

Estruturas de Dados

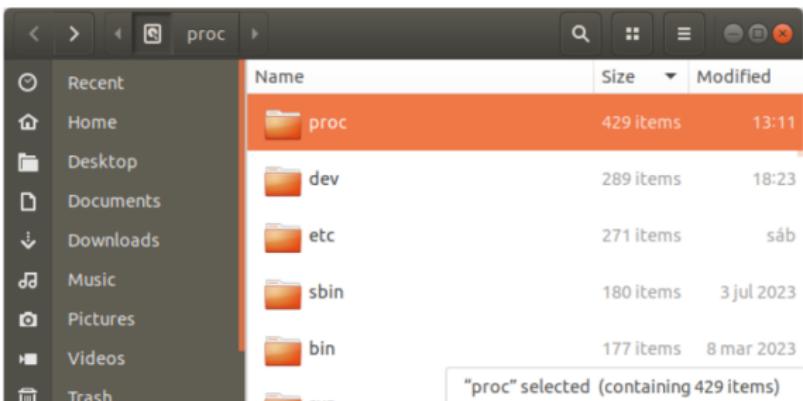
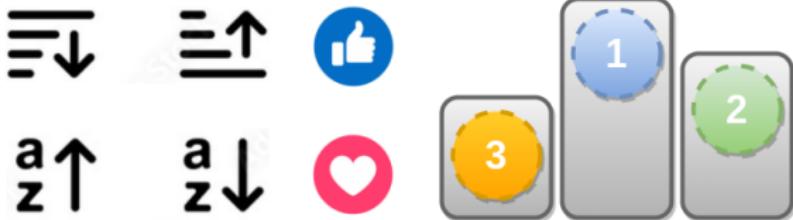
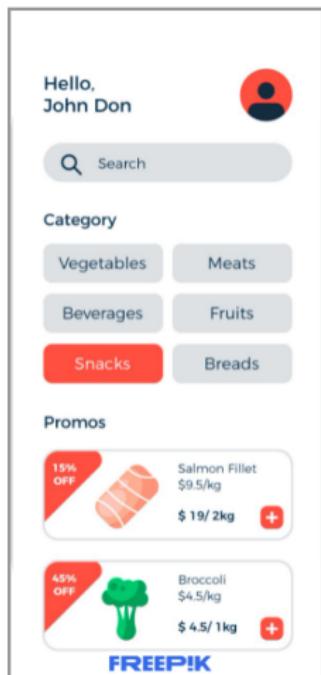
Mayrton Dias

mayrton.queiroz@p.ficr.edu.br

- 1** Introdução
- 2** Bubble Sort
- 3** Insertion Sort
- 4** Quick Sort
- 5** Counting Sort
- 6** Radix Sort

Introdução

Introdução



Entrada:

Uma sequência de n números $\langle a_1, a_2, \dots, a_n \rangle$.

Saída

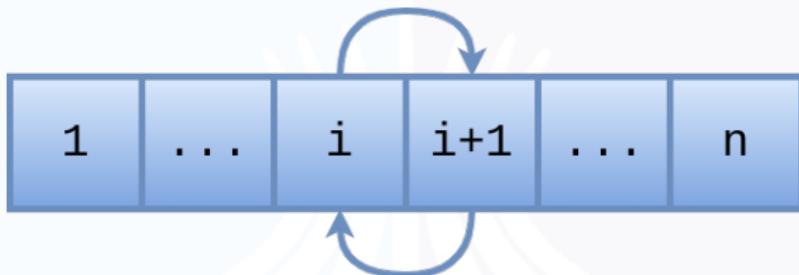
Uma permutação (reordenação) $\langle a'_1, a'_2, \dots, a'_n \rangle$ da sequência de entrada, tal que: $a'_1 \leq a'_2 \leq \dots \leq a'_n$.

- Ascendente ou decrescente
- Ordenação interna e externa

Bubble Sort

Algoritmo de ordenação por troca

- Cada elemento de posição i será comparado com o elemento de posição $i + 1$

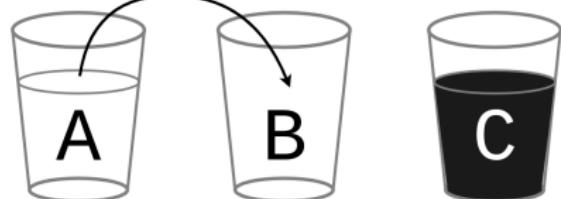


Troça de conteúdo entre duas variáveis

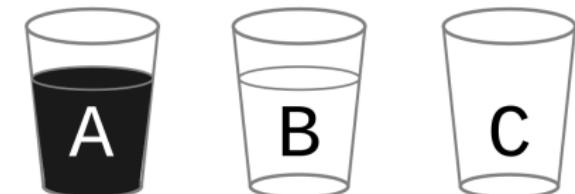
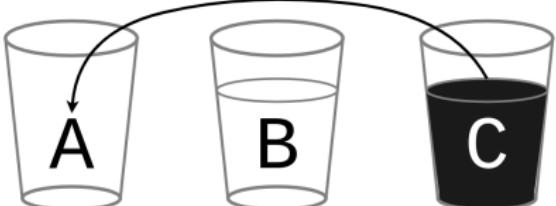
1 C recebe o conteúdo de B



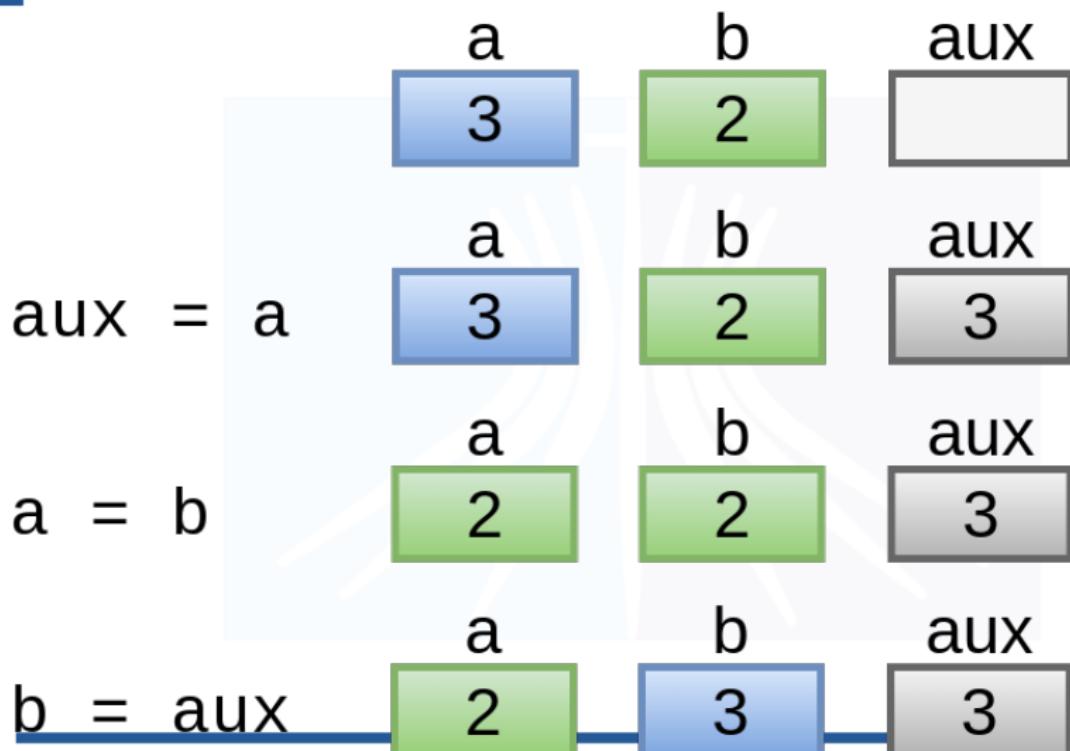
2 B recebe o conteúdo de A



3 A recebe o conteúdo de C



Troça de conteúdo entre duas variáveis



Algoritmo de ordenação por troca

- Passo $i = 4$

0	1	2	3	4
5	4	2	1	8

troca $v[0]$ e $v[1]$

4	5	2	1	8
---	---	---	---	---

troca $v[1]$ e $v[2]$

4	2	5	1	8
---	---	---	---	---

troca $v[2]$ e $v[3]$

4	2	1	5	8
---	---	---	---	---

não troca

Algoritmo de ordenação por troca

- Passo $i = 3$

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

troca $v[0]$ e $v[1]$

troca $v[1]$ e $v[2]$

não troca

Algoritmo de ordenação por troca

- Passo $i = 2$



troca $v[0]$ e $v[1]$

não troca

Algoritmo de ordenação por troca

- Passo $i = 1$



não troca



ordenado

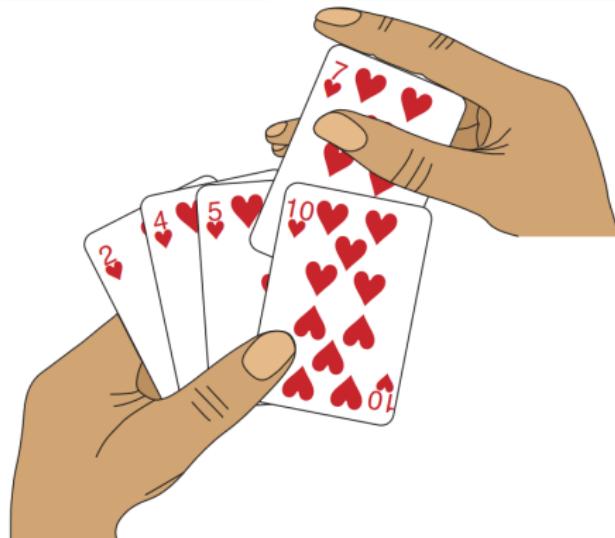
Algoritmo de ordenação por troca

```
1 void bolha(int n, int *v){  
2     int i, j, temp;  
3     for(i = n-1; i >= 1; --i) {  
4         for(j = 0; j < i; ++j) {  
5             if(v[j] > v[j+1]) {  
6                 temp = v[j]; /*troca*/  
7                 v[j] = v[j+1];  
8                 v[j+1] = temp;  
9             }  
10        }  
11    }  
12}
```

Insertion Sort

Algoritmo de inserção

- Similar ao modo em que cartas de baralho são ordenadas na mão de um jogador



Cormen et al. (2012)

Algoritmo de inserção



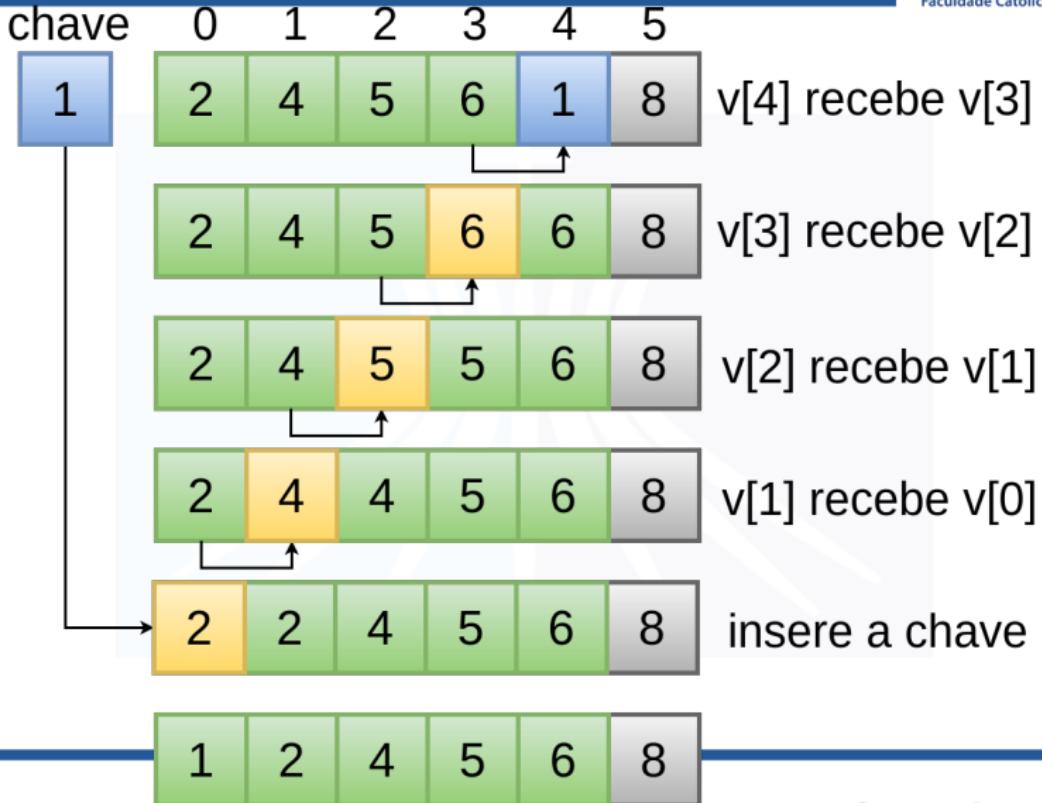
Algoritmo de inserção



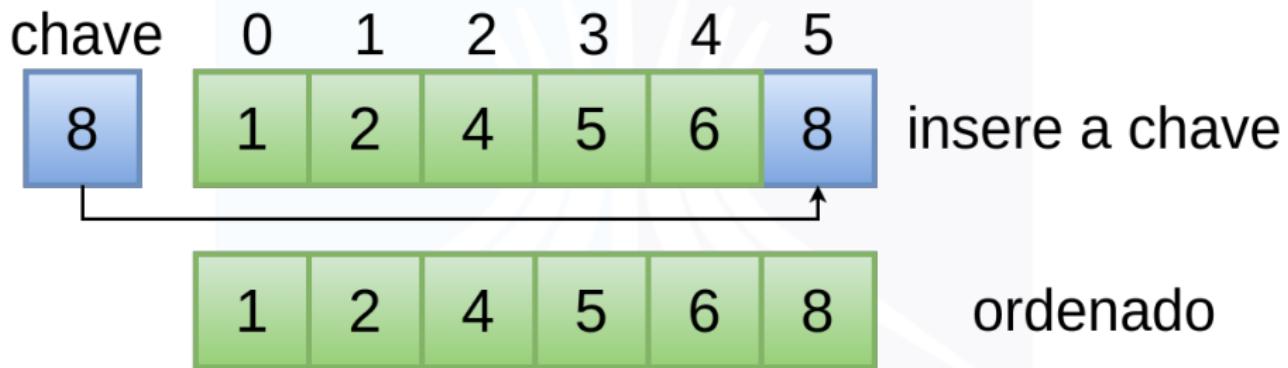
Algoritmo de inserção



Algoritmo de inserção



Algoritmo de inserção



Algoritmo de ordenação por troca

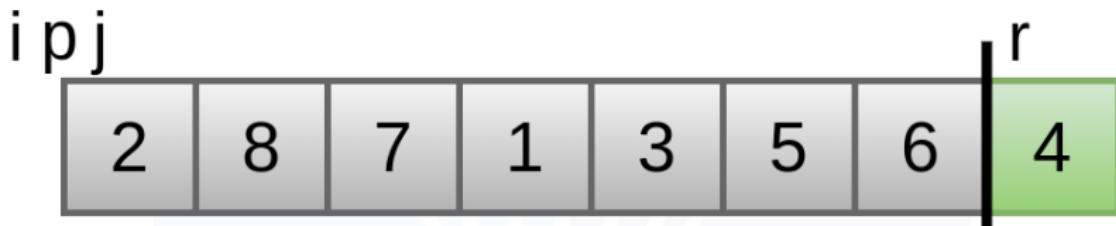
```
13 void insertionSort(int *v, int n){  
14     int i, j, chave;  
15     for (i = 1; i < n; i++) {  
16         chave = v[i];  
17         j = i - 1;  
18         while (j >= 0 && v[j] > chave) {  
19             v[j + 1] = v[j];  
20             j = j - 1;  
21         }  
22         v[j + 1] = chave;  
23     }  
24 }
```

Quick Sort

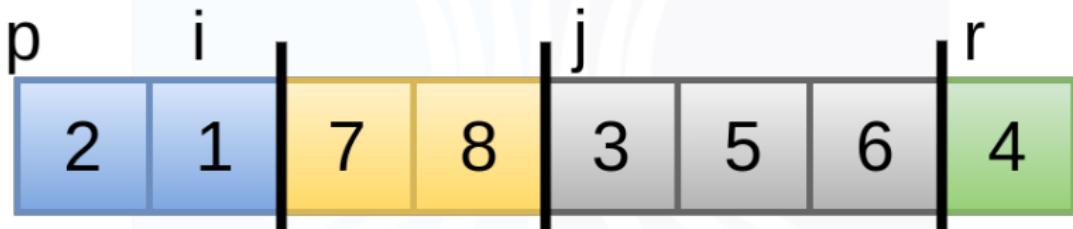
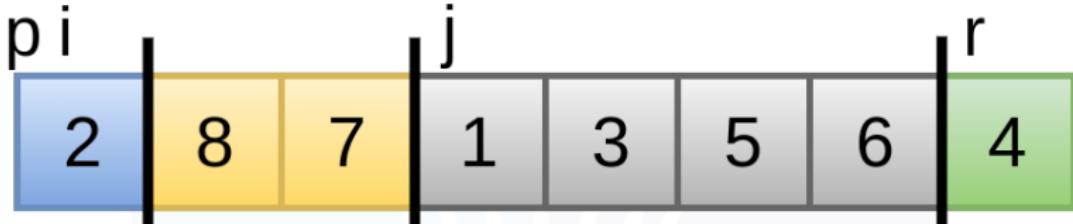
- Usa a técnica de divisão e conquista
- Seleciona um pivô, dividir o vetor em 2 partições
- Uma contendo os elementos com valor maior ou igual ao pivô
- E a outra contendo os elementos com valor menor que o pivô



Quick Sort



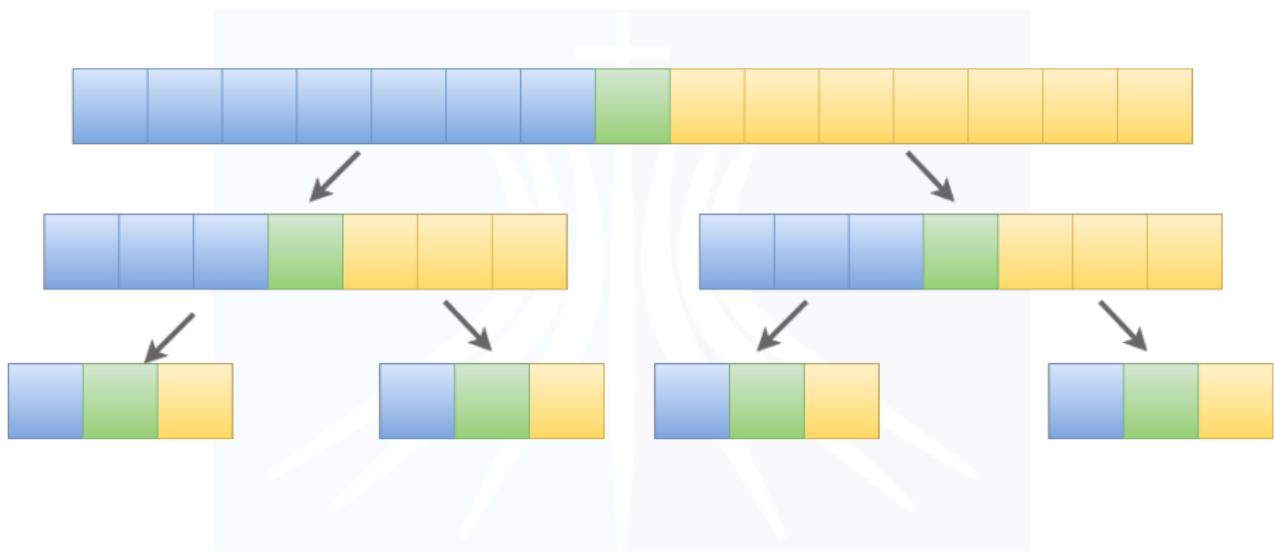
Quicksort



Quicksort



Quicksort



Quicksort

```
25 int partition(int *v, int p, int r){  
26     int x = v[r]; // pivot  
27     int aux, j, i = p;  
28     for (j = p; j < r; ++j){  
29         if(v[j] <= x){  
30             aux = v[i];  
31             v[i] = v[j];  
32             v[j] = aux;  
33             ++i;  
34         }  
35     }  
36     aux = v[i];  
37     v[i] = v[r];  
38     v[r] = aux;  
39     return i;  
40 }
```

Quicksort

```
41 void quickSort(int *v, int p, int r){  
42     int q;  
43     if(p < r){  
44         q = partition(v, p, r);  
45         quickSort(v, p, q-1);  
46         quickSort(v, q+1, r);  
47     }  
48 }
```

Counting Sort

- Pressupõe que cada um dos n elementos de entrada é um inteiro no intervalo de 0 a k , para algum inteiro k
- Determina para cada elemento de entrada x , o número de elementos menores que x
- Com esta informação podemos inserir o elemento diretamente em sua posição no arranjo de saída
- Utiliza 3 vetores:
 - $A[0..n-1]$: vetor de entrada desordenado
 - $B[0..n-1]$: vetor ordenado de saída
 - $C[0..k]$: vetor auxiliar para contagem

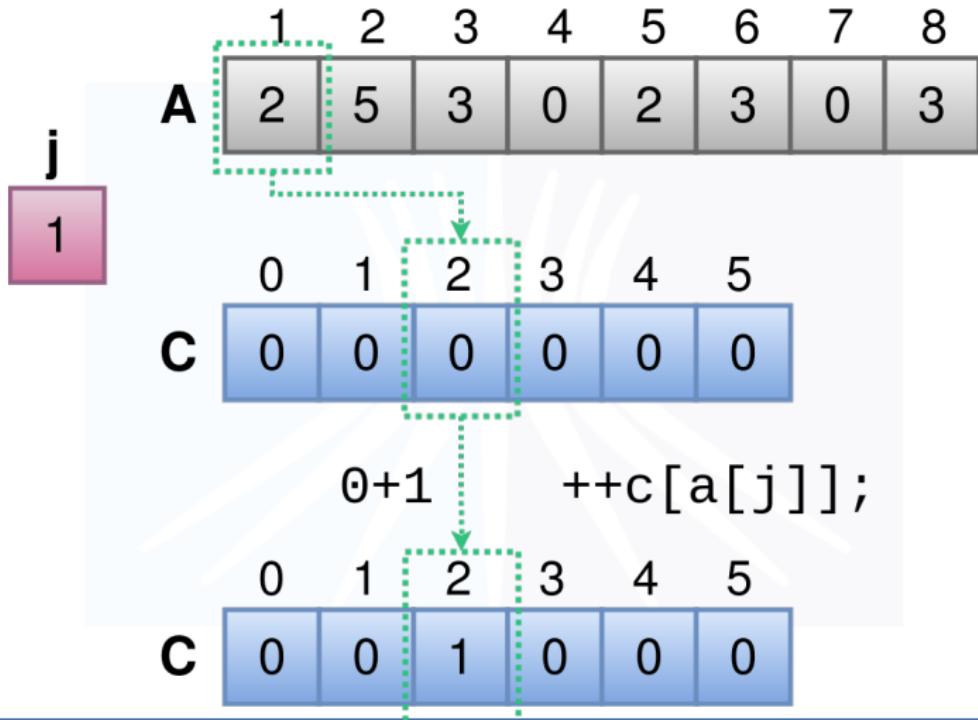
Counting sort

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

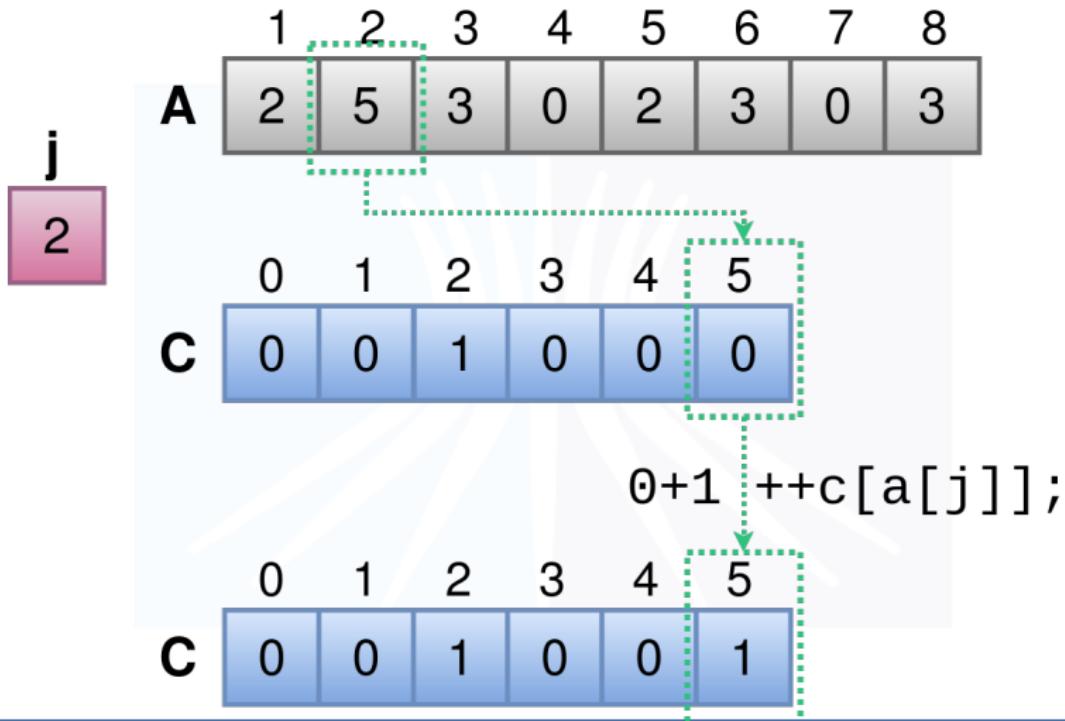
Atualizando as posições

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

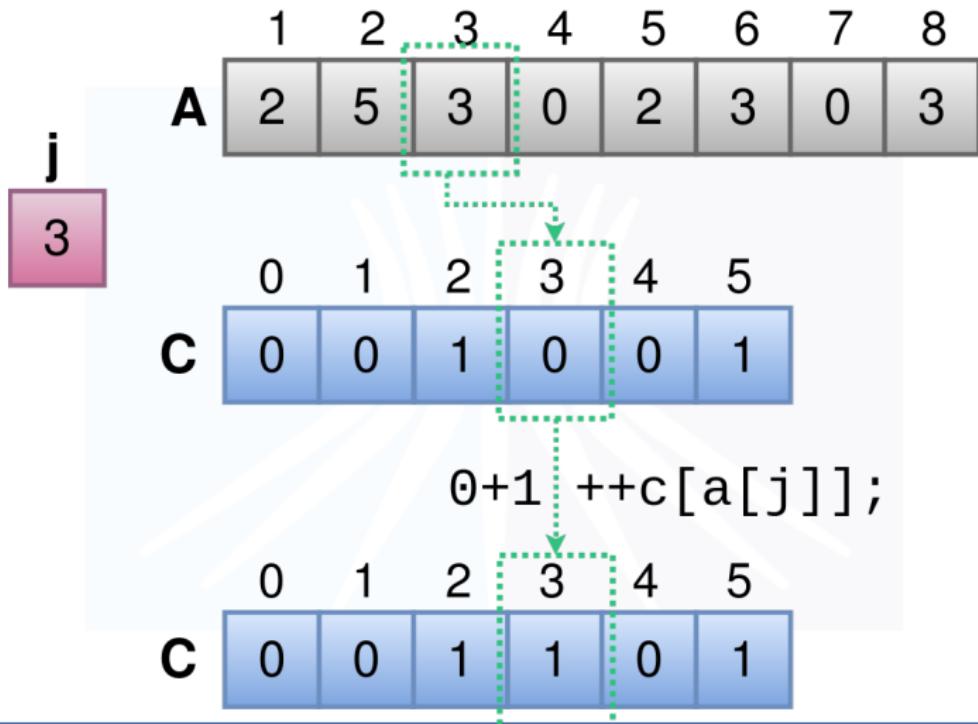
Counting sort



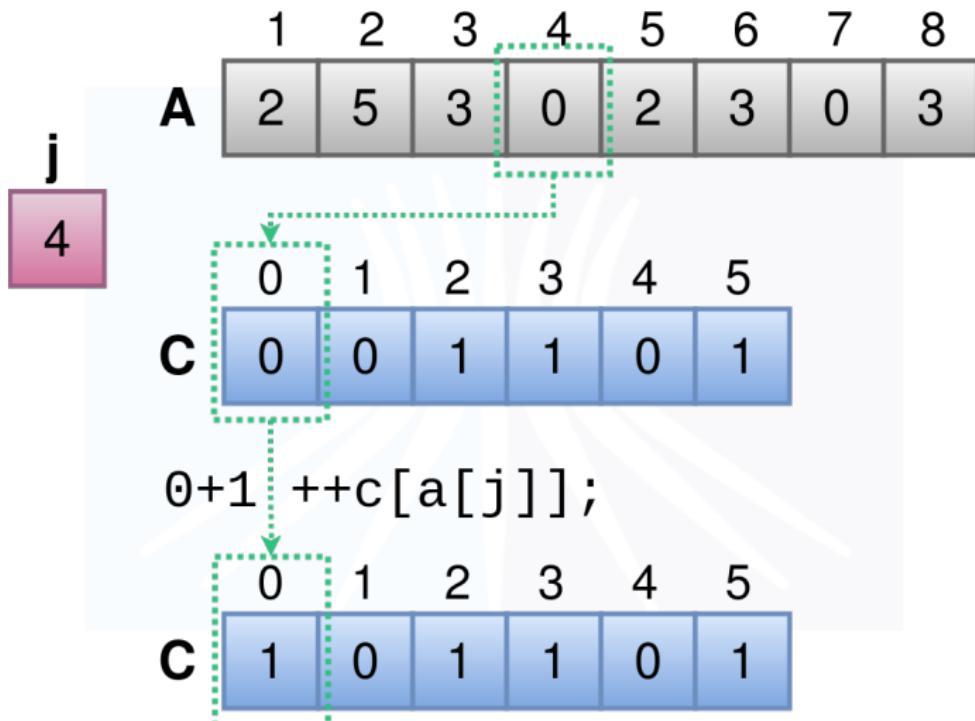
Counting sort



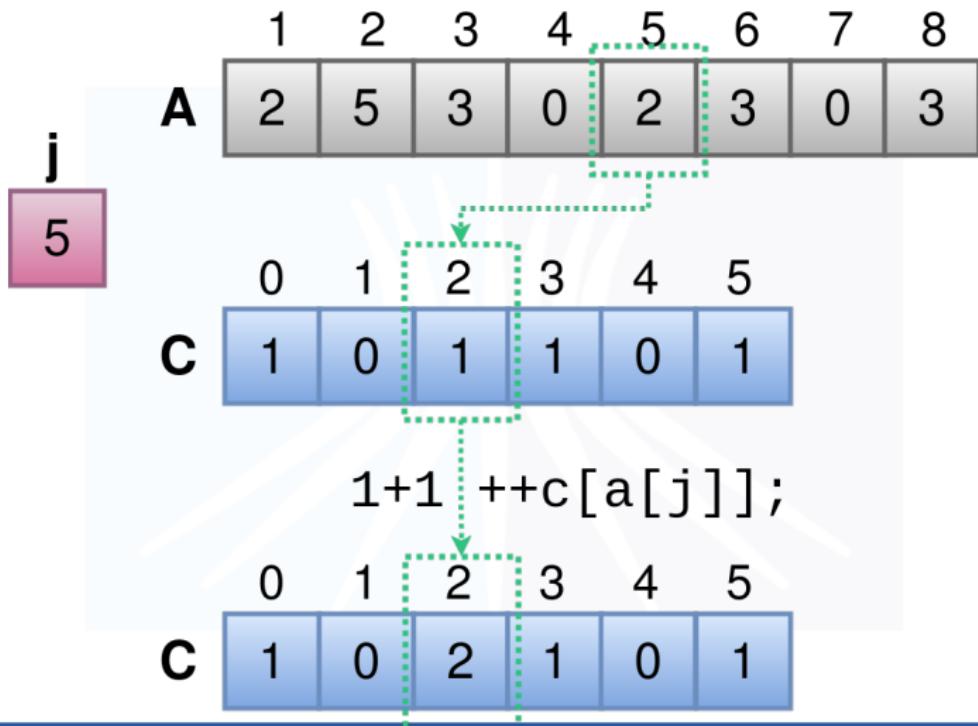
Counting sort



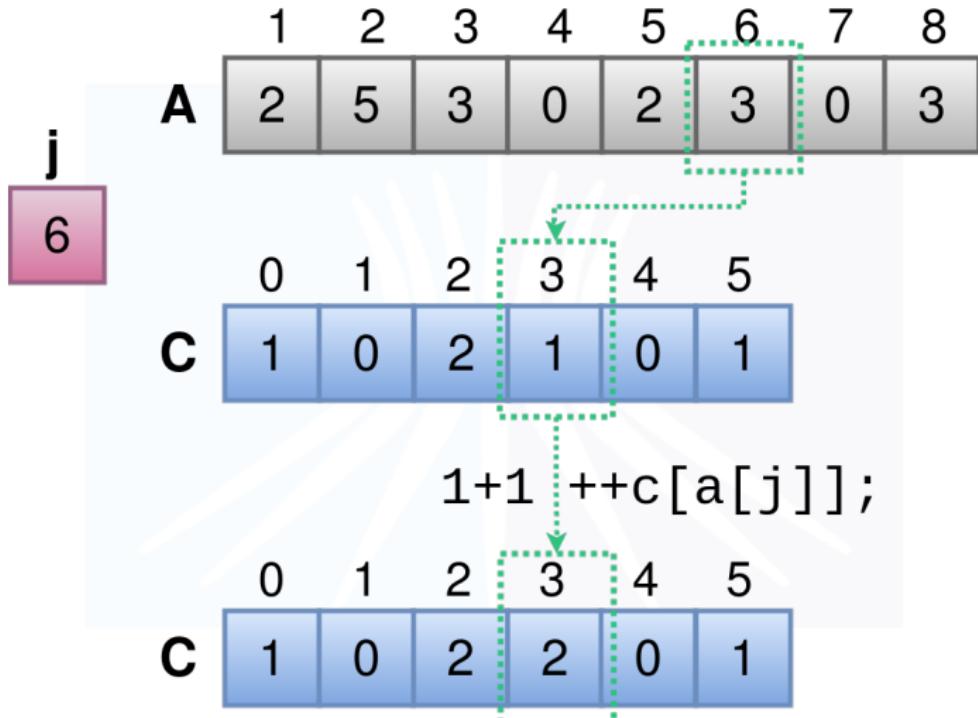
Counting sort



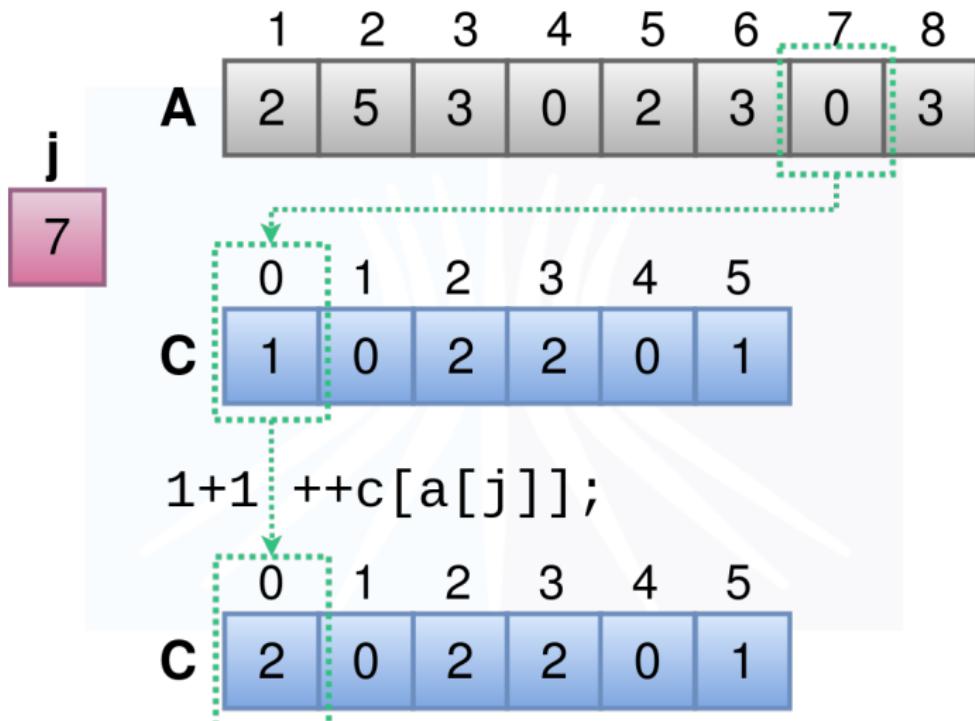
Counting sort



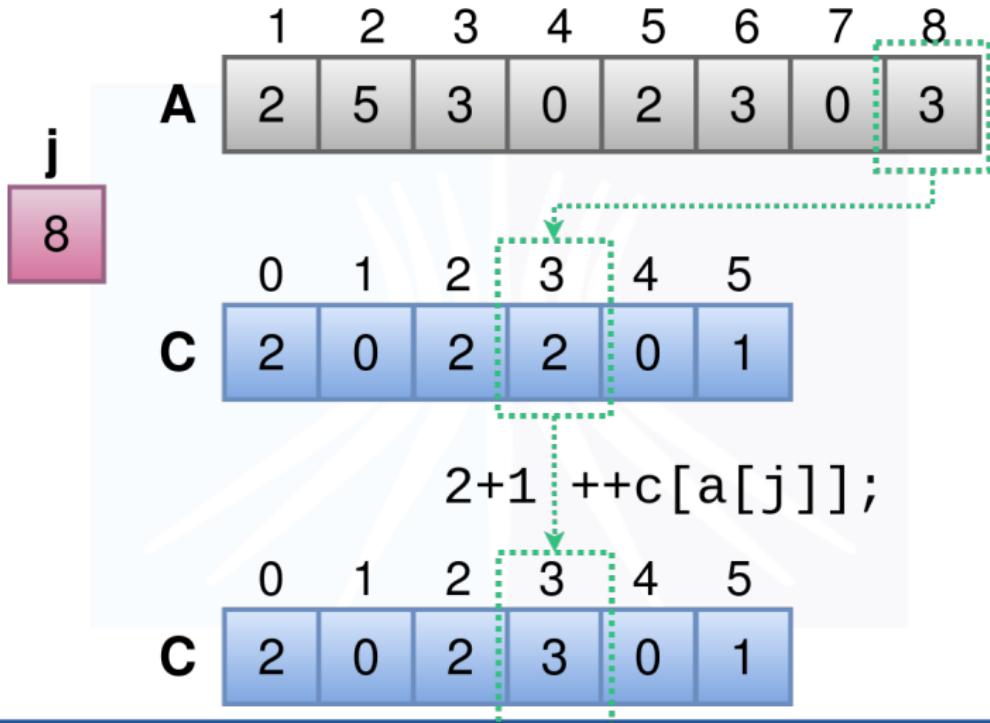
Counting sort



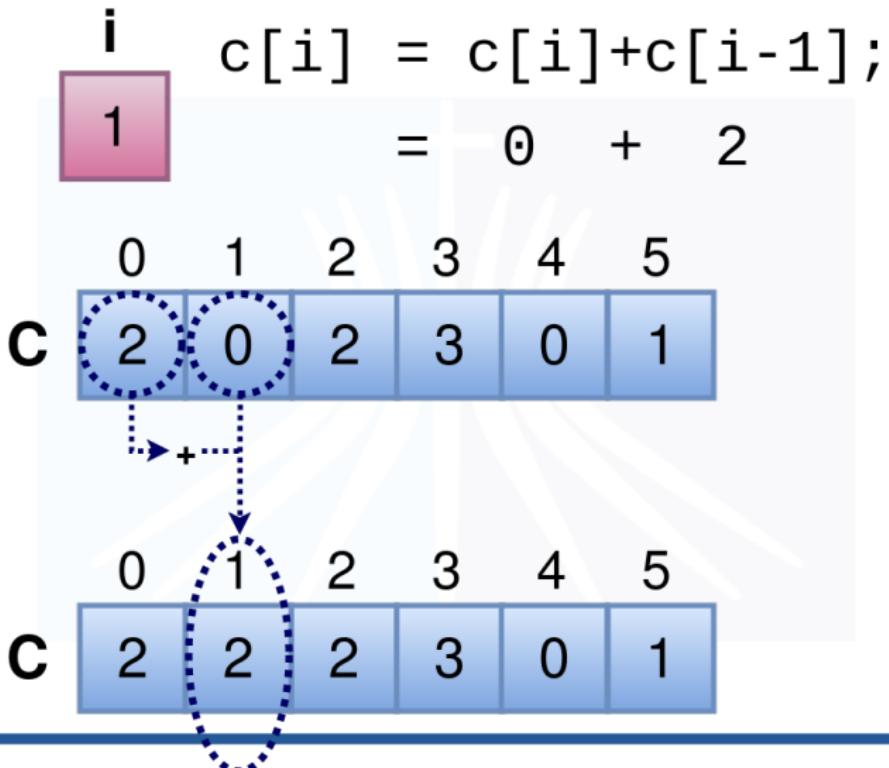
Counting sort



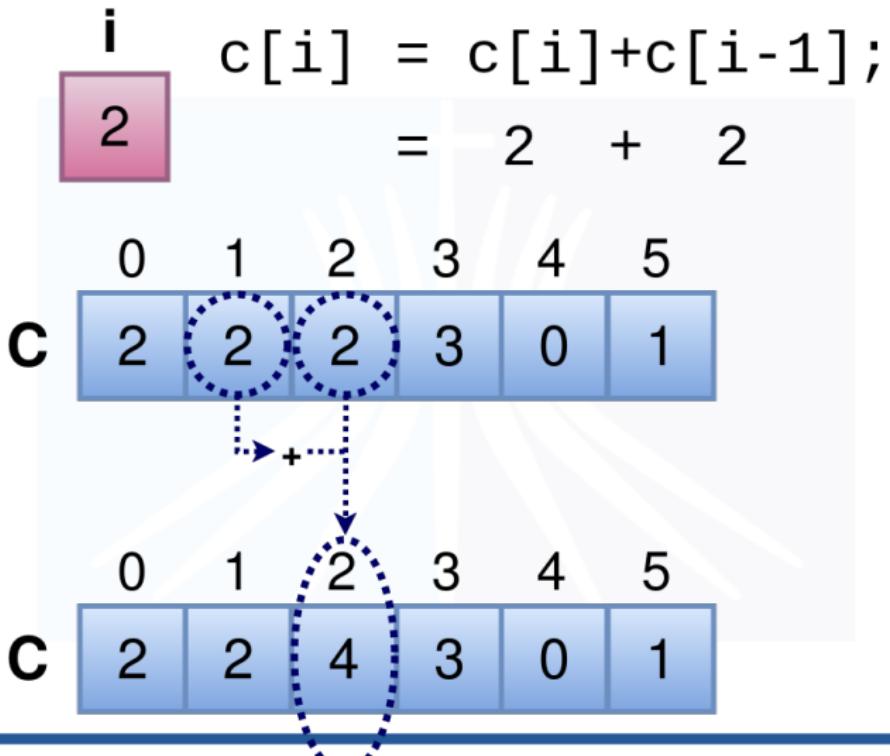
Counting sort



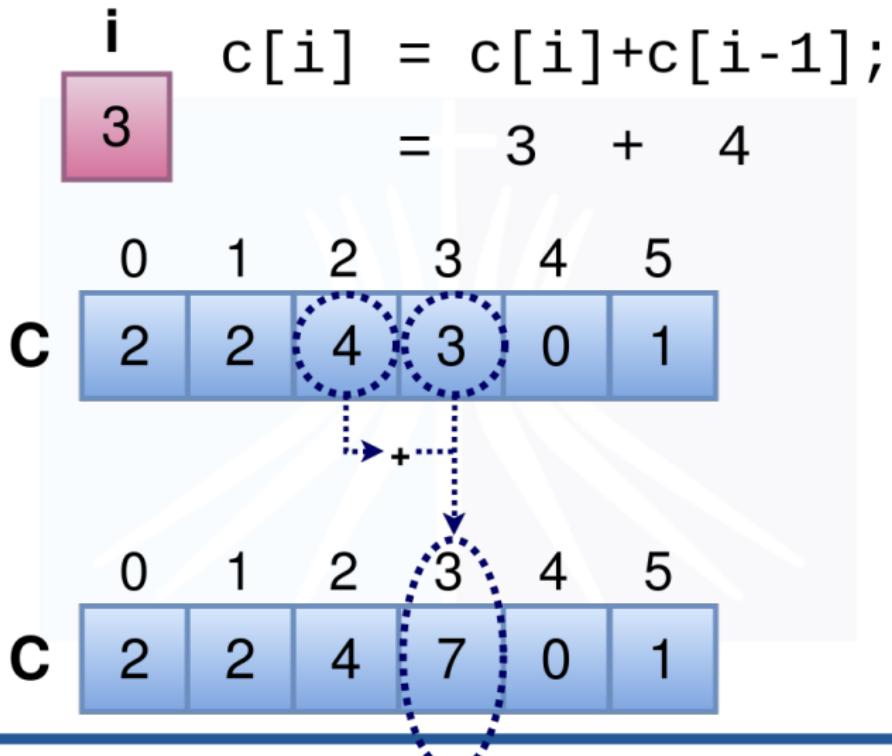
Counting sort



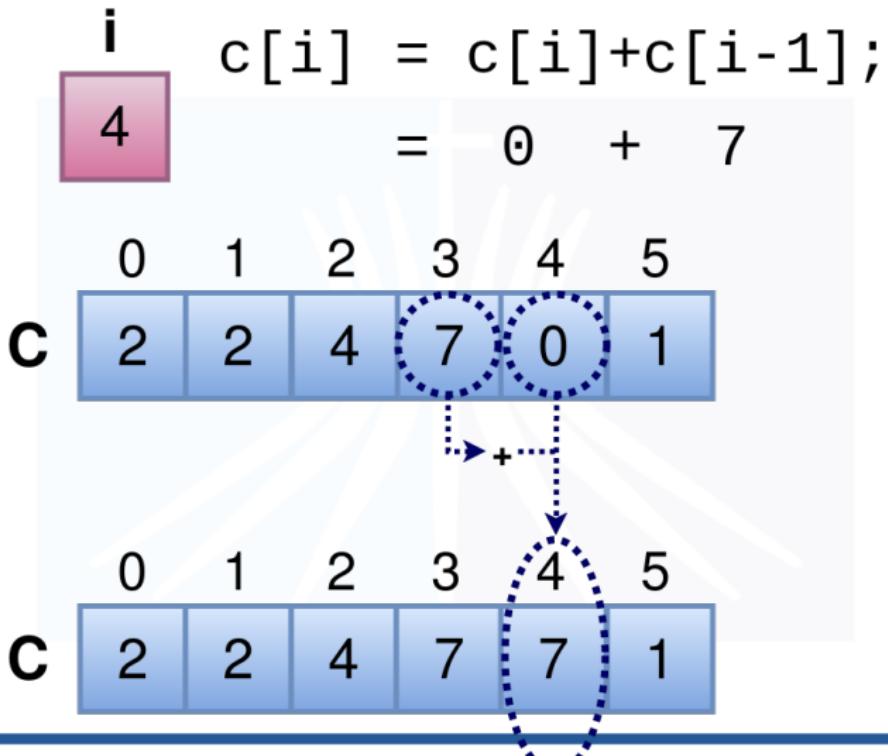
Counting sort



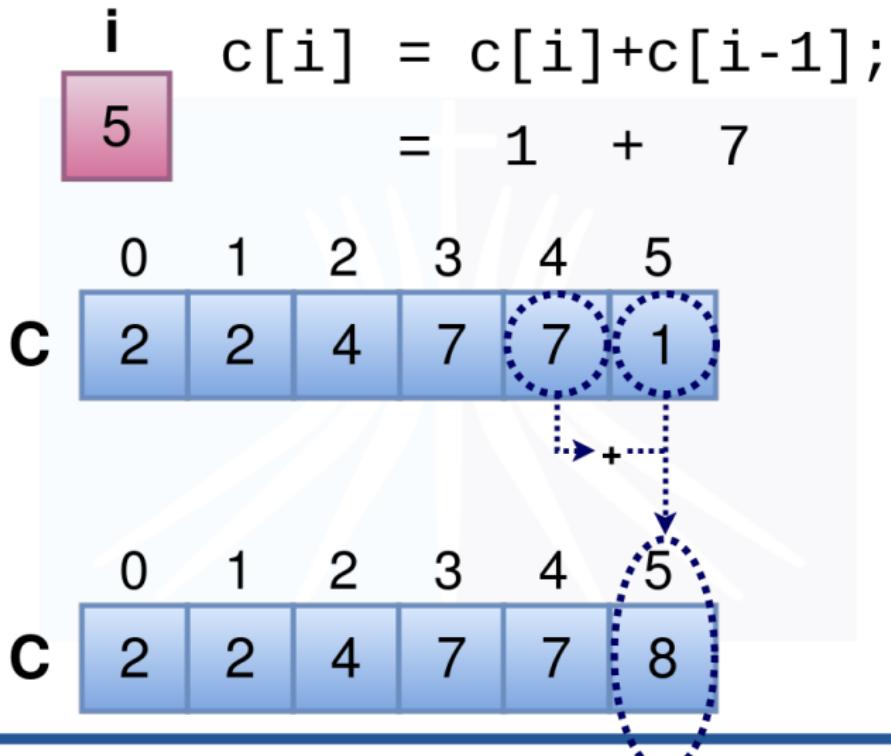
Counting sort



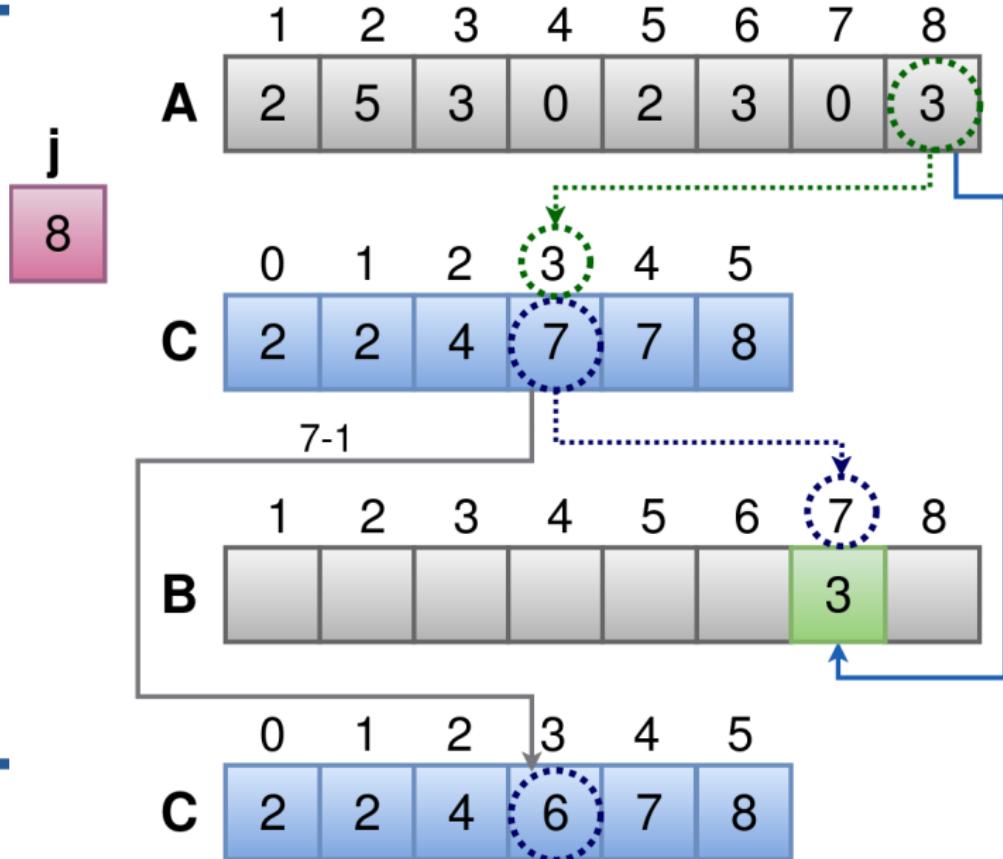
Counting sort



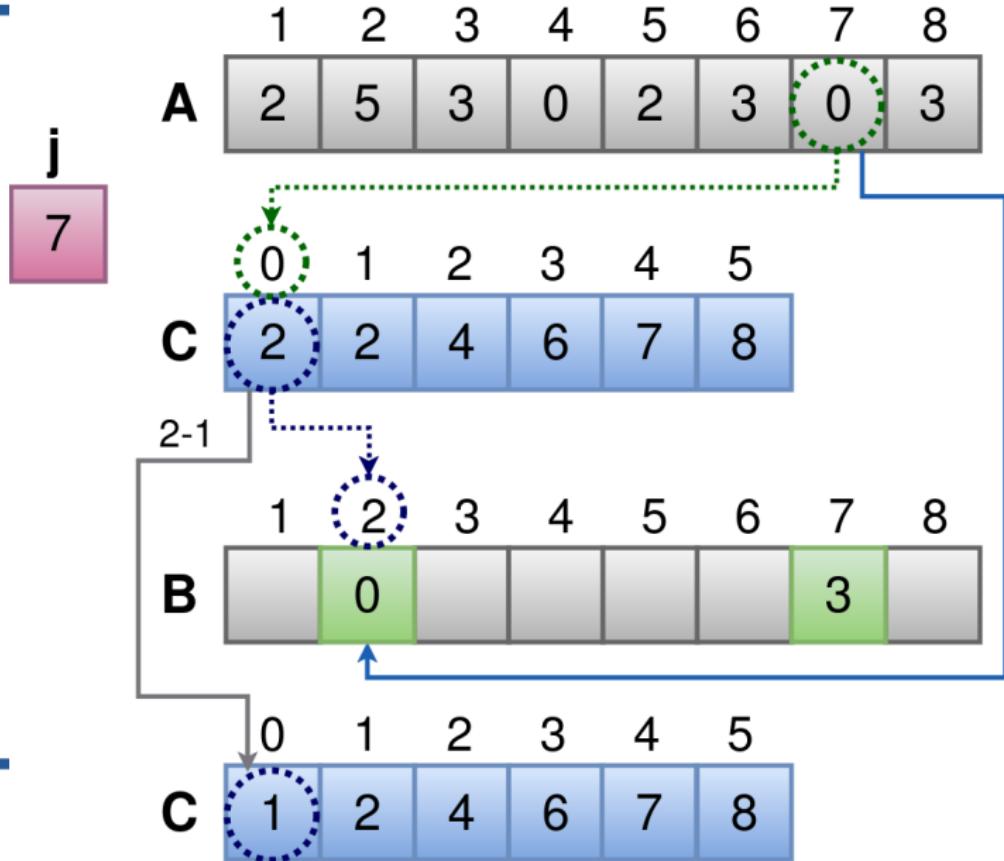
Counting sort



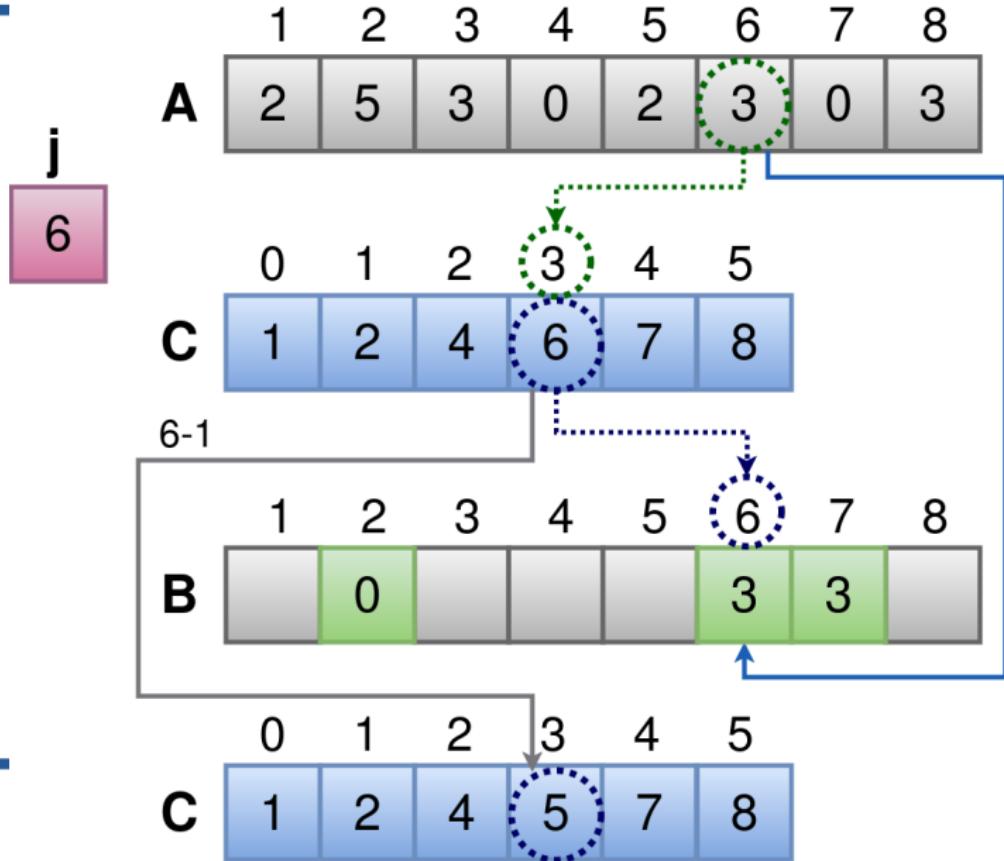
Counting sort



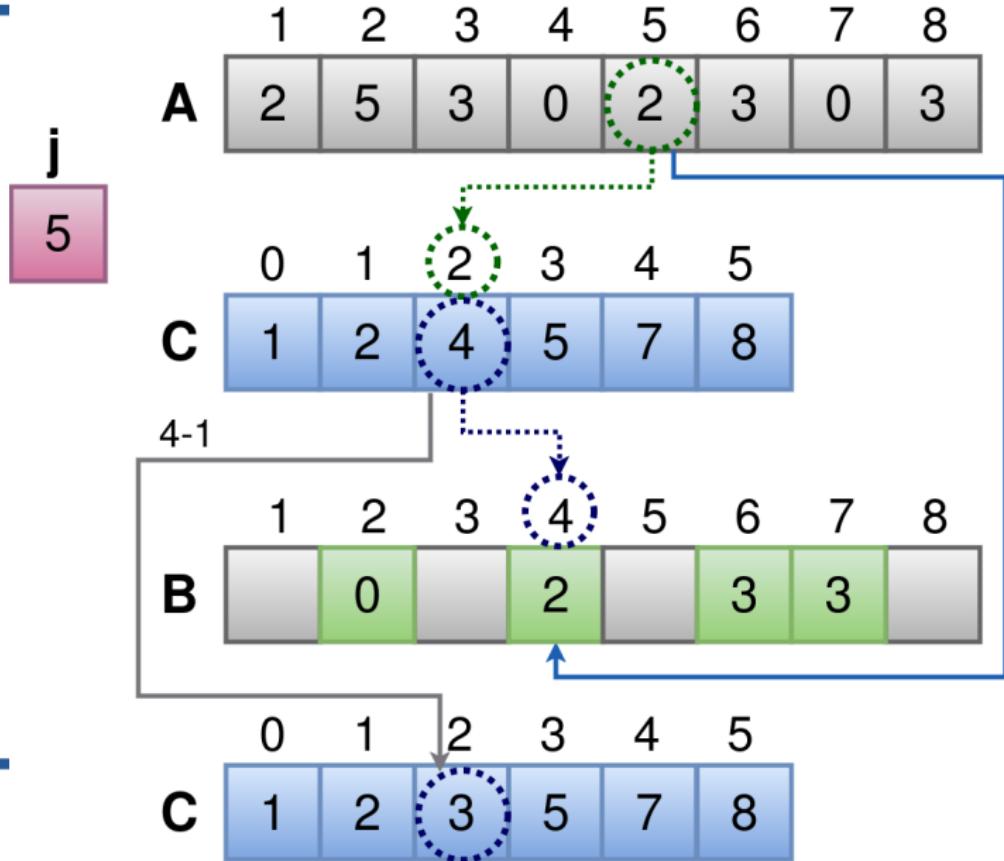
Counting sort



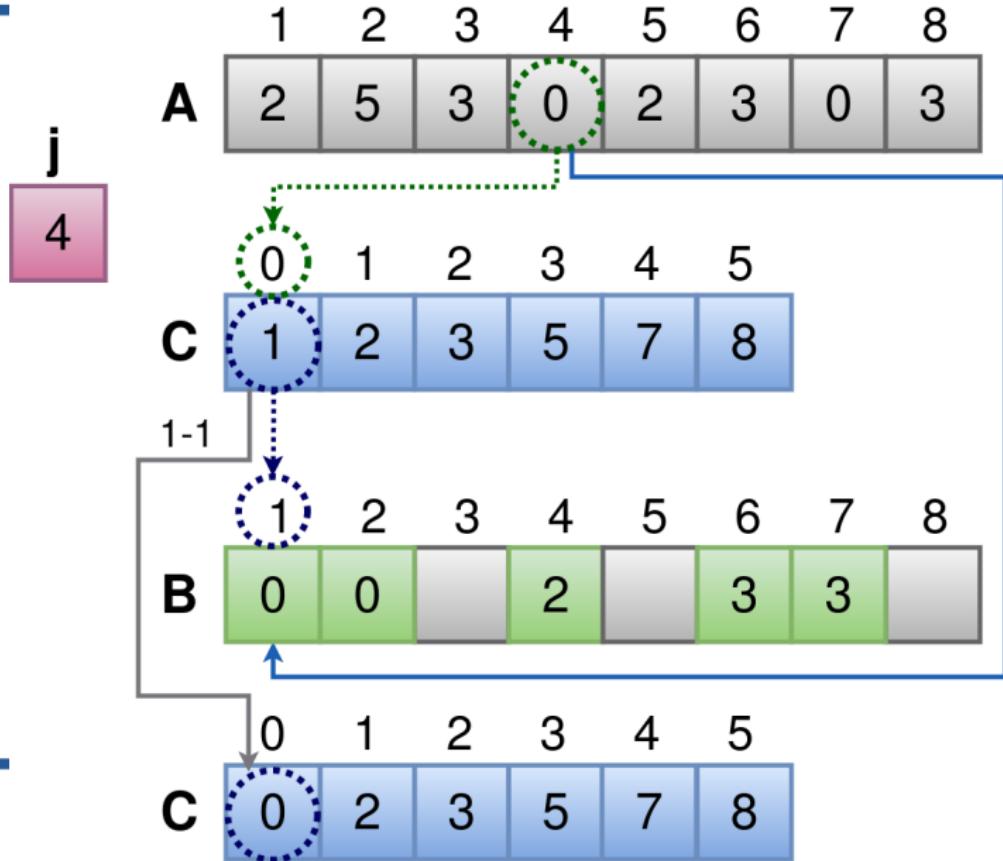
Counting sort



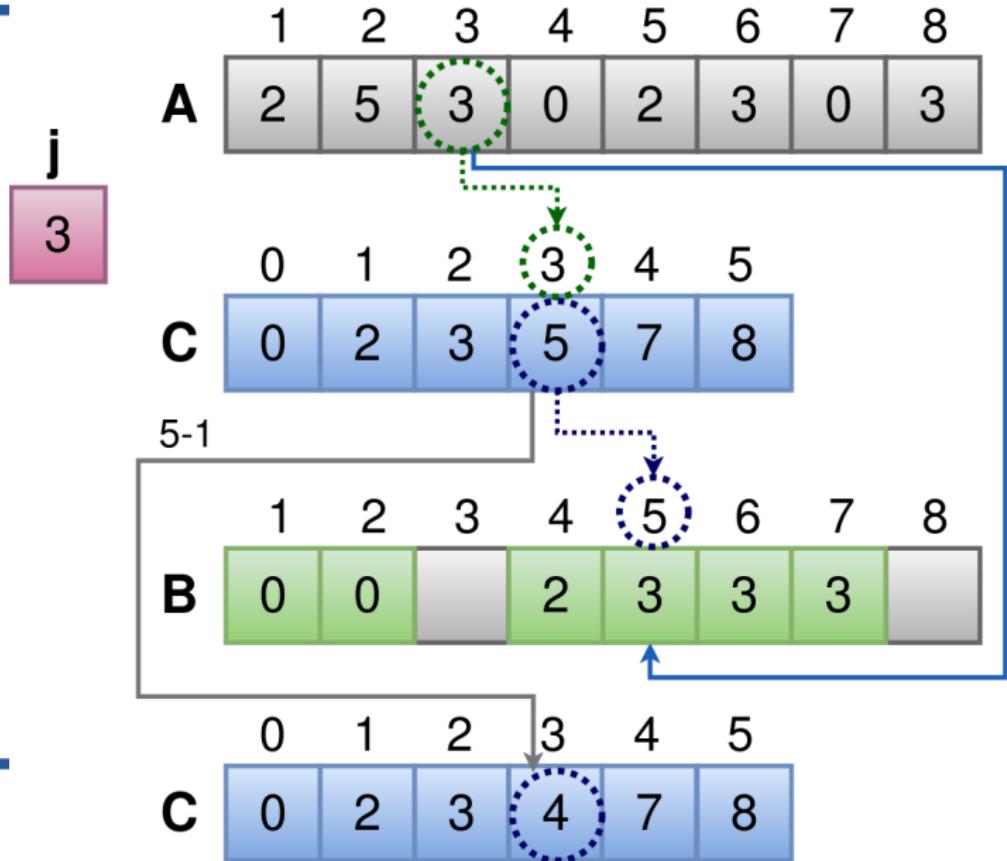
Counting sort



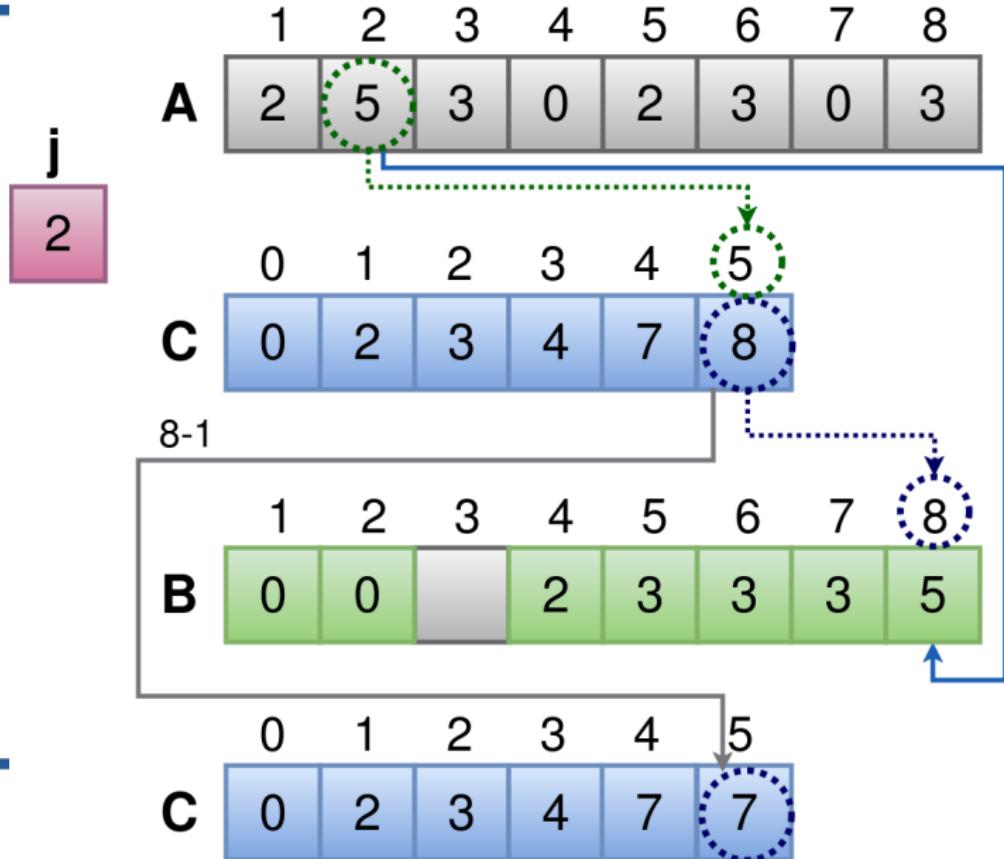
Counting sort



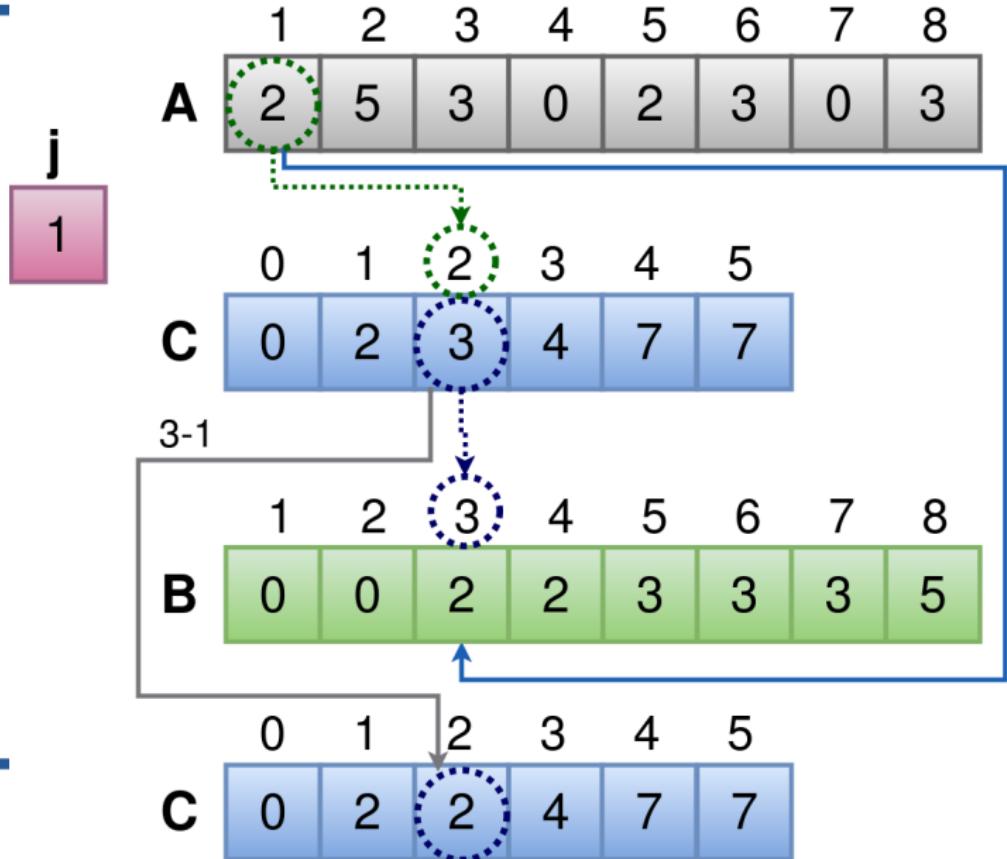
Counting sort



Counting sort



Counting sort



Counting sort

```
49 void countingSort(int *a){  
50     int i, j, c[9], b[11];  
51  
52     for(i = 0; i <= 8; ++i)  
53         c[i] = 0;  
54  
55     for(j = 0; j < 11; ++j)  
56         ++c[a[j]];  
57  
58     for(i = 1; i <= 8; ++i)  
59         c[i] = c[i] + c[i-1];  
60  
61     for(j = 10; j >= 0; --j){  
62         b[c[a[j]]-1] = a[j];  
63         c[a[j]] = c[a[j]] - 1;  
64     }  
65 }
```

Radix Sort

- 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
 - “Aluno” tem 5 caracteres (base 256)
- 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

Radix Sort

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

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

1	7	6	1
---	---	---	---

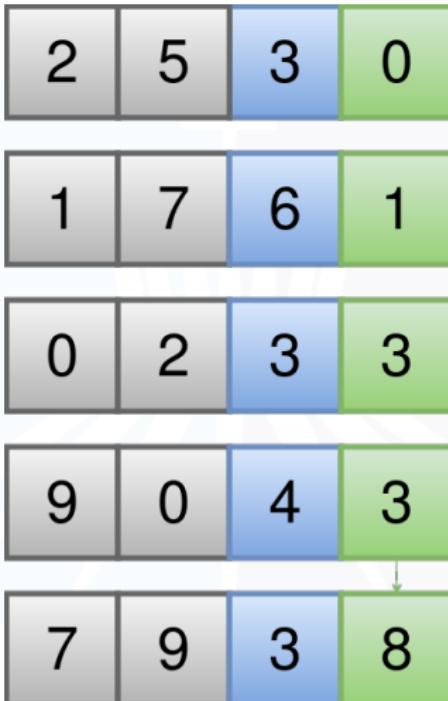
9	0	4	3
---	---	---	---

7	9	3	8
---	---	---	---

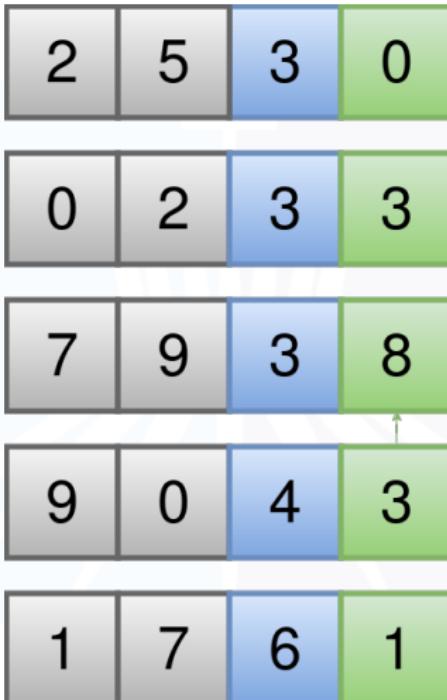
Radix Sort



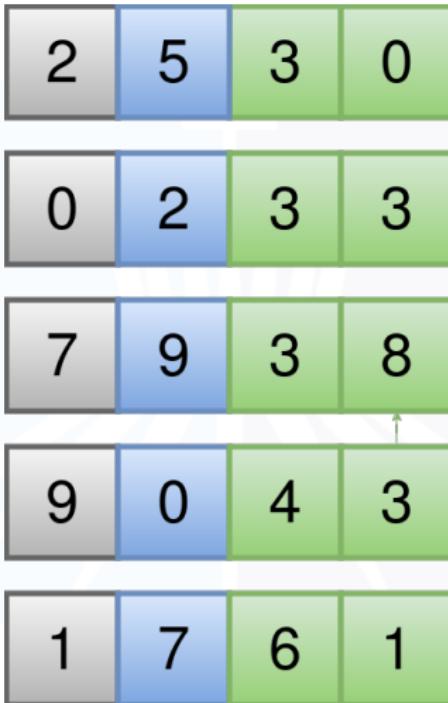
Radix Sort



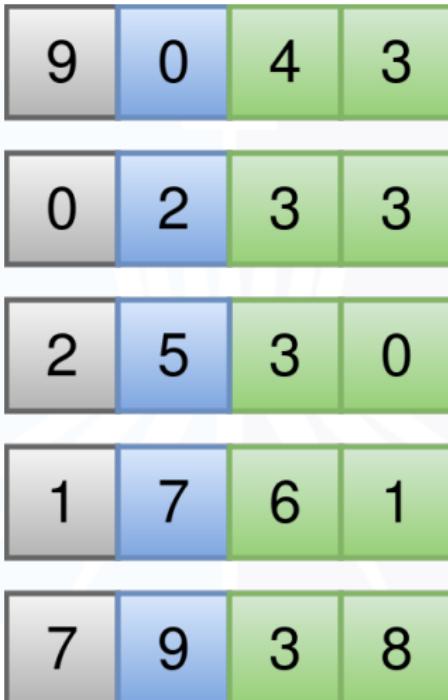
Radix Sort



Radix Sort



Radix Sort



Radix Sort



Radix Sort



Radix Sort

```
66 void countingSort(int *a, int n, int exp){  
67     int i, b[n], c[10] = {0};  
68     for (i = 0; i < n; i++){  
69         c[(a[i] / exp) % 10]++;  
70     }  
71     for (i = 1; i < 10; i++){  
72         c[i] += c[i - 1];  
73     }  
74     for (i = n - 1; i >= 0; i--) {  
75         b[c[(a[i] / exp) % 10] - 1] = a[i];  
76         c[(a[i] / exp) % 10]--;  
77     }  
78     for (i = 0; i < n; i++){  
79         a[i] = b[i];  
80     }  
81 }
```

```
82 int valorMaximo(int *v, int n){  
83     int i, max = v[0];  
84     for (i = 1; i < n; i++)  
85         if (v[i] > max)  
86             max = v[i];  
87     return max;  
88 }  
89 void radixsort(int *v, int n){  
90     int exp, max = valorMaximo(v, n);  
91     for (exp = 1; max / exp > 0; exp *= 10){  
92         countingSort(v, n, exp);  
93     }  
94 }
```

Estruturas de Dados

Mayrton Dias
mayrton.queiroz@p.ficr.edu.br