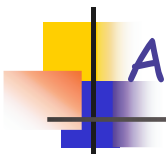
Ordenação em Tempo Linear

Algoritmos e Estuturas de Dados

Cátia Vaz



Algoritmos de Ordenação

- *insertion sort*, *selection sort*, *merge sort*, *quicksort* e *heapsort* são algoritmos de ordenação baseados em comparações.
  - Ω (n lg n) é um limite inferior para algoritmos de ordenação baseados em comparações.
- *counting sort*, *radix sort* e *bucket sort* são algoritmos de ordenação em "tempo linear" que utilizam outras operações para determinar a ordem de ordenação.

Cátia Vaz

Counting Sort

- Assume que cada um dos **n** elementos do array **A** é um inteiro entre **0...k**
- Se **k = O(n)**, a ordenação é em tempo linear
- Algoritmo:
  - Determina para cada elemento **x** **o número de elementos menores ou iguais;**
  - Utiliza a informação anterior para **colocar directamente cada elemento x no array final;**
- Requer dois arrays:
  - Um array **C** com dimensão **k**, para **contar o número de elementos menores ou iguais;**
  - Um array **B** com dimensão **n**, **para colocar os elementos ordenados.**

Cátia Vaz

Counting Sort

```
COUNTING-SORT(A, B, n, k)
for i ← 0 to k
    do C[i] ← 0
for j ← 1 to n
    do C[A[j]] ← C[A[j]] + 1
for i ← 1 to k
    do C[i] ← C[i] + C[i − 1]
for j ← n downto 1
    do B[C[A[j]]] ← A[j]
       C[A[j]] ← C[A[j]] − 1
```

Time: $\Theta(n+k) = \Theta(n)$ se $k = O(n)$

C com dimensão k, em que **C[k]** é atribuído o número de elementos iguais k.

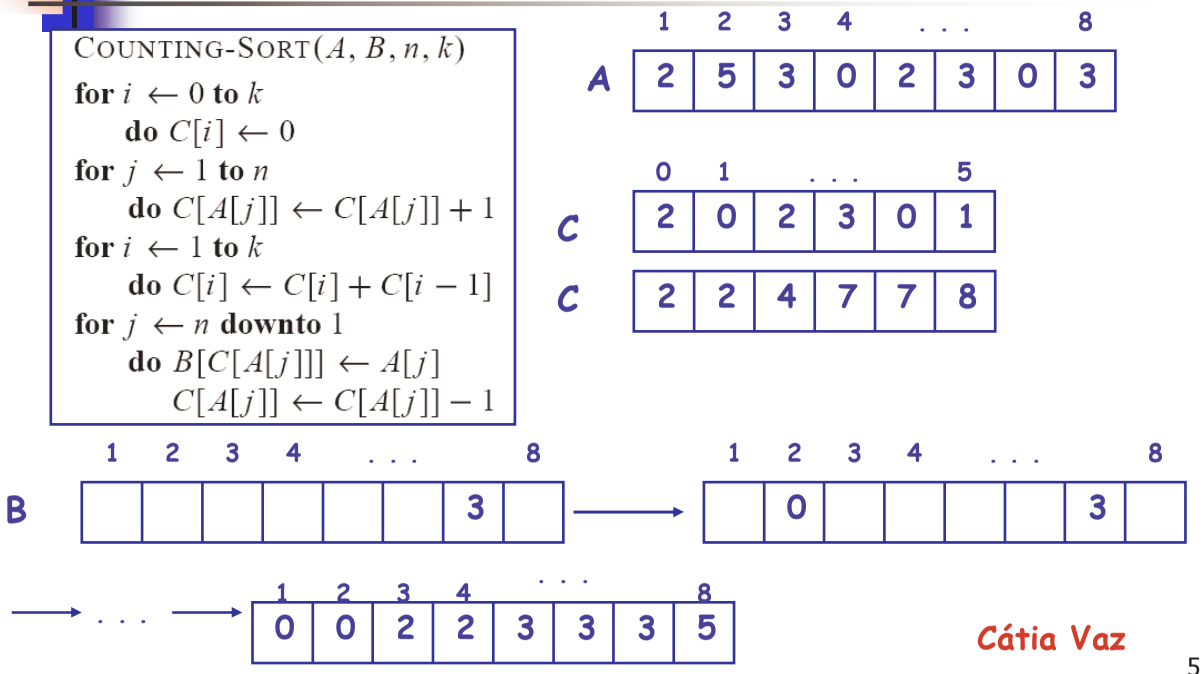
Actualiza **C[k]** com o número de elementos menores ou iguais k.

Como os elementos podem ser iguais, decrementa-se **C[A[j]]** cada vez que se coloca um **A[j]** no array B.

Cátia Vaz

Counting Sort

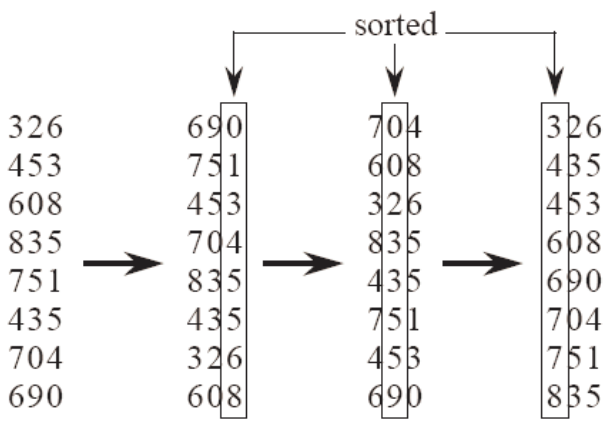
```
COUNTING-SORT(A, B, n, k)
for i ← 0 to k
    do C[i] ← 0
for j ← 1 to n
    do C[A[j]] ← C[A[j]] + 1
for i ← 1 to k
    do C[i] ← C[i] + C[i − 1]
for j ← n downto 1
    do B[C[A[j]]] ← A[j]
       C[A[j]] ← C[A[j]] − 1
```



Radix Sort

- Ideia do Algoritmo:
  - Ordenar primeiro o dígito menos significativo;
  - **Para ordenar d dígitos:**

```
RADIX-SORT(A, d)
for i ← 1 to d
    do use a stable sort to sort array A on digit i
```



Radix-sort - análise

- Assumindo que se usa `counting-sort`
  - $\Theta(n+k)$ por passo (digitos entre $0, \ldots, k$)
  - d passos
  - $\Theta(d(n+k))$ total
  - Se $k = O(n)$, time $= \Theta(dn)$.
- Como quebrar cada chave em dígitos
  - n words e b bits/word.
  - Break em r-bit por digito, $d = \lceil b/r \rceil$.
  - Usando counting sort, $k = 2^r - 1$.

    Exemplo: 32-bit words, 8-bit por digito. $b = 32$, $r = 8$,
    $d = \lceil 32/8 \rceil = 4$, $k = 2^8 - 1 = 255$.
  - Time $= \Theta\left(\frac{b}{r}(n + 2^r)\right)$.

Radix-sort - análise

- **Como escolher o número de bits por dígito**
  - Escolhendo $r \approx \lg n$

    $\Theta\left(\frac{b}{\lg n}(n+n)\right) = \Theta(bn/\lg n)$.
  - Se $r < \lg n$, $b/r > b/\lg n$ e $k = 2^r < n$
  - Se $r > \lg n$, $k = 2^r > n$.

    Exemplo: $r = 2\lg n \Rightarrow 2^r = 2^{2\lg n} = (2^{\lg n})^2 = n^2$.

Ordenar 2^{16} 32-bit numeros, $r = \lg 2^{16} = 16$ bits. $\lceil b/r \rceil = 2$ passos.