INSTITUTO FEDERAL
Ceará
Campus Tianguá

ORDENAÇÃO EM TEMPO LINEAR

Ciência da Computação

Disciplina: Construção e Análise de Algoritmos

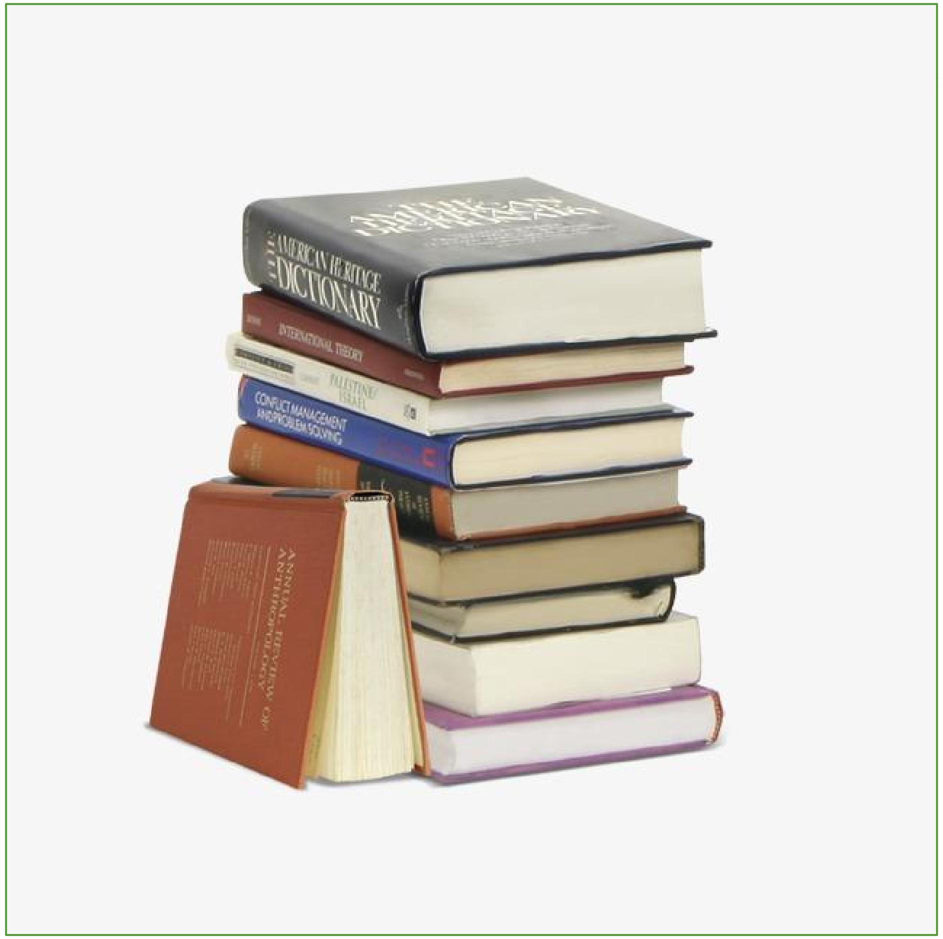
Professor: Adonias Caetano

Objetivos

- Compreender o método de ordenação do Counting Sort, Radix sort e Bucket sort.
- Ser capaz de implementar esses algoritmos em C.

Conteúdo da aula

- Counting sort
- Radix sort
- Bucket sort



COUNTING SORT

Definição

ifce.edu.br

- Outra classe de algoritmos de ordenação é aquela na qual a definição da ordem se dá por **contagem**.
- A ordenação por contagem pressupõe que cada um dos n elementos de entrada é um inteiro no intervalo de 0 a k ou 1 a k .
- Quando $\boldsymbol{k = O(n)}$, a ordenação é executada no tempo $\boldsymbol{\Theta(n)}$.

Ideia Básica

- Considere por exemplo uma coleção de elementos a ordenar na qual as chaves podem assumir ***N*** valores diferentes.

A =

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

Um vetor A com números que podem assumir valores de 0 a 5
Portanto, há N = 6 valores distintos

Ideia Básica

ifce.edu.br

- Cria-se então uma tabela com N contadores e varre-se a coleção do início ao fim, incrementando-se o contador correspondente à chave de valor i cada vez que esse valor for encontrado.

A =

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

Tabela =

0	1	2	3	4	5
2	0	2	3	0	1

Ideia Básica

- Ao final dessa varredura conhece-se exatamente quantas posições serão necessárias para cada valor de chave; os elementos são transferidos para as posições corretas na nova coleção, ordenada.

A =

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

Tabela =

0	1	2	3	4	5
2	0	2	3	0	1

B =

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

Ideia Básica

- Claramente, a aplicação desse princípio básico de contagem a domínios com muitos valores tornar-se-ia inviável.

- Por exemplo, se os elementos são inteiros de 32 bits, o algoritmo de contagem básico precisaria de uma tabela com cerca de quatro bilhões (2^{32}) de contadores.

Vantagens e Desvantagens

ifce.edu.br

- **Vantagens**
  - Ordena vetores em tempo linear para o tamanho do vetor inicial;
  - Não realiza comparações;
  - É um algoritmo de ordenação estável;

- **Desvantagens**
  - Necessita de dois vetores adicionais para sua execução, utilizando, assim, mais espaço na memória.

Funcionamento

ifce.edu.br

- A ordenação por contagem pressupõe que cada um dos $\boldsymbol{n}$ elementos do vetor de entrada é um inteiro entre 1 e $\boldsymbol{k}$ ($\boldsymbol{k}$ representa o maior inteiro presente no vetor ou em intervalo possível de valores).
- A ideia básica é determinar, para cada elemento de entrada $\boldsymbol{x}$, o número de elementos menores ou iguais a $\boldsymbol{x}$.
- Com essa informação é possível determinar exatamente onde o elemento $\boldsymbol{x}$ será inserido.

Algoritmo

```
CountingSort(A, B, k)
    for I←0 to k
      C[i]← 0
    for j←1 to comprimento [A]
      C[A[j]]← C[A[j]] + 1
    for i←1 to k
      C[i] = C[i] + C[i-1]
    for j← comprimento [A] downto 1
      B[C[A[j]]] = A[j]
      C[A[j]] = C[A[j]] - 1
```

Simulação

O algoritmo recebe um vetor desordenado como entrada:

A =

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

Em seguida, gera os vetores adicionais B e C:

B =

1	2	3	4	5	6	7	8

→ O vetor ***B*** é do mesmo tamanho do vetor ***A*** (8 elementos).

C =

0	1	2	3	4	5
0	0	0	0	0	0

→ O vetor ***C*** é do tamanho do maior elemento de ***A*** + 1 (5 + 1 = ***6***).

```
CountingSort(A, B, k)
   for I←0 to k
          C[i]← 0
```

Simulação

Se o valor de um elemento de entrada é i, incrementamos $C[i]$:

A =

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

```
04 for j←1 to comprimento [A]
05     C[A[j]]← C[A[j]] + 1
```

C =

0	1	2	3	4	5
0	0	0	0	0	0

$j = 1$

$$C[A[1]] \rightarrow C[2]++$$

AÇÃO: INCREMENTAR A POSIÇÃO 2 DO VETOR C

Simulação

Se o valor de um elemento de entrada é $\boldsymbol{i}$, incrementamos $\boldsymbol{C}[\boldsymbol{i}]$:

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

```
04 for j←1 to comprimento [A]
05     C[A[j]]← C[A[j]] + 1
```

	0	1	2	3	4	5
C =	0	0	1	0	0	0

$j = 2$

$$C[A[2]] \rightarrow C[5]++$$

AÇÃO: INCREMENTAR A POSIÇÃO 5 DO VETOR C

Simulação

Se o valor de um elemento de entrada é $\boldsymbol{i}$, incrementamos $\boldsymbol{C}[\boldsymbol{i}]$:

A =

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

```
04 for j←1 to comprimento [A]
05     C[A[j]]← C[A[j]] + 1
```

C =

0	1	2	3	4	5
0	0	1	0	0	1

$j = 3$

$$C[A[3]] \rightarrow C[3] ++$$

AÇÃO: INCREMENTAR A POSIÇÃO 3 DO VETOR C

Simulação

ifce.edu.br

Se o valor de um elemento de entrada é $\boldsymbol{i}$, incrementamos $\boldsymbol{C}[\boldsymbol{i}]$:

A =

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

```
04 for j←1 to comprimento [A]
05     C[A[j]]← C[A[j]] + 1
```

C =

0	1	2	3	4	5
0	0	1	1	0	1

$j = 4$

$$C[A[4]] \rightarrow C[0]++$$

AÇÃO: INCREMENTAR A POSIÇÃO 0 DO VETOR C

Simulação

Se o valor de um elemento de entrada é $\boldsymbol{i}$, incrementamos $\boldsymbol{C}[\boldsymbol{i}]$:

A =

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

```
04 for j←1 to comprimento [A]
05     C[A[j]]← C[A[j]] + 1
```

C =

0	1	2	3	4	5
1	0	1	1	0	1

j = 5

$$C[A[5]] \rightarrow C[2]\,{+}{+}$$

AÇÃO: INCREMENTAR A POSIÇÃO 2 DO VETOR C

Simulação

Se o valor de um elemento de entrada é $\boldsymbol{i}$, incrementamos $\boldsymbol{C}[\boldsymbol{i}]$:

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

```
04 for j←1 to comprimento [A]
05     C[A[j]]← C[A[j]] + 1
```

	0	1	2	3	4	5
C =	1	0	2	1	0	1

j = 6

$$C[A[6]] \rightarrow C[3]++$$

AÇÃO: INCREMENTAR A POSIÇÃO 3 DO VETOR C

Simulação

Se o valor de um elemento de entrada é $\boldsymbol{i}$, incrementamos $\boldsymbol{C}[\boldsymbol{i}]$:

A =

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

```
04 for j←1 to comprimento [A]
05     C[A[j]]← C[A[j]] + 1
```

C =

0	1	2	3	4	5
1	0	2	2	0	1

$j = 7$

$$C[A[7]] \rightarrow C[0]++$$

AÇÃO: INCREMENTAR A POSIÇÃO 0 DO VETOR C

Simulação

Se o valor de um elemento de entrada é $\boldsymbol{i}$, incrementamos $\boldsymbol{C}[\boldsymbol{i}]$:

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

```
04 for j←1 to comprimento [A]
05     C[A[j]]← C[A[j]] + 1
```

	0	1	2	3	4	5
C =	2	0	2	2	0	1

$j = 8$

$$C[A[8]] \to C[3]++$$

AÇÃO: INCREMENTAR A POSIÇÃO 3 DO VETOR C

Simulação

Se o valor de um elemento de entrada é $\boldsymbol{i}$, incrementamos $\boldsymbol{C}[\boldsymbol{i}]$:

A =

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

```
04 for j←1 to comprimento [A]
05     C[A[j]]← C[A[j]] + 1
```

j = 9

C =

0	1	2	3	4	5
2	0	2	3	0	1

$\boldsymbol{C}[\boldsymbol{i}]$ contém um número de elementos de entrada igual a $\boldsymbol{i}$ para cada $\boldsymbol{i} = \mathbf{0}, \mathbf{1}, \mathbf{2}, \ldots, \boldsymbol{k}$.

Simulação

Agora fazemos $\boldsymbol{C[i] = C[i] + C[i-1]}$ para determinarmos quantos elementos de entrada são menores que ou iguais a $\boldsymbol{i}$**:**

	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

SOMA: $2 + 0 = 2$

```
06 for i←1 to k
07    C[i] = C[i] + C[i-1]
```

$i = 1$

$$C[1] = C[1] + C[0]$$
$$C[1] = 0 + 2 = 2$$

Simulação

ifce.edu.br

Agora fazemos $\boldsymbol{C}[\boldsymbol{i}] = \boldsymbol{C}[\boldsymbol{i}] + \boldsymbol{C}[\boldsymbol{i}-\mathbf{1}]$ para determinarmos quantos elementos de entrada são menores que ou iguais a $\boldsymbol{i}$:

A =

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

C =

0	1	2	3	4	5
2	2	2	3	0	1

SOMA: 2 + 2= 4

```
06 for i←1 to k
07    C[i] = C[i] + C[i-1]
```

i = 2

$$C[2] = C[2] + C[1]$$
$$C[2] = 2 + 2 = 4$$

Simulação

Agora fazemos $\boldsymbol{C}[\boldsymbol{i}] = \boldsymbol{C}[\boldsymbol{i}] + \boldsymbol{C}[\boldsymbol{i}-\mathbf{1}]$ para determinarmos quantos elementos de entrada são menores que ou iguais a ***i*:**

A =

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

C =

0	1	2	3	4	5
2	2	4	3	0	1

SOMA: 4 + 3= 7

```
06 for i←1 to k
07    C[i] = C[i] + C[i-1]
```

i = 3

$$C[3] = C[3] + C[2]$$
$$C[3] = 3 + 4 = 7$$

Simulação

Agora fazemos $\boldsymbol{C}[\boldsymbol{i}] = \boldsymbol{C}[\boldsymbol{i}] + \boldsymbol{C}[\boldsymbol{i}-\mathbf{1}]$ para determinarmos quantos elementos de entrada são menores que ou iguais a $\boldsymbol{i}$:

	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	2	4	7	0	1

SOMA: 7 + 0= 7

```
06 for i←1 to k
07    C[i] = C[i] + C[i-1]
```

i = 4

$$C[4] = C[4] + C[3]$$
$$C[4] = 0 + 7 = 7$$

Simulação

Agora fazemos $\boldsymbol{C}[\boldsymbol{i}] = \boldsymbol{C}[\boldsymbol{i}] + \boldsymbol{C}[\boldsymbol{i}-\mathbf{1}]$ para determinarmos quantos elementos de entrada são menores que ou iguais a ***i:***

	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	2	4	7	7	1

SOMA: $7 + 1 = 8$

```
06 for i←1 to k
07    C[i] = C[i] + C[i-1]
```

i = 5

$$C[5] = C[5] + C[4]$$
$$C[5] = 1 + 7 = 8$$

Simulação

Agora fazemos $\boldsymbol{C}[\boldsymbol{i}] = \boldsymbol{C}[\boldsymbol{i}] + \boldsymbol{C}[\boldsymbol{i}-\mathbf{1}]$ para determinarmos quantos elementos de entrada são menores que ou iguais a $\boldsymbol{i}$:

	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	2	4	7	7	8

Vetor C modificado

```
06 for i←1 to k
07    C[i] = C[i] + C[i-1]
```

i = 6

Simulação

- Agora, partindo do maior para o menor índice, fazemos $\boldsymbol{B}[\boldsymbol{C}[\boldsymbol{A}[\boldsymbol{j}]]] = \boldsymbol{A}[\boldsymbol{j}]$.
- Assim, colocamos cada elemento $\boldsymbol{A}[\boldsymbol{j}]$ em sua posição ordenada no vetor $\boldsymbol{B}$:

A =

1	2	3	4	5	6	7	8
2	5	3	0	2	3	0	$\boxed{3}$

C =

0	1	2	3	4	5
2	2	4	$\boxed{7}$	7	8

B =

1	2	3	4	5	6	7	8

```
08 for j← comprimento [A] downto 1
09            B[C[A[j]]] = A[j]
10            C[A[j]] = C[A[j]] - 1
```

j = 8

$$B\left[C\left[A[8]\right]\right] \rightarrow B\left[C[3]\right] \rightarrow B[7] = 3$$

Simulação

- Agora, partindo do maior para o menor índice, fazemos $\boldsymbol{B}[\boldsymbol{C}[\boldsymbol{A}[\boldsymbol{j}]]] = \boldsymbol{A}[\boldsymbol{j}]$.
- Assim, colocamos cada elemento $\boldsymbol{A}[\boldsymbol{j}]$ em sua posição ordenada no vetor $\boldsymbol{B}$:

	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	2	4	7	7	8

	1	2	3	4	5	6	7	8
B =							3	

```
for j← comprimento [A] downto 1
         B[C[A[j]]] = A[j]
         C[A[j]] = C[A[j]] - 1
```

j = 8

$B\left[C\left[A[8]\right]\right] \rightarrow B\left[C[3]\right] \rightarrow B[7] = 3$

$C[3] = 7 - 1 = 6$

Simulação

- Agora, partindo do maior para o menor índice, fazemos $\boldsymbol{B}[\boldsymbol{C}[\boldsymbol{A}[\boldsymbol{j}]]] = \boldsymbol{A}[\boldsymbol{j}]$.
- Assim, colocamos cada elemento $\boldsymbol{A}[\boldsymbol{j}]$ em sua posição ordenada no vetor $\boldsymbol{B}$:

	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	2	4	6	7	8

	1	2	3	4	5	6	7	8
B =							3	

```
for j← comprimento [A] downto 1
         B[C[A[j]]] = A[j]
         C[A[j]] = C[A[j]] - 1
```

j = 7

$$B\left[C\left[A[7]\right]\right] \rightarrow B\left[C[0]\right] \rightarrow B[2] = 0$$

Simulação

- Agora, partindo do maior para o menor índice, fazemos $\boldsymbol{B}[\boldsymbol{C}[\boldsymbol{A}[\boldsymbol{j}]]] = \boldsymbol{A}[\boldsymbol{j}]$.
- Assim, colocamos cada elemento $\boldsymbol{A}[\boldsymbol{j}]$ em sua posição ordenada no vetor $\boldsymbol{B}$:

	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	2	4	6	7	8

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

```
for j← comprimento [A] downto 1
          B[C[A[j]]] = A[j]
          C[A[j]] = C[A[j]] - 1
```

j = 7

$B\left[C[A[7]]\right] \rightarrow B[C[0]] \rightarrow B[2] = 0$

$C[0] = 2 - 1 = 1$

Simulação

- Agora, partindo do maior para o menor índice, fazemos $\boldsymbol{B[C[A[j]]] = A[j]}$.
- Assim, colocamos cada elemento $\boldsymbol{A[j]}$ em sua posição ordenada no vetor $\boldsymbol{B}$:

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

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

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

```
for j← comprimento [A] downto 1
          B[C[A[j]]] = A[j]
          C[A[j]] = C[A[j]] - 1
```

$j = 6$

$$B\left[C\left[A[6]\right]\right] \rightarrow B\left[C[3]\right] \rightarrow B[6] = 3$$

Simulação

- Agora, partindo do maior para o menor índice, fazemos $\boldsymbol{B}[\boldsymbol{C}[\boldsymbol{A}[\boldsymbol{j}]]] = \boldsymbol{A}[\boldsymbol{j}]$.
- Assim, colocamos cada elemento $\boldsymbol{A}[\boldsymbol{j}]$ em sua posição ordenada no vetor $\boldsymbol{B}$:

A =

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

C =

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

B =

1	2	3	4	5	6	7	8
	0				3	3	

```
for j← comprimento [A] downto 1
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] - 1
```

j = 6

$B\left[C\left[A[6]\right]\right] \rightarrow B\left[C[3]\right] \rightarrow B[6] = 3$

$C[3] = 6 - 1 = 5$

Simulação

- Agora, partindo do maior para o menor índice, fazemos $\boldsymbol{B}[\boldsymbol{C}[\boldsymbol{A}[\boldsymbol{j}]]] = \boldsymbol{A}[\boldsymbol{j}]$.
- Assim, colocamos cada elemento $\boldsymbol{A}[\boldsymbol{j}]$ em sua posição ordenada no vetor $\boldsymbol{B}$:

A =

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

C =

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

B =

1	2	3	4	5	6	7	8
	0				3	3	

```
for j← comprimento [A] downto 1
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] - 1
```

$j = 5$

$$B\left[C[A[5]]\right] \rightarrow B[C[2]] \rightarrow B[4] = 2$$

Simulação

- Agora, partindo do maior para o menor índice, fazemos $\boldsymbol{B}[\boldsymbol{C}[\boldsymbol{A}[\boldsymbol{j}]]] = \boldsymbol{A}[\boldsymbol{j}]$.
- Assim, colocamos cada elemento $\boldsymbol{A}[\boldsymbol{j}]$ em sua posição ordenada no vetor $\boldsymbol{B}$:

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

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

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

```
for j← comprimento [A] downto 1
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] - 1
```

j = 5

$$B\left[C[A[5]]\right] \to B[C[2]] \to B[4] = 2$$

$$C[2] = 4 - 1 = 3$$

Simulação

- Agora, partindo do maior para o menor índice, fazemos $\boldsymbol{B}[\boldsymbol{C}[\boldsymbol{A}[\boldsymbol{j}]]] = \boldsymbol{A}[\boldsymbol{j}]$.
- Assim, colocamos cada elemento $\boldsymbol{A}[\boldsymbol{j}]$ em sua posição ordenada no vetor $\boldsymbol{B}$:

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

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

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

```
for j← comprimento [A] downto 1
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] - 1
```

$j = 4$

$$B\left[C[A[4]]\right] \rightarrow B[C[0]] \rightarrow B[1] = 0$$

Simulação

- Agora, partindo do maior para o menor índice, fazemos $\boldsymbol{B}[\boldsymbol{C}[\boldsymbol{A}[\boldsymbol{j}]]] = \boldsymbol{A}[\boldsymbol{j}]$.
- Assim, colocamos cada elemento $\boldsymbol{A}[\boldsymbol{j}]$ em sua posição ordenada no vetor $\boldsymbol{B}$:

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

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

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

```
for j← comprimento [A] downto 1
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] - 1
```

$\mathrm{j} = 4$

$B\left[C[A[4]]\right] \rightarrow B[C[0]] \rightarrow B[1] = 0$

$C[0] = 1 - 1 = 0$

Simulação

- Agora, partindo do maior para o menor índice, fazemos $\boldsymbol{B}[\boldsymbol{C}[\boldsymbol{A}[\boldsymbol{j}]]] = \boldsymbol{A}[\boldsymbol{j}]$.
- Assim, colocamos cada elemento $\boldsymbol{A}[\boldsymbol{j}]$ em sua posição ordenada no vetor $\boldsymbol{B}$:

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

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

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

```
08 for j← comprimento [A] downto 1
09           B[C[A[j]]] = A[j]
10           C[A[j]] = C[A[j]] - 1
```

j = 3

$$B\left[C\left[A[3]\right]\right] \rightarrow B\left[C[3]\right] \rightarrow B[5] = 3$$

Simulação

- Agora, partindo do maior para o menor índice, fazemos $\boldsymbol{B}[\boldsymbol{C}[\boldsymbol{A}[\boldsymbol{j}]]] = \boldsymbol{A}[\boldsymbol{j}]$.
- Assim, colocamos cada elemento $\boldsymbol{A}[\boldsymbol{j}]$ em sua posição ordenada no vetor $\boldsymbol{B}$:

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

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

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

```
for j← comprimento [A] downto 1
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] - 1
```

j = 3

$$B\left[C\left[A[3]\right]\right] \rightarrow B\left[C[3]\right] \rightarrow B[5] = 3$$

$$C[3] = 5 - 1 = 4$$

Simulação

- Agora, partindo do maior para o menor índice, fazemos $\boldsymbol{B}[\boldsymbol{C}[\boldsymbol{A}[\boldsymbol{j}]]] = \boldsymbol{A}[\boldsymbol{j}]$.
- Assim, colocamos cada elemento $\boldsymbol{A}[\boldsymbol{j}]$ em sua posição ordenada no vetor $\boldsymbol{B}$:

A =

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

C =

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

B =

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

```
for j← comprimento [A] downto 1
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] - 1
```

$j = 2$

$$B\left[C\left[A[2]\right]\right] \rightarrow B\left[C[5]\right] \rightarrow B[8] = 5$$

Simulação

- Agora, partindo do maior para o menor índice, fazemos $\boldsymbol{B}[\boldsymbol{C}[\boldsymbol{A}[\boldsymbol{j}]]] = \boldsymbol{A}[\boldsymbol{j}]$.
- Assim, colocamos cada elemento $\boldsymbol{A}[\boldsymbol{j}]$ em sua posição ordenada no vetor $\boldsymbol{B}$:

A =

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

C =

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

B =

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

```
for j← comprimento [A] downto 1
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] - 1
```

$j = 2$

$B\left[C\left[A[2]\right]\right] \rightarrow B\left[C[5]\right] \rightarrow B[8] = 5$

$C[5] = 8 - 1 = 7$

Simulação

- Agora, partindo do maior para o menor índice, fazemos $\boldsymbol{B}[\boldsymbol{C}[\boldsymbol{A}[\boldsymbol{j}]]] = \boldsymbol{A}[\boldsymbol{j}]$.
- Assim, colocamos cada elemento $\boldsymbol{A}[\boldsymbol{j}]$ em sua posição ordenada no vetor $\boldsymbol{B}$:

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

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

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

```
08 for j← comprimento [A] downto 1
09           B[C[A[j]]] = A[j]
10           C[A[j]] = C[A[j]] - 1
```

j = 1

$$B\big[C[A[1]]\big] \to B\big[C[2]\big] \to B[3] = 2$$

Simulação

- Agora, partindo do maior para o menor índice, fazemos $\boldsymbol{B}[\boldsymbol{C}[\boldsymbol{A}[\boldsymbol{j}]]] = \boldsymbol{A}[\boldsymbol{j}]$.
- Assim, colocamos cada elemento $\boldsymbol{A}[\boldsymbol{j}]$ em sua posição ordenada no vetor $\boldsymbol{B}$:

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

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

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

```
for j← comprimento [A] downto 1
        B[C[A[j]]] = A[j]
        C[A[j]] = C[A[j]] - 1
```

j = 1

$$B\left[C\left[A[1]\right]\right] \rightarrow B\left[C[2]\right] \rightarrow B[3] = 2$$

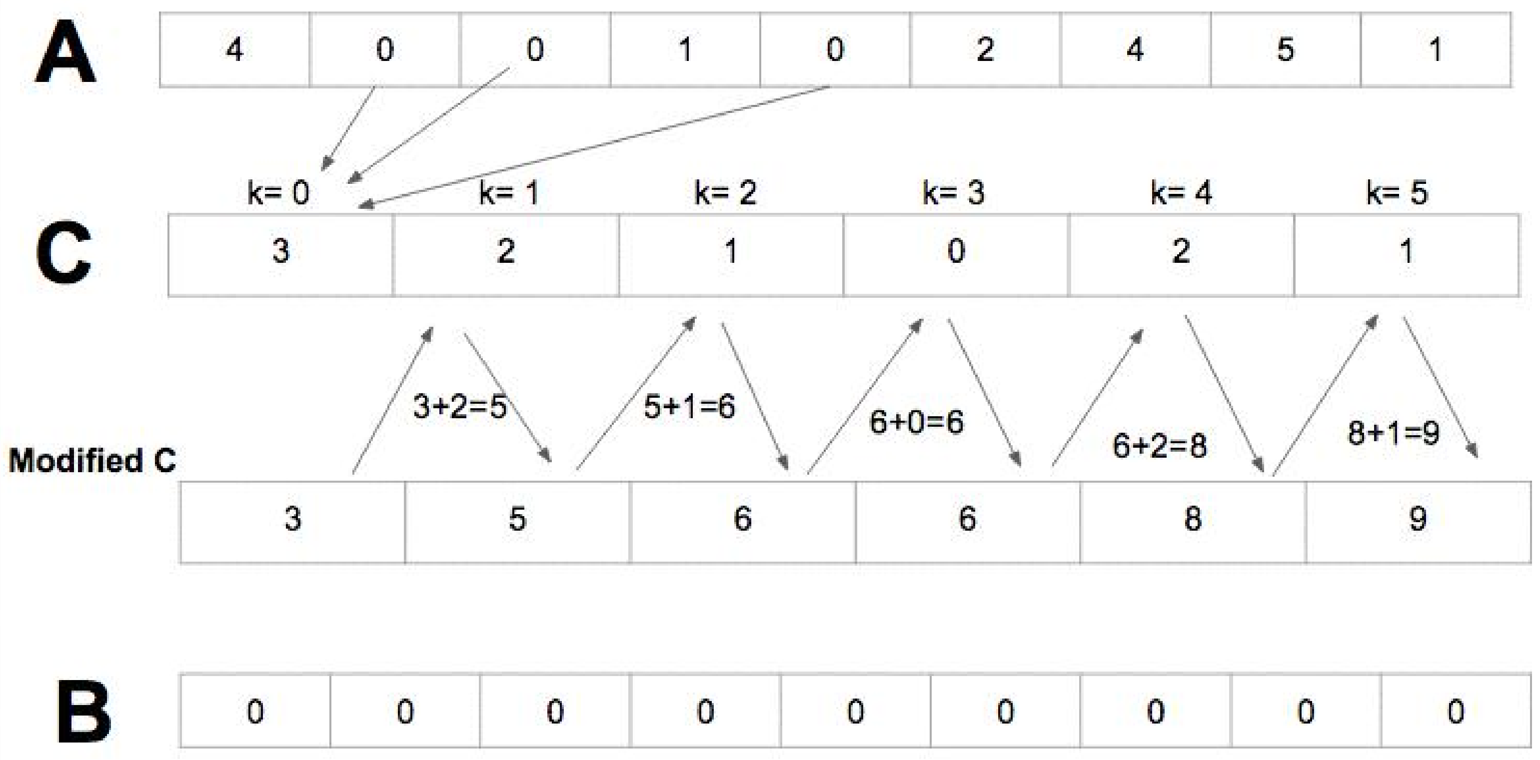
$$C[2] = 3 - 1 = 2$$

Simulação

- Após as iterações de (**comprimento** ***de*** $[A]$ ***até 1***) temos o vetor de saída ***B*** ordenado!!!

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

Animação



A
4
0
0
1
0
2
4
5
1
k= 0
k= 1
k= 2
k= 3
k= 4
k= 5
C
3
2
1
0
2
1
3+2=5
5+1=6
6+0=6
6+2=8
8+1=9
Modified C
3
5
6
6
8
9
B
0
0
0
0
0
0
0
0
0

Animação

Counting Sort... N=10 , K=5

Step - III Fill Result Array

Input Array A.

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

Count Array C.

0	2	4	6	9
0	1	2	3	4

Result Array B.

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

ifce.edu.br

Animação

ifce.edu.br

Counting Sort... N=10 , K=5

Input Array A.

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

Count Array C.

0	0	0	0	0
0	1	2	3	4

Result Array B.

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

Código em Python

```
def Countingsort(lista):
        k = maximo(lista)
        B = [0 for w in range(len(lista))]
        C = [0 for w in range(k+1)]
        for j in range(0,len(lista)):
                C[lista[j]] = C[lista[j]] + 1
        for i in range(1,k+1):
                C[i] += C[i-1]
        for j in range(len(lista)-1,0,-1):
                B[C[lista[j]]-1] = lista[j]
                C[lista[j]] = C[lista[j]] - 1
        return B
```

Exercício de Fixação

- Implemente na linguagem C o algoritmo de ordenação por contagem.

Solução

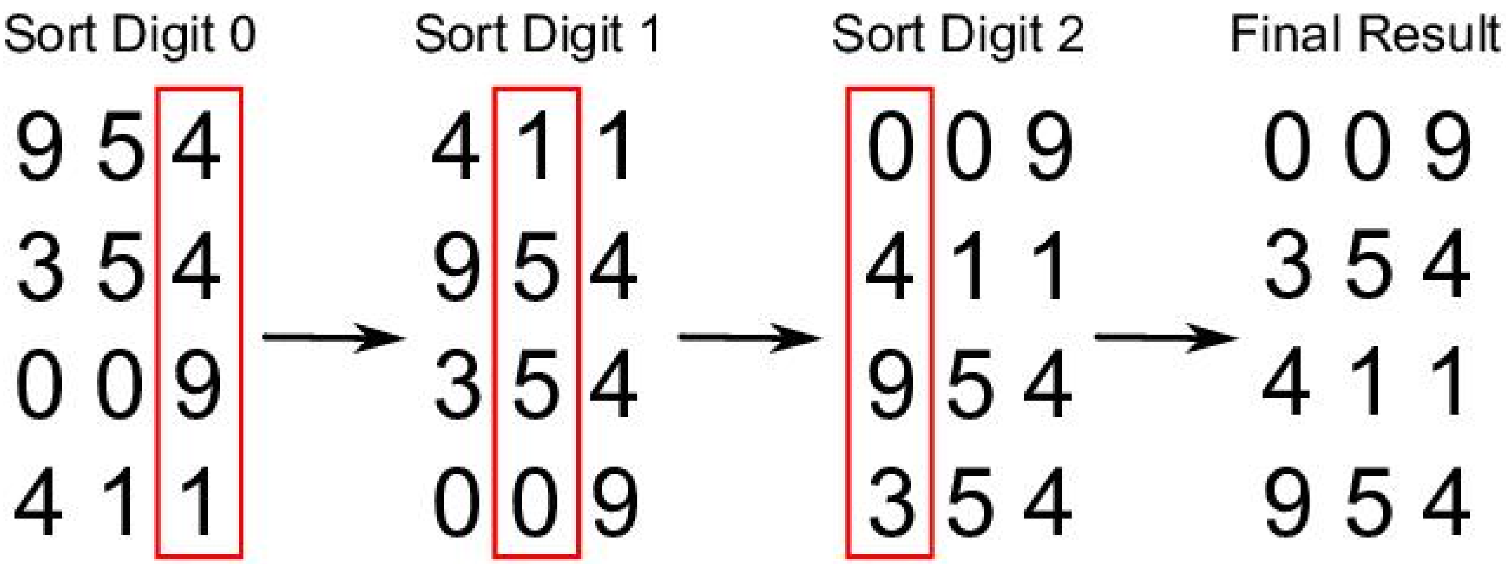
```
void coutingSort(int *A, int n)
{
    int i;
    int k = maximo(A, n); //recebe o maior elemento do vetor
    int B[n], C[k + 1];

    //inicializa o vetor B
    for(i = 0; i < n; i++)
        B[i] = 0;

    //inicializa o vetor C
    for(i = 0; i < k; i++)
        C[i] = 0;

    for(i=0;i<n;i++) C[A[i]-1]++;

    for(i=1;i<k;i++) C[i] += C[i-1];

    for(i=n-1;i>=0;i--)
    {
        B[C[A[i]-1]-1] = A[i];
        C[A[i]-1]--;
    }

    for(i=0;i<n;i++) A[i] = B[i];
}
```

ifce.edu.br

RADIX SORT

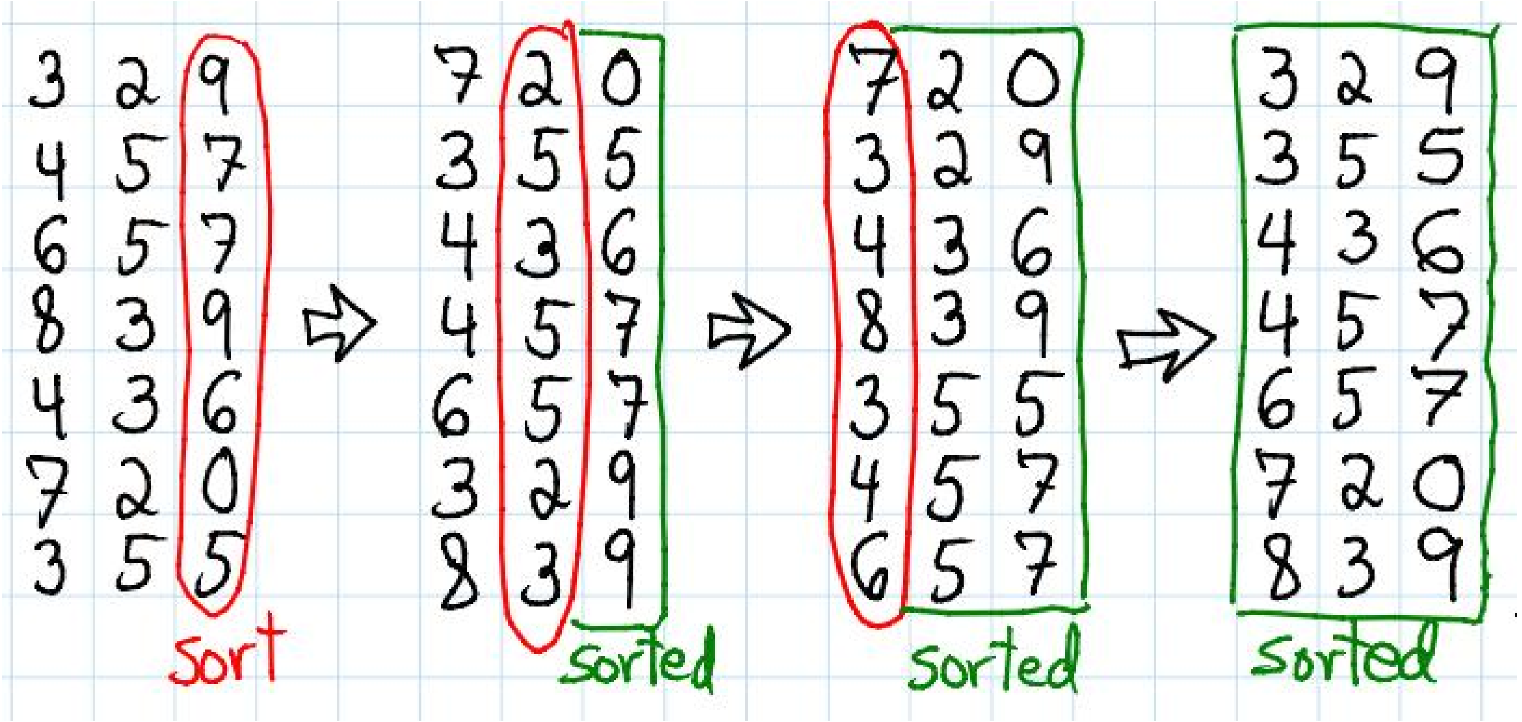
Definição

- **Radix sort** ou **ordenação da raiz** é um algoritmo baseado em ordenação por contagem aplicada a uma parte da representação do elemento, a raiz.



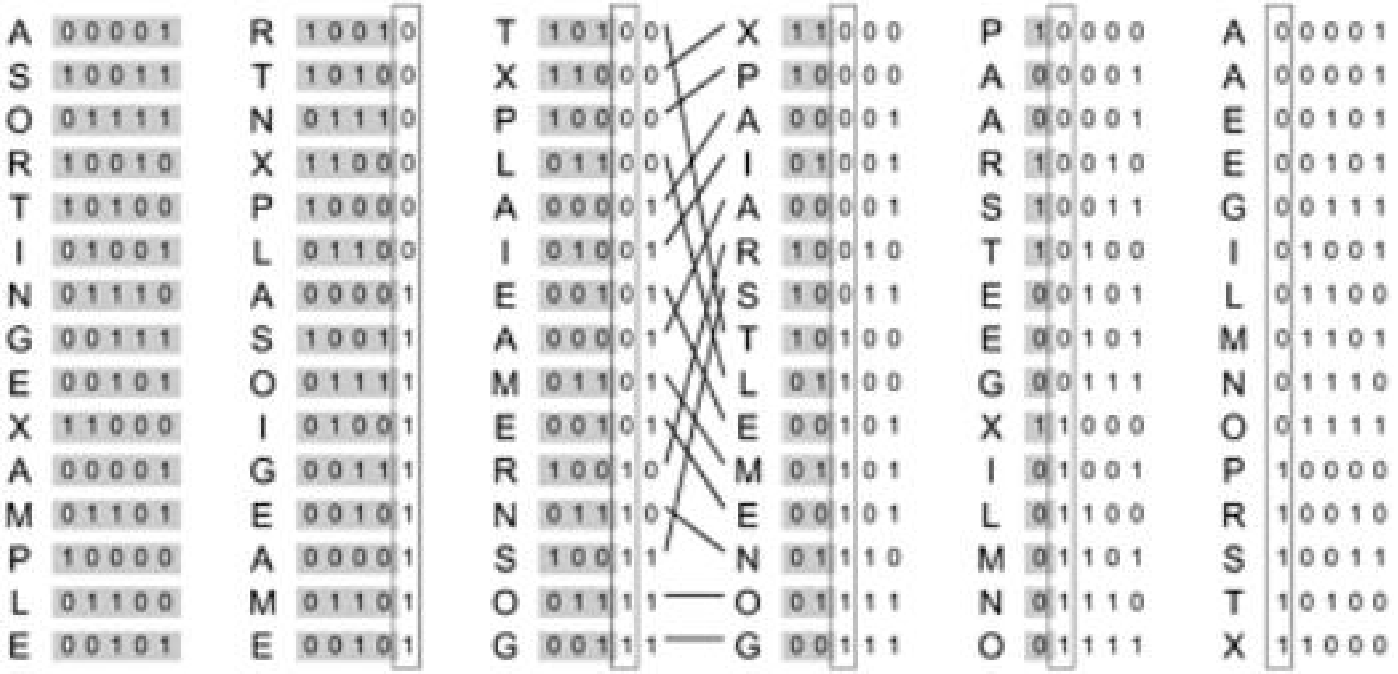
Definição

- O procedimento é repetido para a raiz seguinte até que toda a representação dos elementos tenha sido analisada.



Definição

- Por exemplo, a ordenação de chaves inteiras com 32 bits pode ser realizada em quatro passos usando uma raiz de oito bits, sendo que a tabela de contadores requer apenas 256 (2^8) entradas.



Ideia Básica: 1º passo

Quebrar uma chave em vários pedaços

- Dígitos de um número em uma dada base ($radix$)
  - 312 tem os dígitos 3, 1 e 2 na base 10
  - 312 tem os dígitos 100111000 na base 2
  - “exemplo” tem 6 caracteres (base 256)

Ideia Básica: 2º passo

Ordenar de acordo com o primeiro pedaço

- Números cujo dígito mais à esquerda começa com 0 vêm antes de números cujo dígito mais à esquerda é 1
- Podemos ordenar repetindo esse processo para todos os pedaços.

Simulando:

Ordenando um dígito

- Para isso iremos contar quantos elementos existem de cada valor

0	1	2	3	4	5	6
12**3**	14**2**	08**7**	26**3**	23**3**	01**4**	13**2**

Digito	Contador
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	0
8	0
9	0

Simulando:

Ordenando um dígito

- Para isso iremos contar quantos elementos existem de cada valor

12**3**	14**2**	08**7**	26**3**	23**3**	01**4**	13**2**
0	1	2	3	4	5	6

Digito	Contador
0	0
1	0
2	**2**
3	**3**
4	**1**
5	0
6	0
7	**1**
8	0
9	0

Simulando:

Ordenando um dígito

- Depois calcular a posição deles no vetor ordenado

12**3**	14**2**	08**7**	26**3**	23**3**	01**4**	13**2**
0	**1**	**2**	**3**	**4**	**5**	**6**

Dig	C	Posicao
0	0	0
1	0	0
2	2	0
3	3	2
4	1	5
5	0	0
6	0	0
7	1	6
8	0	0
9	0	0

Simulando:

Ordenando um dígito

- E finalmente colocar os elementos em suas posições

123	142	087	263	233	014	132
		123				
0	1	2	3	4	5	6

Dig	C	Posicao
0	0	0
1	0	0
2	2	0
3	3	3
4	1	5
5	0	0
6	0	0
7	1	6
8	0	0
9	0	0

Atualiza a nova posição

ifce.edu.br

Simulando:

Ordenando um dígito

- Para isso iremos contar quantos elementos existem de cada valor

0	1	2	3	4	5	6
123	142	087	263	233	014	132
142		123				

Dig	C	Posicao
0	0	0
1	0	0
2	2	1
3	3	3
4	1	5
5	0	0
6	0	0
7	1	6
8	0	0
9	0	0

Atualiza a nova posição

Simulando:

Ordenando um dígito

- Para isso iremos contar quantos elementos existem de cada valor

123	142	087	263	233	014	132
142		123				087
0	1	2	3	4	5	6

Dig	C	Posicao
0	0	0
1	0	0
2	2	1
3	3	3
4	1	5
5	0	0
6	0	0
7	1	7
8	0	0
9	0	0

Atualiza a nova posição

Simulando:

Ordenando um dígito

- Para isso iremos contar quantos elementos existem de cada valor

0	1	2	3	4	5	6
123	142	087	263	233	014	132
142		123	263			087

Dig	C	Posicao
0	0	0
1	0	0
2	2	1
3	3	4
4	1	5
5	0	0
6	0	0
7	1	7
8	0	0
9	0	0

Atualiza a nova posição

Simulando:

Ordenando um dígito

- Para isso iremos contar quantos elementos existem de cada valor

123	142	087	263	233	014	132
142		123	263	233		087
0	1	2	3	4	5	6

Dig	C	Posicao
0	0	0
1	0	0
2	2	1
3	3	5
4	1	5
5	0	0
6	0	0
7	1	7
8	0	0
9	0	0

Atualiza a nova posição

Simulando:

Ordenando um dígito

- Para isso iremos contar quantos elementos existem de cada valor

123	142	087	263	233	014	132
142		123	263	233	014	087
0	1	2	3	4	5	6

Dig	C	Posicao
0	0	0
1	0	0
2	2	1
3	3	5
4	1	6
5	0	0
6	0	0
7	1	7
8	0	0
9	0	0

Atualiza a nova posição

Simulando:

Ordenando um dígito

- Para isso iremos contar quantos elementos existem de cada valor

0	1	2	3	4	5	6
123	142	087	263	233	014	132
142	132	123	263	233	014	087

Dig	C	Posicao
0	0	0
1	0	0
2	2	1
3	3	5
4	1	6
5	0	0
6	0	0
7	1	7
8	0	0
9	0	0

Simulando:

Ordenando um dígito

- Repetimos o mesmo processo para o próximo dígito
- Funciona porque o método do contador que usamos anteriormente é estável!

12**3**	14**2**	08**7**	26**3**	23**3**	01**4**	13**2**
1**4**2	1**3**2	1**2**3	2**6**3	2**3**3	0**1**4	0**8**7
0**1**4	1**2**3	1**3**2	2**3**3	1**4**2	2**6**3	0**8**7

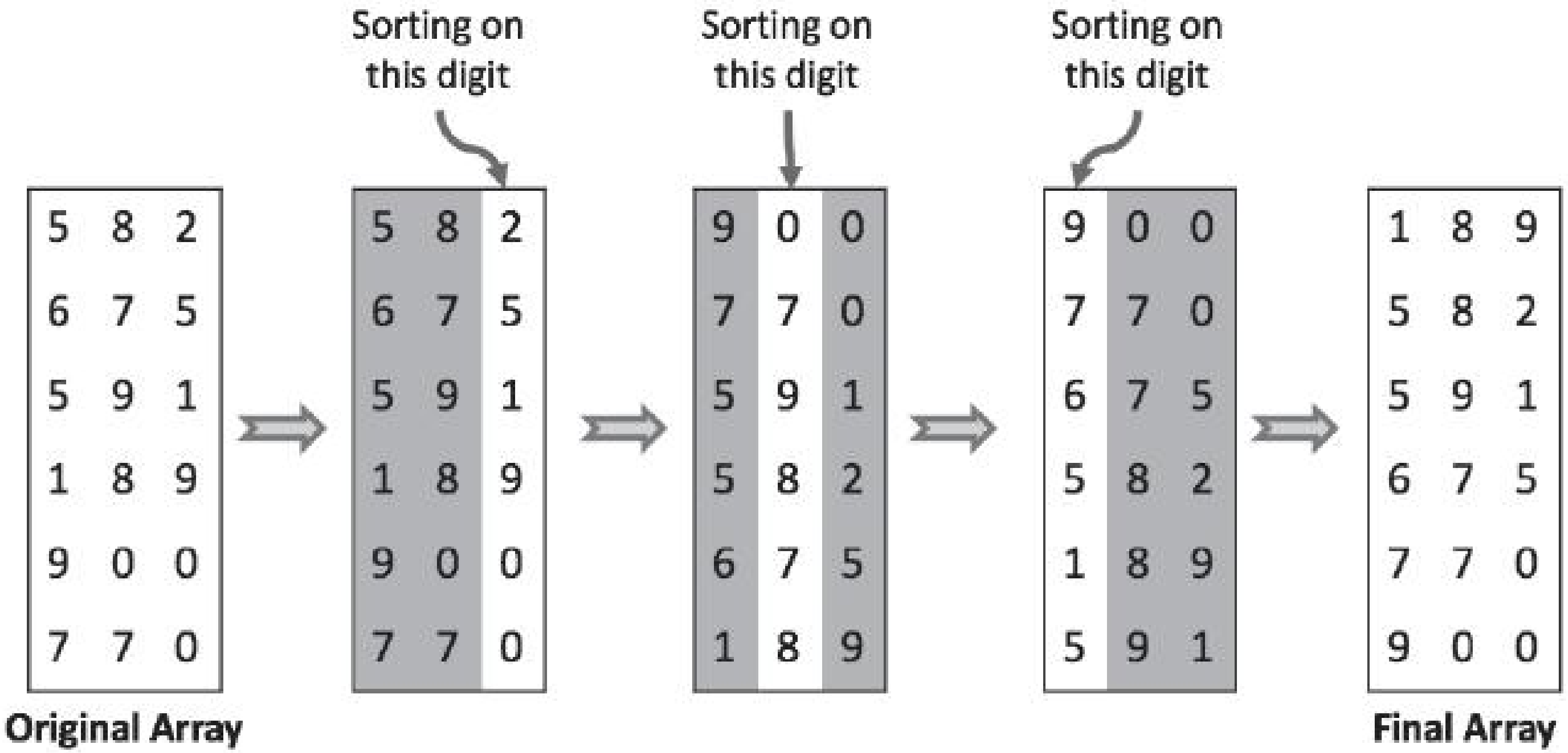
Simulando:

Ordenando um dígito

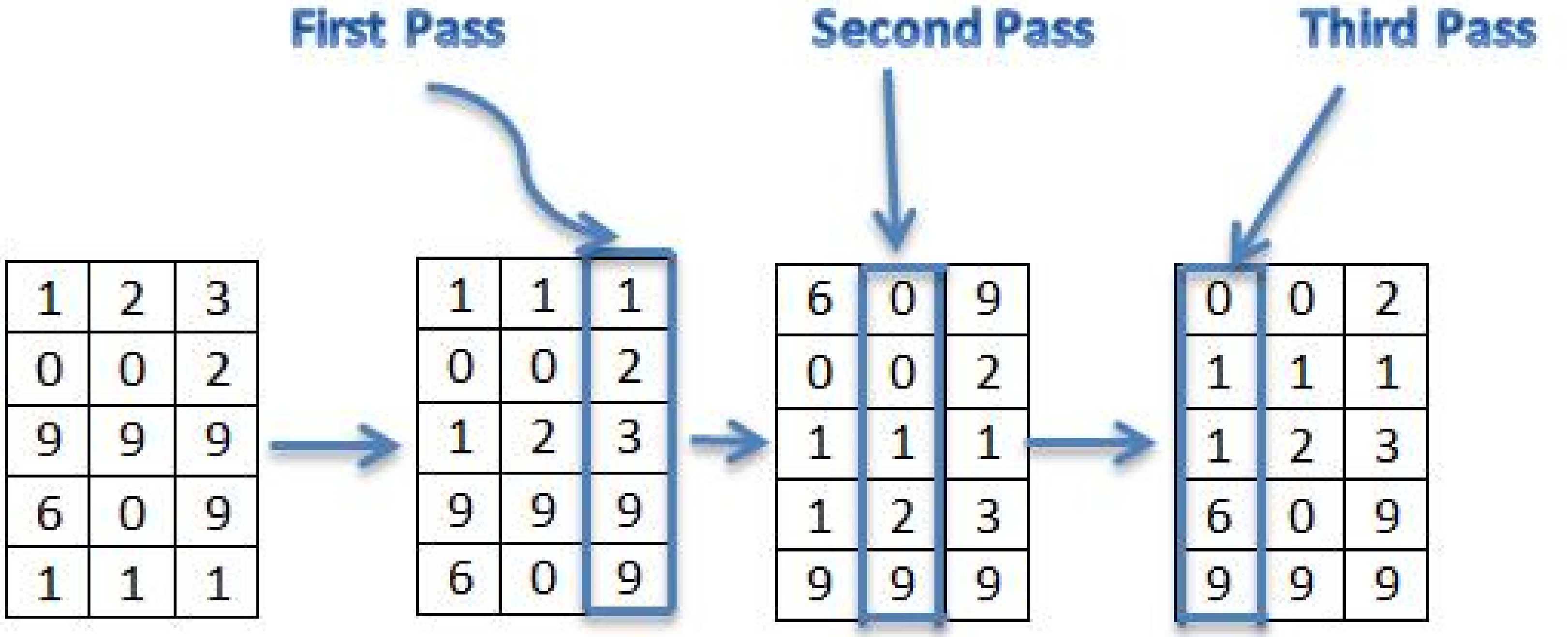
- Repetimos o mesmo processo para o próximo dígito
- Funciona porque o método do contador que usamos anteriormente é estável!

12**3**	14**2**	08**7**	26**3**	23**3**	01**4**	13**2**
1**4**2	1**3**2	1**2**3	2**6**3	2**3**3	0**1**4	0**8**7
014	**1**23	**1**32	**2**33	**1**42	**2**63	**0**87
014	087	123	132	142	233	263

Ilustração


Sorting on this digit
Sorting on this digit
Sorting on this digit
5 8 2
6 7 5
5 9 1
1 8 9
9 0 0
7 7 0
5 8 2
6 7 5
5 9 1
1 8 9
9 0 0
7 7 0
9 0 0
7 7 0
5 9 1
5 8 2
6 7 5
1 8 9
9 0 0
7 7 0
6 7 5
5 8 2
1 8 9
5 9 1
1 8 9
5 8 2
5 9 1
6 7 5
7 7 0
9 0 0
Original Array
Final Array

Ilustração


First Pass
Second Pass
Third Pass
1 2 3
0 0 2
9 9 9
6 0 9
1 1 1
1 1 1
0 0 2
1 2 3
9 9 9
6 0 9
6 0 9
0 0 2
1 1 1
1 2 3
9 9 9
0 0 2
1 1 1
1 2 3
6 0 9
9 9 9

Radix Sort em C

```c
for(w = 0; w < num_digitos; w++)
{
    for(j = 0; j < base; j++)
    {
        count[j] = 0;
        posicao[j] = 0;
    }

    for(i = 0; i < n; i++)
    {
        d = digito(v[i], w, base);
        count[d]++;
    }

    for(j = 1; j < base; j++)
        posicao[j] = posicao[j-1] + count[j-1];

    for(i = 0; i < n; i++)
    {
        d = digito(v[i], w, base);
        aux[posicao[d]++] = v[i];
    }

    for(i = 0; i < n; i++)
        v[i] = aux[i];
}
```

```c
int digito(int v, int w, int base)
{
    int i = -1, algarismo;

    do{

        i++;
        algarismo = v % base;
        v = v/base;

    }while( i != w );

    return algarismo;

}
```

Vantagens vs. Desvantagens

- **Vantagens:**
  - Estável
  - Não compara as chaves

- **Desvantagens:**
  - Nem sempre é fácil otimizar a inspeção de dígitos
  - Depende do hardware
  - Só é bom se o número de dígitos for pequeno
  - Em geral o número de dígitos tem crescimento $O(\lg(n))$

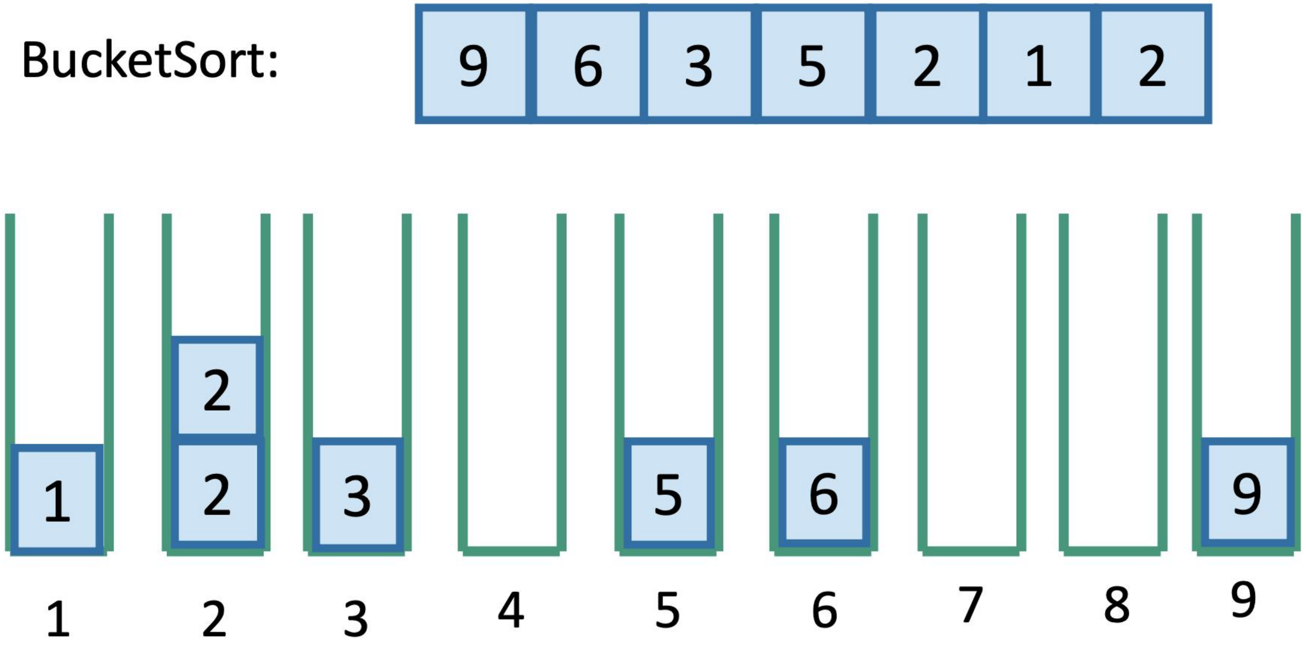
Análise de Complexidade

- Nenhuma comparação
- **Inspeções de dígitos:**
  - $\boldsymbol{2 \times n \times num\_digitos}$
  - Se **num_digitos** for pequeno ou constante, então radixsort tem custo linear $O(n)$
- **Trocas:**
  - $\boldsymbol{n \times num\_digitos}$
  - Número de trocas também é $O(n)$
- Quicksort é comparável ao radixsort porque o número de dígitos é da ordem de $\lg(n)$ na base 2 e $\log_{10}(n)$ na base 10

BUCKET SORT

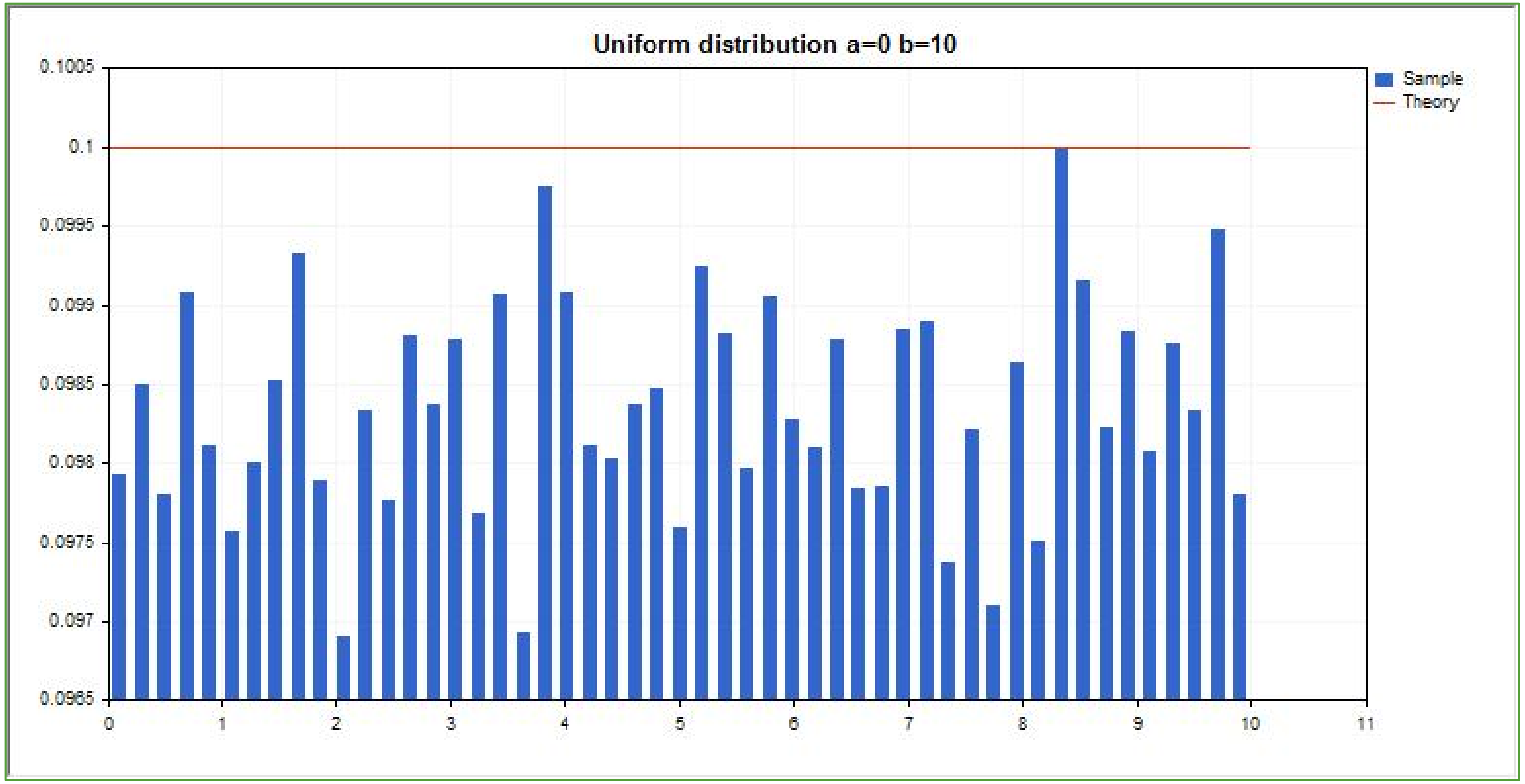
Definição

- **Bucket sort (ou ordenação por balde)** é um algoritmo de ordenação que funciona dividindo um vetor em um número finito de recipientes.



Definição

- Funciona em tempo linear quando a entrada é gerada a partir de uma distribuição uniforme.



Counting Sort vs Bucket Sort

A ordenação por contagem presume que a entrada consiste em inteiros em um intervalo pequeno,

Bucket sort presume que a entrada é gerada por um processo aleatório que distribui elementos uniformemente sobre o intervalo $[0, 1)$

Etapas

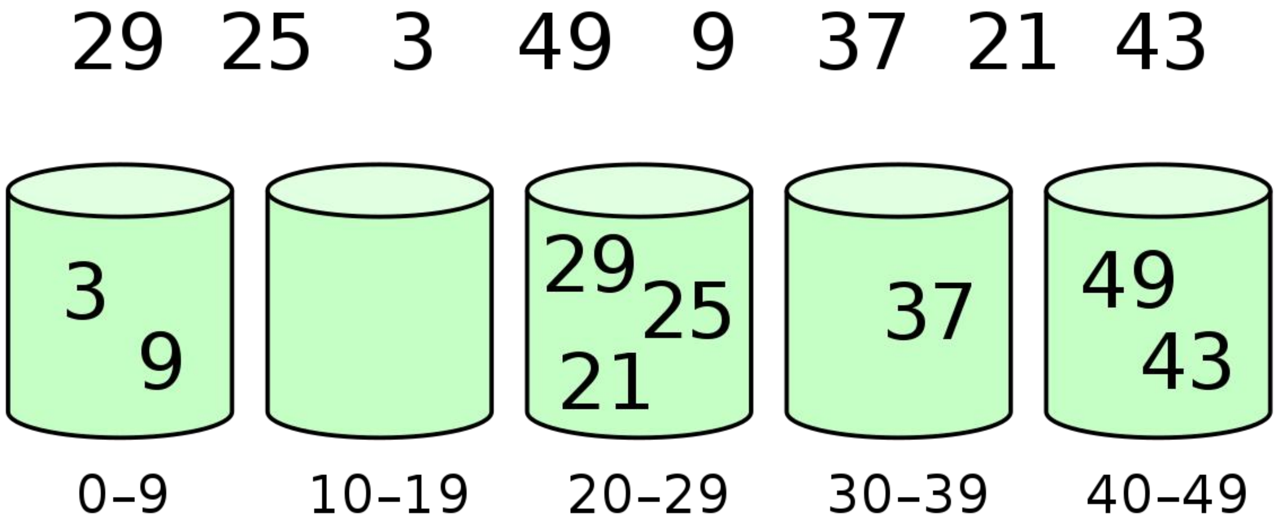
Funciona dividindo um vetor em um número finito de recipientes.

Cada recipiente é então ordenado individualmente, seja usando um algoritmo de ordenação diferente, ou usando o algoritmo bucket sort recursivamente.

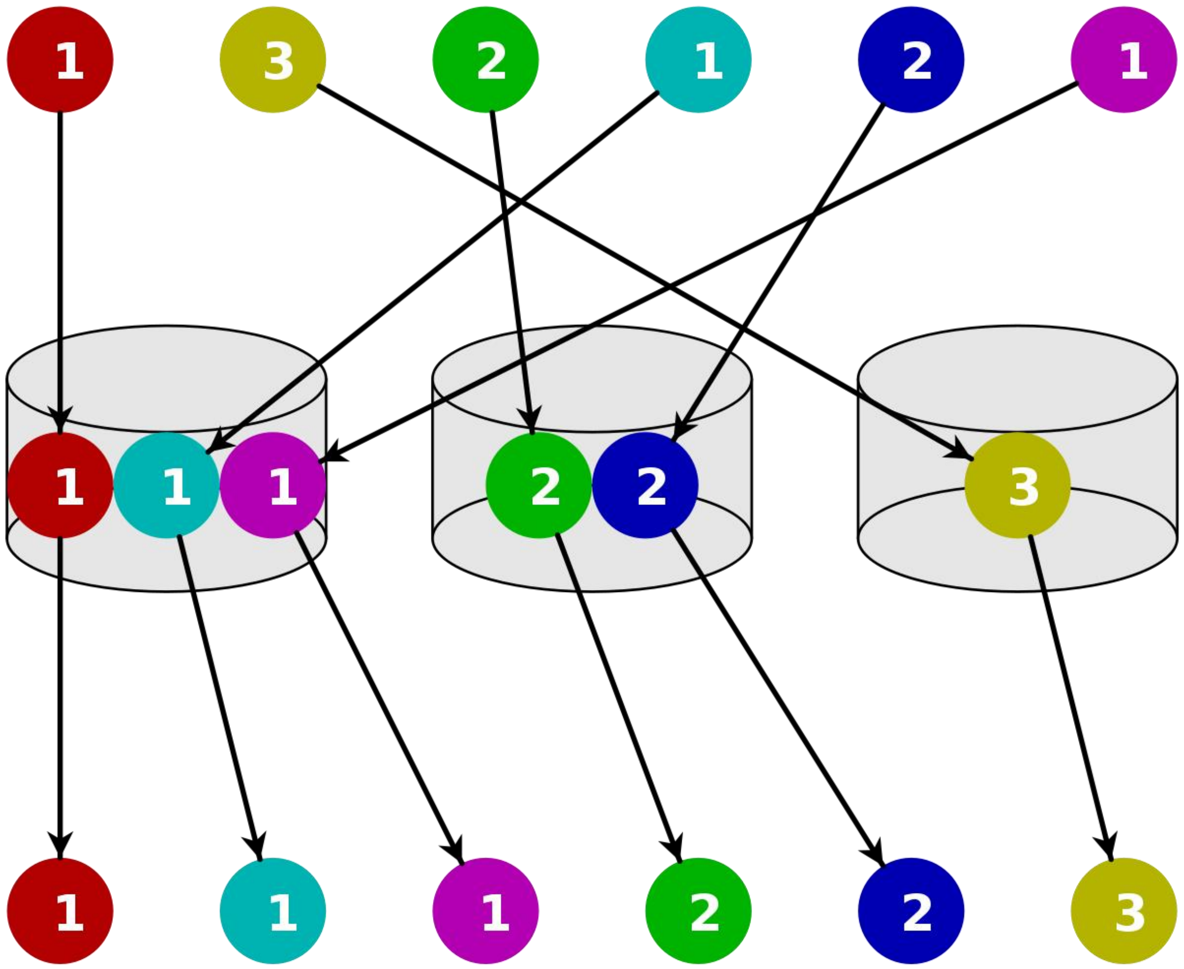
Supõe que os n elementos da entrada estão distribuídos uniformemente no intervalo $[0, 1)$.

Ideia Básica

- A ideia é dividir o intervalo $[0, 1)$ em $\boldsymbol{n}$ segmentos de mesmo tamanho (buckets) e distribuir os $\boldsymbol{n}$ elementos nos seus respectivos segmentos.
- Como os elementos estão distribuídos uniformemente, espera-se que o número de elementos seja aproximadamente o mesmo em todos os segmentos.



Ideia Básica



- Em seguida, os elementos de cada segmento são ordenados por um método qualquer.
- Finalmente, os segmentos ordenados são concatenados em ordem crescente.

Simulação

- Recebe um inteiro $\boldsymbol{n}$ e um vetor $A[1 \cdots n]$ onde cada elemento é um número no intervalo $[\mathbf{0},\ \mathbf{1})$.

10 elementos

47	93	82	12	42	3	62	38	77	91

0.47	0.93	0.82	0.12	0.42	0.03	0.62	0.38	0.77	0.91

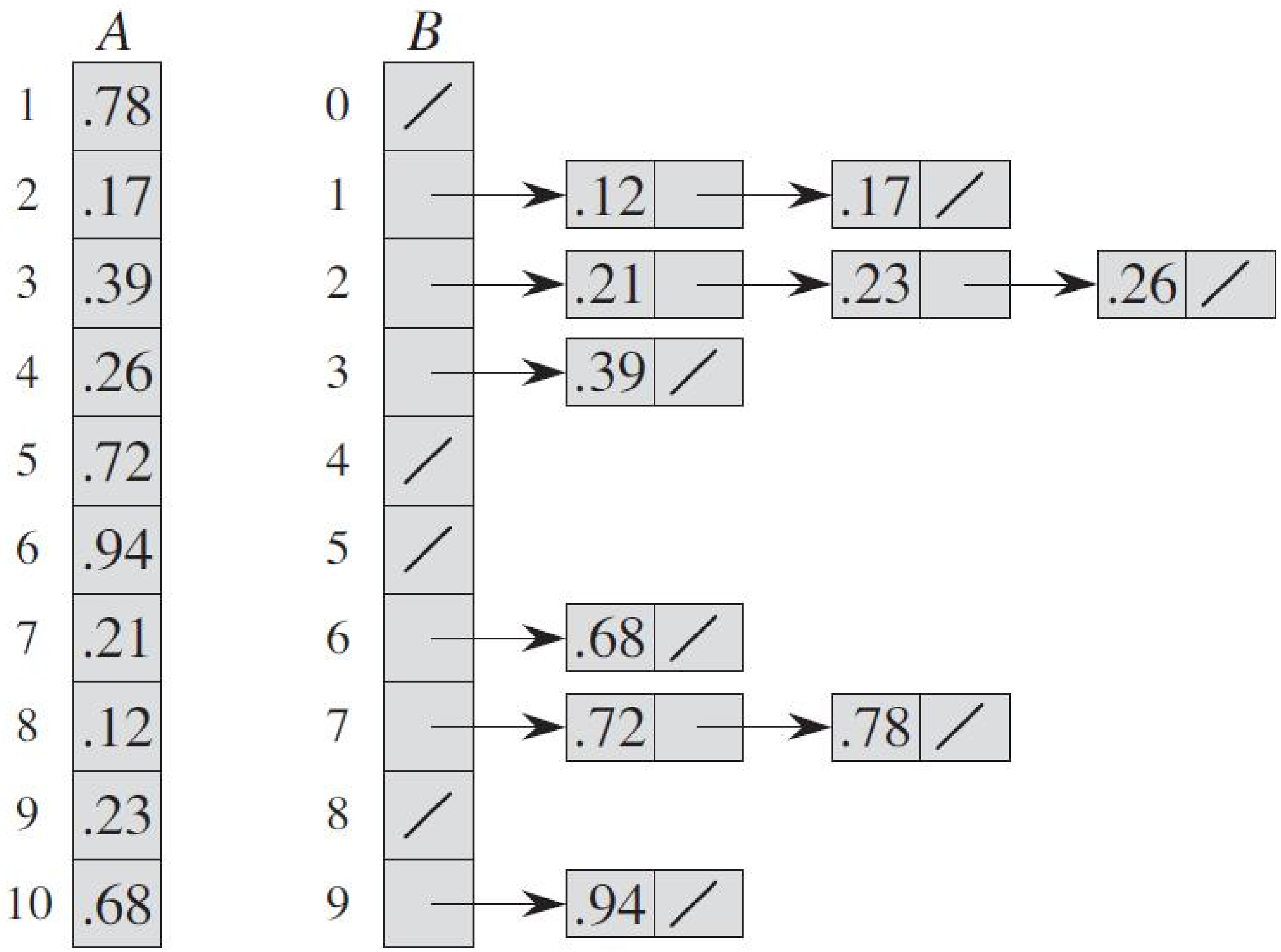
Simulação

- Distribui os elementos em cada recipiente de 0 até 9 de acordo com o dígito mais à esquerda na parte fracionária.

0.47	0.93	0.82	0.12	0.42	0.03	0.62	0.38	0.77	0.91

$B[0]$	0.03
$B[1]$	0.12
$B[2]$	
$B[3]$	0.38
$B[4]$	0.47, 0.42
$B[5]$	
$B[6]$	0.62
$B[7]$	0.77
$B[8]$	0.82
$B[9]$	0.93, 0.91

Simulação



A
B
1 .78
2 .17
3 .39
4 .26
5 .72
6 .94
7 .21
8 .12
9 .23
10 .68
0
1 .12 .17
2 .21 .23 .26
3 .39
4
5
6 .68
7 .72 .78
8
9 .94

Simulação

- Aplica uma ordenação em cada recipiente

$\boldsymbol{B}[0]$	0.03
$\boldsymbol{B}[1]$	0.12
$\boldsymbol{B}[2]$	
$\boldsymbol{B}[3]$	0.38
$\boldsymbol{B}[4]$	$\mathbf{0.47,\ 0.42}$
$\boldsymbol{B}[5]$	
$\boldsymbol{B}[6]$	0.62
$\boldsymbol{B}[7]$	0.77
$\boldsymbol{B}[8]$	0.82
$\boldsymbol{B}[9]$	$\mathbf{0.93,\ 0.91}$

$\boldsymbol{B}[0]$	0.03
$\boldsymbol{B}[1]$	0.12
$\boldsymbol{B}[2]$	
$\boldsymbol{B}[3]$	0.38
$\boldsymbol{B}[4]$	0.42, 0.47
$\boldsymbol{B}[5]$	
$\boldsymbol{B}[6]$	0.62
$\boldsymbol{B}[7]$	0.77
$\boldsymbol{B}[8]$	0.82
$\boldsymbol{B}[9]$	0.91, 0.93

ifce.edu.br

Simulação

- Concatena os números a partir de $\boldsymbol{B}[\mathbf{0}]$ até $\boldsymbol{B}[\mathbf{9}]$

$\boldsymbol{B}[\mathbf{0}]$	0.03
$\boldsymbol{B}[\mathbf{1}]$	0.12
$\boldsymbol{B}[\mathbf{2}]$	
$\boldsymbol{B}[\mathbf{3}]$	0.38
$\boldsymbol{B}[\mathbf{4}]$	0.42, 0.47
$\boldsymbol{B}[\mathbf{5}]$	
$\boldsymbol{B}[\mathbf{6}]$	0.62
$\boldsymbol{B}[\mathbf{7}]$	0.77
$\boldsymbol{B}[\mathbf{8}]$	0.82
$\boldsymbol{B}[\mathbf{9}]$	0.91, 0.93

0.03	0.12	0.38	0.42	0.47	0.62	0.77	0.82	0.91	0.93

03	12	38	42	47	62	77	82	91	93

Vetor Ordenado de modo crescente

Algoritmo

- Recebe um inteiro $\boldsymbol{n}$ e um vetor $\boldsymbol{A}[\mathbf{1} \cdots \boldsymbol{n}]$ onde cada elemento é um número no intervalo $[\mathbf{0},\ \mathbf{1})$.
- Devolve um vetor $C[1 \cdots n]$ com os elementos de $A[1 \cdots n]$ em ordem crescente

```
BUCKETSORT(A, n)
    para i ← 0 até n − 1 faça
        B[i] ← NIL
    para i ← 1 até n faça
        INSIRA(B[⌊n A[i]⌋], A[i])
    para i ← 0 até n − 1 faça
        ORDENELISTA(B[i])
    C ← CONCATENE(B, n)
    devolva C
```

Algoritmo

BUCKETSORT(A, n)

para $i \leftarrow 0$ **até** $n - 1$ **faça**
    $B[i] \leftarrow$ NIL
para $i \leftarrow 1$ **até** n **faça**
    INSIRA$(B[\lfloor n\, A[i] \rfloor], A[i])$
para $i \leftarrow 0$ **até** $n - 1$ **faça**
    ORDENELISTA$(B[i])$
$C \leftarrow$ CONCATENE(B, n)
devolva C

INSIRA(p, x): insere x na lista apontada por p

ORDENELISTA(p): ordena a lista apontada por p

CONCATENE(B, n): devolve a lista obtida da concatenação das listas apontadas por $B[0], \ldots, B[n-1]$.

Análise de Complexidade

- Suponha que os números em $\boldsymbol{A}[\mathbf{1}\cdots\boldsymbol{n}]$ são uniformemente distribuídos no intervalo $[\mathbf{0},\ \mathbf{1})$.
- Suponha que o $OrdeneLista(\cdots)$ seja o Insertion Sort
- Seja $\boldsymbol{X_i}$ o número de elementos na lista $\boldsymbol{B}[\boldsymbol{i}]$.

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

- Observe que $X_i = \sum_j X_{ij}$

Análise de Complexidade

- $\boldsymbol{X_i}$: o número de elementos na lista $\boldsymbol{B}[\boldsymbol{i}]$.

$$X_{ij} = \begin{cases} 1 & \text{se o } j\text{-ésimo elemento foi para a lista } B[i] \\ 0 & \text{se o } j\text{-ésimo elemento não foi para a lista } B[i]. \end{cases}$$

- $\boldsymbol{Y_i}$: número de comparações para ordenar a lista $\boldsymbol{B}[\boldsymbol{i}]$.
- Observe que $Y_i \leq X_i^2$
- Logo, $E[Y_i] \leq E[X_i^2] = E\left[\left(\sum_j X_{ij}\right)^2\right]$

Análise de Complexidade

$$E\left[\left(\sum_{j} X_{ij}\right)^2\right] = E\left[\sum_{k}\sum_{j} X_{ij} \cdot X_{ik}\right]$$

$$= E\left[\sum_{j} (X_{ij})^2 + \sum_{j}\sum_{k \neq j} X_{ij}X_{ik}\right] = E\left[\sum_{j} (X_{ij})^2\right] + E\left[\sum_{j}\sum_{k \neq j} X_{ij}X_{ik}\right]$$

$$= \sum_{j} E\left[(X_{ij})^2\right] + \sum_{j}\sum_{k \neq j} E[X_{ij}X_{ik}] \Rightarrow E[Y_i] \leq \sum_{j} E\left[(X_{ij})^2\right] + \sum_{j}\sum_{k \neq j} E[X_{ij}X_{ik}]$$

Análise de Complexidade

- Observe que $(X_{ij})^2$ é uma variável aleatória binária.
- Vamos calcular sua esperança:

$$E\left[\left(X_{ij}\right)^2\right] = Pr\left[\left(X_{ij}\right)^2 = 1\right] = Pr\left[X_{ij} = 1\right] = \frac{1}{n}$$

- Para calcular $E[X_{ij} \cdot X_{ik}]$ para $j \neq k$, primeiro note que X_{ij} e X_{ik} são variáveis aleatórias independentes.
- Portanto, $E\left[X_{ij} \cdot X_{ik}\right] = E\left[X_{ij}\right] \cdot E[X_{ik}]$.
- Ademais, $E[X_{ij}] = \Pr[X_{ij} = 1] = \frac{1}{n}$

Análise de Complexidade

$$E\left[\left(X_{ij}\right)^2\right] = Pr\left[\left(X_{ij}\right)^2 = 1\right] = Pr\left[X_{ij} = 1\right] = \frac{1}{n}$$

$$E[Y_i] \leq \sum_j E\left[\left(X_{ij}\right)^2\right] + \sum_j \sum_{k \neq j} E\left[X_{ij}X_{ik}\right] \Longrightarrow E[Y_i] \leq \sum_j \frac{1}{n} + \sum_j \sum_{k \neq j} \frac{1}{n^2}$$

$$E[Y_i] \leq \frac{n}{n} + n(n-1)\frac{1}{n^2} = 1 + (n-1)\frac{1}{n}$$

$$E[Y_i] \leq 2 - \frac{1}{n}$$

Análise de Complexidade

- Agora, seja $Y = \sum_i Y_i$.
- Note que $\boldsymbol{Y}$ é o número de comparações realizadas pelo **BUCKETSORT** no total.
- Assim $\boldsymbol{E}[\boldsymbol{Y}]$ é o número esperado de comparações realizadas pelo algoritmo, e tal número determina o consumo assintótico de tempo do **BUCKETSORT**.
- Mas então $E[Y] = \sum_i E[Yi] \leq 2n - 1 = O(n)$.

Análise de Complexidade

O consumo de tempo esperado do BUCKETSORT quando os números em $A[1\,..\,n]$ são uniformemente distribuídos no intervalo $[0,1)$ é $\mathrm{O}(n)$.

Dúvidas?

Videoaula

- Counting Sort | GeeksforGeeks
- https://www.geeksforgeeks.org/counting-sort/

https://youtu.be/7zuGmKfUt7s

Vídeo-Aula

- Counting Sort | GeeksforGeeks
- https://www.geeksforgeeks.org/radix-sort/

https://youtu.be/nu4gDuFabIM

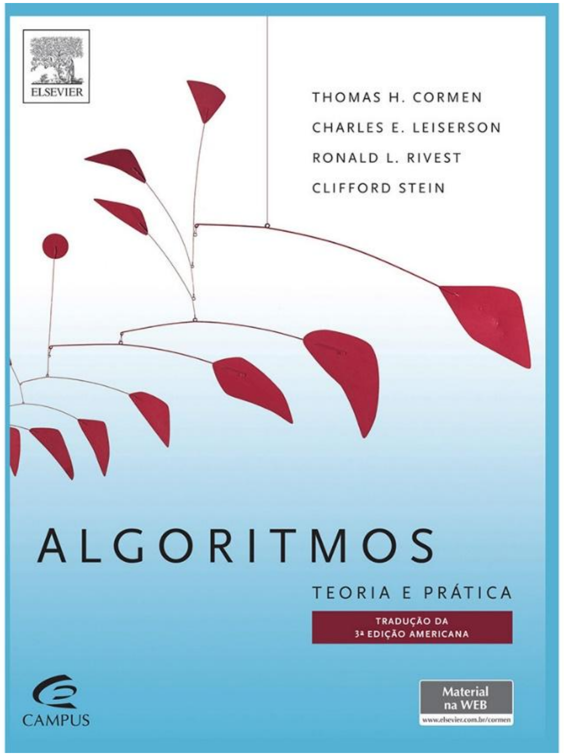
GeeksforGeeks

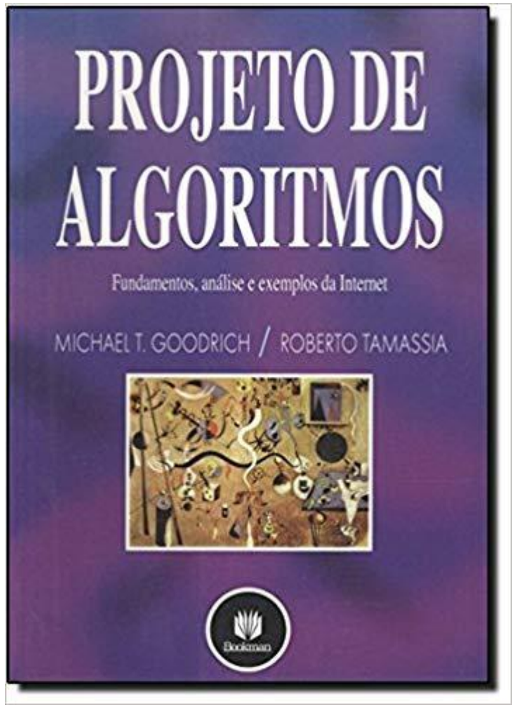
https://youtu.be/VuXbEb5ywrU

Bibliografia

- CORMEN, Thomas *et al*. Algoritmos: teoria e prática. Rio de Janeiro: Elsevier, 2002.

- Capítulo 08

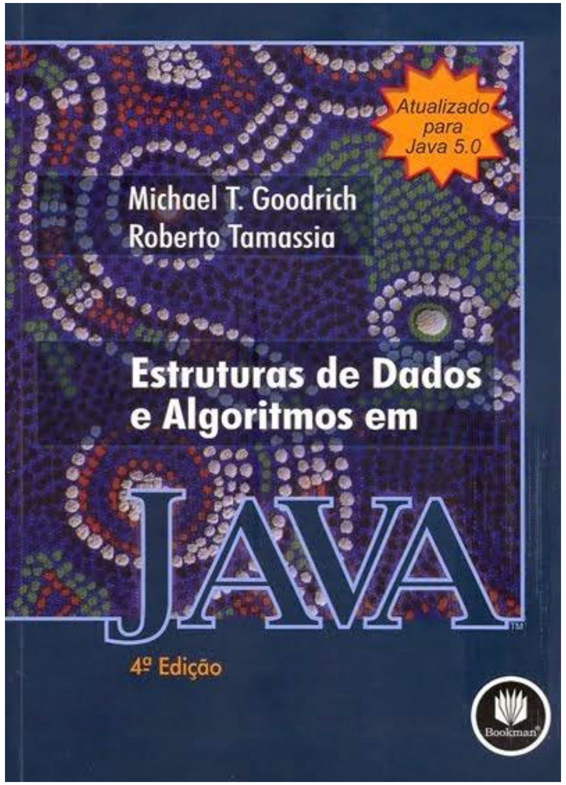




- GOODRICH, Michael T.; TAMASSIA, Roberto. **Projeto de algoritmos: fundamentos, análise e exemplos da internet**. Bookman Editora, 2009.

- Capítulo 04

Bibliografia



- GOODRICH, Michael T.; TAMASSIA, Roberto. **Estruturas de Dados & Algoritmos em Java**. Bookman Editora, 2013.
- Capítulo 11