Počítací třídění *Counting sort*

- předpoklad: pro nějaké (ne moc velké) K všechny klíče jsou typu *integer* $\in$ 0..K
- základní myšlenka: když pro každý prvek X víme, **kolik** prvků je menších, můžeme X rovnou zatřídit
- příklad: jestliže víme, že 17 prvků je menších než X, tak X musí ležet na 18. pozici
- s malou modifikací lze zpracovat i případ více shodných klíčů

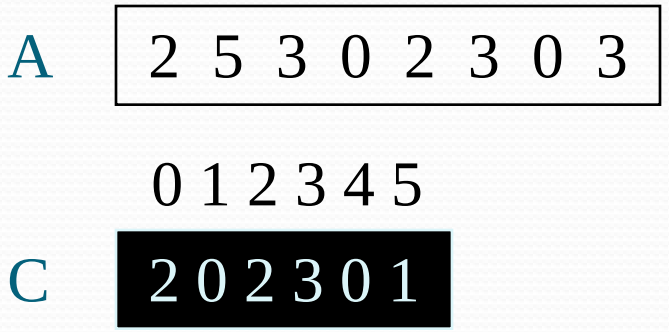
Counting sort princip (1)

- Budeme pracovat se 3 poli: A[1..N] bude obsahovat vstupní data, B[1..N] setříděná data a C[1..**K**] je pomocné pole. N je počet dat a K je limit velikosti klíče
- s pomocí této znalosti přeřadíme prvky z B do A

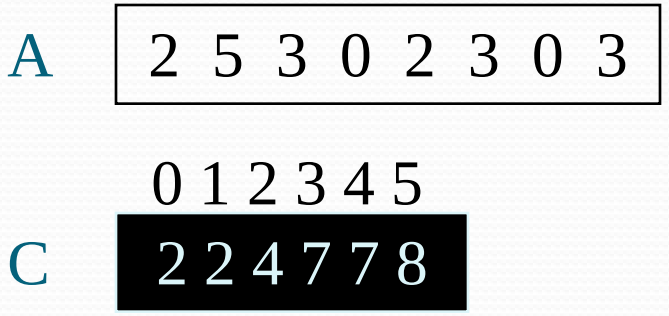
A	2 5 3 0 2 3 0 3
B	? ? ? ? ? ? ? ?
C	0 0 0 0 0 0

Counting sort princip (2)

- do C[**i**] napočítáme počet prvků s klíčem rovným **i**

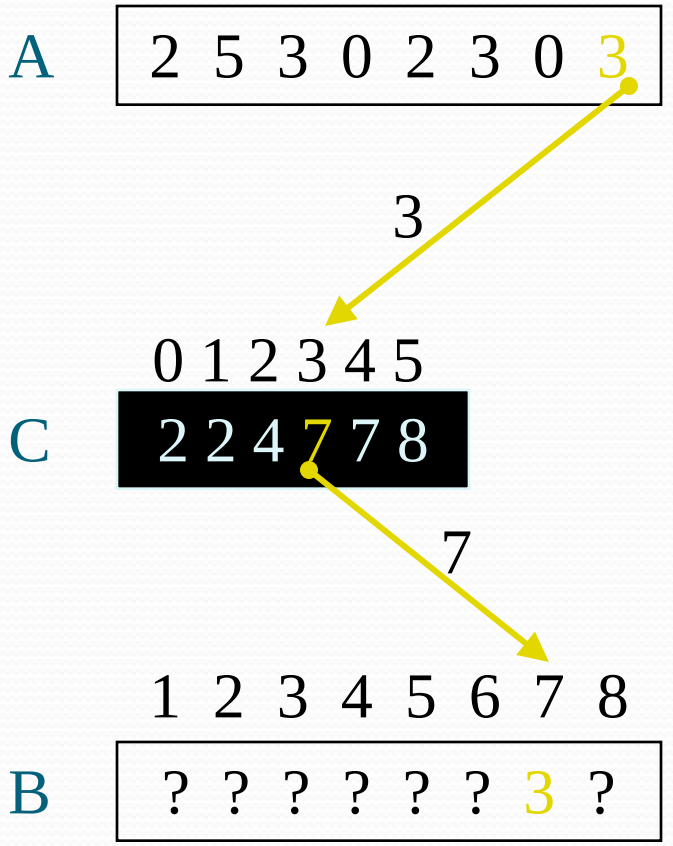


- hodnoty sečteme tak, aby C[**i**] obsahovalo počet prvků s klíčem menším než **i**



Counting sort princip (3)

- postupně pro všechny prvky z A, zprava doleva:
- vezmeme prvek
- jeho klíč použijeme jako index do C
- hodnotu nalezenou v C použijeme jako index do B, kam uložíme prvek



Counting sort princip (4)

- předcházející postup by vyhovoval, kdyby všechny klíče byly jedinečné
- připouštíme-li opakování klíčů, po každém zápisu do B musíme zmenšit C[**i**] aby následující zápis šel na předchozí index takže v našem příkladu se C[**i**] zmenší ze 7 na 6 a následující klíč 3 už půjde do B[6]
- výsledné pole B po zpracování:

	1	2	3	4	5	6	7	8
B	0	0	2	2	3	3	3	5

Counting sort program

```
for i := 0 to K do C[i] := 0;
for i := 1 to High(A) do
   C[ A[i] ] := C[ A[i] ] + 1;
for i := 1 to K do
   C[i] := C[i] + C[i+1];
for i := High(A) downto 1 do
   begin
   B[ C[A[i]] ] := A[i];
   C[ A[i] ] := C[ A[i] ] - 1;          // dec(….)
   end;
```

Bucket sort

- *bucket* = kbelík
- dává dobrou rychlost (skoro lineární ≈N) za předpokladu **rovnoměrného rozložení,** tzn. že všechny klíče jsou stejně pravděpodobné
- funguje na klíče *real* ∈ <0..1)
- nebo když lze klíče na tento interval přetransformovat

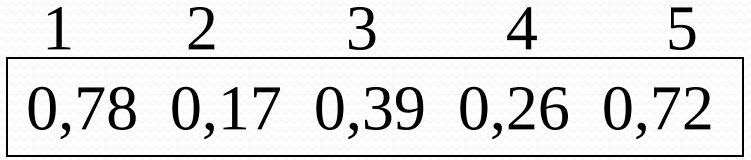
Bucket sort princip

- interval <0..1) se rozdělí na N stejných podintervalů (to jsou ty kýble ☺)
- protože rozdělení je rovnoměrné, do každého intervalu padne jen málo klíčů, takže setřídění uvnitř intervalu je snadné a rychlé (trochu to připomíná hashování)
- postupně projdeme intervaly a v nich klíče podle velikosti

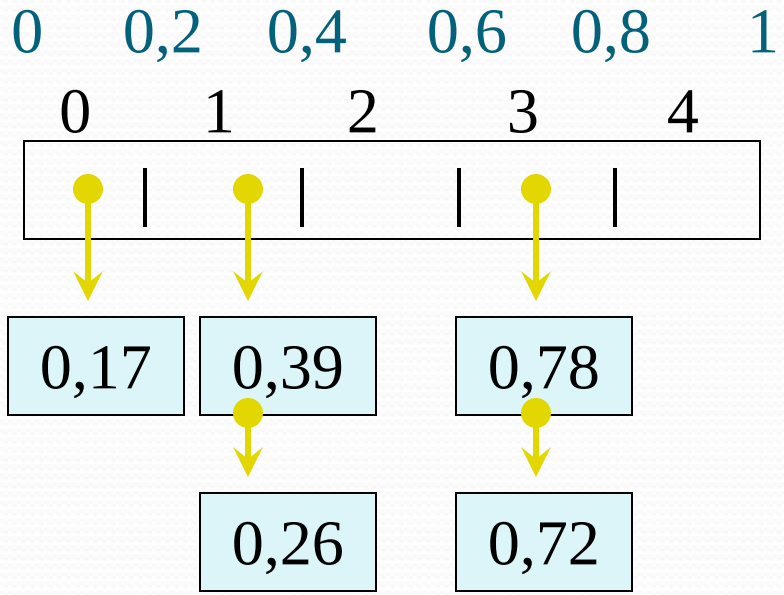
Bucket sort příklad

```
for i := 1 to High(A) do
   begin
   Podle A[i] určit interval
   j;
   Přidat A[i] k seznamu
   začínajícímu v B[j];
   end;
for j := 0 to 4 do
   Setřídit seznam B[j];
Spojit seznamy
```

Vstup (pole A):



Pracovní pole B:



Radix Sort

- Algoritmus řazení
- Příklady použití
- Vlastnosti
- Příklad
- Složitost

Algoritmus řazení

```
Var pole: array [1..10] of integer;
Procedure RadixSort (N, K: integer);
Var i, j, rad, C: integer;
begin
  rad:= 10;
  for C:= 0 to 9 do
    Vytvor(Prihradka[C]);
  for i:=K downto 1 do
  begin
    for j:= 1 to N do
      Pridej(Prihradka[pole[j] mod rad], pole[j];
   j:=1;
   for C:= 0 to 9 do
     while (not (JePrazdna(Prihradka[C]))) do
     begin
       Odeber(Prihradka[C], pole[j]);
       j:=j+1;
     end;
  rad:=rad* 10;
 end;
end;
```

Příklady použití

Pokud potřebujeme algoritmus, který není závislý na druhu dat, použijeme Radix Sort.

Historické použití: třídění děrných štítků na mechanické třídičce.

Současné použití (softwarové verze): třídění dat s vícenásobnými hierarchicky uspořádanými klíči (např. rok, měsíc, den), třídění alfanumerických klíčů (slov).

Upozornění

Pokud jsou na vstupu:

- čísla s různým počtem cifer-> pak doplníme zleva nulami
- Slova různých délek-> pak doplníme zprava mezerami

Vlastnosti

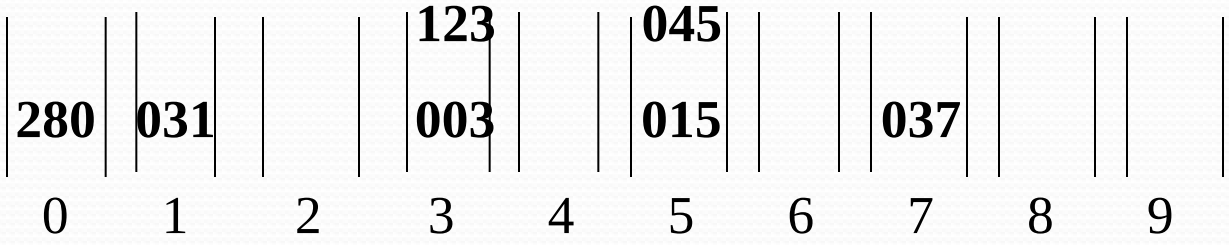
- Radix Sort je z časového hlediska nejefektivnější
- Tato metoda je rychlejší než Quick Sort
- Radix Sort je podstatně paměťově náročný
- Tato metoda je stabilní

Vstup: **123 45 31 15 37 280 3**

Zleva doplním nuly:

Mezivýsledek: **123 045 031 015 037 280 003**

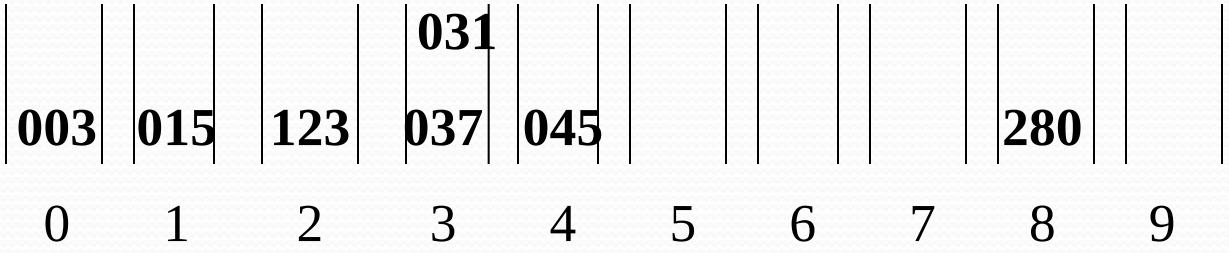
Zavedu 10 přihrádek typu fronta pro každou číslici z intervalu 0 až 9. Do každé přihrádky přidělím číslici podle poslední číslice:



Čísla spojíme z přihrádek do nové posloupnosti:

Mezivýsledek: **280 031 123 003 045 015 037**

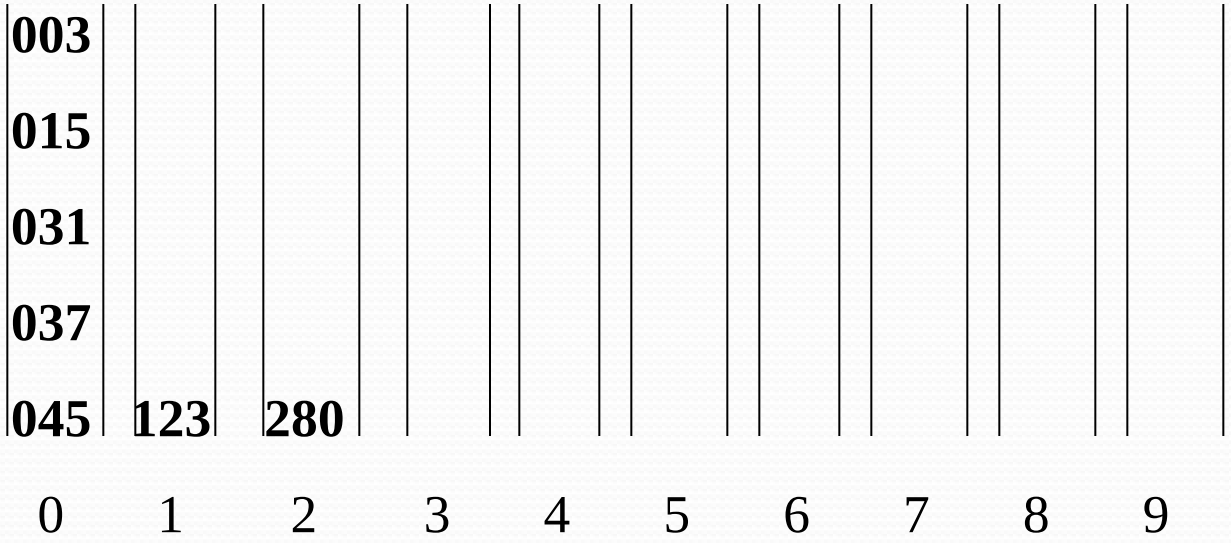
Novou posloupnost zařazujeme podle jejich předposlední číslice:



Poté čísla spojíme do další nové posloupnosti:

<u>Mezivýsledek:</u> **003 015 123 031 037 045 280**

Novou posloupnost zařazujeme podle jejich 3 číslice od konce:



Poté čísla spojíme do další nové posloupnosti:

Výstup: **003 015 031 045 037 123 280**

Upozornění: čísla z přihrádek vybíráme do nové posloupnosti v takovém pořadí, v jakém jsme je do přihrádek vložili – proto používáme datovou strukturu fronta (a ne zásobník)

Složitost

Časová složitost metody je O(n), protože zpracováváme posloupnost délky N a to právě tolikrát, kolik je počet cifer největšího čísla v posloupnosti (nejdelšího slova)