# BCC202 - Estruturas de Dados I

## Aula 18: Ordenação em Tempo Linear

**Pedro Silva**

Universidade Federal de Ouro Preto, UFOP
Departamento de Computação, DECOM
Email: **silvap@ufop.edu.br**

2021

## Conteúdo

## Conteúdo

## Ordenação em tempo linear

Algoritmos de ordenação por **comparação**

- *InsertSort*;
- *SelectSort*;
- *QuickSort*;
- *MergeSort*;
- *HeapSort*...

Possuem *limite assintótico inferior*: $O(n \lg n)$;

Podem existir algoritmos melhores?

## Ordenação em tempo linear

Algoritmos de ordenação por **comparação**

- *InsertSort*;
- *SelectSort*;
- *QuickSort*;
- *MergeSort*;
- *HeapSort*...

Possuem *limite assintótico inferior*: $O(n \lg n)$;

Podem existir algoritmos melhores?

## Ordenação em tempo linear

A resposta é **SIM**, desde que:

- A entrada possui características especiais;
- Algumas restrições devem ser respeitadas;
- O algoritmo não é baseado puramente em comparações;
- A implementação deve ser feita de maneira adequada.

Tempo linear: $\Theta(n)$;

## Algoritmos de ordenação em tempo linear

**Algoritmos:**

- Ordenação por contagem (*Counting Sort*);
- *Radix Sort*;
- *Bucket Sort*.

## Conteúdo

## Definição

Pressupõe que cada elemento da entrada é um inteiro na faixa de 0 a $k$, para algum inteiro $k$.

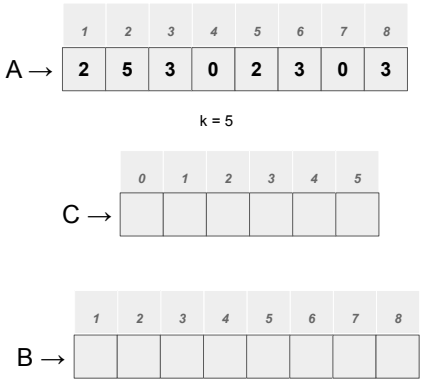
Ideia básica:

- Determinar para cada elemento da entrada $x$ o número de elementos maiores que $x$.
- Com esta informação, determinar a posição de cada elemento.
  - Ex.: Se 17 elementos forem menores que x então x ocupa a posição de saída 18.

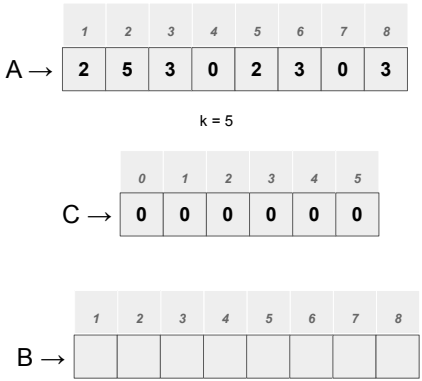
**Algoritmo:**

- Assumimos que o vetor de entrada é $A[1, \cdots, n]$;
- Outros dois vetores são utilizados:
  - $B[1, \cdots, n]$ – armazena a saída ordenada;
  - $C[0, \cdots, k]$ – é utilizado para armazenamento temporário.

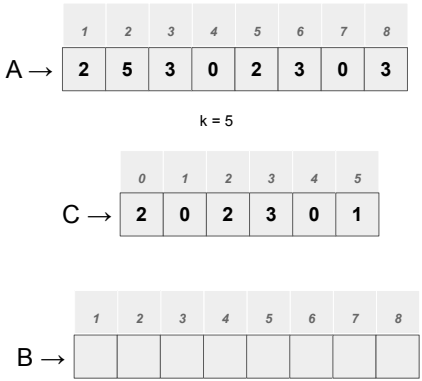
```
1  Algorithm: COUNTING_SORT
   Input: int* A, int n, int k
2  begin
3      for i ← 0 to k do
4          C[i] ← 0
5      end
6      for j ← 1 to n do
7          C[A[j]] ← C[A[j]] + 1
8      end
9      for i ← 1 to k do
10         C[i] ← C[i] + C[i − 1]
11     end
12     for j ← n to 1 do
13         B[C[A[j]]] ← A[j]
14         C[A[j]] ← C[A[j]] − 1
15     end
16 end
```



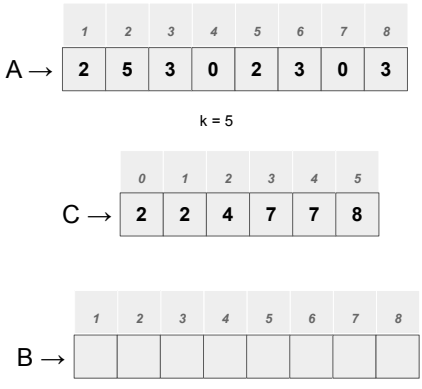
```
1  Algorithm: COUNTING_SORT
   Input: int* A, int n, int k
2  begin
3      for i ← 0 to k do
4          C[i] ← 0
5      end
6      for j ← 1 to n do
7          C[A[j]] ← C[A[j]] + 1
8      end
9      for i ← 1 to k do
10         C[i] ← C[i] + C[i − 1]
11     end
12     for j ← n to 1 do
13         B[C[A[j]]] ← A[j]
14         C[A[j]] ← C[A[j]] − 1
15     end
16 end
```



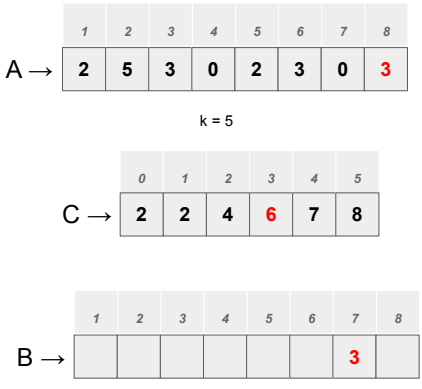
```
1  Algorithm: COUNTING_SORT
   Input: int* A, int n, int k
2  begin
3      for i ← 0 to k do
4          C[i] ← 0
5      end
6      for j ← 1 to n do
7          C[A[j]] ← C[A[j]] + 1
8      end
9      for i ← 1 to k do
10         C[i] ← C[i] + C[i − 1]
11     end
12     for j ← n to 1 do
13         B[C[A[j]]] ← A[j]
14         C[A[j]] ← C[A[j]] − 1
15     end
16 end
```



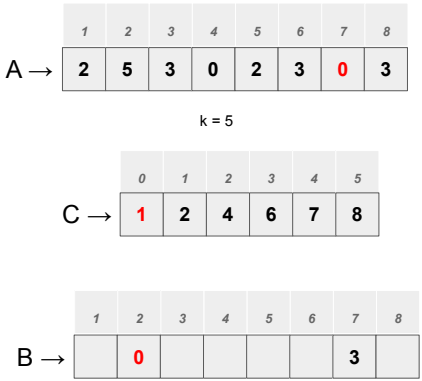
```
1  Algorithm: COUNTING_SORT
   Input: int* A, int n, int k
2  begin
3      for i ← 0 to k do
4          C[i] ← 0
5      end
6      for j ← 1 to n do
7          C[A[j]] ← C[A[j]] + 1
8      end
9      for i ← 1 to k do
10         C[i] ← C[i] + C[i − 1]
11     end
12     for j ← n to 1 do
13         B[C[A[j]]] ← A[j]
14         C[A[j]] ← C[A[j]] − 1
15     end
16 end
```



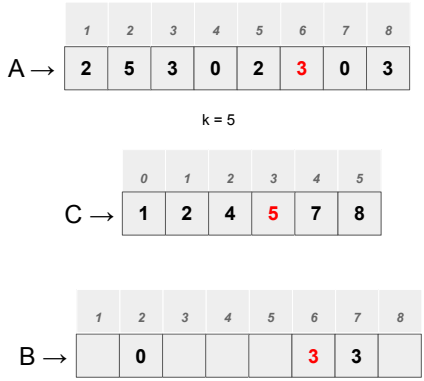
```
1  Algorithm: COUNTING_SORT
   Input: int* A, int n, int k
2  begin
3      for i ← 0 to k do
4          C[i] ← 0
5      end
6      for j ← 1 to n do
7          C[A[j]] ← C[A[j]] + 1
8      end
9      for i ← 1 to k do
10         C[i] ← C[i] + C[i − 1]
11     end
12     for j ← n to 1 do
13         B[C[A[j]]] ← A[j]
14         C[A[j]] ← C[A[j]] − 1
15     end
16 end
```



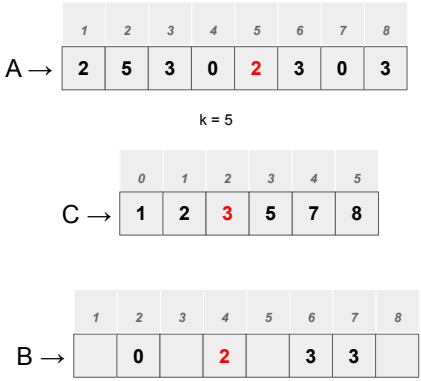
```
1  Algorithm: COUNTING_SORT
   Input: int* A, int n, int k
2  begin
3      for i ← 0 to k do
4          C[i] ← 0
5      end
6      for j ← 1 to n do
7          C[A[j]] ← C[A[j]] + 1
8      end
9      for i ← 1 to k do
10         C[i] ← C[i] + C[i − 1]
11     end
12     for j ← n to 1 do
13         B[C[A[j]]] ← A[j]
14         C[A[j]] ← C[A[j]] − 1
15     end
16 end
```



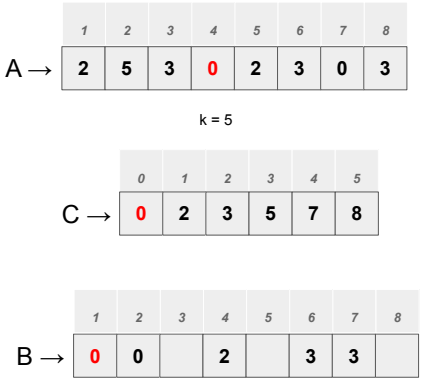
```
1  Algorithm: COUNTING_SORT
   Input: int* A, int n, int k
2  begin
3      for i ← 0 to k do
4          C[i] ← 0
5      end
6      for j ← 1 to n do
7          C[A[j]] ← C[A[j]] + 1
8      end
9      for i ← 1 to k do
10         C[i] ← C[i] + C[i − 1]
11     end
12     for j ← n to 1 do
13         B[C[A[j]]] ← A[j]
14         C[A[j]] ← C[A[j]] − 1
15     end
16 end
```



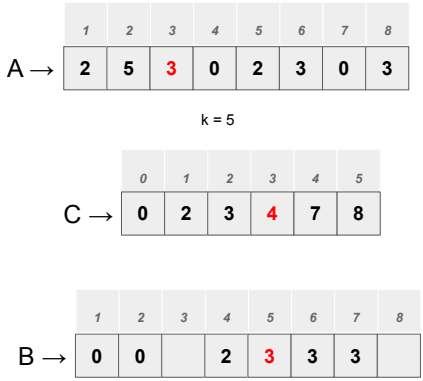
```
1  Algorithm: COUNTING_SORT
   Input: int* A, int n, int k
2  begin
3      for i ← 0 to k do
4          C[i] ← 0
5      end
6      for j ← 1 to n do
7          C[A[j]] ← C[A[j]] + 1
8      end
9      for i ← 1 to k do
10         C[i] ← C[i] + C[i − 1]
11     end
12     for j ← n to 1 do
13         B[C[A[j]]] ← A[j]
14         C[A[j]] ← C[A[j]] − 1
15     end
16 end
```



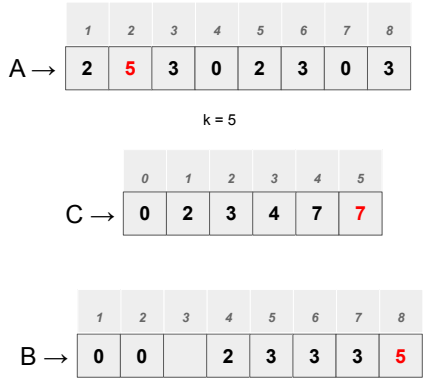
```
1  Algorithm: COUNTING_SORT
   Input: int* A, int n, int k
2  begin
3      for i ← 0 to k do
4          C[i] ← 0
5      end
6      for j ← 1 to n do
7          C[A[j]] ← C[A[j]] + 1
8      end
9      for i ← 1 to k do
10         C[i] ← C[i] + C[i − 1]
11     end
12     for j ← n to 1 do
13         B[C[A[j]]] ← A[j]
14         C[A[j]] ← C[A[j]] − 1
15     end
16 end
```



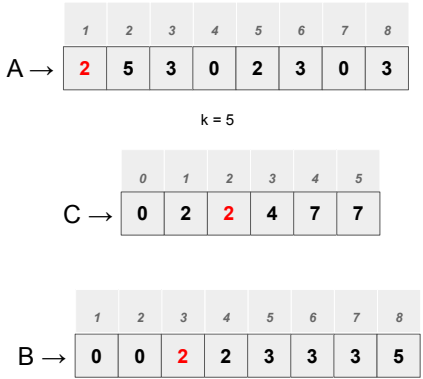
```
1  Algorithm: COUNTING_SORT
   Input: int* A, int n, int k
2  begin
3      for i ← 0 to k do
4          C[i] ← 0
5      end
6      for j ← 1 to n do
7          C[A[j]] ← C[A[j]] + 1
8      end
9      for i ← 1 to k do
10         C[i] ← C[i] + C[i − 1]
11     end
12     for j ← n to 1 do
13         B[C[A[j]]] ← A[j]
14         C[A[j]] ← C[A[j]] − 1
15     end
16 end
```



```
1  Algorithm: COUNTING_SORT
   Input: int* A, int n, int k
2  begin
3      for i ← 0 to k do
4          C[i] ← 0
5      end
6      for j ← 1 to n do
7          C[A[j]] ← C[A[j]] + 1
8      end
9      for i ← 1 to k do
10         C[i] ← C[i] + C[i − 1]
11     end
12     for j ← n to 1 do
13         B[C[A[j]]] ← A[j]
14         C[A[j]] ← C[A[j]] − 1
15     end
16 end
```



```
1  Algorithm: COUNTING_SORT
   Input: int* A, int n, int k
2  begin
3      for i ← 0 to k do
4          C[i] ← 0
5      end
6      for j ← 1 to n do
7          C[A[j]] ← C[A[j]] + 1
8      end
9      for i ← 1 to k do
10         C[i] ← C[i] + C[i − 1]
11     end
12     for j ← n to 1 do
13         B[C[A[j]]] ← A[j]
14         C[A[j]] ← C[A[j]] − 1
15     end
16 end
```



## Considerações sobre o método *CoutingSort*

- O tempo de execução é dado em função do valor de k.
- Roda em tempo $\Theta(n+k)$.
- Se tivermos $k = O(n)$, então o algoritmo executa em tempo $\Theta(n)$.
- Exemplo prático de uso: vídeo locadora.

## Conteúdo

## Definição

Pressupõe que as chaves de entrada possuem limite no valor e no tamanho (quantidade de dígitos).

Ideia básica:

- Ordena em função dos dígitos (um de cada vez):
  - A partir do mais significativo.
  - Ou a partir do menos significativo?
- É essencial utilizar um segundo algoritmo estável para realizar a ordenação de cada dígito.

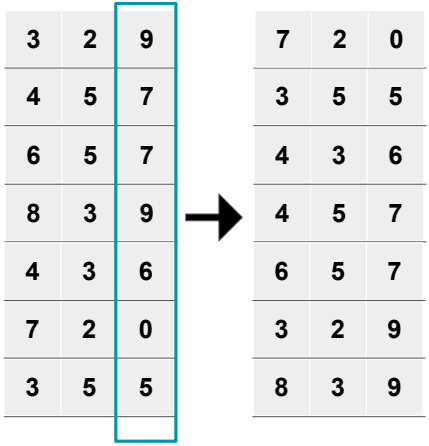
## Funcionamento

### A partir dos dígitos menos significativos:

| | | |
|---|---|---|
| 3 | 2 | 9 |
| 4 | 5 | 7 |
| 6 | 5 | 7 |
| 8 | 3 | 9 |
| 4 | 3 | 6 |
| 7 | 2 | 0 |
| 3 | 5 | 5 |

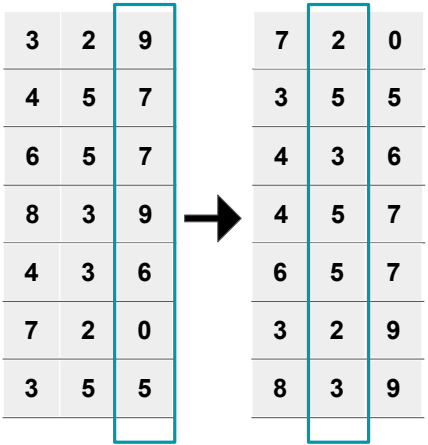
## Funcionamento

### A partir dos dígitos menos significativos:



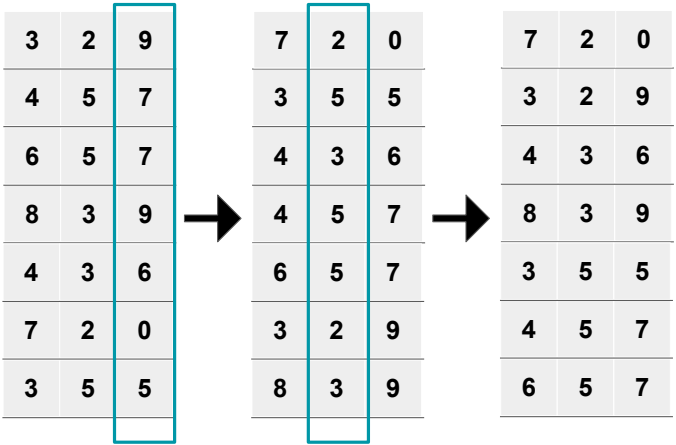
## Funcionamento

### A partir dos dígitos menos significativos:



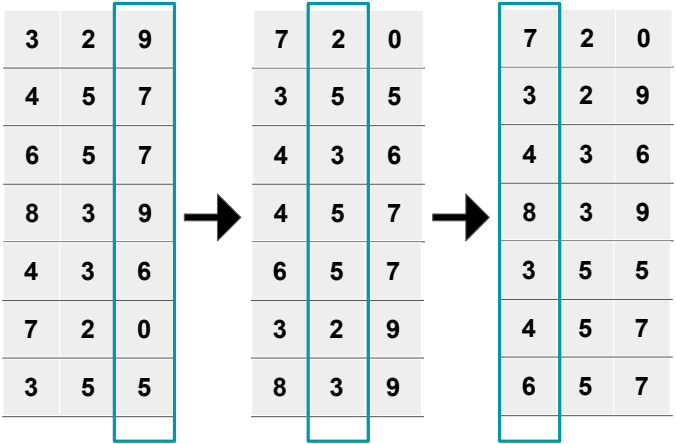
## Funcionamento

### A partir dos dígitos menos significativos:



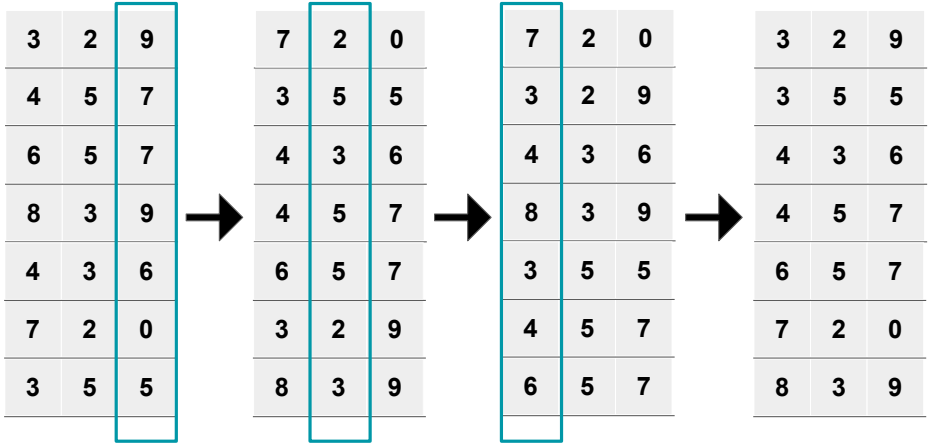
## Funcionamento

### A partir dos dígitos menos significativos:



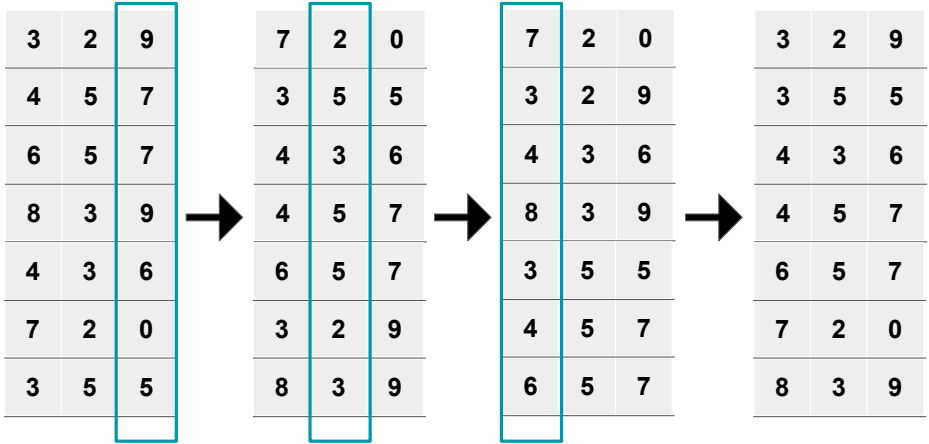
## Funcionamento

### A partir dos dígitos menos significativos:



## Funcionamento

**A partir dos dígitos menos significativos: (Como ficaria a partir do dígito mais significativo?)**



## *Radix Sort* - Pseudo Código

- Como dito anteriormente, o *Radix Sort* consiste em usar um outro método de ordenação (estável) para ordenar as chaves em relação a cada dígito.
- O código, portanto, é muito simples:

```
1 Algorithm: RADIX_SORT
  Input: int* v, int n, int d
2 begin
3     for i ← 0 to d do
4         // utilizar um algoritmo estável para ordenar o array v pelo i-ésimo dígito
5         ORDENA_I_ESIMO_DIGITO(v, i, n)
6     end
7 end
```

- Onde:
  - *d* é número de dígitos.
  - *v* é o array de entrada.
  - *n* é o tamanho do array de entrada.

## Conteúdo

## Definição

Pressupõe que a entrada consiste em elementos distribuídos de forma uniforme sobre o intervalo $[0,1)$.

Ideia básica:

- A ideia do *Bucket Sort* é dividir o intervalo $[0,1)$ em $n$ subintervalos de mesmo tamanho (baldes), e então distribuir os n números nos baldes.
- Uma vez que as entradas são uniformemente distribuídas não se espera que muitos números caiam em cada balde.

## Conceitos do *Bucket Sort*

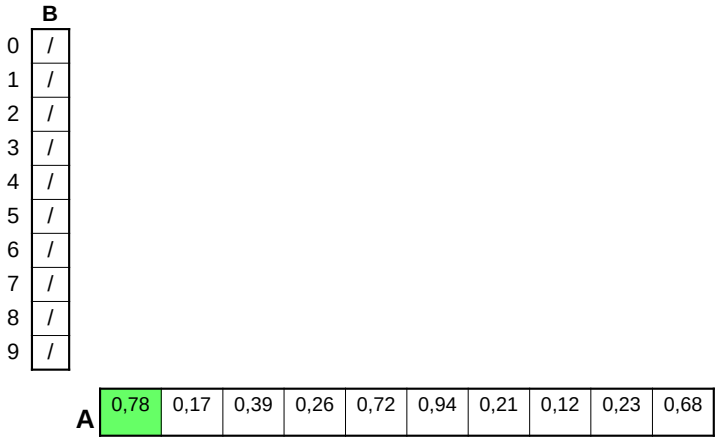
Para produzir a saída ordenada, basta ordenar os números em cada balde, e depois examinar os baldes em ordem, listando seus elementos.

A função para determinação do índice do balde correto é $\lfloor n \times A[i] \rfloor$.

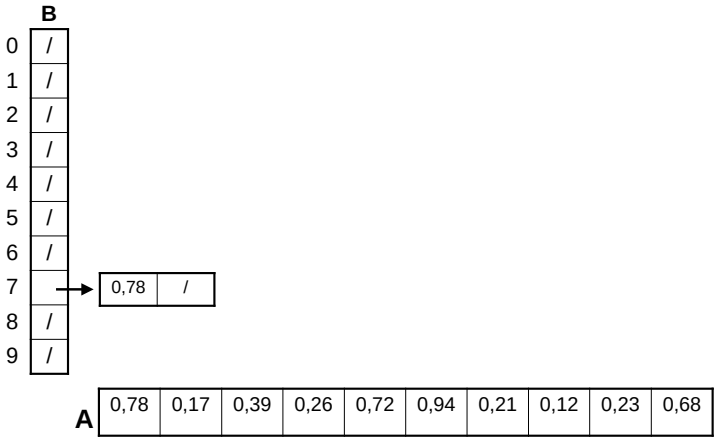
Vamos a um exemplo com 10 números:

- A é o array de entrada.
- B é o array com os baldes.

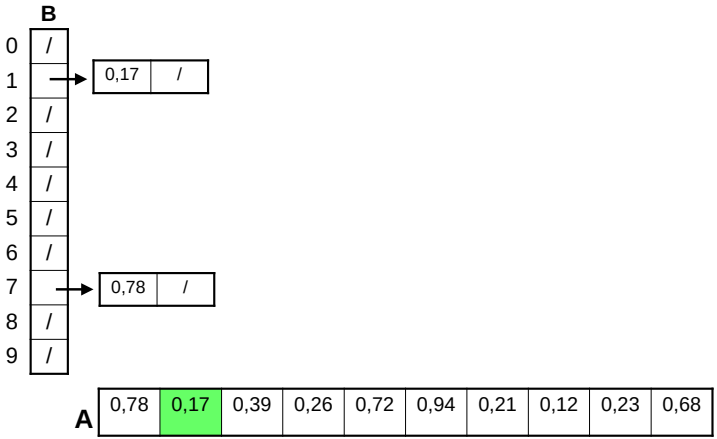
# Funcionamento



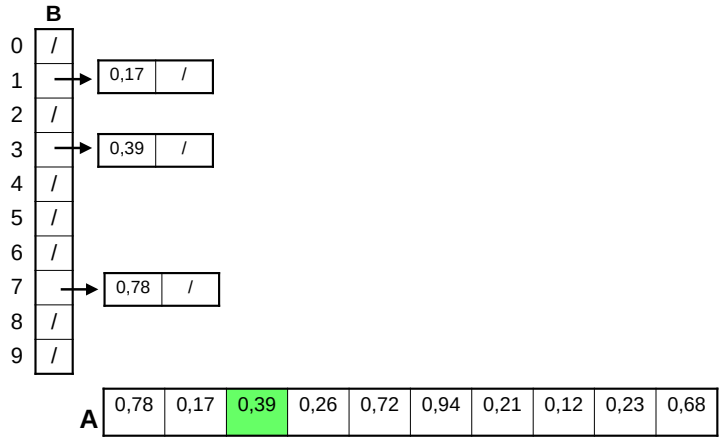
# Funcionamento



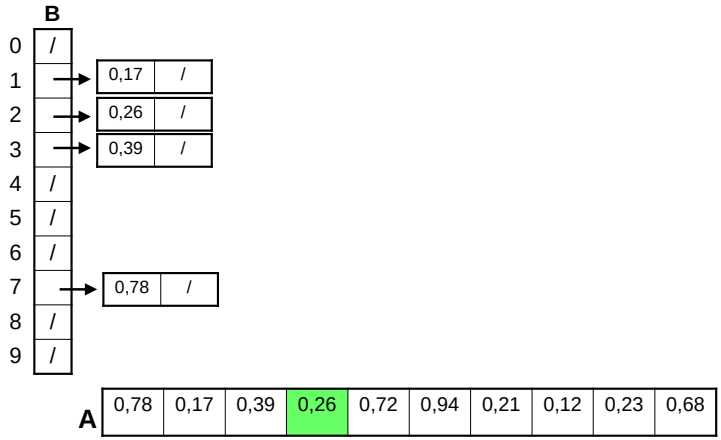
# Funcionamento



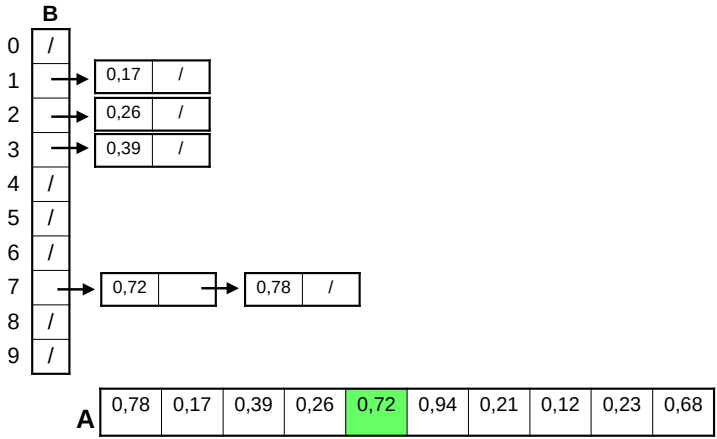
# Funcionamento



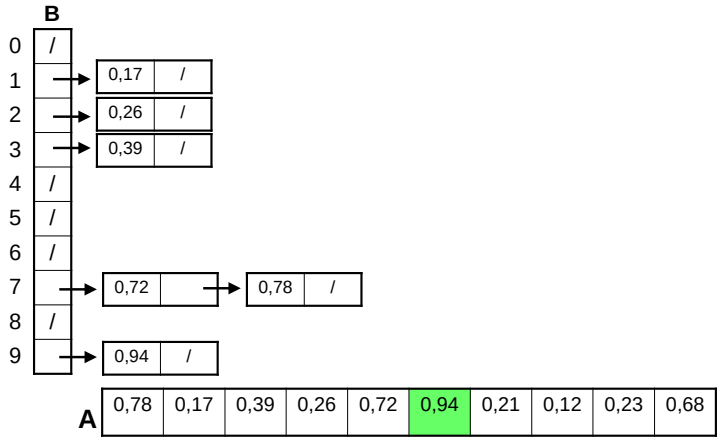
# Funcionamento



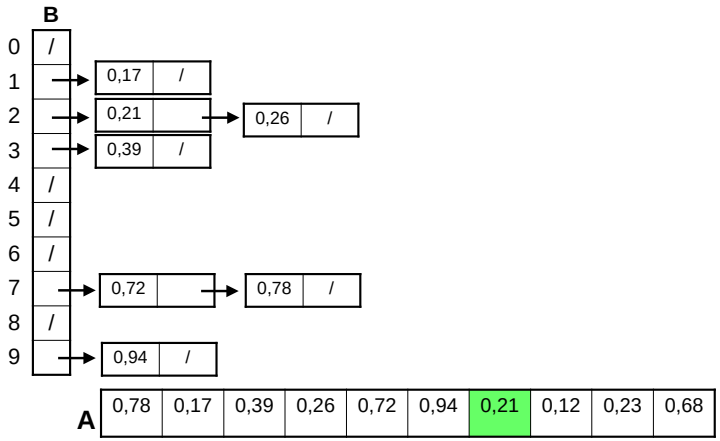
# Funcionamento



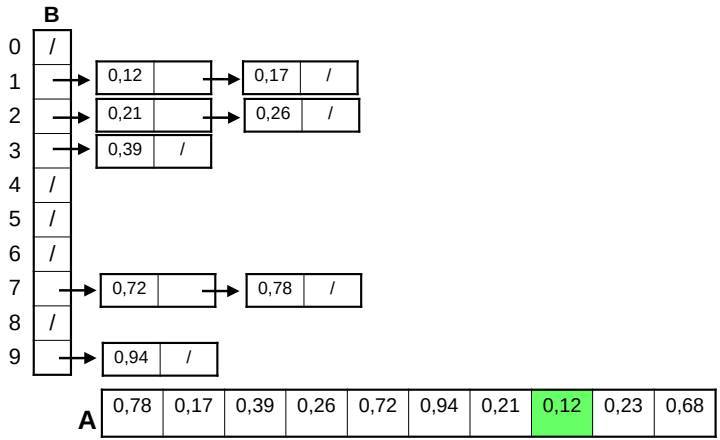
# Funcionamento



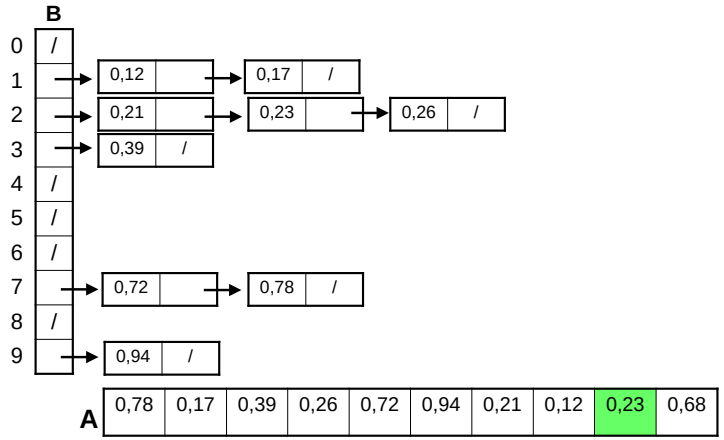
# Funcionamento



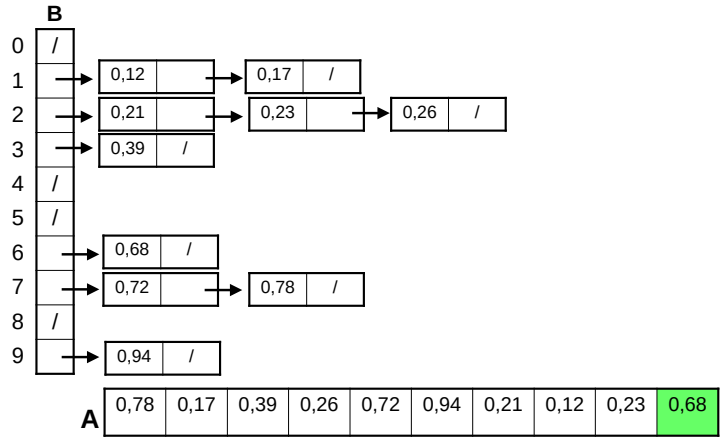
# Funcionamento



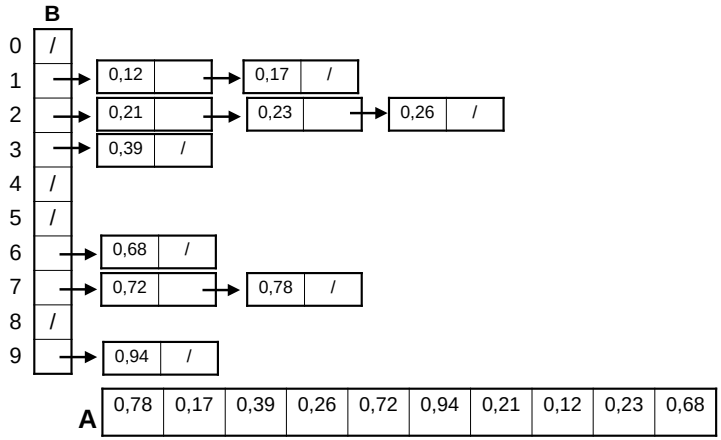
# Funcionamento



# Funcionamento



# Funcionamento



## Conteúdo

## Conclusão

- Foram vistos três algoritmos de ordenação linear (tempo $\Theta(n)$). Que são então melhores que os algoritmos de ordenação por comparação (tempo $O(n \lg n)$);
- Entretanto, nem sempre é interessante utilizar um destes três algoritmos:
  - Todos eles pressupõem algo sobre os dados de entrada a serem ordenados.

*Pesquisa Sequencial e Pesquisa Binária*

## Conteúdo

## Bibliografia

Os conteúdos deste material, incluindo figuras, textos e códigos, foram extraídos ou adaptados de:


Fernandes, Cristina G.
*MAC0338 - Ordenação em tempo linear*.
`https://www.ime.usp.br/~cris/aulas/11_1_338/slides/aula10.pdf`.
Acessado em 2021.


Cormen, Thomas H. and Leiserson, Charles E. and Rivest, Ronald L. and Stein, Clifford.
*Introduction to Algorithms*.
The MIT Press, 2011.

## Exercício

- Dada a sequência de números: `3 4 0 2 5 1 3`.
- Ordene em ordem crescente utilizando o algoritmo aprendido em sala (**Couting Sort**), apresentando a sequência dos números a cada passo (Teste de Mesa).