Counting Sort a Radix Sort

Bc. Katarína Olejková

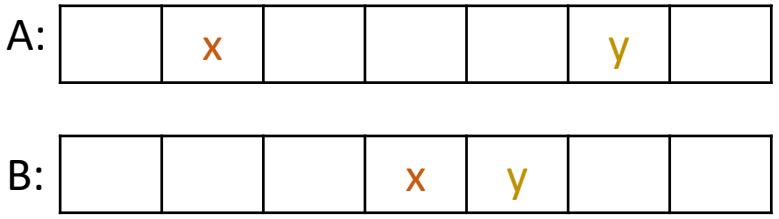
KATEDRA INFORMATIKY

UNIVERZITA PALACKÉHO V OLOMOUCI

Stabilné vs Nestabilné triedenie

Stabilné

- Pokiaľ x = y a x stojí pred y vo vstupnom poli, tak vo výstupnom poli musí byť tiež x pred y (x „nepredbehne“ y)



1	3	0	5	2	3	4

0	1	2	3	3	4	5

Nestabilné

- naopak

Counting Sort

- Algoritmus, ktorý **nie je** založený na porovnávaní
- Základnou myšlienkou je spočítať frekvenciu výskytu každého prvku vo vstupnom poli a použiť túto informáciu k správnemu umiestneniu prvku vo výstupnom poli
- „You count how many of each digit there is, and then you determine each digit's starting position by counting how many cells are taken up by the digits before it“
- Je efektívny keď rozsah (range) prvkov vo vstupnom poli je malý

```
Counting-Sort(A, B, k)
1.      for i ← 0 to k                          } Inicializuje frekvenciu výskytu každého prvku na 0
2.          C[i] ← 0
3.      for j ← 0 to n - 1                      } Spočíta frekvenciu výskytu každého prvku
4.          C[A[j]] ← C[A[j]] + 1
5.          // C[i] obsahuje pocet prvku v A rovnych i
6.      for i ← 1 to k                          } Akumuluje frekvencie výskytu (dostaneme štartovacie indexy)
7.          C[i] ← C[i] + C[i – 1]
8.          // C[i] obsahuje pocet prvku v A ≤ i
9.      for j ← n – 1 downto 0                  } Zaradí prvok do výstupného poľa
10.         B[C[A[j]] – 1] ← A[j]
11.         C[A[j]] ← C[A[j]] - 1
```

A – vstupné pole
B – výstupné pole
C – „counting“ pole

i, j – indexy
n – počet prvkov
k – maximálny prvok
zo vstupného poľa(range)

Counting Sort - časová zložitosť

- V najhoršom prípade Θ(n + k) - lineárna
  - n – počet prvkov, k – range

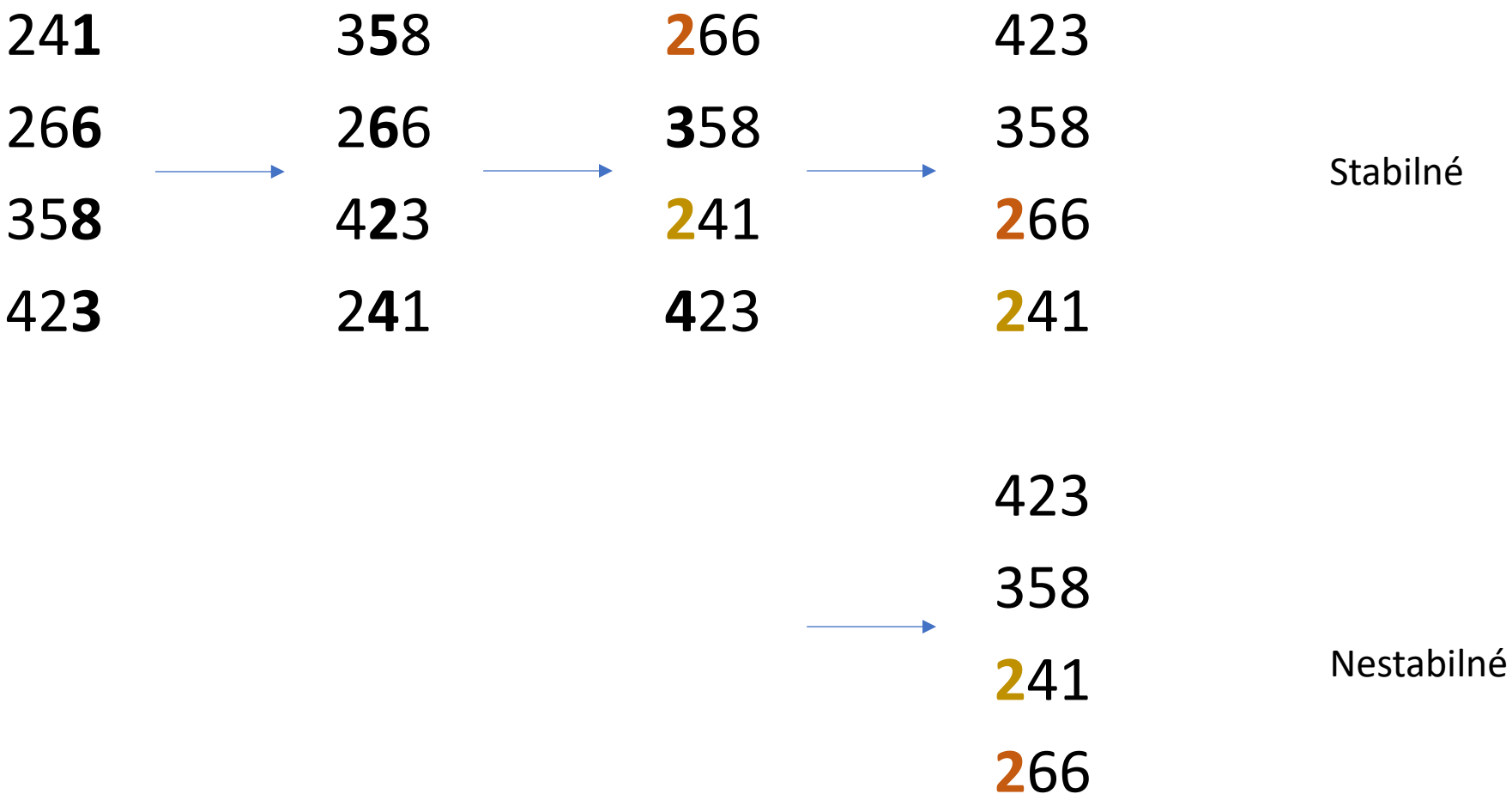
Radix Sort

- Triedime **d-miestne čísla** (viacmiestne) v **d priechodoch**, pričom triedenie musí byť v každom priechode stabilné

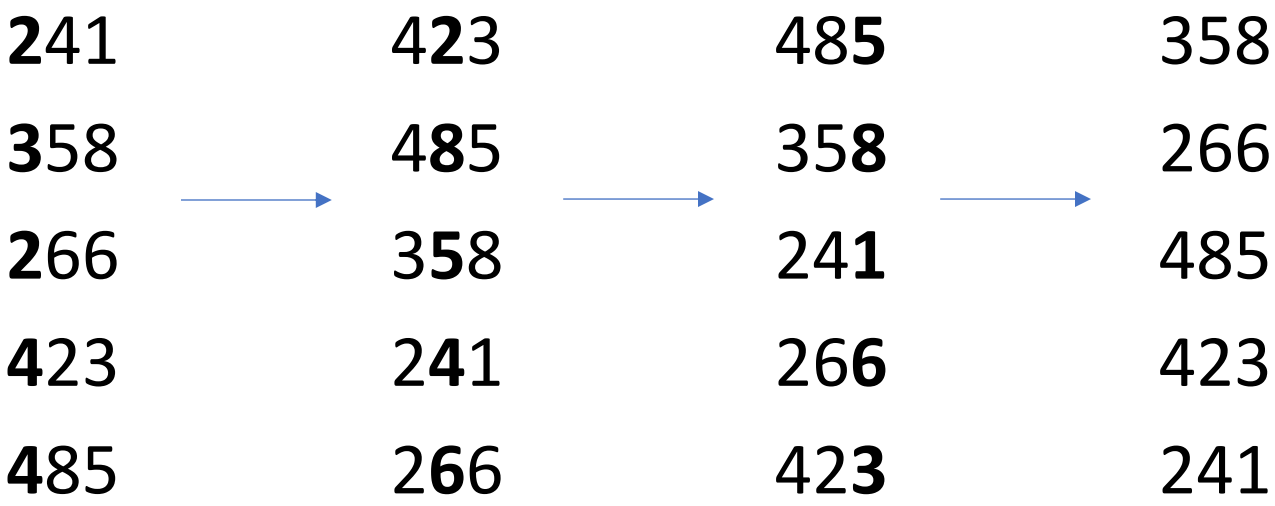
- Môžeme ich triediť od:
  - poslednej číslice – least significant digit (LSD)
  - Prvej číslice – most significant digit (MSD)

- Môžeme takto triediť aj iné dáta s d položkami napr:
  - Dátum - deň, mesiac, rok
  - Textové reťazce – podľa abecedy
  - Bitové reťazce – 32bit ako 8 podreťazcov dĺžky 4

Radix Sort - LSD

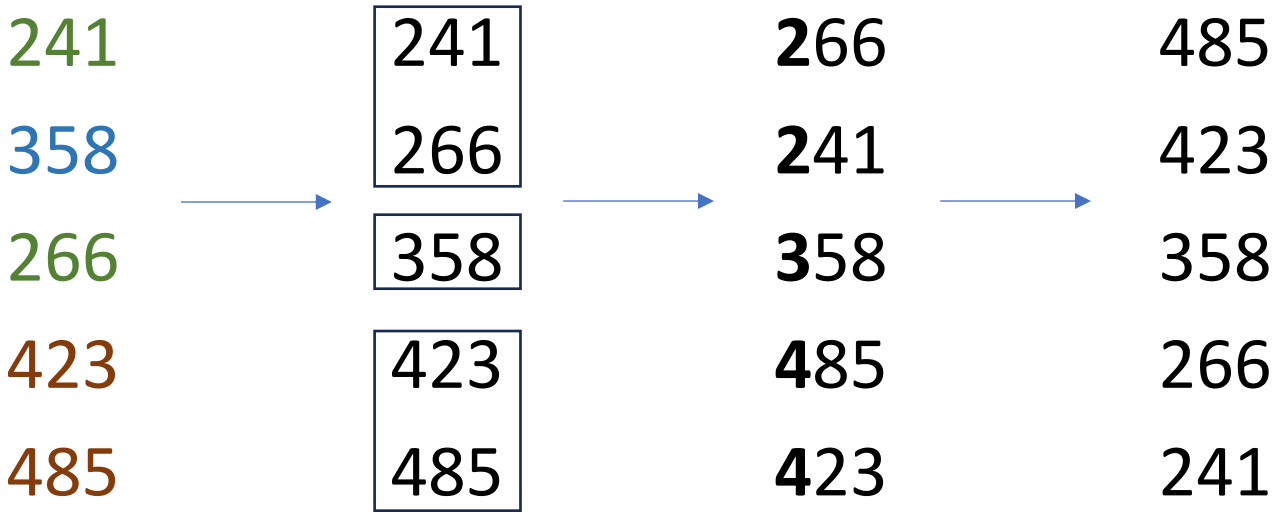


Radix Sort – MSD - nesprávne



Triedili sme stabilne ale dostali sme nesprávny výsledok

Radix Sort – MSD - správne



- Musíme triediť v rámci skupín 2jky, 3jky, 4ky osobitne a potom podľa 1. číslice (alebo naopak)

RadixSort(A, d)

1. **for** i ← 1 **to** d
2. StableSort(A, i)

Každý priechod vykoná pomocou ľubovolného stabilného triediaceho algoritmu

	Time complexity			Space complexity	Stability	In-place	Adaptivity	Comparison
	Best	Average	Worst	Worst				
Selection sort	$O(n^2)$	$O(n^2)$	$O(n^2)$	O(1)	Unstable	In-place	Non-adaptive	Comparison-based
Bubble sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Stable	In-place	Adaptive	Comparison-based
Insertion sort	O(n)	$O(n^2)$	$O(n^2)$	O(1)	Stable	In-place	Adaptive	Comparison-based
Quick sort	O(n log n)	O(n log n)	$O(n^2)$	O(log n)	Unstable	In-place	Non-adaptive	Comparison-based
Merge sort	O(n log n)	O(n log n)	O(n log n)	O(n)	Stable	Not in-place	Non-adaptive	Comparison-based
Heap sort	O(n log n)	O(n log n)	O(n log n)	O(1)	Unstable	In-place	Non-adaptive	Comparison-based
Bucket sort	O(n + k)	O(n + k)	$O(n^2)$	O(n + k)	Stable	Not in-place	Non-adaptive	Non-comparison
Counting sort	O(n + m)	O(n + m)	O(n + m)	O(n + m)	Stable	Not in-place	Non-adaptive	Non-comparison
Radic sort	O(n k)	O(n k)	O(n k)	O(n + b)	Stable	Not in-place	Non-adaptive	Non-comparison

Poor | Fair | Good

- **n** is the size of the data
- In bucket sort, **k** is the number of buckets
- In counting sort, **m** is the data range
- In radix sort, **k** is the maximum number of digits, data in **b** base

www.hello-algo.com

Radix Sort - časová zložitosť

- V najhoršom prípade:
  - Θ(d * čas. zlož. stabilného triediaceho algoritmu)
  - Ak zvolíme napr. Counting Sort ako stabilný triediaci algoritmus (čas. zlož. CountingSort = Θ(n + k)) potom bude čas. zlož. RadixSort Θ(d * (n + k)) => Θ(n)

Úkol

- Simulácia algoritmu
  - Simulujte kroky algoritmu CountingSort na postupnosti
    A = [9, 1, 3, 6, 0, 1, 9, 0]
  - Simulujte kroky algoritmu RadixSort na postupnosti
    A = [130, 111, 212, 321, 330]
    môžete si vybrať ľubovolný stabilný triediaci algoritmus

- Spôsob odovzdávania – info na GitHube na konci README