

## Fontes principais

1. Cormen T. H.; Leiserson C. E.; Rivest R.; Stein C.. *Introduction to Algorithms*, 3<sup>a</sup> edição, MIT Press, 2009
2. Análise de algoritmo - IME/USP (prof. Paulo Feofiloff)  
[http://www.ime.usp.br/~pf/analise\\_de\\_algoritmos](http://www.ime.usp.br/~pf/analise_de_algoritmos)

## Limite inferior para ordenação

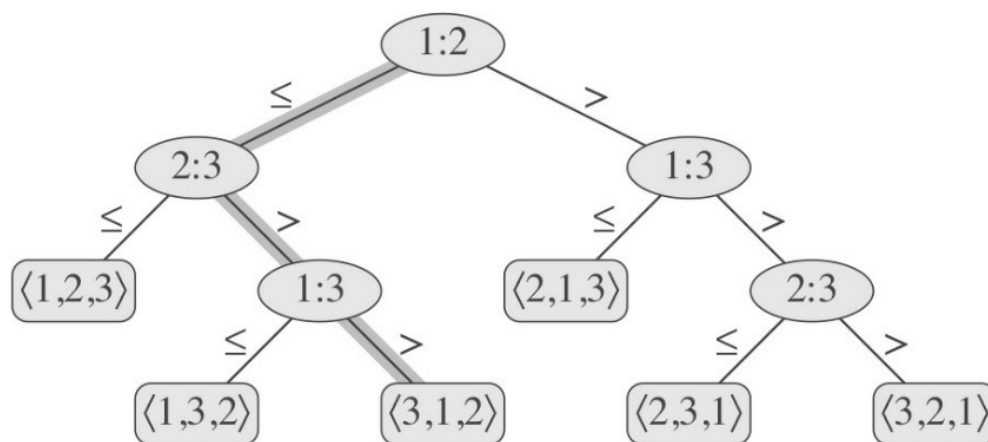
## Limite inferior para ordenação

Um algoritmo de ordenação baseia-se em comparações se o fluxo do algoritmo para uma entrada de tamanho  $n$  depende apenas de comparações do tipo  $a_i \leq a_j$ . Todos os algoritmos de ordenação que estudamos até o momento baseiam-se em comparações.

## Limite inferior para ordenação

**Theorem 1.** *Qualquer algoritmo de ordenação por comparação exige  $\Omega(n \lg n)$  comparações no pior caso.*

**Dem.:** Por árvore de decisão.



## Limite inferior para ordenação

Dado um vetor com  $n$  elementos, o vetor ordenado pode ser qualquer uma das  $n!$  permutações.

Um algoritmo de ordenação efetua um número de comparações equivalente a altura da árvore com  $n!$  folhas.

Uma árvore binária com altura  $h$  tem no máximo  $2^h$  folhas.

## Limite inferior para ordenação

**Theorem 2.** *Qualquer árvore de decisão que ordena  $n$  elementos tem altura  $\Omega(n \log n)$ .*

**Dem.:** Considere uma árvore de decisão de altura  $h$ , que ordena  $n$  elementos. Como existem  $n!$  permutações distintas de  $n$  elementos, cada uma apresentando uma ordenação desses elementos, então a árvore de decisão deve ter  $n!$  folhas.

## Limite inferior para ordenação

Também como uma árvore binária de altura  $h$  tem no máximo  $2^h$  folhas, então a árvore de decisão tem no máximo  $2^h$  folhas.

Assim,

$$n! \leq \text{número de folhas} \leq 2^h$$

Logo, temos:

$$\begin{aligned} n! \leq 2^h &\Rightarrow \lg(n!) \leq h & (a \leq b \Rightarrow \lg a \leq \lg b) \\ h &\geq \lg(n!) \end{aligned}$$

## Limite inferior para ordenação

Pela aproximação de Stirling

$$n! = \sqrt{2\pi n} \left(\frac{n}{e}\right)^n > \left(\frac{n}{e}\right)^n$$

Portanto, temos

$$\begin{aligned} h &\geq \log(n!) \Rightarrow h > \log \left(\frac{n}{e}\right)^n \\ &= n \log \frac{n}{e} \\ &= n \log n - n \log e = \Omega(n \log n) \end{aligned}$$





## Ordenação em tempo linear

## Algoritmo Counting Sort

Cada elemento é um inteiro  $i$  no intervalo  $[1..k]$

Idéia:

- ▶ Contar para cada elemento  $x$  da entrada o número de elementos menores que  $x$ .
- ▶ Colocar  $x$  diretamente na sua posição no vetor de saída.

Utiliza dois vetores auxiliares para ordenar.

## Algoritmo Counting Sort

Counting-Sort( $A, B, n, k$ )		custo
1	<b>para</b> $i = 1$ <b>até</b> $k$ <b>faça</b>	$\Theta(k)$
2	$C[i] = 0$	$\Theta(k)$
3	<b>para</b> $j = 1$ <b>até</b> $n$ <b>faça</b>	$\Theta(n)$
4	$C[A[j]] = C[A[j]] + 1$	$\Theta(n)$
5	▷ $C[i]$ contém núm. de elementos iguais a $i$	
6	<b>para</b> $i = 2$ <b>até</b> $k$ <b>faça</b>	$\Theta(k)$
7	$C[i] = C[i] + C[i - 1]$	$\Theta(k)$
8	▷ $C[i]$ contém núm. de elementos $\leq i$	
9	<b>para</b> $j = n$ <b>até</b> $1$ <b>faça</b>	$\Theta(n)$
10	$B[C[A[j]]] = A[j]$	$\Theta(n)$
11	$C[A[j]] = C[A[j]] - 1$	$\Theta(n)$

## Algoritmo Counting Sort

Entrada:

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

	1	2	3	4	5	6	7	8	9	10
B										

Saída:

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

Estão na mesma ordem em que aparece em A.

## Algoritmo Counting Sort

Passo a passo:

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

	1	2	3	4	5	6	7	8	9	10
B										

	0	1	2	3	4	5
C						

Atividade: "Simular" a aplicação do algoritmo Counting-Sort.

## Algoritmo Counting Sort

Passo a passo:

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

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

	0	1	2	3	4	5
C	0	3	3	5	8	8

Perceba a estabilidade do algoritmo

## Classificação dos algoritmos de ordenação

### Estabilidade

- Um algoritmo é **estável** se elementos idênticos ocorrem no vetor ordenado na mesma ordem que foram recebidos como entrada

### Localidade

- Um algoritmo é **local** se a quantidade de memória adicional requerida é constante.

Note que **Counting Sort** é estável mas não local.

Quais são os algoritmos estáveis?

- ▷ Insertion-Sort
- ▷ Merge-Sort
- ▷ Bubble-Sort
- ▷ Counting-Sort
- ▷ Bucket-Sort

Quais não são estáveis?

- ▷ Heap-Sort
- ▷ Quick-Sort
- ▷ Selection-Sort (Depende do algoritmo)
- ▷ Shell-Sort



## Radix-Sort

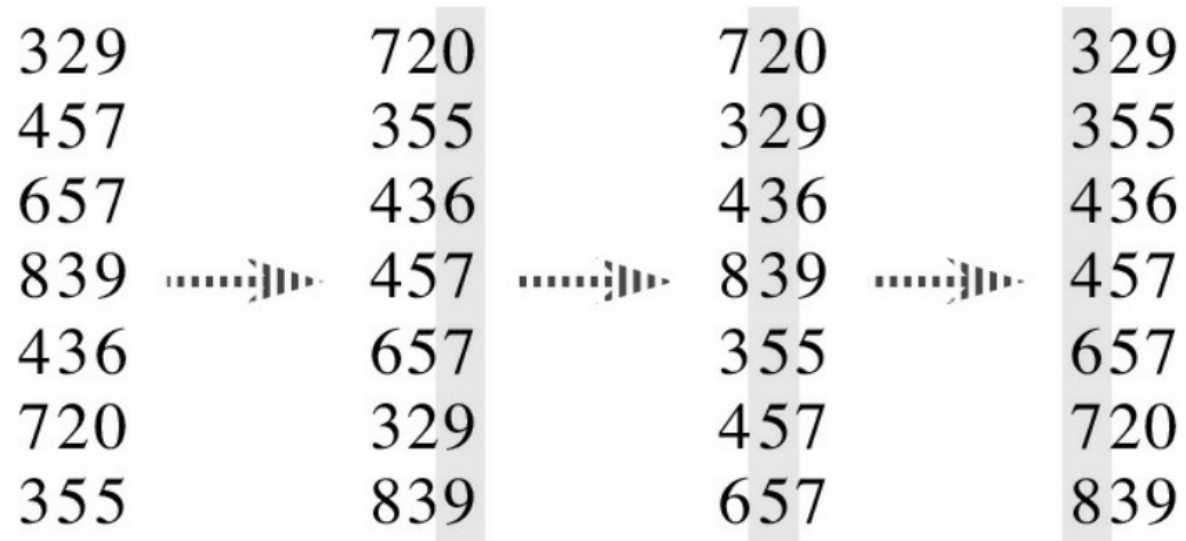
## Radix-Sort

- ▷ Números a serem ordenados tem  $d$  dígitos
- ▷ Inicia-se pelo dígito menos significativo

Radix-Sort( $A, n, d$ )

- 1 **para**  $i = 1$  **até**  $d$  **faça**
- 2     ordena  $A[1..n]$  pelo  $i$ -ésimo dígito com um algarismo **estável**

## Radix-Sort



## Complexidade do Radix-Sort

A análise do tempo de execução depende da ordenação estável usada como algoritmo ordenação intermediária.

Usando o **Counting-Sort** para dígitos na faixa  $[0 \cdots k-1]$ , a complexidade de pior caso é  $\Theta(d(n+k))$

Se  $k = O(n)$  e  $d = O(1)$ , então  $T(n) = \Theta(n)$ .

Note que se for utilizada, por exemplo, **Insertion-Sort**, como algoritmo estável a complexidade de tempo **não será** a linear.

## Bucket-Sort (Ordenação por "balde")

## Bucket-Sort (Ordenação por "balde")

Os  $n$  elementos do vetor são valores reais distribuídos uniformemente em  $[0, 1)$

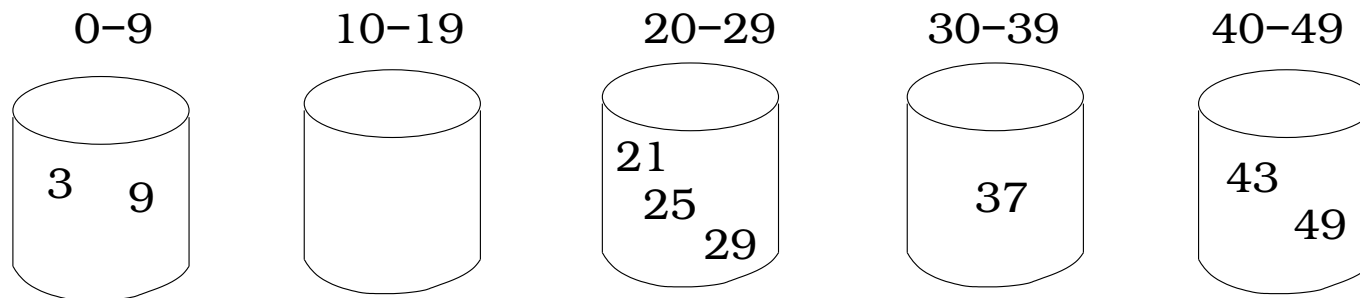
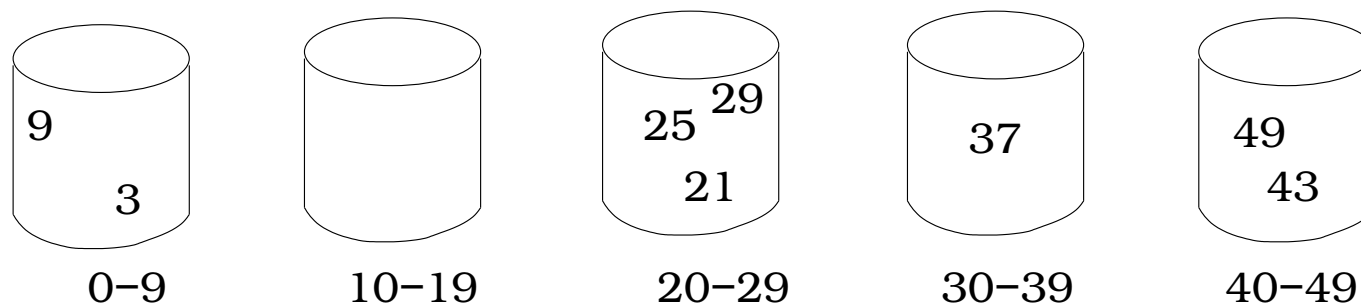
Divide-se o intervalo  $[0, 1)$  em  $n$  buckets de mesmo tamanho e distribui-se os  $n$  elementos nos respectivos buckets

Cada bucket é ordenado por um método "qualquer"

Por fim, os buckets ordenados são concatenados em ordem crescente

## Baldes

Entrada: 29 25 3 49 9 37 21 43



Saída: 3 9 21 25 29 37 43 49

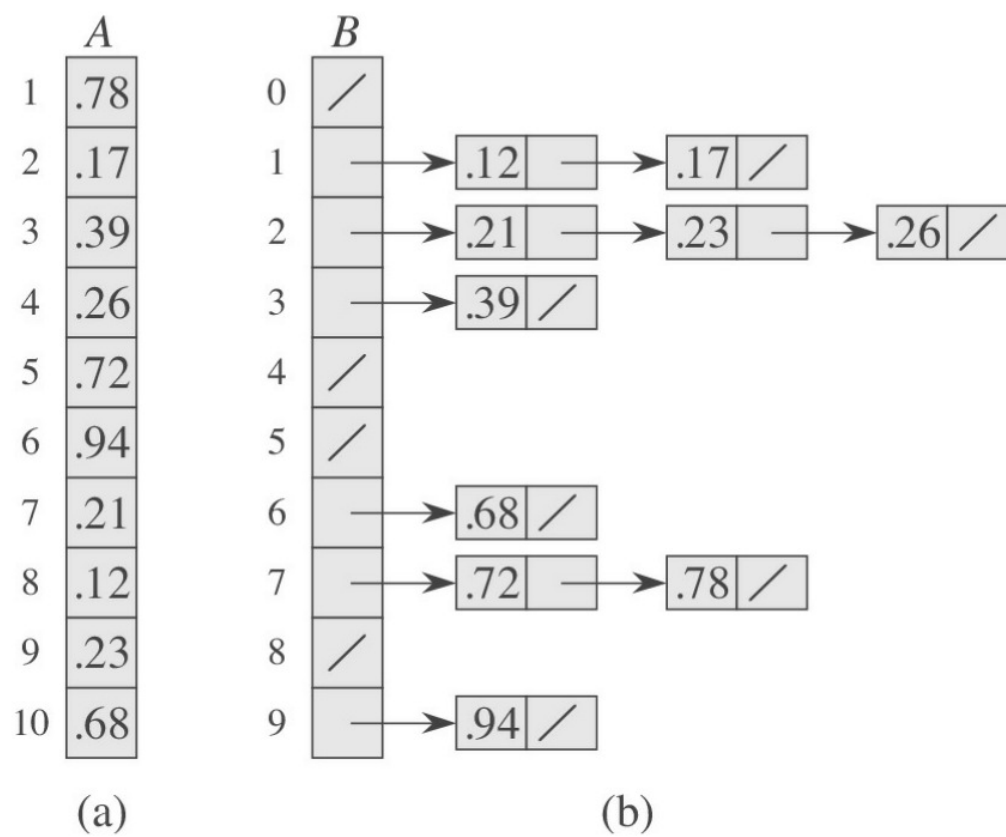
## Bucket-Sort (Ordenação por "balde")

Bucket-Sort( $A, n$ )

```
1  para  $i = 0$  até  $n - 1$ 
2      faça  $B[i] = null$ 
3  para  $i = 1$  até  $n$ 
4      faça insira  $A[i]$  na lista  $B[\lfloor n \cdot A[i] \rfloor]$ 
5  para  $i = 0$  até  $n - 1$ 
6      faça Insertion-Sort( $B[i]$ )
7  Concatene as listas  $B[0], B[1], \dots, B[n - 1]$ 
```



## Exemplo



## Complexidade do Bucket-Sort

**Pior caso:** Supondo *Insertion-Sort* para ordenar as listas  $\Theta(n^2)$

**Caso médio:** Número de elementos em cada lista é  $\Theta(1)$ , e o tempo esperado para ordenar uma lista  $B[i]$  também é  $\Theta(1)$

**Melhor caso:**  $\Theta(n)$

**Complexidade de espaço:**  $\Theta(n)$

## Dicas de website

<https://www.cs.usfca.edu/~galles/visualization/CountingSort.html>

<https://www.cs.usfca.edu/~galles/visualization/RadixSort.html>

<https://www.cs.usfca.edu/~galles/visualization/BucketSort.html>

Neste mesmo site tem visualizações de outros algoritmos

Obrigado