Ordenação em Tempo Linear

Prof. Túlio Toffolo
http://www.toffolo.com.br

BCC202 – Aula 19
Algoritmos e Estruturas de Dados I

Ordenação em tempo linear

- Algoritmos de ordenação por **comparação**:
  - *InsertSort*;
  - *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 possua características especiais;
  - Algumas restrições sejam respeitadas;
  - O algoritmo não seja puramente baseado em comparações;
  - A implementação seja feita da maneira adequada.
- Tempo linear: $\Theta(n)$;
- Algoritmos:
  - Ordenação por contagem (*Counting sort*);
  - *Radix sort*;
  - *Bucket sort*.

ORDENAÇÃO EM TEMPO LINEAR

ORDENAÇÃO POR CONTAGEM

Ordenação por contagem

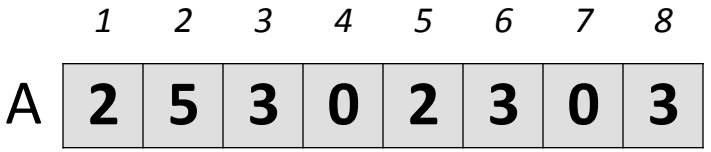
- Pressupõe que cada elemento da entrada é um inteiro na faixa de *0* a *k*, para algum inteiro *k*;
- Idéia básica:
  - Determinar para cada elemento da entrada *x* o número de elementos maiores que *x*;
  - Com esta informação, determinar a posição de cada elemento
    - Ex.: Se 17 elementos forem menores que *x* então *x* ocupa a posição de saída 18.

Ordenação por contagem

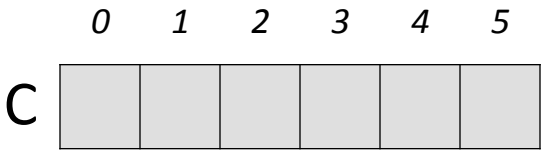
- Algoritmo:
  - Assumimos que o vetor de entrada é *A[1,...,n];*
  - Outros dois vetores são utilizados:
    - *B[1,...,n]* – armazena a saída ordenada;
    - *C[1,...,k]* – é utilizado para armazenamento temporário.

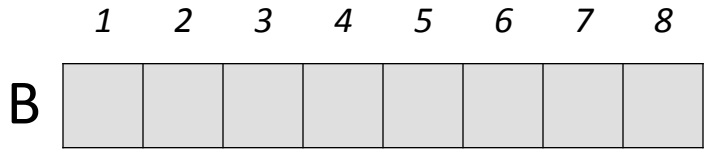
Ordenação por contagem

```
1 for i ← 0 to k
2       do C[i] ← 0
3 for j ← 1 to length[A]
4       do C[A[j]] ← C[A[j]] + 1
5 for i ← 1 to k
6       do C[i] ← C[i] + C[i −1]
7 for j ← length[A] down to 1
8       do B[C[A[j]]] ← A[j]
9          C[A[j]] ← C[A[j]] −1
```



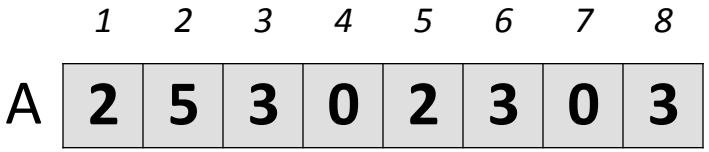
k = 5



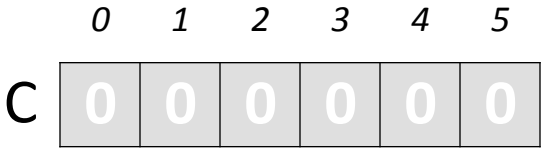


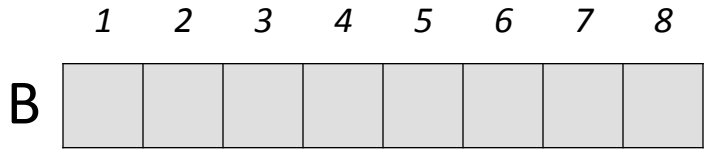
Ordenação por contagem

```
1 for i ← 0 to k
2       do C[i] ← 0
3 for j ← 1 to length[A]
4       do C[A[j]] ← C[A[j]] + 1
5 for i ← 1 to k
6       do C[i] ← C[i] + C[i −1]
7 for j ← length[A] down to 1
8       do B[C[A[j]]] ← A[j]
9          C[A[j]] ← C[A[j]] −1
```



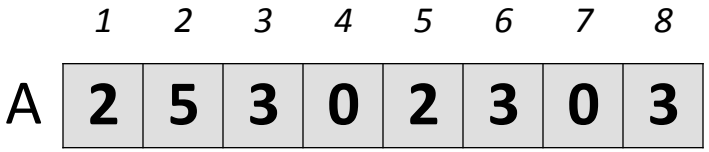
k = 5



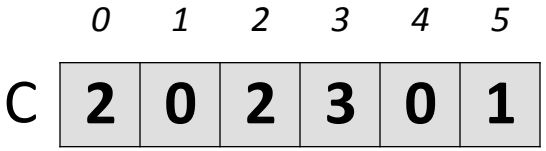


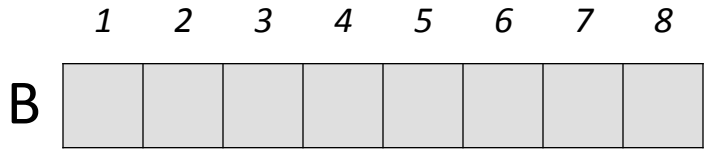
Ordenação por contagem

```
1 for i ← 0 to k
2       do C[i] ← 0
3 for j ← 1 to length[A]
4       do C[A[j]] ← C[A[j]] + 1
5 for i ← 1 to k
6       do C[i] ← C[i] + C[i −1]
7 for j ← length[A] down to 1
8       do B[C[A[j]]] ← A[j]
9          C[A[j]] ← C[A[j]] −1
```



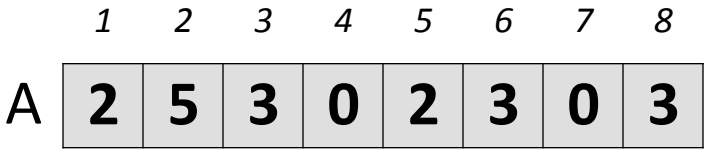
k = 5



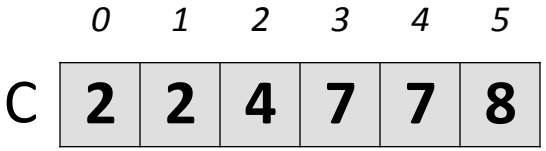


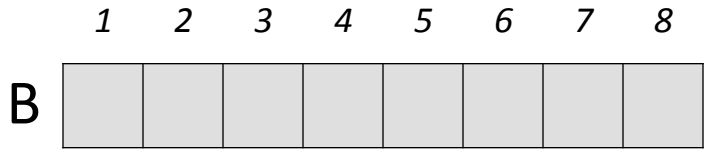
Ordenação por contagem

```
1 for i ← 0 to k
2       do C[i] ← 0
3 for j ← 1 to length[A]
4       do C[A[j]] ← C[A[j]] + 1
5 for i ← 1 to k
6       do C[i] ← C[i] + C[i −1]
7 for j ← length[A] downto 1
8       do B[C[A[j]]] ← A[j]
9          C[A[j]] ← C[A[j]] −1
```



k = 5





Ordenação por contagem

```
1 for i ← 0 to k
2       do C[i] ← 0
3 for j ← 1 to length[A]
4       do C[A[j]] ← C[A[j]] + 1
5 for i ← 1 to k
6       do C[i] ← C[i] + C[i −1]
7 for j ← length[A] down to 1
8       do B[C[A[j]]] ← A[j]
9          C[A[j]] ← C[A[j]] −1
```

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

k = 5

	0	1	2	3	4	5
C	0	2	2	4	7	7

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

Ordenação por contagem

- 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

ORDENAÇÃO EM TEMPO LINEAR

RADIXSORT

Radix Sort

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

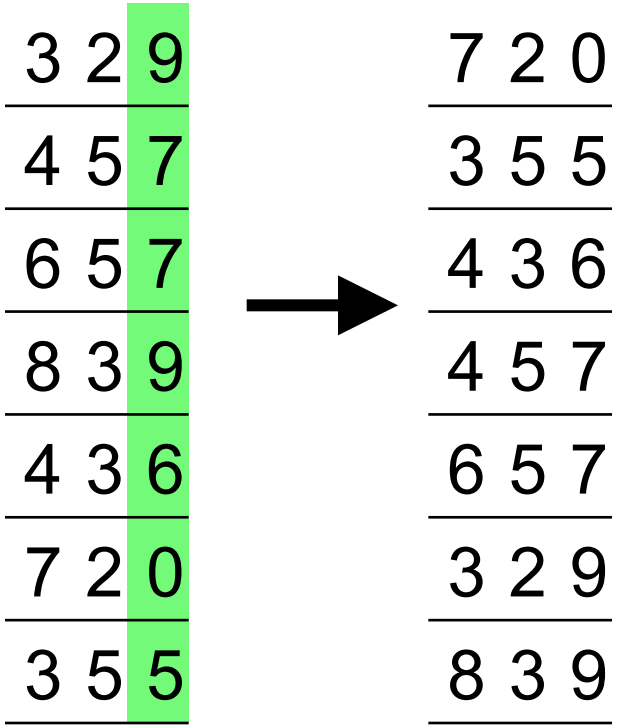
Radix Sort – Funcionamento

- 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

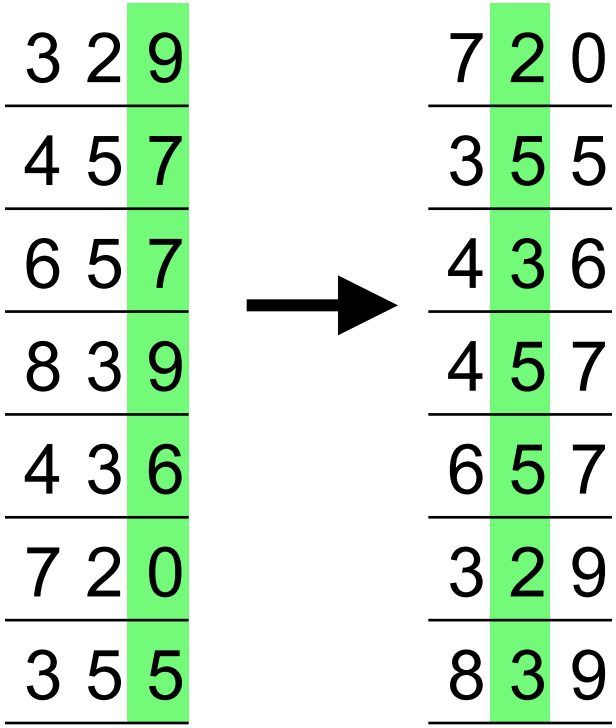
Radix Sort – Funcionamento

- A partir dos dígitos menos significativos:



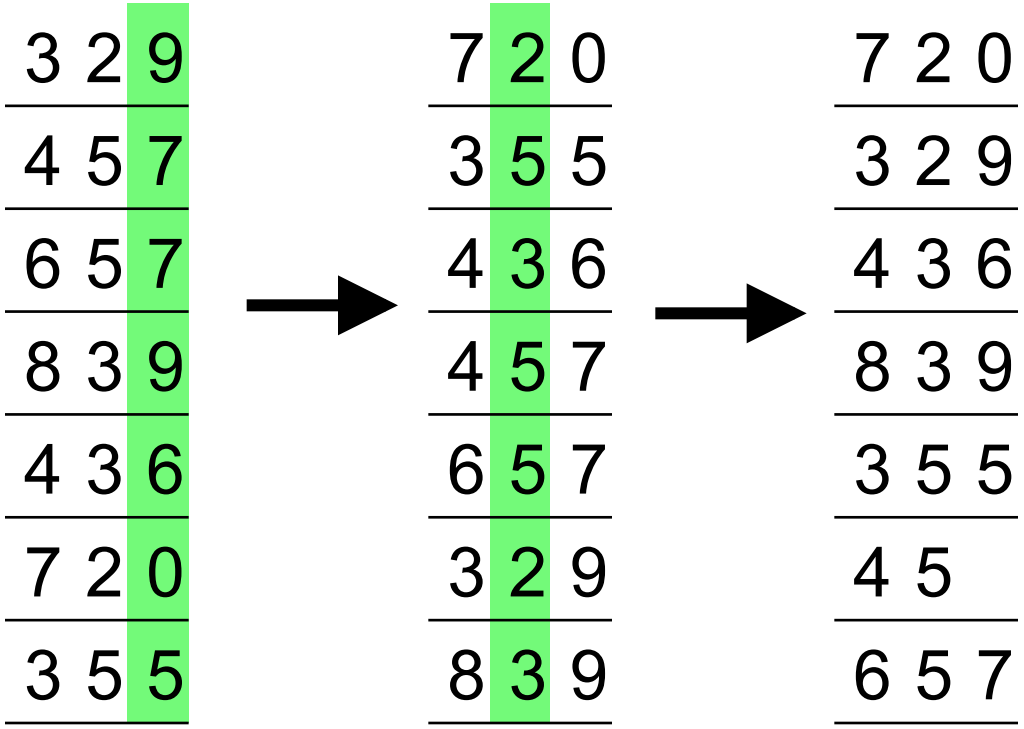
Radix Sort – Funcionamento

- A partir dos dígitos menos significativos:



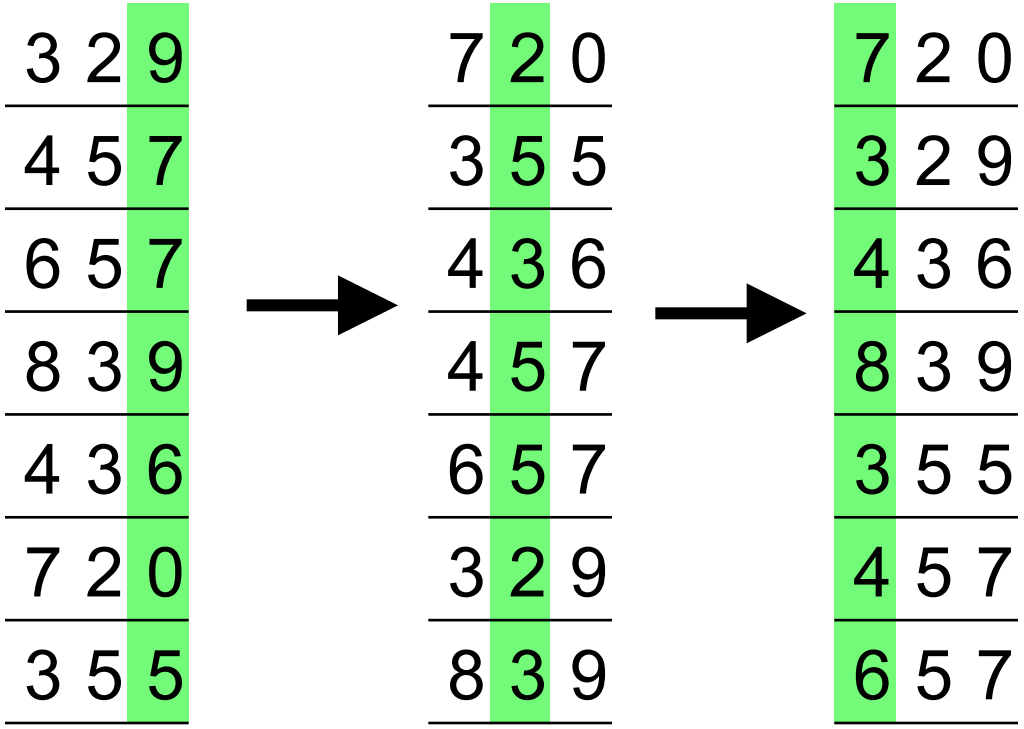
Radix Sort – Funcionamento

- A partir dos dígitos menos significativos:



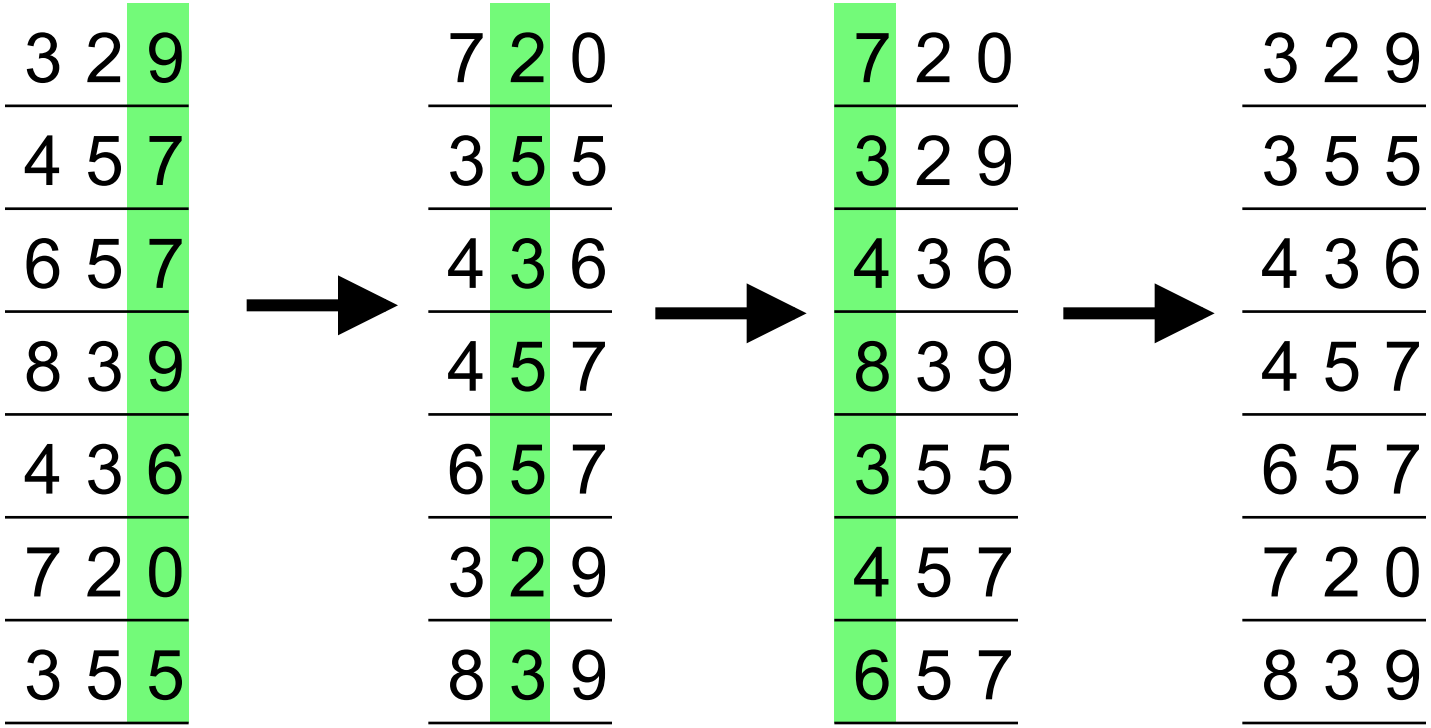
Radix Sort – Funcionamento

- A partir dos dígitos menos significativos:



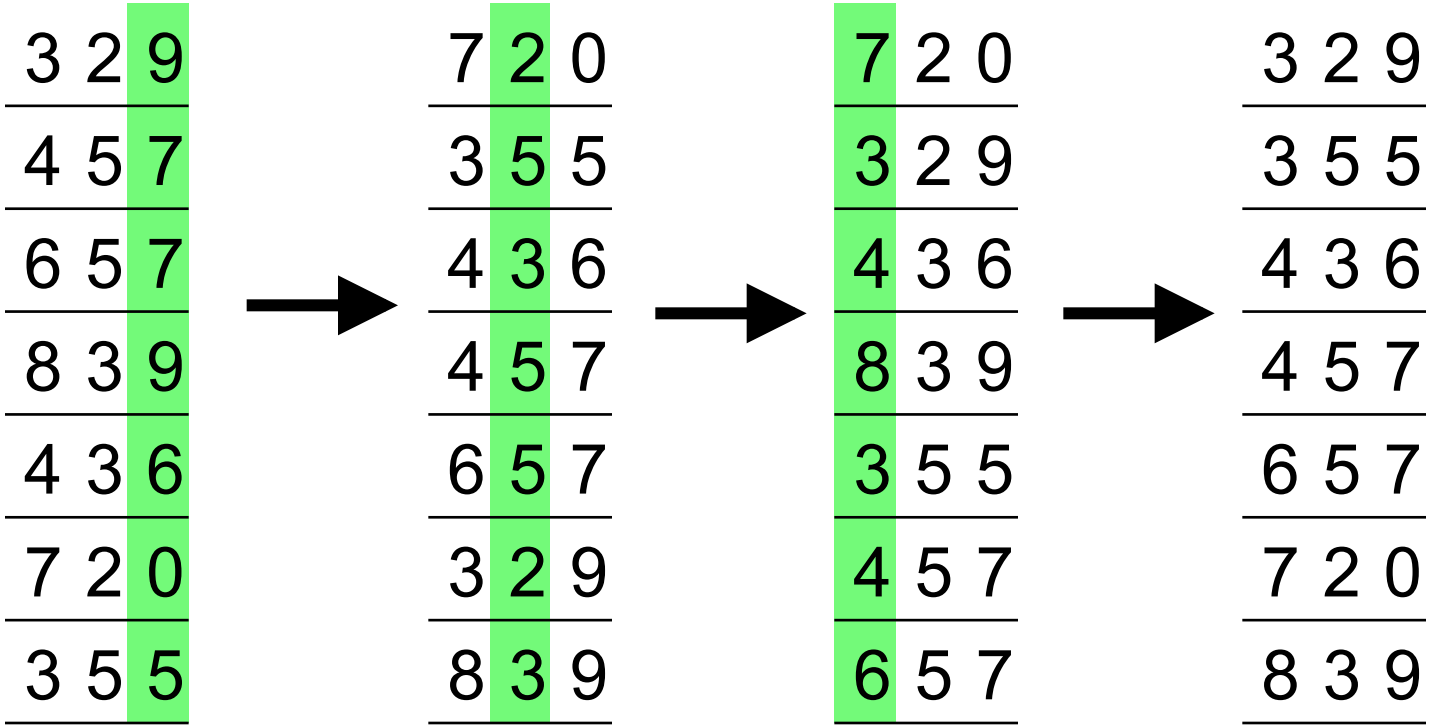
Radix Sort – Funcionamento

- A partir dos dígitos menos significativos:



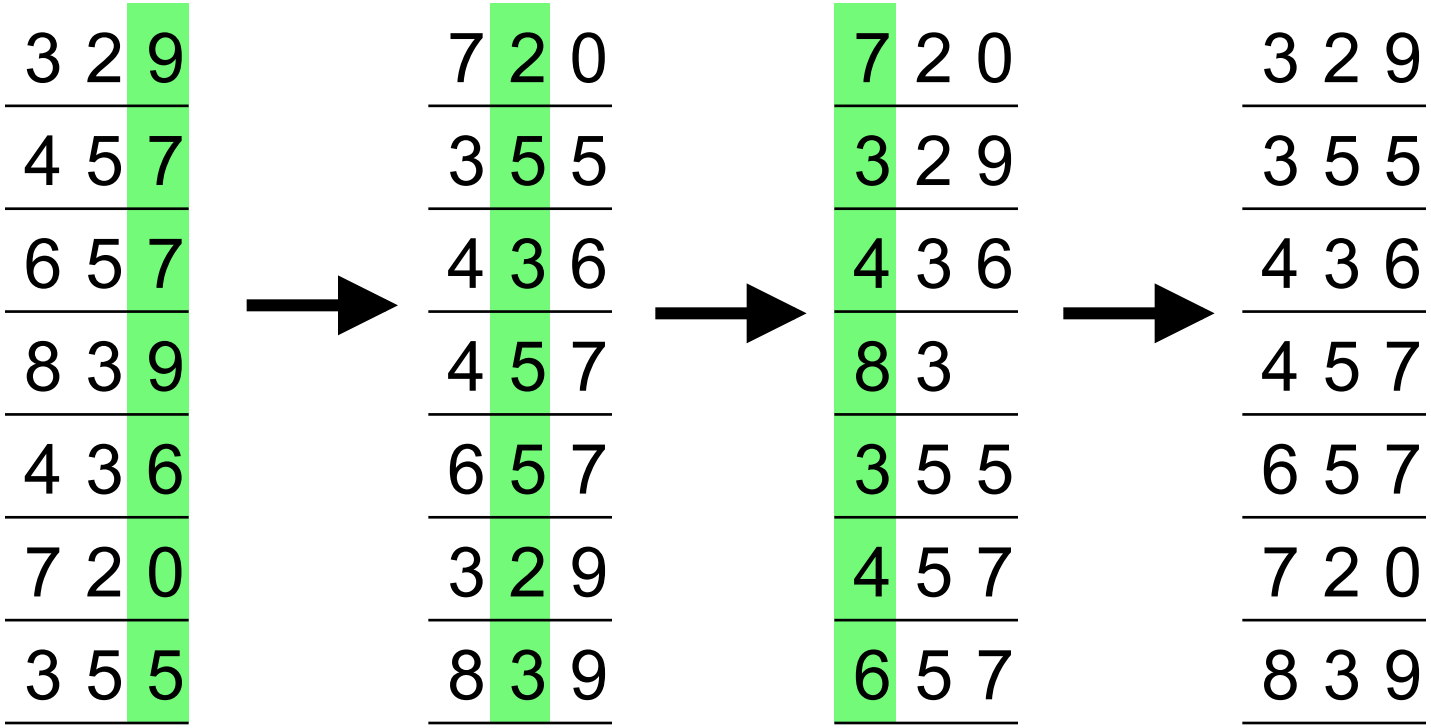
Radix Sort – Funcionamento

- A partir dos dígitos menos significativos:
  Como ficaria a partir o dígito mais significativo?



Radix Sort – Funcionamento

- A partir dos dígitos menos significativos:
  E se a ordenação não fosse estável?



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  for i ← 1 to d
2        utilize um algoritmo estável para ordenar o
         array A pelo i-ésimo dígito
```

- Onde:
  - *d* é número de dígitos;
  - *A* é o *array* de entrada.

ORDENAÇÃO EM TEMPO LINEAR

BUCKETSORT

Bucket Sort

- Assume que a entrada consiste em elementos distribuídos de forma **uniforme** sobre o intervalo *[0,1)*;
- A idéia 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*;

Bucket Sort

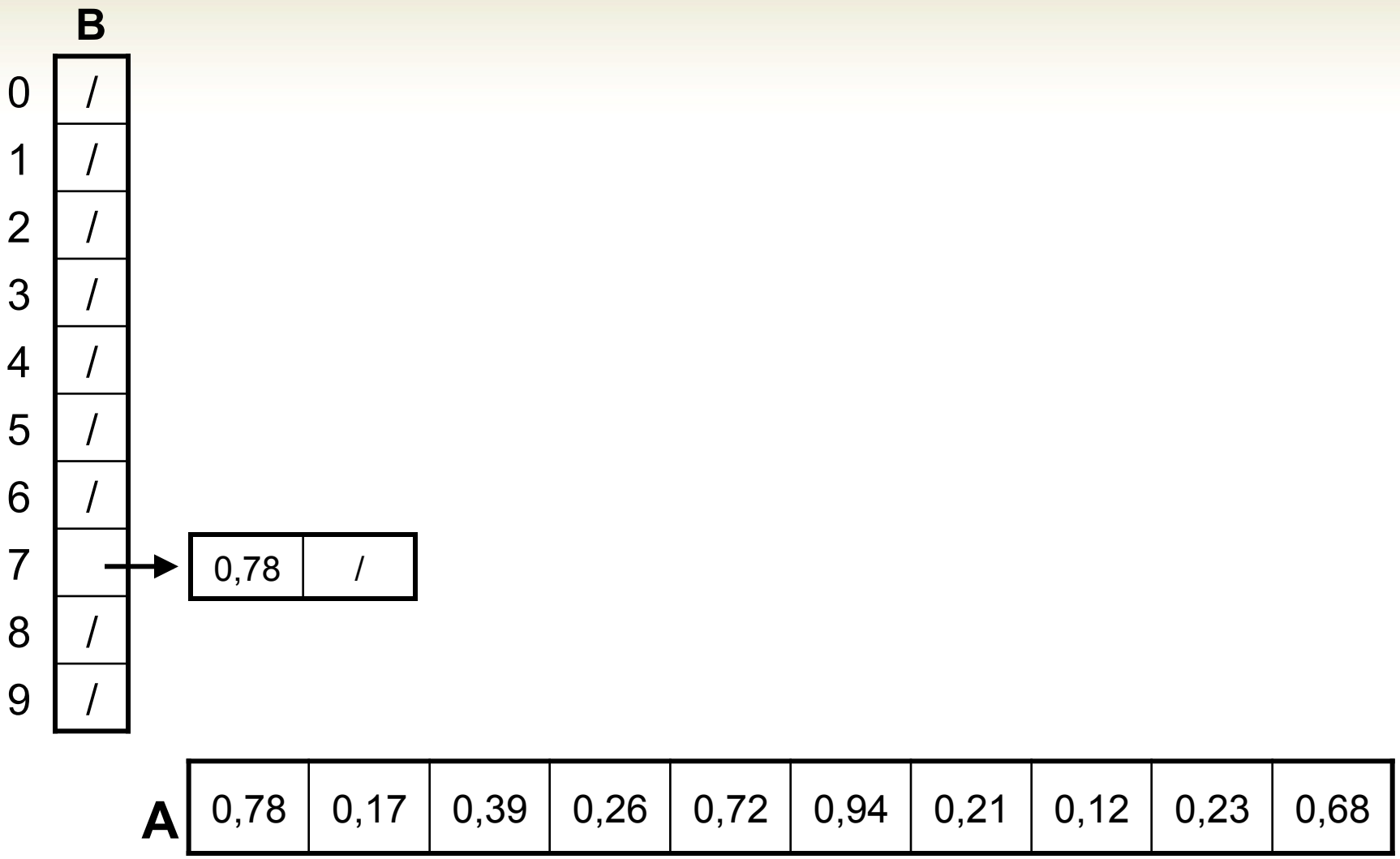
- Para produzir a saída ordenada, basta ordenar os números em cada *balde*, e depois examinar os *baldes* em ordem, listando seus elementos;
- A função para determinação do índice do *balde* correto é $\lfloor n \times A[i] \rfloor$;
- Vamos a um exemplo com 10 números
  - *A* é o *array* de entrada;
  - *B* é o *array* com os baldes.

Bucket Sort – Funcionamento

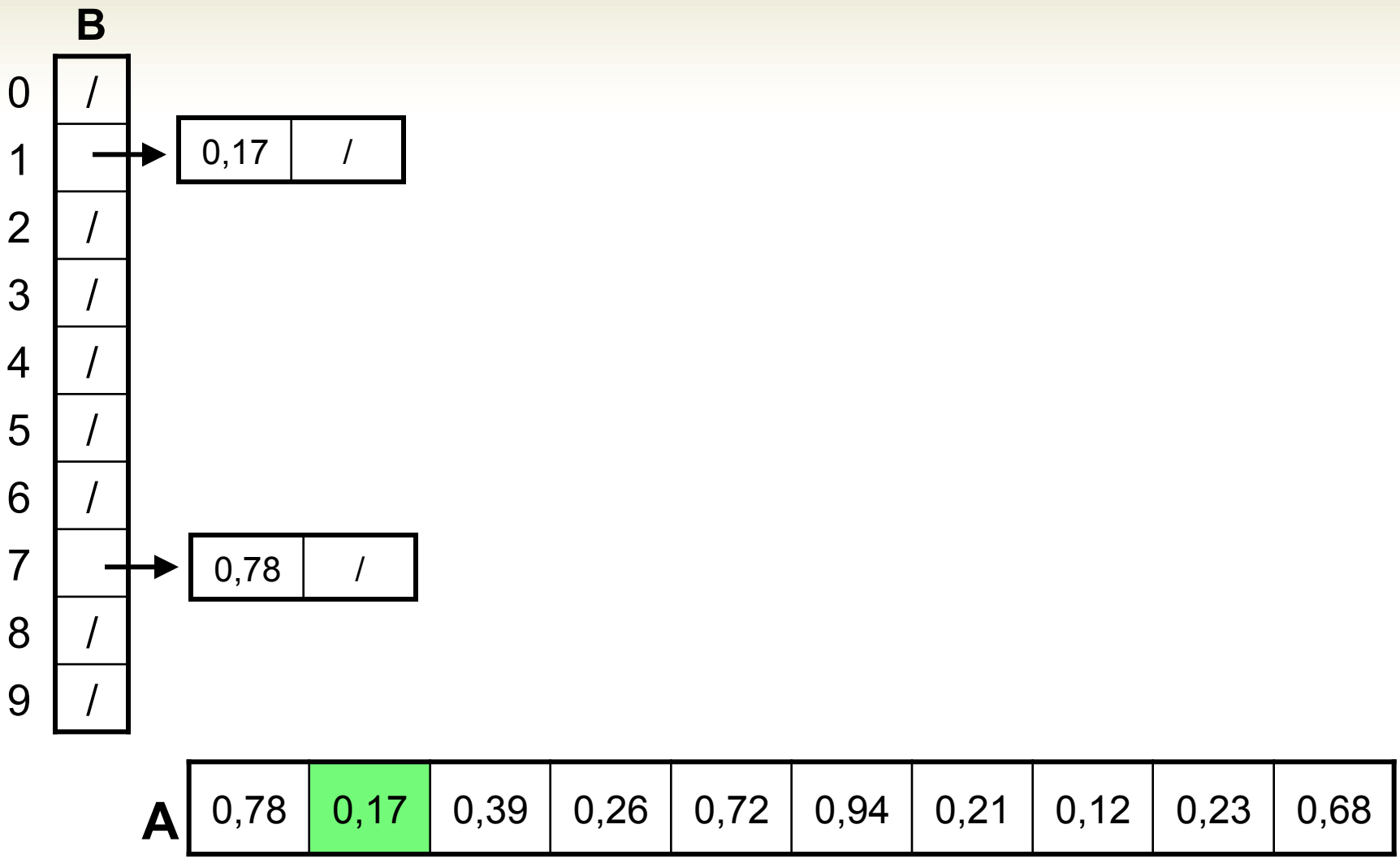
	B
0	/
1	/
2	/
3	/
4	/
5	/
6	/
7	/
8	/
9	/

A										
	0,78	0,17	0,39	0,26	0,72	0,94	0,21	0,12	0,23	0,68

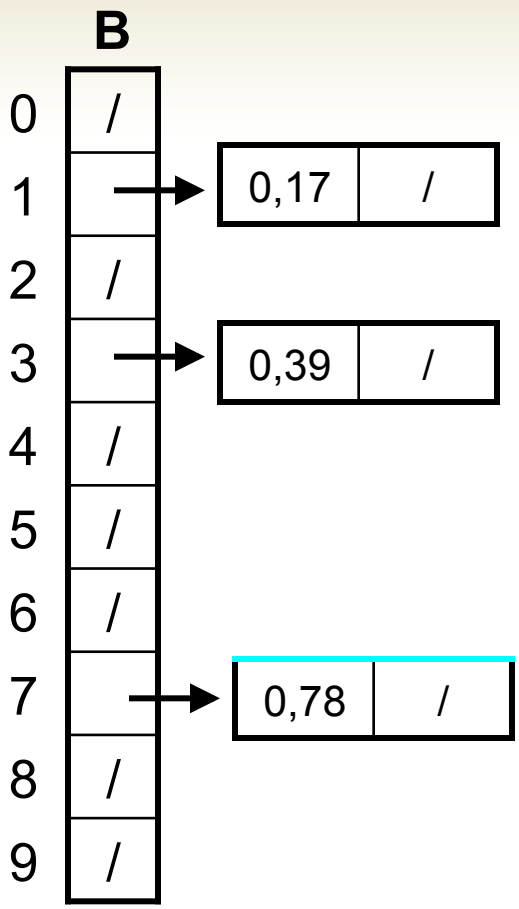
Bucket Sort – Funcionamento



Bucket Sort – Funcionamento

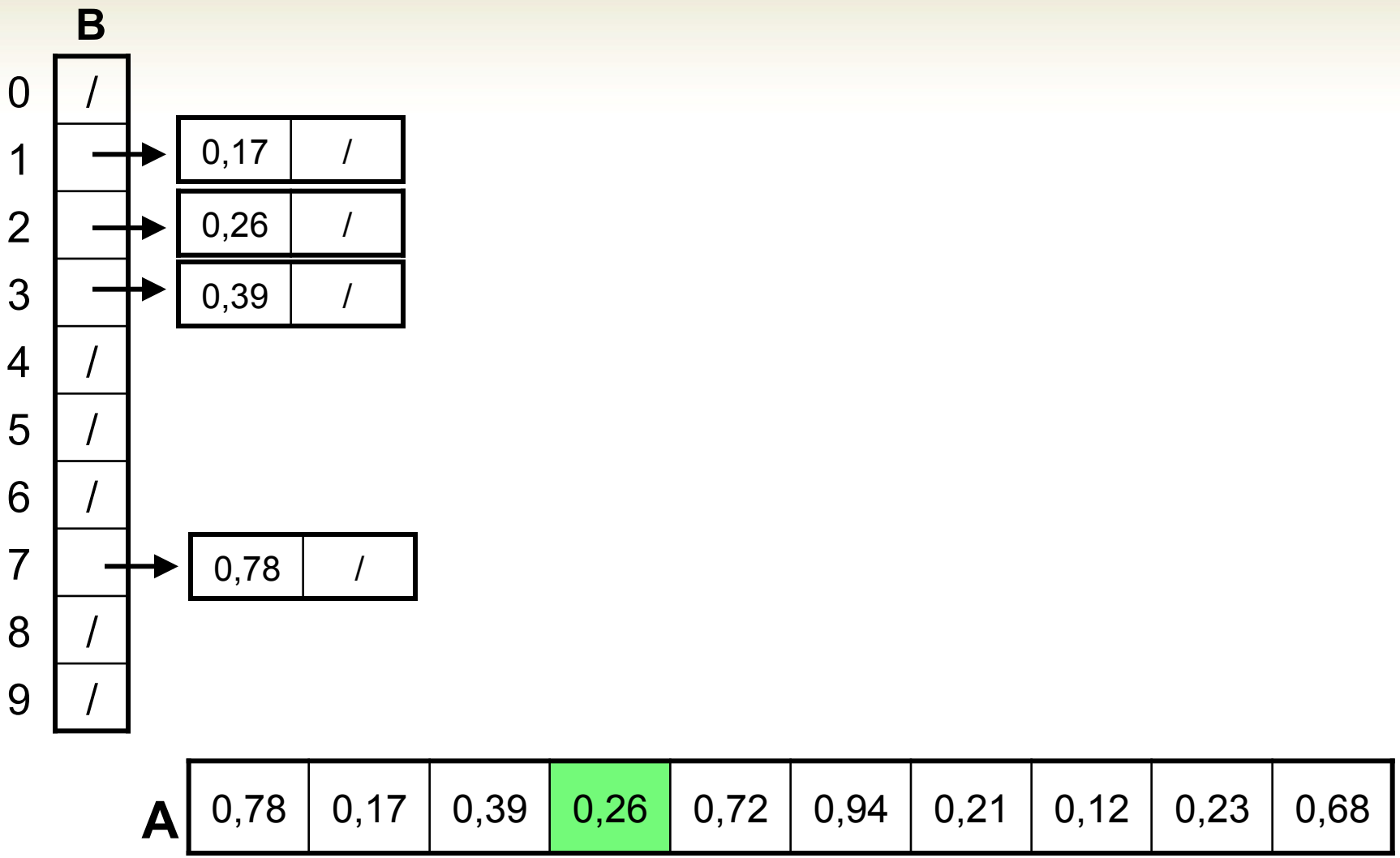


Bucket Sort – Funcionamento

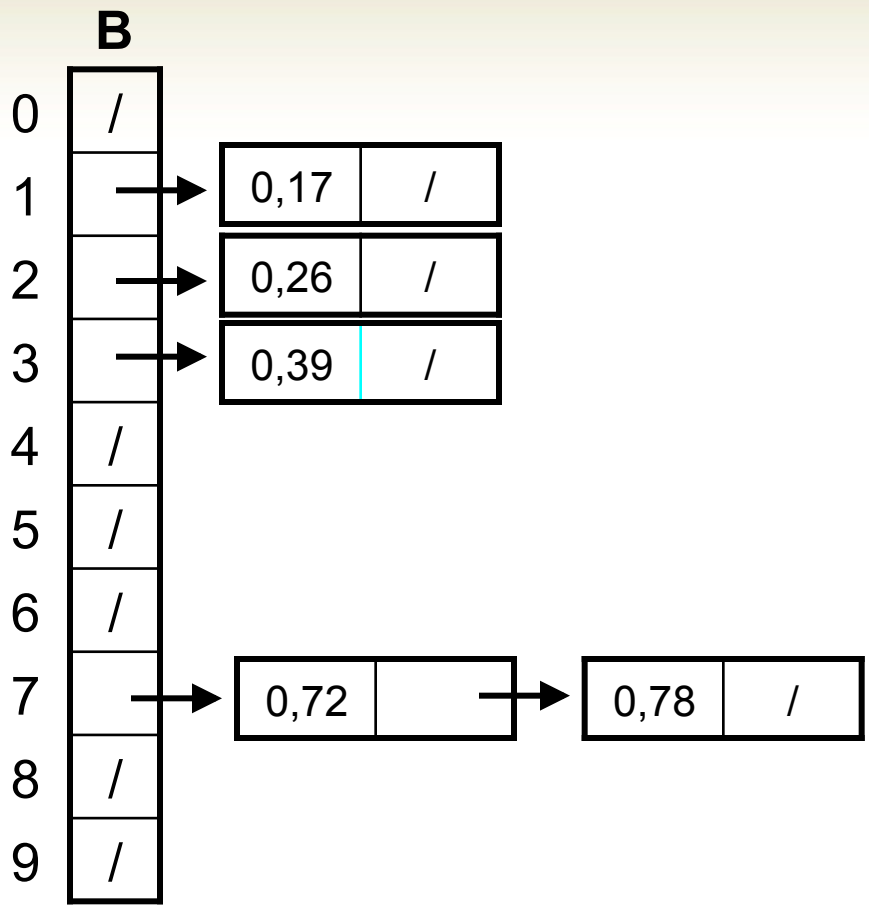


A	0,78	0,17	0,39	0,26	0,72	0,94	0,21	0,12	0,23	0,68

Bucket Sort – Funcionamento

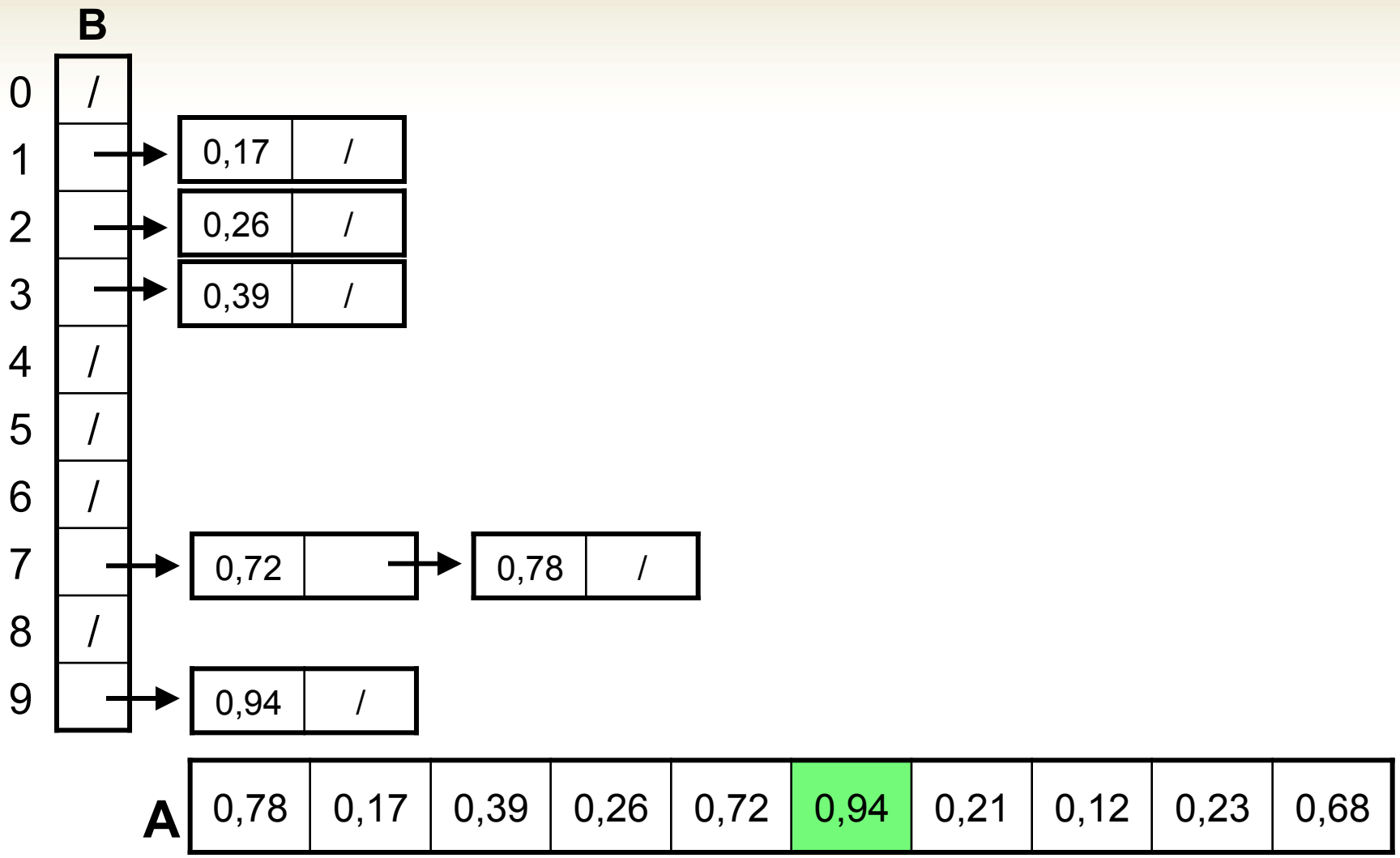


Bucket Sort – Funcionamento

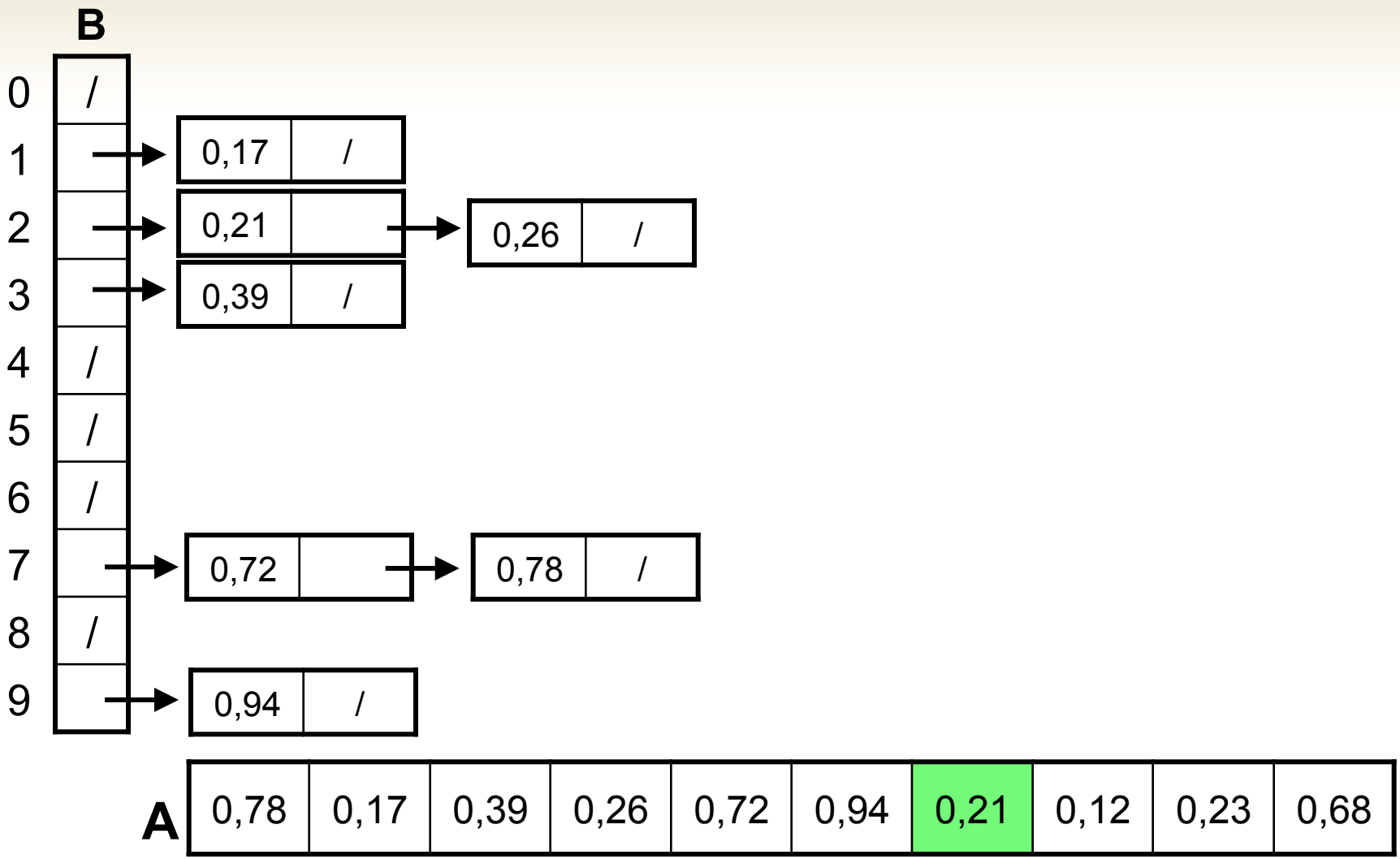


A	0,78	0,17	0,39	0,26	0,72	0,94	0,21	0,12	0,23	0,68

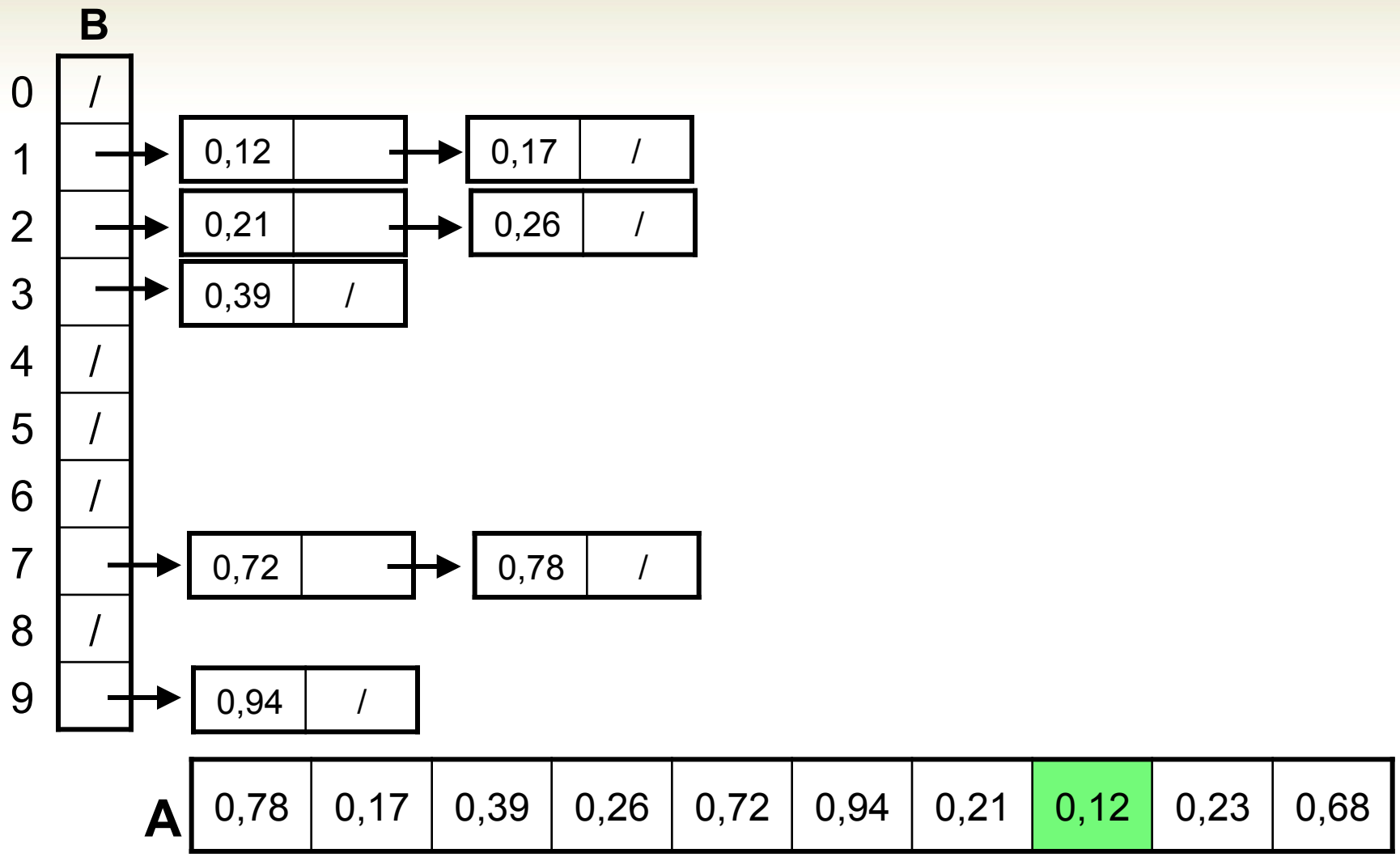
Bucket Sort – Funcionamento



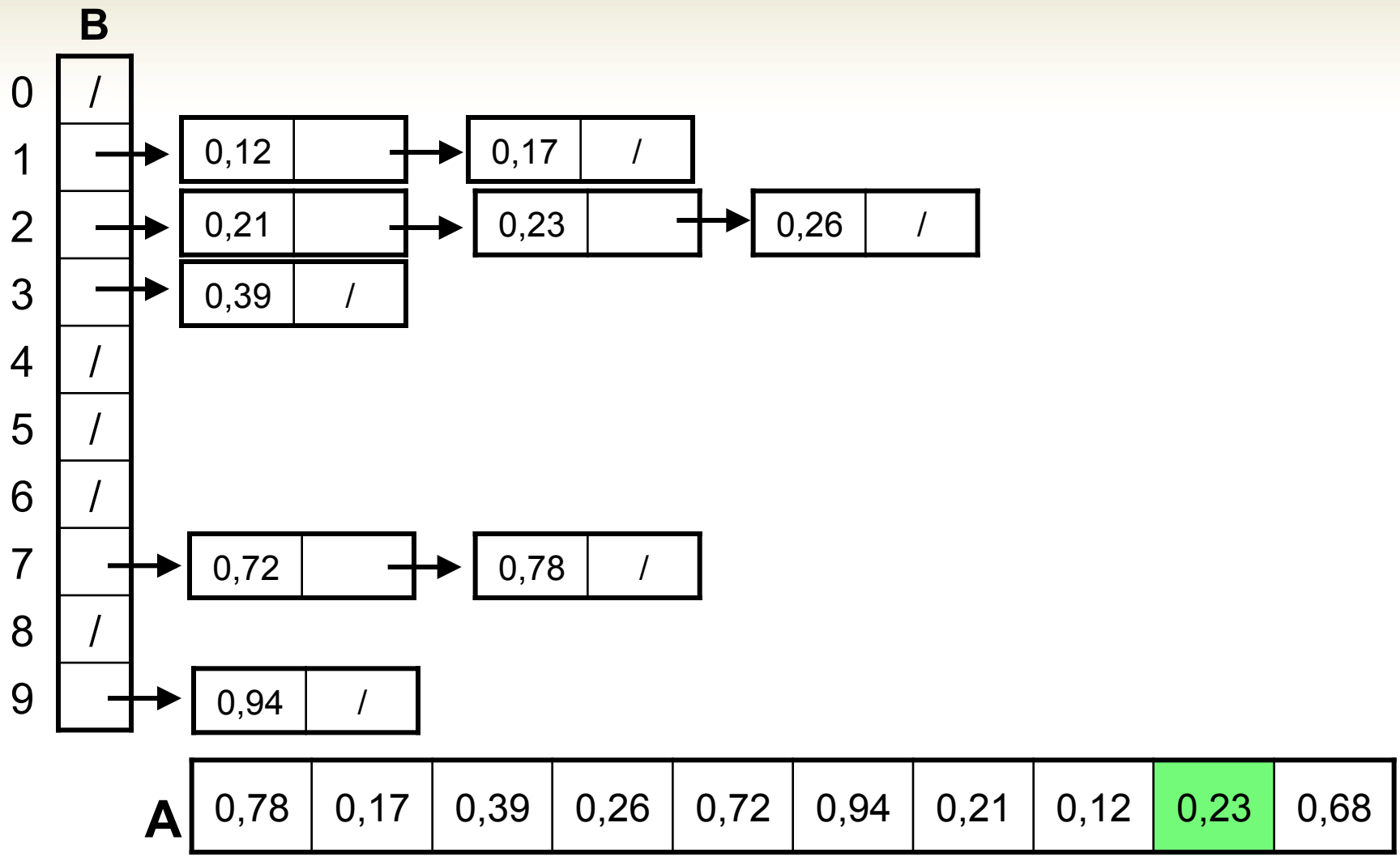
Bucket Sort – Funcionamento



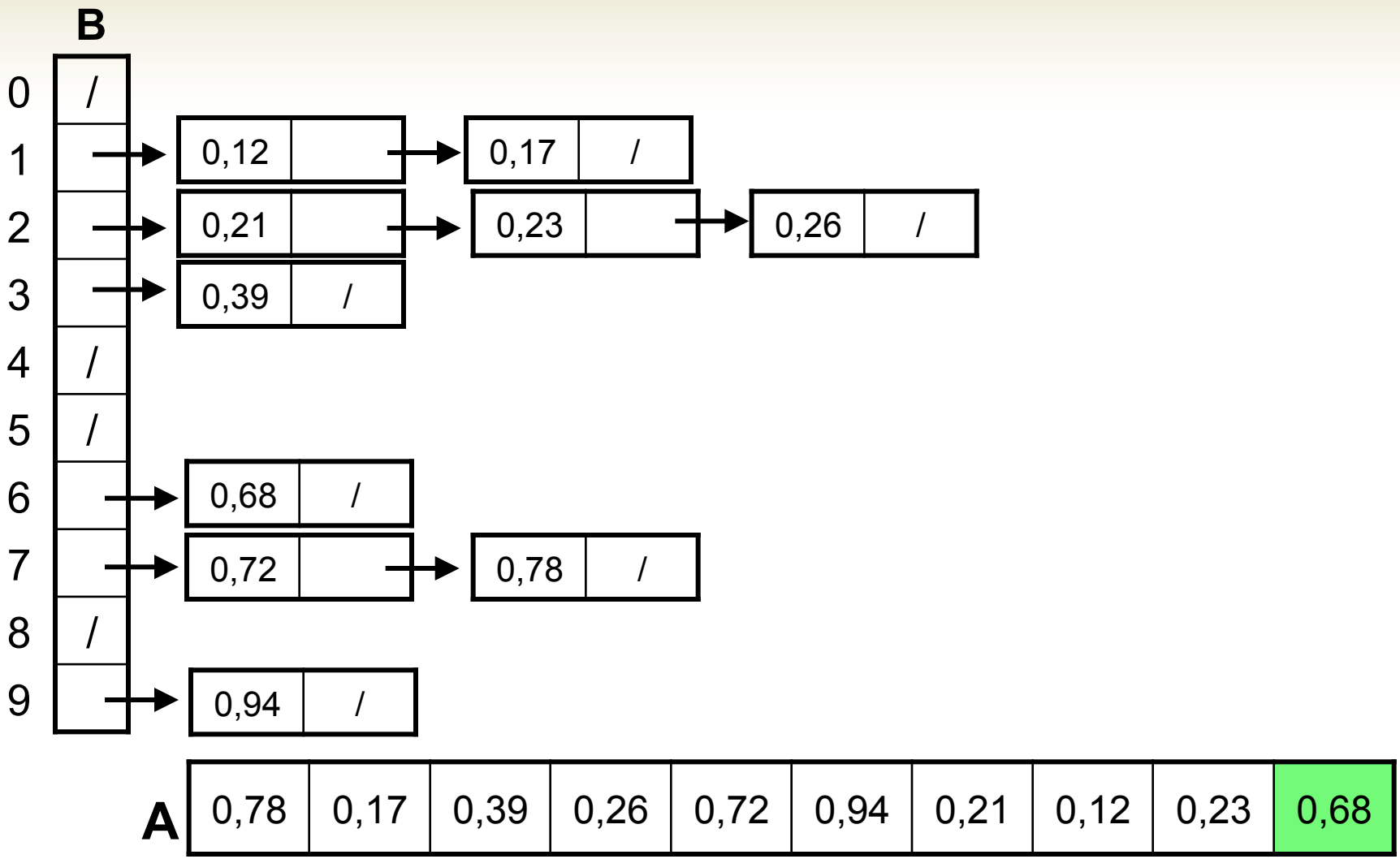
Bucket Sort – Funcionamento



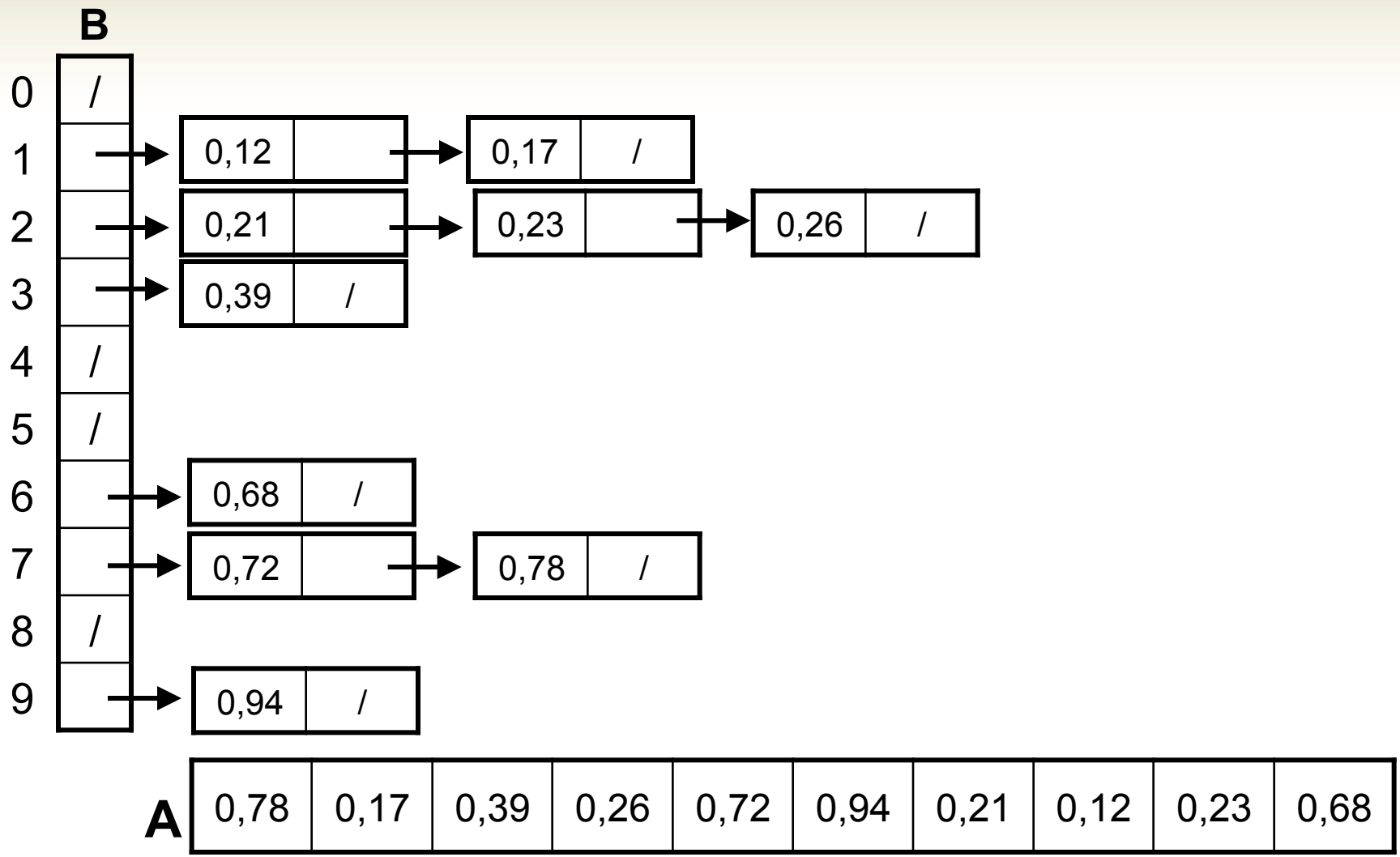
Bucket Sort – Funcionamento



Bucket Sort – Funcionamento



Bucket Sort – Funcionamento



ORDENAÇÃO EM TEMPO LINEAR

CONCLUSÕES

Ordenação em tempo linear

- 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.

Perguntas?