

# **Ordenação Interna**

## **Algoritmos de ordem Linear**

Disciplina: Algoritmos e Estruturas de Dados II

Álvaro Luiz Fazenda

[alvaro.fazenda@unifesp.br](mailto:alvaro.fazenda@unifesp.br)

# Bibliografia básica

- Algoritmos – Teoria e Prática
  - Thomas H. Cormen et al – Elsevier – 2002
    - Seções 8.2 a 8.4

# Ordenação em Tempo Linear

- Possível apenas em casos particulares

# *CountingSort*

## Ordenação por contagem

- Pressupõe que cada um dos  $n$  elementos de entrada deve ser um inteiro no intervalo  $0..k$ , para algum inteiro  $k$ 
  - Quando  $k=O(n)$  a ordenação é executada em  $O(n)$

# *CountingSort*

## Algoritmo

```
COUNTING-SORT (A, B, k) //input: A, k ; output: B
1) n ← lenght [A]
2) for i ← 0 to k
3)     C[i] ← 0
4) for j ← 1 to n
5)     C[A[j]] ← C[A[j]] + 1 //núm.elem.de tamanho i
6) for i ← 1 to k
7)     C[i] ← C[i] + C[i-1] //núm.elem. <= i
8) for j ← n downto 1
9)     B[C[A[j]]] ← A[j]
10)    C[A[j]] ← C[A[j]] - 1
```

# **CountingSort - Algoritmo (II)**

- Primeiro laço zera vetor auxiliar  $C$ 
  - $\Theta(k)$
- Segundo laço incrementa  $C[i]$  para cada valor  $i$  em  $A[i]$ 
  - $\Theta(n)$
- Terceiro laço determina quantos valores são menores que  $i$  para cada elemento  $A[i]$ 
  - $\Theta(k)$
- Quarto laço põe os valores  $A[j]$  em sua posição correta ordenada no vetor  $B$ 
  - $\Theta(n)$
- $\Theta(k + n)$ , sendo para  $k=O(n) \Rightarrow \Theta(n)$

# CoutingSort

## Exemplo

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

(a)

	0	1	2	3	4	5
C	2	2	4	7	7	8

(b)

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

(c)

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

(d)

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

(e)

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

(f)

- $k=5$
- (a) arranjos  $A$  e  $C$  após segundo laço
- (b) arranjo  $C$  após terceiro laço
- (c)-(e) arranjos  $B$  e  $C$  após uma, duas e três iterações do quarto laço
- (f) arranjo final  $B$

# Método Contagem

	1									n
A	2	5	3	0	2	3	0	5	3	0

# Método Contagem

	1									n
A	2	5	3	0	2	3	0	5	3	0

	1									n
B										

	0					k
C						

# Método Contagem

	1										n
A	2	5	3	0	2	3	0	5	3	0	

	1										n
B											

	0					k
C	0	0	0	0	0	0

# Método Contagem

A  $j$

2	5	3	0	2	3	0	5	3	0
---	---	---	---	---	---	---	---	---	---

B  $1$   $n$

--	--	--	--	--	--	--	--	--	--

C  $0$   $k$

0	0	0	0	0	0
---	---	---	---	---	---

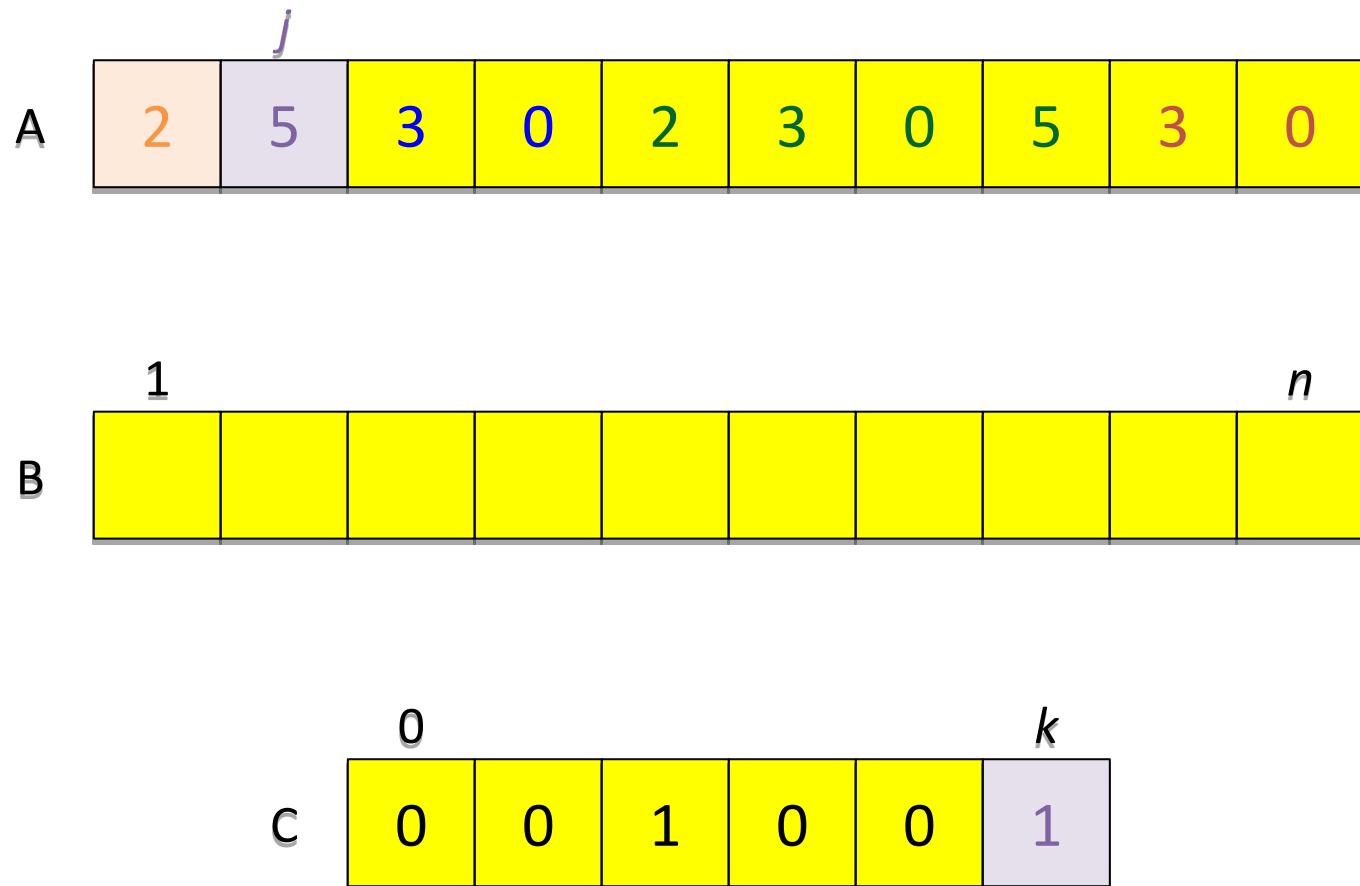
# Método Contagem

A	$j$	2	5	3	0	2	3	0	5	3	0
---	-----	---	---	---	---	---	---	---	---	---	---

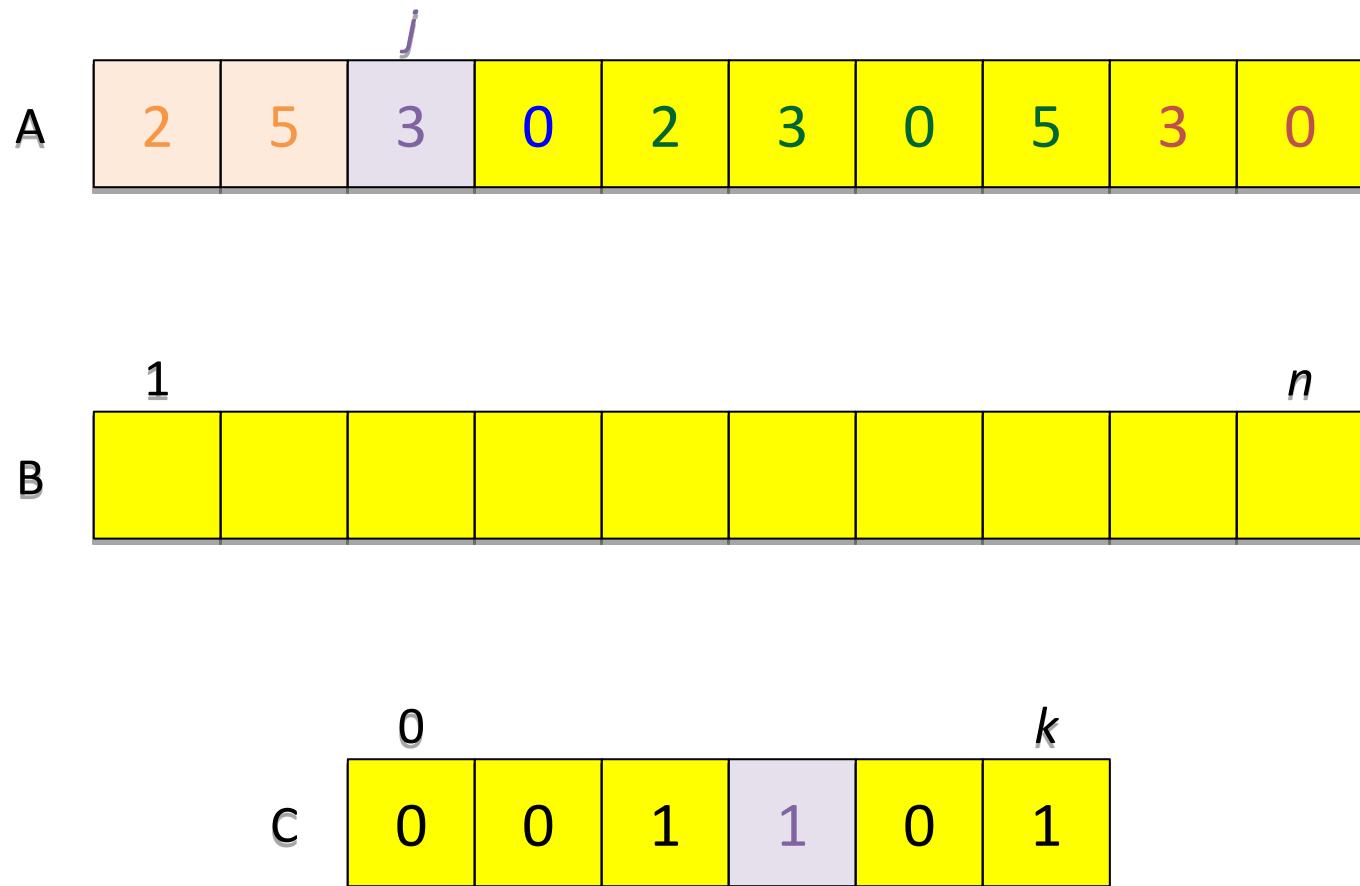
B	1										$n$
---	---	--	--	--	--	--	--	--	--	--	-----

C	0	0	1	0	0	0
---	---	---	---	---	---	---

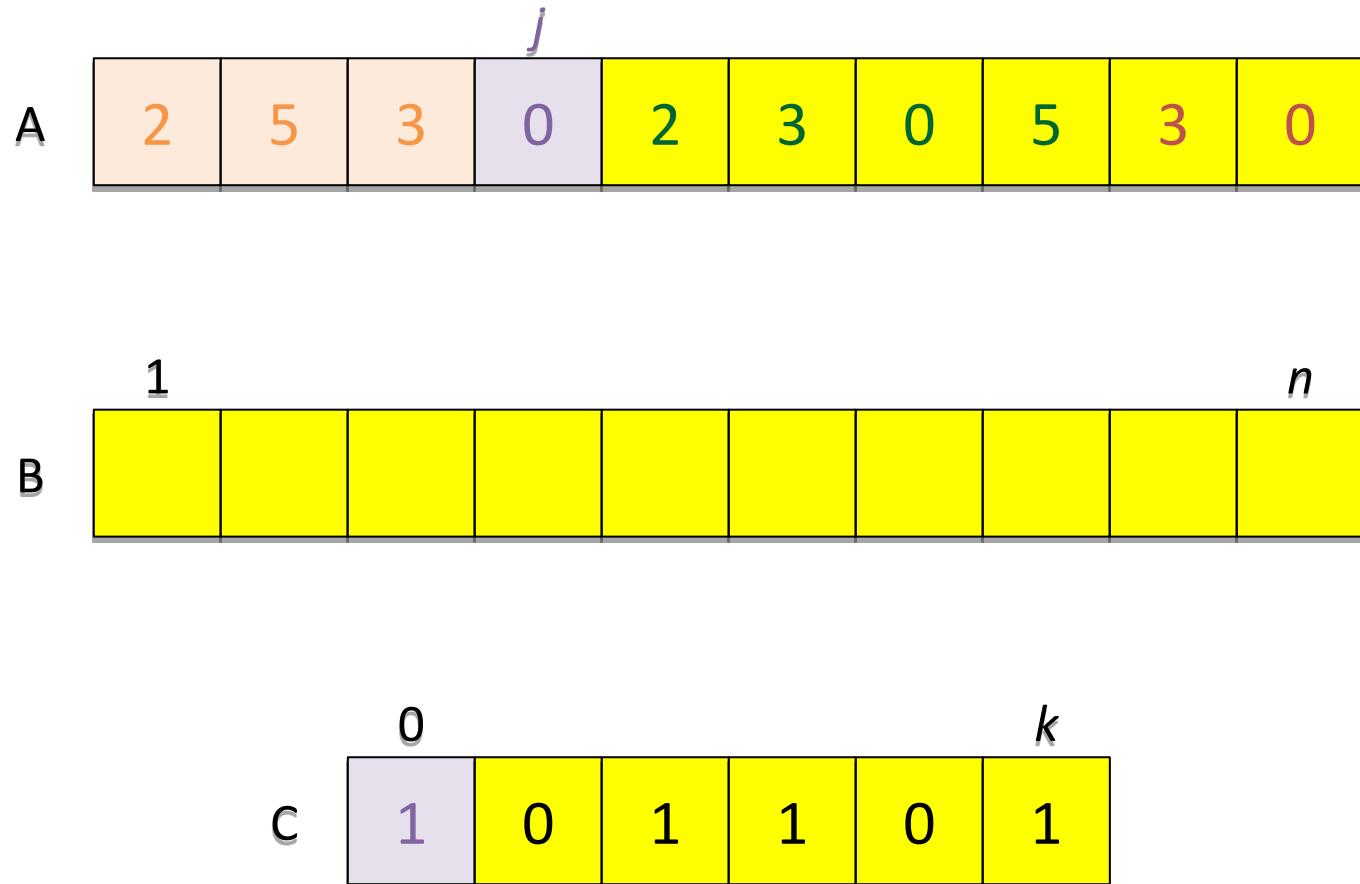
# Método Contagem



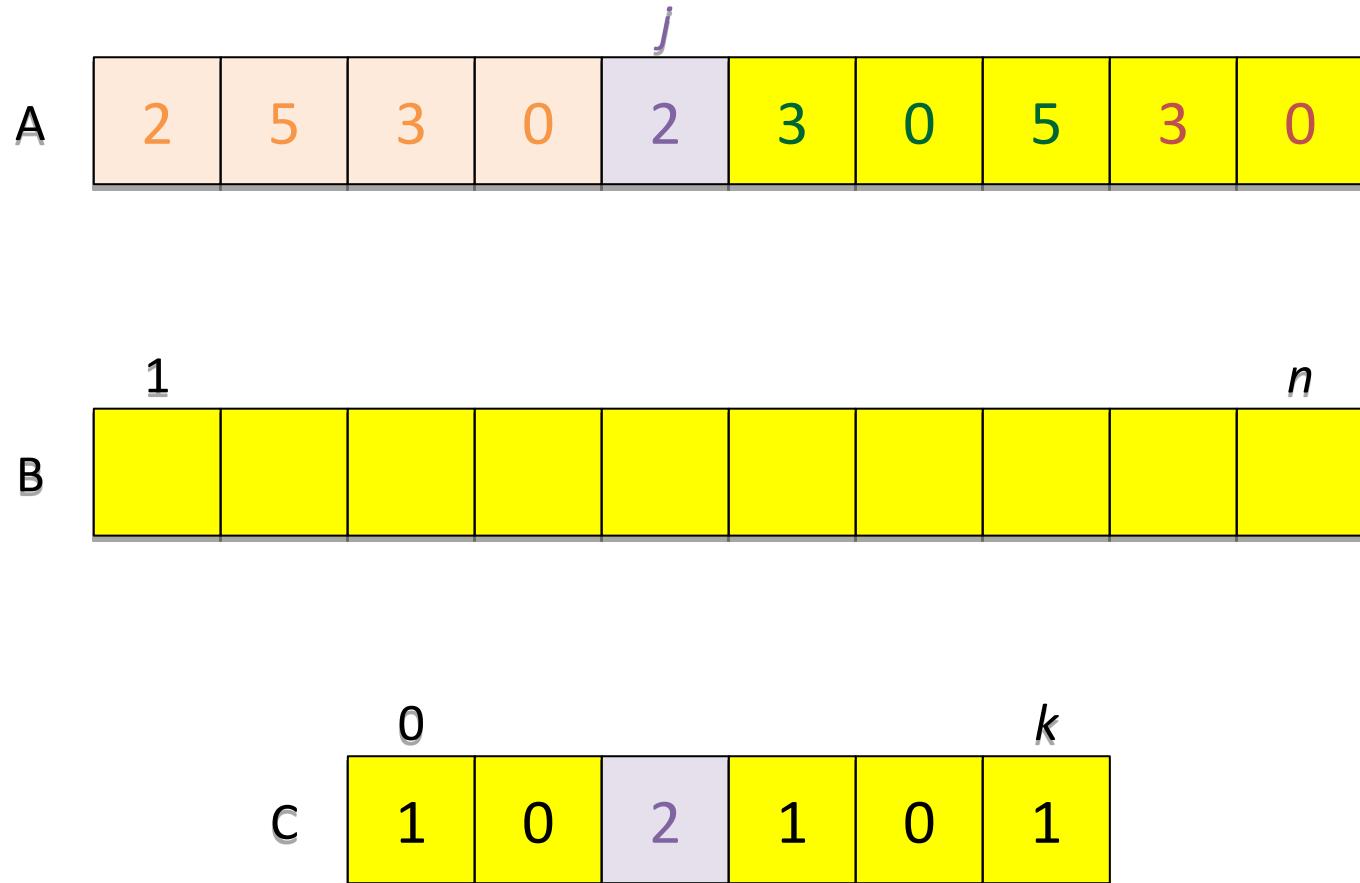
# Método Contagem



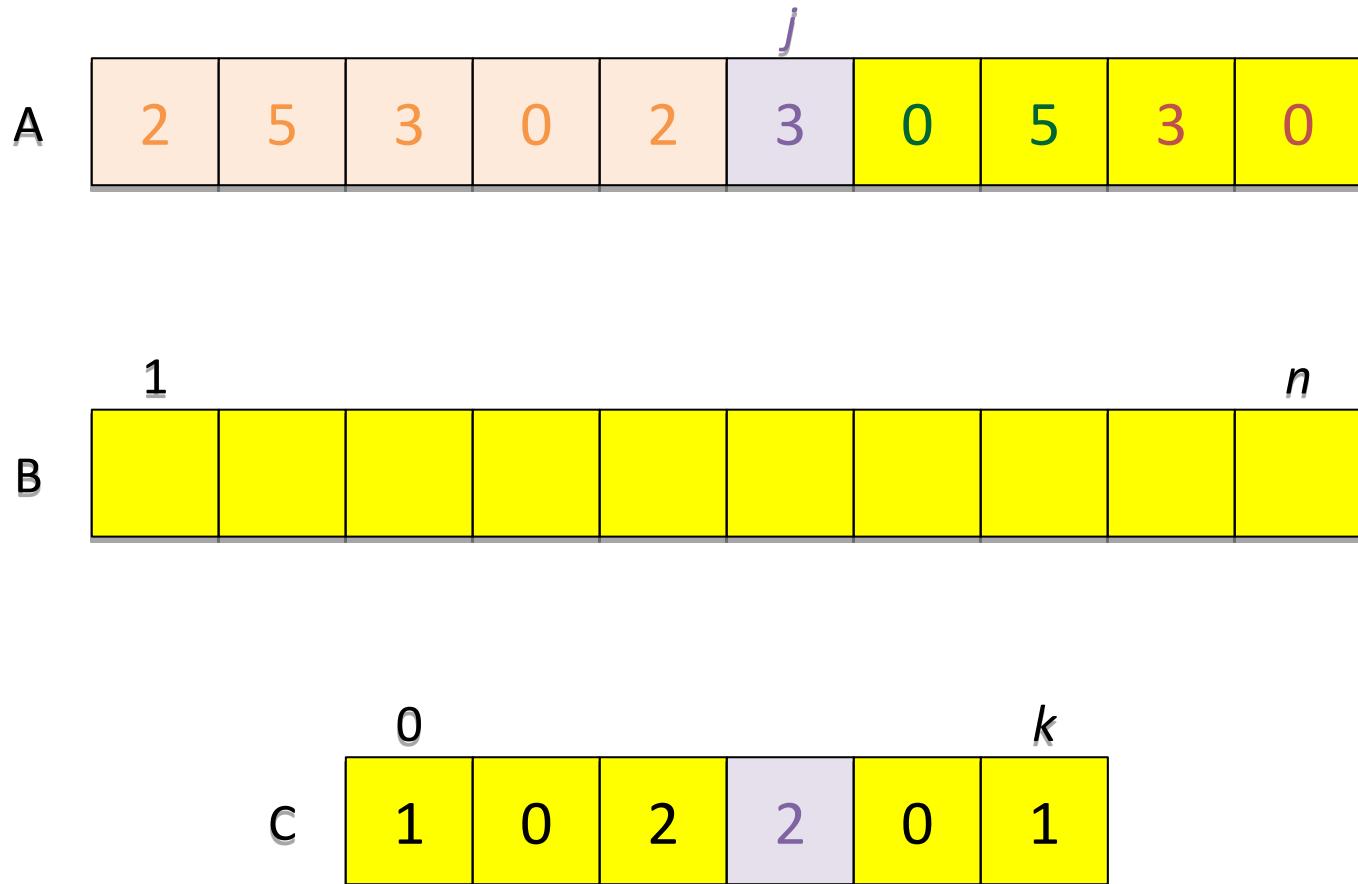
# Método Contagem



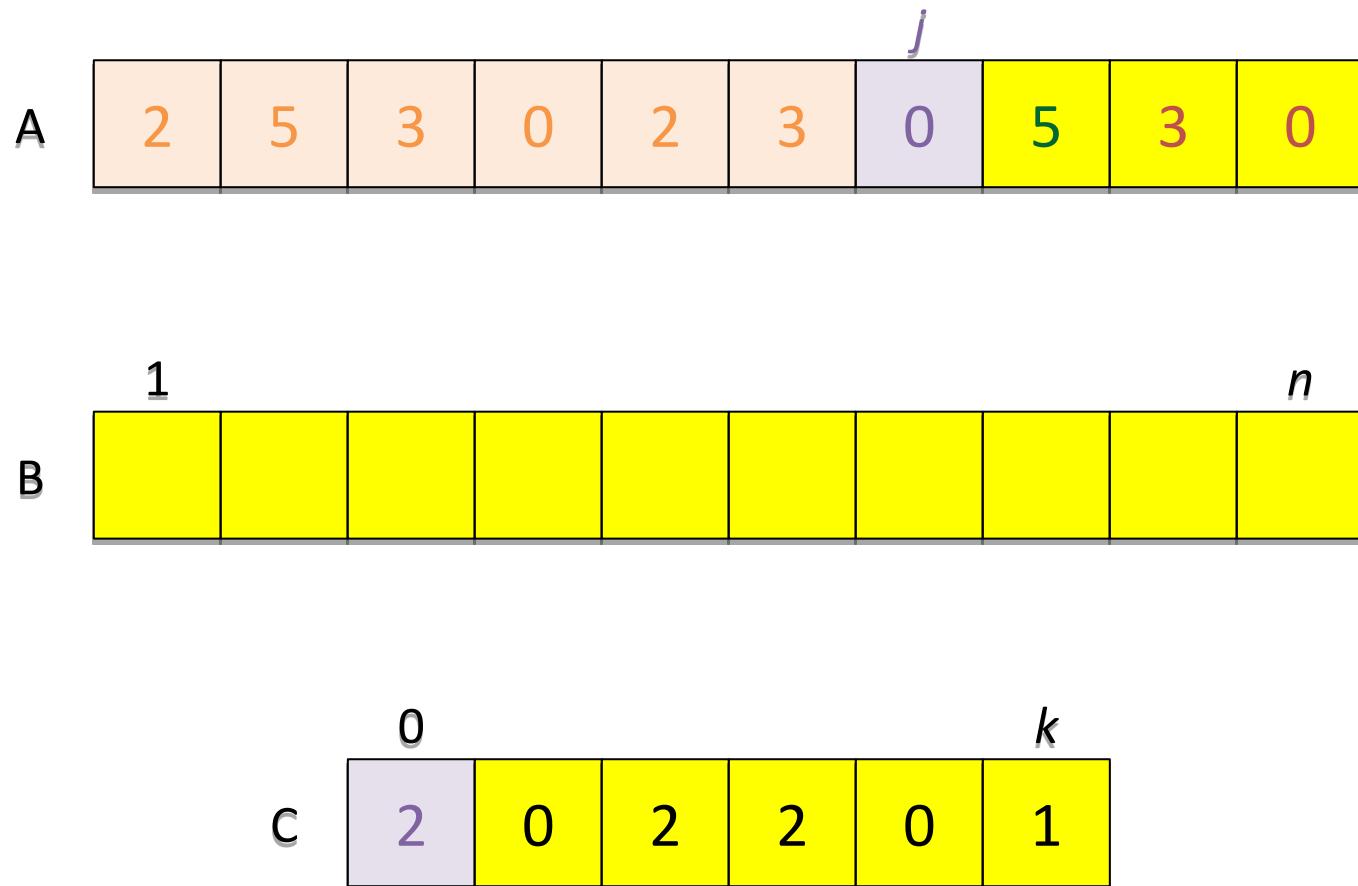
# Método Contagem



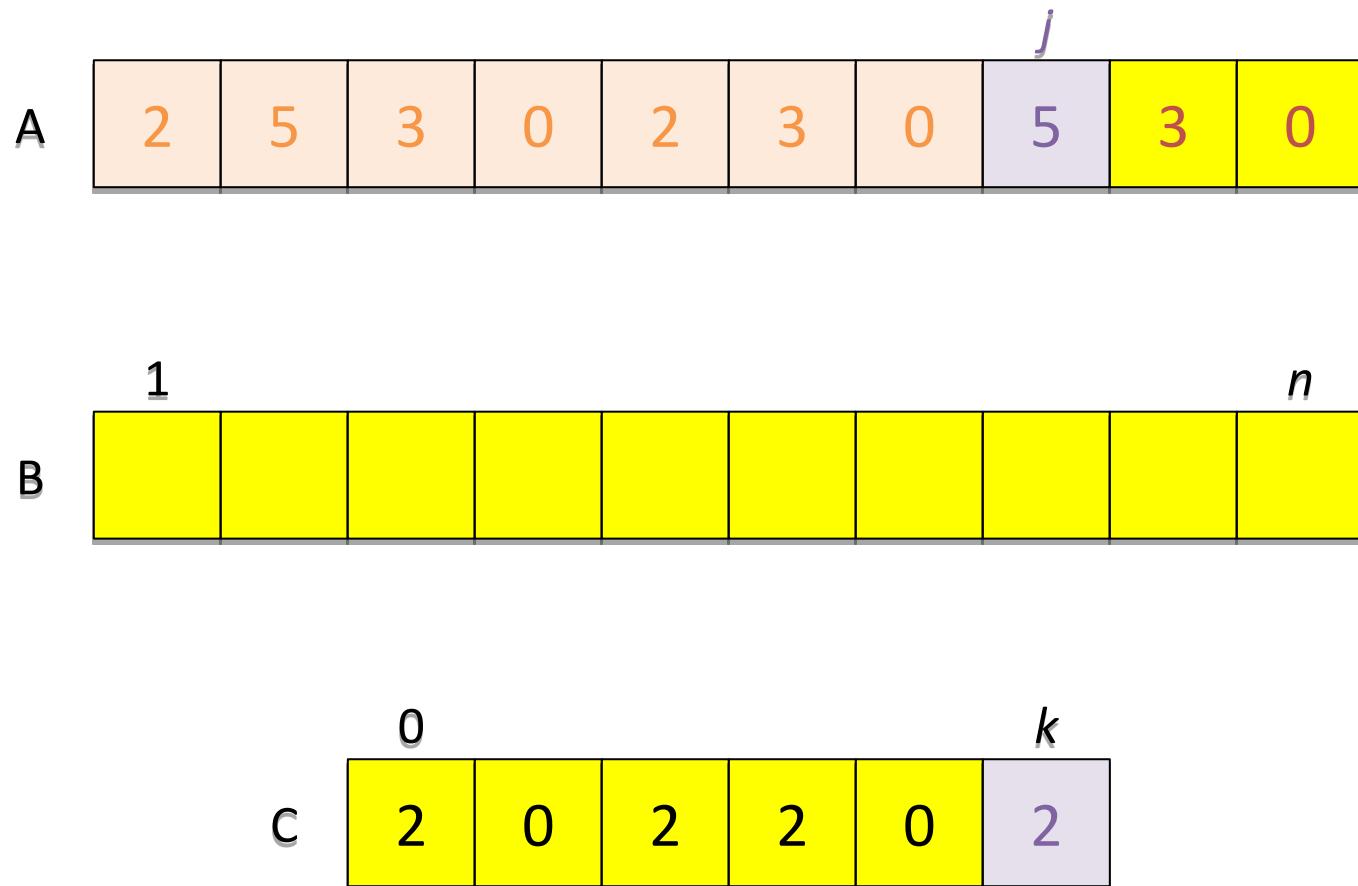
# Método Contagem



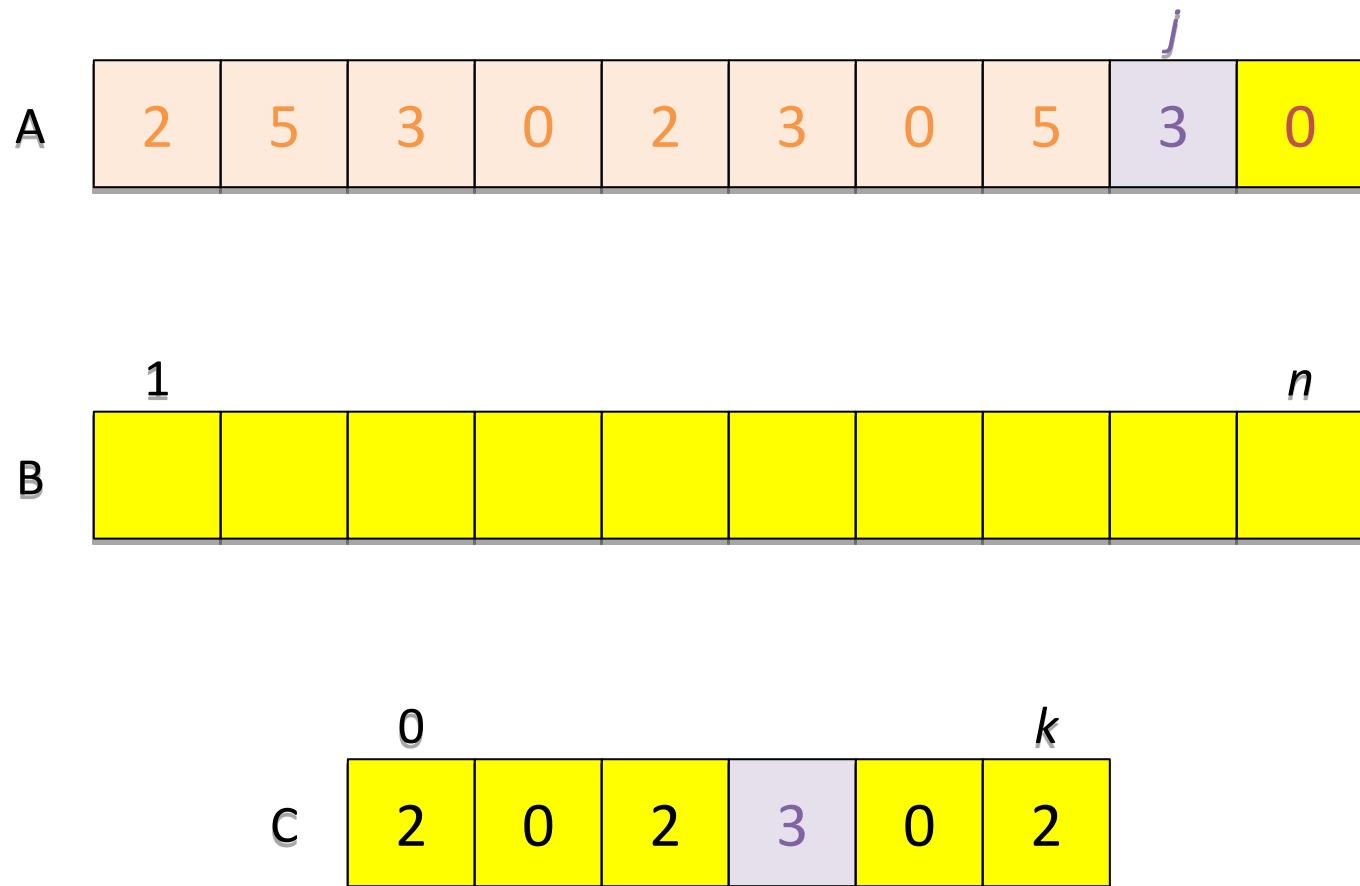
# Método Contagem



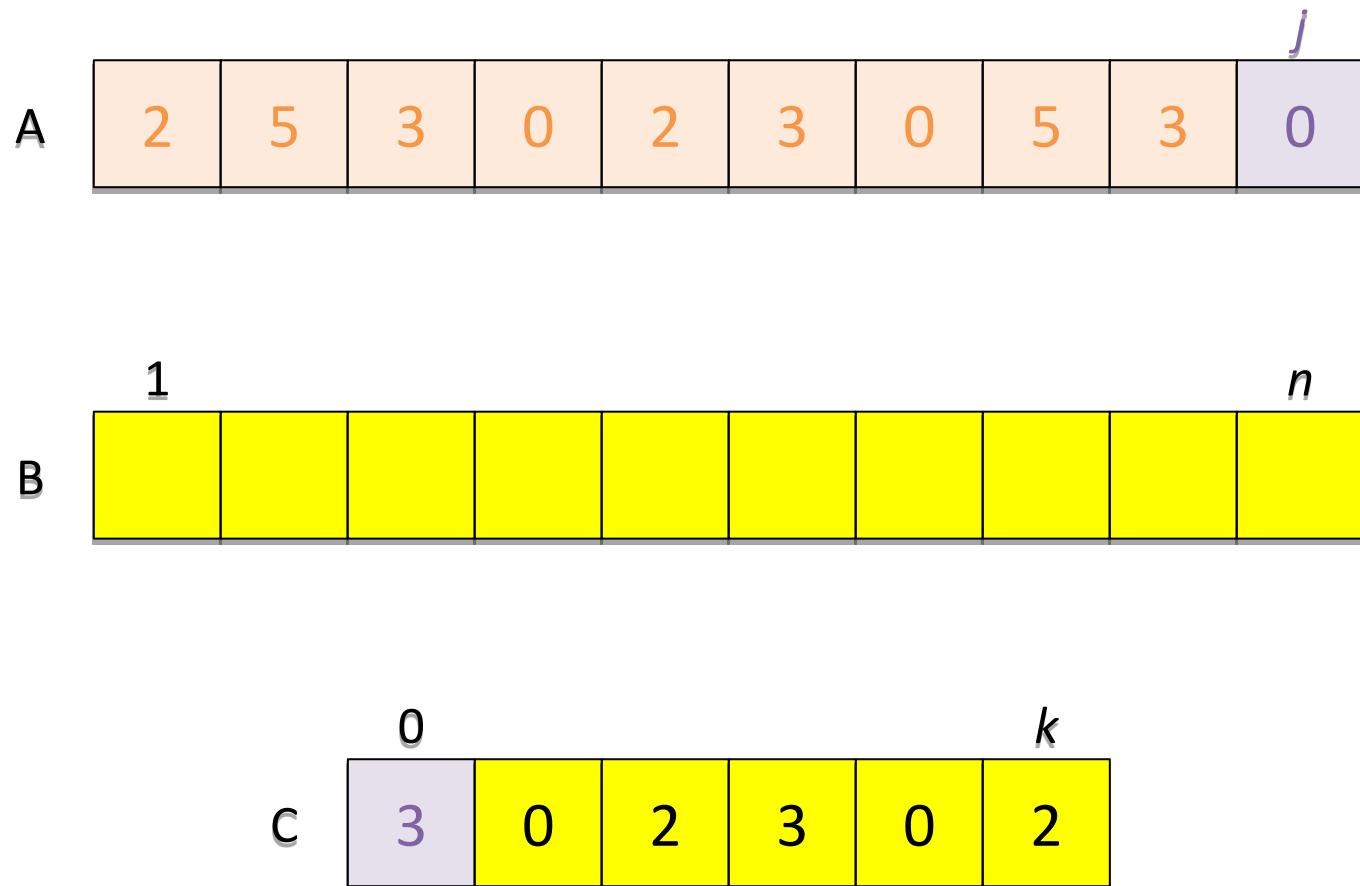
# Método Contagem



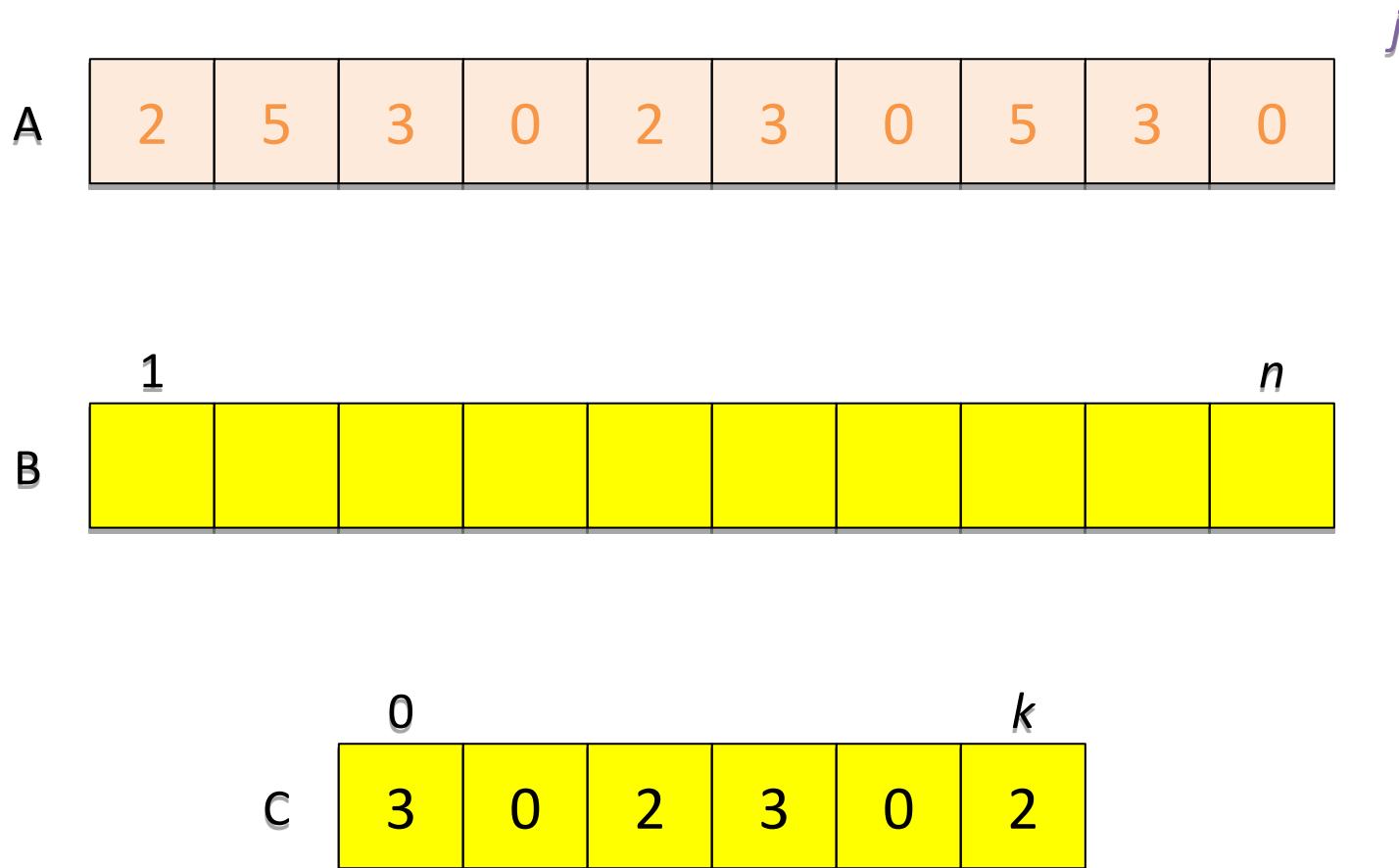
# Método Contagem



# Método Contagem



# Método Contagem



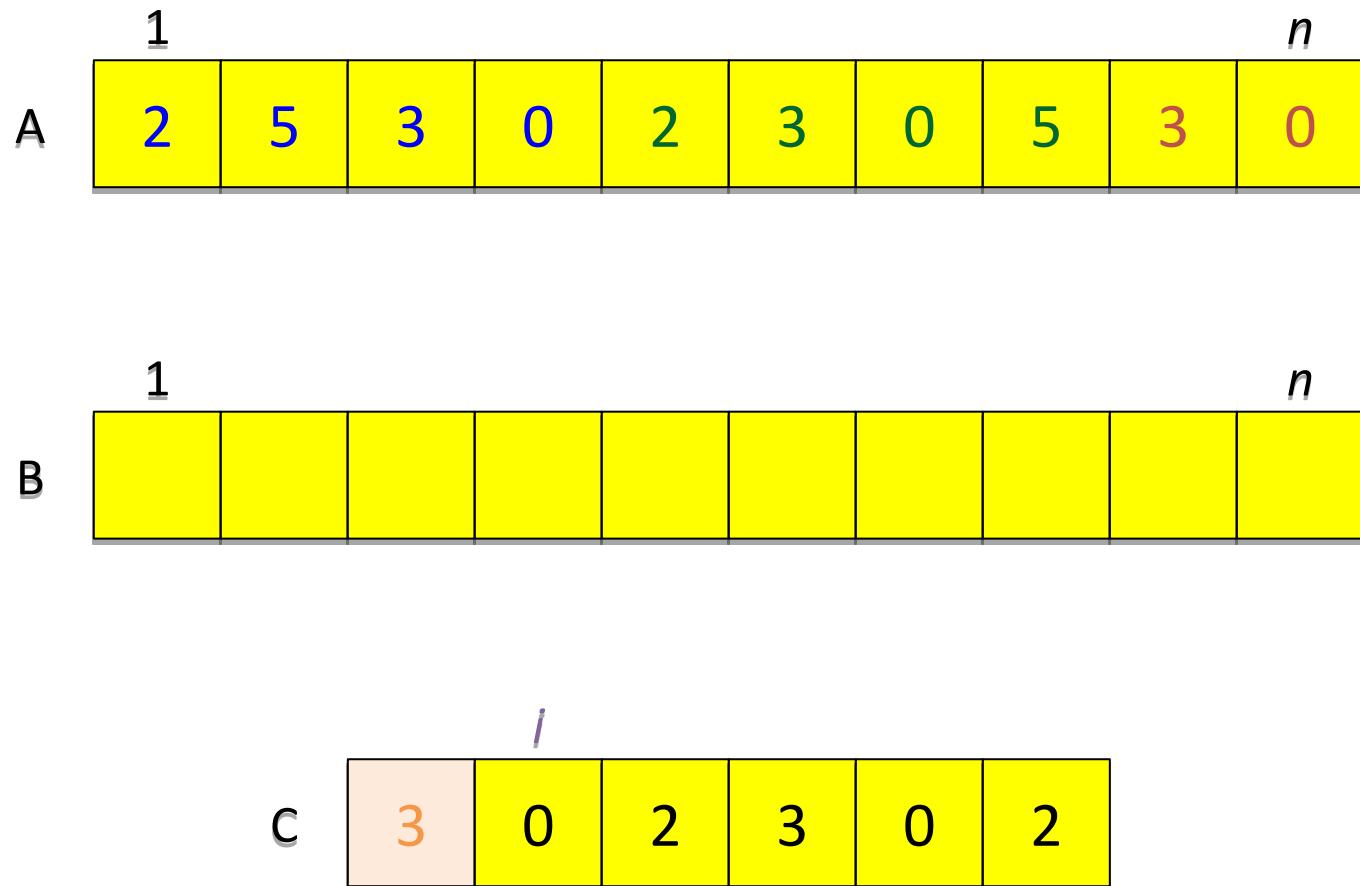
# Método Contagem

	1										n
A	2	5	3	0	2	3	0	5	3	0	

	1										n
B											

	0					k
C	3	0	2	3	0	2

# Método Contagem



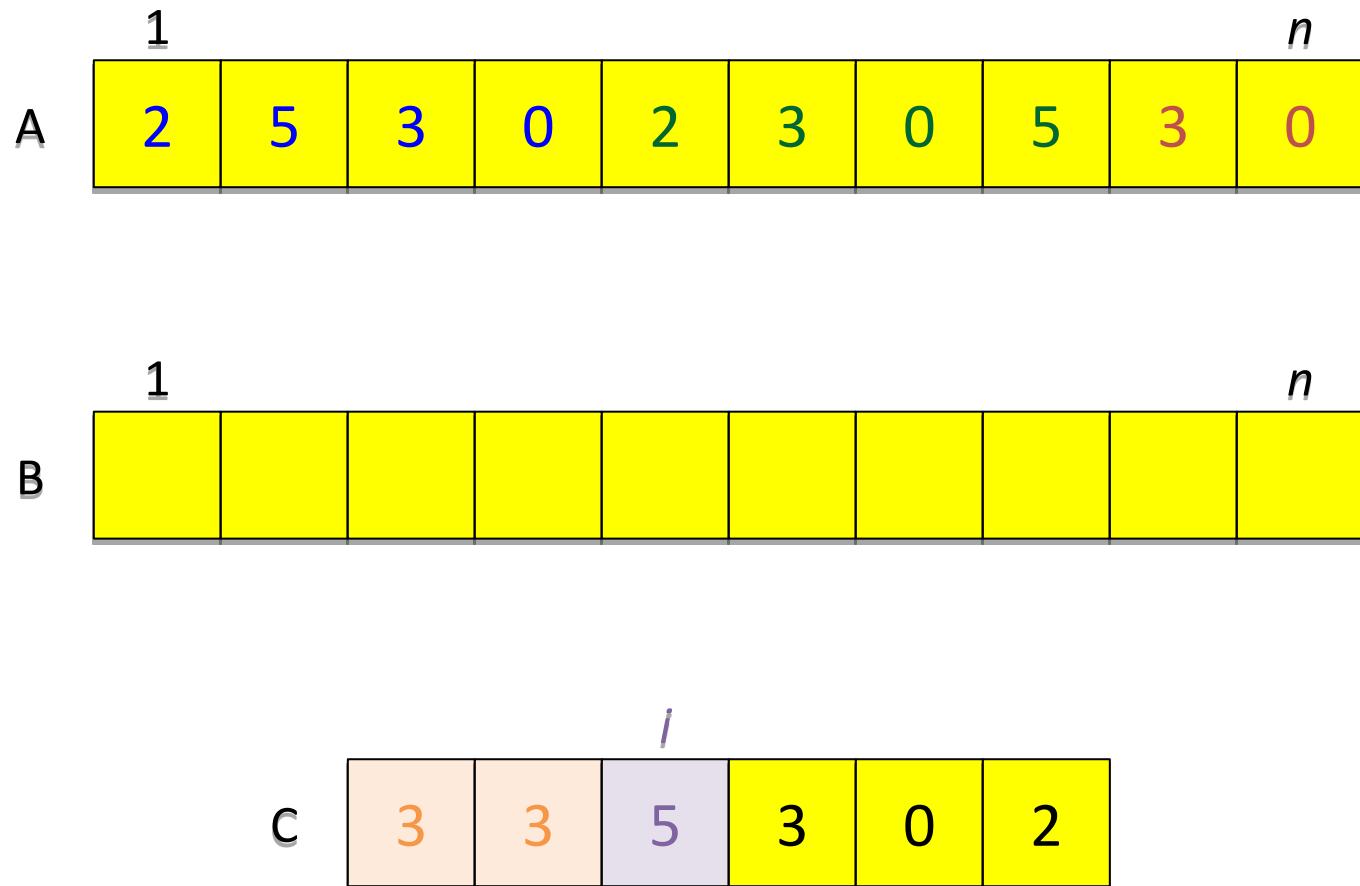
# Método Contagem

	1										n
A	2	5	3	0	2	3	0	5	3	0	

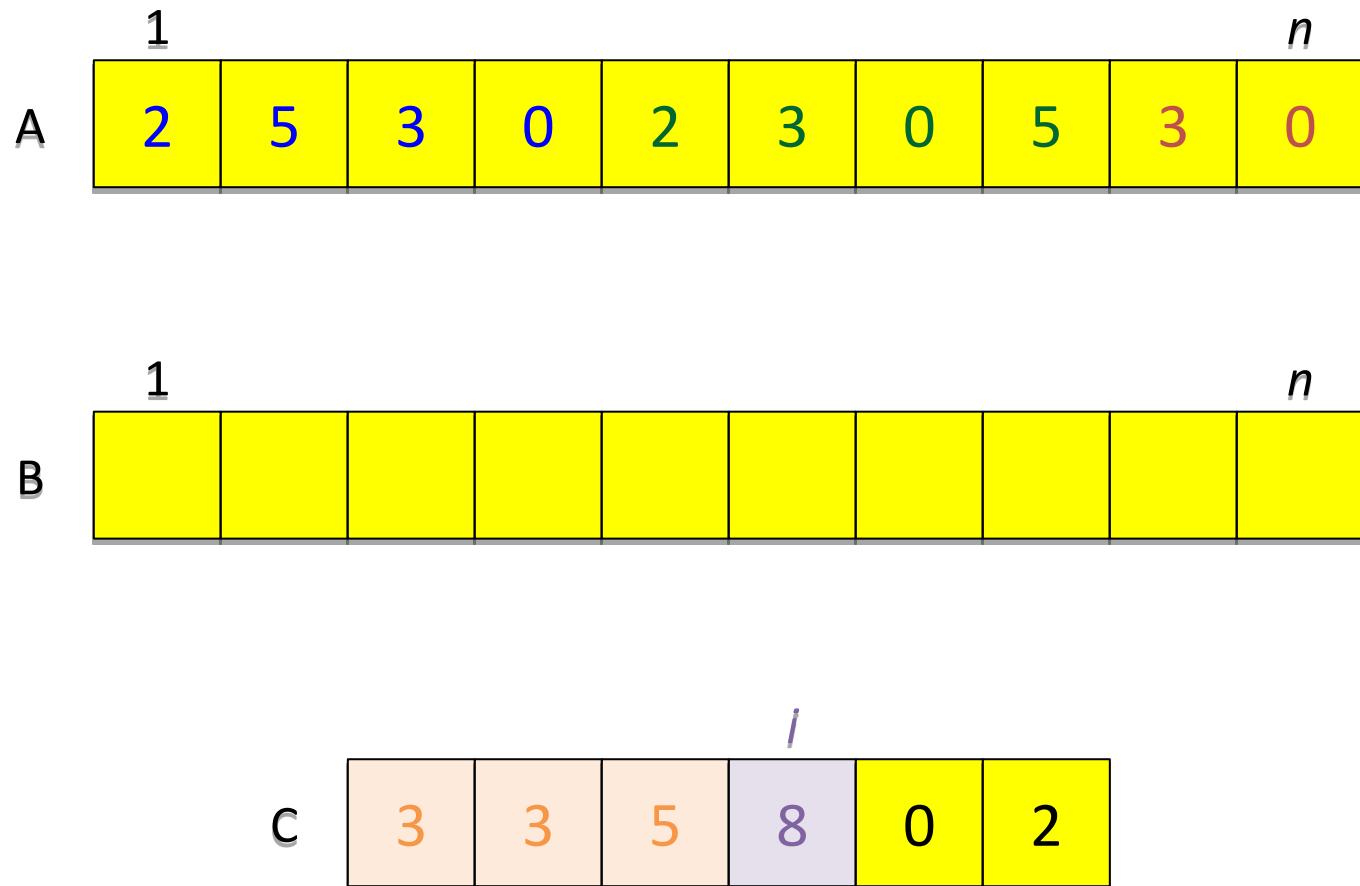
	1										n
B											

C		<i>i</i>									
	3	3	2	3	0	2					

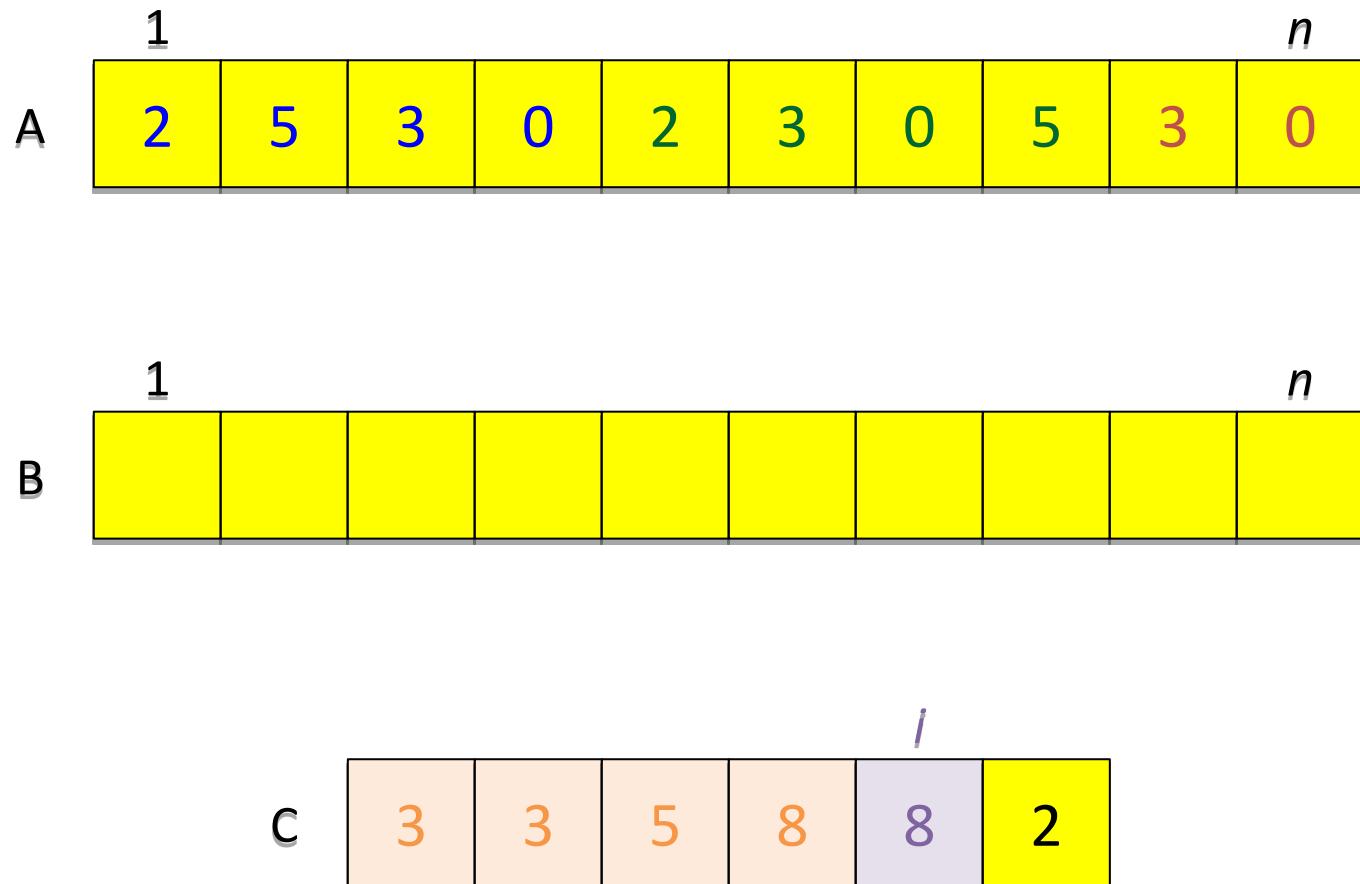
# Método Contagem



# Método Contagem



# Método Contagem



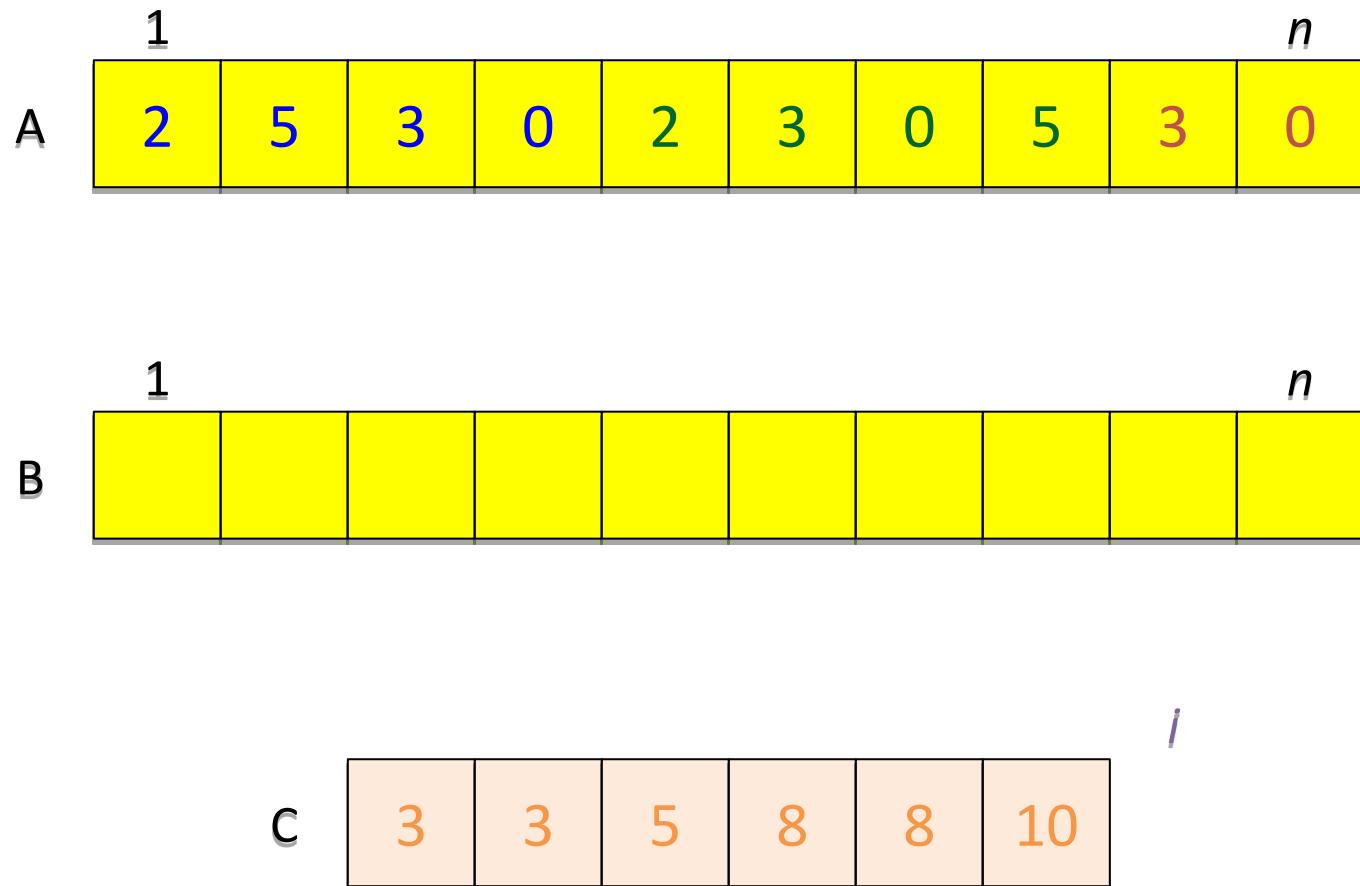
# Método Contagem

	1										n
A	2	5	3	0	2	3	0	5	3	0	

	1										n
B											

												i
C	3	3	5	8	8		10					

# Método Contagem



# Método Contagem

	1										n
A	2	5	3	0	2	3	0	5	3	0	

	1										n
B											

	0					k
C	3	3	5	8	8	10

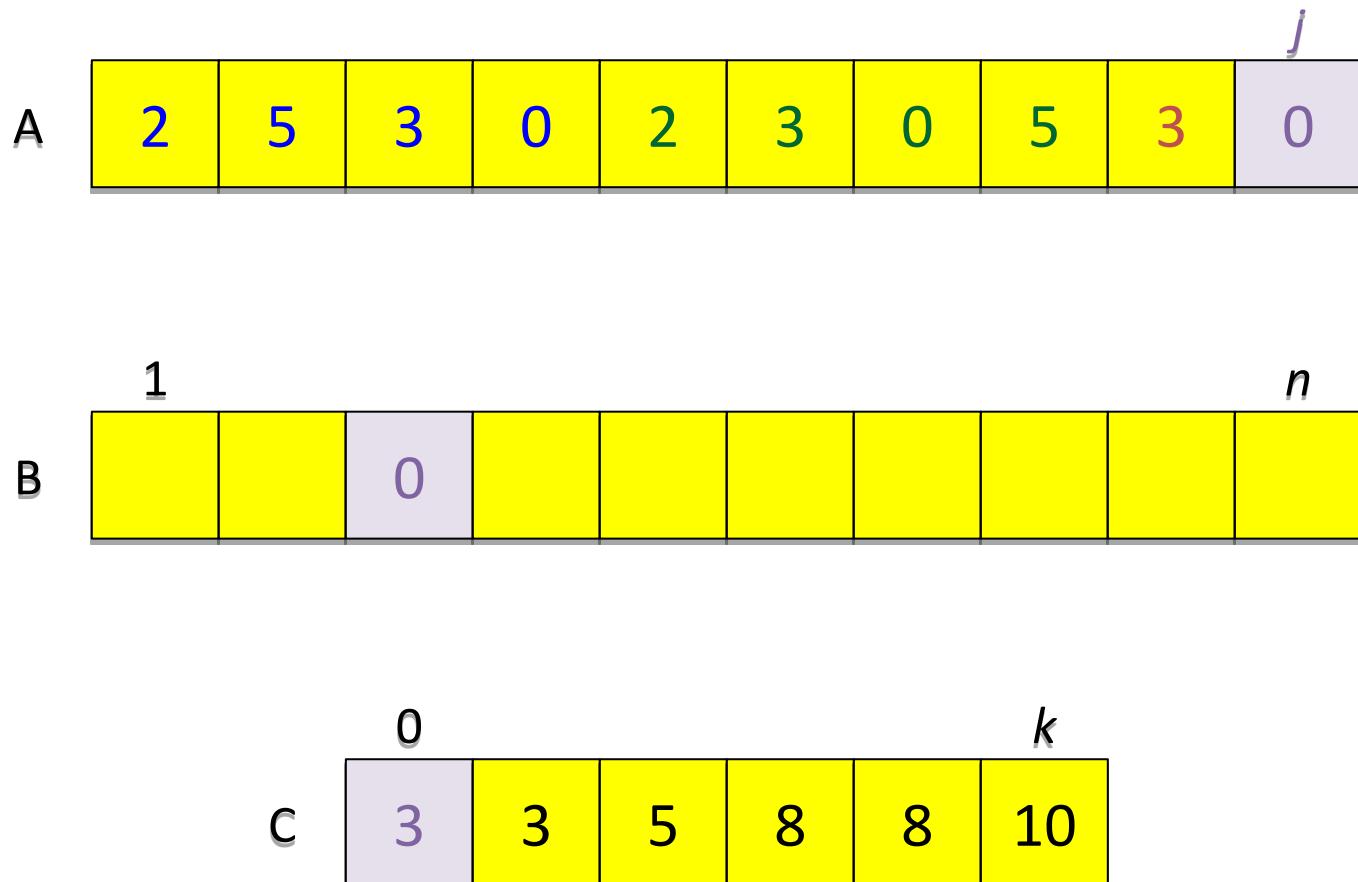
# Método Contagem

A	2	5	3	0	2	3	0	5	3	0	<i>j</i>
---	---	---	---	---	---	---	---	---	---	---	----------

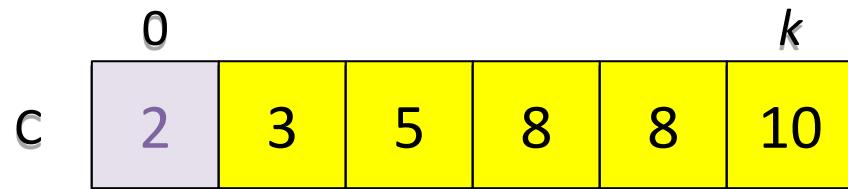
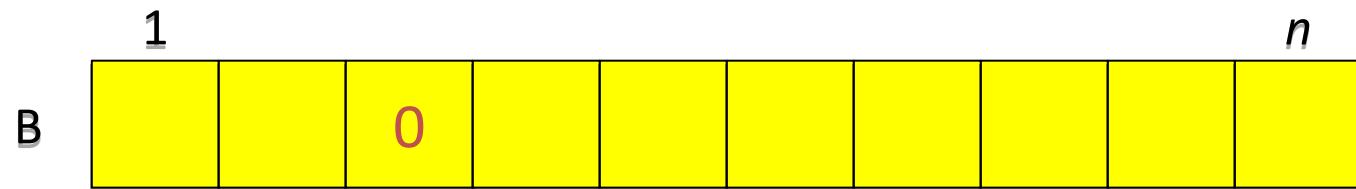
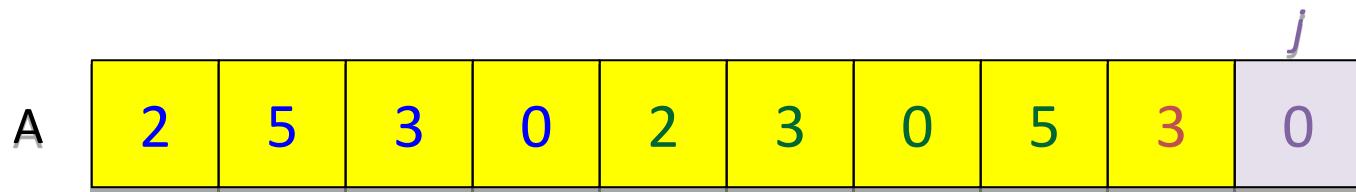
B	1										<i>n</i>
---	---	--	--	--	--	--	--	--	--	--	----------

C	0	3	3	5	8	8	10	<i>k</i>
---	---	---	---	---	---	---	----	----------

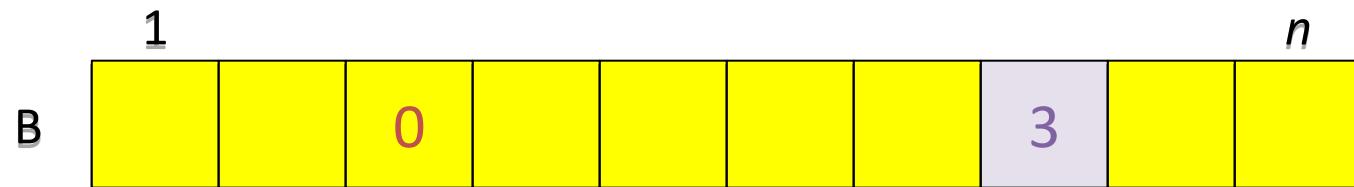
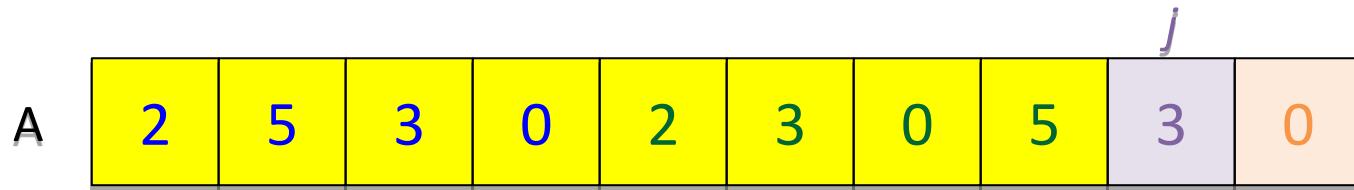
# Método Contagem



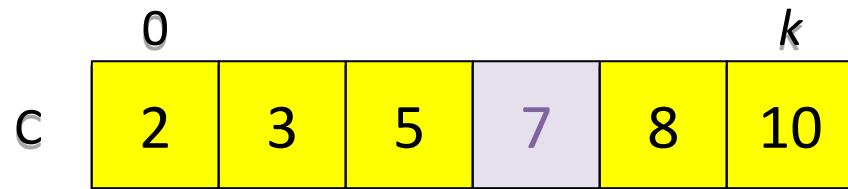
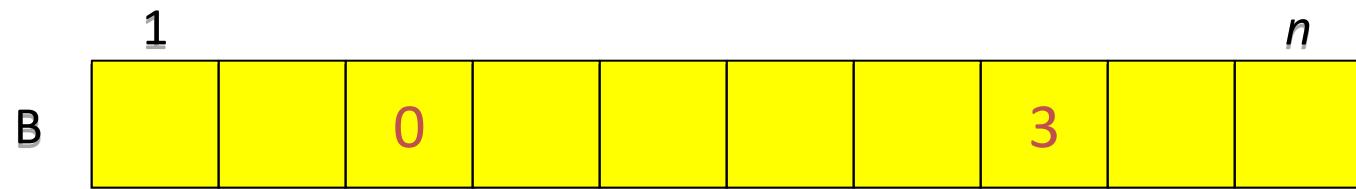
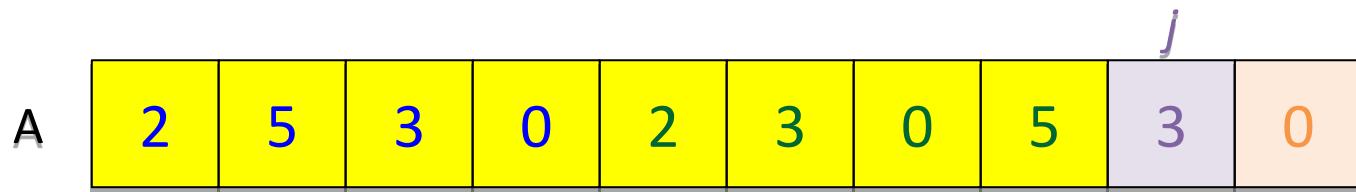
# Método Contagem



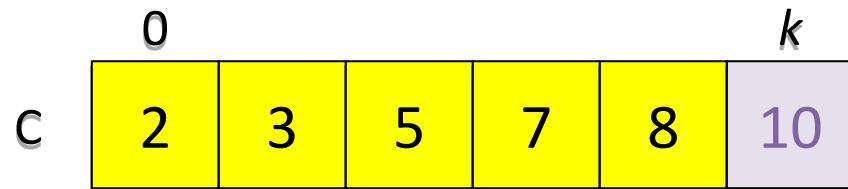
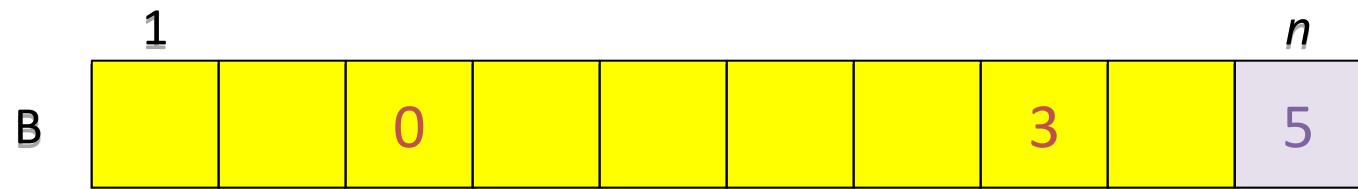
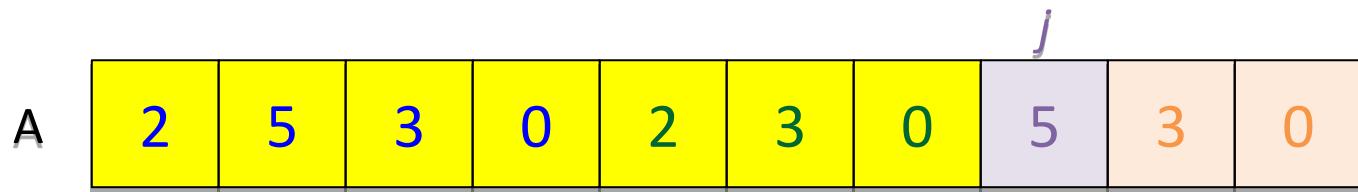
# Método Contagem



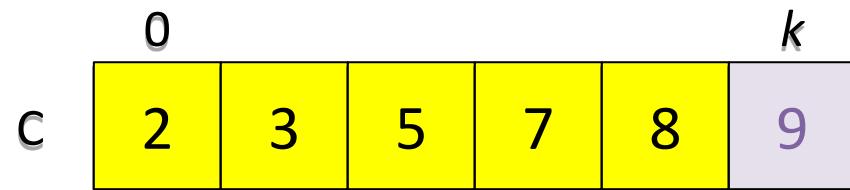
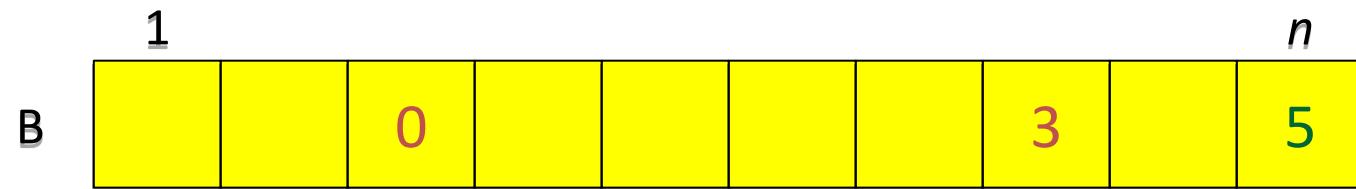
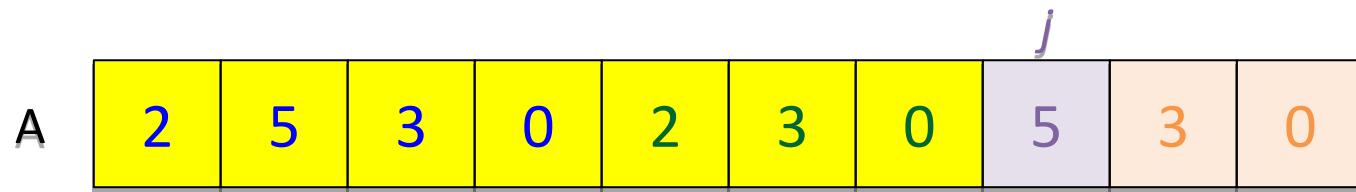
# Método Contagem



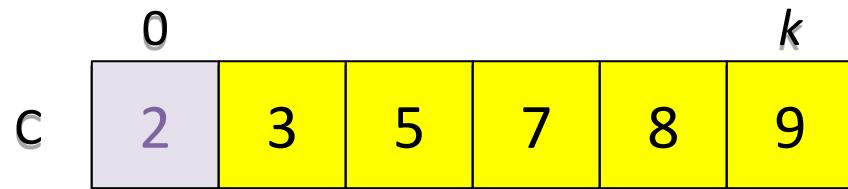
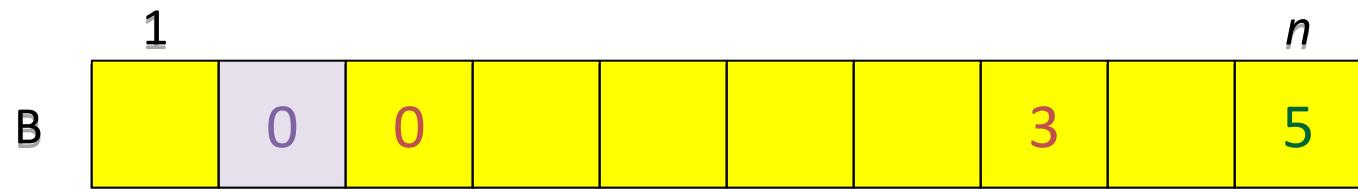
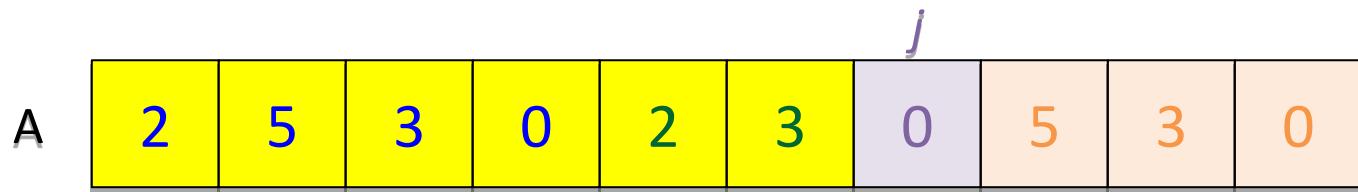
# Método Contagem



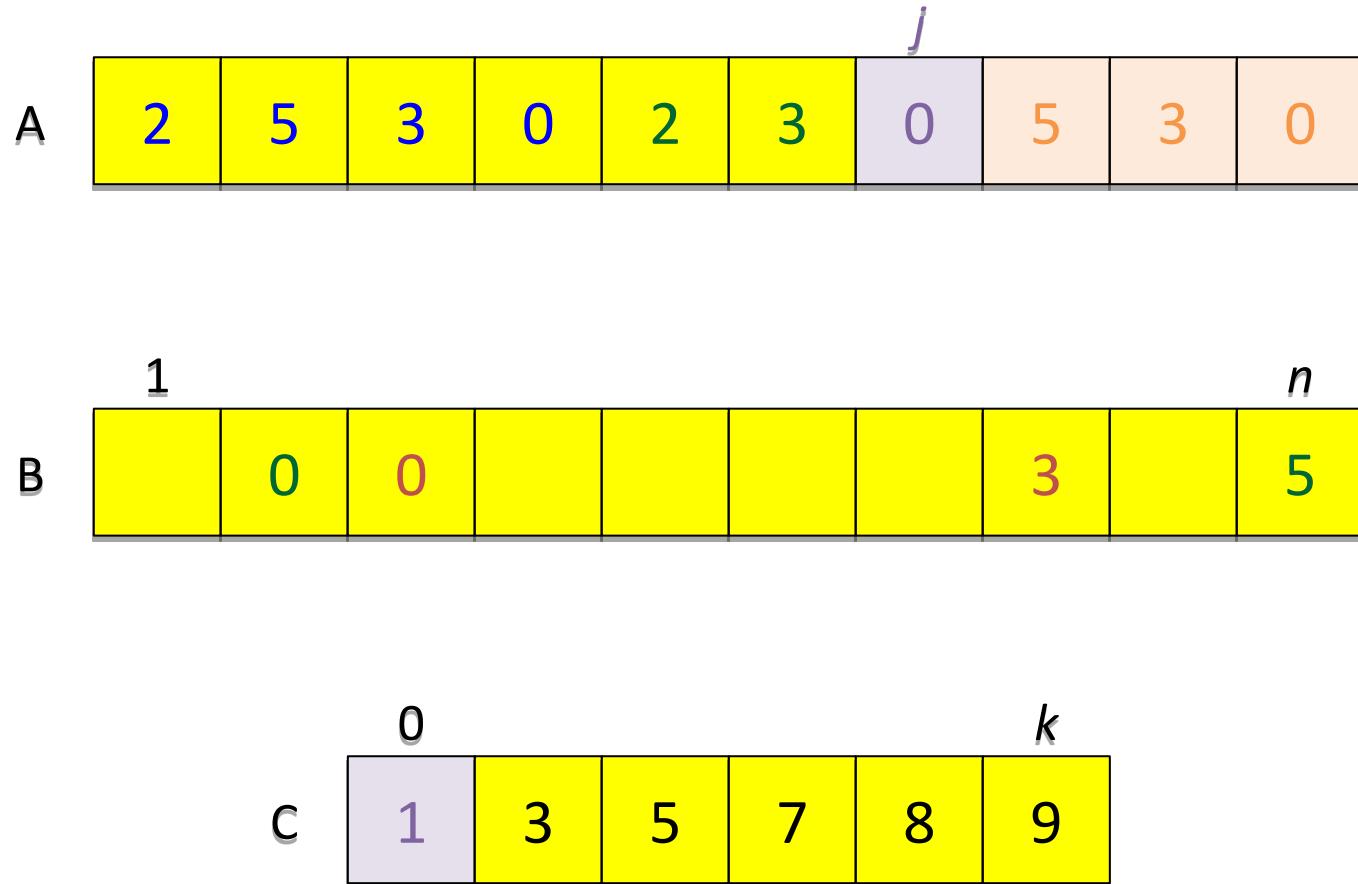
# Método Contagem



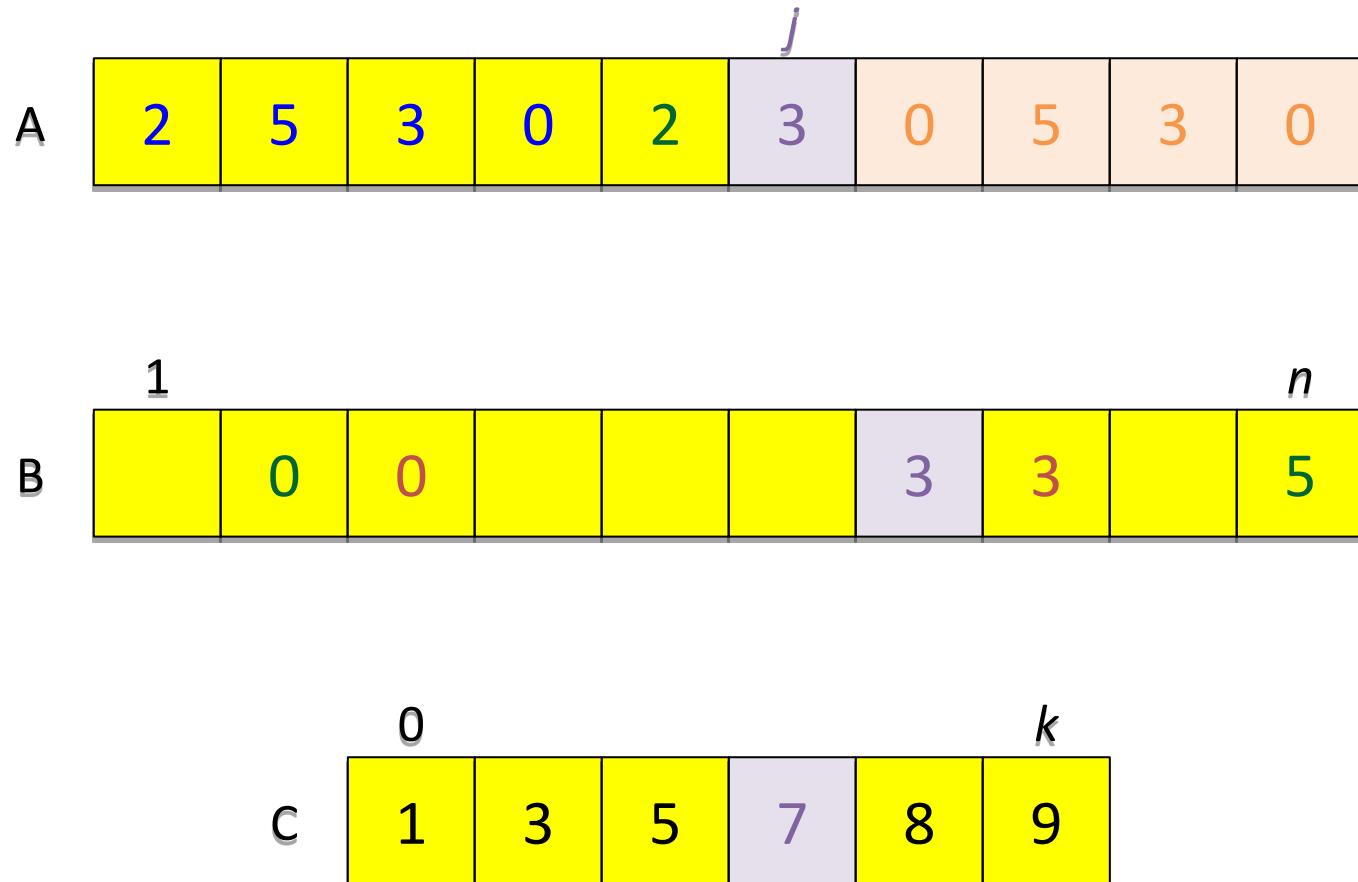
# Método Contagem



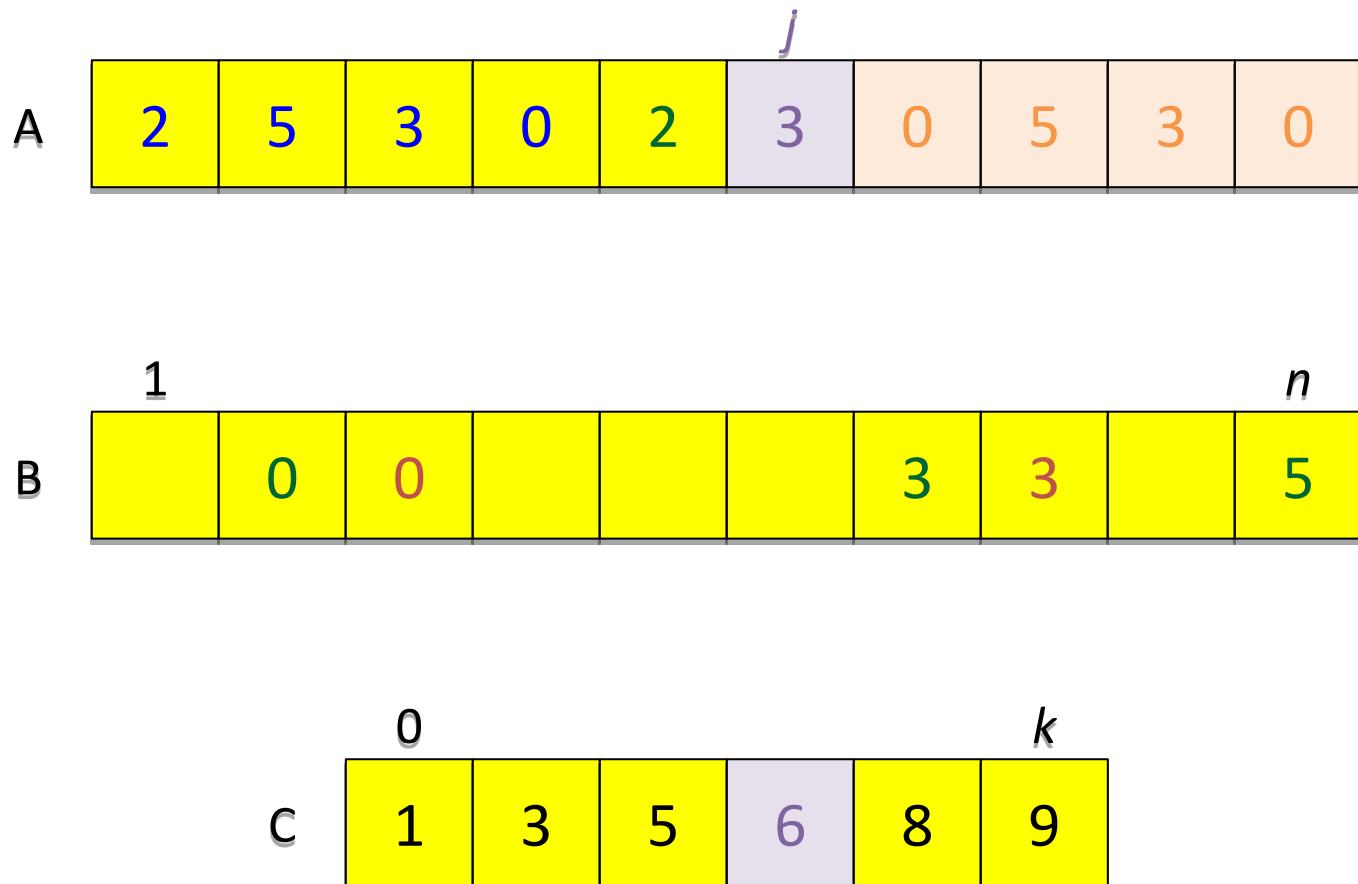
# Método Contagem



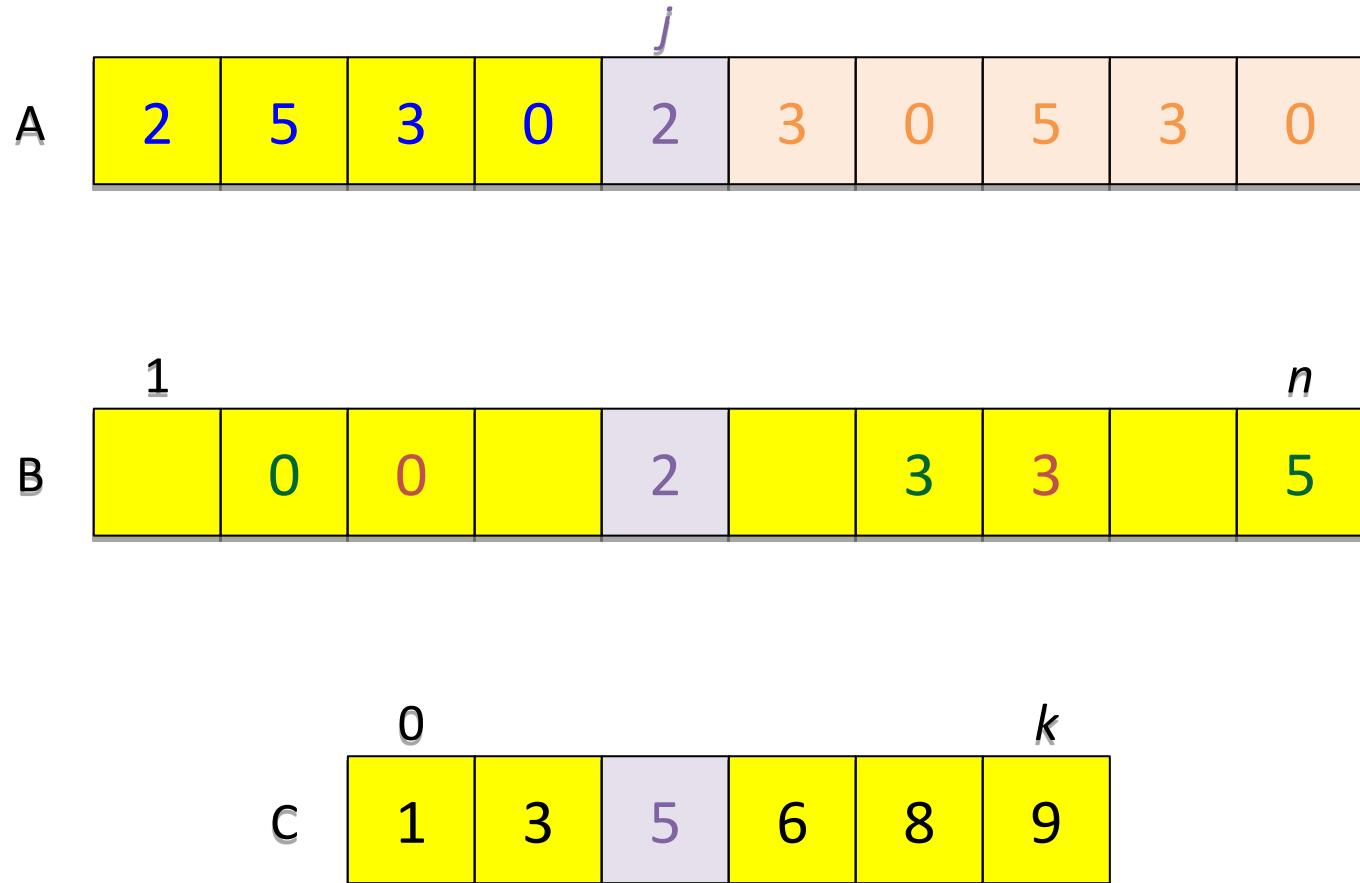
# Método Contagem



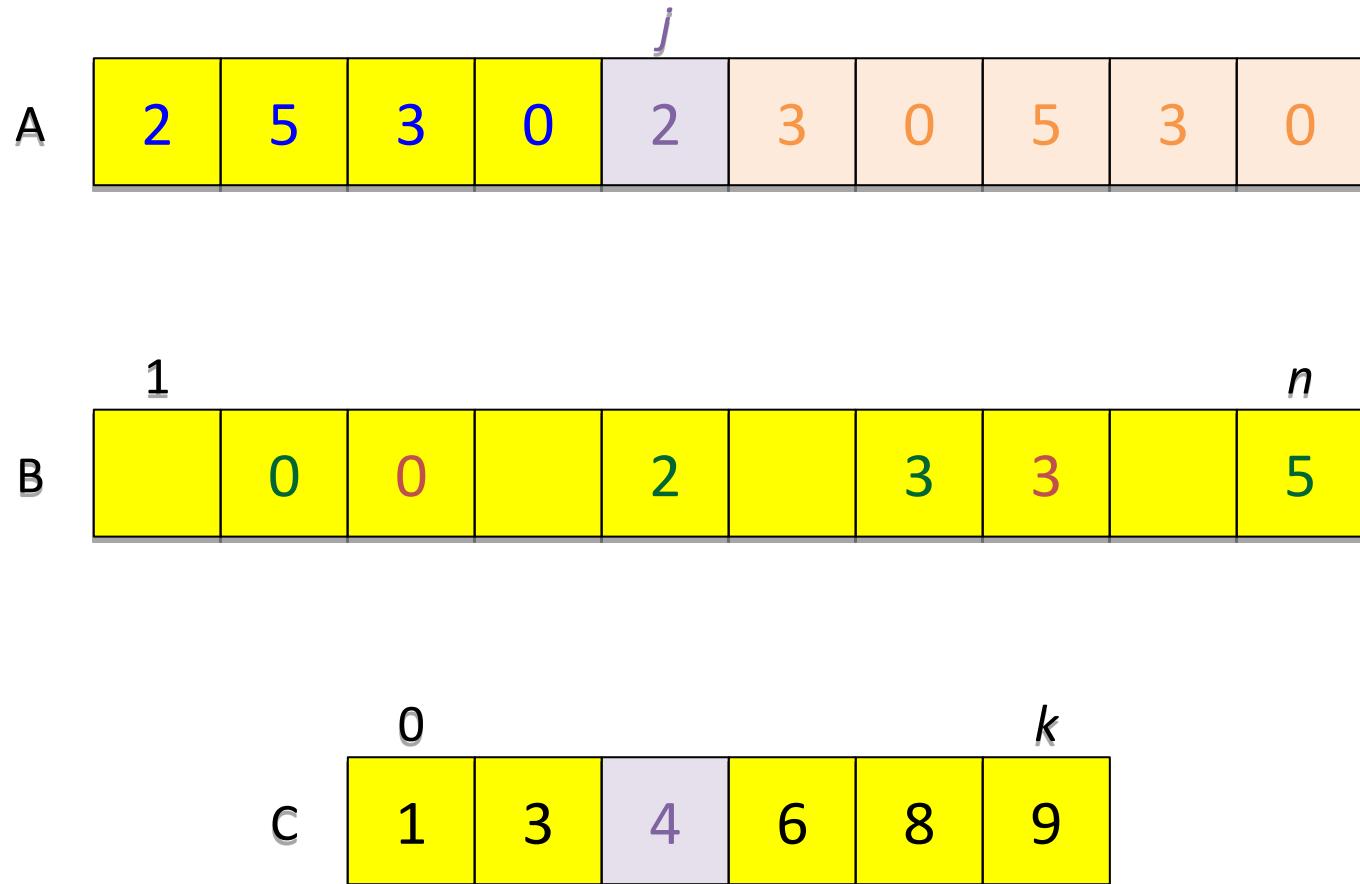
# Método Contagem



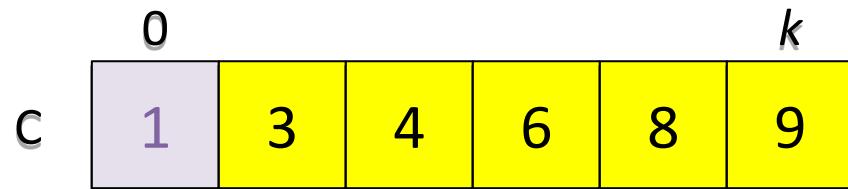
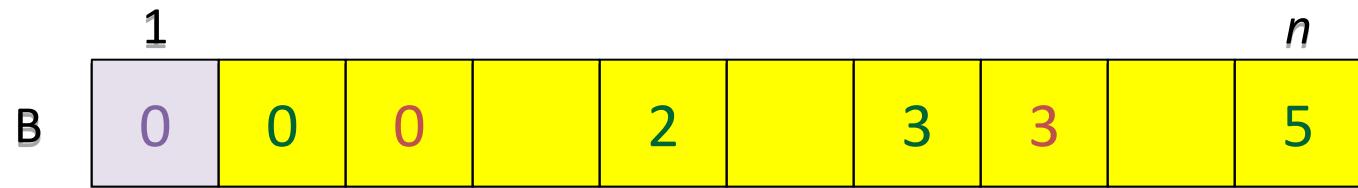
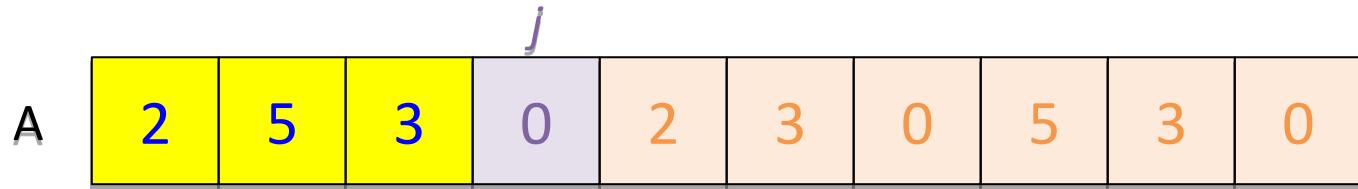
# Método Contagem



# Método Contagem



# Método Contagem



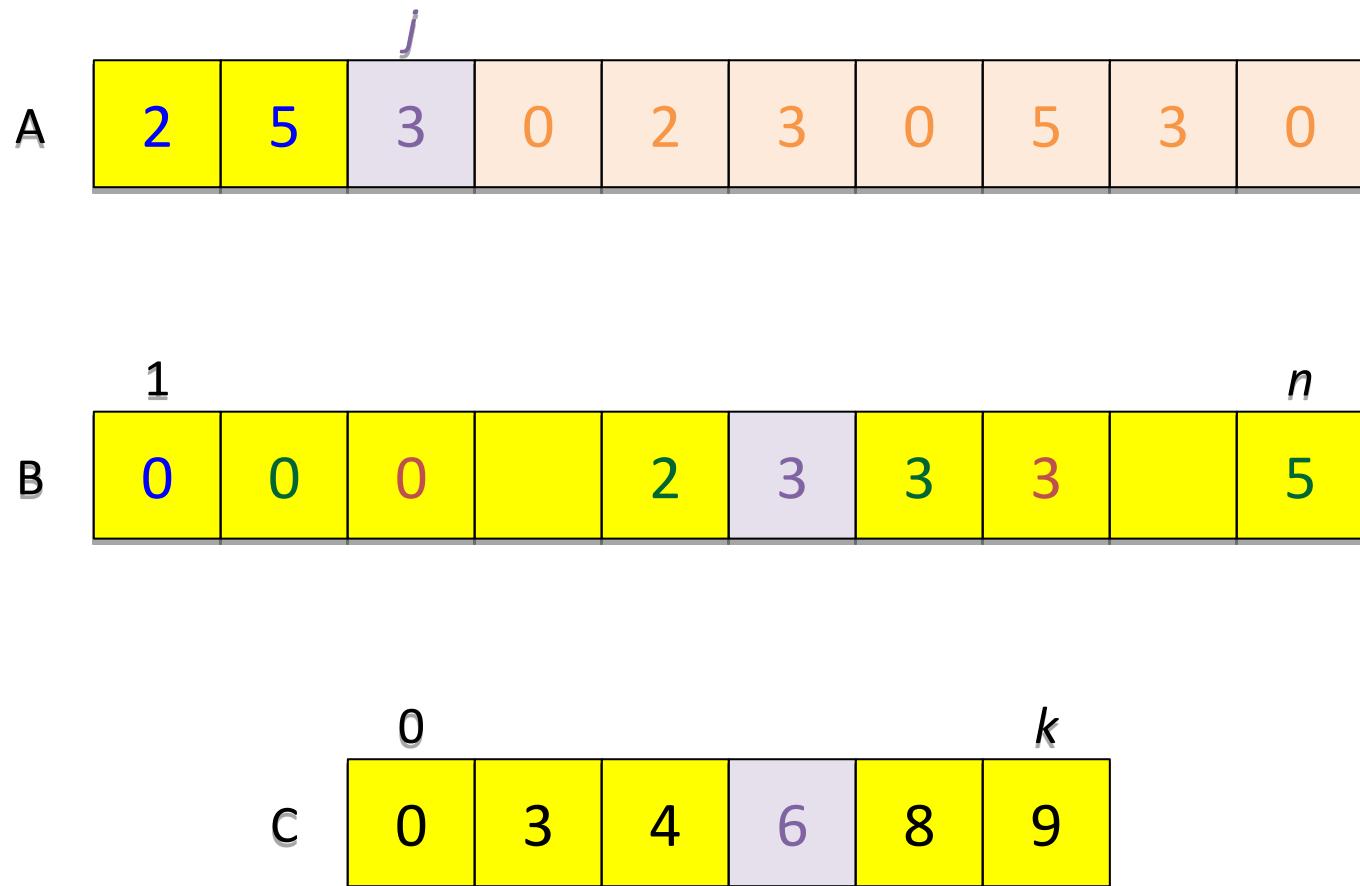
# Método Contagem

A	2	5	3	0	2	3	0	5	3	0
---	---	---	---	---	---	---	---	---	---	---

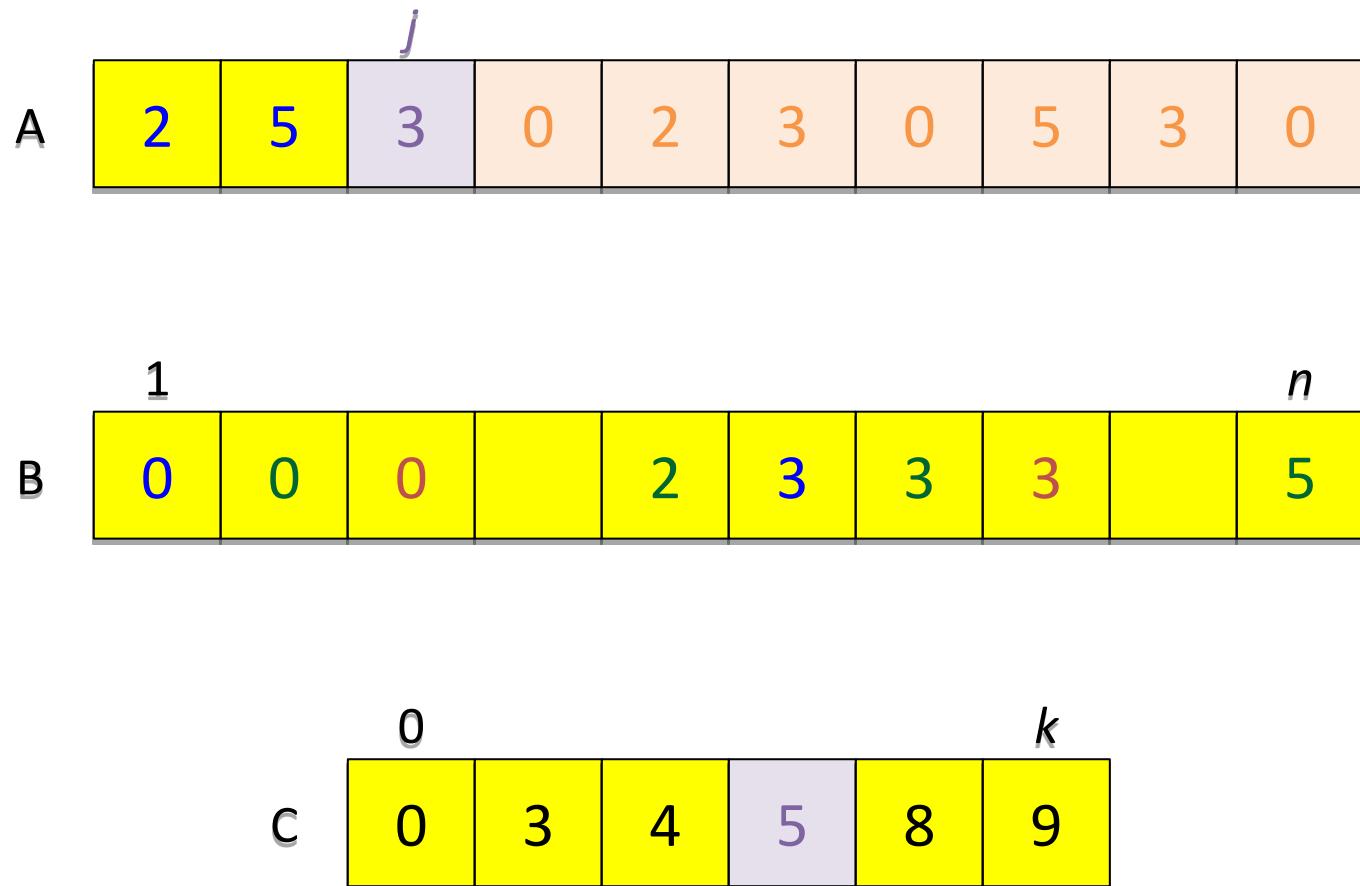
B	0	0	0		2		3	3		5
---	---	---	---	--	---	--	---	---	--	---

C	0	3	4	6	8	9
---	---	---	---	---	---	---

# Método Contagem



# Método Contagem



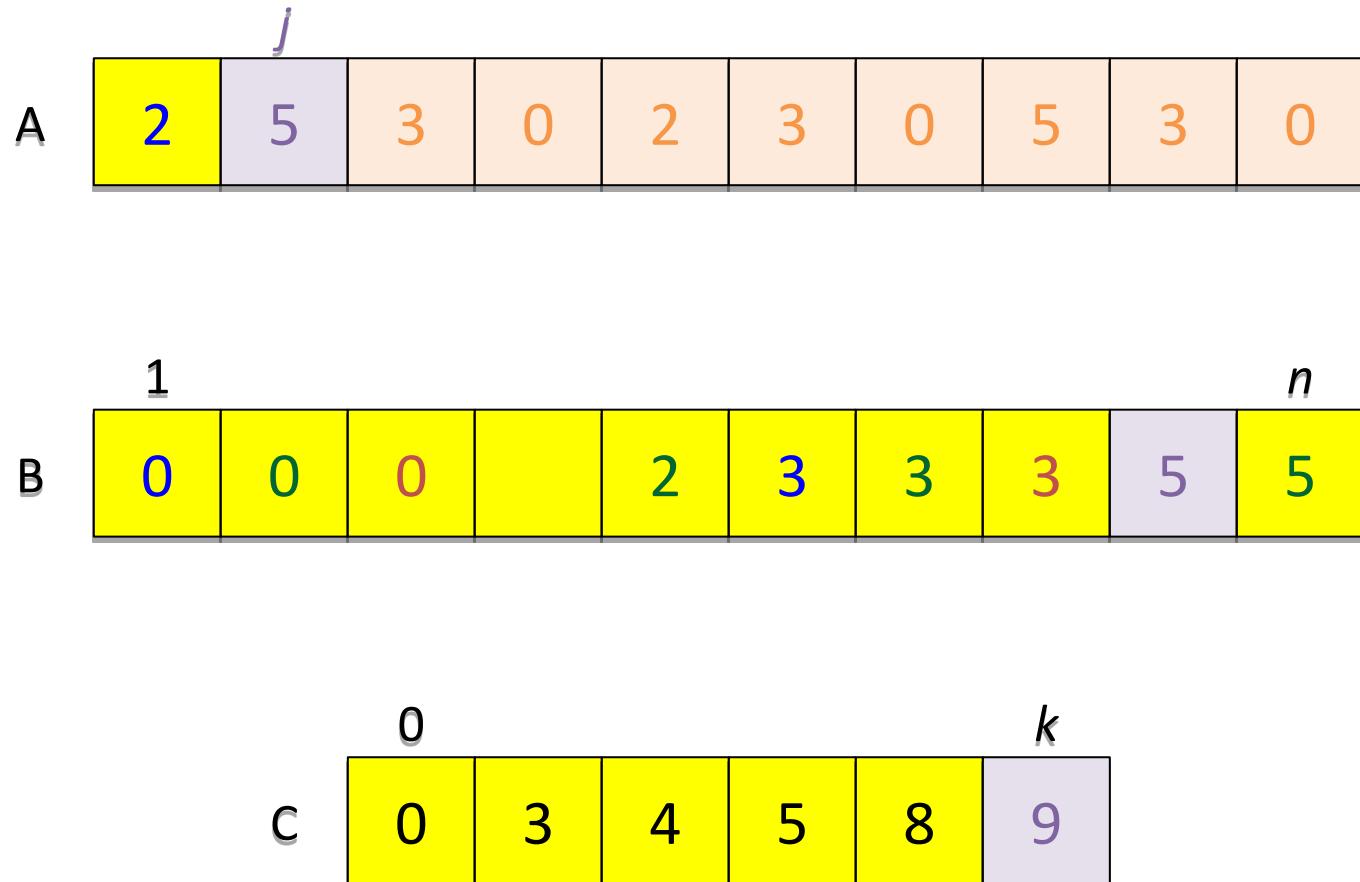
# Método Contagem

A	2	5	3	0	2	3	0	5	3	0
---	---	---	---	---	---	---	---	---	---	---

B	0	0	0		2	3	3	3	5	5
---	---	---	---	--	---	---	---	---	---	---

C	0	3	4	5	8	8
---	---	---	---	---	---	---

# Método Contagem



# Método Contagem

A  $j$

2	5	3	0	2	3	0	5	3	0
---	---	---	---	---	---	---	---	---	---

B  $1$   $n$

0	0	0	2	2	3	3	3	5	5
---	---	---	---	---	---	---	---	---	---

C  $0$   $k$

0	3	4	5	8	8
---	---	---	---	---	---

# Método Contagem

A  $j$

2	5	3	0	2	3	0	5	3	0
---	---	---	---	---	---	---	---	---	---

B  $1$   $n$

0	0	0	2	2	3	3	3	5	5
---	---	---	---	---	---	---	---	---	---

C  $0$   $k$

0	3	3	5	8	8
---	---	---	---	---	---

# Método Contagem

*j*

A	2	5	3	0	2	3	0	5	3	0
---	---	---	---	---	---	---	---	---	---	---

$1$   $n$

B	0	0	0	2	2	3	3	3	5	5
---	---	---	---	---	---	---	---	---	---	---

$0$   $k$

C	0	3	3	5	8	8
---	---	---	---	---	---	---

# Método Contagem

A	1										n
	2	5	3	0	2	3	0	5	3	0	

B	1										n
	0	0	0	2	2	3	3	3	5	5	

C	0					k
	0	3	3	5	8	8

- O número de operações é dado em função de  $k$
- Executa  $O(n + k)$  operações
  - se  $k \leq n$  então executa  $O(n)$  operações
  - se  $k \leq 10n$  então executa  $O(n)$  operações
  - se  $k = O(n)$  então executa  $O(n)$  operações
  - se  $k \geq n^2$  então executa  $O(k)$  operações

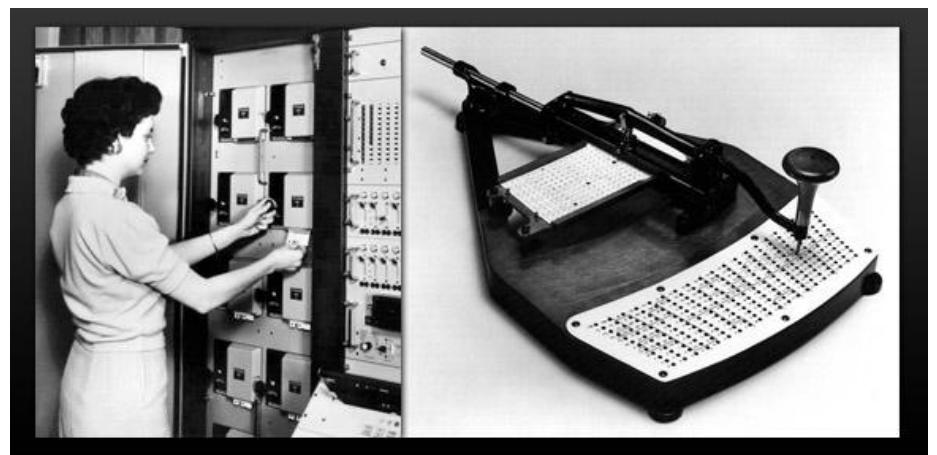
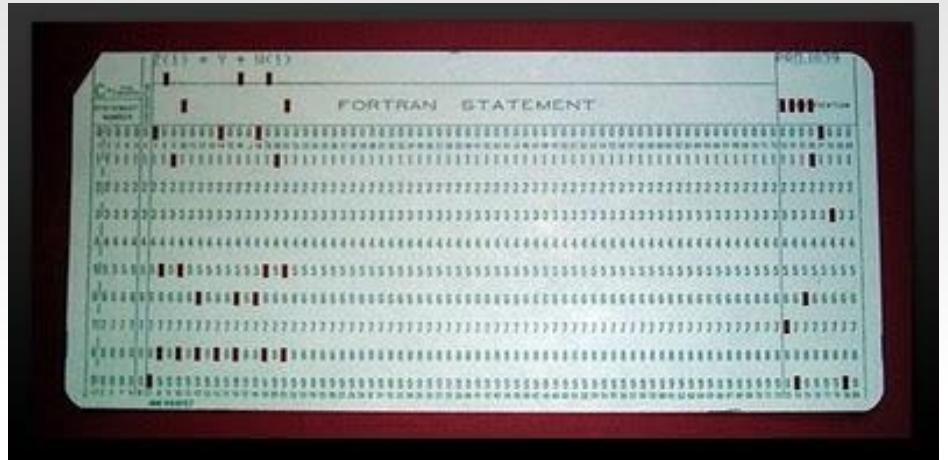
# Ordenação por Contagem

- O número de operações é dado em função de  $k$
- Executa  $O(n + k)$  operações
- Utiliza memória auxiliar
- Adequado quando o número de chaves não é muito maior do que o número de elementos
- O algoritmo de ordenação por contagem é **estável**

# *Radix-Sort*

## (ordenação da raiz)

- Algoritmo muito utilizado pelas máquinas de ordenação de cartões
  - Os cartões tinham 80 colunas, cada uma perfurada em até 12 posições
  - Uma máquina examina uma coluna e distribui os cartões em caixas



## *Radix-Sort (cont.)*

- Para números decimais, apenas 10 posições são usadas em cada coluna
  - Um número de  $d$ -dígitos ocuparia, cada dígito, um campo de 10 colunas, resultando em  $d$ -colunas de 10 posições.
  - Lembrando que a máquina de ordenação pode examinar apenas uma coluna de cada vez

## ***Radix-Sort (cont.)***

- Intuitivamente, poderíamos ordenar os números/cartões de acordo com o dígito mais significativo, depois:
  - ordenar cada caixa recursivamente e
  - combinar os cartões de acordo com a ordem desejada
- Contudo, uma vez que 9 das 10 caixas devem ser deixadas de lado, este procedimento gera muitas pilhas intermediárias que são difíceis de serem organizadas

## ***Radix-Sort (cont.)***

- De forma contra-intuitiva, *Radix-Sort* ordena a partir do dígito menos significativo
  - Os cartões são então combinados em uma pilha única, com os cartões de dígito 0 precedendo os de dígito 1, que precedem os de dígito 2 e assim sucessivamente
  - A pilha é ordenada novamente segundo o segundo dígito menos significativo
  - O processo é repetido até o dígito  $d$ 
    - É essencial que os dígitos sejam ordenados por um algoritmo estável.

# Método RadixSort

3	2	9
---	---	---

4	5	7
---	---	---

6	5	7
---	---	---

8	3	9
---	---	---

4	3	6
---	---	---

7	2	0
---	---	---

3	5	5
---	---	---

# Método RadixSort

3	2	9
---	---	---

4	5	7
---	---	---

6	5	7
---	---	---

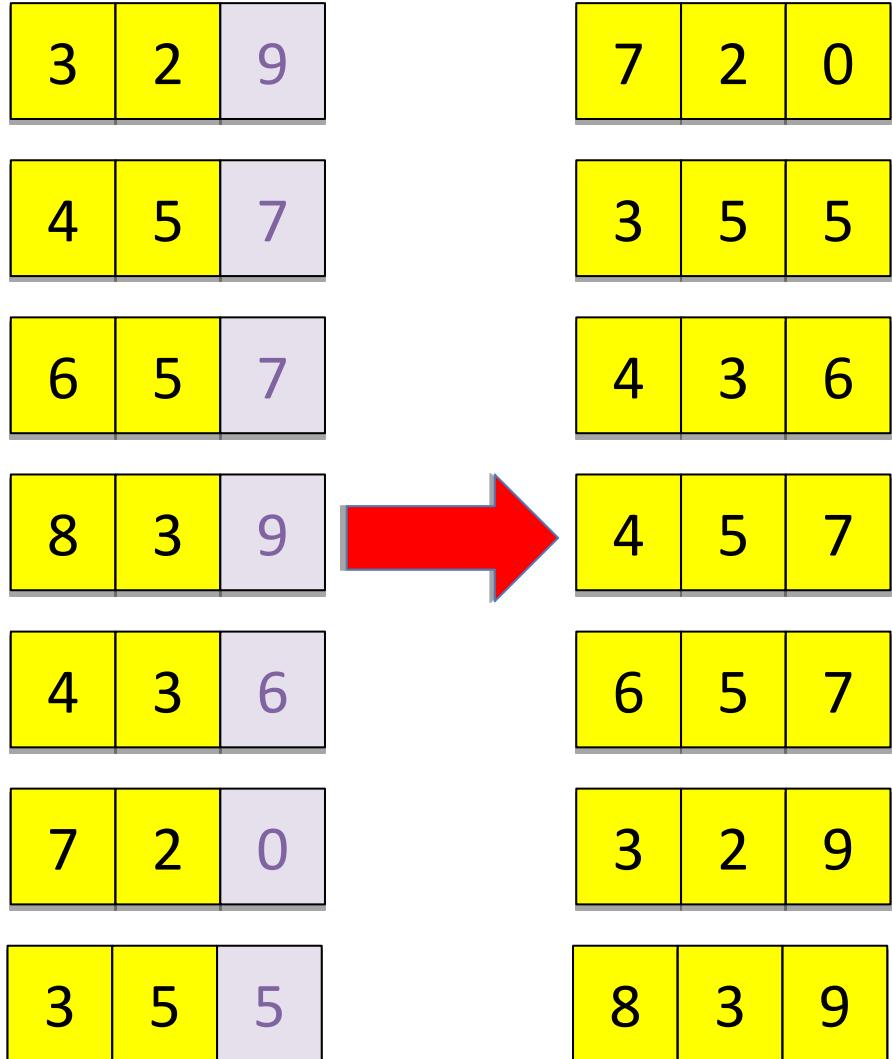
8	3	9
---	---	---

4	3	6
---	---	---

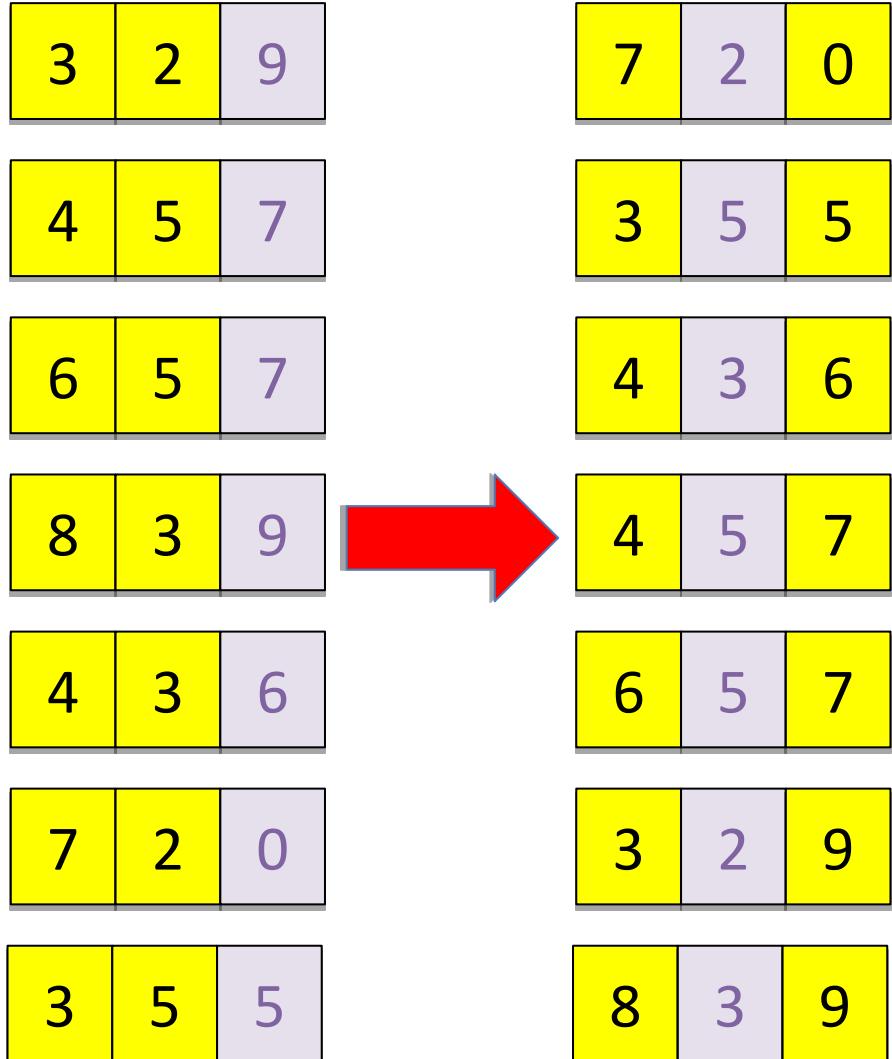
7	2	0
---	---	---

3	5	5
---	---	---

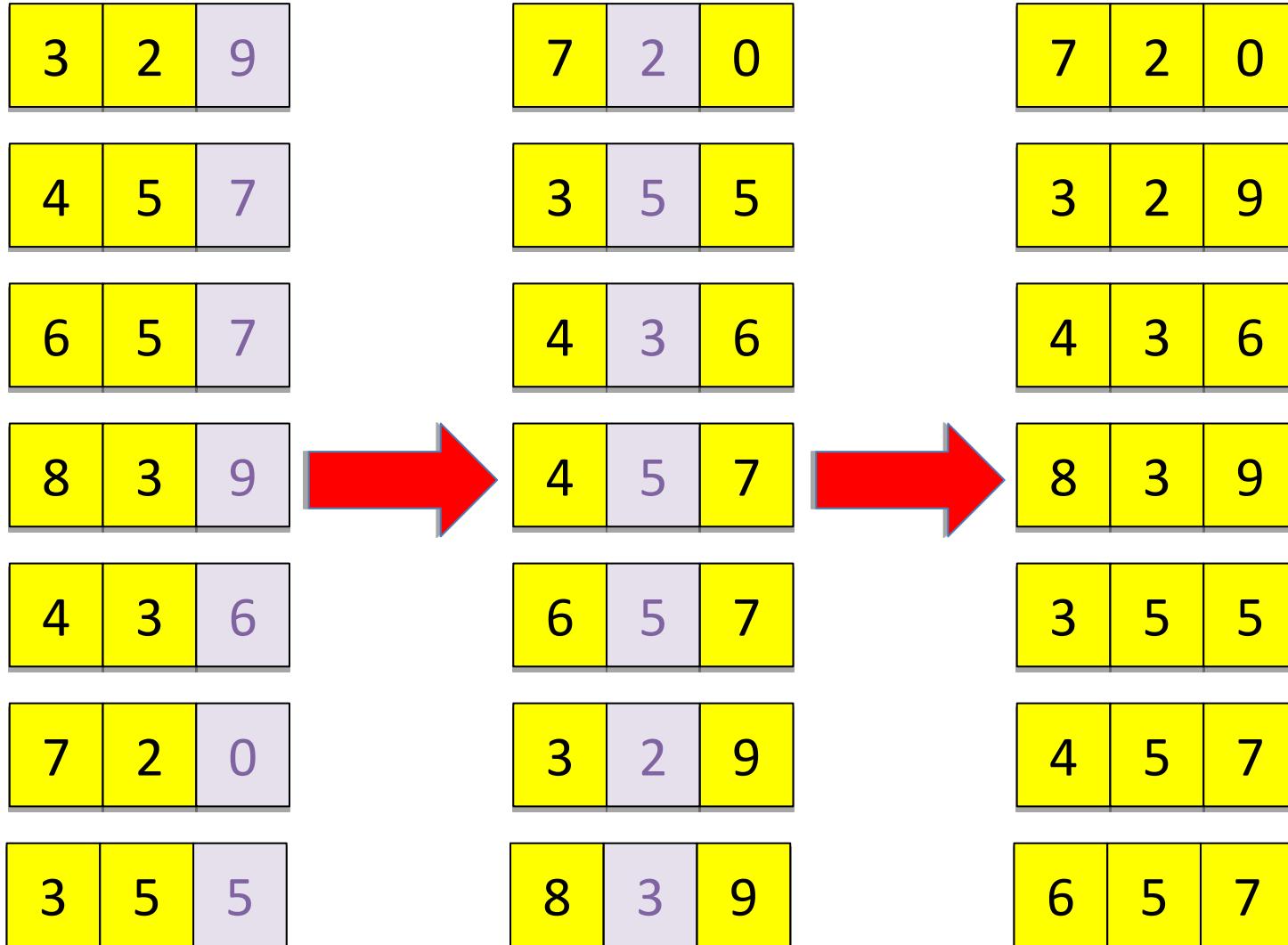
# Método RadixSort



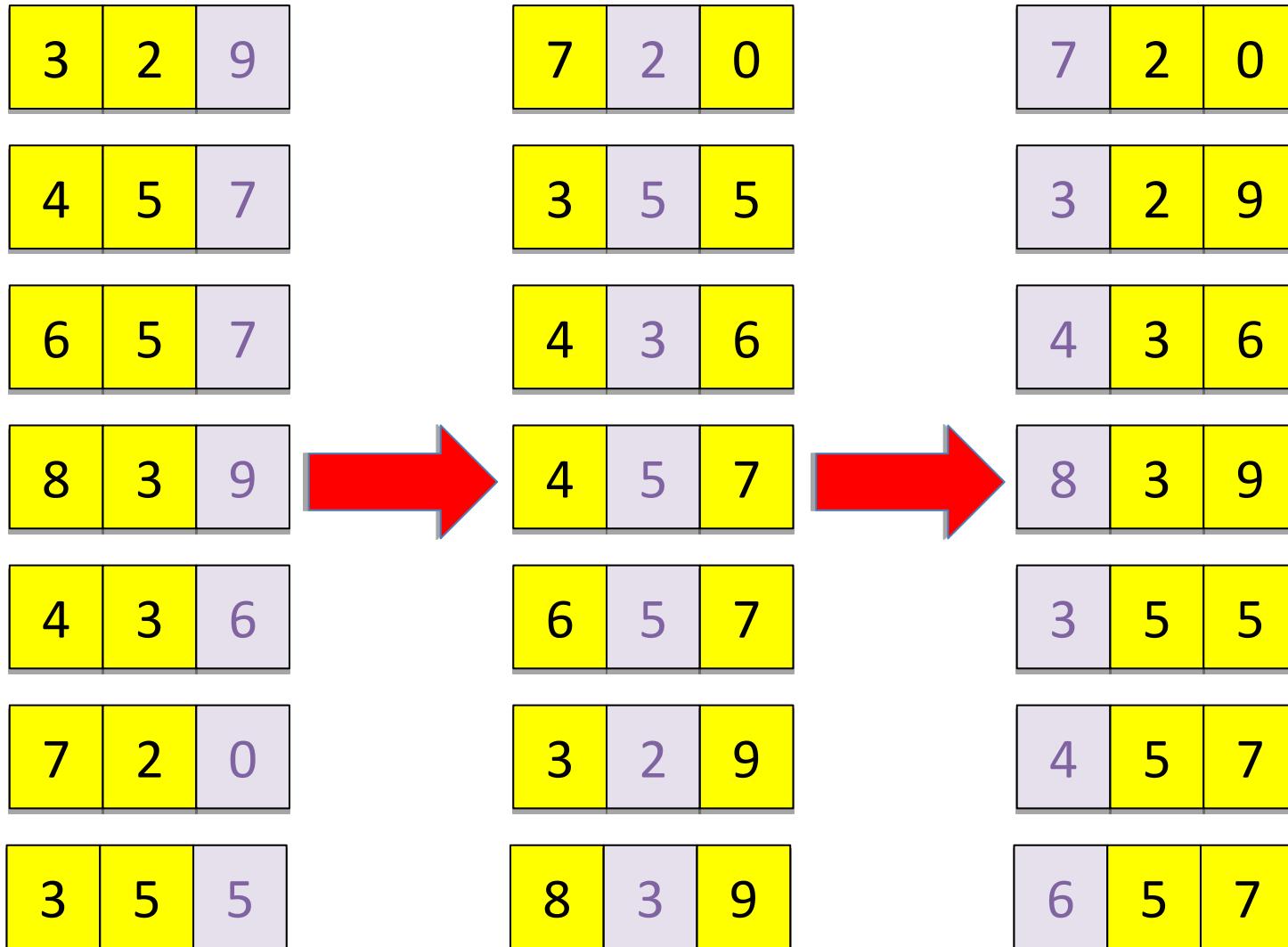
# Método RadixSort



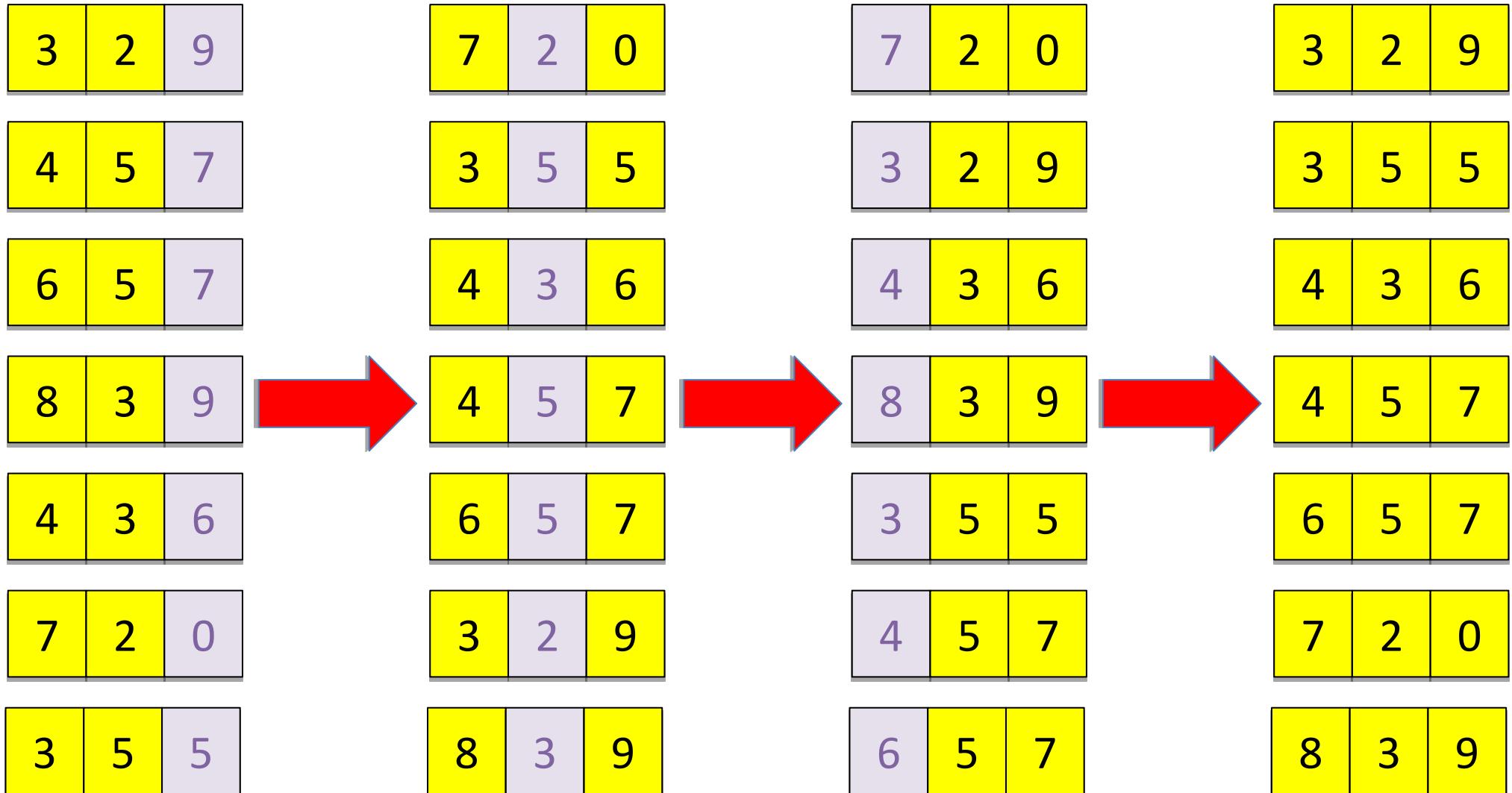
# Método RadixSort



# Método RadixSort



# Método RadixSort



# Algoritmo do *Radix-Sort*

- Radix-Sort( $A, d$ )

1. for  $i \leftarrow 1$  to  $d$
2. do use a stable sort to sort array  $A$  on digit  $i$

# Análise do Radix-Sort

- A análise depende do algoritmo de ordenação estável utilizado no passo intermediário
- Quando cada dígito varia entre 1 e  $k$ , e  $k$  não é muito grande, *Counting-sort* é uma escolha óbvia
  - Neste caso cada passagem sobre  $n$  números de  $d$  dígitos leva tempo  $\Theta(n + k)$
  - Há  $d$  passos, portanto o tempo total de Radix-sort é:  $\Theta(dn + dk)$
  - Quando  $d$  é uma constante e  $k = O(n)$ , Radix-sort roda em tempo linear ou  $O(n)$

# **Bucket-Sort (Bin-Sort ou Ordenação por Balde)**

- O algoritmo roda em tempo linear no caso médio.
- Deve-se assumir que a entrada é gerada por um processo aleatório que distribui os elementos de forma uniforme sobre o intervalo  $[0, 1)$ 
  - Pode-se converter uma chave de qualquer tipo (inteiro, alfanumérica, real, etc) para assumir valores normalizados no intervalo  $[0, 1)$

## ***Bucket-Sort (cont.)***

- A idéia de *Bucket-Sort* é dividir o intervalo  $[0, 1)$  em “n” subintervalos (*buckets*) de mesmo tamanho, e então distribuir os “n” números nos *buckets*
- Uma vez que as entradas são uniformemente distribuídas espera-se que a quantidade de elementos por bucket seja próxima do uniforme
- Para produzir a saída ordenada, basta ordenar os números em cada *bucket* e depois colocá-los em sequência.

## ***Bucket-Sort (cont.)***

- O algoritmo tem as seguintes características:
  - assume que a entrada é um vetor  $A[1\dots n]$
  - utiliza um vetor auxiliar  $B[0 \dots n-1]$  de listas ligadas
  - assume que existe um mecanismo para tratamento das listas

# Algoritmo Bucket-Sort

- BUCKET-SORT(A)

1.  $n \leftarrow \text{length}[A]$
2. for  $i \leftarrow 1$  to  $n$ 
  - 3. do insert  $A[i]$  into list  $B[ nA[i] ]$
  - 4. for  $i \leftarrow 0$  to  $n - 1$ 
    - 5. do sort list  $B[i]$  with insertion sort
  - 6. Concatenate the lists  $B[0], B[1], \dots, B[n - 1]$  together in order

# Método BucketSort

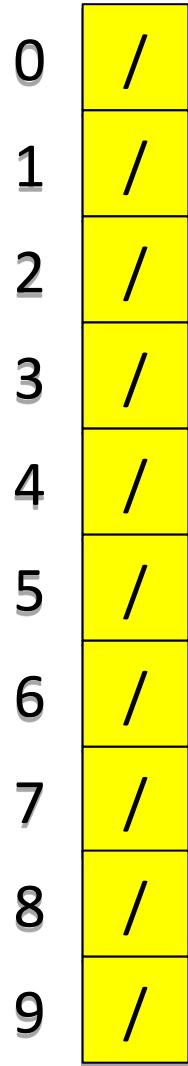
A	0,78	0,17	0,39	0,26	0,72	0,94	0,21	0,12	0,23	0,68
---	------	------	------	------	------	------	------	------	------	------

# Método BucketSort

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

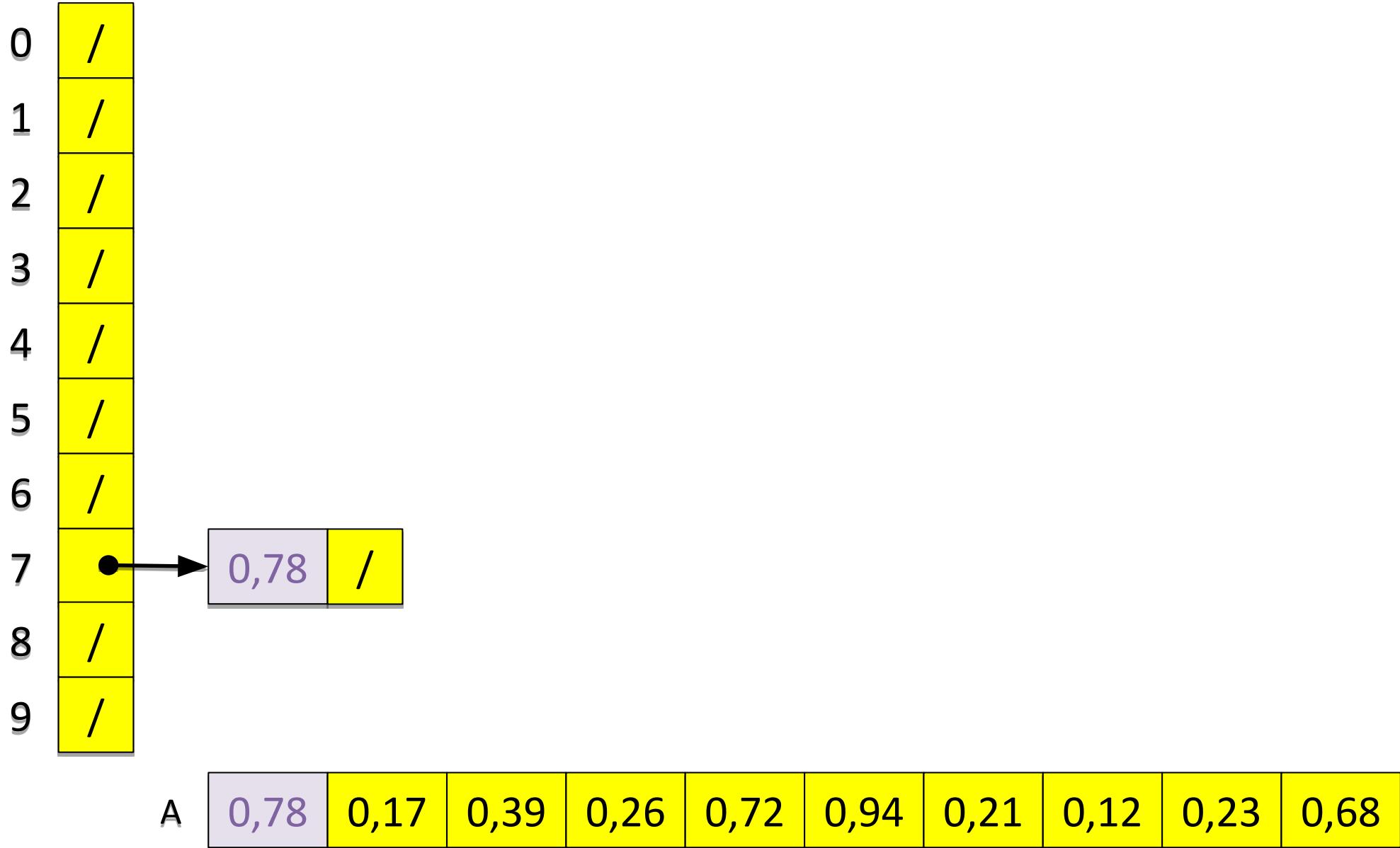
A	0,78	0,17	0,39	0,26	0,72	0,94	0,21	0,12	0,23	0,68
---	------	------	------	------	------	------	------	------	------	------

# Método BucketSort

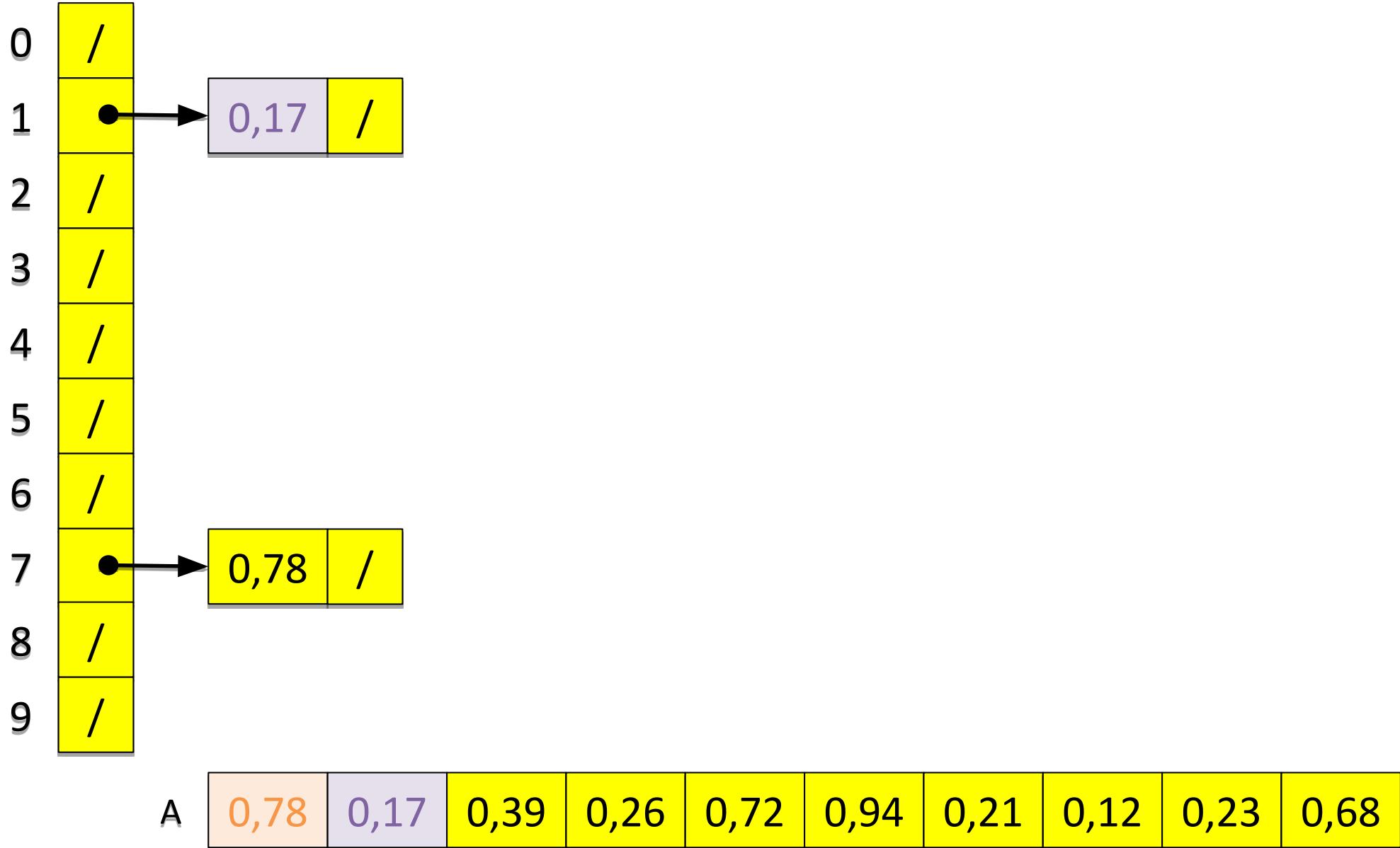


A	0,78	0,17	0,39	0,26	0,72	0,94	0,21	0,12	0,23	0,68
---	------	------	------	------	------	------	------	------	------	------

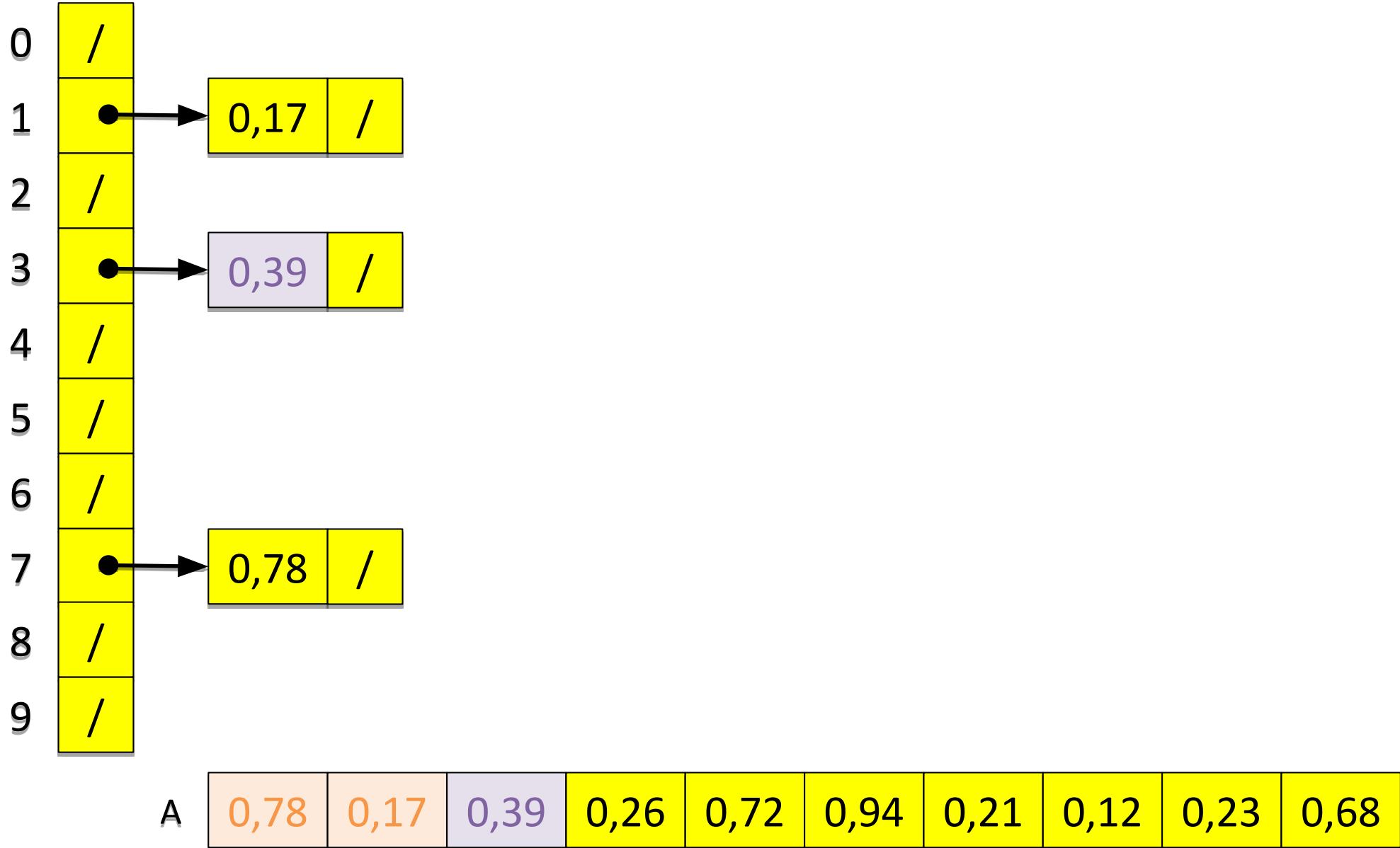
# Método BucketSort



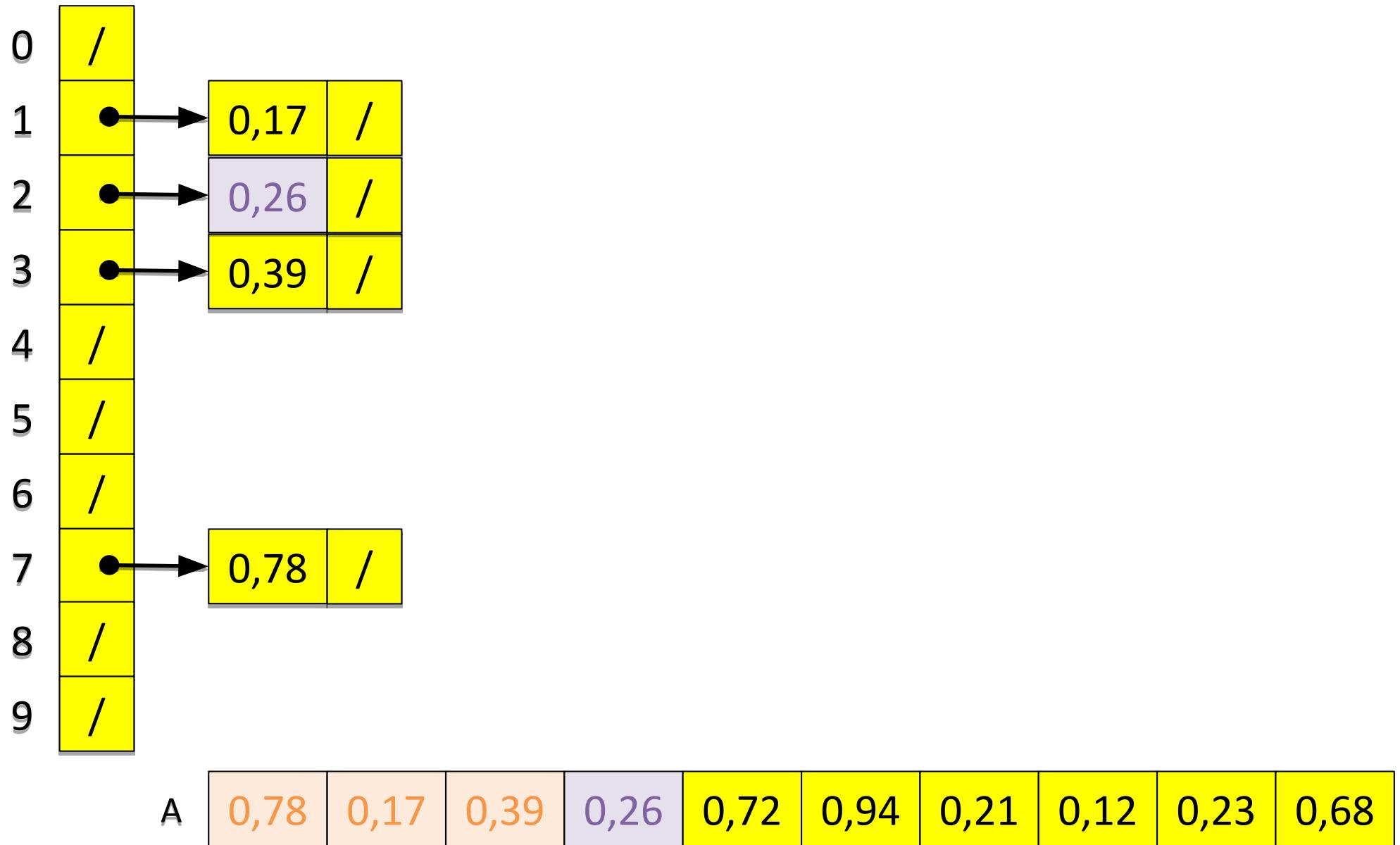
# Método BucketSort



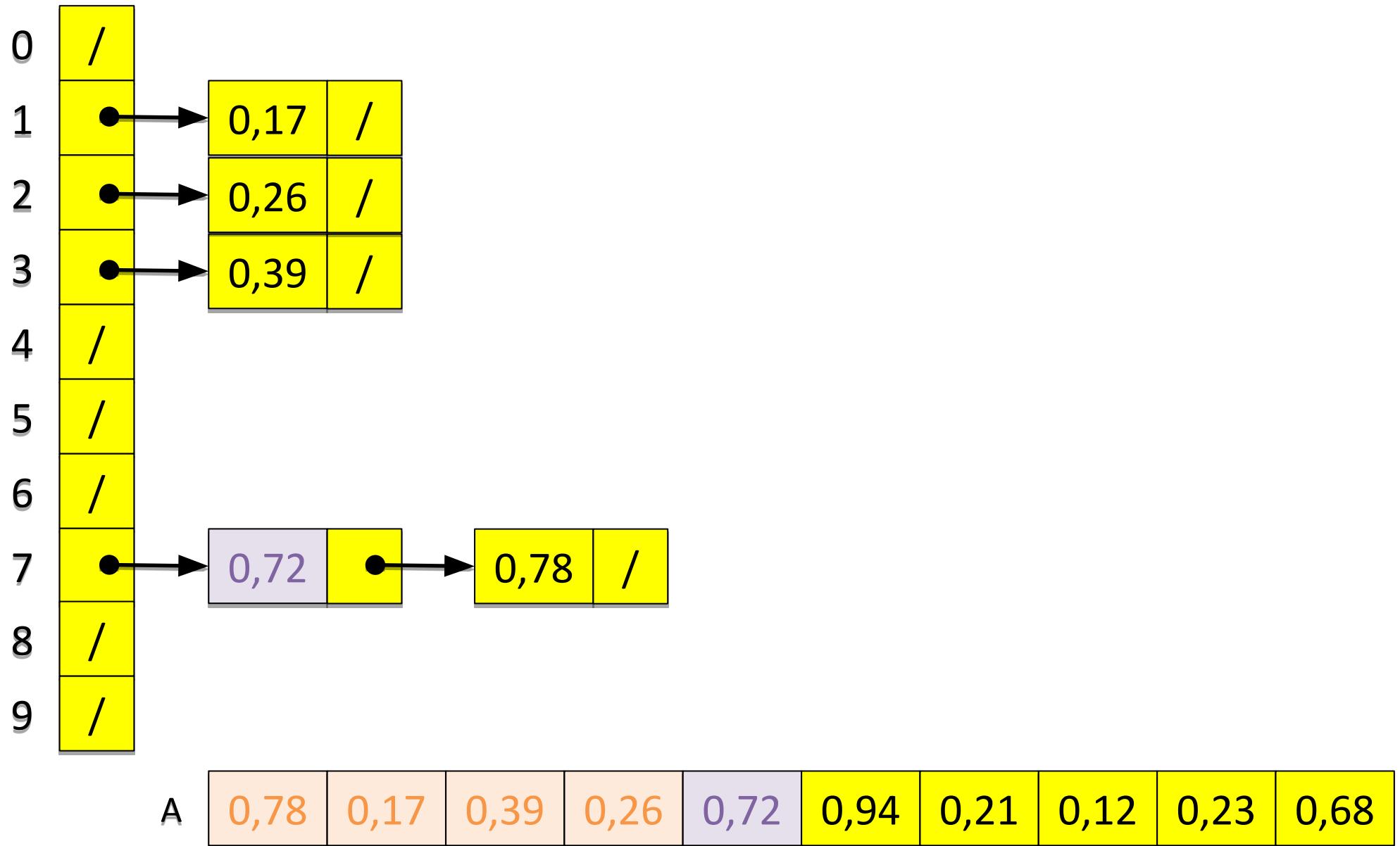
# Método BucketSort



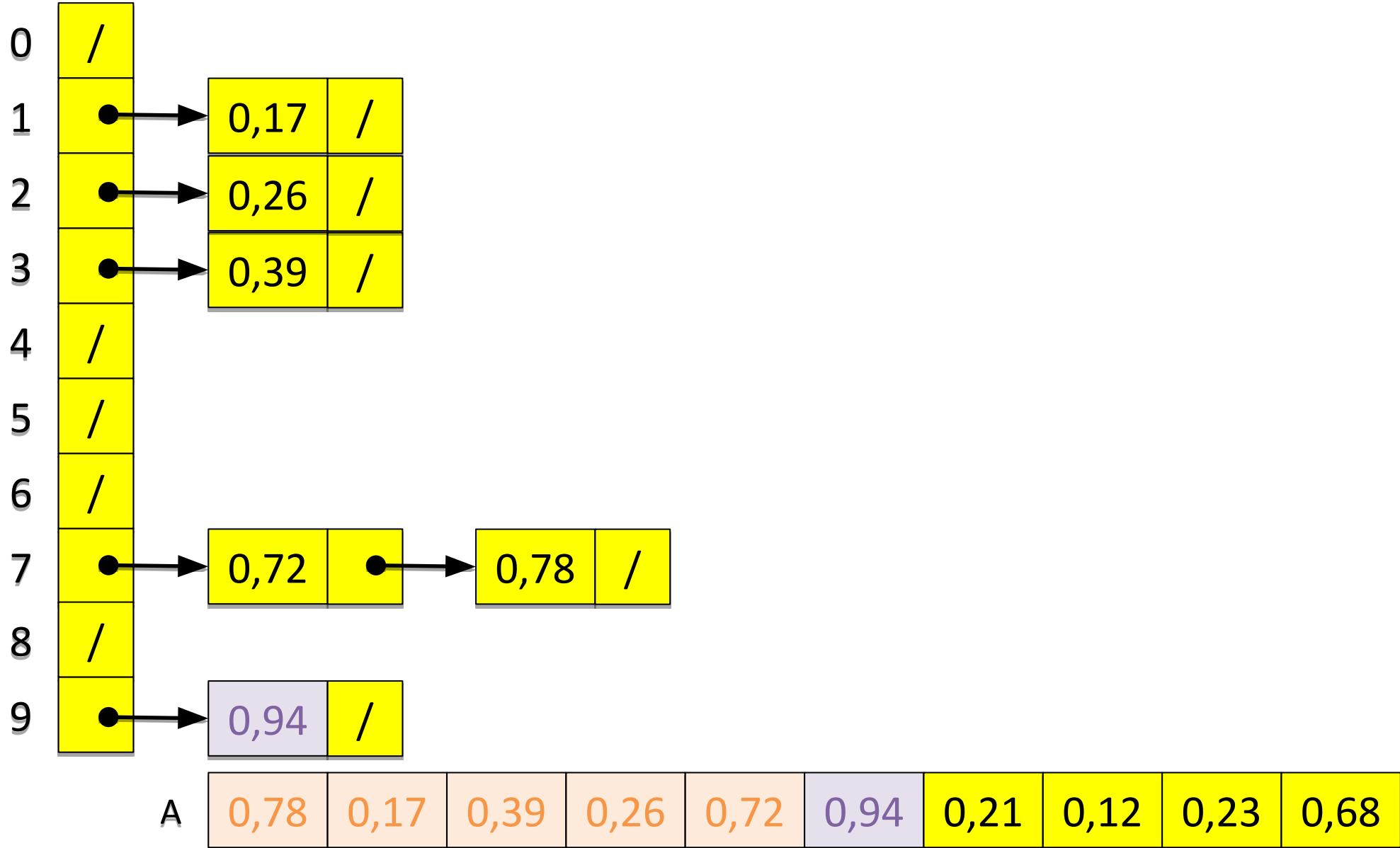
# Método BucketSort



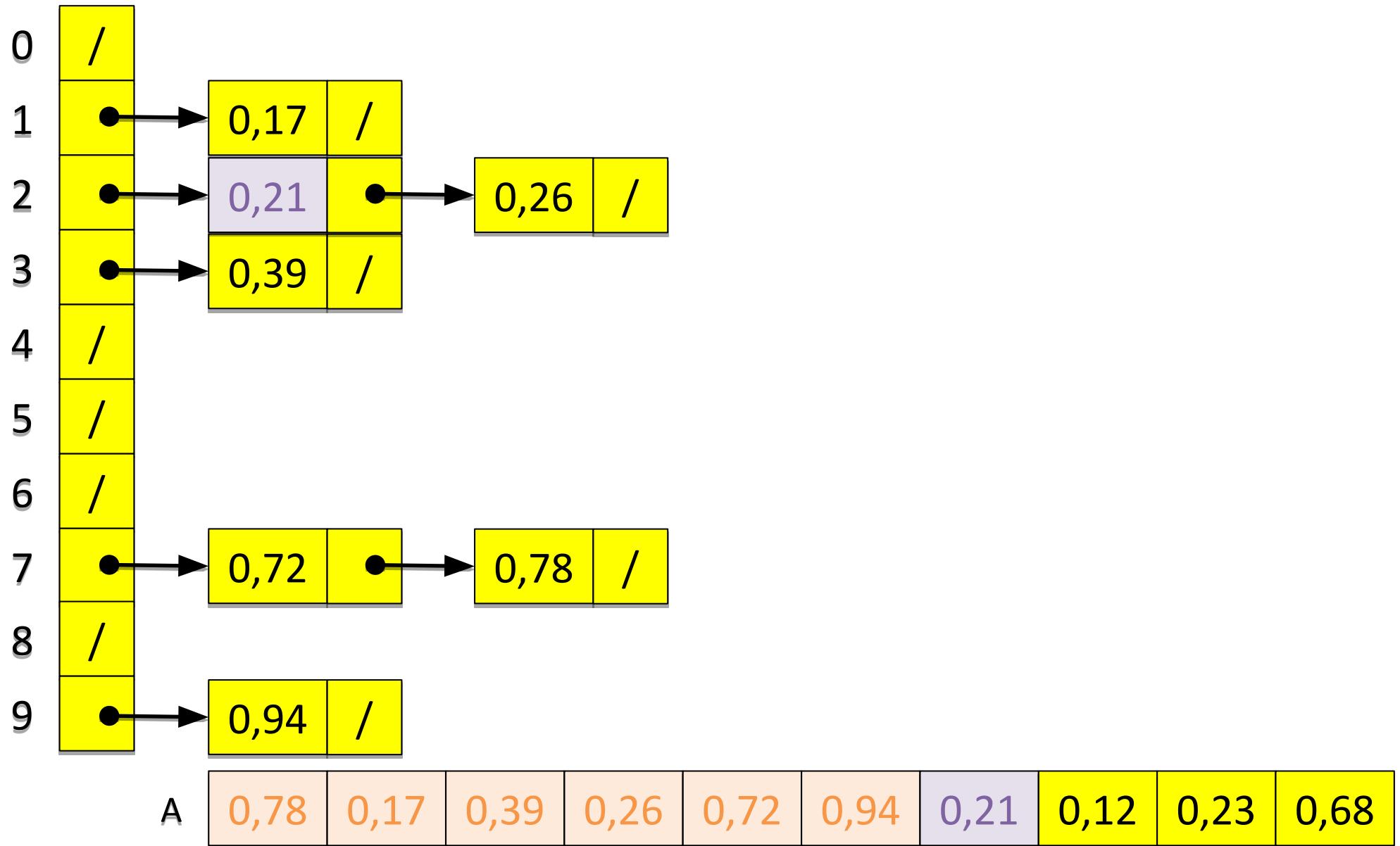
# Método BucketSort



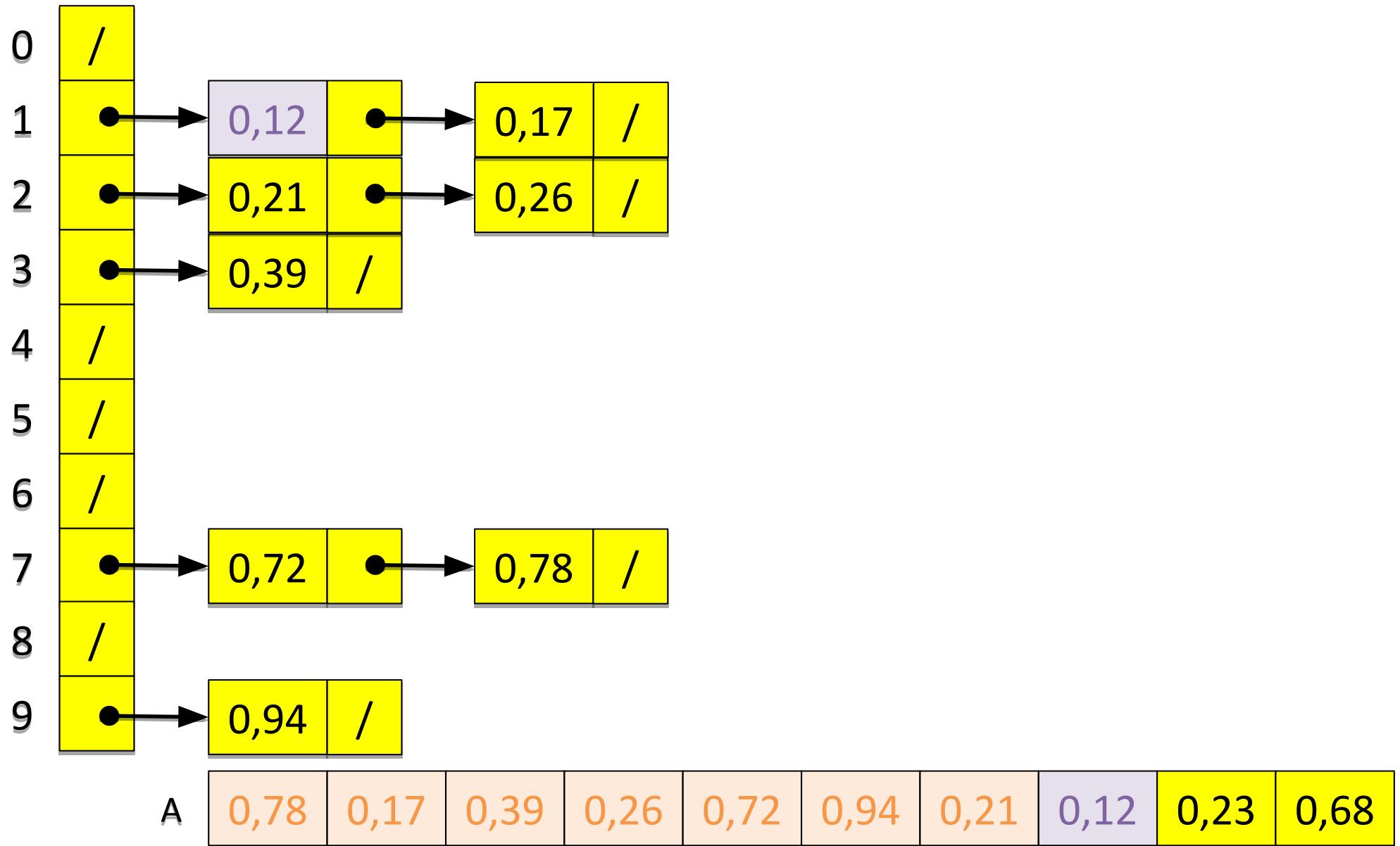
# Método BucketSort



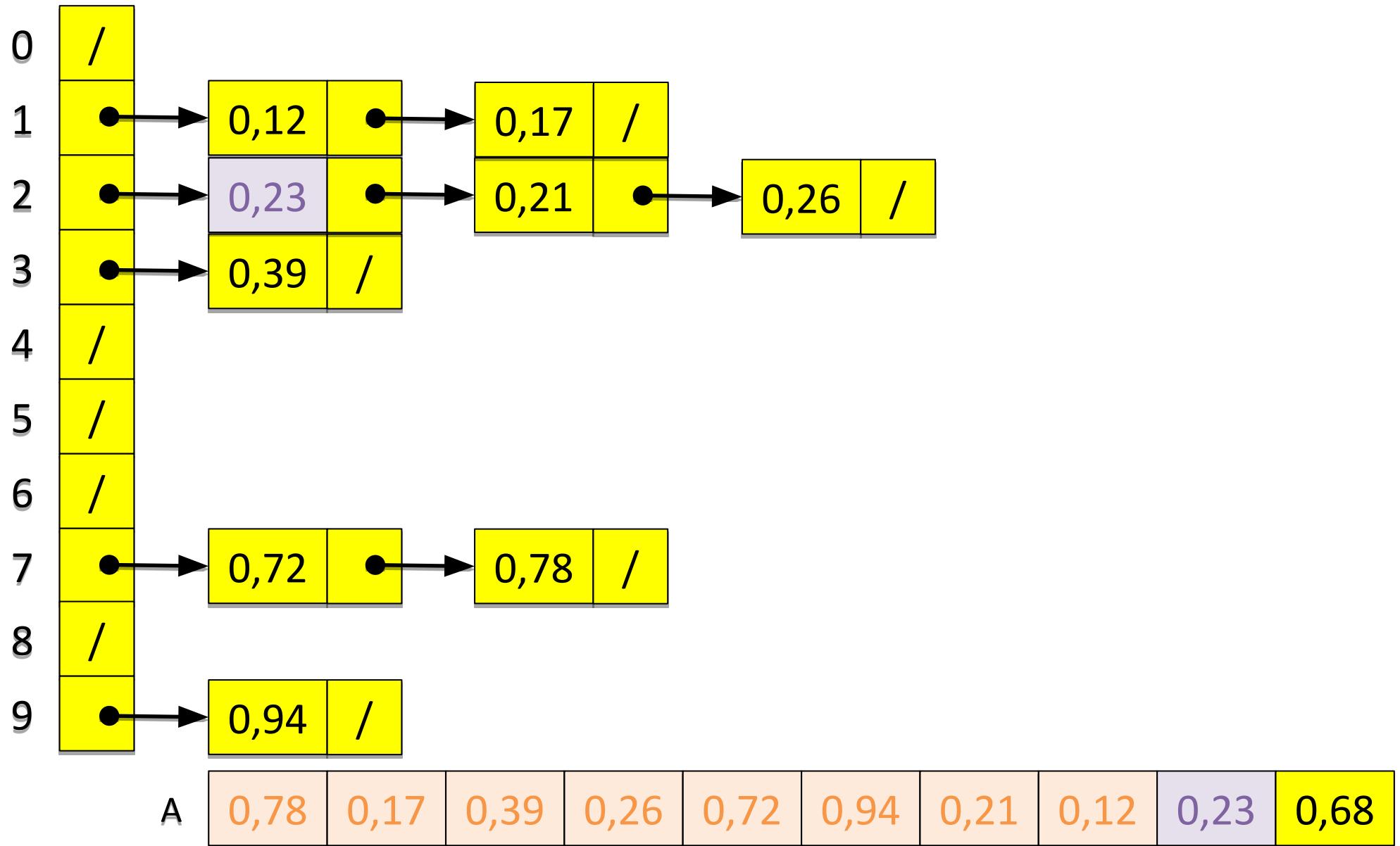
# Método BucketSort



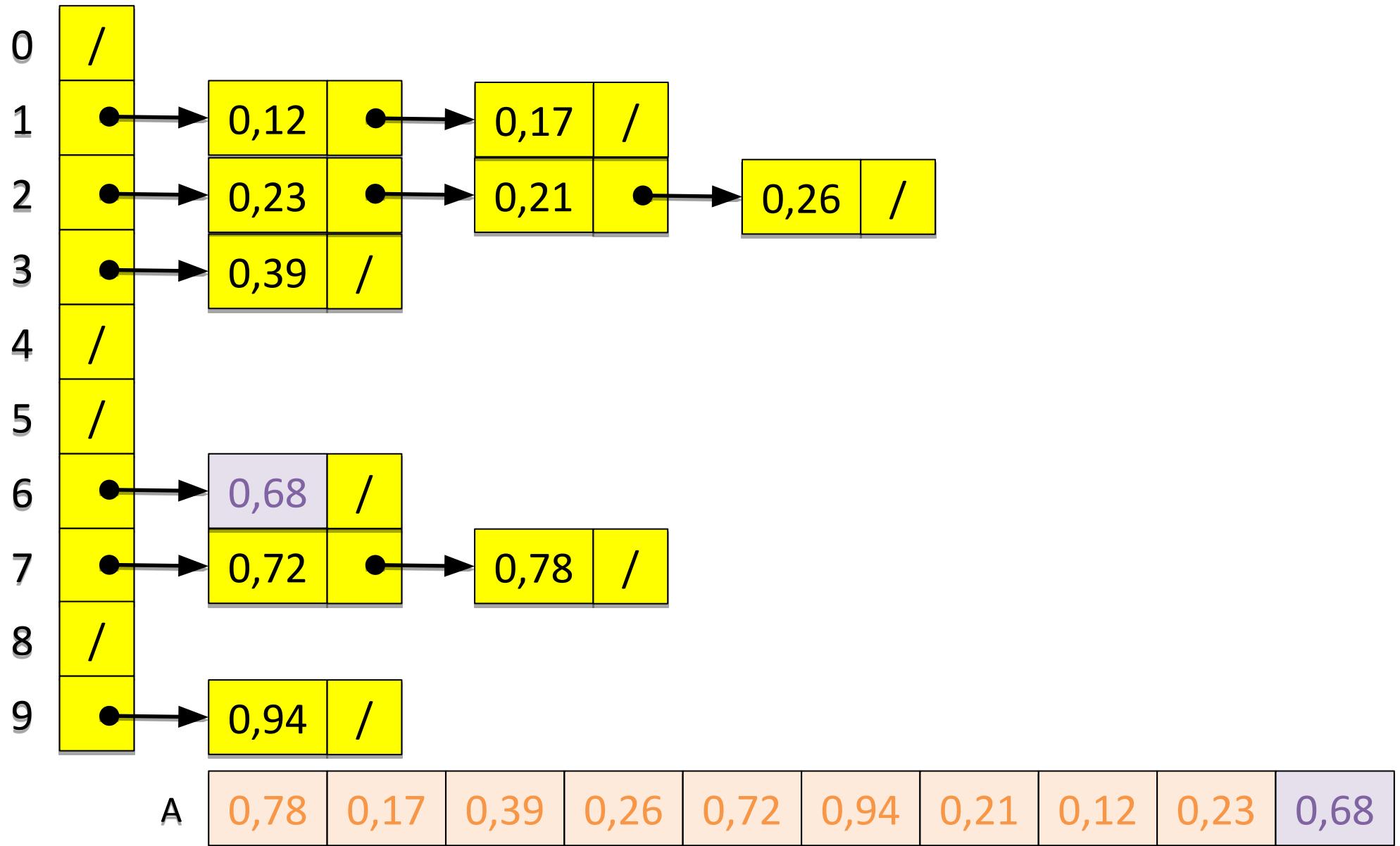
# Método BucketSort



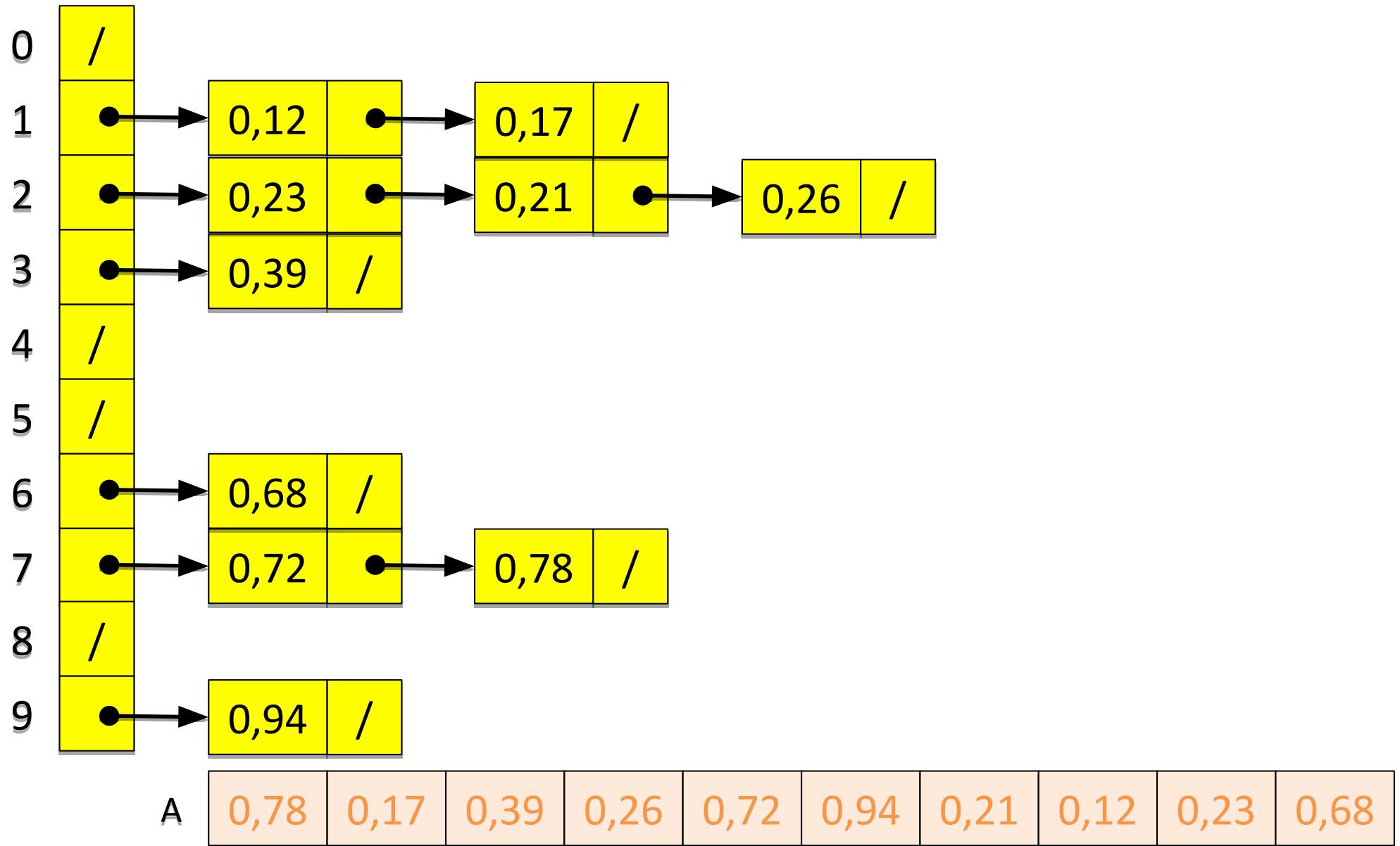
# Método BucketSort



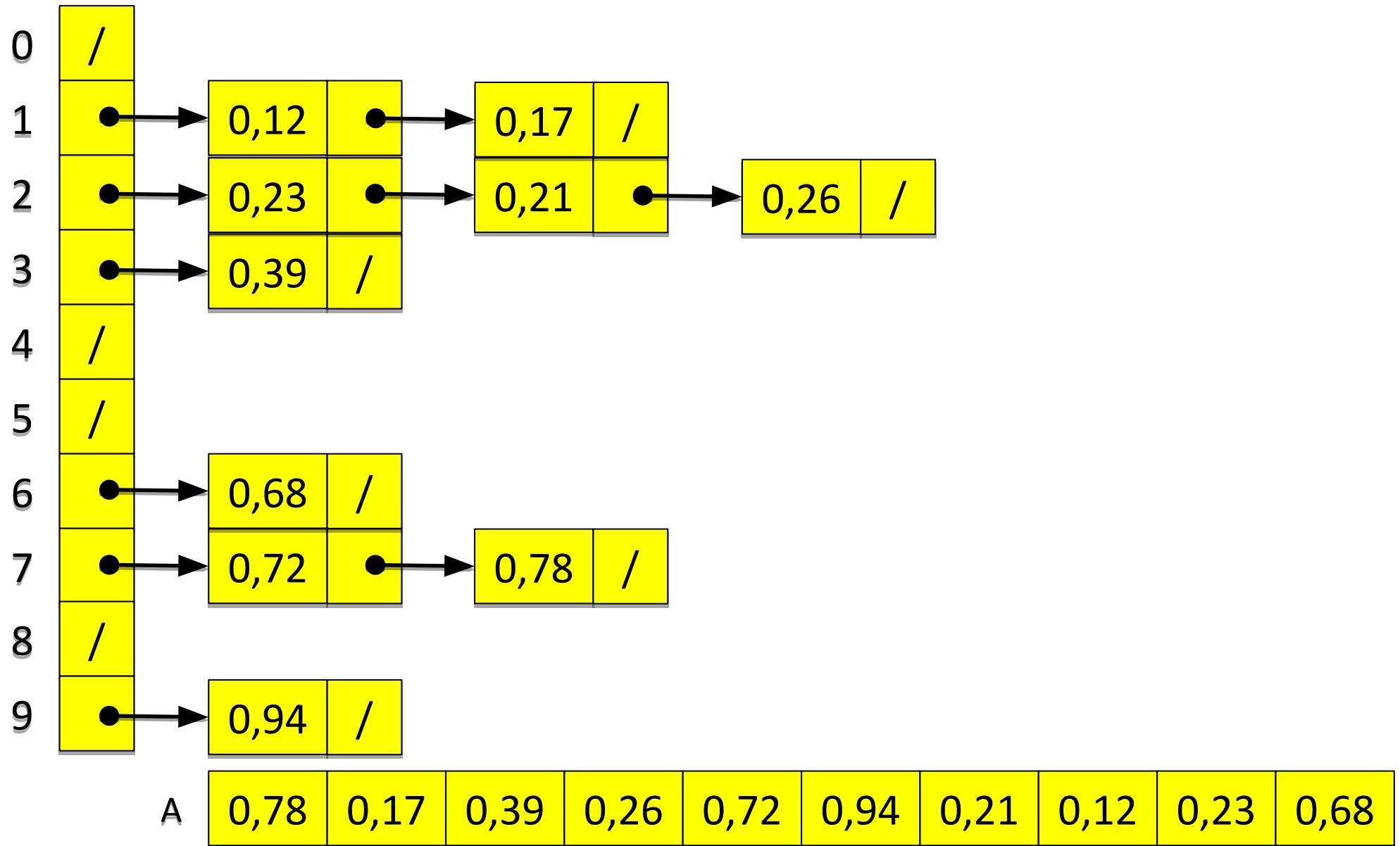
# Método BucketSort



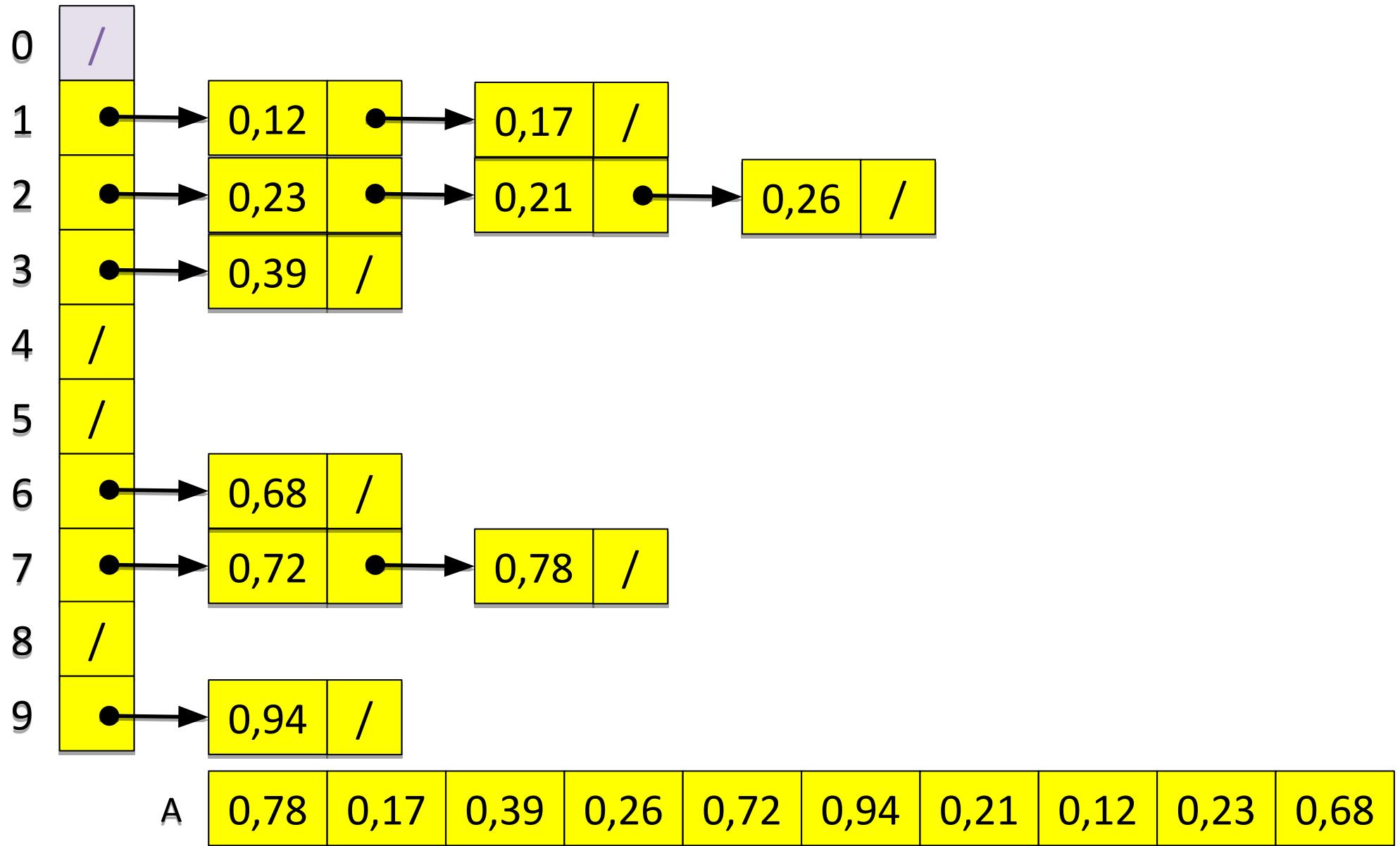
# Método BucketSort



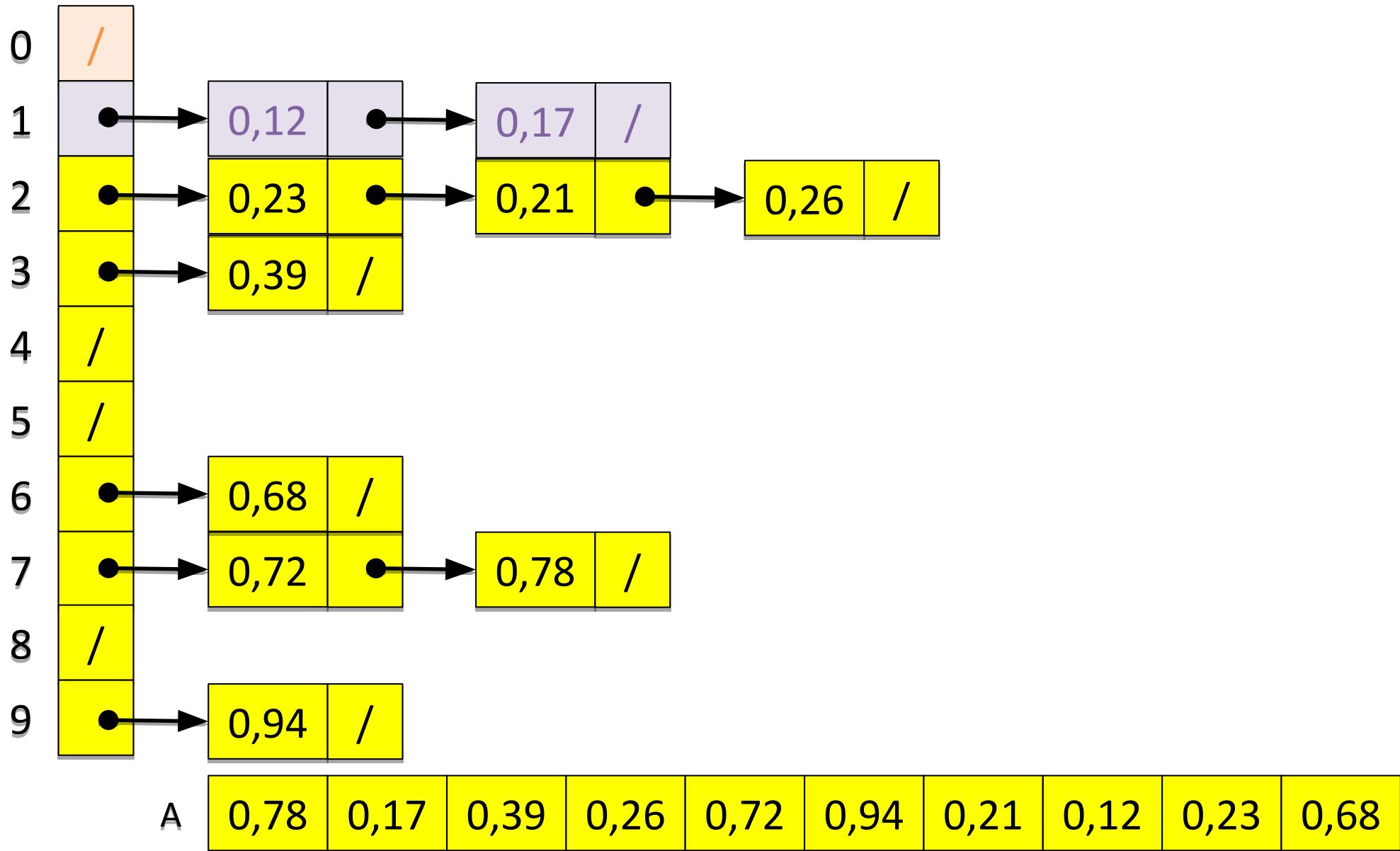
# Método BucketSort



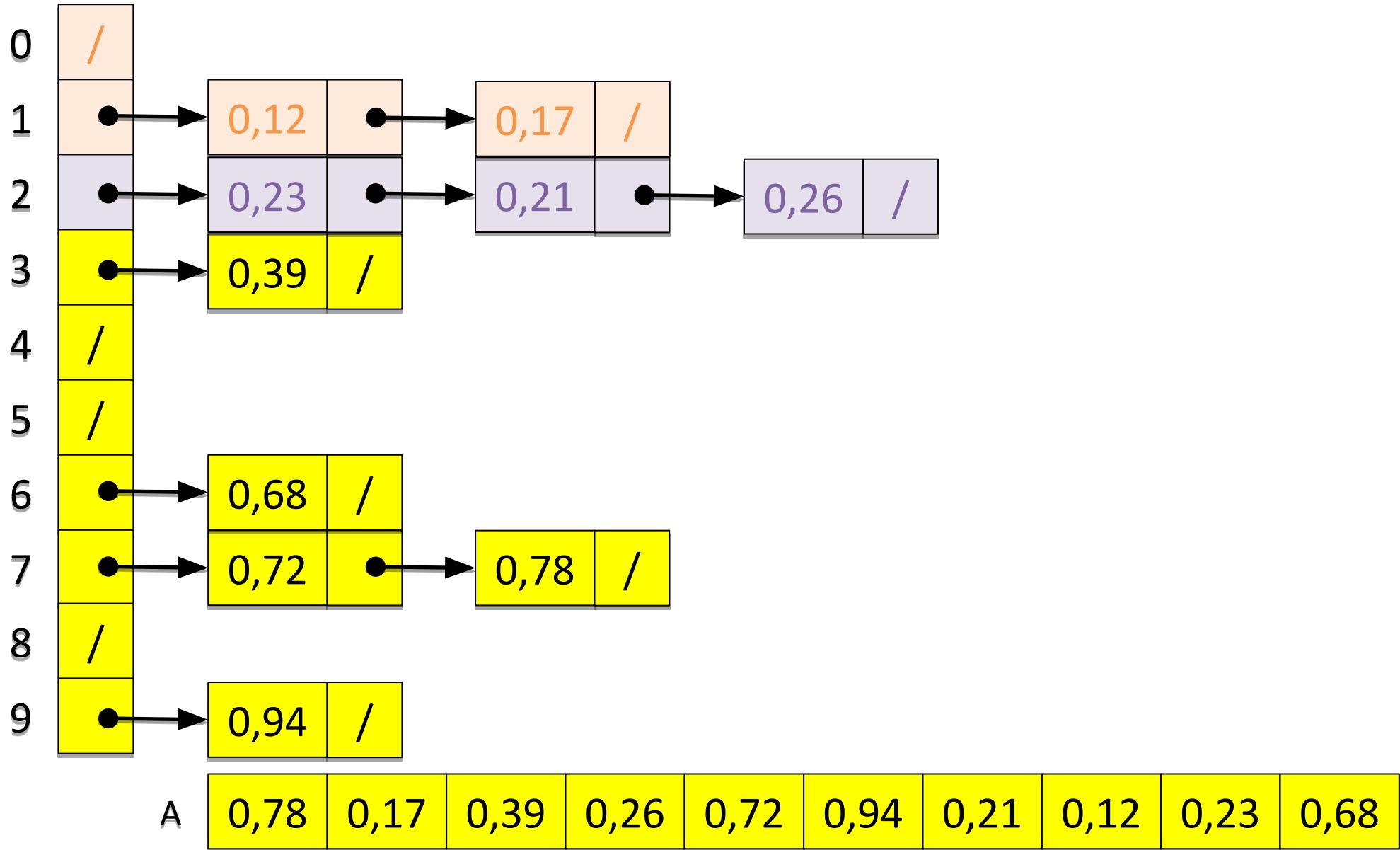
# Método BucketSort



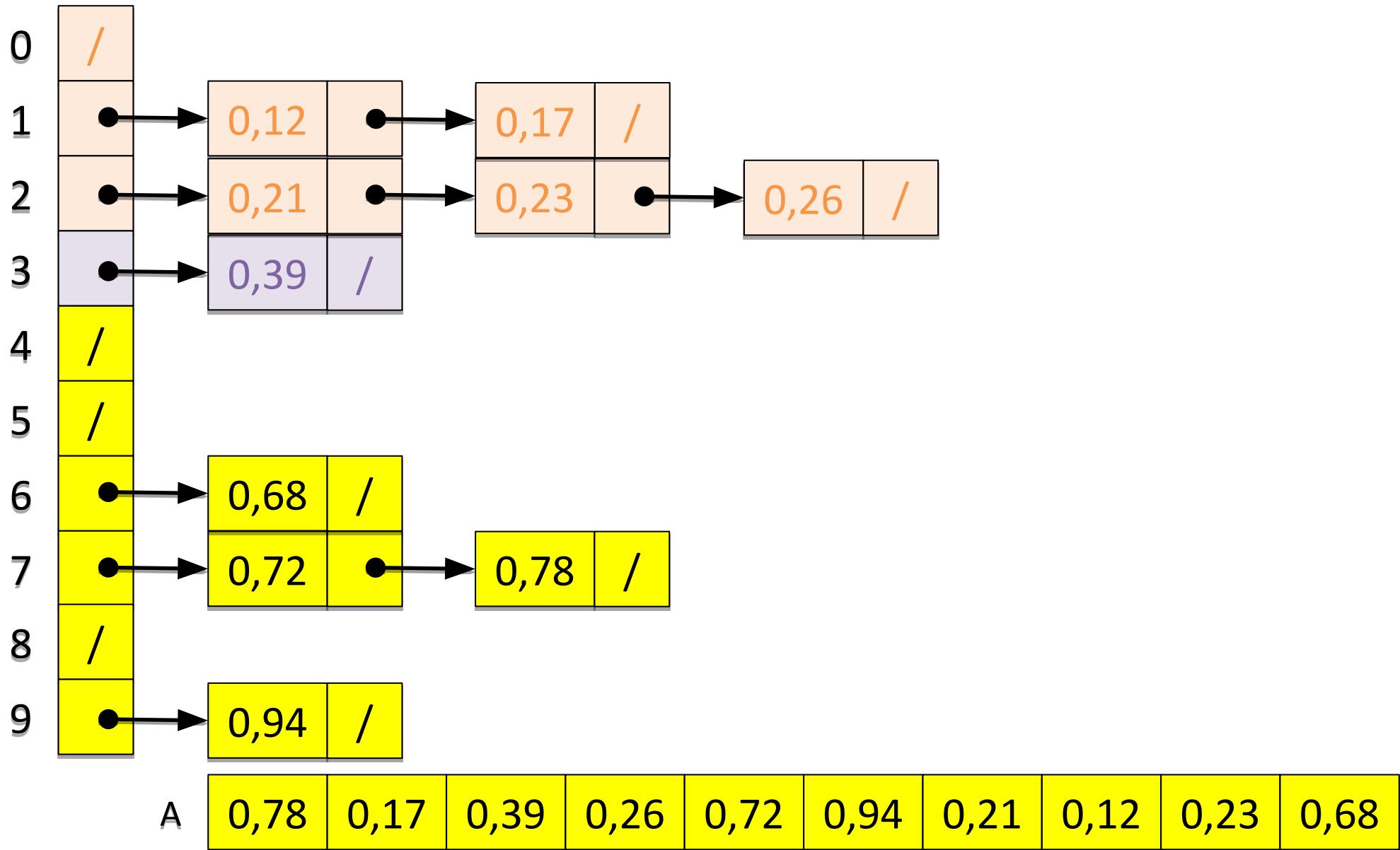
# Método BucketSort



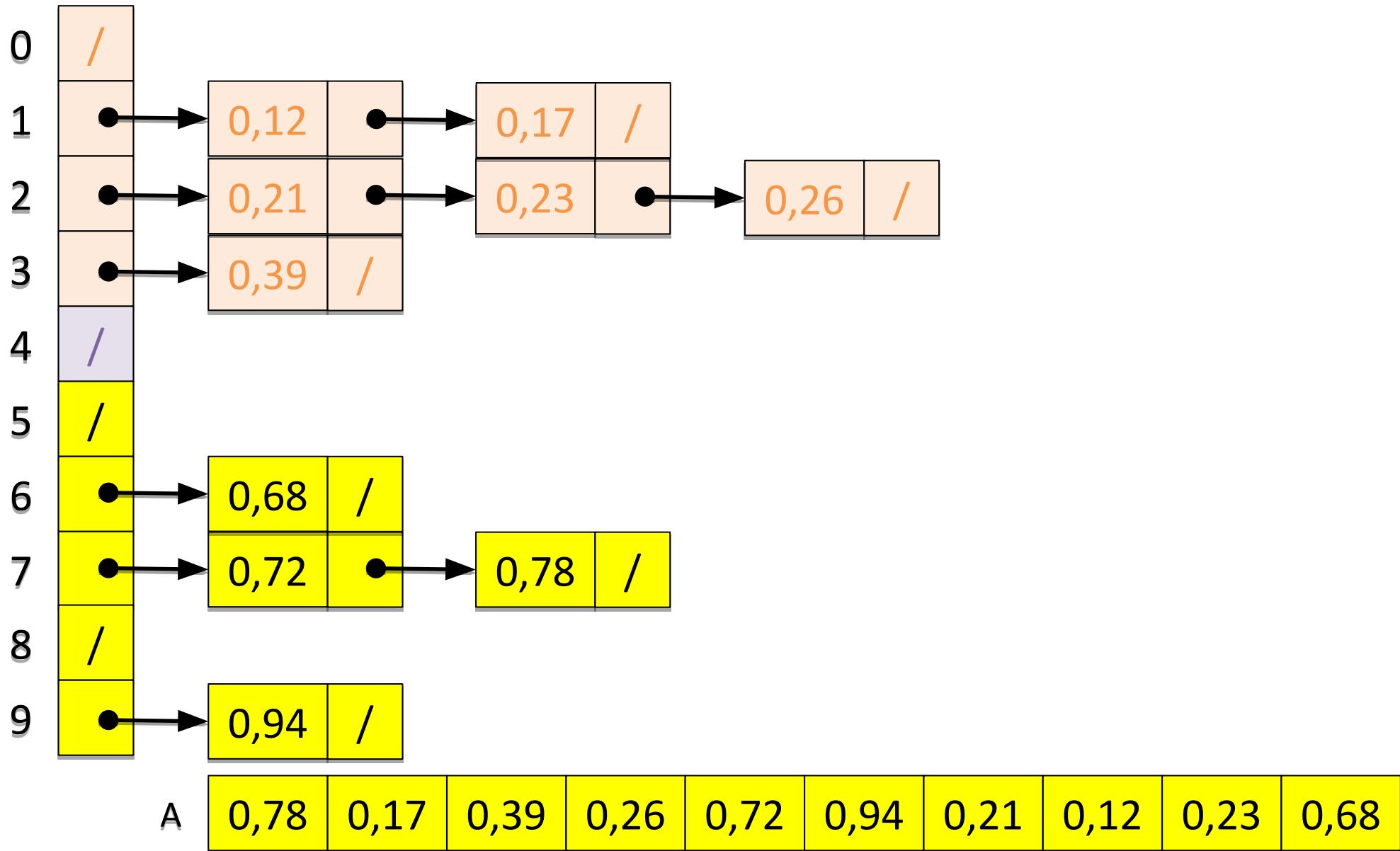
# Método BucketSort



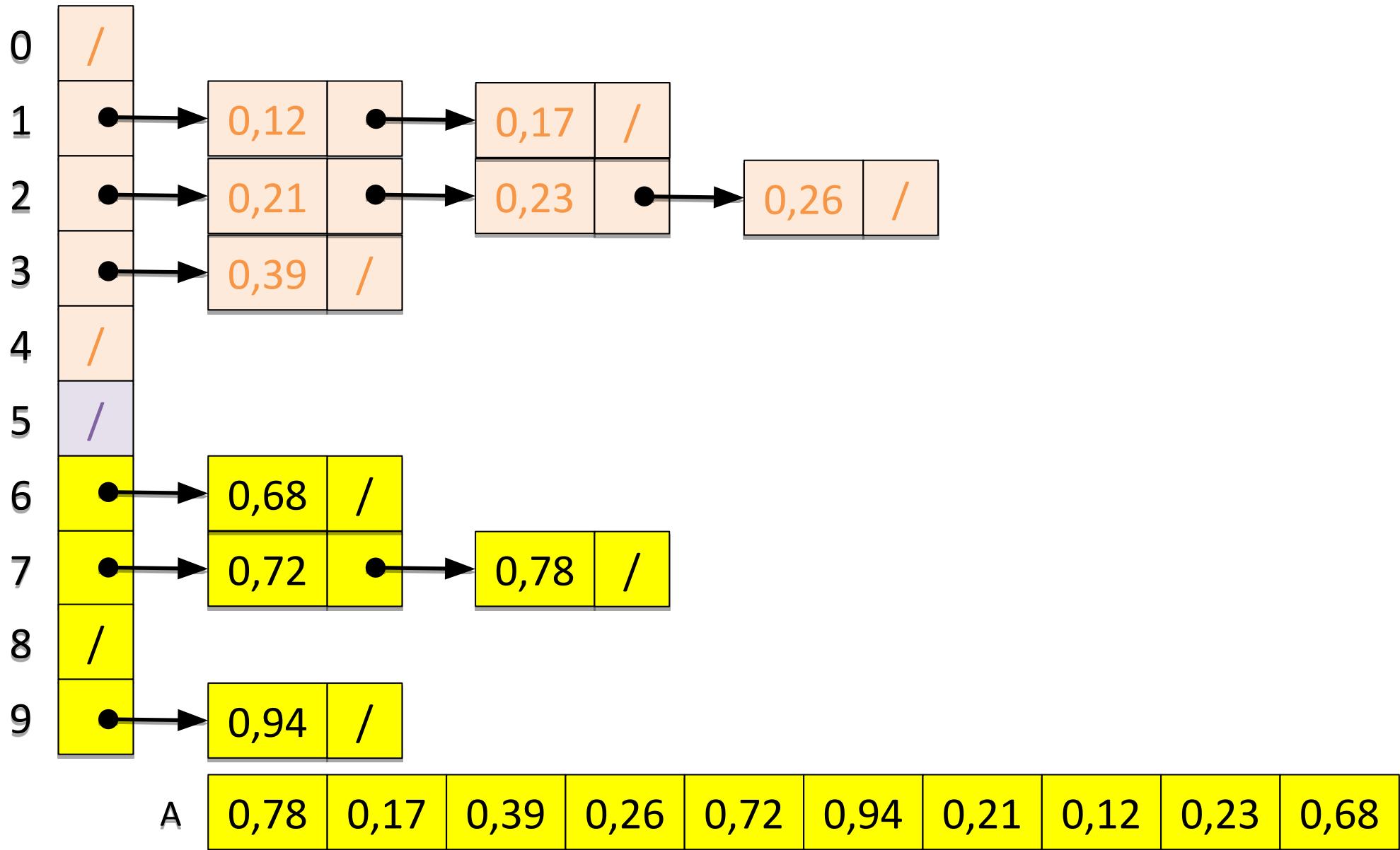
# Método BucketSort



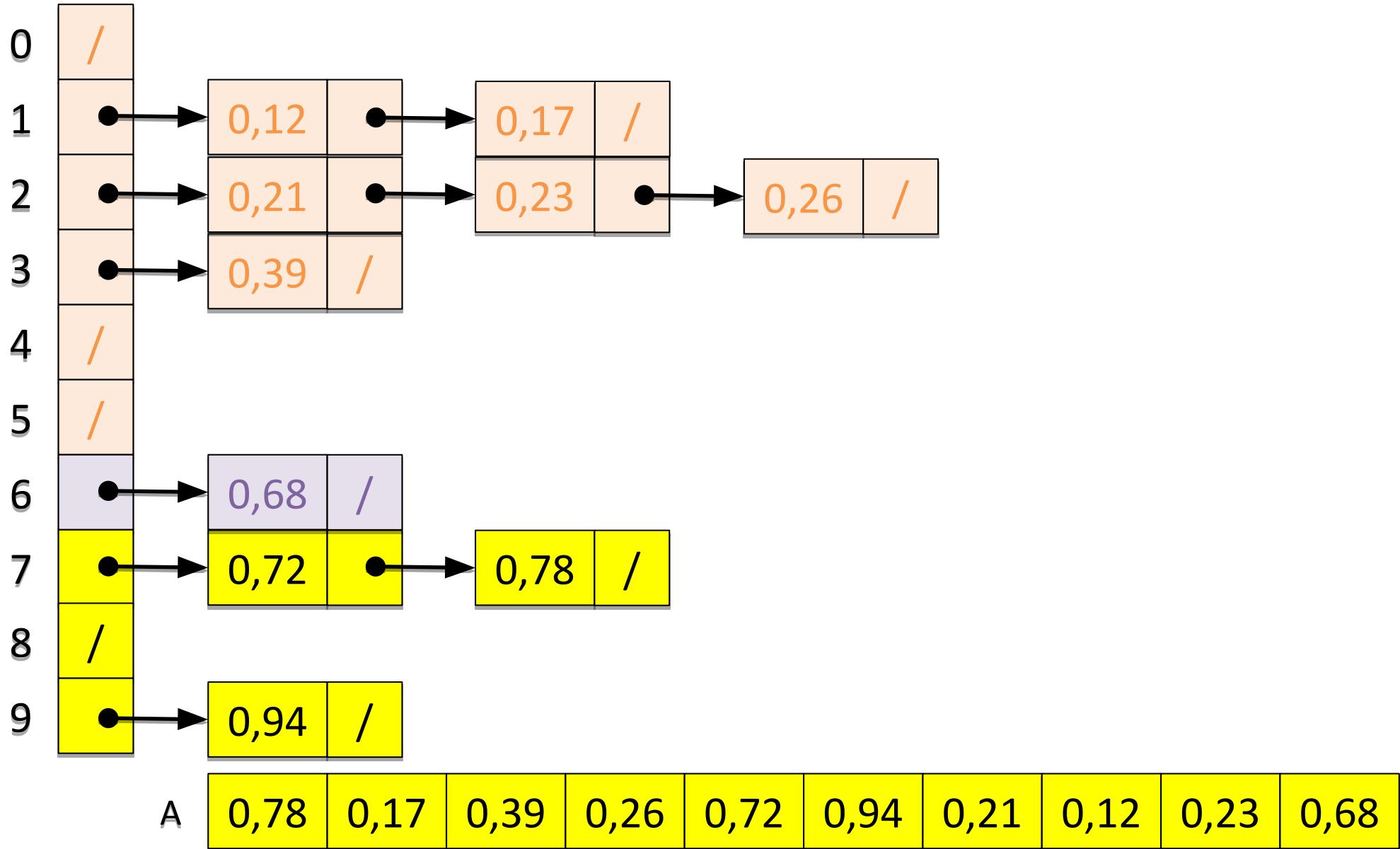
# Método BucketSort



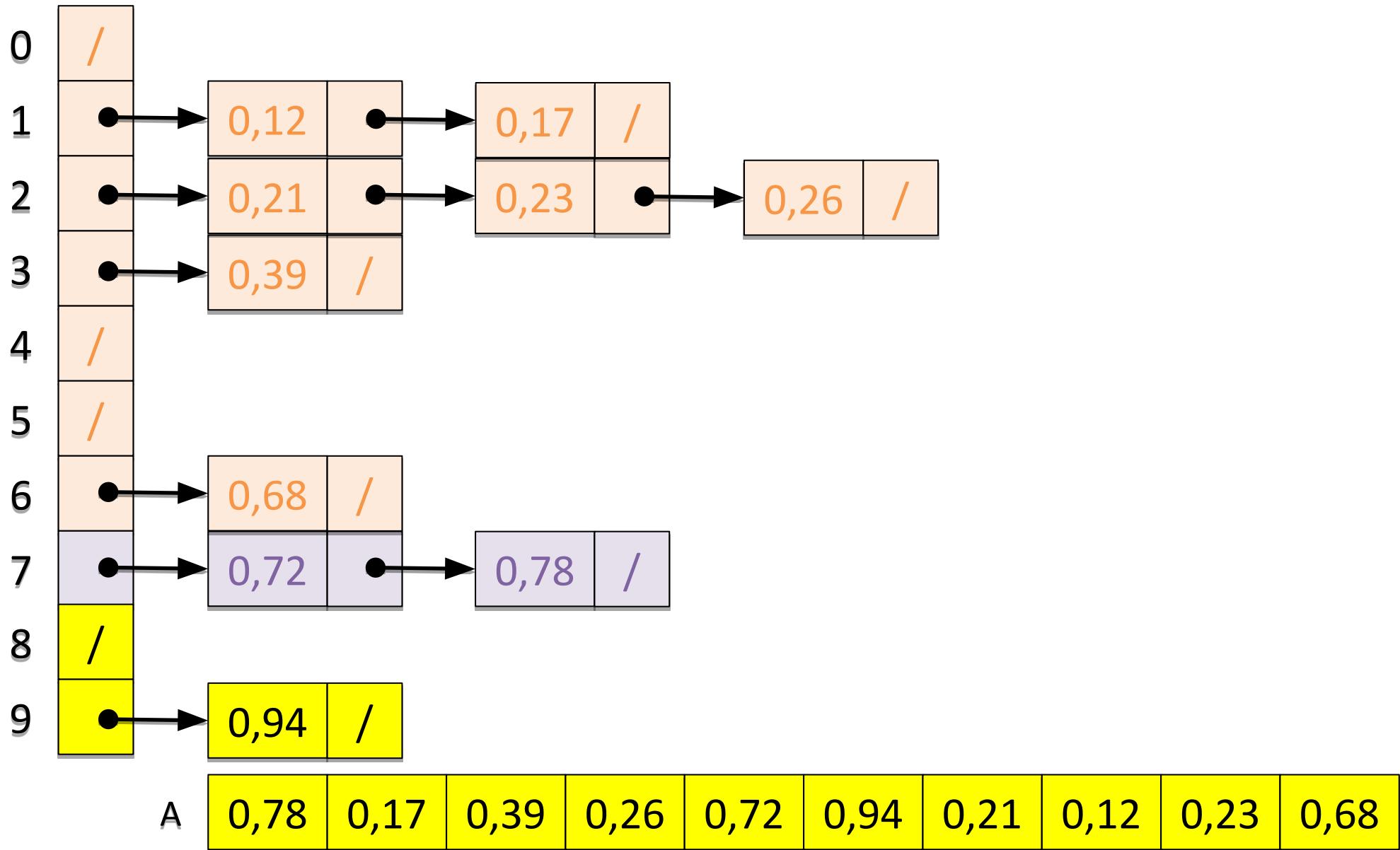
# Método BucketSort



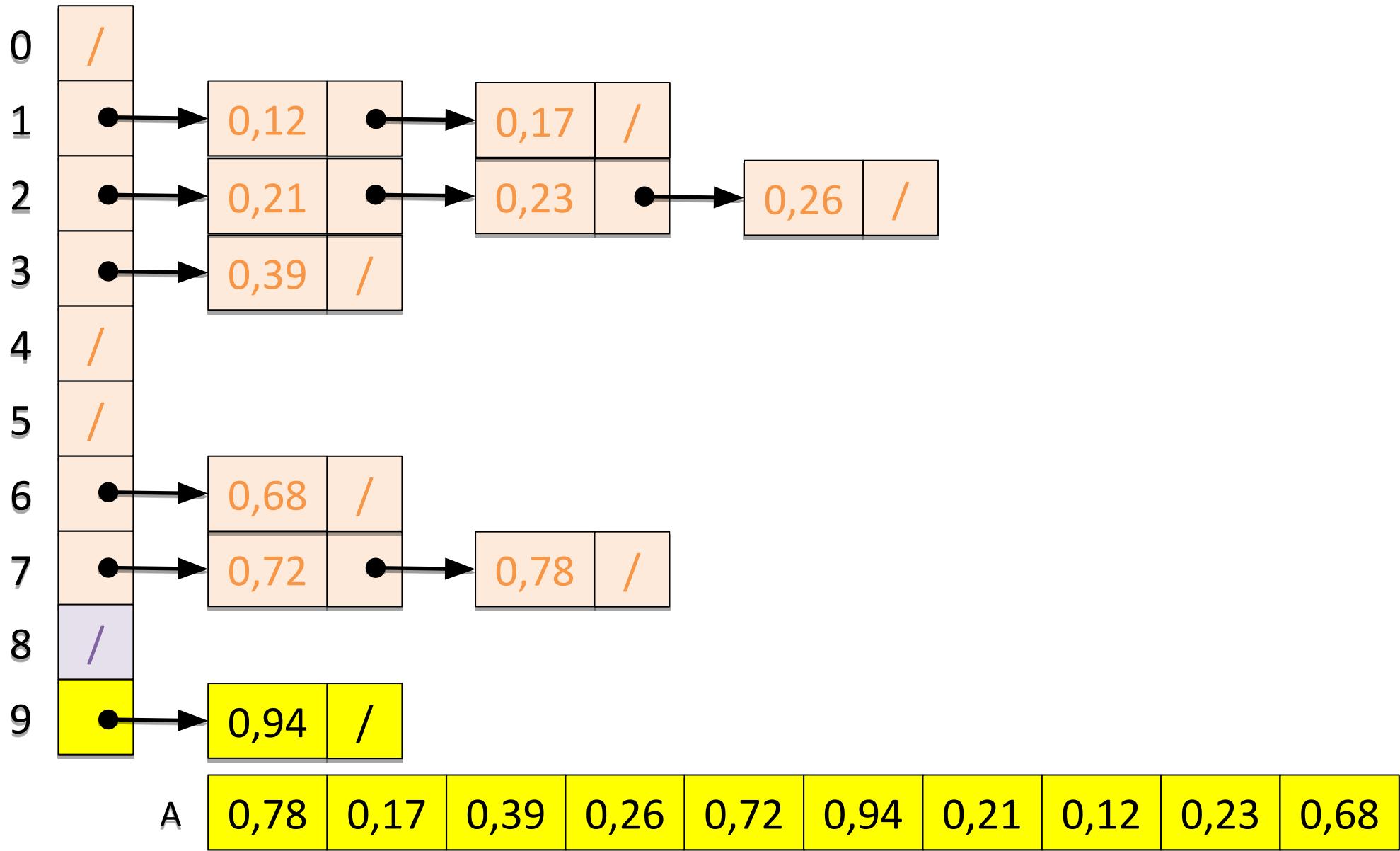
# Método BucketSort



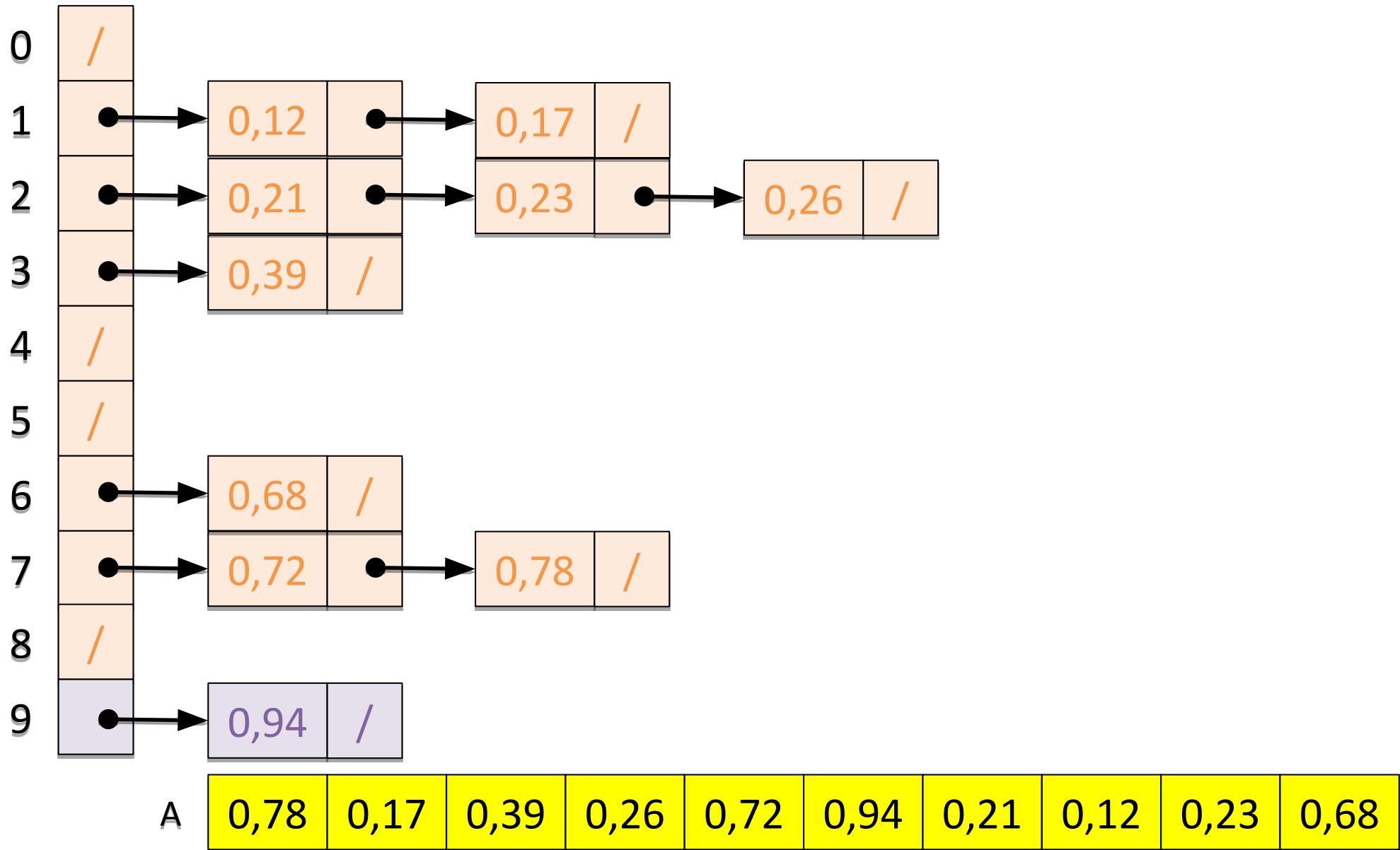
# Método BucketSort



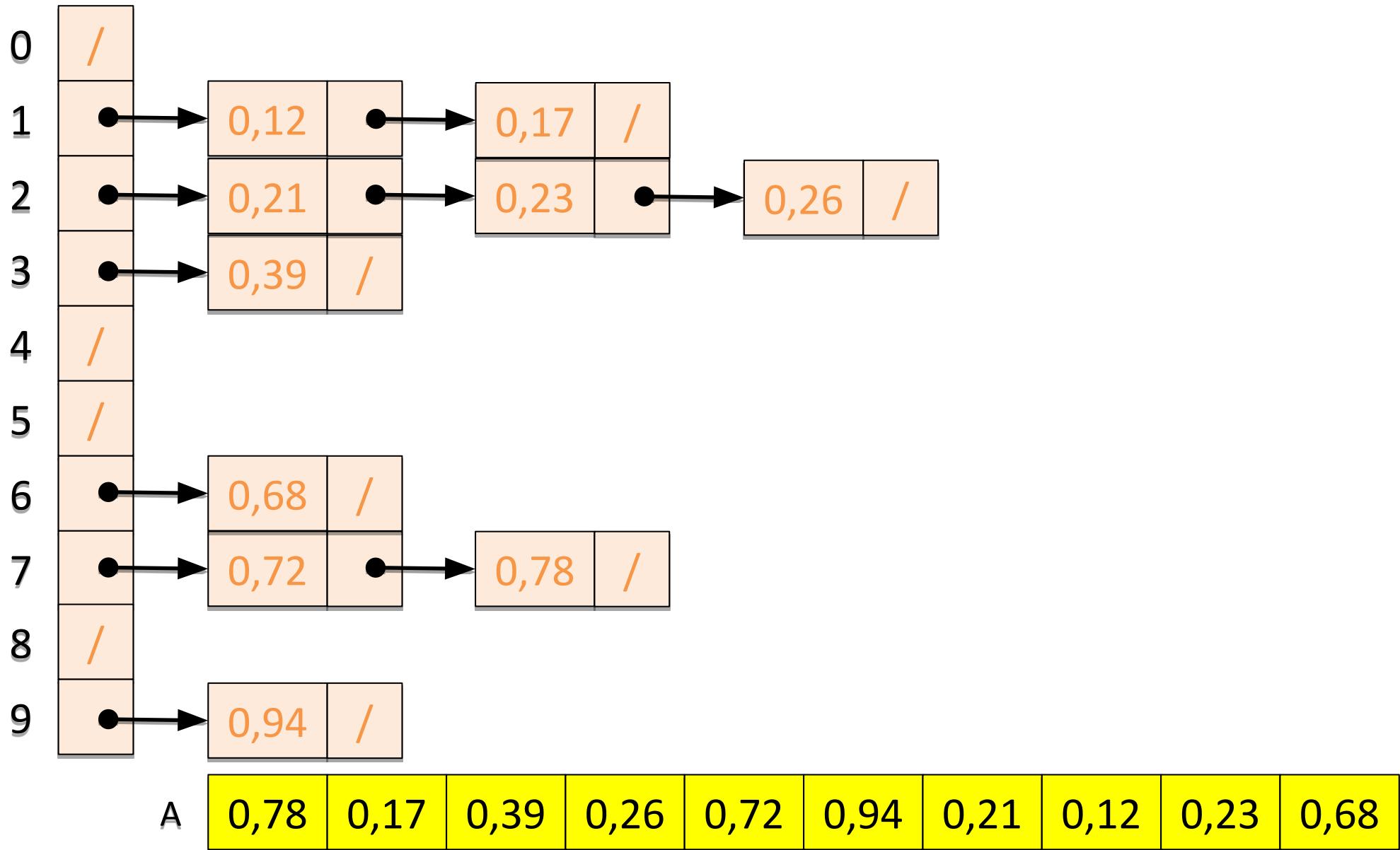
# Método BucketSort



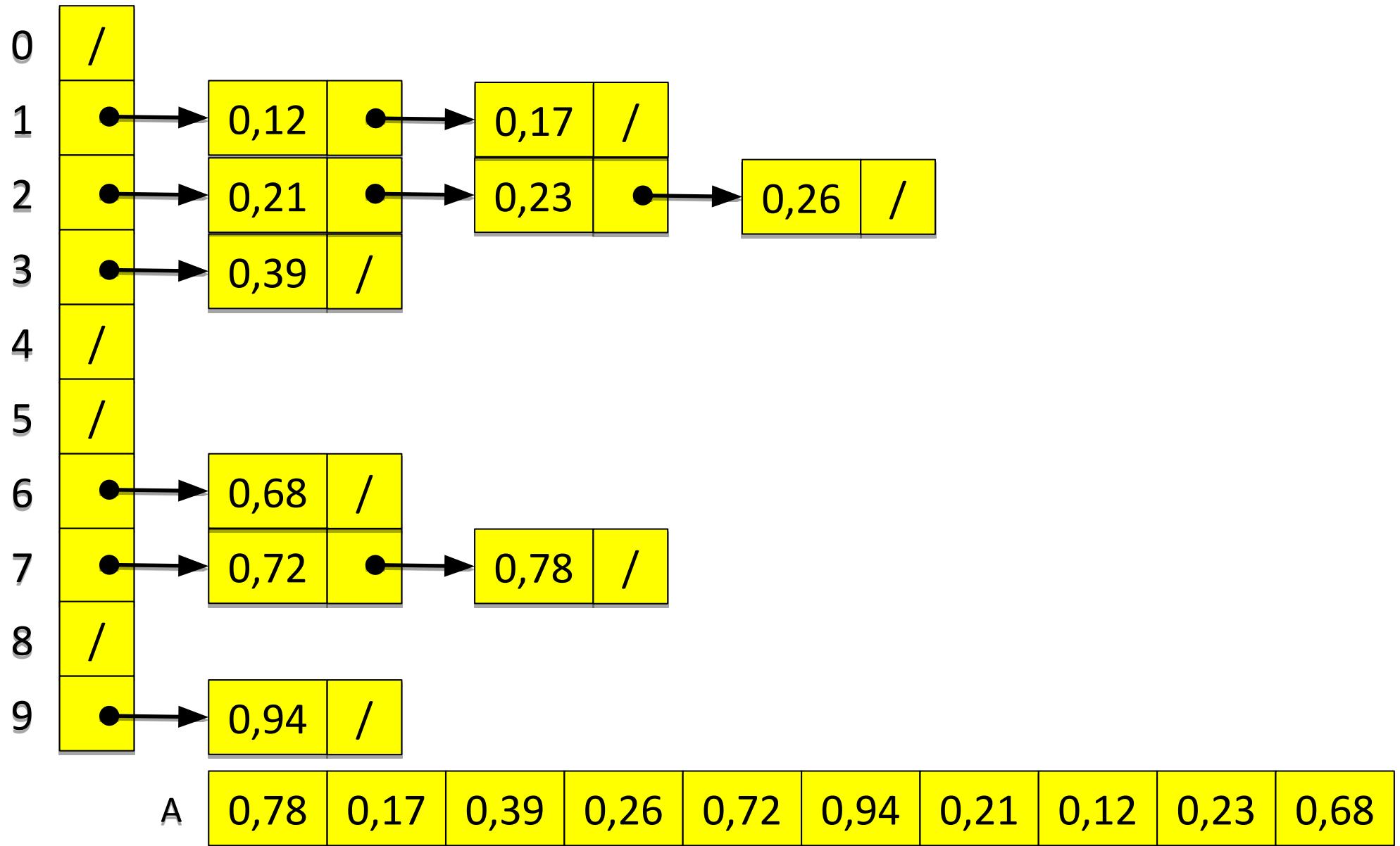
# Método BucketSort



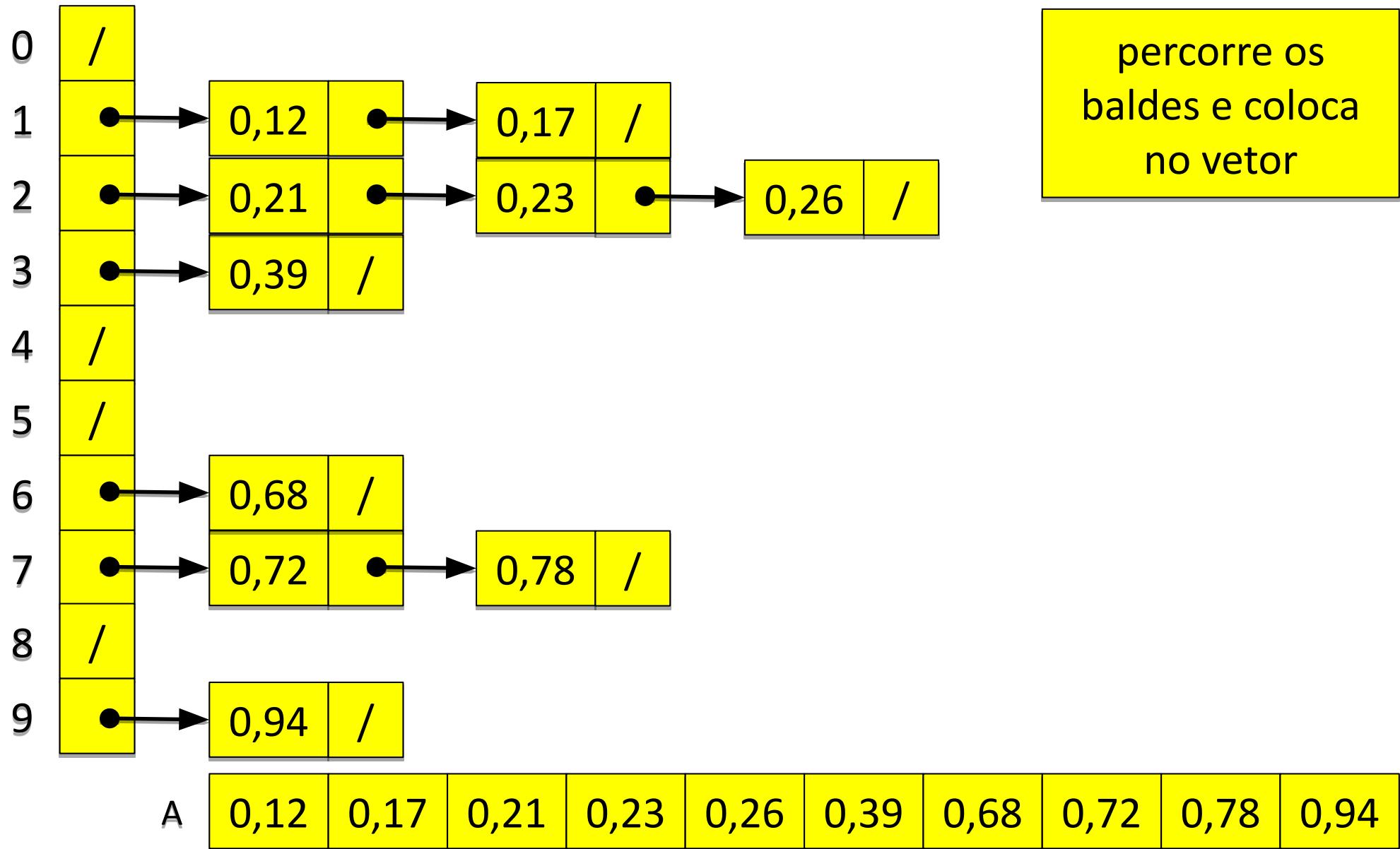
# Método BucketSort



# Método BucketSort



# Método BucketSort



# Método BucketSort

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

# Análise do Bucket-Sort

- A análise depende de conhecimentos estatísticos
  - É possível ser  $O(N)$  quando:
  - Os elementos estão distribuídos de forma uniforme sobre o intervalo  $[0,1)$ , onde cada balde contenha somente 1 elemento
  - Logo, o número esperado de operações para ordenar cada balde é  $O(1)$
  - Portanto, o número de operações esperado do algoritmo de ordenação por baldes é  $O(n)$

# Ordenação em Tempo Linear

- Foram vistos três algoritmos de ordenação linear:
  - Ordenação por Contagem
  - Ordenação Digital
  - Ordenação em Baldes
- Esses algoritmos são melhores do que os algoritmos de ordenação por comparação, onde se tem  $O(n \log_2 n)$  operações
- **Entretanto**, nem sempre é interessante utilizar um desses três algoritmos:
  - Todos eles pressupõem algo sobre os dados de entrada a serem ordenados