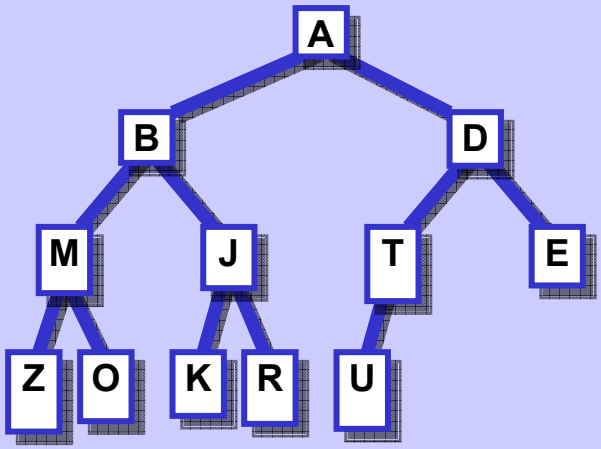
Nová halda

Heap sort

I Neseřazeno

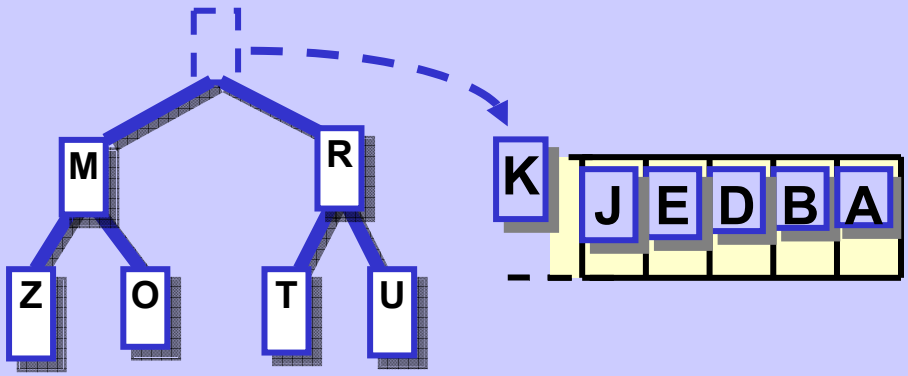


II Vytvoř haldu

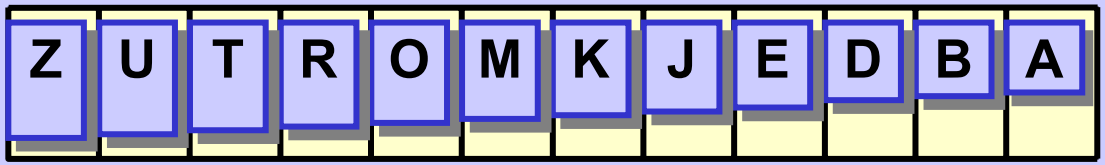


III

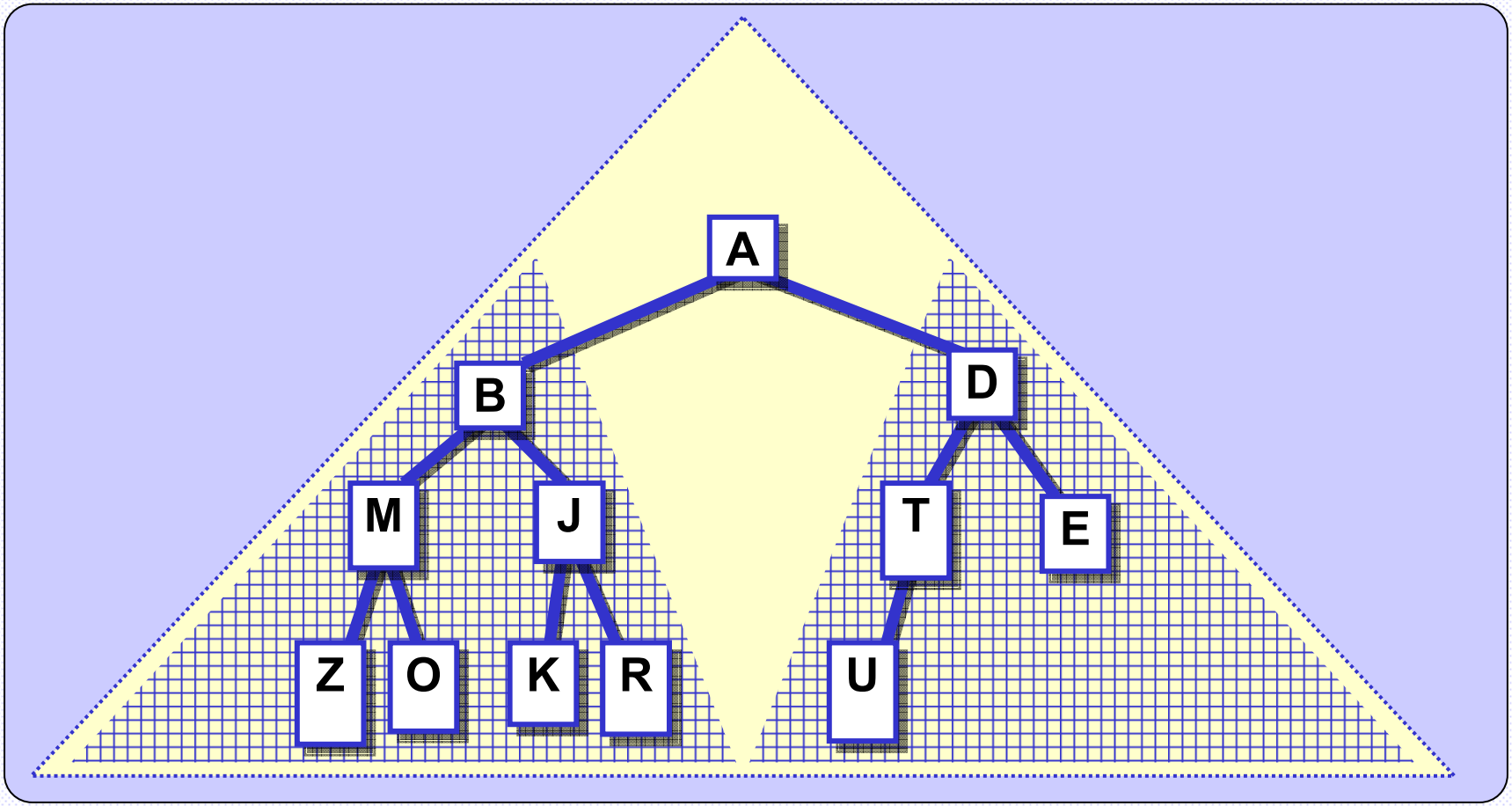
```
for (i = 0; i < n; i++)
    a[i] = “odstraň vrchol”;
```



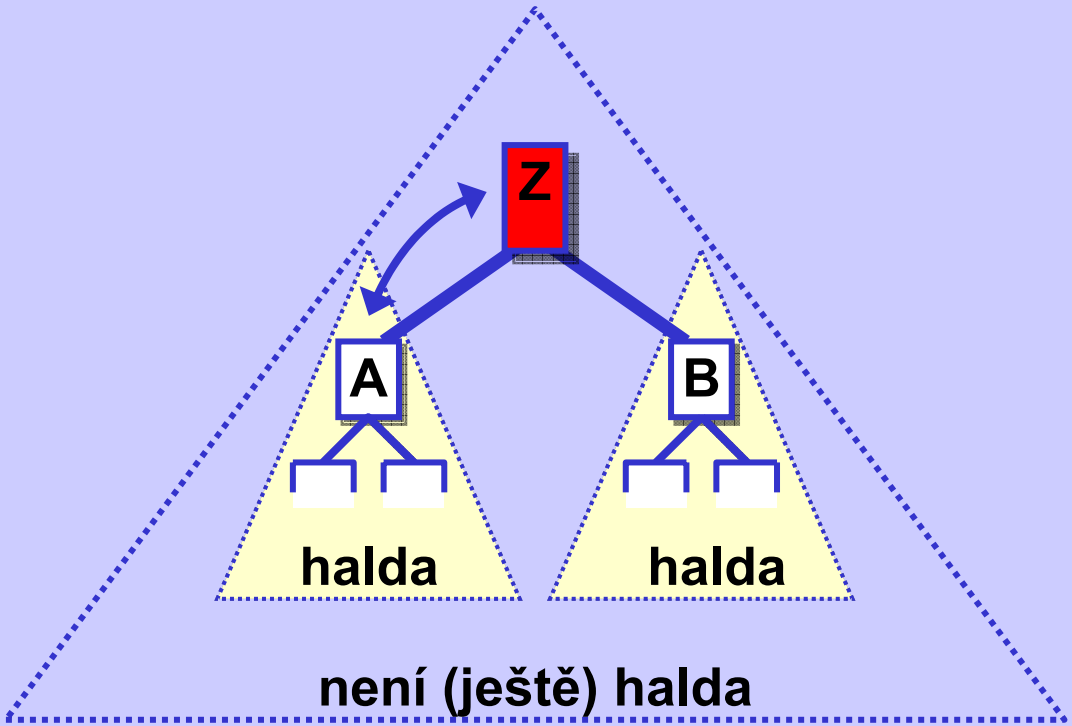
IV Seřazeno



Rekurzivní vlastnost „býti haldou”



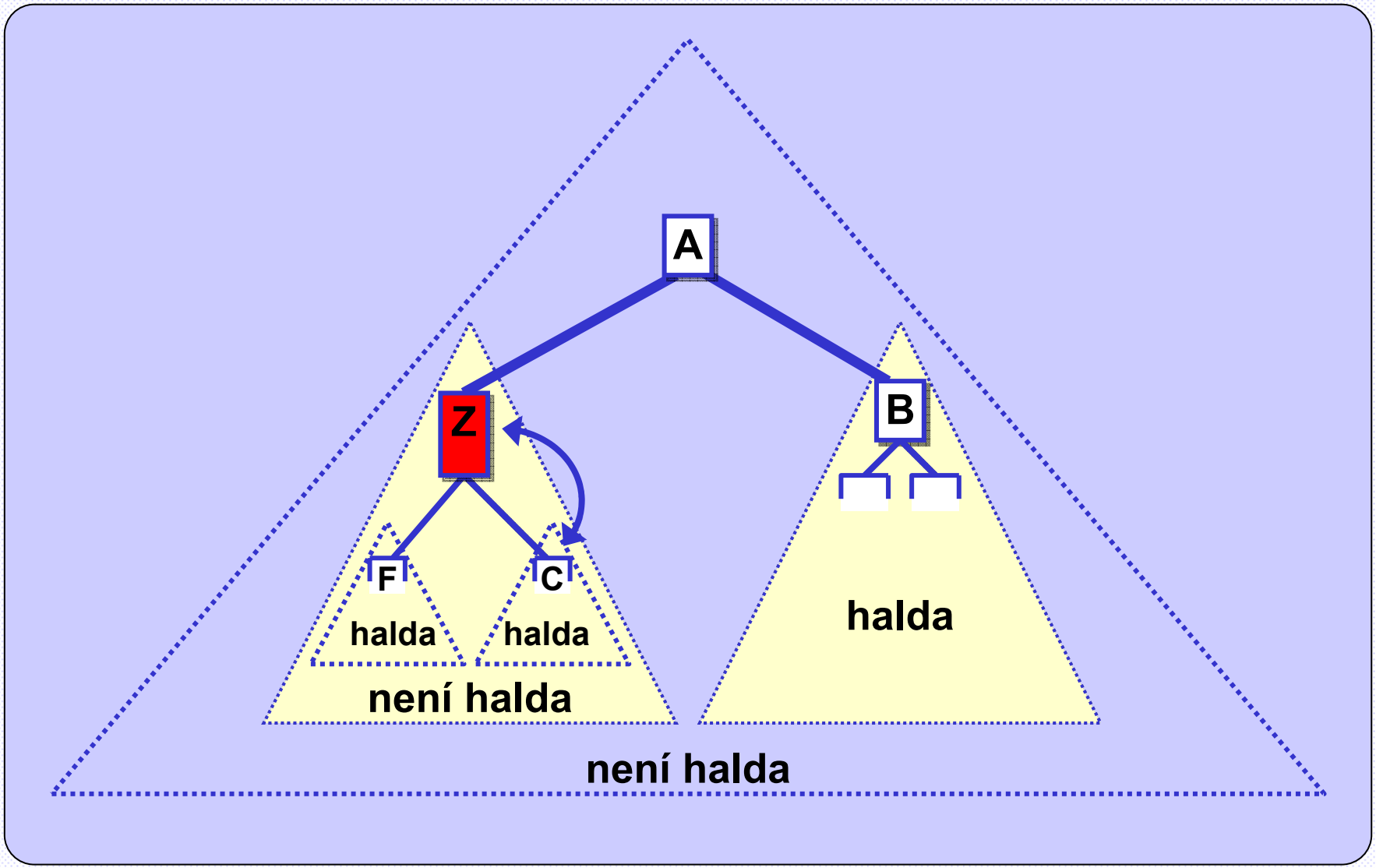
Vytvoř jednu haldu ze dvou menších



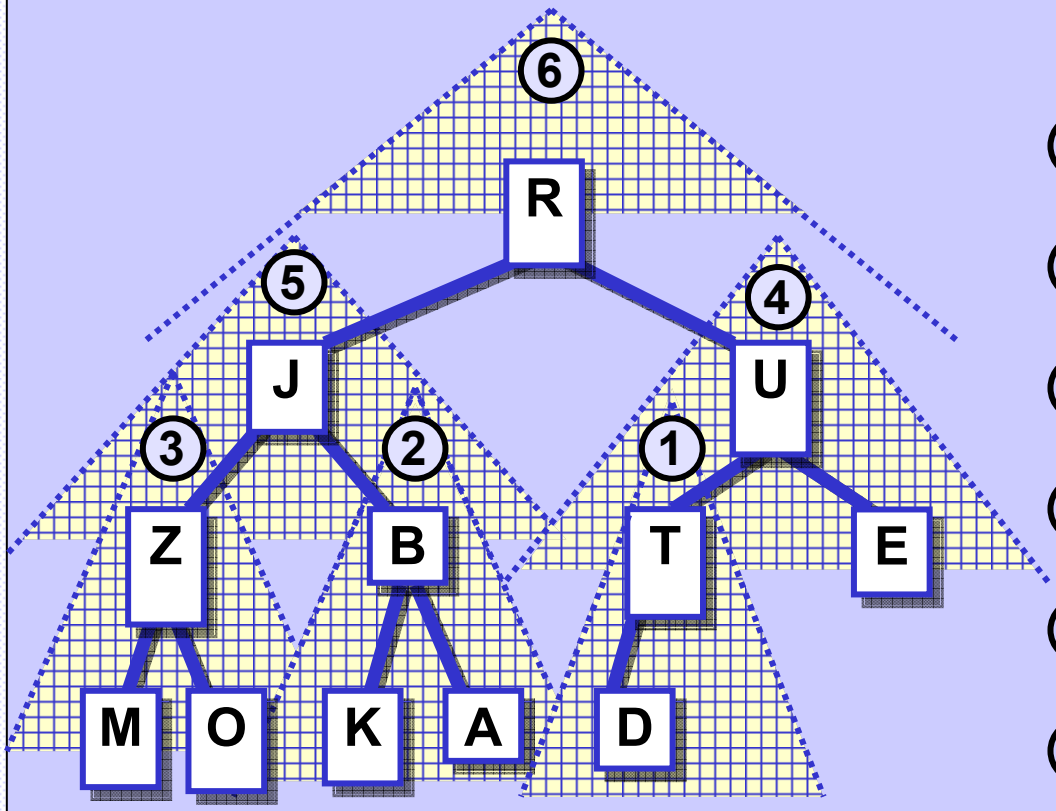
Z > A nebo Z > B

⇒ prohoď: Z ↔ min(A,B)

Vytvoř jednu haldu ze dvou menších



Vytvoř haldu



① Vytvoř haldu zde…

② … vytvoř haldu zde...

③ … vytvoř haldu zde...

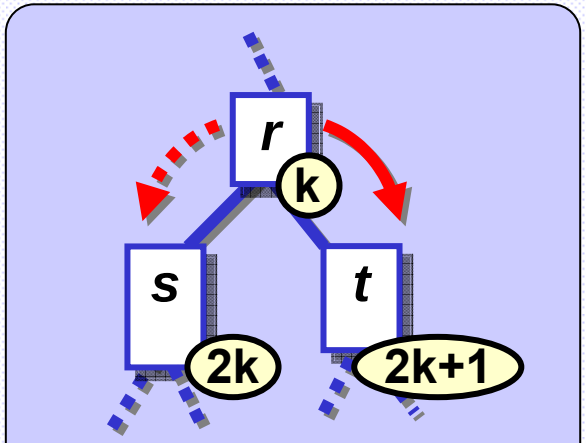
④ … vytvoř haldu zde...

⑤ … vytvoř haldu zde...

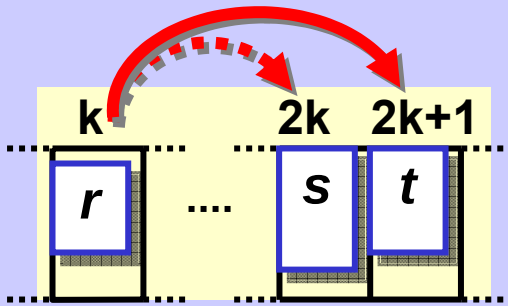
⑥ … vytvoř haldu zde, celá halda je hotova.

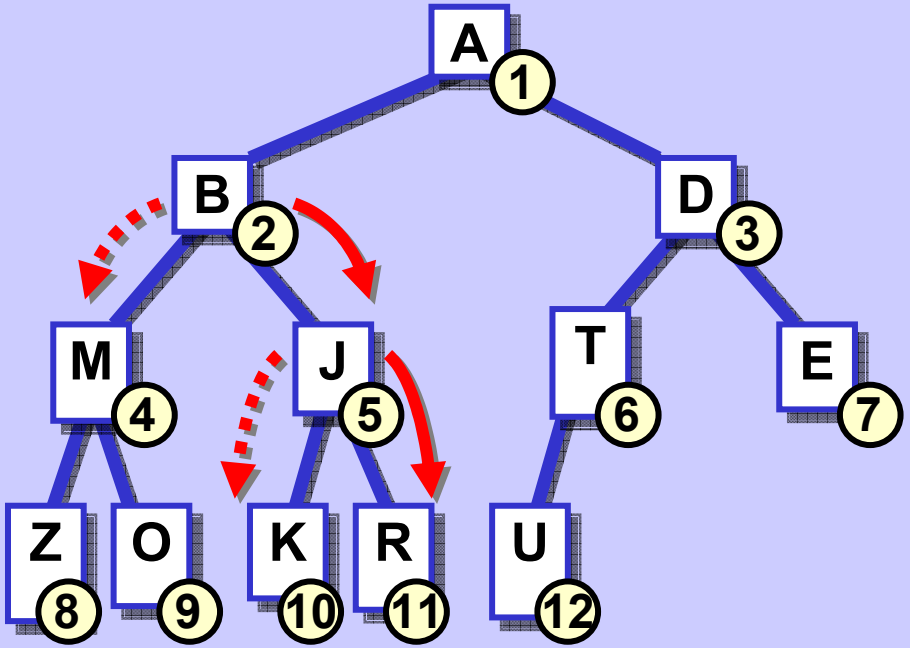
Halda v poli

Halda uložená v poli

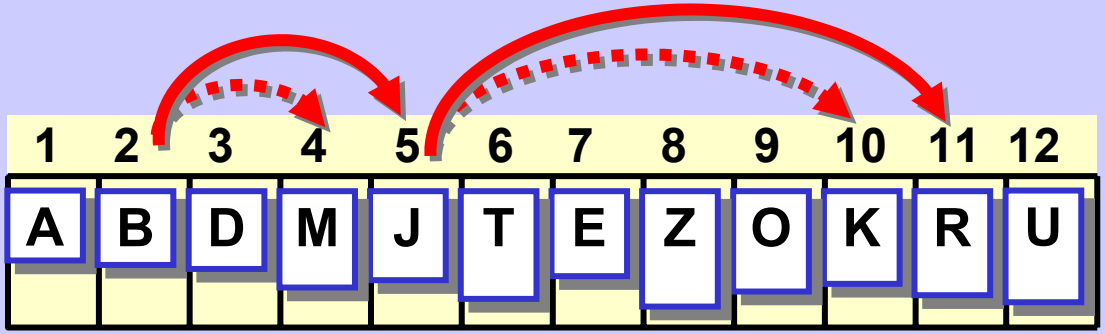


následníci



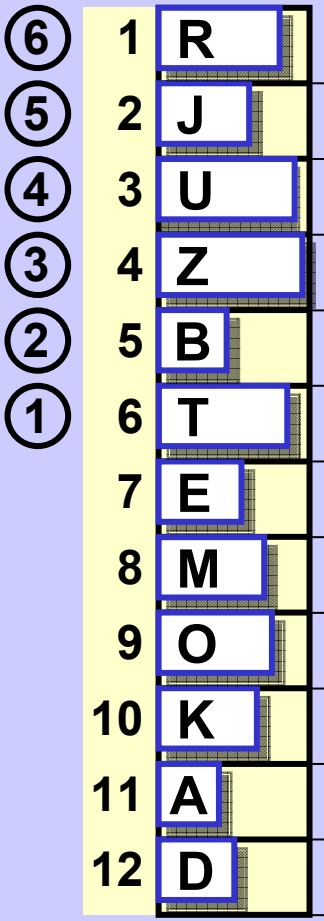


následnící

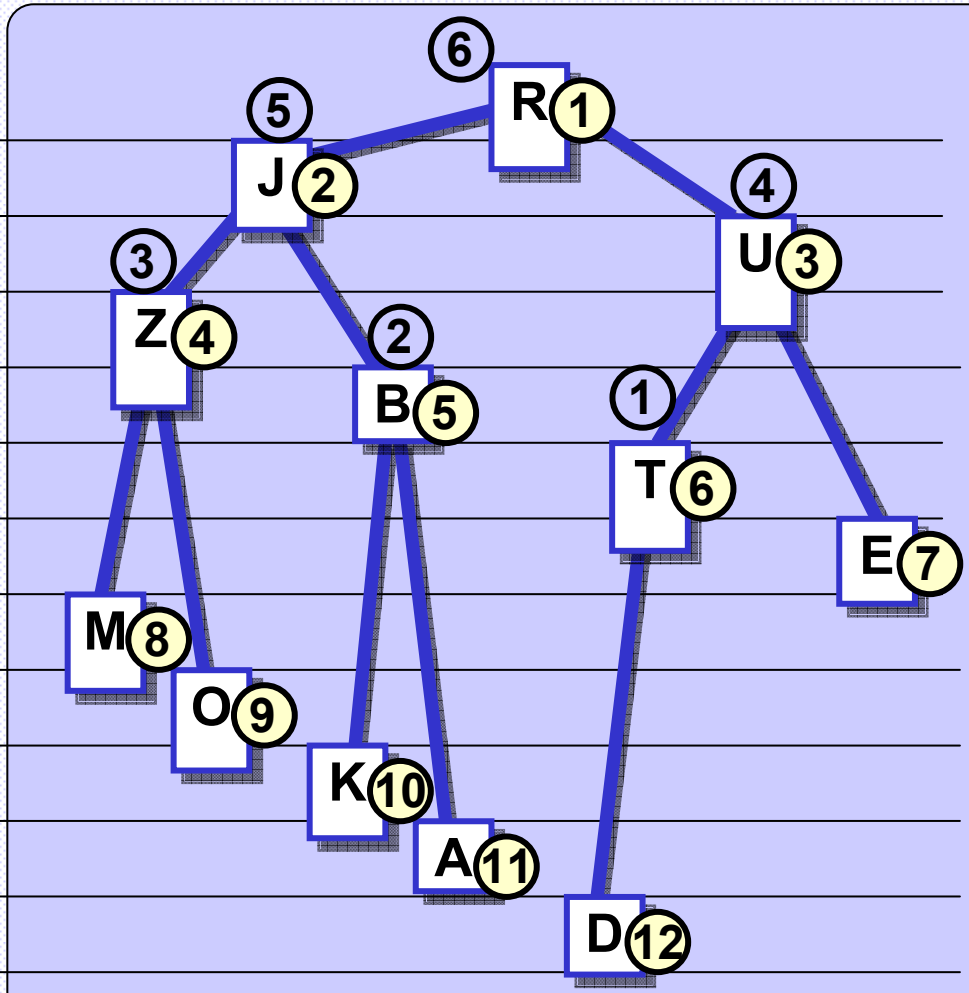


Tvorba haldy v poli

Neseřazeno

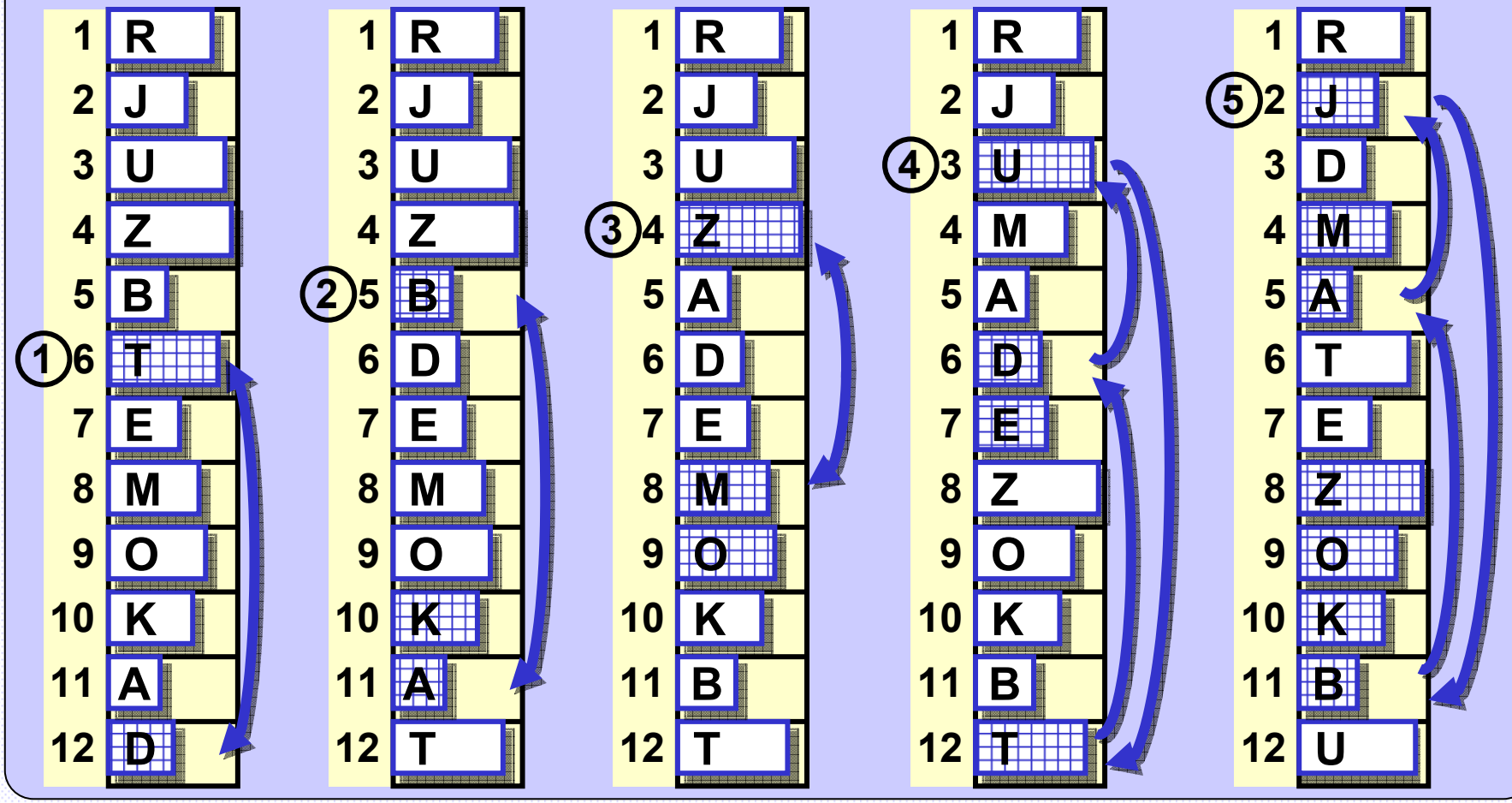


Není halda



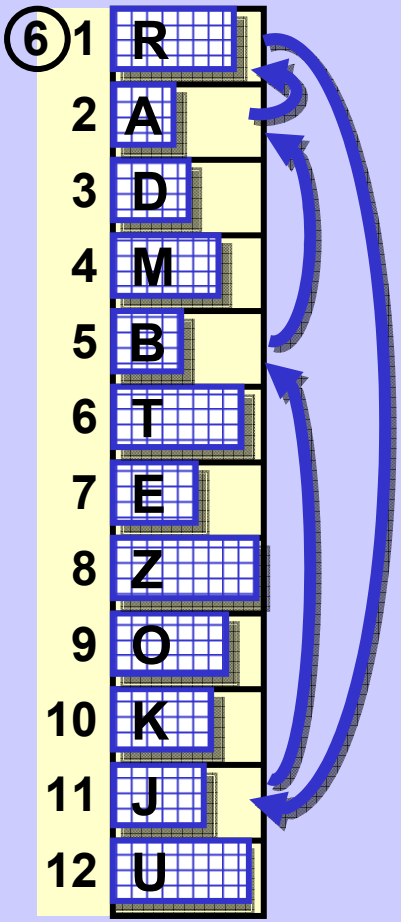
Tvorba haldy v poli

tvorba haldy:

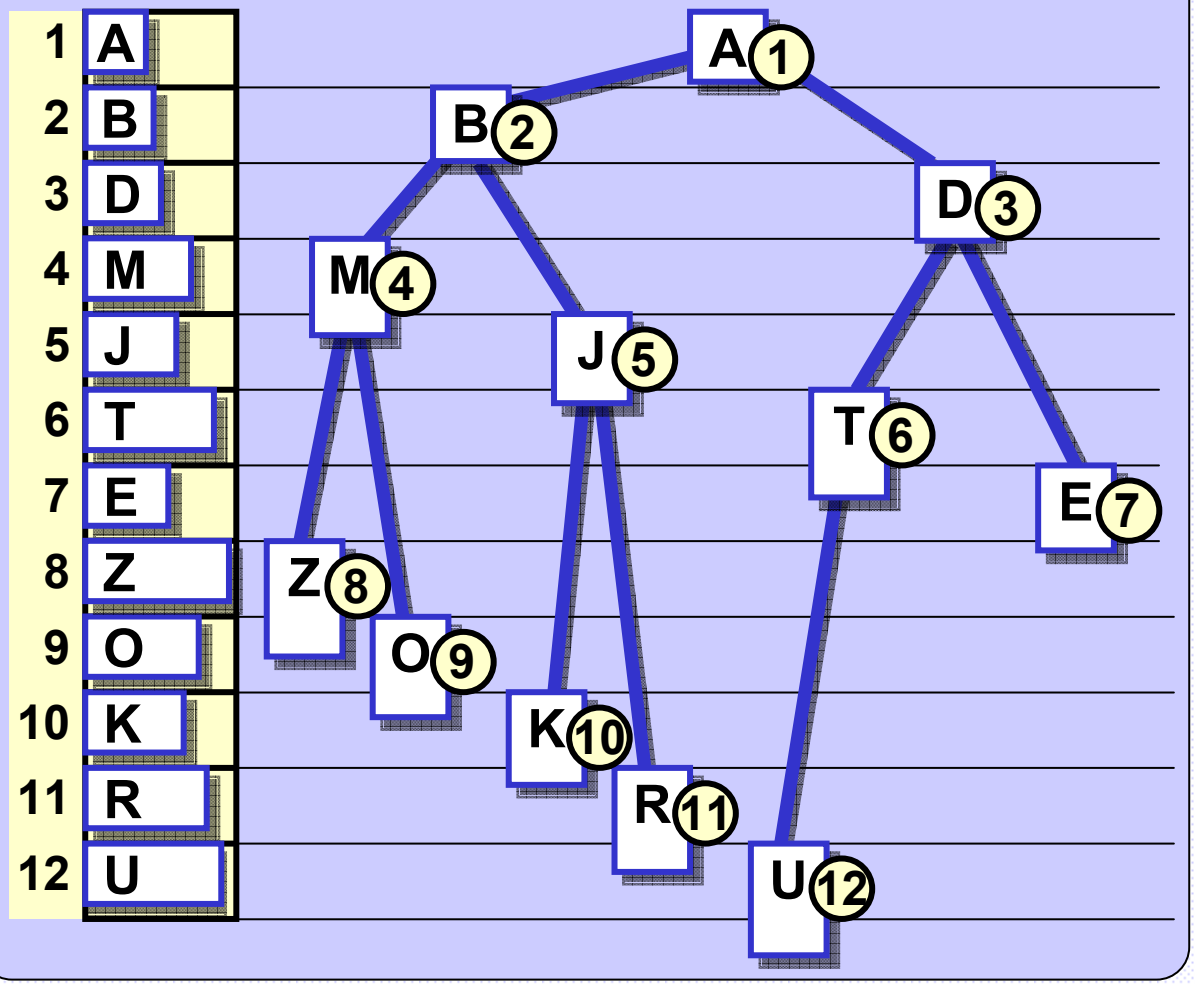


Tvorba haldy v poli

Tvorba haldy

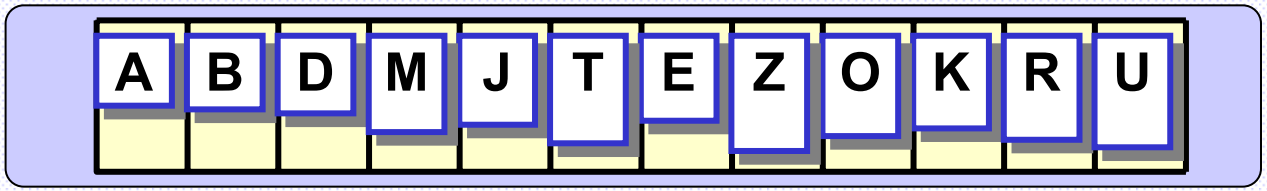


Halda

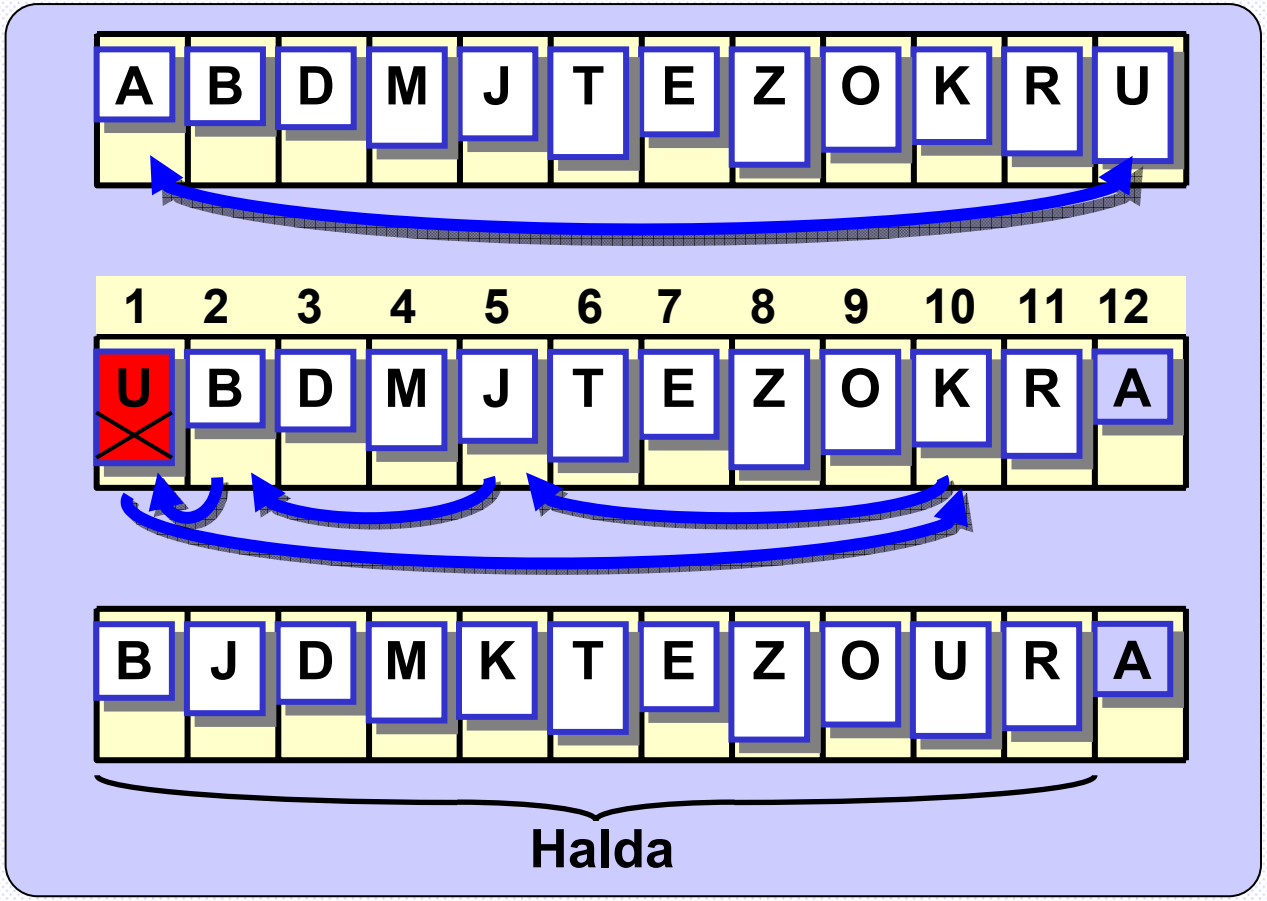


Heap sort

Halda

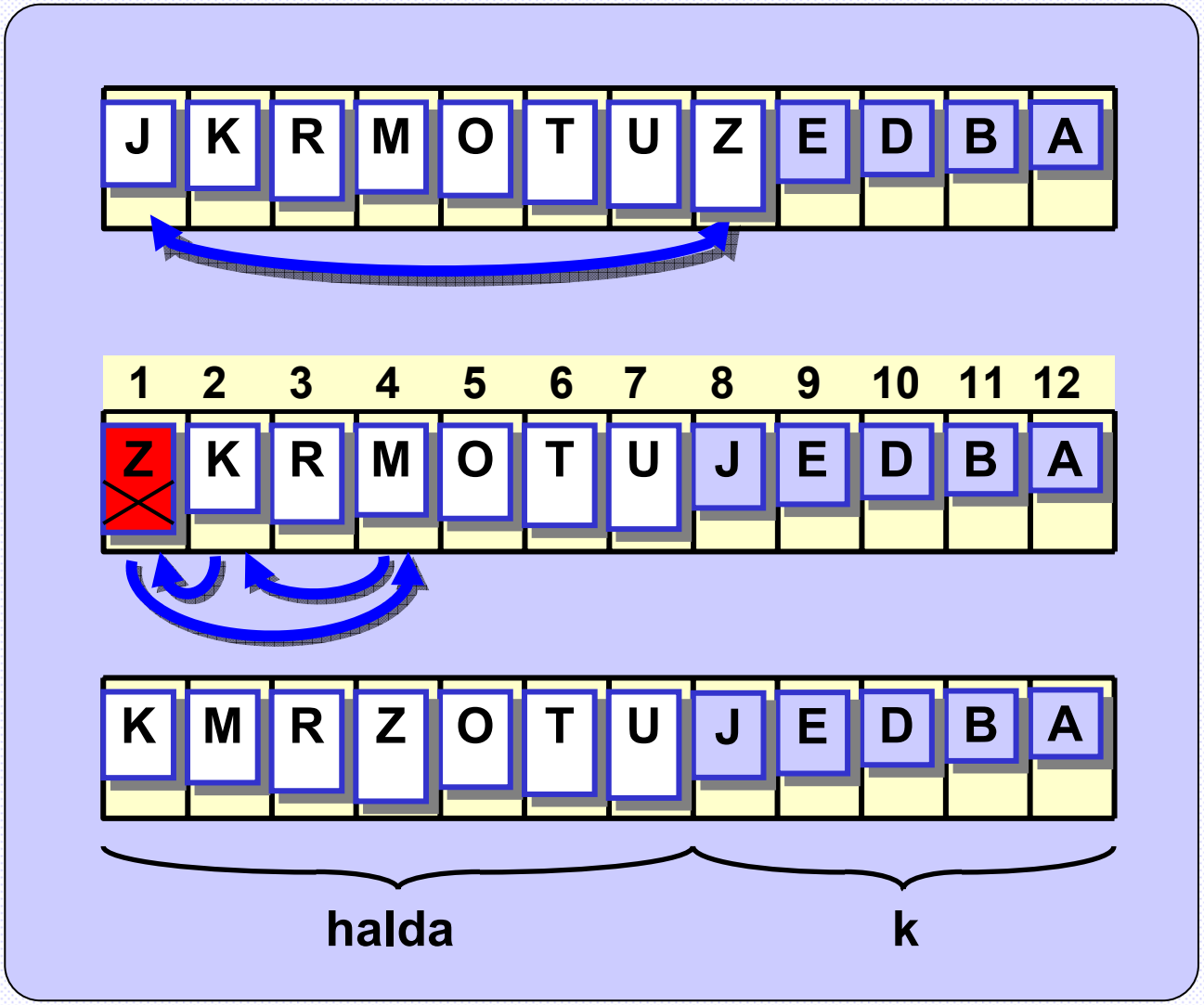


Krok 1



Heap sort

Krok k



Heap sort

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

void heapSort(Item a[], int n) {
  int i, j,
                              // create a heap
  for (i = n/2; i > 0; i--)
    repairTop(array, i, n);

                              // sort
  for (i = n; i > 1; i--) {
    swap(a, 1, i);
    repairTop(array, 1, i-1);
  }
}
```

Heap sort

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

  Item topVal = a[top];

                          // try to find a successor < topVal
  if ((j < bott) && (a[j] > a[j+1]))  j++;

                         // while (successors < topVal)
                         //         move successors up
  while ((j <= bott) && (topVal > a[j])) {
    a[i] = a[j];
    i = j;  j = j*2; // skip to next successor
    if ((j < bott) && (a[j] > a[j+1])) j++;
  }
  a[i] = topVal;    // put the topVal
}
```

Heap sort

repairTop operace nejhorší případ ... $\log_2(n)$ (n=velikost haldy)

vytvoř haldu ... n/2 repairTop operací

$\log_2(n/2) + \log_2(n/2+1) + ... + \log_2(n) \leq (n/2)(\log_2(n)) = \underline{O(n \cdot \log_2(n))}$

seřaď haldy ... n-1 repairTop operací, nejhorší případ:

$\log_2(n) + \log_2(n-1) + ... + 1 \leq n \cdot \log_2(n) = O(n \cdot \log_2(n))$

ale i nejlepší případ = $\underline{\Theta(n \cdot \log_2(n))}$

celkem ... vytvoř haldu + seřaď haldu = $\underline{\Theta(n \cdot \log_2(n))}$

Asymptotická složitost Heap sortu je $\Theta(n \cdot \log_2(n))$

Heap sort není stabilní

Řazení

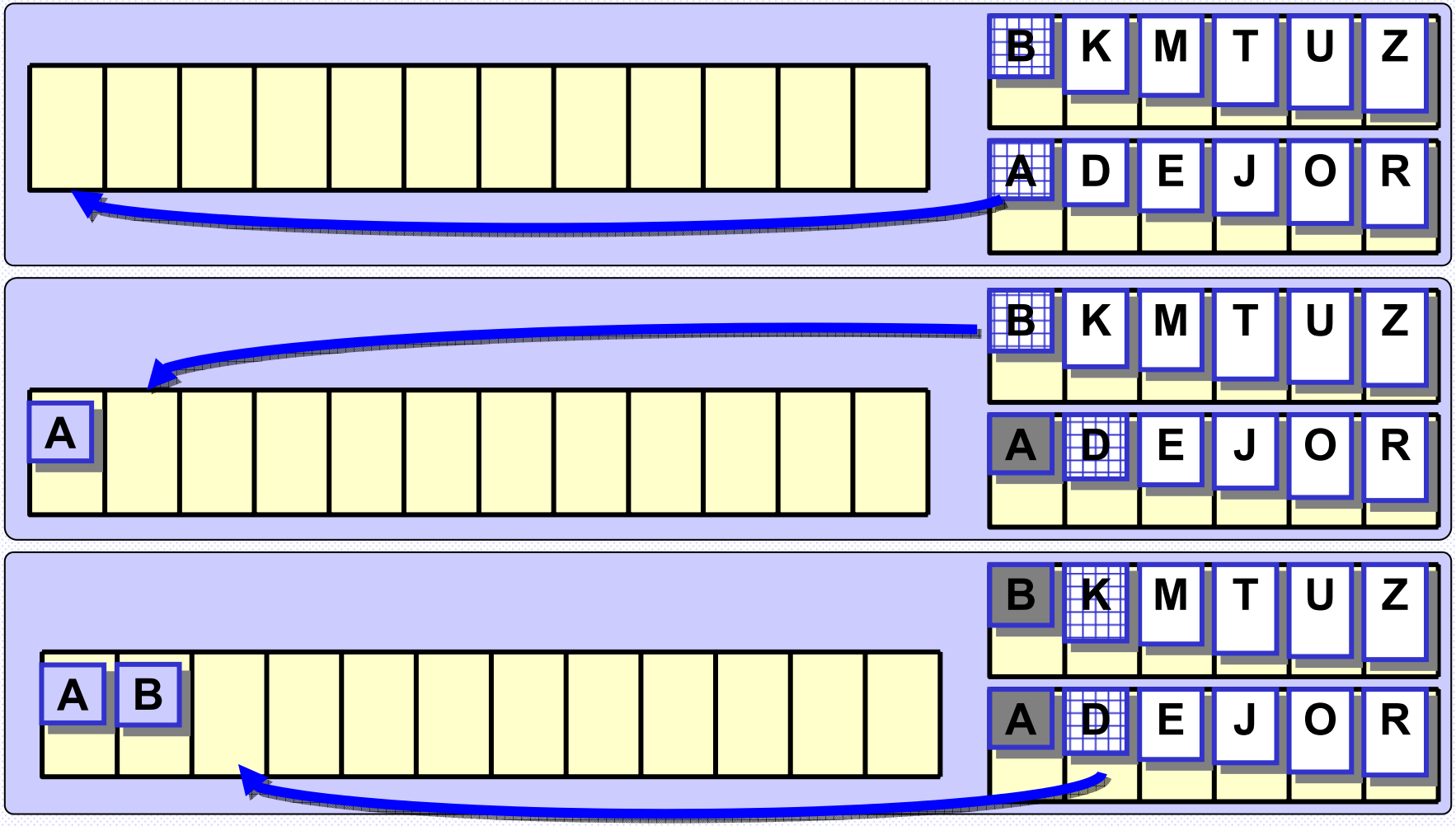
Merge Sort

Řazení sléváním

Merge sort

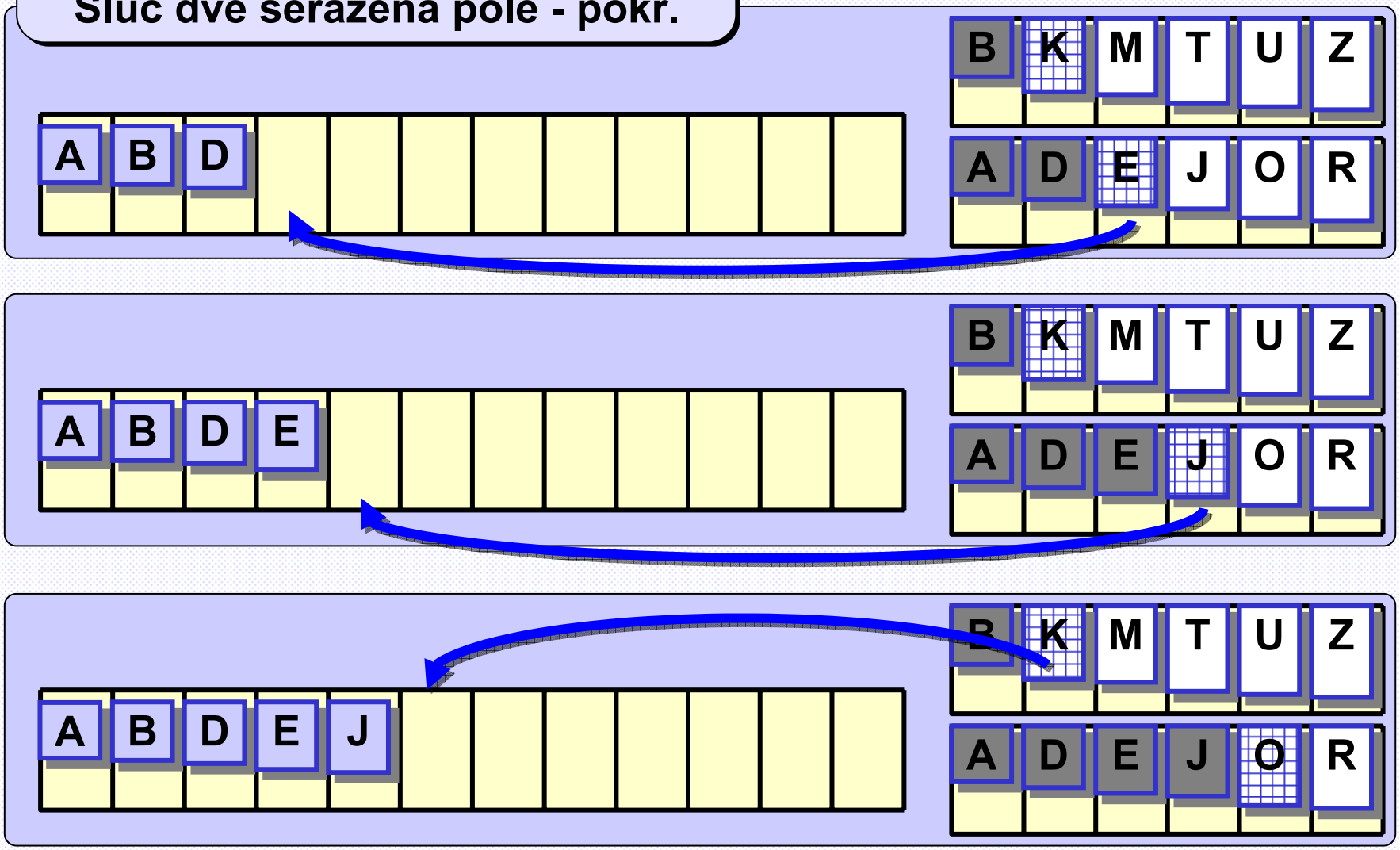
Sluč (slij?) dvě seřazená pole

porovnávané prvky



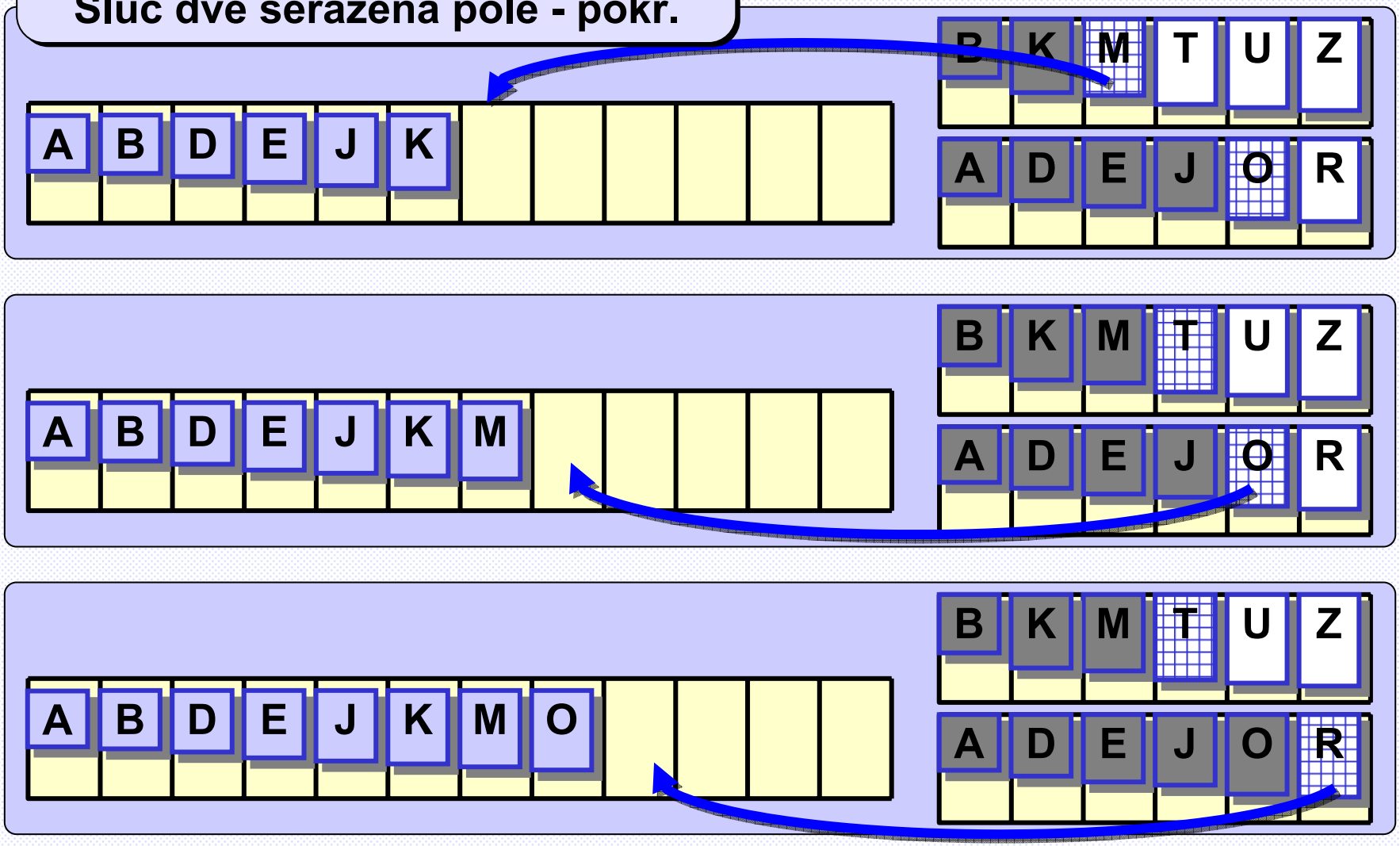
Merge sort

Sluč dvě seřazená pole - pokr.



Merge sort

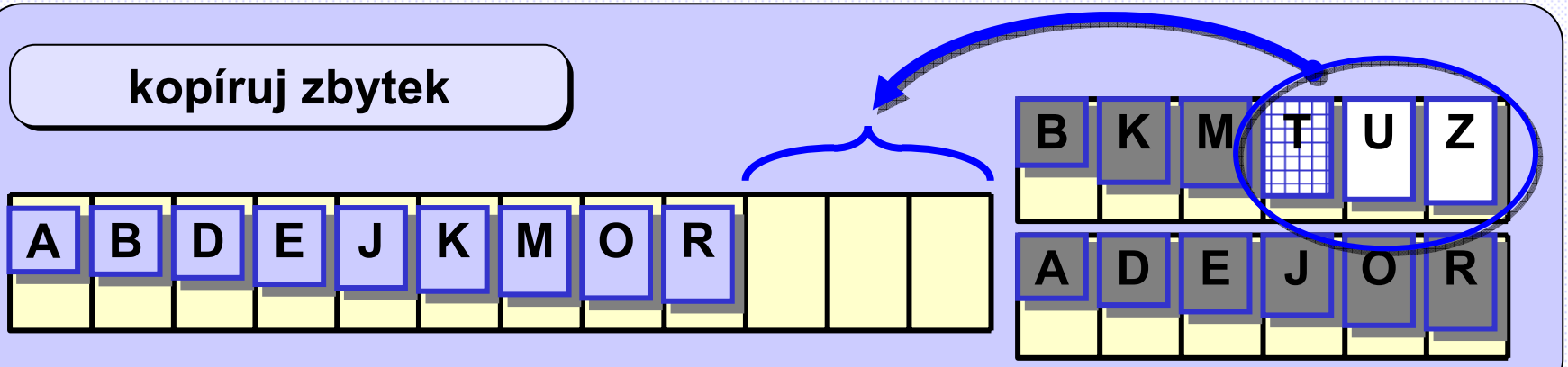
Sluč dvě seřazená pole - pokr.



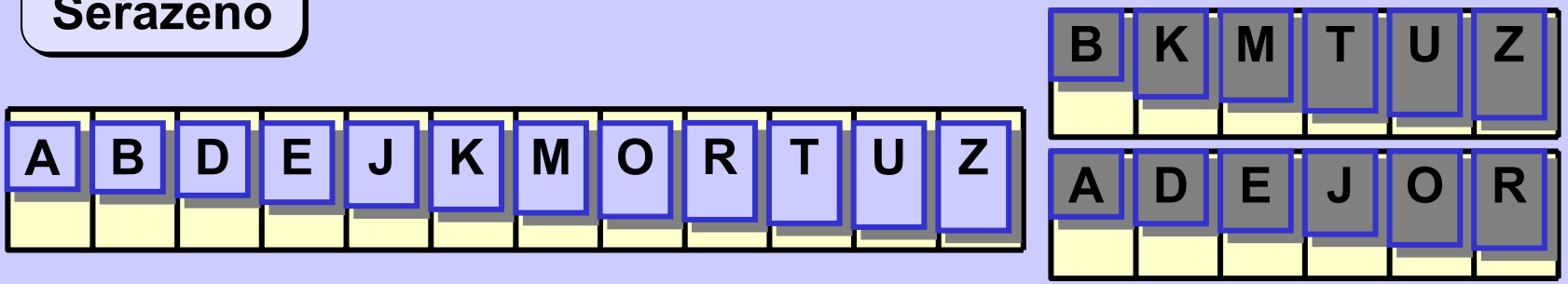
Merge sort

Sluč dvě seřazená pole - pokr.

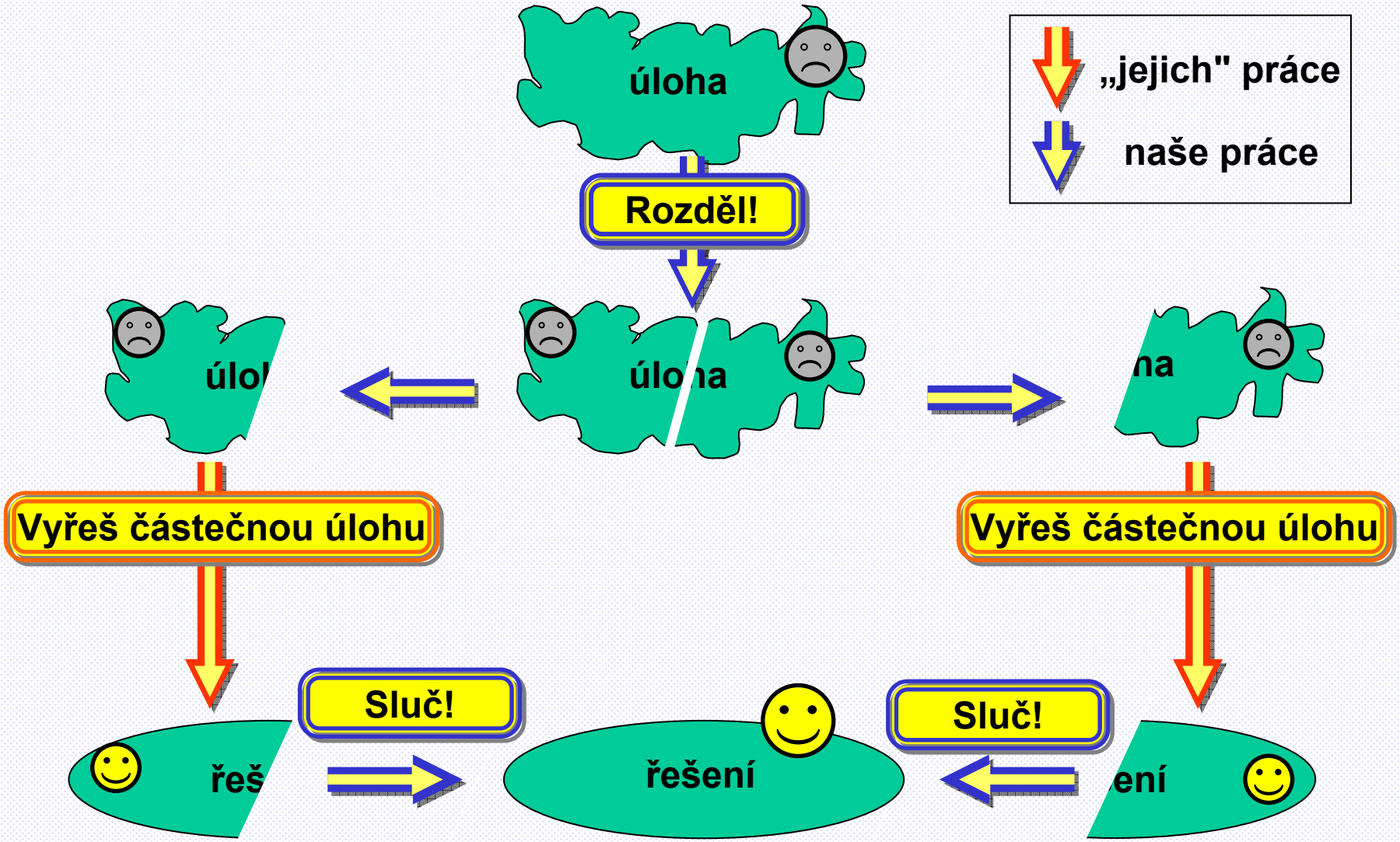
kopíruj zbytek



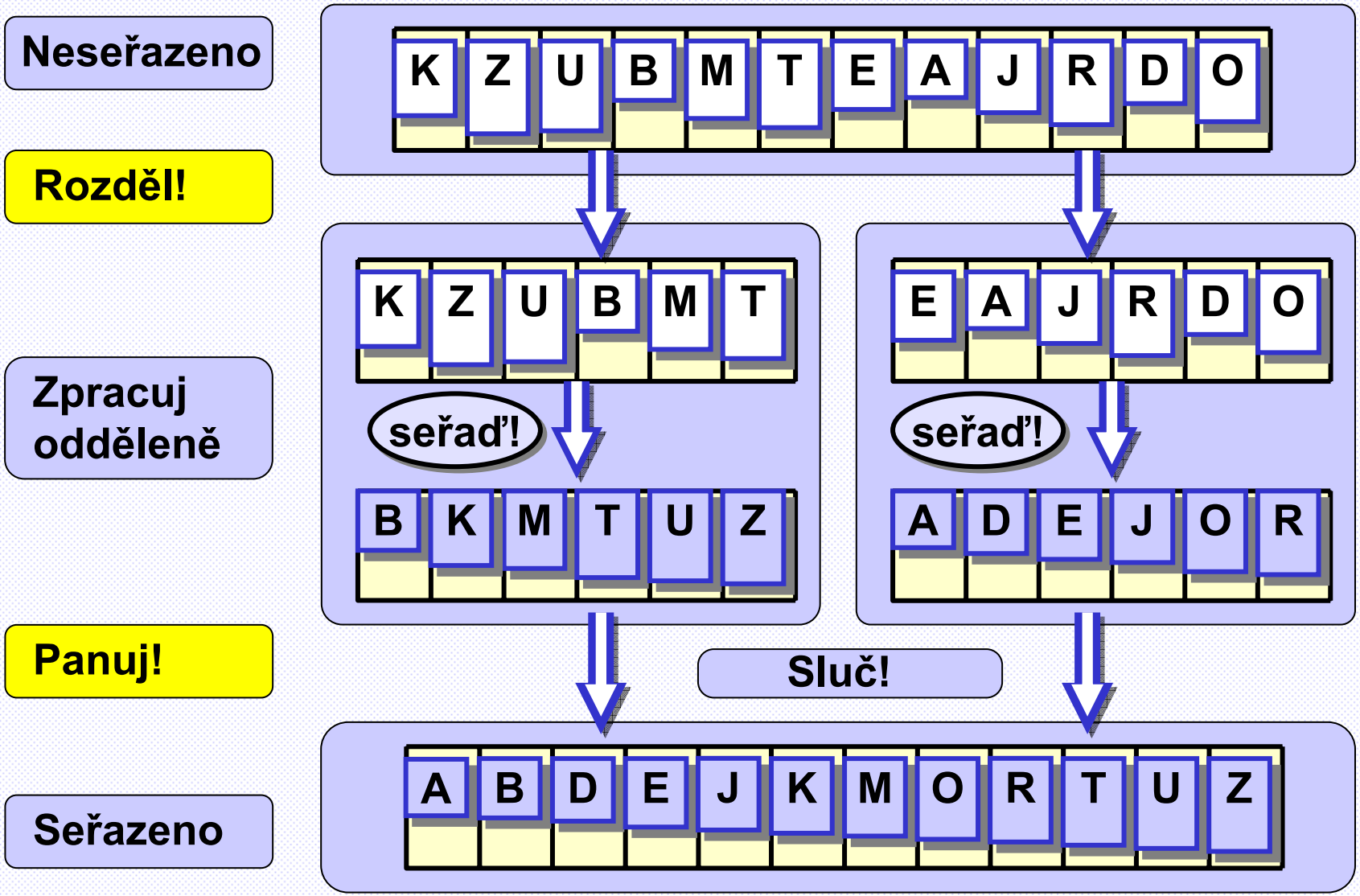
Seřazeno



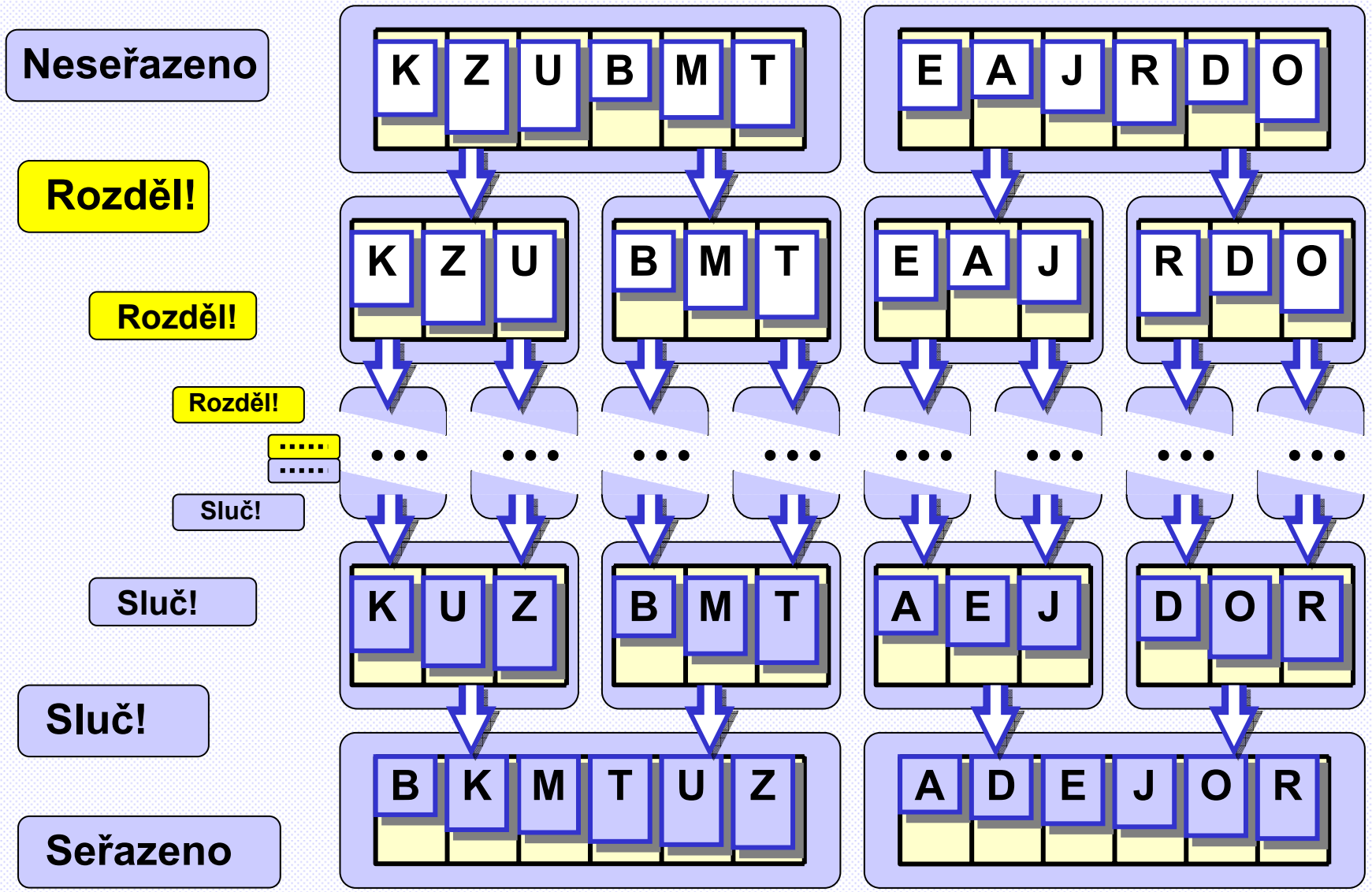
Rozděl a Panuj! Divide & Conquer! Divide et Impera!



Merge sort


Neseřazeno
Rozděl!
Zpracuj odděleně
Panuj!
Seřazeno
K Z U B M T E A J R D O
K Z U B M T
E A J R D O
seřaď!
seřaď!
B K M T U Z
A D E J O R
Sluč!
A B D E J K M O R T U Z

Merge sort


Neseřazeno
Rozděl!
Rozděl!
Rozděl!
Sluč!
Sluč!
Sluč!
Seřazeno
K Z U B M T
E A J R D O
K Z U
B M T
E A J
R D O
K U Z
B M T
A E J
D O R
B K M T U Z
A D E J O R

Merge sort

```
void mergeSort (int a[], int aux[],
                                int low, int high)  {
  int half = (low+high)/2;
  int i;
  if (low >= high) return;          // too small!
                                // sort
  mergeSort(a, aux, low, half);    // left half
  mergeSort(a, aux, half+1, high); // right half
  merge(a, aux, low, high);        // merge halves

                                // put result back to a
  for (i = low; i <= high; i++)  a[i] = aux[i];

 // optimization idea:
  /* swapArray(a, aux) */ // better to swap
                                // references to a & aux!
}
```

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

Asymptotická sožitost

Rozděl! ..... $\log_2(n)$ krát $\Rightarrow$

$\Rightarrow$ Sluč! ..... $\log_2(n)$ krát

Rozděl! ..... $\Theta(1)$ operací

Sluč! ..... $\Theta(n)$ operací

Celkem ..... $\Theta(n) \cdot \Theta(\log_2(n)) = \Theta(n \cdot \log_2(n))$ operací

Asymptotická složitost Merge sortu je $\Theta(n \cdot \log_2(n))$

Merge sort

Stabilita

Rozděl! ..... Nepohybuje prvky

Sluč! ..... “ if (in[i1] <= in[i2]) { out[j] = in[i1]; …”

Zařaď nejprve levý prvek
když slučuješ stejné hodnoty

MergeSort je stabilní

Řazení

Radix Sort

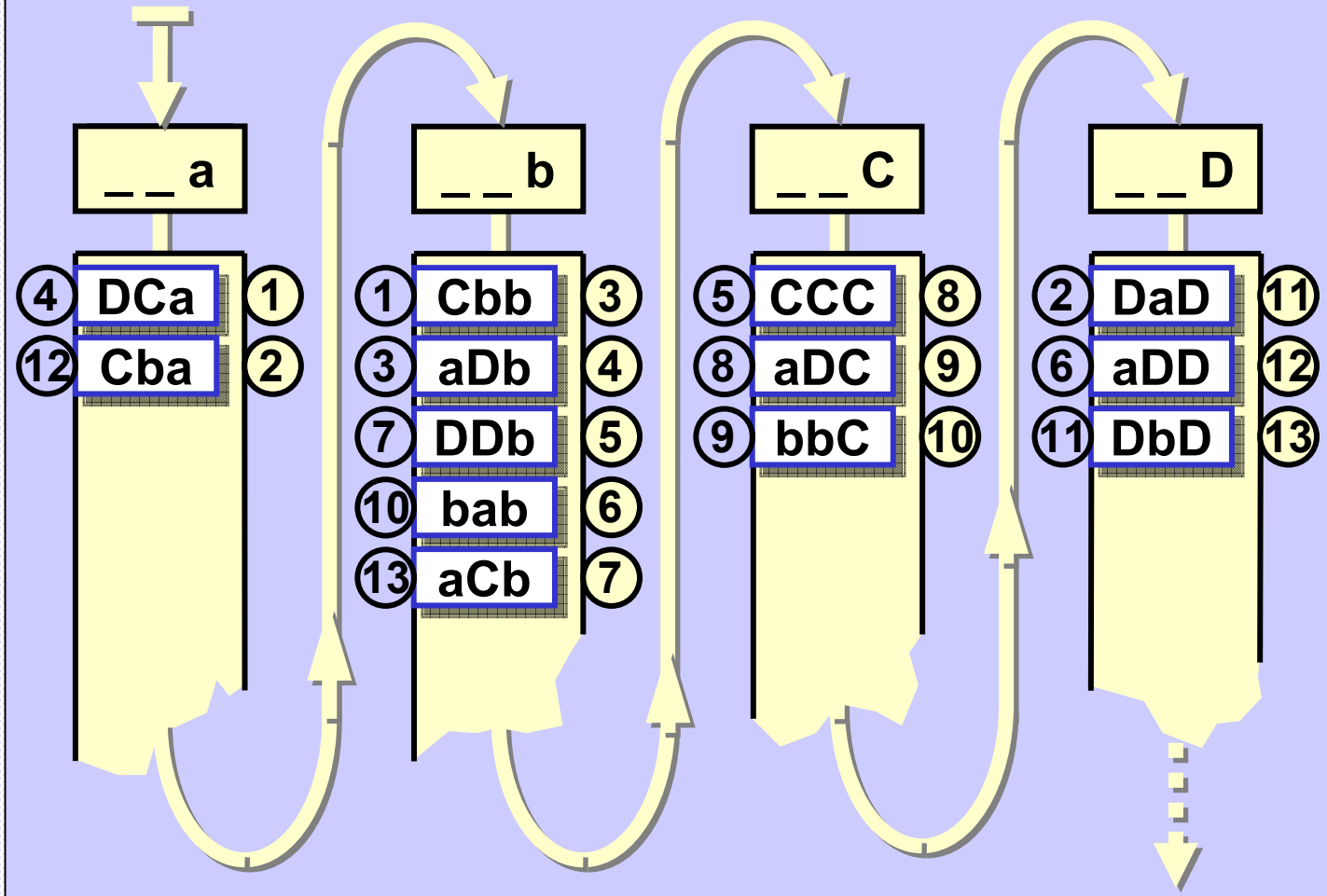
Přihrádkové řazení

Radix sort

Neseřazeno

1 Cbb
2 DaD
3 aDb
4 DCa
5 CCC
6 aDD
7 DDb
8 aDC
9 bbC
10 bab
11 DbD
12 Cba
13 aCb

řaď podle 3. znaku

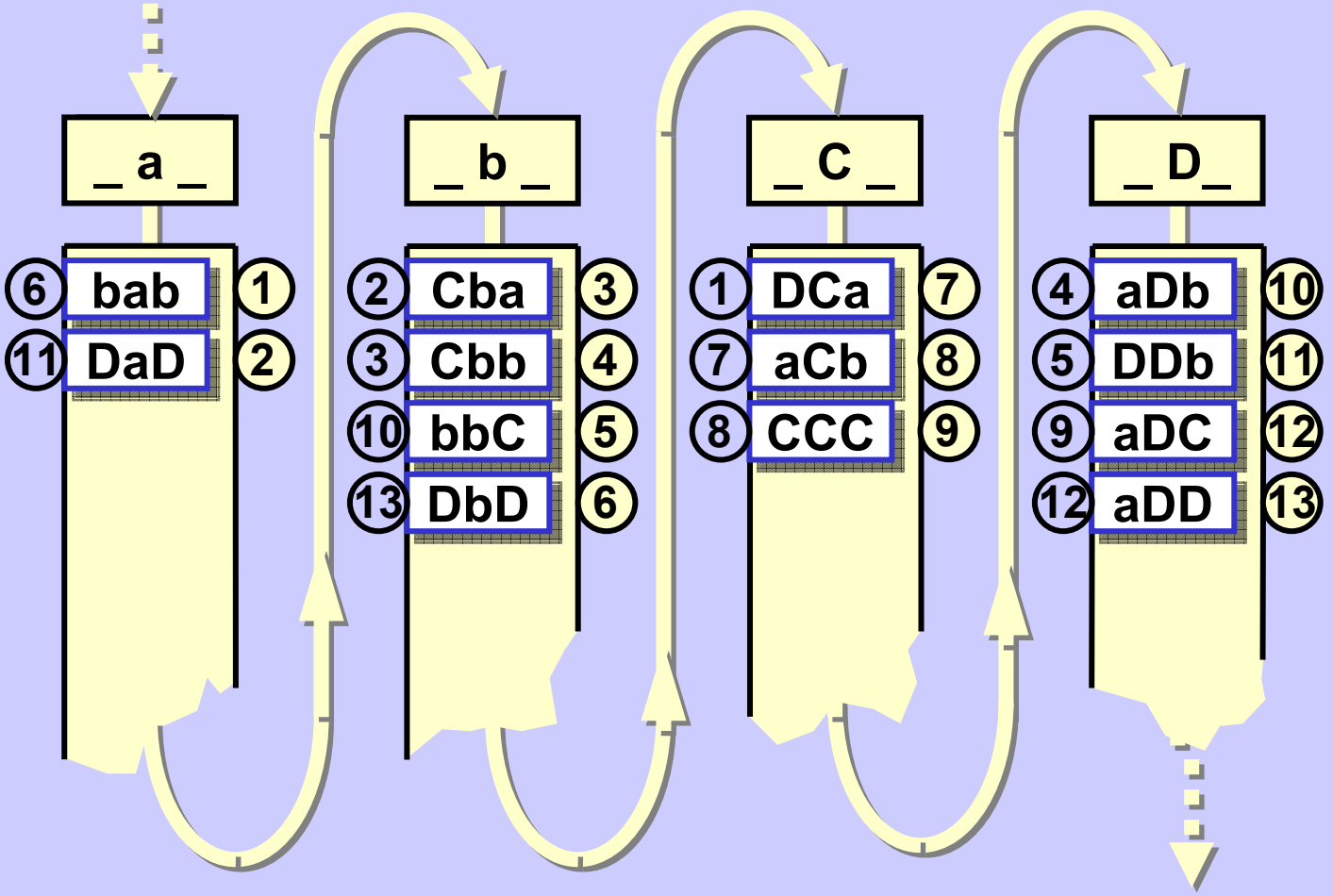


Radix sort

Seřazeno od 3. znaku

1 DCa
2 Cba
3 Cbb
4 aDb
5 DDb
6 bab
7 aCb
8 CCC
9 aDC
10 bbC
11 DaD
12 aDD
13 DbD

řaď podle 2. znaku

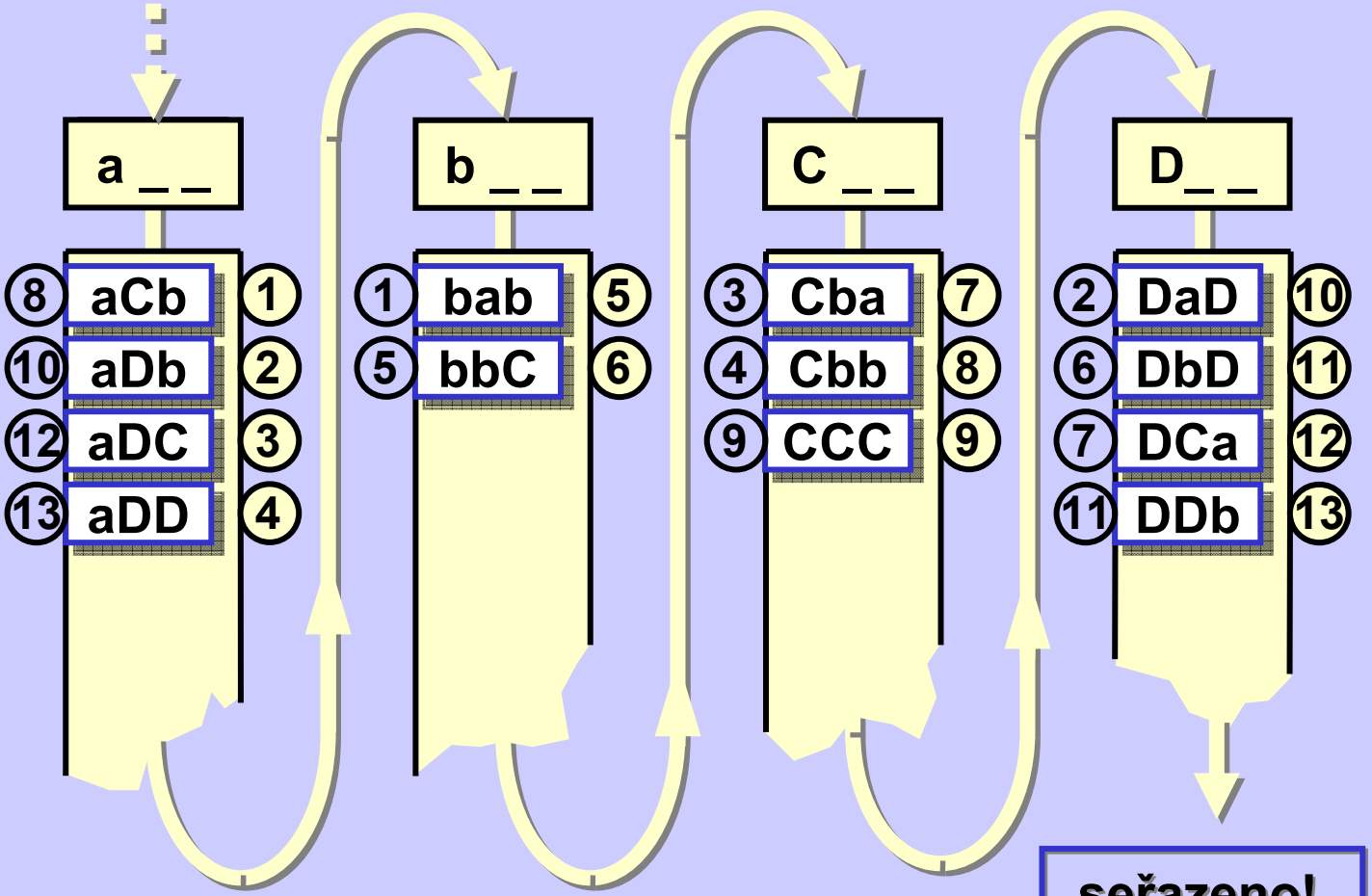


Radix sort

Seřazeno od 2. znaku

1. bab
2. DaD
3. Cba
4. Cbb
5. bbC
6. DbD
7. DCa
8. aCb
9. CCC
10. aDb
11. DDb
12. aDC
13. aDD

řaď podle 1. znaku

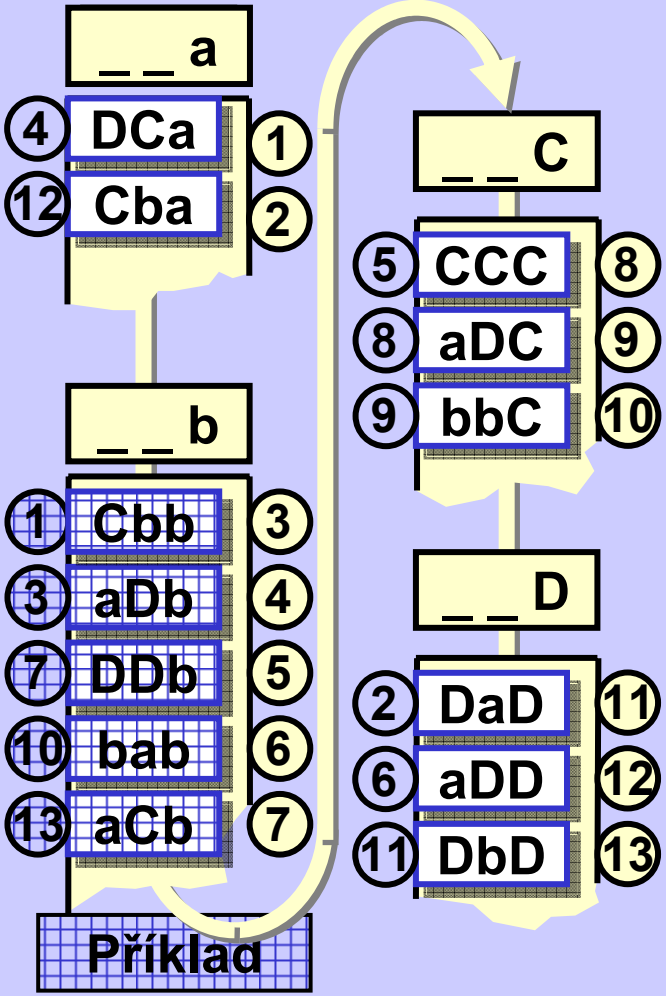


Implementace Radix sortu

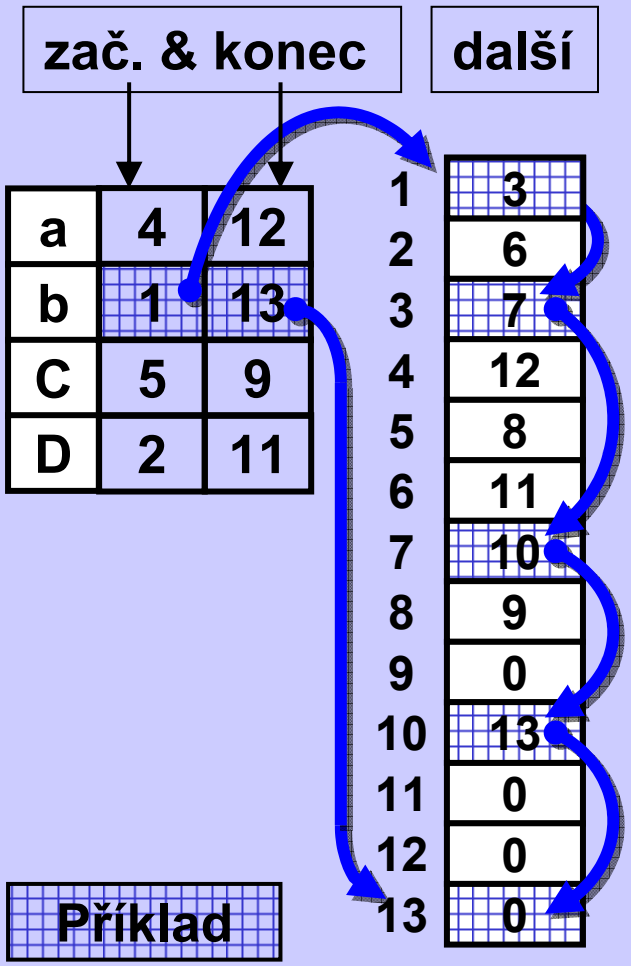
Neseřazeno

1	Cbb
2	DaD
3	aDb
4	DCa
5	CCC
6	aDD
7	DDb
8	aDC
9	bbC
10	bab
11	DbD
12	Cba
13	aCb

seřazeno dle 3. znaku



pomocná pole indexů



Shrnutí

d znaků ..... d cyklů

cyklus ..... $\Theta(n)$ operací

---

celkem ..... $\Theta(d \cdot n)$ operací

$d << n \Rightarrow$ ..... $\Theta(n)$ operací

Přihrádkové řazení nemění pořadí stejných hodnot

Asymptotická složitost Radix sortu je $\Theta(n)$

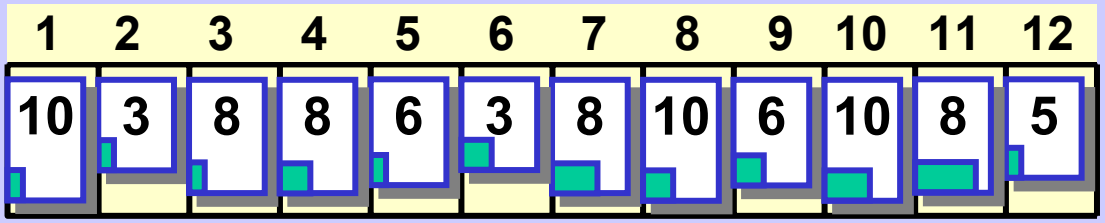
Je to stabilní řazení

Řazení

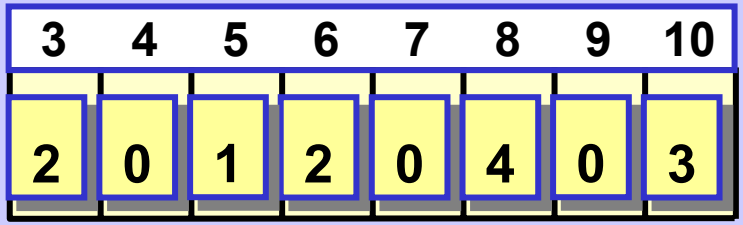
Counting sort

Counting sort

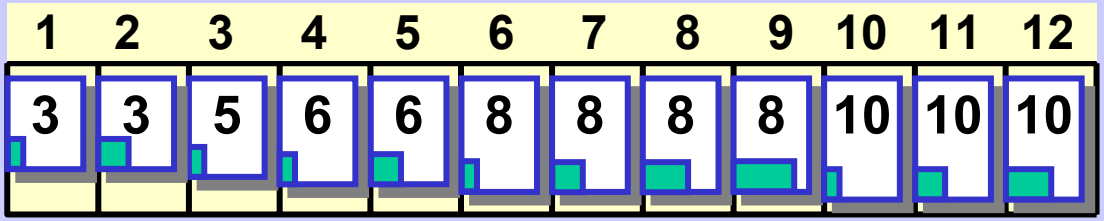
vstup
vstup.length == N



cetnost
cetnost.length == k
k = max(vstup) - min(vstup) + 1



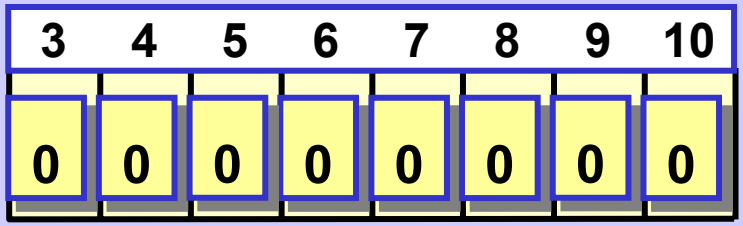
vystup
vstup.length == N



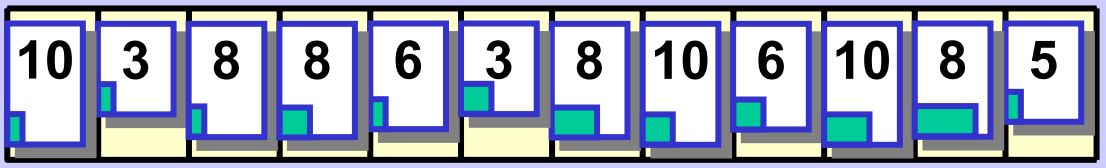
Counting sort

Krok 1

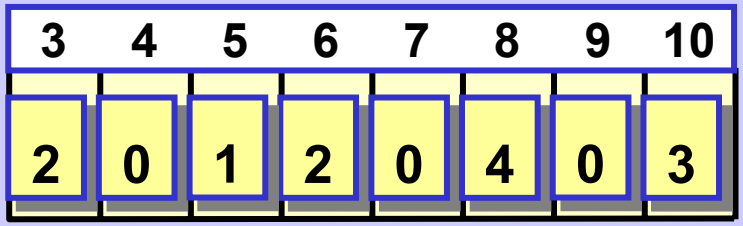
vynulování pole četností



jeden průchod vstupním polem



naplnění pole četností



Counting sort

Krok 2

jeden průchod

3	4	5	6	7	8	9	10
2	0	1	2	0	4	0	3

úprava pole četností

```
for(i = dMez+1; i <= hMez; i++)
    cetnost[i] += cetnost[i-1]
```

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

Prvek **cetnost[j]** obsahuje pozici posledního prvku s hodnotou **j** v budoucím výstupním poli.

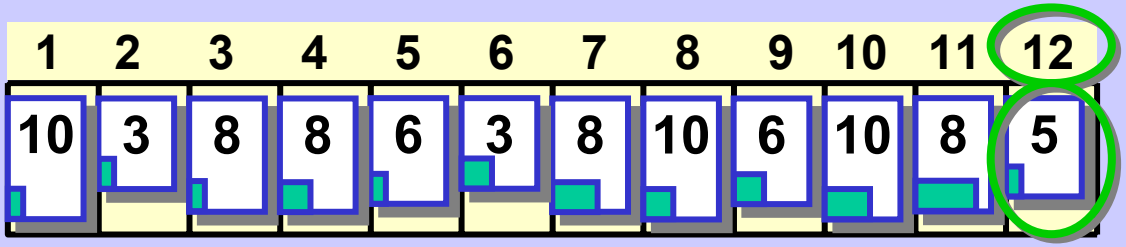
1	2	3	4	5	6	7	8	9	10	11	12
3	3	5	6	6	8	8	8	8	10	10	10

Counting sort

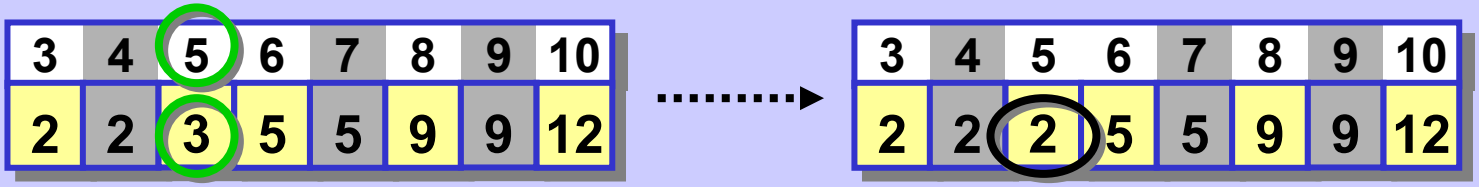
Krok 3

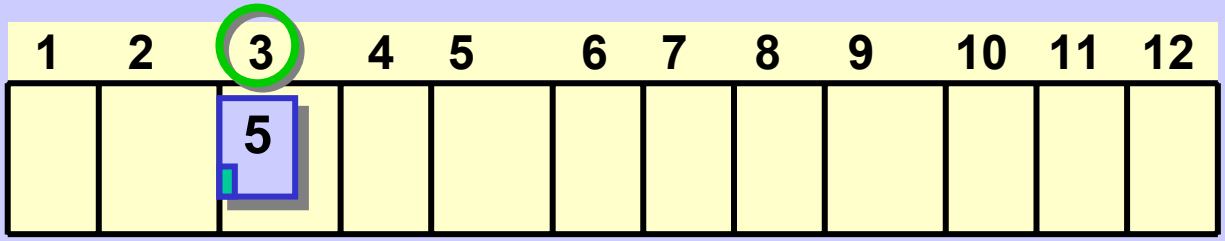
i == N

```
for(i = N; i > 0; i--) {
  vystup[cetnost[vstup[i]] = vstup[i];
  cetnost[vstup[i]]--;
}
```



i == 12



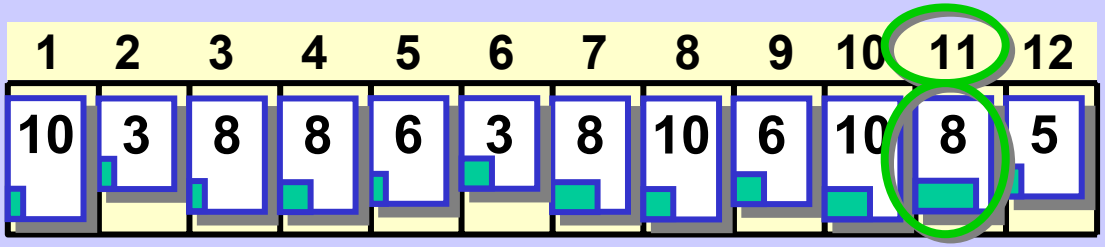


Counting sort

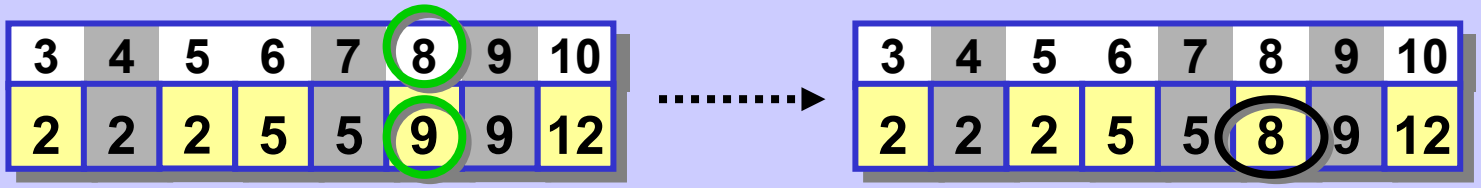
Krok 3

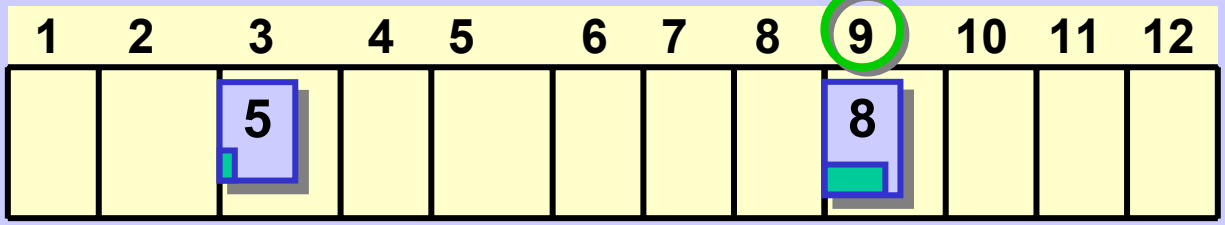
i == N-1

```
for(i = N; i > 0; i--) {
  vystup[cetnost[vstup[i]] = vstup[i];
  cetnost[vstup[i]]--;
}
```



i == 11



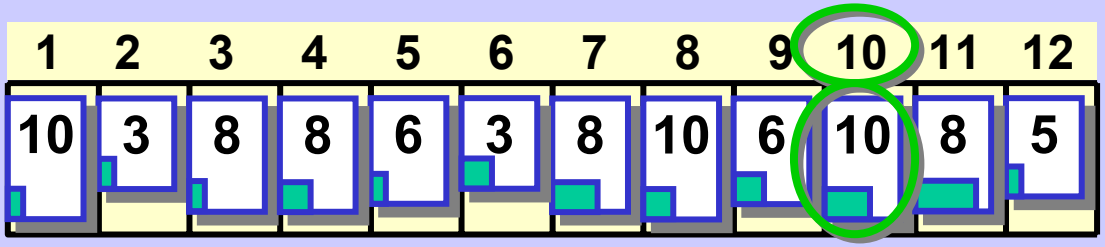


Counting sort

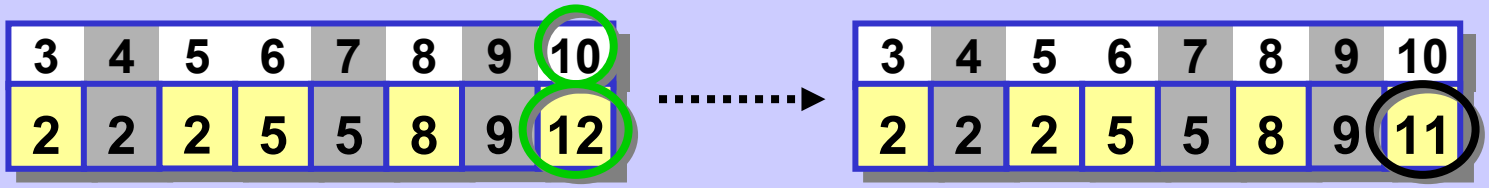
Krok 3

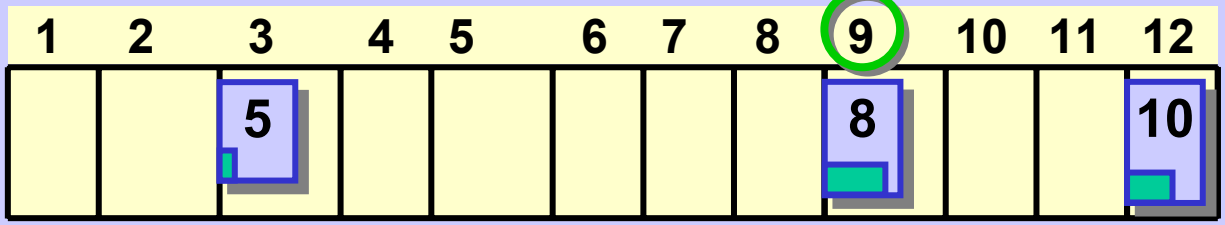
i == N-2

```
for(i = N; i > 0; i--) {
  vystup[cetnost[vstup[i]] = vstup[i];
  cetnost[vstup[i]]--;
}
```



i == 10





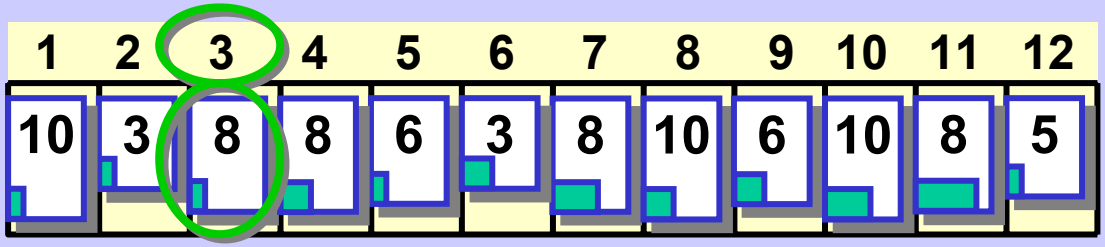
atd...

Counting sort

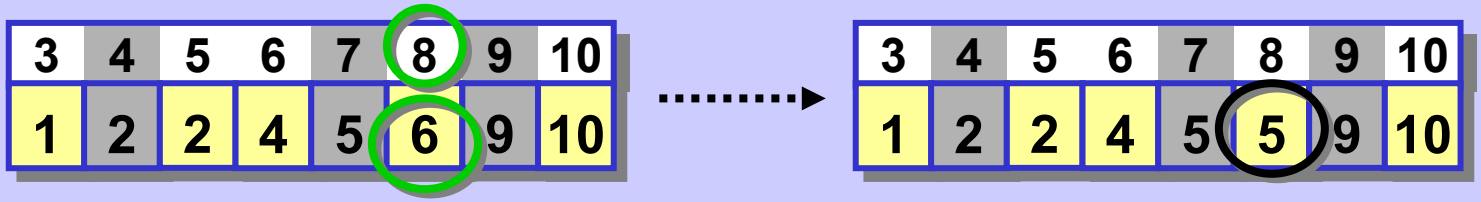
Krok 3

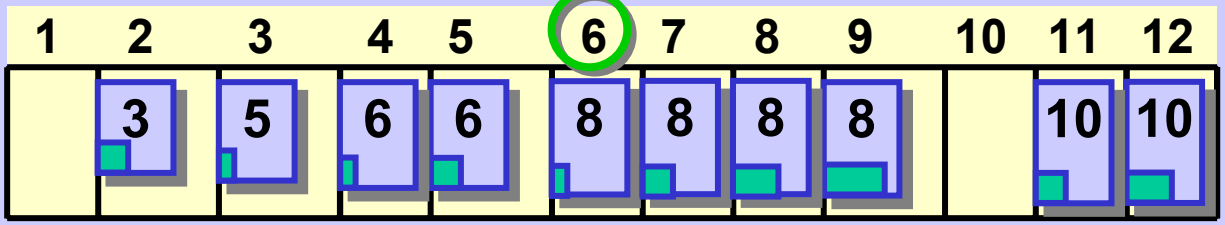
i == 3

```
for(i = N; i > 0; i--) {
  vystup[cetnost[vstup[i]] = vstup[i];
  cetnost[vstup[i]]--;
}
```



i == 3





atd...

Counting sort

Shrnutí

Krok 1 ... $\Theta(N)$ operací

Krok 2 ... $\Theta(k)$ operací

Krok 3 ... $\Theta(N)$ operací

Celkem ... $\Theta(k + N)$ operací

Pokud $k \leq N$,
je asymptotická složitost Counting sortu $\Theta(N)$

Counting sort je stabilní

N je velikost vstupního pole,
k je velikost intervalu, v němž všechny vstupní hodnoty leží.

Různé algoritmy mají různou složitost

Algoritmus a program není totéž