



# **Algoritmos e Estruturas de Dados**

Ordenação (parte II) – Capítulo 12  
2019/20

# Ordenação por indexação

Existem algoritmos de ordenação que possam correr mais rápido que  $O(n \log n)$  ?

Sim!

No entanto, para que isto possa acontecer, a sequência a ordenar tem de seguir um conjunto de requisitos.

PRÉ-REQUISITOS



# Bucket Sort (1)

## Requisitos:

- Uma sequência  $S$  de  $n$  elementos, cujas chaves são valores inteiros entre 0 e  $C-1$ , sendo  $C \geq 2$ .

## Objetivo:

- $S$  deverá ser ordenada de acordo com as chaves dos elementos.



100

## Características:

- As chaves variam entre 0 e 10.
- Os registos são as chaves.
- Não existem chaves repetidas.

Sequência de entrada: 7 4 1 9 6

[illegible]

100


7 4 1 9 6

[illegible]

# Bucket Sort (3)

## exemplo 1

Sequência de entrada: 7 4 1 9 6



0	1	2	3	4	5	6	7	8	9	10
<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>F</i>

# Bucket Sort (3)

## exemplo 1

Sequência de entrada: 7 4 1 9 6



0	1	2	3	4	5	6	7	8	9	10
<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>F</i>

# Bucket Sort (3)

## exemplo 1

Sequência de entrada:

7 4 1 9 6



0	1	2	3	4	5	6	7	8	9	10
<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>F</i>



# Bucket Sort (3)

## exemplo 1

Sequência de entrada: 7 4 1 9 6



0	1	2	3	4	5	6	7	8	9	10
<i>F</i>	<i>F</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>F</i>

# Bucket Sort (3)

## exemplo 1

Sequência de entrada: 7 4 1 9 6



0	1	2	3	4	5	6	7	8	9	10
<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>F</i>

# Bucket Sort (3)

## exemplo 1

Sequência de entrada: 7 4 1 9 6



0	1	2	3	4	5	6	7	8	9	10
<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>F</i>

# Bucket Sort (3)

## exemplo 1

Sequência de entrada: 7 4 1 9 6



0	1	2	3	4	5	6	7	8	9	10
<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>	<i>F</i>

# Bucket Sort (3)

## exemplo 1

Sequência de entrada: 7 4 1 9 6



0	1	2	3	4	5	6	7	8	9	10
<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>	<i>F</i>

# Bucket Sort (3)

## exemplo 1

Sequência de entrada: 7 4 1 9 6



0	1	2	3	4	5	6	7	8	9	10
<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>	<i>T</i>	<i>F</i>	<i>T</i>	<i>F</i>

# Bucket Sort (4)

## exemplo 1

Sequência de entrada: 7 4 1 9 6

0	1	2	3	4	5	6	7	8	9	10
<i>F</i>	<i>T</i>	<i>F</i>	<i>F</i>	<i>T</i>	<i>F</i>	<i>T</i>	<i>T</i>	<i>F</i>	<i>T</i>	<i>F</i>

Sequência Ordenada: 1 4 6 7 9

Complexidade Temporal:  $O(n+C)$

$n$  – número de elementos a ordenar

$C$  – número de chaves possíveis distintas (no exemplo 11)

100

## Características:

- As chaves variam entre 0 e 10.
- Os registos são as chaves.
- Podem existir chaves repetidas.

Sequência de entrada: 7 4 1 9 7 1 6

[illegible]




100

[illegible]

# Bucket Sort (6)

## exemplo 2

Sequência de entrada: 7 4 1 9 7 1 6



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

# Bucket Sort (6)

## exemplo 2

Sequência de entrada: 7 4 1 9 7 1 6



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

# Bucket Sort (6)

## exemplo 2

Sequência de entrada: 7 4 1 9 7 1 6



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

# Bucket Sort (6)

## exemplo 2

Sequência de entrada: 7 4 1 9 7 1 6



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

# Bucket Sort (6)

## exemplo 2

Sequência de entrada: 7 4 1 9 7 1 6



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

# Bucket Sort (6)

## exemplo 2

Sequência de entrada: 7 4 1 9 7 1 6



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

# Bucket Sort (6)

## exemplo 2

Sequência de entrada: 7 4 1 9 7 1 6



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



# Bucket Sort (6)

## exemplo 2

Sequência de entrada: 7 4 1 9 7 1 6



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

# Bucket Sort (6)

## exemplo 2

Sequência de entrada: 7 4 1 9 7 1 6



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

# Bucket Sort (6)

## exemplo 2

Sequência de entrada: 7 4 1 9 7 1 6



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

# Bucket Sort (6)

## exemplo 2

Sequência de entrada: 7 4 1 9 7 1 6



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

# Bucket Sort (6)

## exemplo 2

Sequência de entrada: 7 4 1 9 7 1 6



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

# Bucket Sort (6)

## exemplo 2

Sequência de entrada: 7 4 1 9 7 1 6



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

# Bucket Sort (7)

## exemplo 2

Sequência de entrada: 7 4 1 9 7 1 6

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

Sequência Ordenada: 1 1 4 6 7 7 9

Complexidade Temporal:  $O(n+C)$

$n$  – número de elementos a ordenar

$C$  – número de chaves possíveis distintas (no exemplo 11)

# Bucket Sort (8)

## exemplo 3

### Características:

- As chaves variam entre 0 e 10.
- Os registos não possuem apenas chave.
- Podem existir chaves repetidas.

Sequência de entrada: (7, K) (1, A) (9, M) (7,B) (1,X) (4, J)

0	1	2	3	4	5	6	7	8	9	10

Cada elemento do vetor é uma Fila (implementada em lista ligada simples)



# Bucket Sort (9)

## exemplo 3

## Sequência de entrada:

(7, K) (1, A) (9, M) (7,B) (1,X) (4, J)



0 1 2 3 4 5 6 7 8 9 10

[illegible]

# Bucket Sort (9)

## exemplo 3

Sequência de entrada:

(7, K) (1, A) (9, M) (7, B) (1, X) (4, J)



0	1	2	3	4	5	6	7	8	9	10

(7, K)

# Bucket Sort (9)

## exemplo 3

Sequência de entrada: (7, K) (1, A) (9, M) (7, B) (1, X) (4, J)



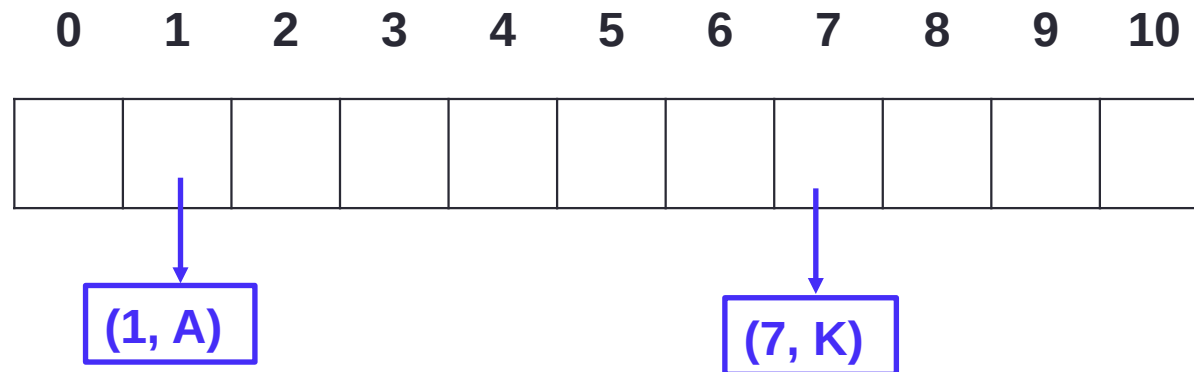
0	1	2	3	4	5	6	7	8	9	10

(7, K)

# Bucket Sort (9)

## exemplo 3

Sequência de entrada: (7, K) (1, A) (9, M) (7, B) (1, X) (4, J)

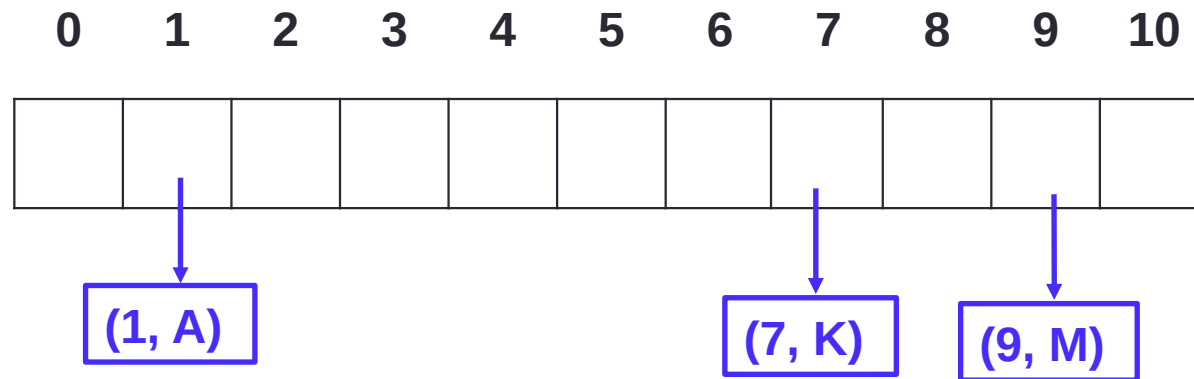




# Bucket Sort (9)

## exemplo 3

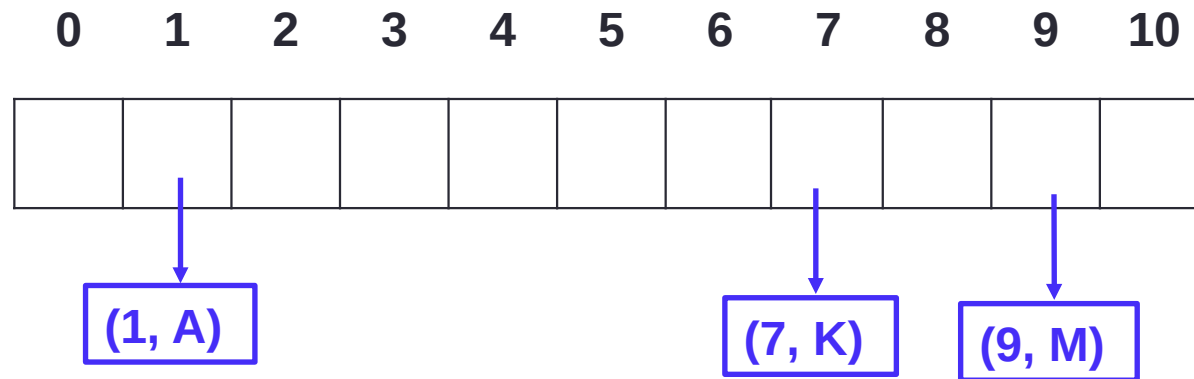
Sequência de entrada: (7, K) (1, A) (9, M) (7, B) (1, X) (4, J)



# Bucket Sort (9)

## exemplo 3

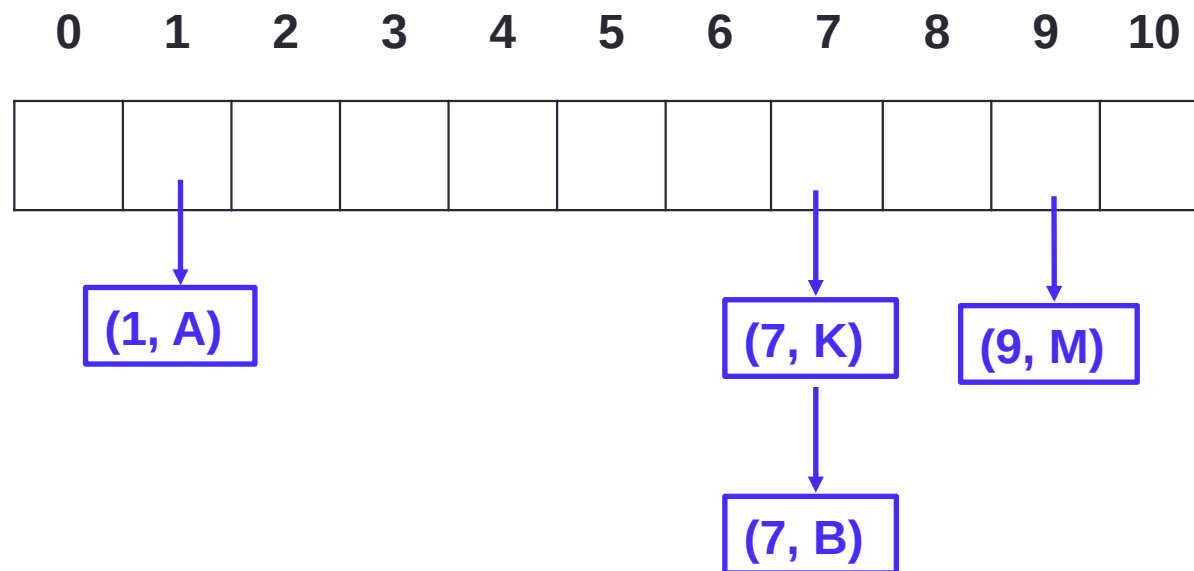
Sequência de entrada: (7, K) (1, A) (9, M) (7,B) (1,X) (4, J)



# Bucket Sort (9)

## exemplo 3

Sequência de entrada: (7, K) (1, A) (9, M) (7, B) (1, X) (4, J)

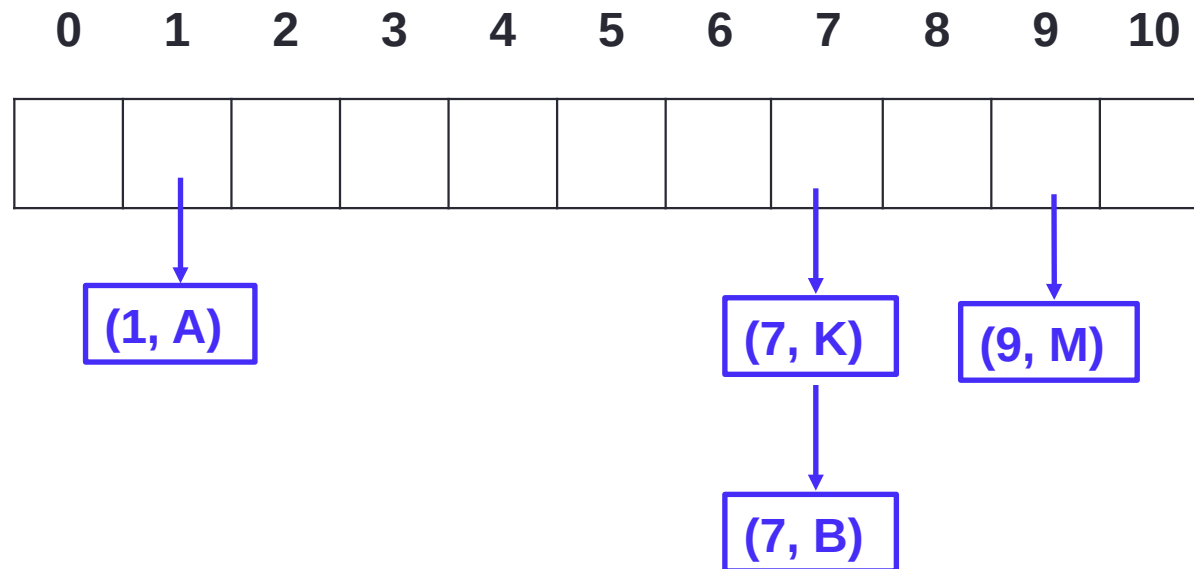




# Bucket Sort (9)

## exemplo 3

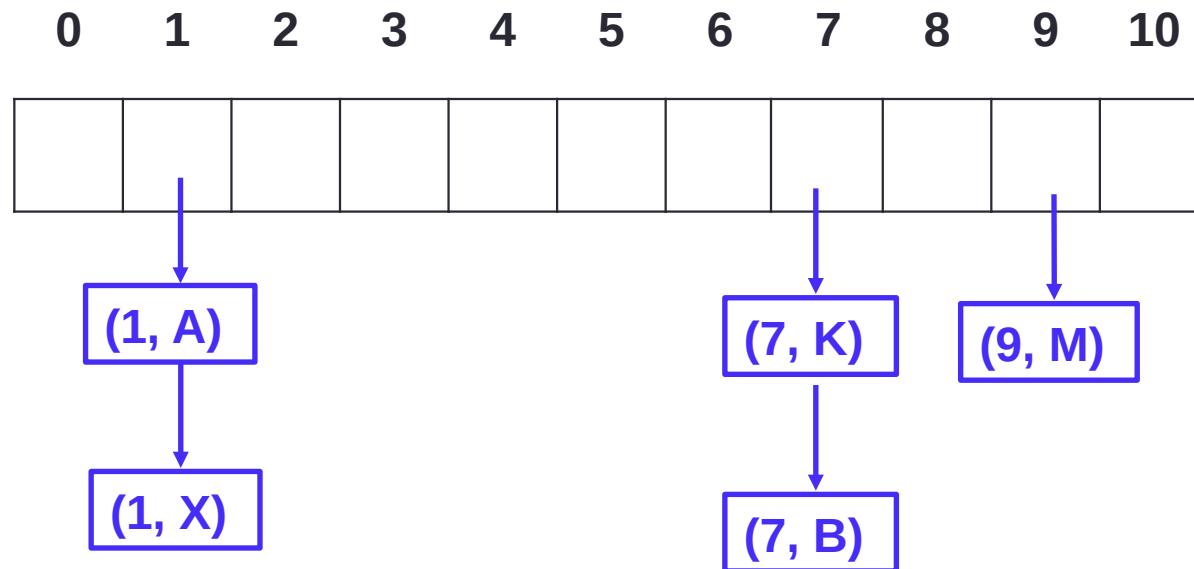
Sequência de entrada: (7, K) (1, A) (9, M) (7, B) (1, X) (4, J)



# Bucket Sort (9)

## exemplo 3

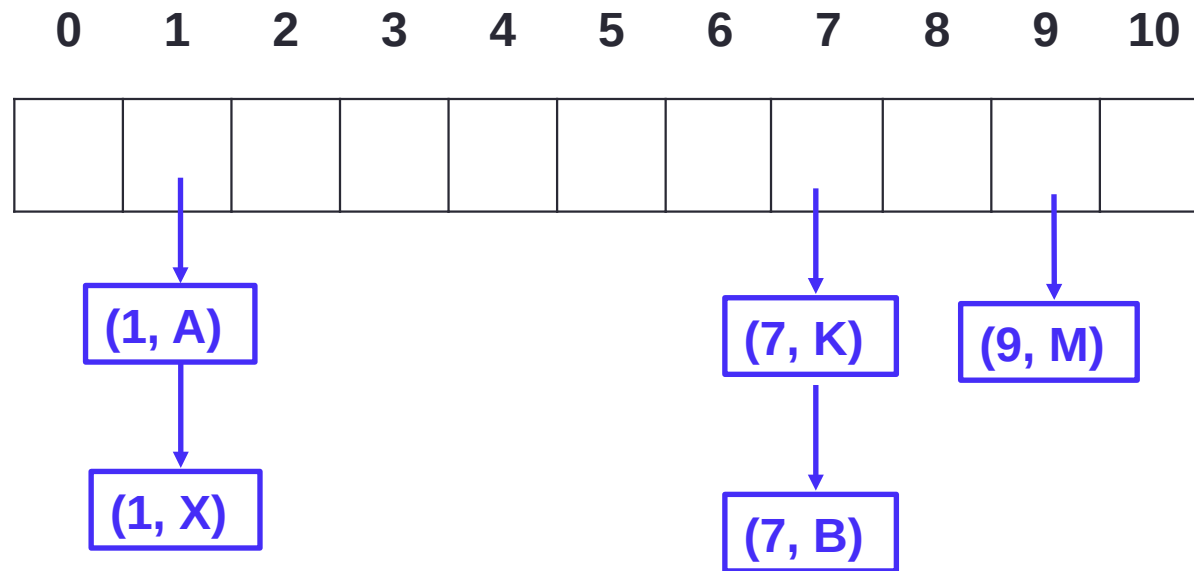
Sequência de entrada: (7, K) (1, A) (9, M) (7, B) (1, X) (4, J)



# Bucket Sort (9)

## exemplo 3

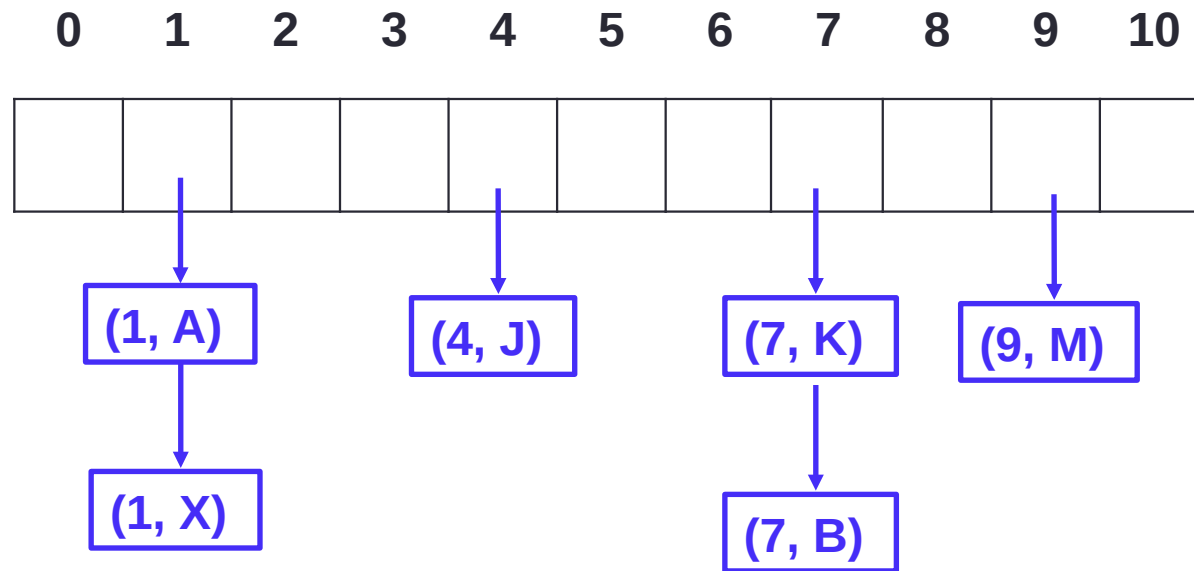
Sequência de entrada: (7, K) (1, A) (9, M) (7, B) (1, X) (4, J)



# Bucket Sort (9)

## exemplo 3

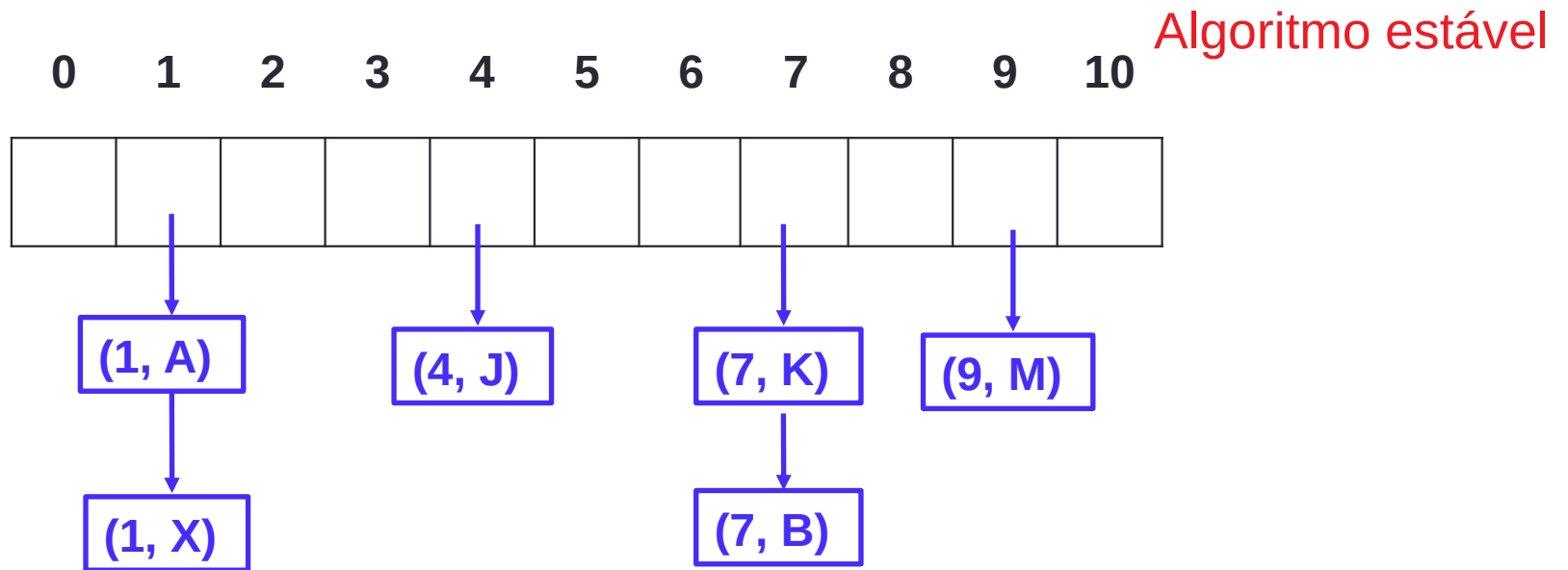
Sequência de entrada: (7, K) (1, A) (9, M) (7, B) (1, X) (4, J)



# Bucket Sort (10)

## exemplo 3

Sequência de entrada: (7, K) (1, A) (9, M) (7,B) (1,X) (4, J)



Sequência Ordenada: (1, A) (1, X) (4, J) (7,K) (7,B) (9, M)

Complexidade Temporal:  $O(n+C)$

$n$  – número de elementos a ordenar

$C$  – número de chaves possíveis distintas (no exemplo 11)



# Radix Sort (1)

## Requisitos:

- Uma sequência  $S$  de  $n$  elementos, cujas chaves são compostas por  $d$  dígitos.
- Os dígitos variam num intervalo  $I$  pequeno (e.g., de 0 a 9).

## Objetivo:

- $S$  deverá ser ordenada de acordo com as chaves dos elementos.

# Radix Sort (2)

## exemplo

Sequência Inicial: 587 945 540 285 983 517

0	1	2	3	4	5	6	7	8	9

O tamanho do vetor é o número de dígitos das possíveis chaves.

Cada elemento do vetor é uma Fila (implementada em lista ligada simples)

# Radix Sort (3)

## exemplo

Sequência Inicial:

587

945

540

285

983

517



0

1

2

3

4

5

6

7

8

9

Passo1:

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



587



# Radix Sort (3)

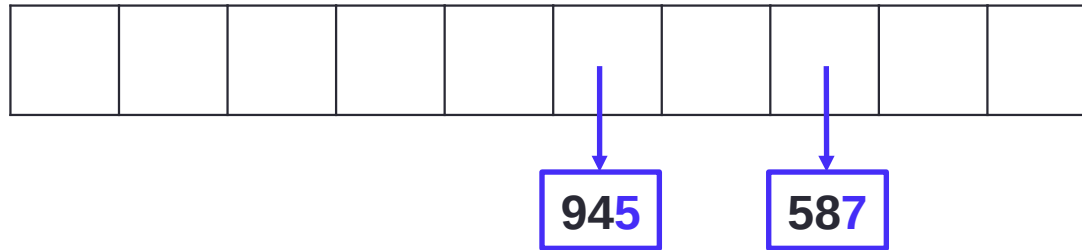
## exemplo

Sequência Inicial: 587 945 540 285 983 517



0 1 2 3 4 5 6 7 8 9

Passo1:



# Radix Sort (3)

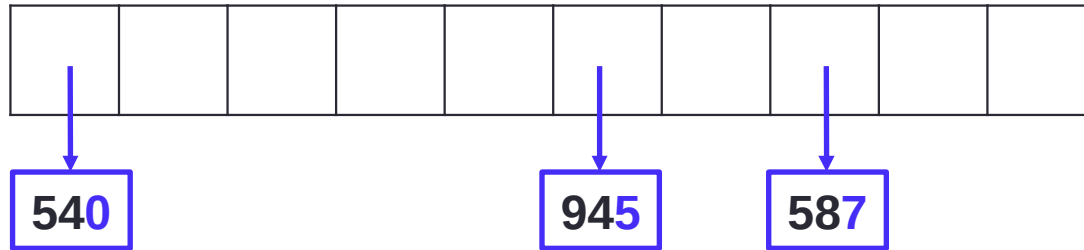
## exemplo

Sequência Inicial: 587 945 540 285 983 517



0 1 2 3 4 5 6 7 8 9

Passo1:



# Radix Sort (3)

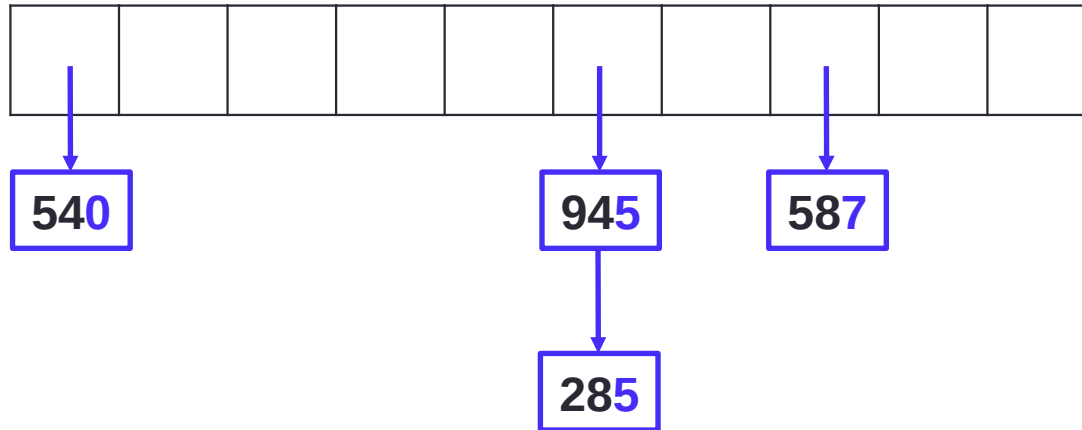
## exemplo

Sequência Inicial: 587 945 540 285 983 517



0 1 2 3 4 5 6 7 8 9

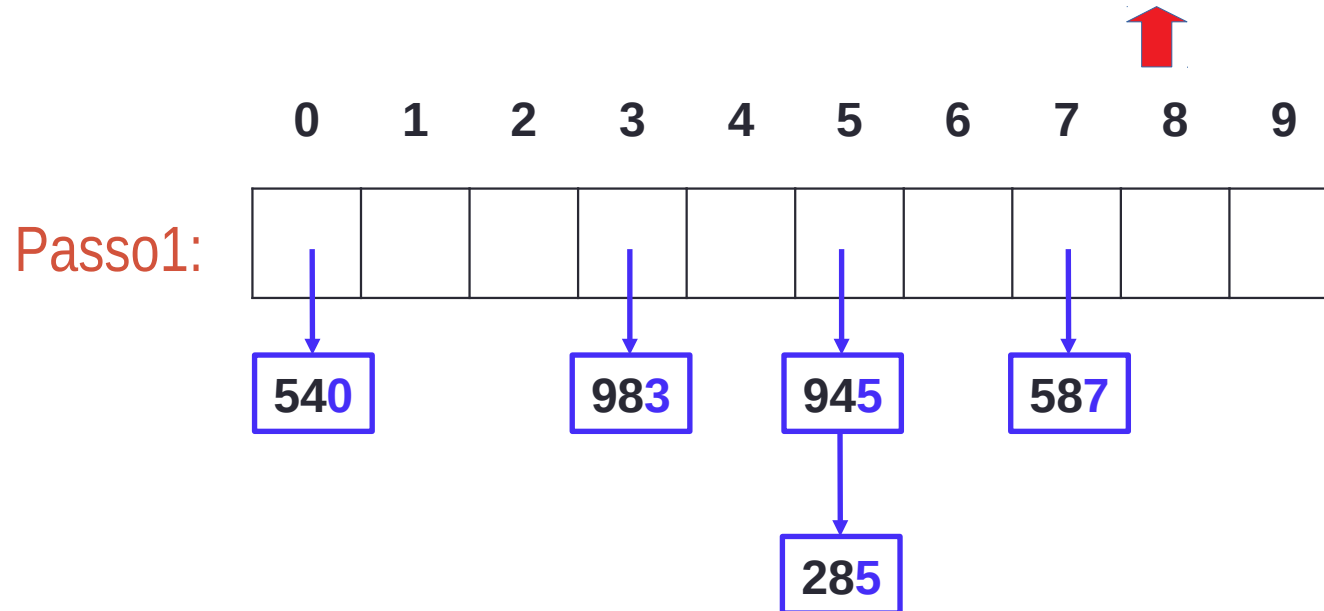
Passo1:



# Radix Sort (3)

## exemplo

Sequência Inicial: 587 945 540 285 983 517



# Radix Sort (3)

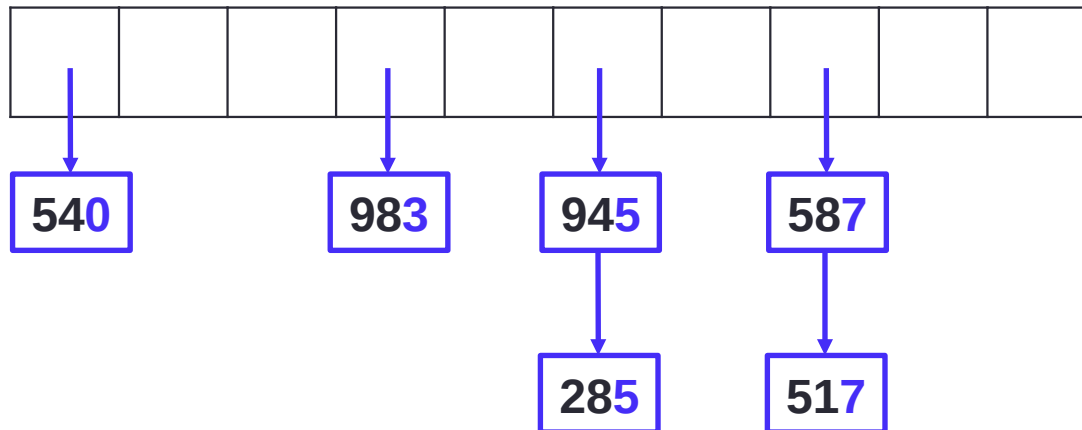
## exemplo

Sequência Inicial: 587 945 540 285 983 517



0 1 2 3 4 5 6 7 8 9

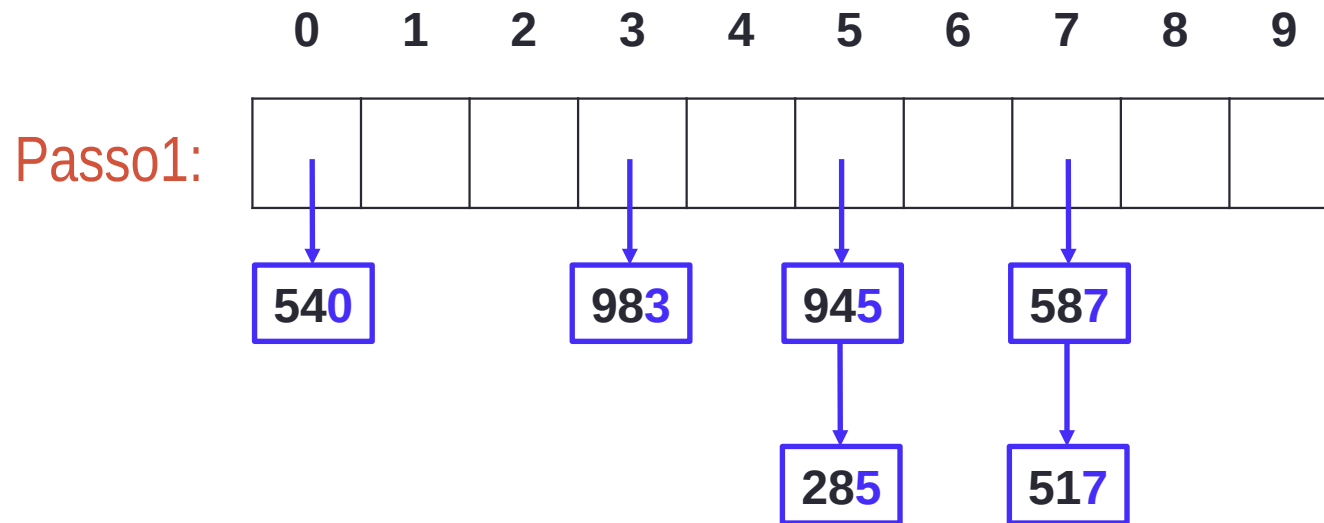
Passo1:



# Radix Sort (4)

## exemplo

Sequência Inicial: 587 945 540 285 983 517



Resultado do passo 1: 540 983 945 285 587 517

(sequência ordenada em relação ao último dígito)

100

## Resultado do passo 1:

983

945

285

587

517

9

[illegible]

# Radix Sort (5)

## exemplo

Resultado do passo 1:

540 983 945 285 587 517



0 1 2 3 4 5 6 7 8 9

Passo 2:

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

540



# Radix Sort (5)

## exemplo

Resultado do passo 1:

540 983 945 285 587 517



0 1 2 3 4 5 6 7 8 9

Passo 2:

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

540

983

# Radix Sort (5)

## exemplo

Resultado do passo 1:

540 983 945 285 587 517



0 1 2 3 4 5 6 7 8 9

Passo 2:

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

540

983

945

# Radix Sort (5) exemplo

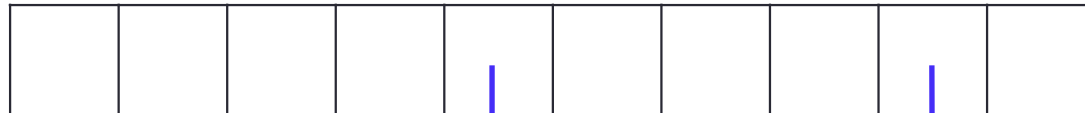
Resultado do passo 1:

540 983 945 285 587 517



0 1 2 3 4 5 6 7 8 9

Passo 2:



# Radix Sort (5)

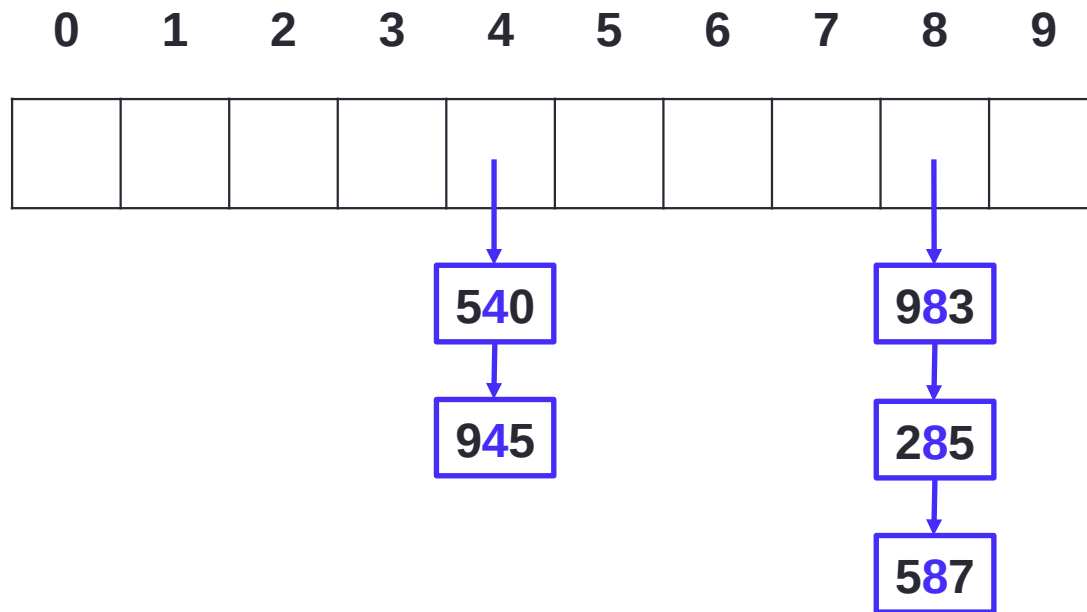
## exemplo

Resultado do passo 1:

540 983 945 285 587 517



Passo 2:

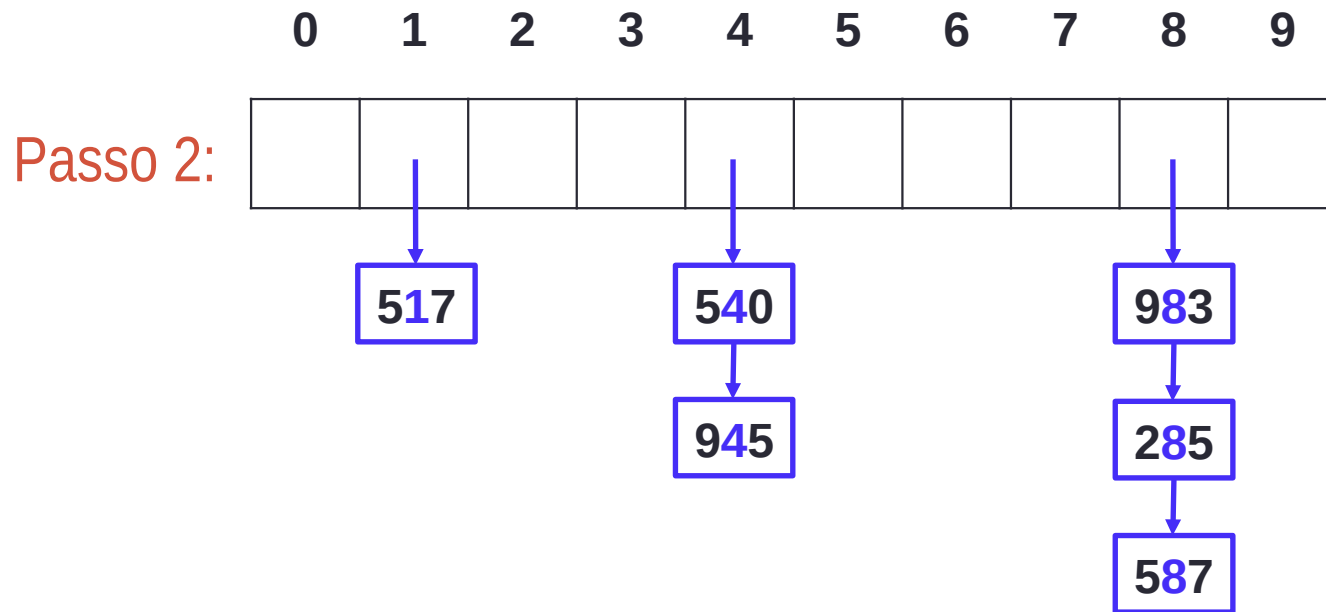


# Radix Sort (5)

## exemplo

Resultado do passo 1:      540    983    945    285    587    517

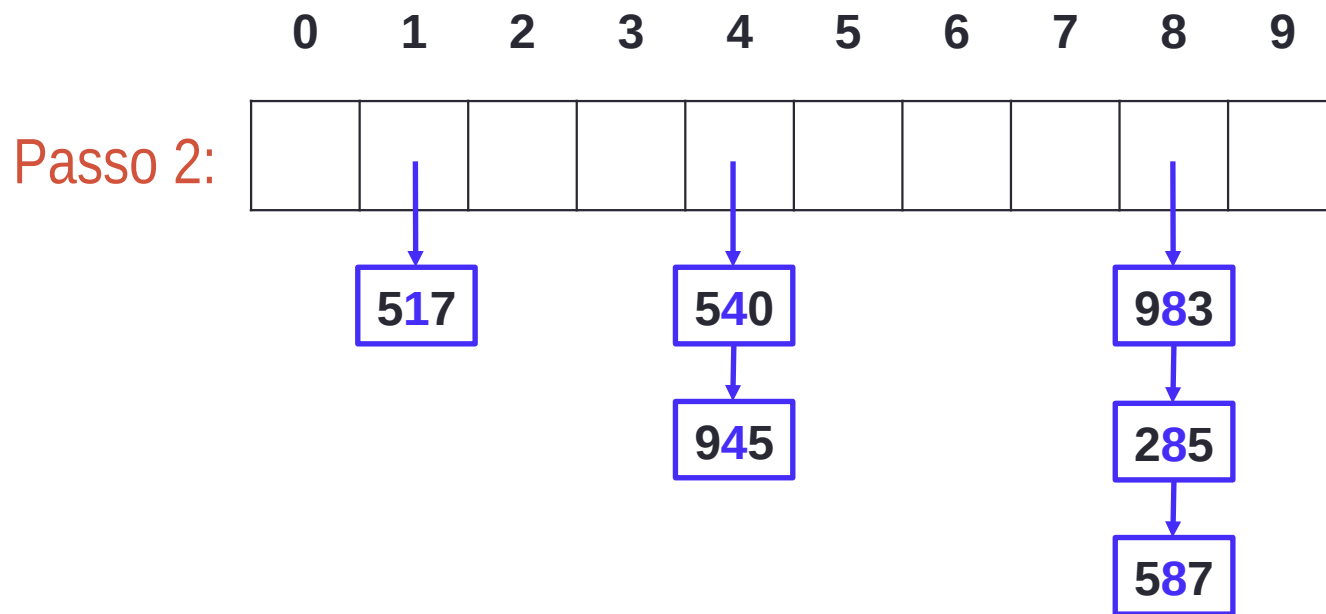




# Radix Sort (6)

## exemplo

Resultado do passo 1:      540      983      945      285      587      517



Resultado do passo 2:   517      540      945      983      285      587

(sequência ordenada em relação aos dois últimos dígitos)

100

Resultado do passo 2: 517 540 945 983 285 587

0 1 2 3 4 5 6 7 8 9

## Passo 3:

[illegible]

100

## Resultado do passo 2:

540

945

983

285

587

1

2

# 3

4

5

6

7

8

9

## Passo 3:

517



# Radix Sort (7)

## exemplo

Resultado do passo 2: 517 540 945 983 285 587

0 1 2 3 4 5 6 7 8 9

Passo 3:

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

517

540

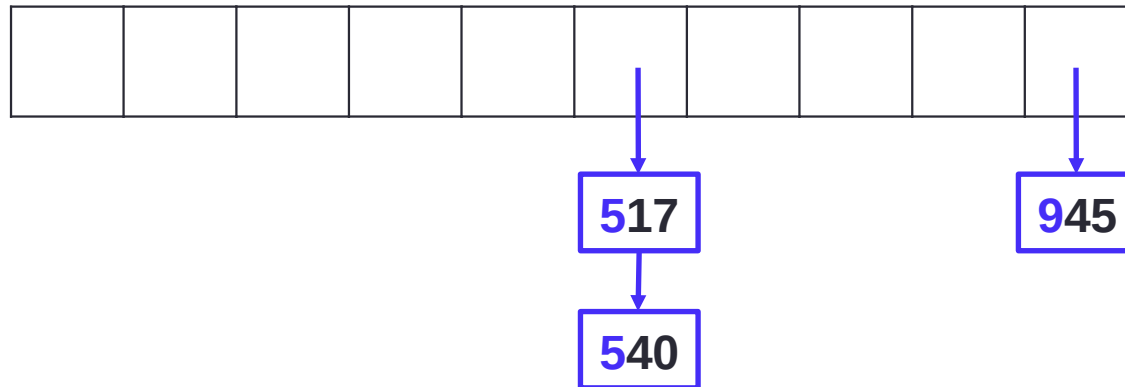
# Radix Sort (7)

## exemplo

Resultado do passo 2: 517 540 945 983 285 587

0 1 2 3 4 5 6 7 8 9

Passo 3:



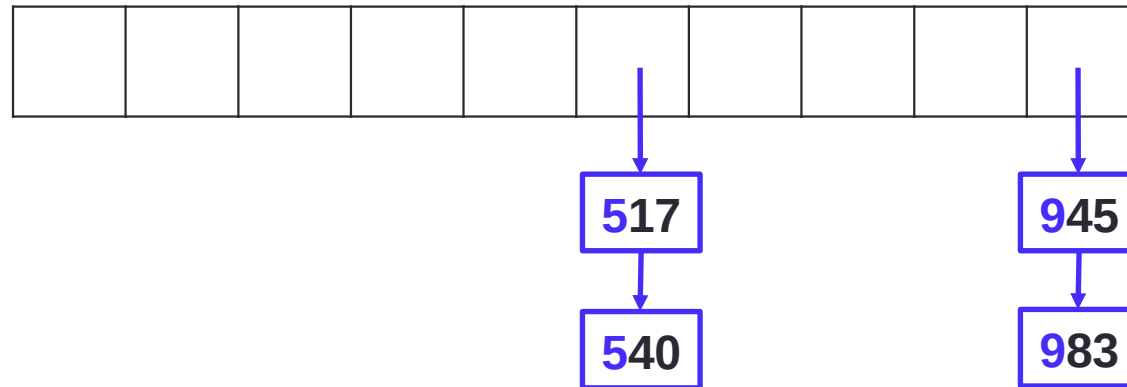
# Radix Sort (7)

## exemplo

Resultado do passo 2: 517 540 945 983 285 587

0 1 2 3 4 5 6 7 8 9

Passo 3:



# Radix Sort (7)

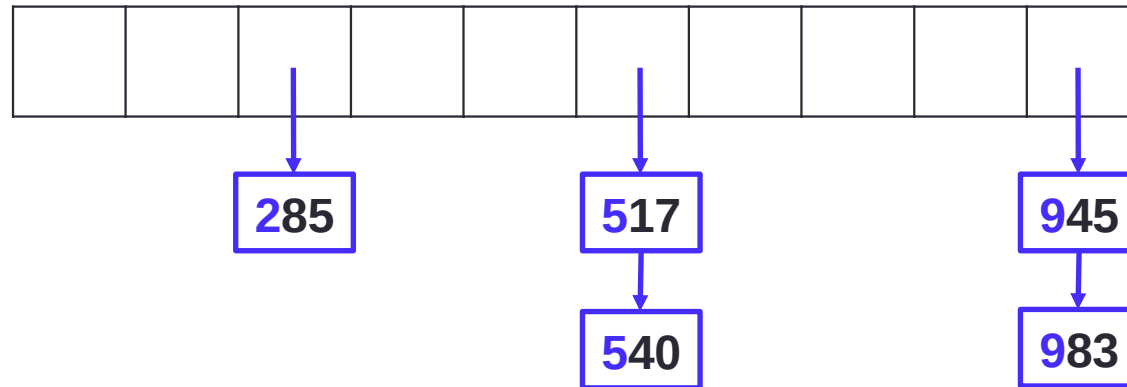
## exemplo

Resultado do passo 2: 5**1**7 5**4**0 9**4**5 9**8**3 2**8**5 5**8**7



0 1 2 3 4 5 6 7 8 9

Passo 3:



# Radix Sort (7)

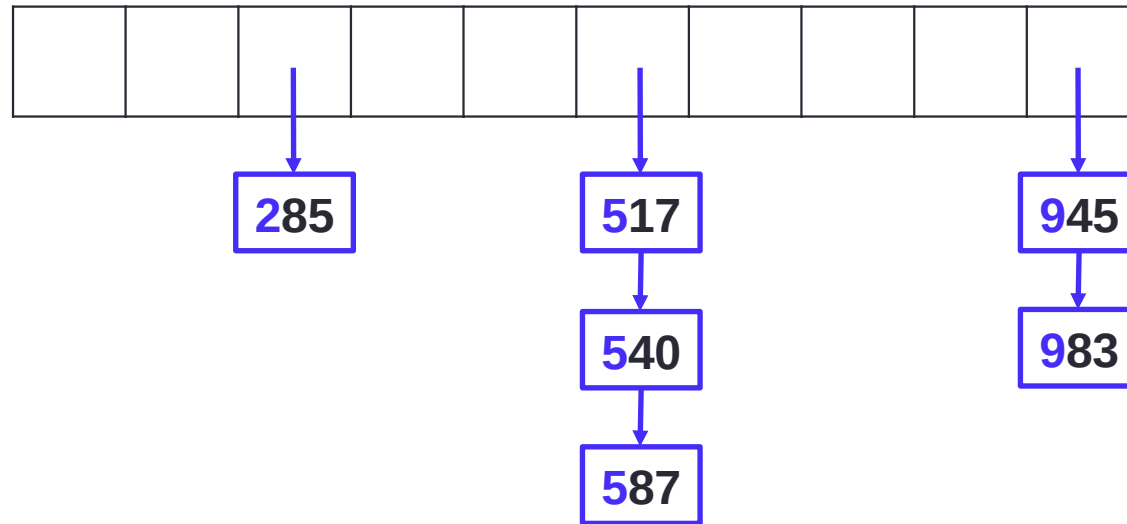
## exemplo

Resultado do passo 2: 517 540 945 983 285 587



0 1 2 3 4 5 6 7 8 9

Passo 3:

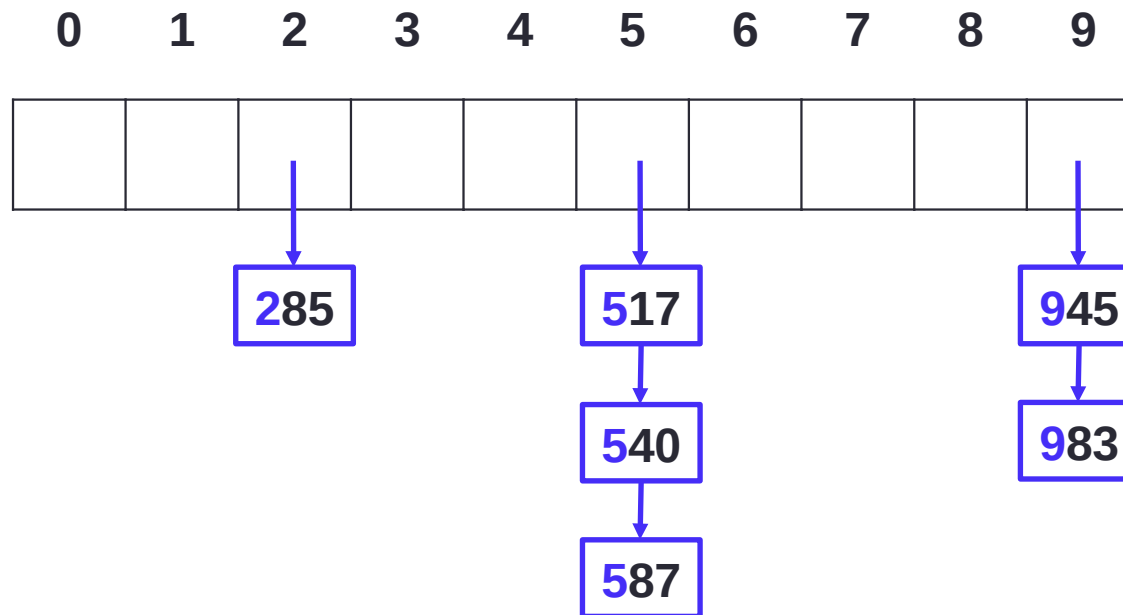


# Radix Sort (8)

## exemplo

Resultado do passo 2: 517 540 945 983 285 587

Passo 3:



Resultado do passo 3: 285 517 540 587 945 983

(sequência ordenada em relação aos três últimos dígitos)

# Radix Sort (9)

- Efectuam-se  $d$  passos, do dígito menos significativo para o mais significativo.

Algoritmo estável

- No passo  $k$ :
  - distribuem-se os  $n$  elementos de acordo com o  $k$ -ésimo dígito menos significativo, usando esse dígito para indexar a fila concatenável (disciplina FIFO) onde se insere o elemento;
  - concatenam-se as filas, percorrendo o vector da posição referente ao dígito de menor valor para a posição referente ao dígito de maior valor.

Complexidade Temporal:  $O(d(n+I))$

$n$  – número de elementos a ordenar

$d$  – número de dígitos das chaves dos elementos a ordenar (no exemplo 3)

$I$  – número de dígitos das chaves (no exemplo 10)