DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Algoritmos Eficientes de Ordenação (continuação)

Sedgewick: Capítulo 6

IAED

Algoritmos Eficientes de Ordenação

- Quick Sort
- Merge Sort
- Heap Sort

Utilizar informação das chaves:

- Counting Sort
- Radix Sort

DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Counting Sort

IAED

Ordenação por Comparação

- Algoritmos de ordenação baseados em comparações são pelo menos $O(N \lg N)$
  - Para N chaves existem $N!$ ordenações possíveis das chaves
  - Algoritmo de ordenação por comparação utiliza comparações de pares de chaves para selecionar uma das $N!$ ordenações
    - Escolher uma folha em árvore com $N!$ folhas
    - Altura da árvore é não inferior lg(N!) ≈ N lg N
- **É possível obter algoritmos mais eficientes *desde que não sejam apenas baseados em comparações***

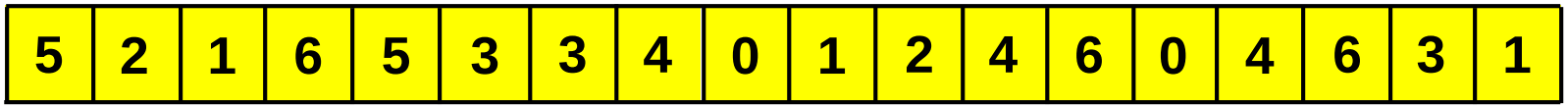
- ***Alternativa***: Utilizar informação quanto às chaves utilizadas

Counting Sort - Motivação

- Ordenar $N = r - l + 1$ elementos
- Chaves podem tomar valor inteiro entre 0 e $M - 1$

- Exemplo: $N = 18,\ M = 7$

original



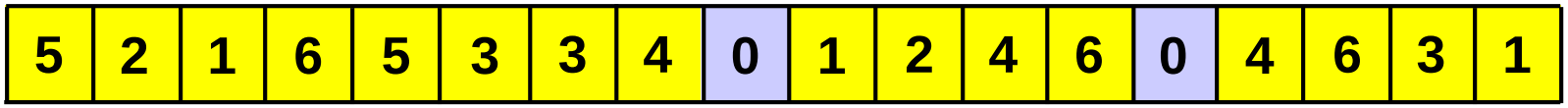
ordenado

Counting Sort - Motivação

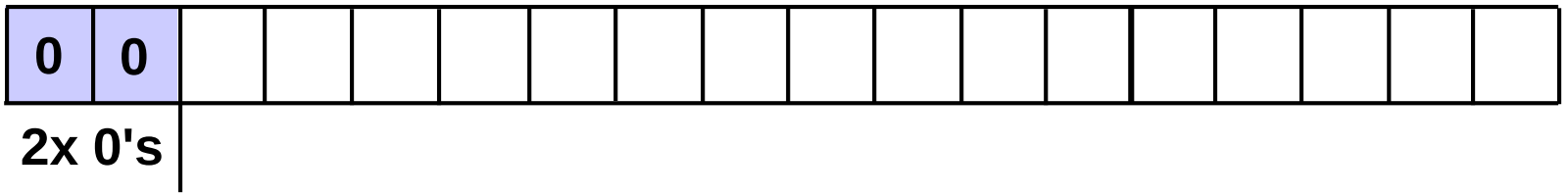
- Ordenar $N = r - l + 1$ elementos
- Chaves podem tomar valor inteiro entre 0 e $M - 1$

- Exemplo: $N = 18,\ M = 7$

original



ordenado

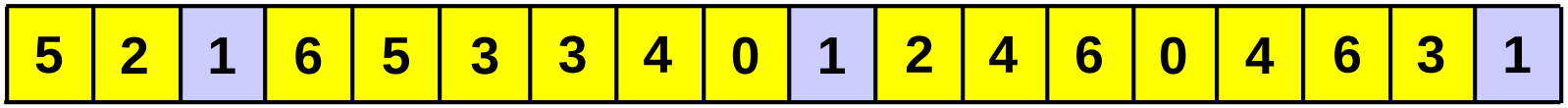


Counting Sort - Motivação

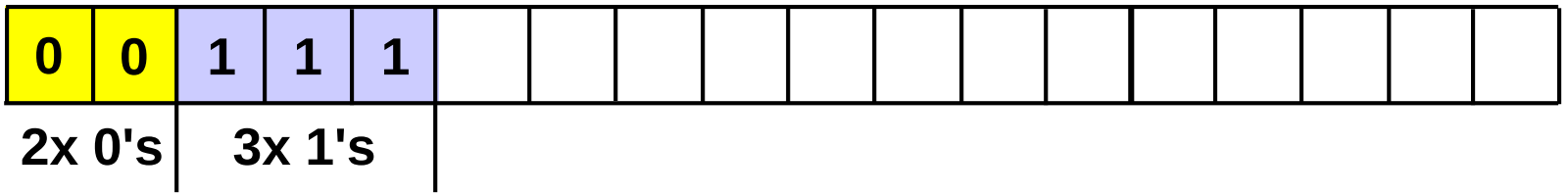
- Ordenar $N = r - l + 1$ elementos
- Chaves podem tomar valor inteiro entre 0 e $M - 1$

- Exemplo: $N = 18,\ M = 7$

original



ordenado

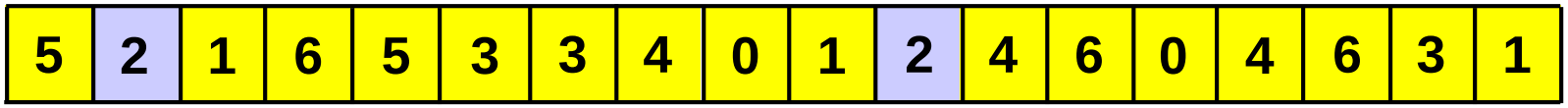


Counting Sort - Motivação

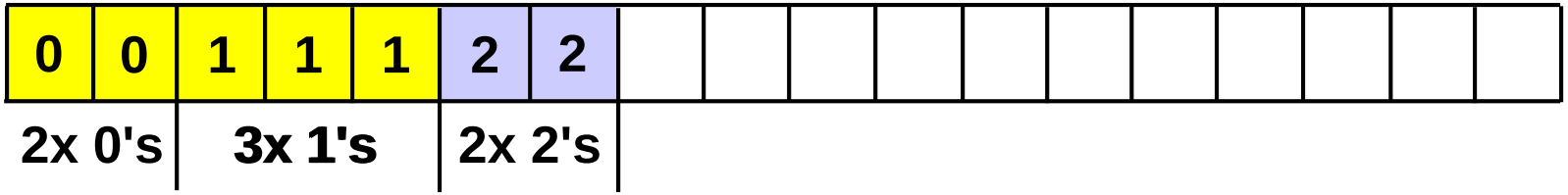
- Ordenar $N = r - l + 1$ elementos
- Chaves podem tomar valor inteiro entre 0 e $M - 1$

- Exemplo: $N = 18,\ M = 7$

original



ordenado

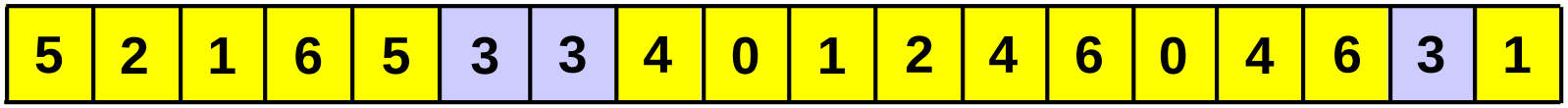


Counting Sort - Motivação

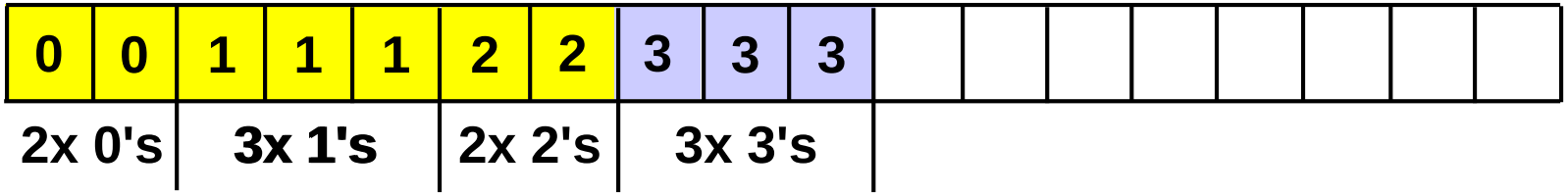
- Ordenar $N = r - l + 1$ elementos
- Chaves podem tomar valor inteiro entre 0 e $M - 1$

- Exemplo: $N = 18,\ M = 7$

original



ordenado

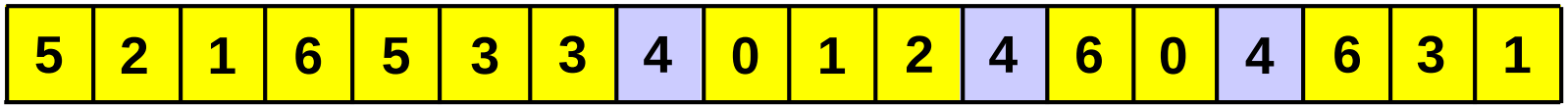


Counting Sort - Motivação

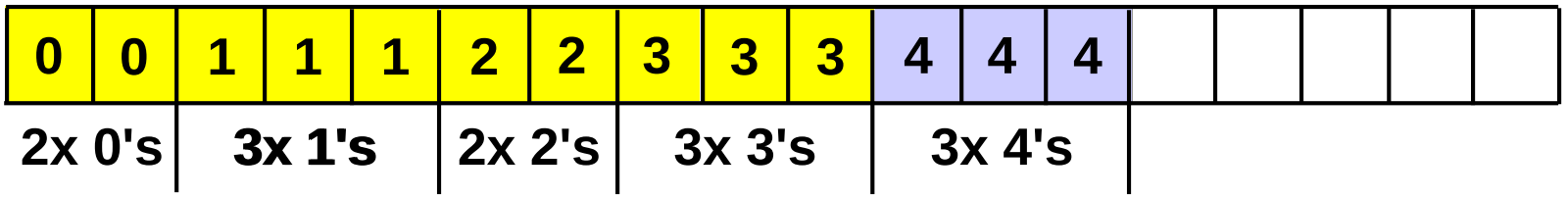
- Ordenar $N = r - l + 1$ elementos
- Chaves podem tomar valor inteiro entre 0 e $M - 1$

- Exemplo: $N = 18,\ M = 7$

original



ordenado



Counting Sort - Motivação

- Ordenar $N = r - l + 1$ elementos
- Chaves podem tomar valor inteiro entre 0 e $M - 1$

- Exemplo: $N = 18,\ M = 7$

original

5	2	1	6	5	3	3	4	0	1	2	4	6	0	4	6	3	1

ordenado

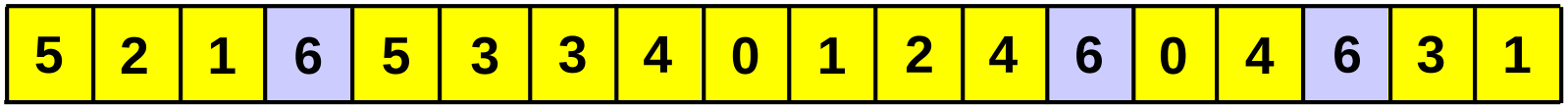
0	0	1	1	1	2	2	3	3	3	4	4	4	5	5			
2x 0's		3x 1's			2x 2's		3x 3's			3x 4's			2x 5's				

Counting Sort - Motivação

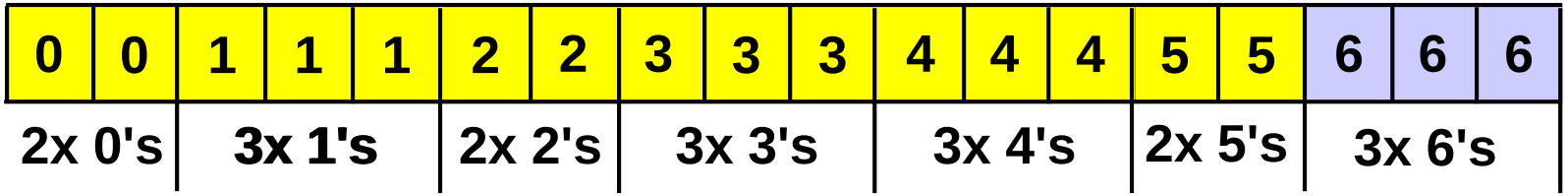
- Ordenar $N = r - l + 1$ elementos
- Chaves podem tomar valor inteiro entre 0 e $M - 1$

- Exemplo: $N = 18,\ M = 7$

original



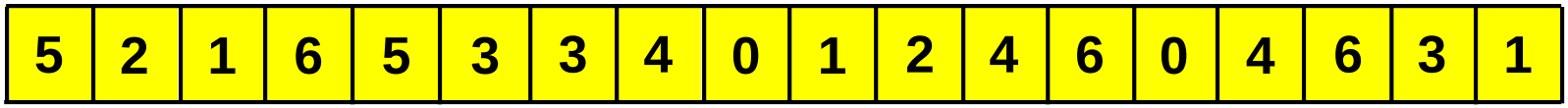
ordenado



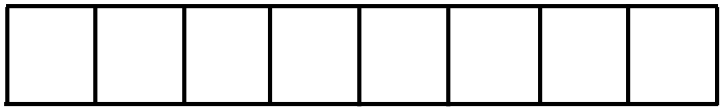
Counting Sort

$N = 18 \quad M = 7$

`int a[]`



`int cnt[M+1]`

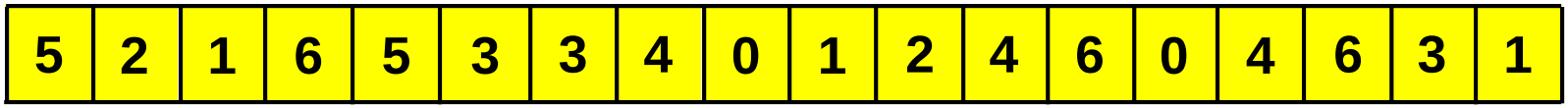


`int b[maxN]`

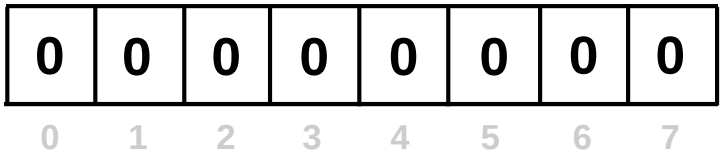
Counting Sort: passo 1

$N = 18 \quad M = 7$

`int a[]`



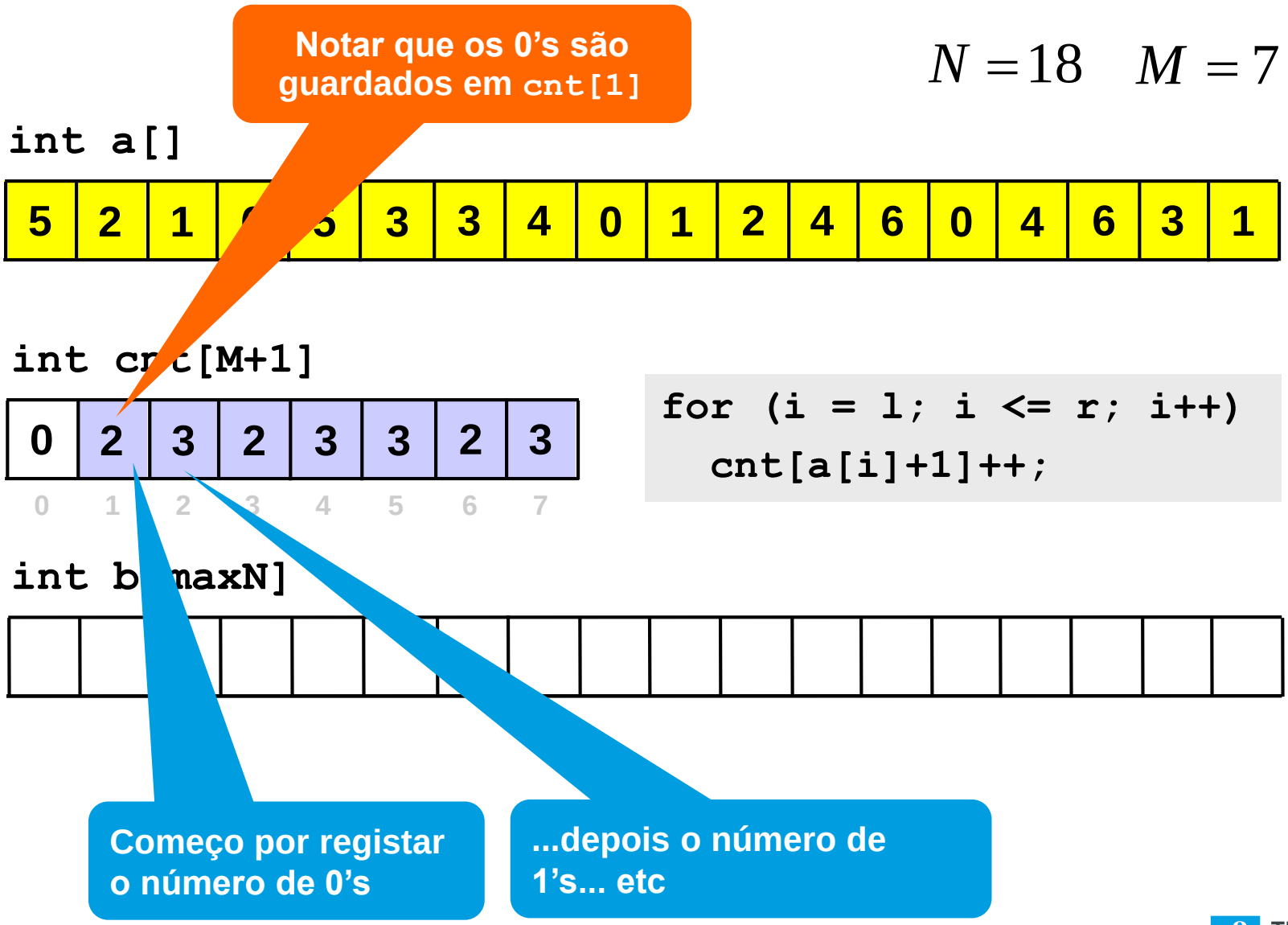
`int cnt[M+1]`



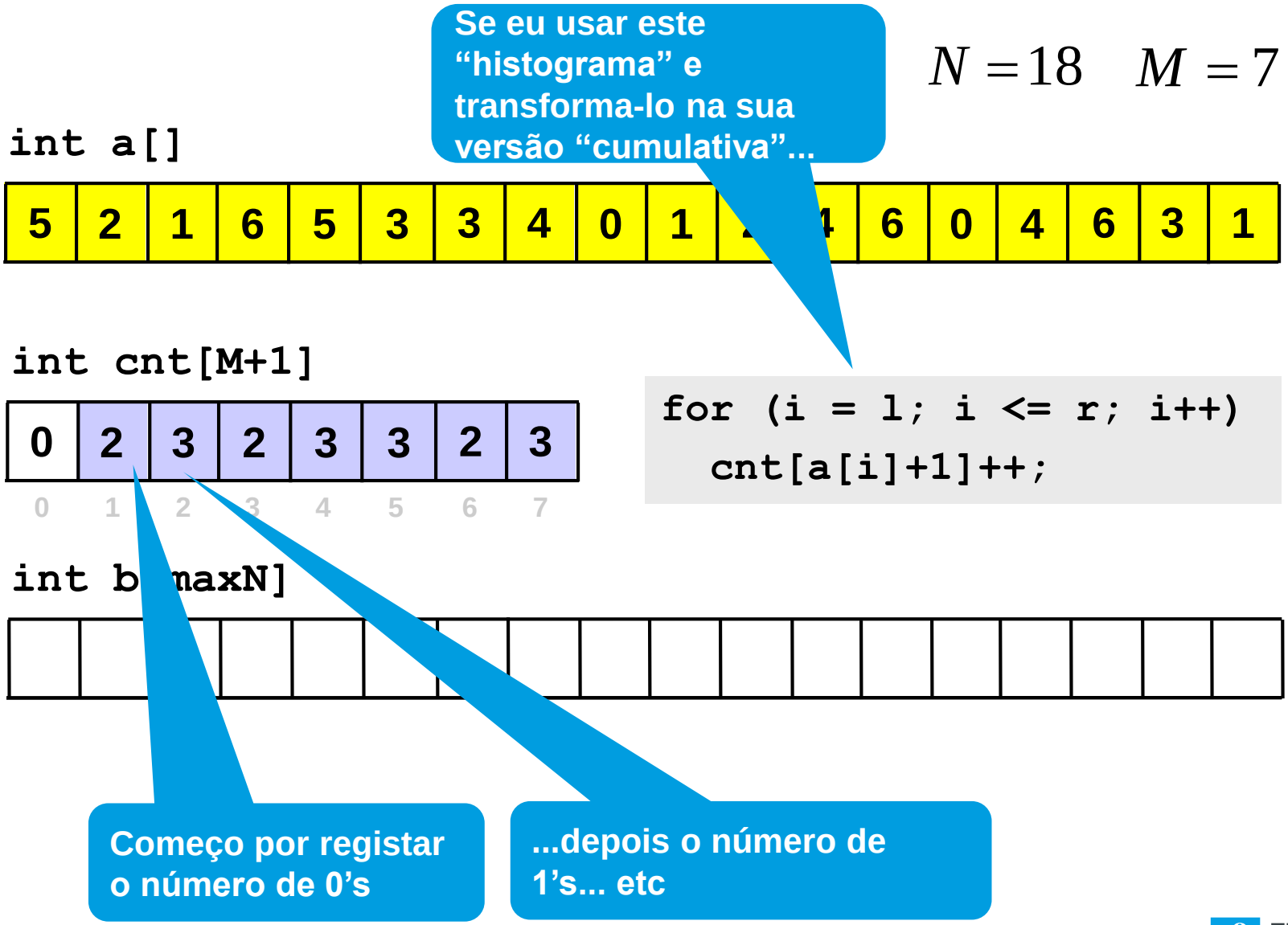

```
for (j = 0; j <= M; j++)
  cnt[j] = 0;
```

`int b[maxN]`

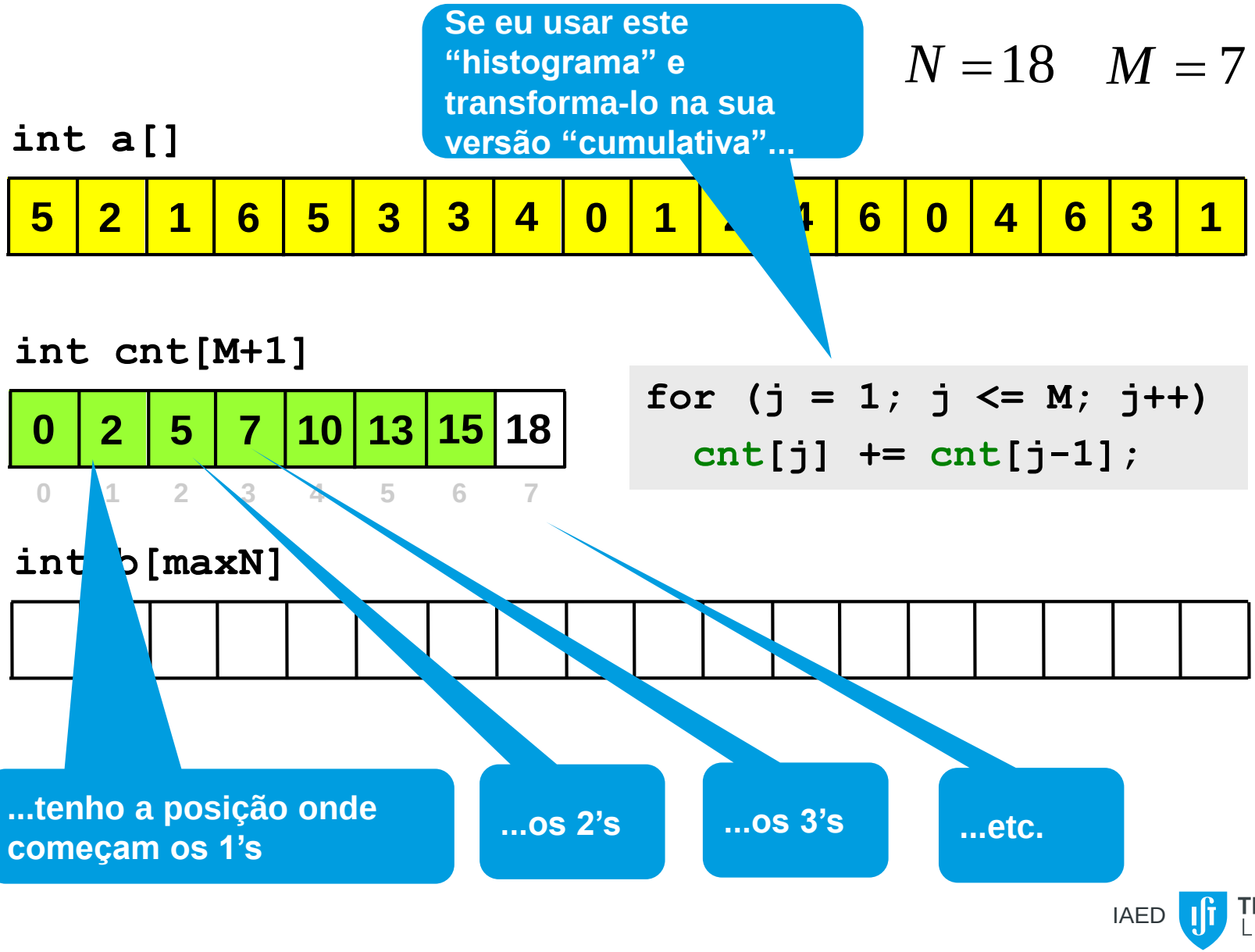
Counting Sort: passo 2



Counting Sort: passo 2

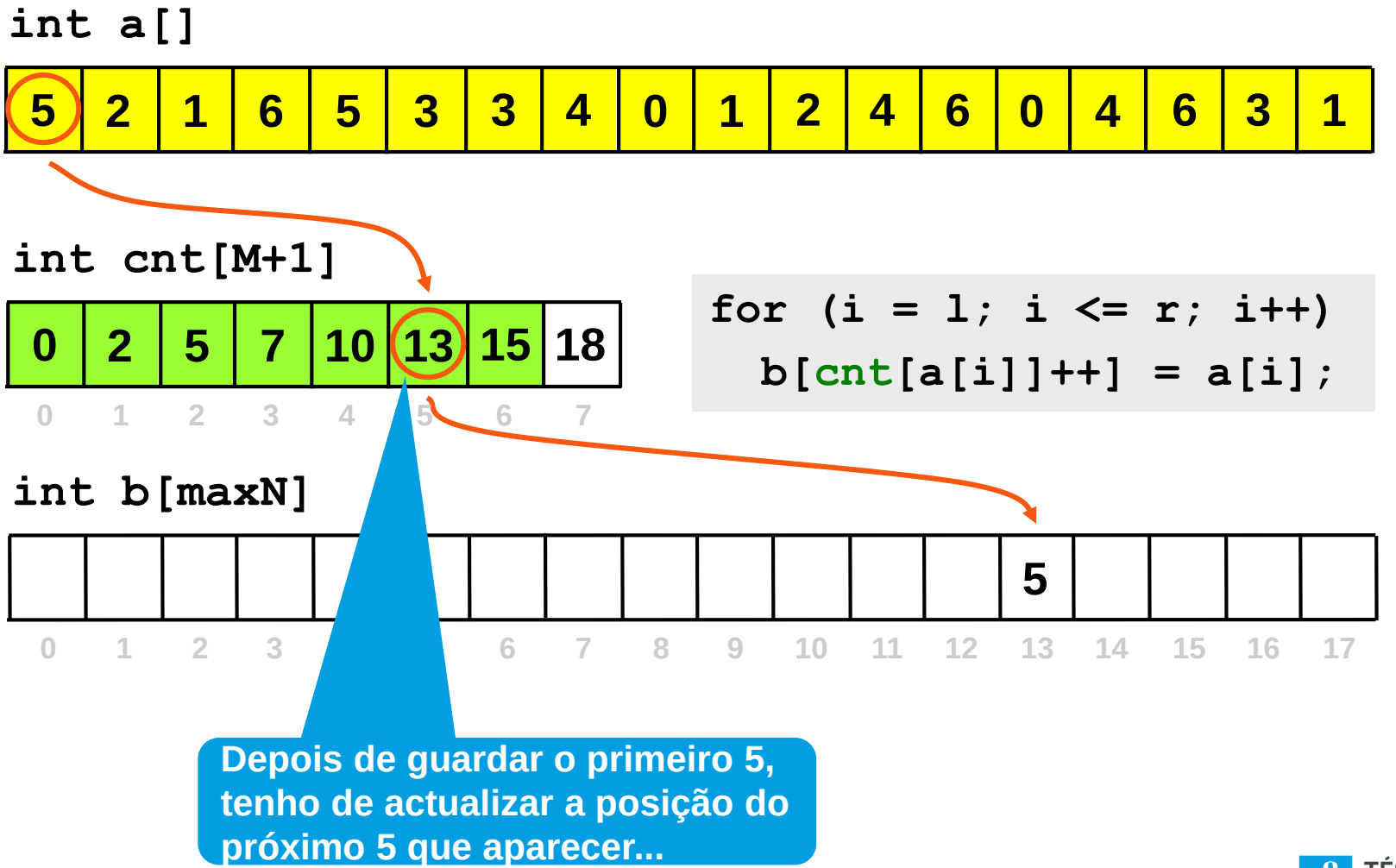


Counting Sort: passo 3



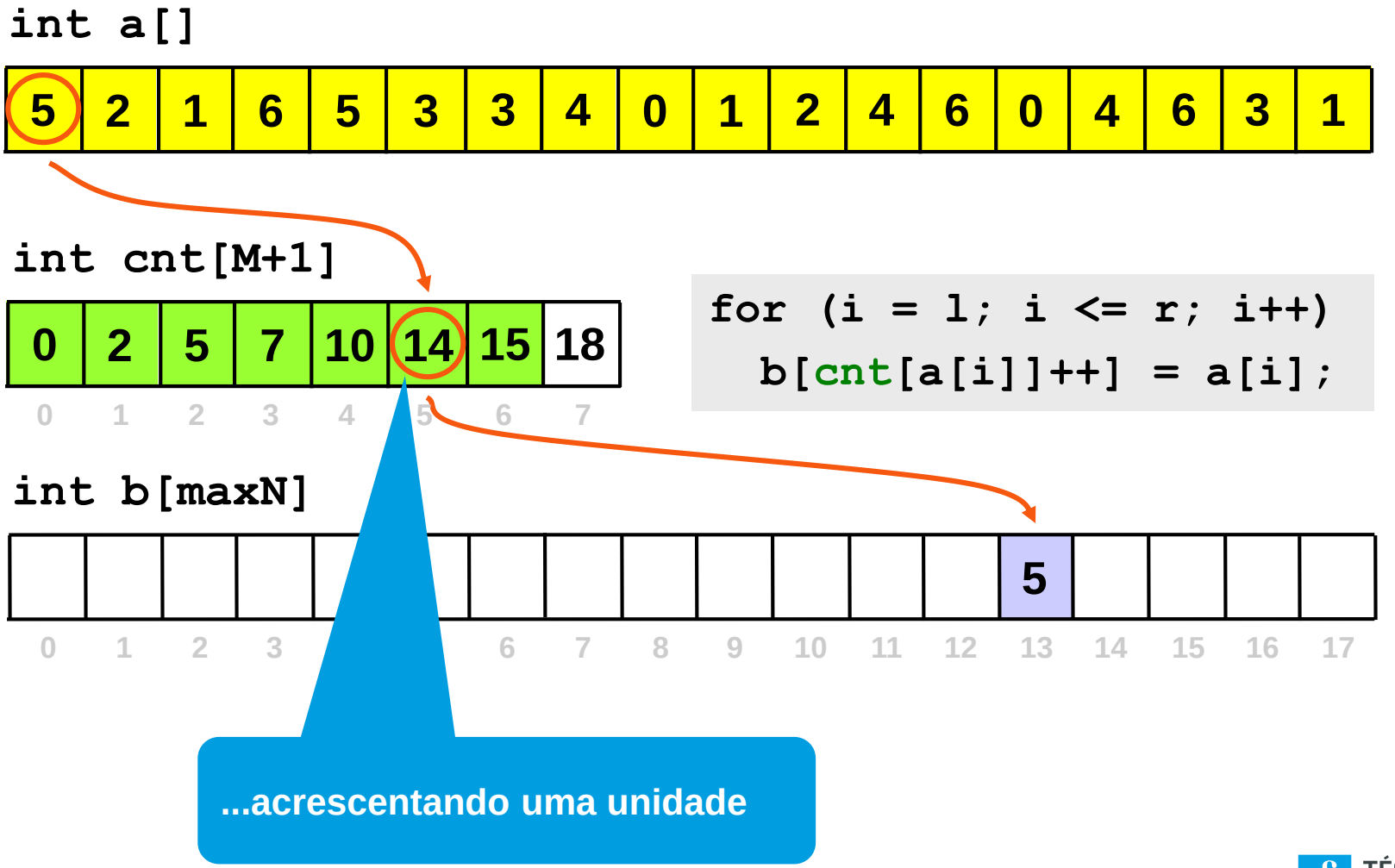
Counting Sort: passo 4

$N = 18 \quad M = 7$



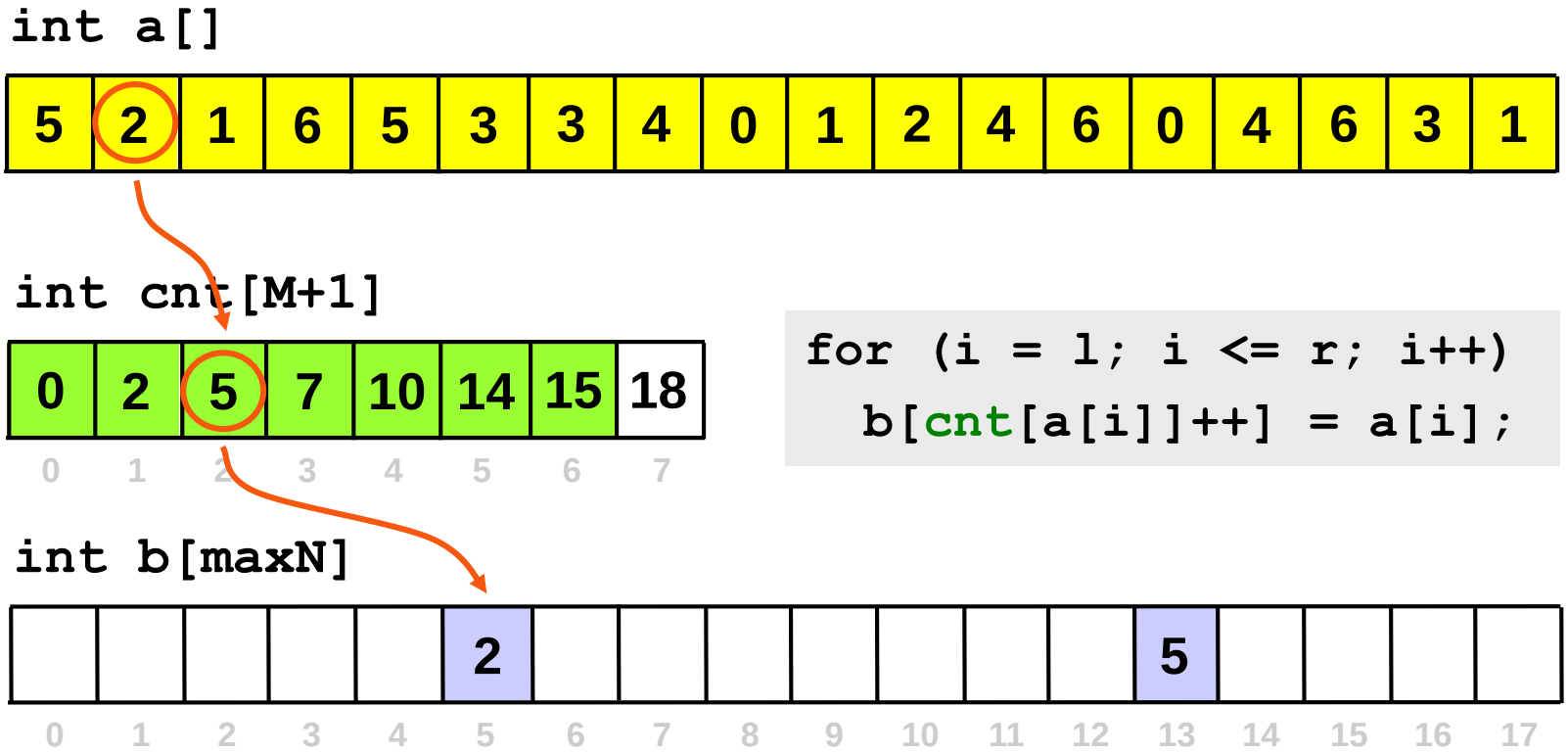
Counting Sort: passo 4

$N = 18 \quad M = 7$



Counting Sort: passo 4

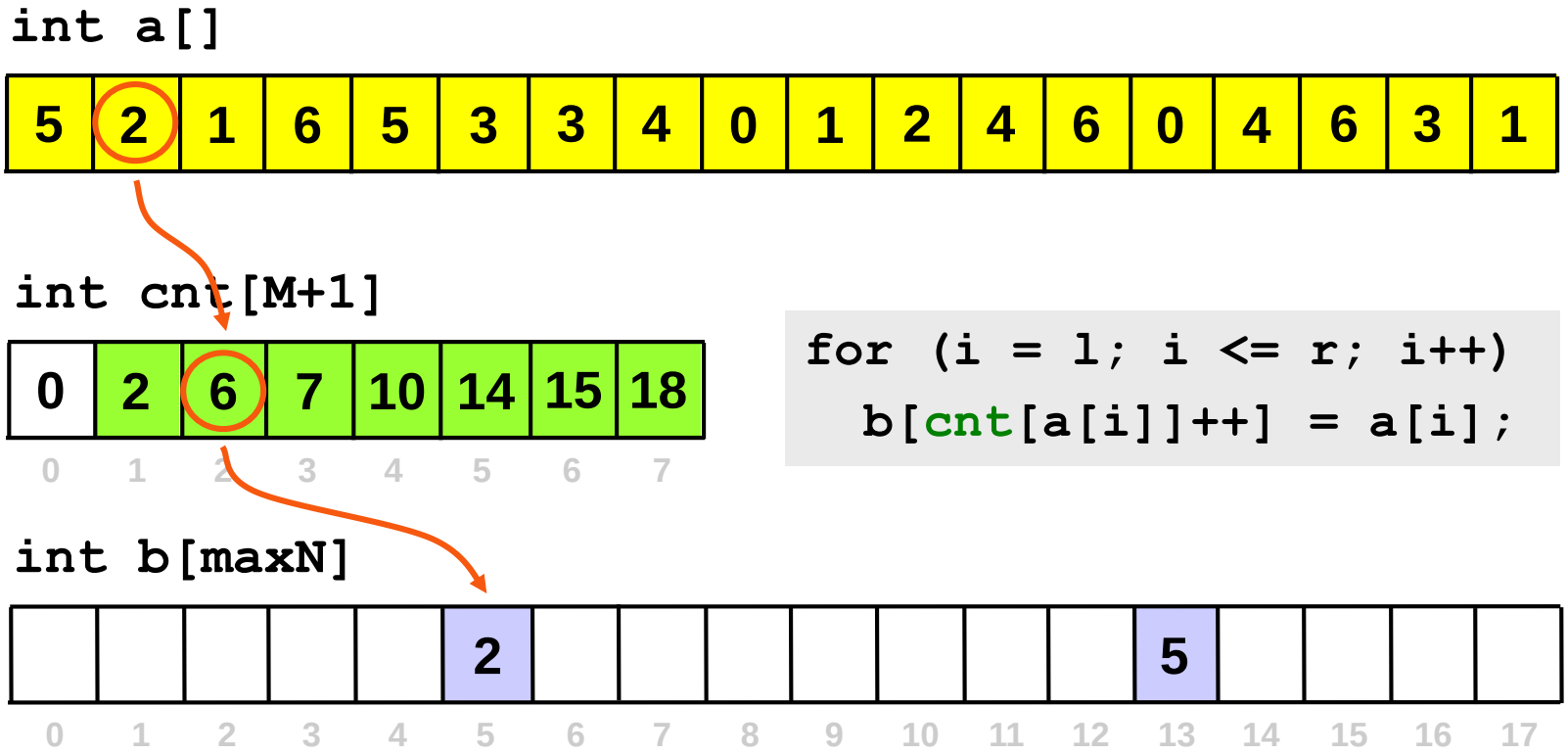
$N = 18 \quad M = 7$




```
for (i = l; i <= r; i++)
  b[cnt[a[i]]++] = a[i];
```

Counting Sort: passo 4

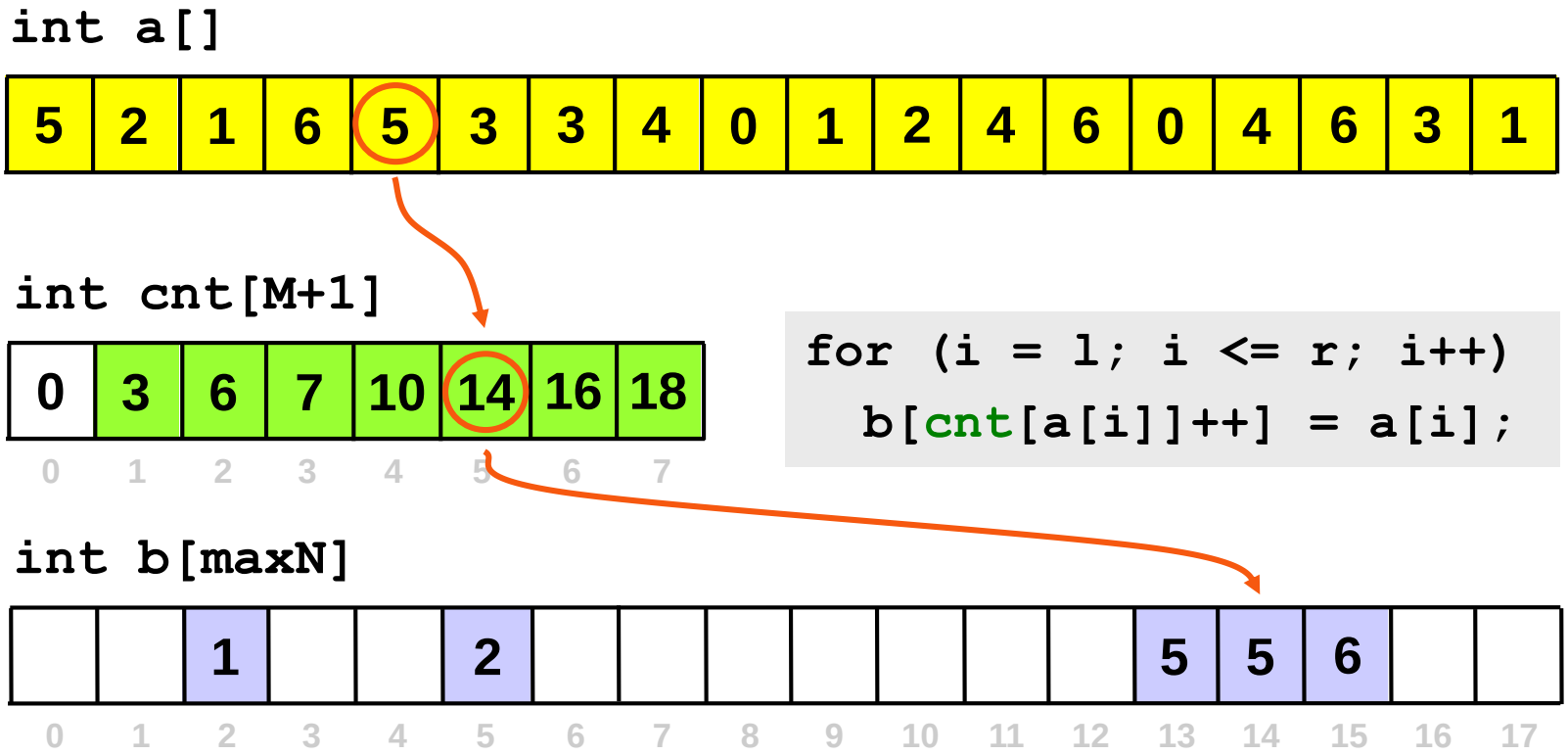
$N = 18 \quad M = 7$




```
for (i = l; i <= r; i++)
  b[cnt[a[i]]++] = a[i];
```

Counting Sort: passo 4

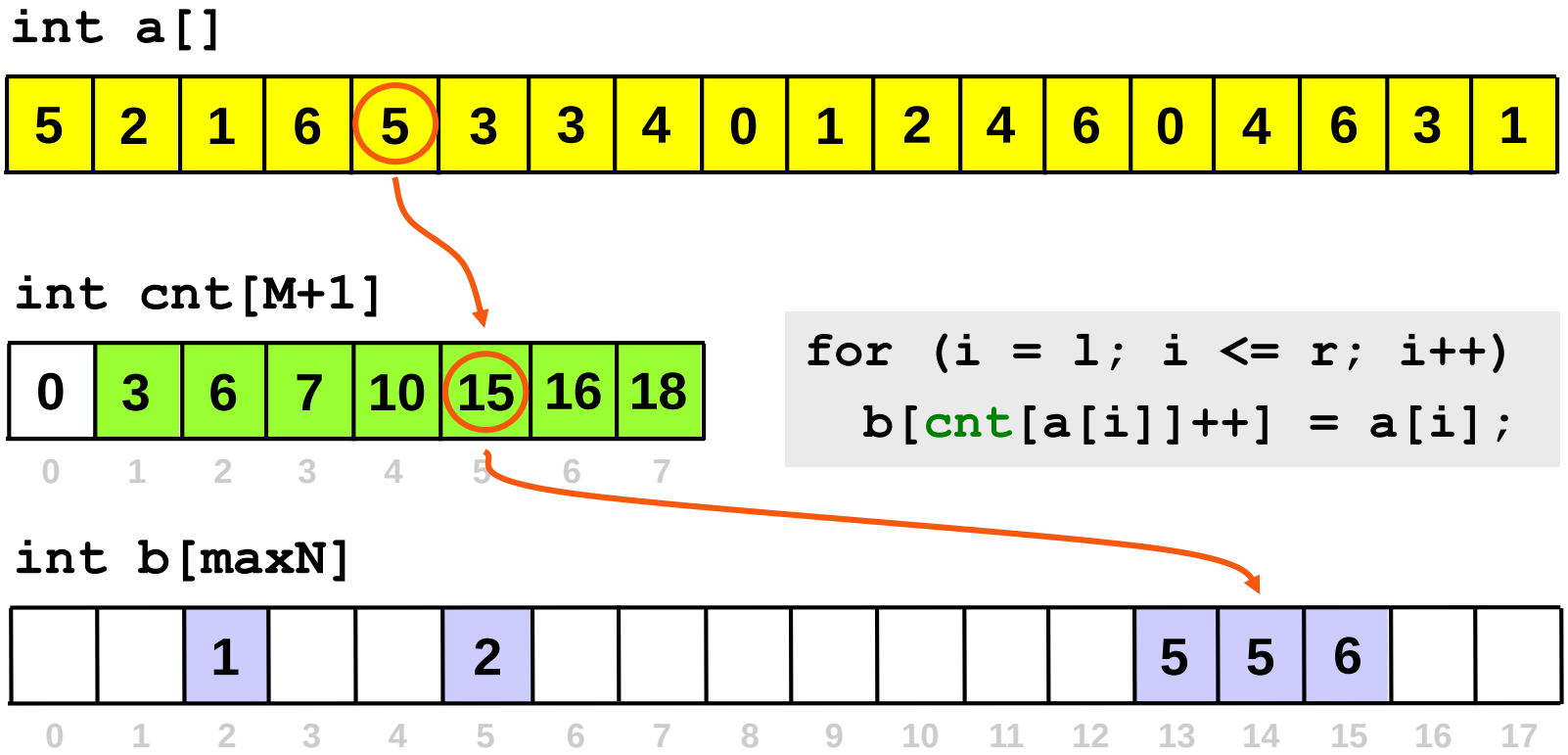
$N = 18 \quad M = 7$




```
for (i = l; i <= r; i++)
  b[cnt[a[i]]++] = a[i];
```

Counting Sort: passo 4

$N = 18 \quad M = 7$




```
for (i = l; i <= r; i++)
  b[cnt[a[i]]++] = a[i];
```

Counting Sort: passo 4

$N = 18 \quad M = 7$

`int a[]`

5	2	1	6	5	3	3	4	0	1	2	4	6	0	4	6	3	1

`int cnt[M+1]`

2	5	7	10	13	15	18	18
0	1	2	3	4	5	6	7

```
for (i = l; i <= r; i++)
  b[cnt[a[i]]++] = a[i];
```

`int b[maxN]`

0	0	1	1	1	2	2	3	3	3	4	4	4	5	5	6	6	6
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Counting Sort: passo 5

$N = 18 \quad M = 7$

`int a[]`

0	0	1	1	1	2	2	3	3	3	4	4	4	5	5	6	6	6

`int cnt[M+1]`

2	5	7	10	13	15	18	18
0	1	2	3	4	5	6	7

```
for (i = l; i <= r; i++)
  a[i] = b[i-l];
```

`int b[maxN]`

0	0	1	1	1	2	2	3	3	3	4	4	4	5	5	6	6	6
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

Counting Sort - Passos

- Utiliza vectores auxiliares `cnt[M + 1]` e `b[maxN]`
- Passo 1: inicializa cada posição de `cnt` a `0`
- Passo 2
  - Para cada posição `i` de `a`, faz `cnt[a[i]+1]++`
  - No fim, cada posição `i` de `cnt` tem o número de vezes que a chave `i-1` aparece em `a`
- Passo 3: acumula em cada elemento de `cnt` os elementos anteriores: `cnt[i]` indica a posição ordenada do primeiro elemento com chave `i`
- Passo 4: guarda em `b` os valores de a ordenados: `b[cnt[a[i]]++]=a[i]`
- Passo 5: Copia `b` para `a`

Counting Sort

```
void distcount(int a[], int left, int right)
{
  int i, j, cnt[M+1];
  int b[maxN];
  for (j = 0; j <= M; j++)
    cnt[j] = 0;
  for (i = left; i <= right; i++)
    cnt[a[i]+1]++;
  for (j = 1; j <= M; j++)
    cnt[j] += cnt[j-1];
  for (i = left; i <= right; i++)
    b[cnt[a[i]]++] = a[i];
  for (i = left; i <= right; i++)
    a[i] = b[i-left];
}
```

b é um vector auxiliar do tamanho MaxN maior que o tamanho de a

Este vai ser o meu histograma

Acrescento uma ocorrência à casa certa + 1

Transformo o histograma numa distribuição cumulativa

Agora já sei onde acaba cada um dos números possíveis, e vou preenchendo o b

Resta-me copiar o conteudo do vector auxiliar b para a

Counting Sort - Complexidade

- Complexidade em tempo de execução
  - Dois ciclos relativos à dimensão das chaves M
  - Três ciclos relativos às N chaves
  - Complexidade: $O(N+M)$
- É estável
- Não é *in-place*

DEI
DEPARTAMENTO
DE ENGENHARIA INFORMÁTICA
TÉCNICO LISBOA

Radix Sort

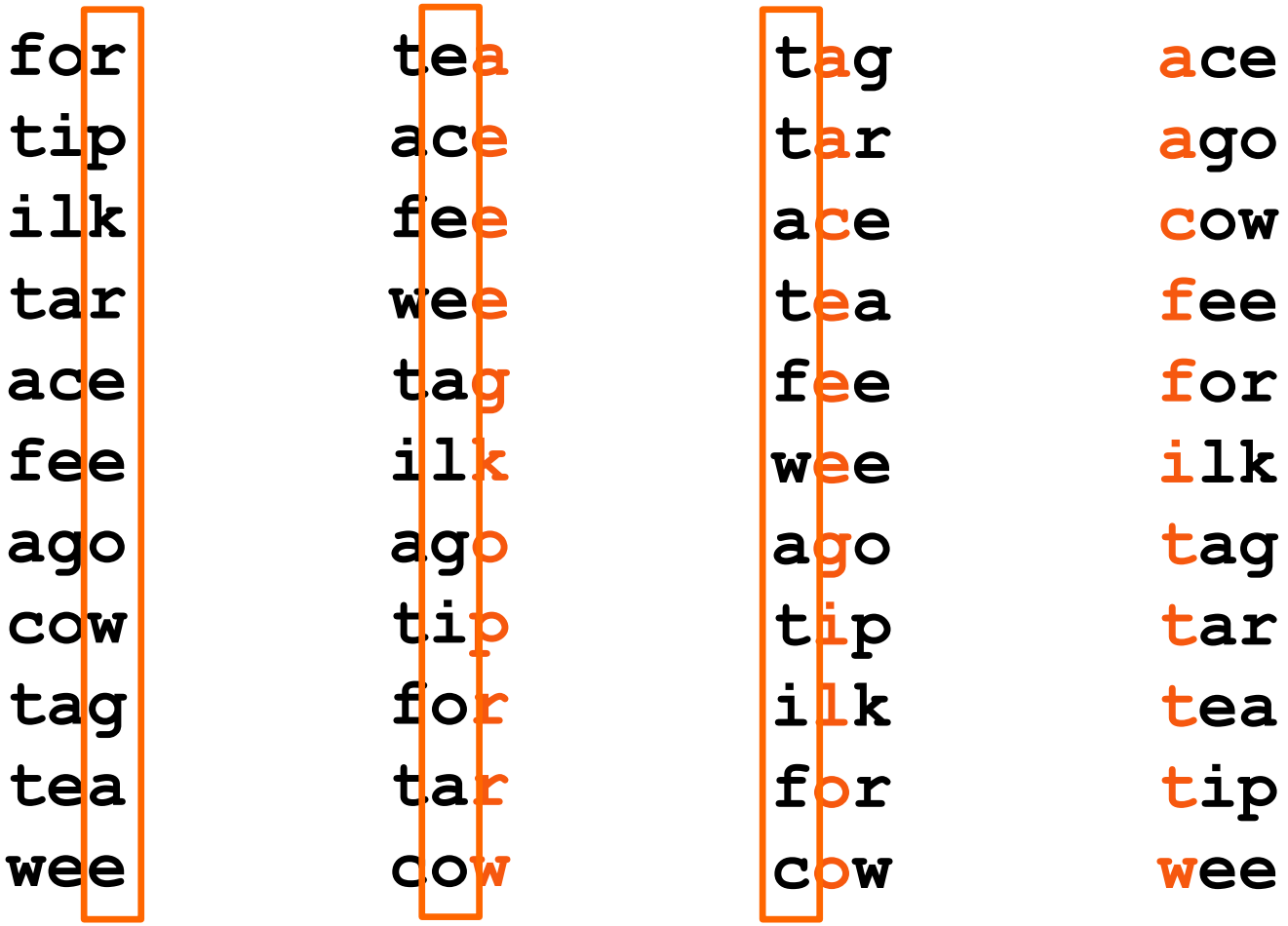
IAED

Radix Sort - Motivação

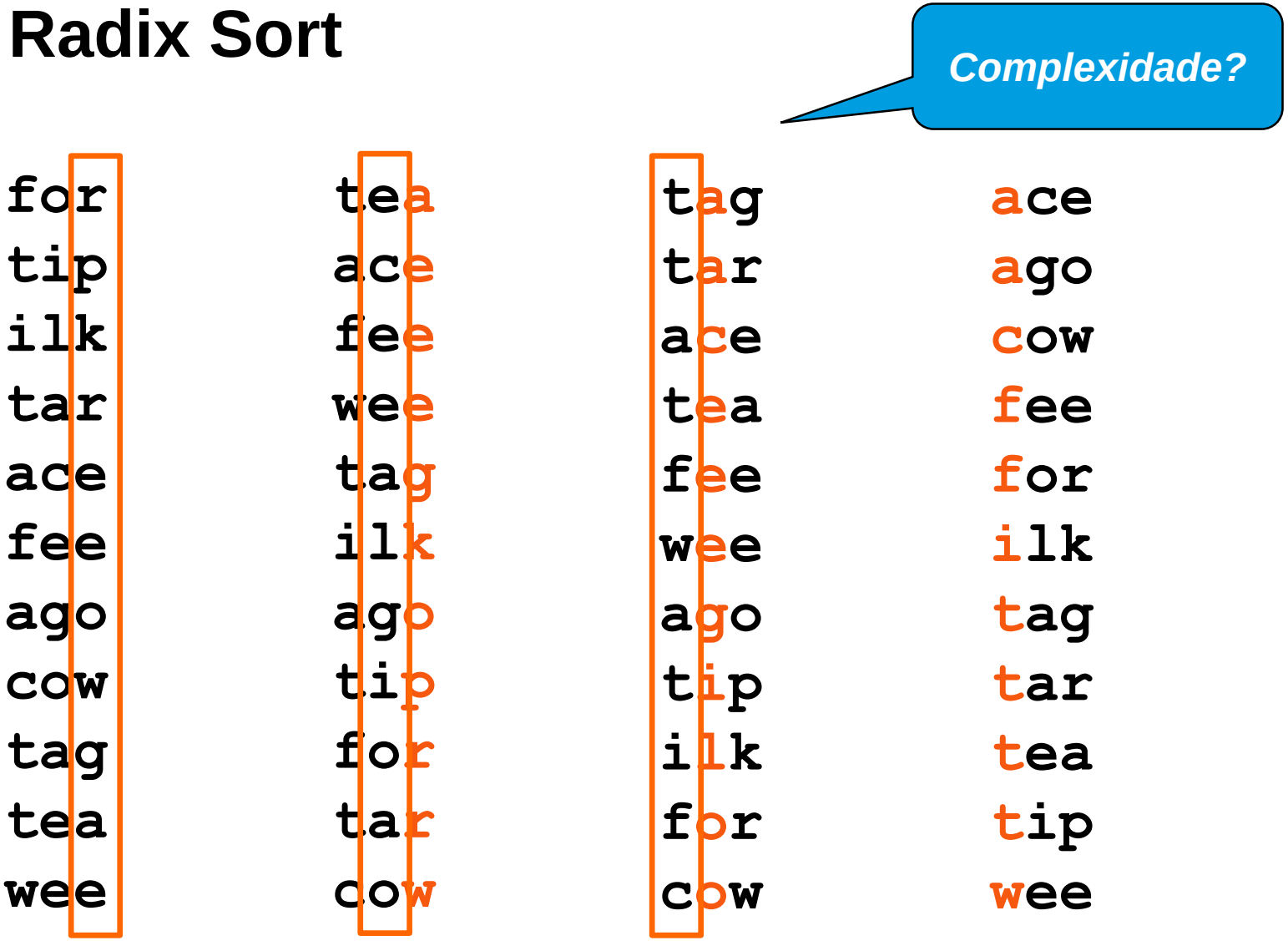
- Baseia-se na estrutura dos elementos a ordenar.
- Ordena elementos processando cada digito/bit/carácter separadamente, usando e.g. o Counting Sort
  - Ex.: organização de strings, tomamos cada letra como um elemento distinto, ou seja, temos M=26.
  - Ex.: ordenação de inteiros, tomamos cada digito como um elemento distinto, ou seja, temos M=10.

Primeira versão: **RADIX LSD**
Aplica o Counting Sort sucessivamente dos dígitos menos significativos *(**L**ess **S**ignificant **D**igits)* para os dígitos mais significativos.

LSD Radix Sort



LSD Radix Sort

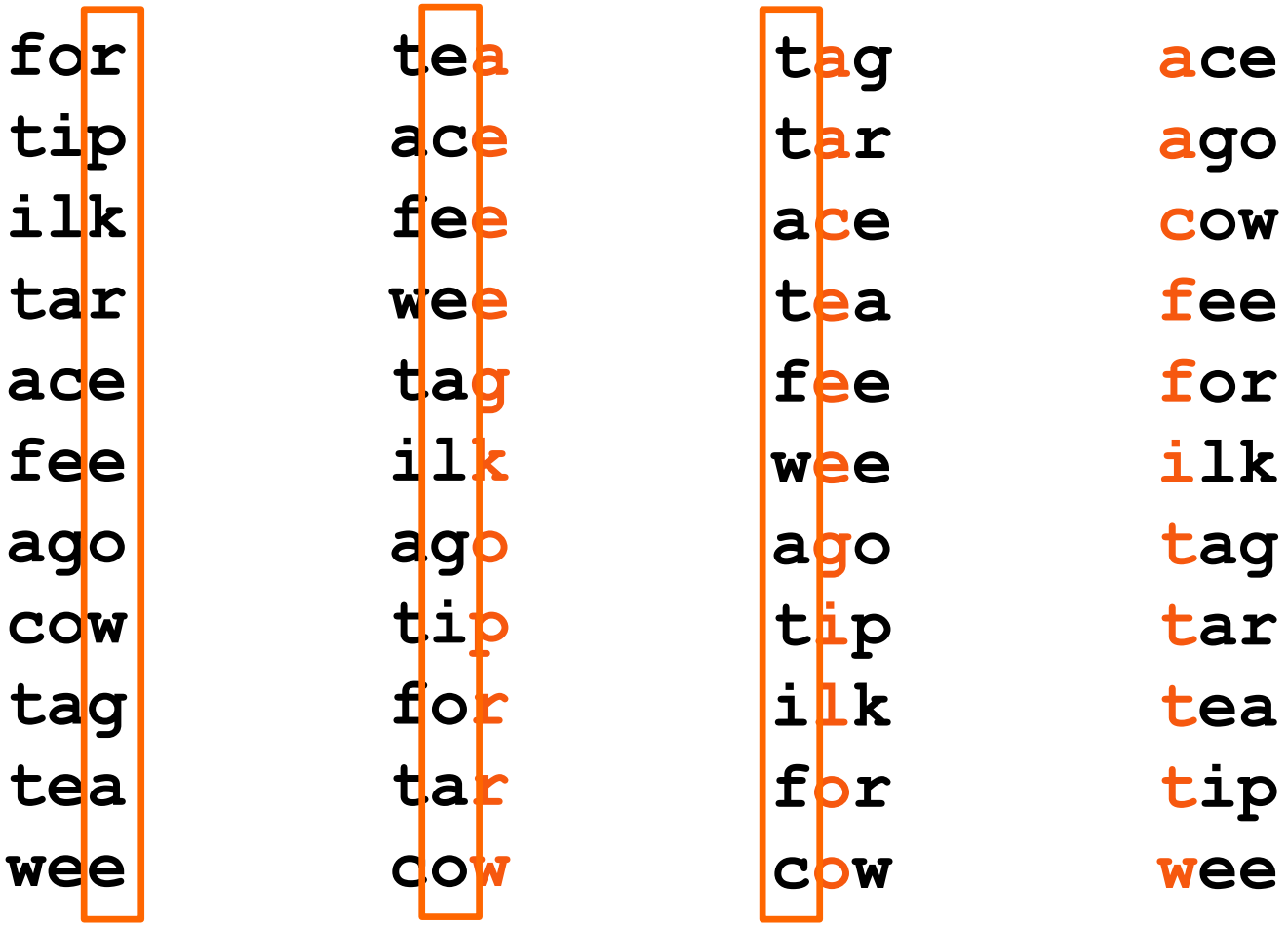


LSD Radix Sort

(N colunas, M chars)

Complexidade?

O(N.M)



LSD Radix Sort

- Ordena aplicando sucessivamente Counting Sort, dos dígitos menos significativos para os dígitos mais significativos

- Aplicável apenas a chaves de dimensão fixa

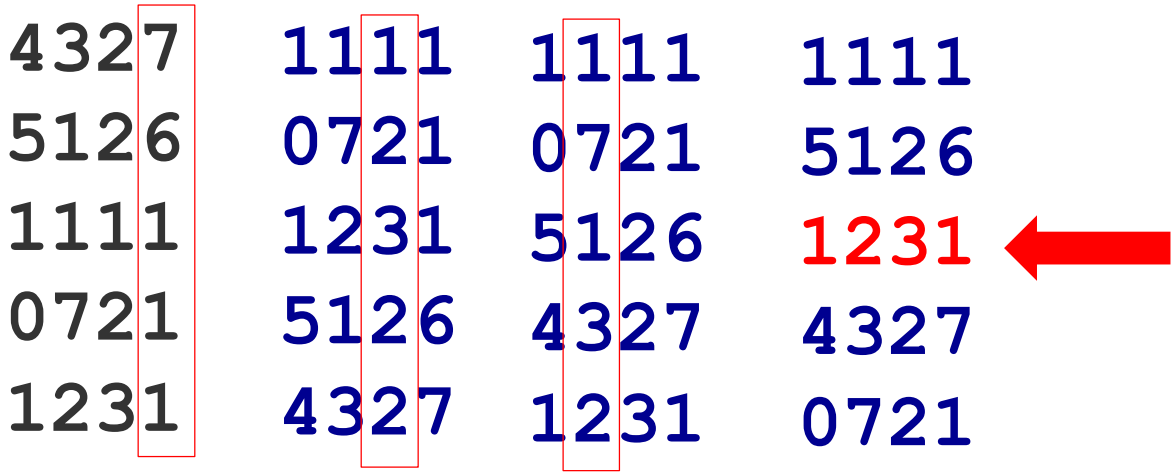
- Funciona porque o Counting Sort é estável
  - Preserva as ordenações das iterações anteriores

Exercício

Considere a aplicação do algoritmo **radix sort LSD**, em que cada passo os elementos são ordenados considerando um dígito, ao seguinte vector:

A = {4327, 5126, 1111, 0721, 1231}

Qual é o terceiro número da sequência, após o algoritmo ter considerado três dígitos?

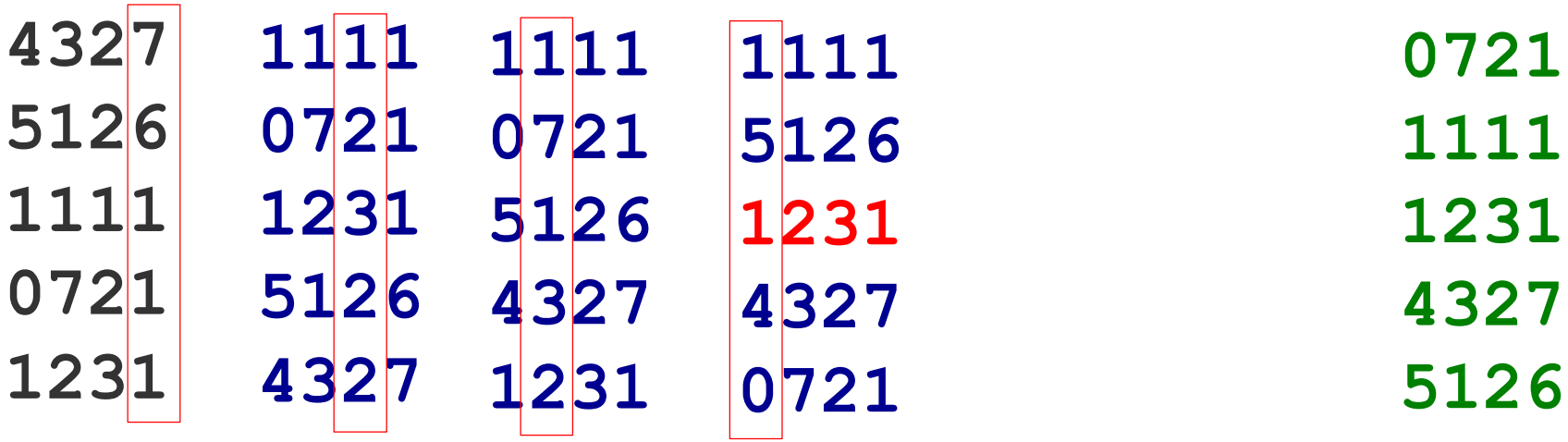


Exercício

Considere a aplicação do algoritmo **radix sort LSD**, em que cada passo os elementos são ordenados considerando um dígito, ao seguinte vector:

A = {4327, 5126, 1111, 0721, 1231}

Qual é o terceiro número da sequência, após o algoritmo ter considerado três dígitos?



MSD Radix Sort

Segunda versão: **RADIX MSD**
Aplica o counting Sort sucessivamente começando pelos dígitos mais significativos *(**M**ost **S**ignificant **D**igits).*

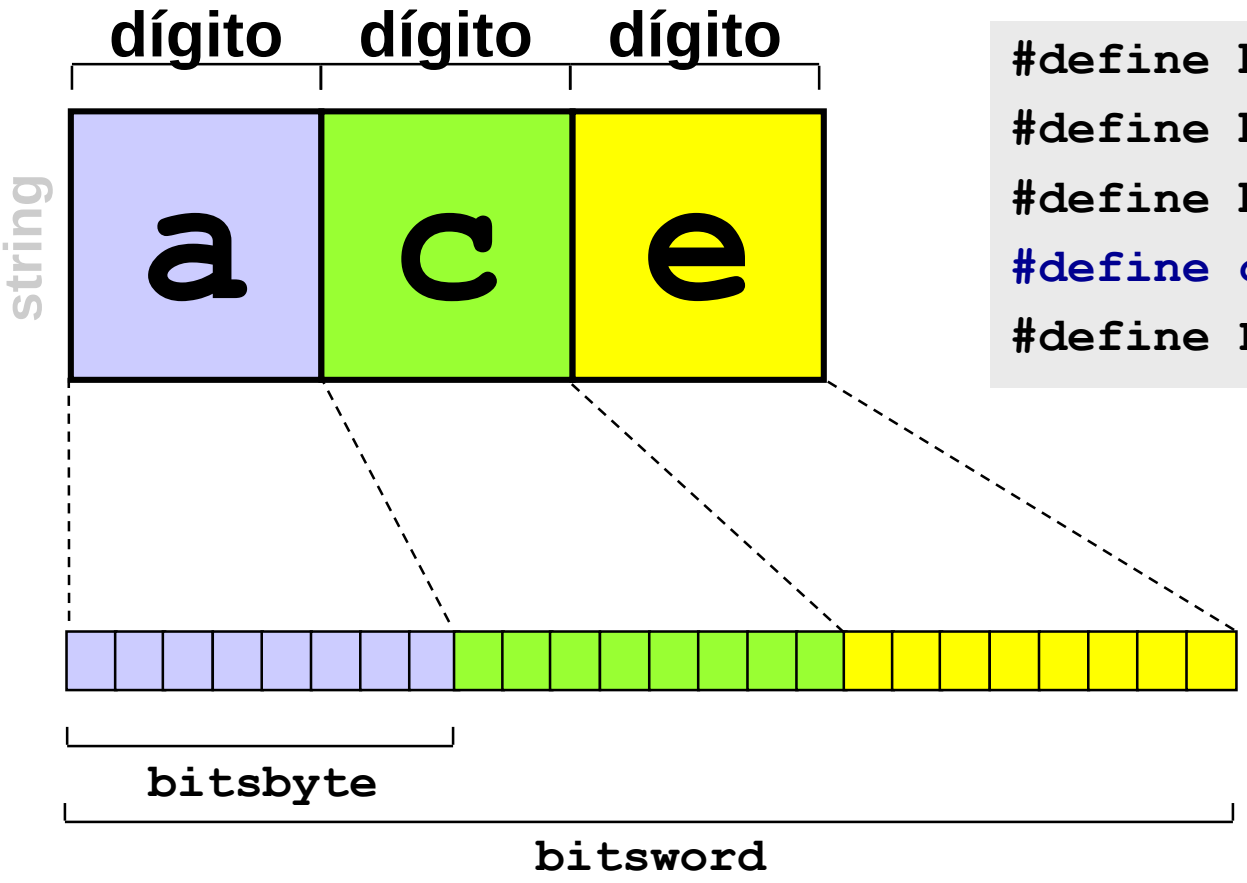
MSD Radix Sort

Radix Sort MSD considera um digito da chave de cada vez, começando do digito mais significativo

MSD Radix Sort

- Começando no dígito mais significativo (Most Significant Digit), considerar cada *n*-ésimo dígito e ordenar vector usando apenas esse dígito
- Realização: Para cada dígito, do de maior peso para o de menor peso:
  - Colocar chaves em *M caixas* (uma para cada valor possível)
  - Ordenar elementos de cada caixa utilizando dígitos subsequentes
  - Se número de elementos não for superior a *M*, utilizar Insertion Sort
- Resumo: utilizar dígitos, do maior peso para o menor peso e ordenar o vector utilizando o Counting Sort para cada dígito

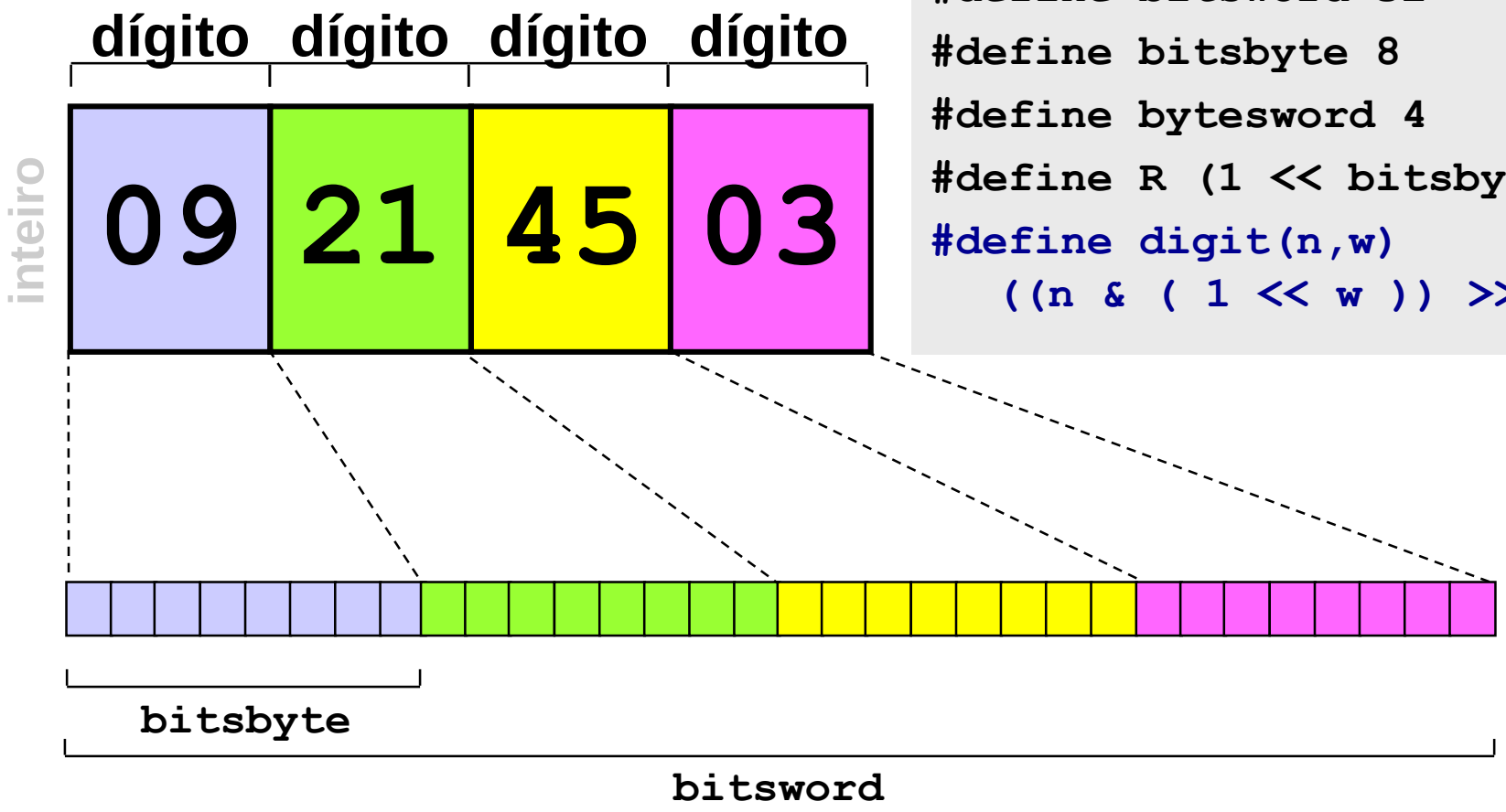
Implementação do Radix Sort | Definições




```
#define bitsword 24
#define bitsbyte 8
#define bytesword 3
#define digit(a,b) a[b]
#define R (1 << bitsbyte)
```

número de valores possíveis para um digito (e.g, 256)

Implementação do Radix Sort | Definições




```
#define bitsword 32
#define bitsbyte 8
#define bytesword 4
#define R (1 << bitsbyte)
#define digit(n,w)
   ((n & ( 1 << w )) >> w)
```

LSD Radix Sort

```
void radixLSD(Item a[], int l, int r)
{
  int i, j, w, count[R+1];
  for (w = bytesword-1; w >= 0; w--) {
    for (j = 0; j < R; j++)
      count[j] = 0;
    for (i = l; i <= r; i++)
      count[digit(a[i], w) + 1]++;
    for (j = 1; j < R; j++)
      count[j] += count[j-1];
    for (i = l; i <= r; i++)
      aux[count[digit(a[i], w)]++] = a[i];
    for (i = l; i <= r; i++)
      a[i] = aux[i];
  }
}
```

LSD Radix Sort

```
void radixLSD(Item a[], int l, int r)
{
  int i, j, w, count[R+1];
  for (w = bytesword-1; w >= 0; w--) {
    for (j = 0; j < R; j++)
      count[j] = 0;
    for (i = l; i <= r; i++)
      count[digit(a[i], w) + 1]++;
    for (j = 1; j < R; j++)
      count[j] += count[j-1];
    for (i = l; i <= r; i++)
      aux[count[digit(a[i], w)]++] = a[i];
    for (i = l; i <= r; i++)
      a[i] = aux[i];
  }
}
```

counting sort para o dígito `w`

MSD Radix Sort

```
#define bin(A) l+count[A]

void radixMSD(Item a[], int l, int r, int w)
{
  int i, j, count[R+1];

  if (w > bytesword)
    return;

  if (r-l <= M) {
    insertion(a, l, r);
    return;
  }
...
```

MSD Radix Sort

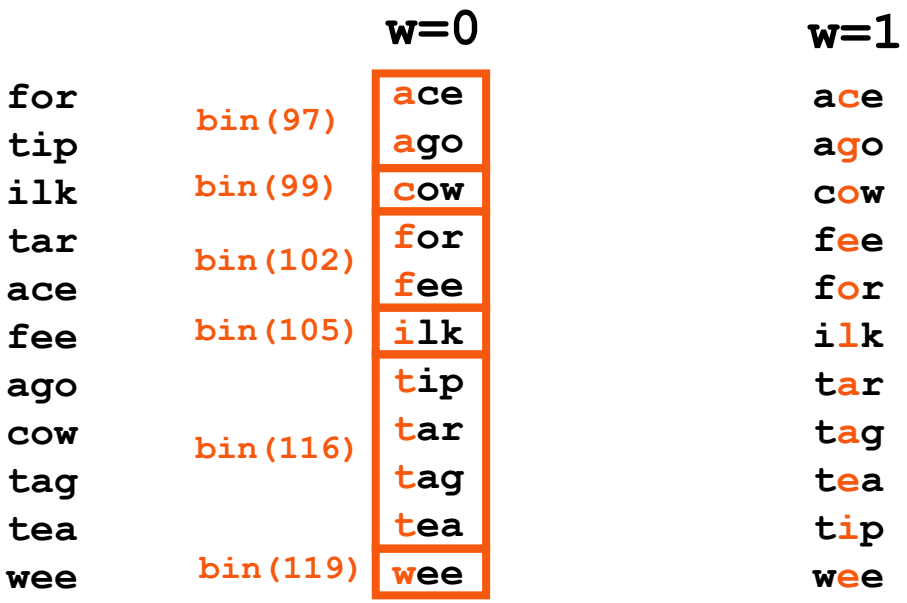
```
...

for (j = 0; j < R; j++)
  count[j] = 0;
for (i = l; i <= r; i++)
  count[digit(a[i], w) + 1]++;
for (j = 1; j < R; j++)
  count[j] += count[j-1];
for (i = l; i <= r; i++)
  aux[l+count[digit(a[i], w)]++] = a[i];
for (i = l; i <= r; i++)
  a[i] = aux[i];

...
```

MSD Radix Sort

```
...
radixMSD(a, l, bin(0)-1, w+1);
for (j = 0; j < R-1; j++)
   radixMSD(a, bin(j), bin(j+1)-1, w+1);
}
```



Eficiência dos Radix Sorts

- Radix Sort
  - Tempo de execução cresce com número de bytes dos elementos a ordenar (~*nDigits* × *N*)
- MSD Radix Sort
  - Pode ser sublinear na quantidade total de informação das chaves
- LSD Radix Sort
  - $O(N \cdot w / \log R)$, para chaves com w bits
  - Mais preciso: $O(w / \log R \cdot (N + 2^{\log R}))$
  - Mínimo ocorre para $\log R = 0.83 \log N$
  - Espaço adicional: R contadores e vector com tamanho N.

Exercício

Considere a aplicação do algoritmo **radix sort LSD**, em que cada passo os elementos são ordenados considerando um dígito, ao seguinte vector:

```
a=
{48372, 62309, 83861, 91874, 18913, 33829,
47812, 95954, 52377, 22394, 56108, 60991}
```

Qual é o terceiro número da sequência, após o algoritmo ter considerado três dígitos?

Exercício

Considere a aplicação do algoritmo **radix sort LSD**, em que cada passo os elementos são ordenados considerando um dígito, ao seguinte vector:

```
a=
{48372, 62309, 83861, 91874, 18913, 33829,
47812, 95954, 52377, 22394, 56108, 60991}
```

Qual é o terceiro número da sequência, após o algoritmo ter considerado três dígitos?

56108, 62309, 48372, 52377, 22394, 47812, 33829, 83861, 91874, 18913, 95954, 60991