Counting Sort

Prof. Luciano Antonio Digiampietri

Counting Sort

Este algoritmo realiza a ordenação baseada na contagem da quantidade de elementos de diferentes valores.

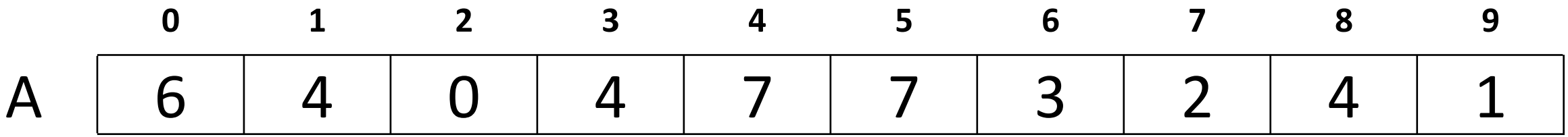
Entrada: conjunto não ordenado números **inteiros cujos valores são menores do que k**.

Saída: arranjo ordenado de elementos.

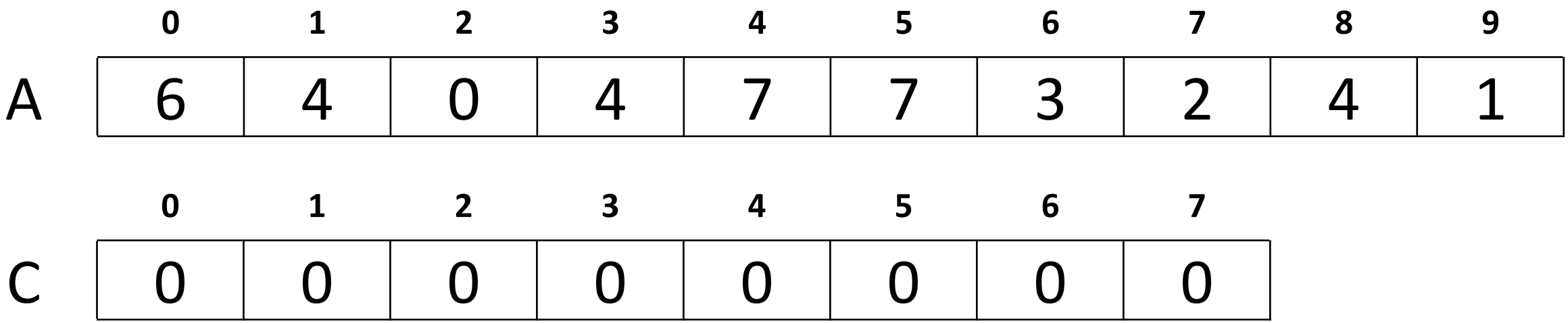
Algoritmo

```
int[] countingSort(int A[], int k){
      int n = A.length;
      int[] B = new int[n];
      int[] C = new int[k];
      for (int i=0;i<k;i++) C[i] = 0;
      for (int j=0;j<n;j++) C[A[j]]++;
      for (int i=1;i<k;i++) C[i] += C[i-1];
      for (int j=n-1;j>=0;j--) {
            B[C[A[j]]-1] = A[j];
            C[A[j]]--;
      }
      return B;
}
```

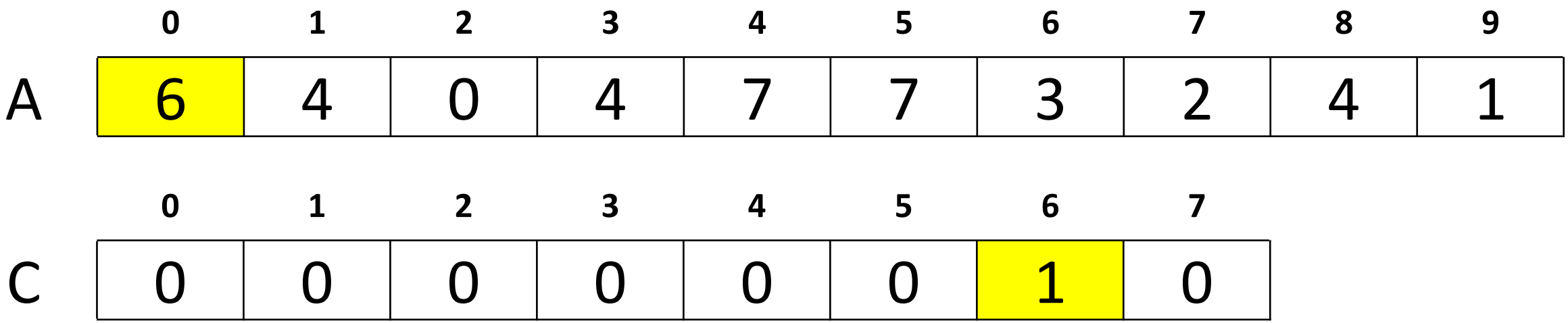
Arranjo de Entrada



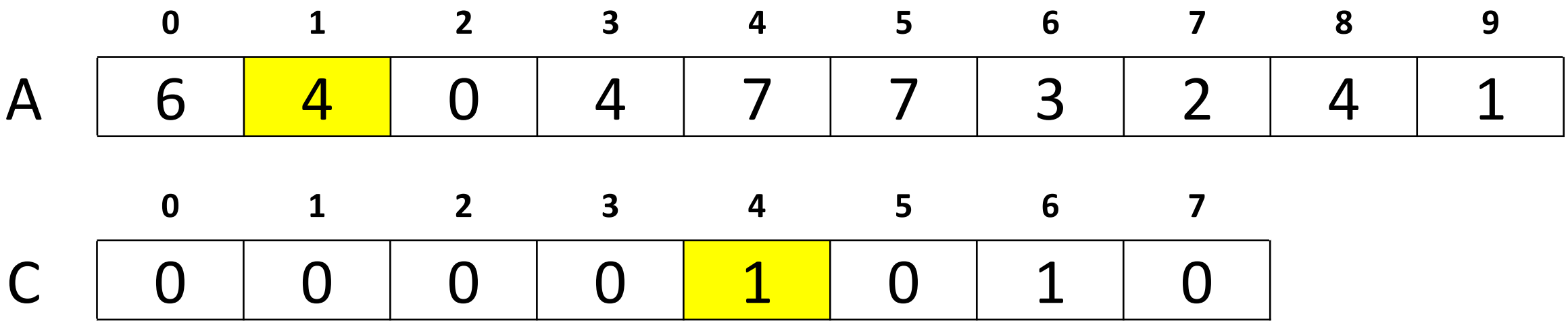
```
for (int i=0;i<k;i++) C[i] = 0;
```



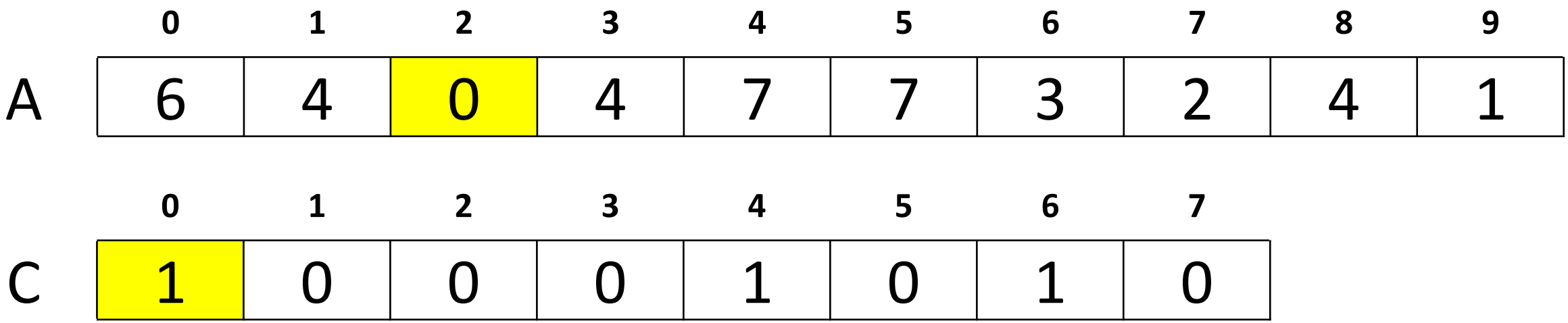
```
for (int j=0;j<n;j++) C[A[j]]++;
```



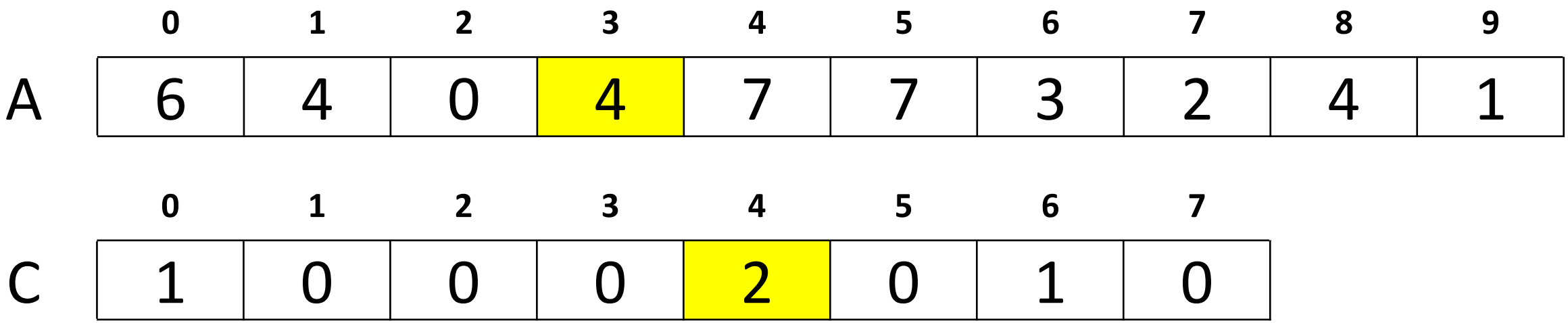
```
for (int j=0;j<n;j++) C[A[j]]++;
```



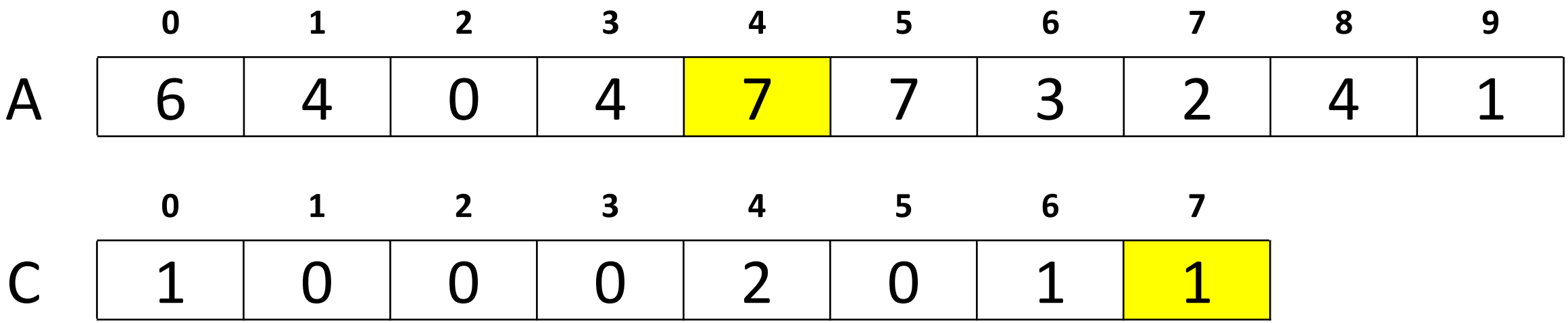
```
for (int j=0;j<n;j++) C[A[j]]++;
```



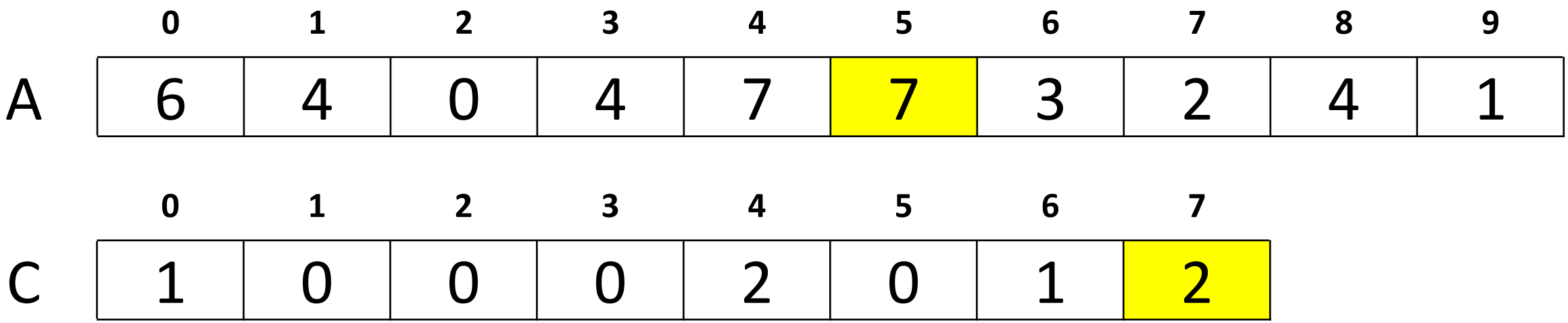
```
for (int j=0;j<n;j++) C[A[j]]++;
```



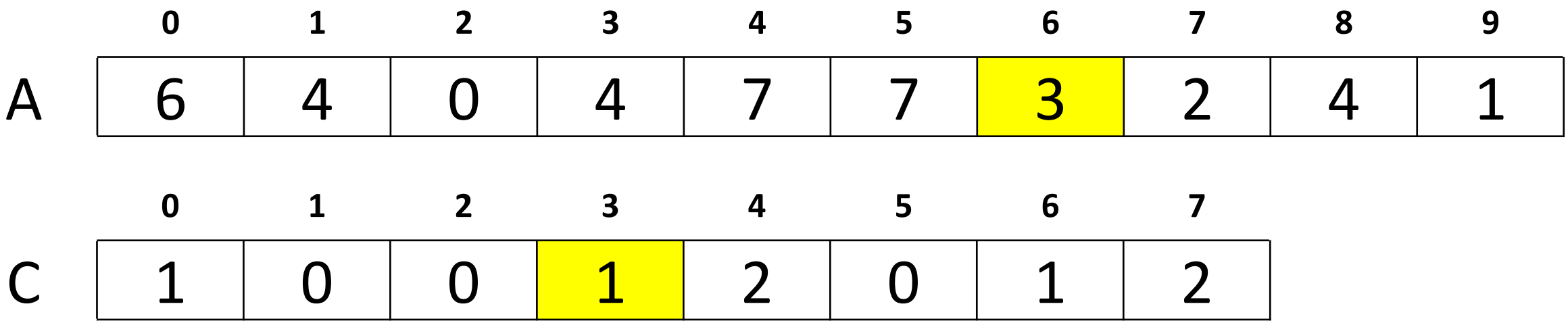
```
for (int j=0;j<n;j++) C[A[j]]++;
```



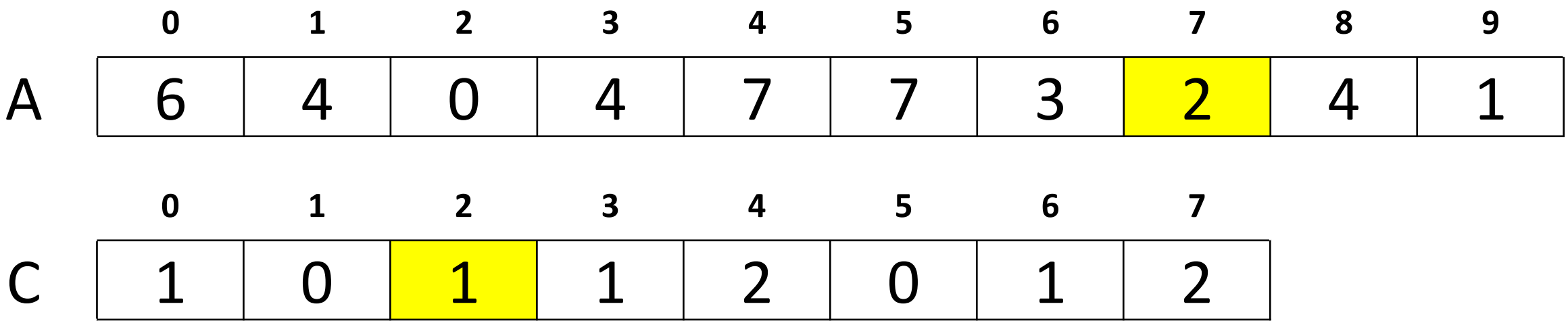
```
for (int j=0;j<n;j++) C[A[j]]++;
```



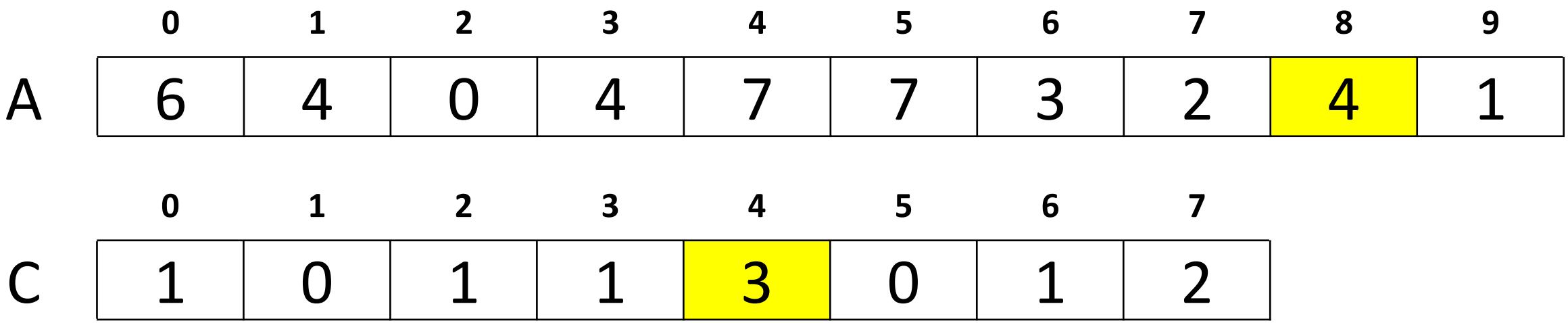
```
for (int j=0;j<n;j++) C[A[j]]++;
```



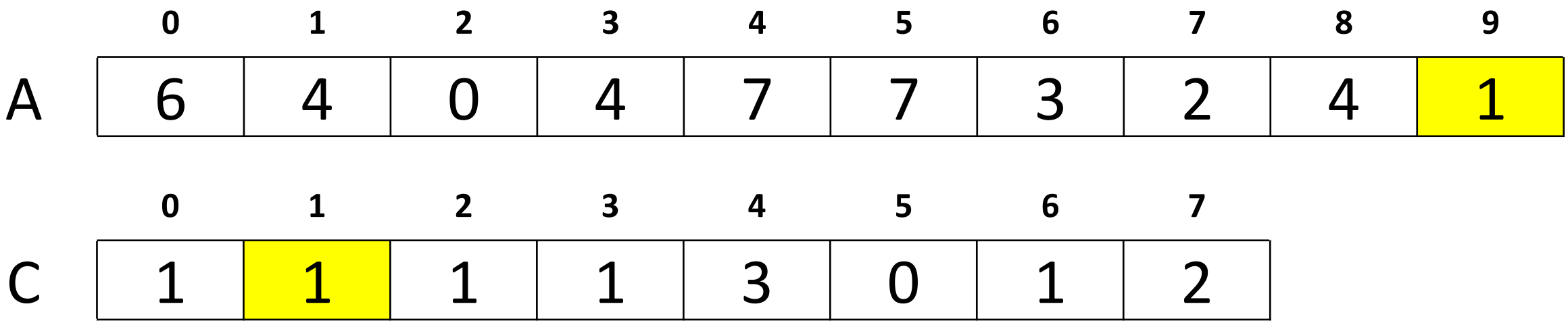
```
for (int j=0;j<n;j++) C[A[j]]++;
```



```
for (int j=0;j<n;j++) C[A[j]]++;
```



```
for (int j=0;j<n;j++) C[A[j]]++;
```



Cada posição do arranjo de contagem contém a quantidade do respectivo valor no arranjo de entrada

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

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

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

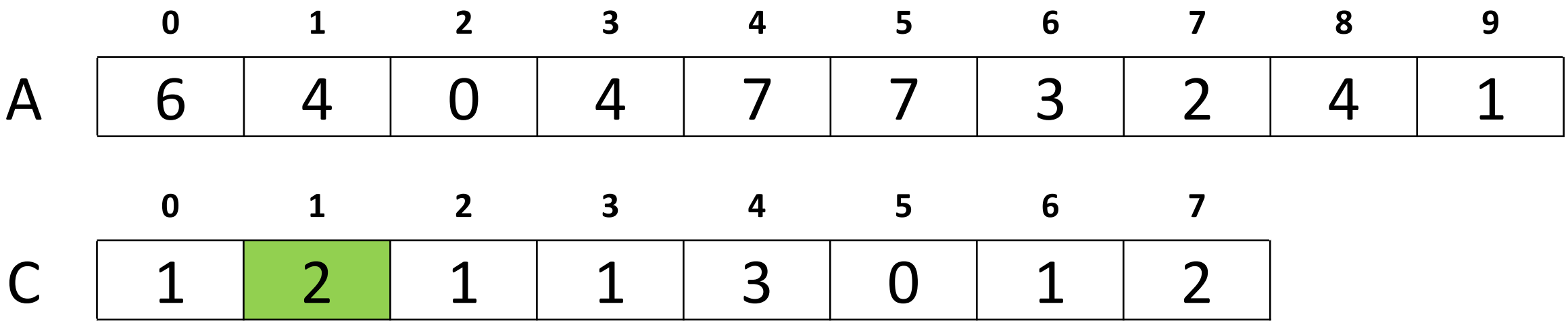
A

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

C

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

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

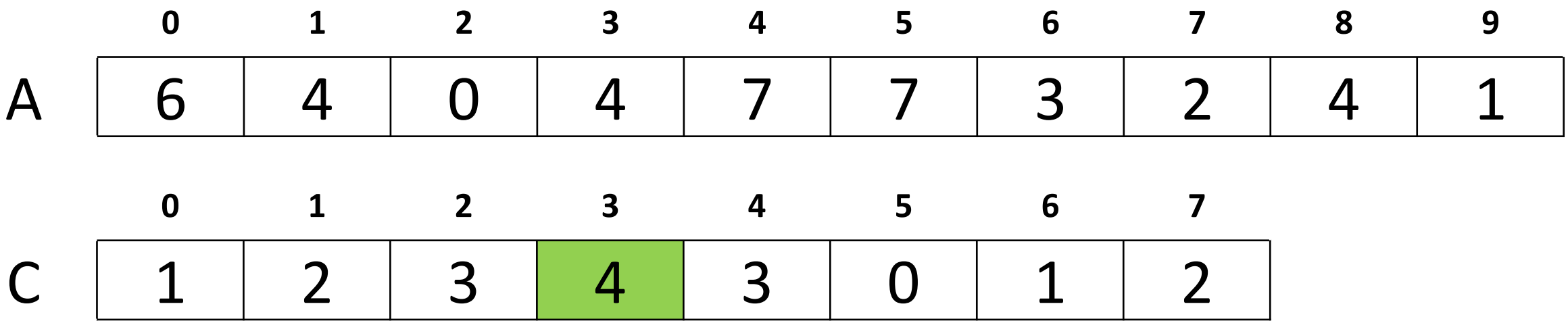


```
for (int i=1;i<k;i++) C[i] += C[i-1];
```

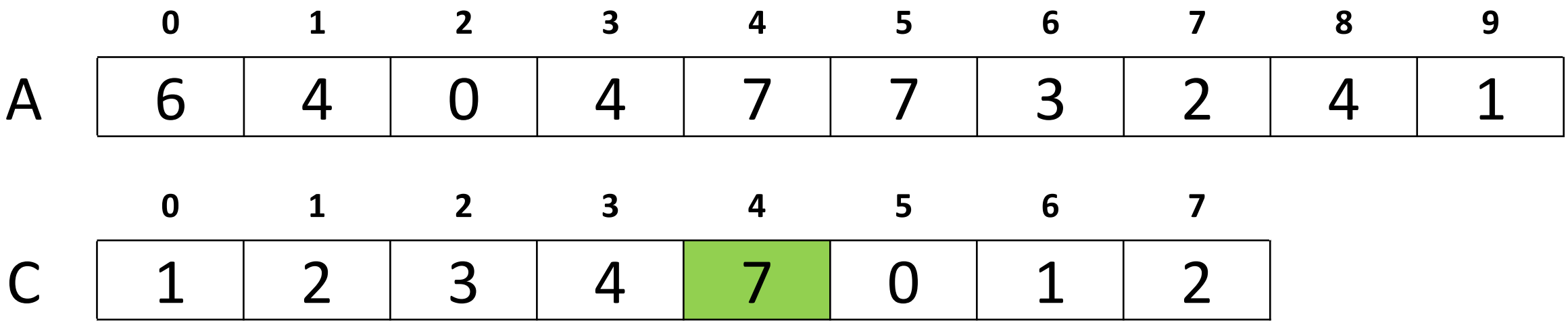
	0	1	2	3	4	5	6	7	8	9
A	6	4	0	4	7	7	3	2	4	1

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

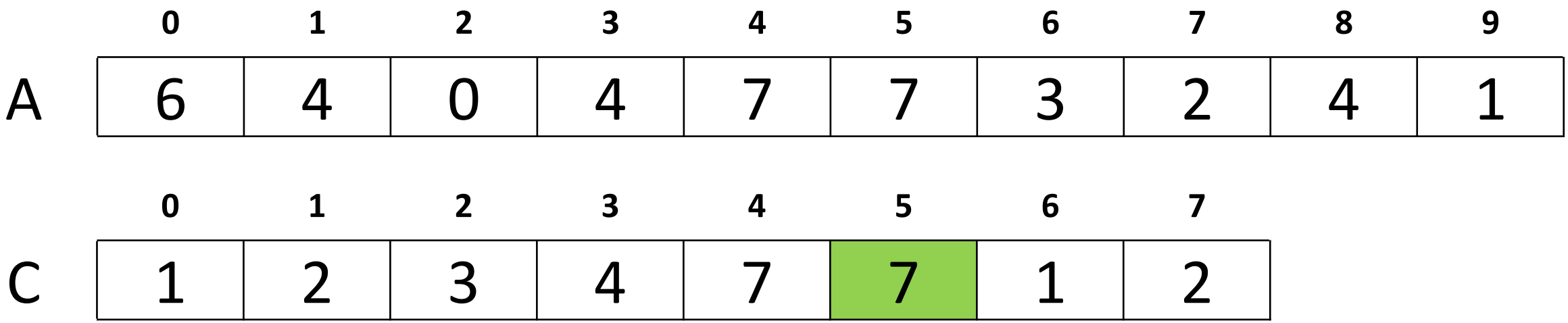
```
for (int i=1;i<k;i++) C[i] += C[i-1];
```



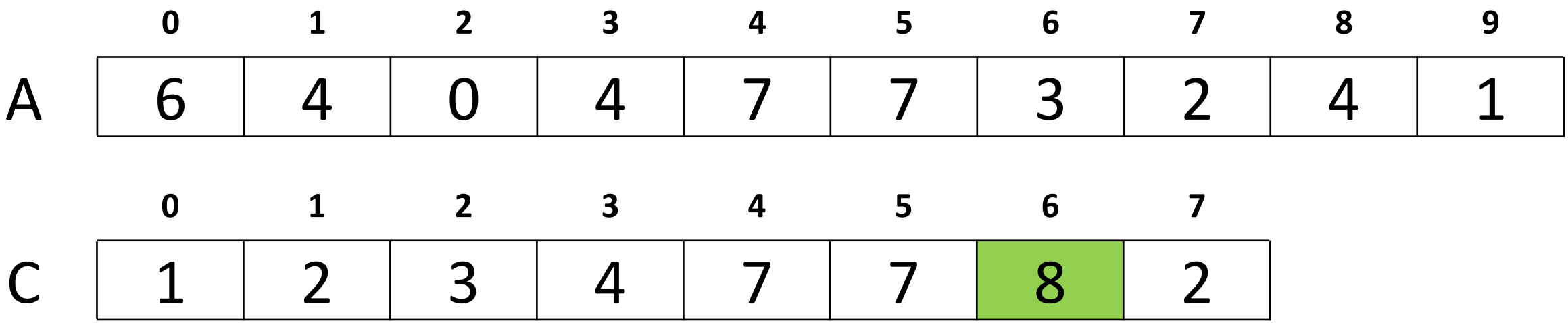
```
for (int i=1;i<k;i++) C[i] += C[i-1];
```



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



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



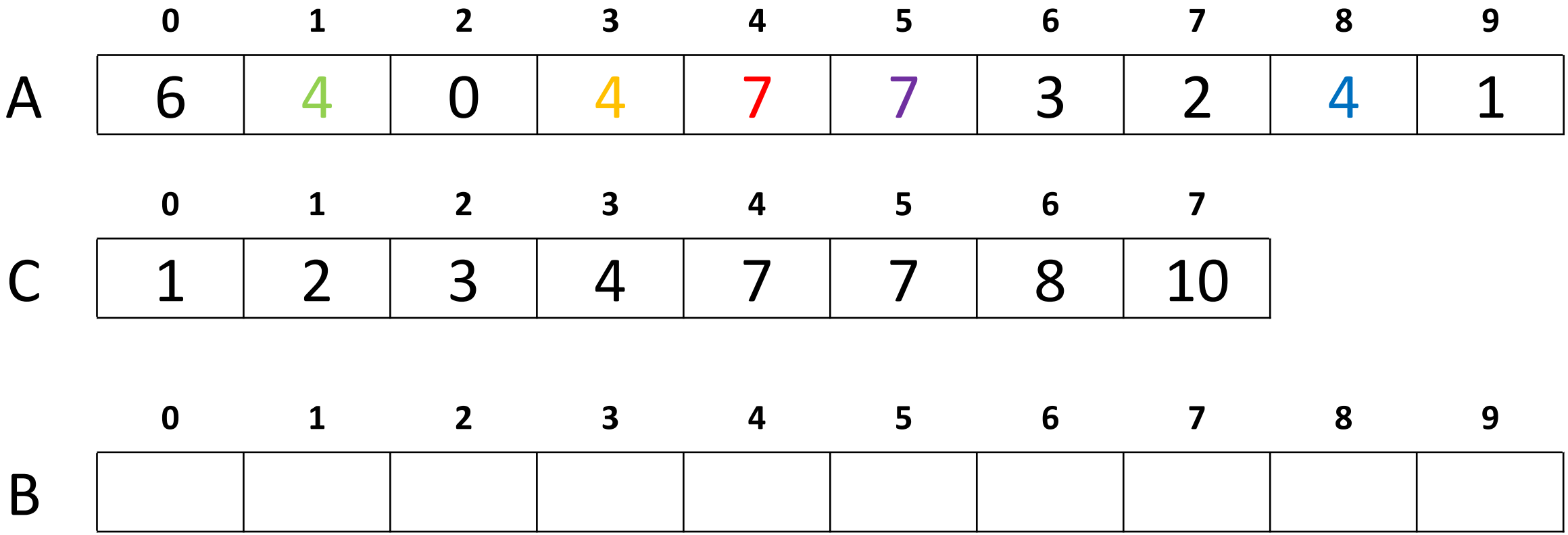
```
for (int i=1;i<k;i++) C[i] += C[i-1];
```

A

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

C

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


0 1 2 3 4 5 6 7 8 9
A
6 4 0 4 7 7 3 2 4 1
0 1 2 3 4 5 6 7
C
1 2 3 4 7 7 8 10
0 1 2 3 4 5 6 7 8 9
B

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

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

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

	0	1	2	3	4	5	6	7	8	9
B										

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

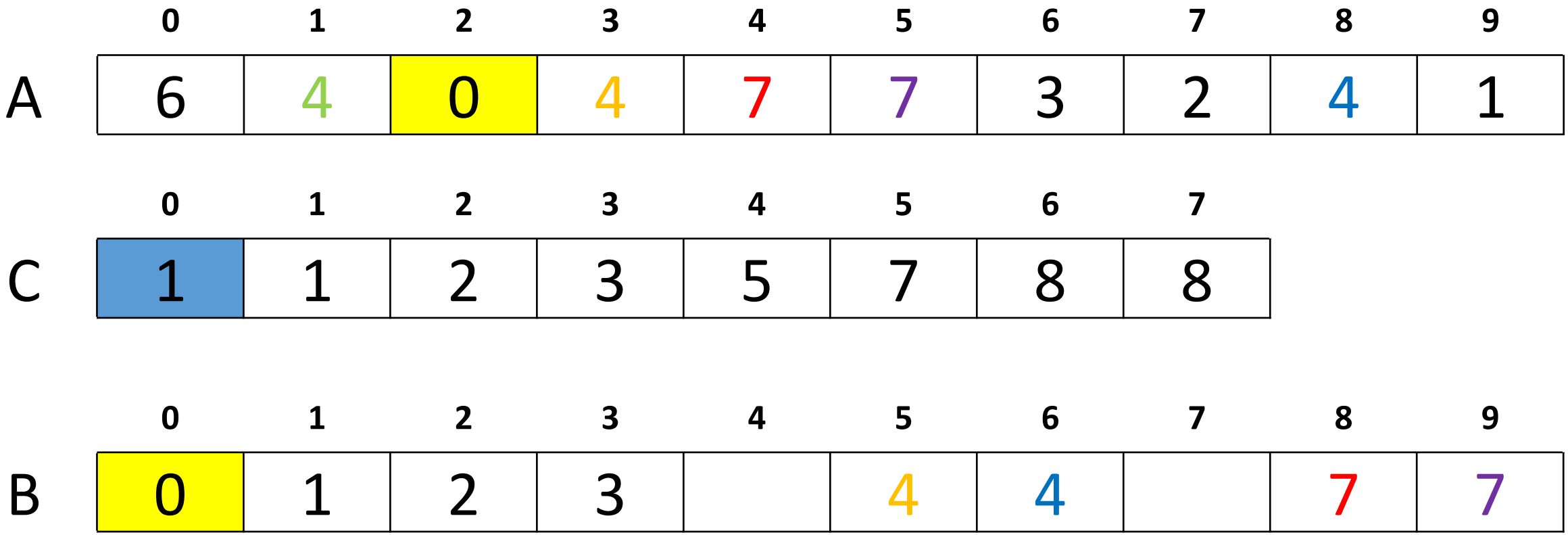
```
for (int j=n-1;j>=0;j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
}
```

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

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

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

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



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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

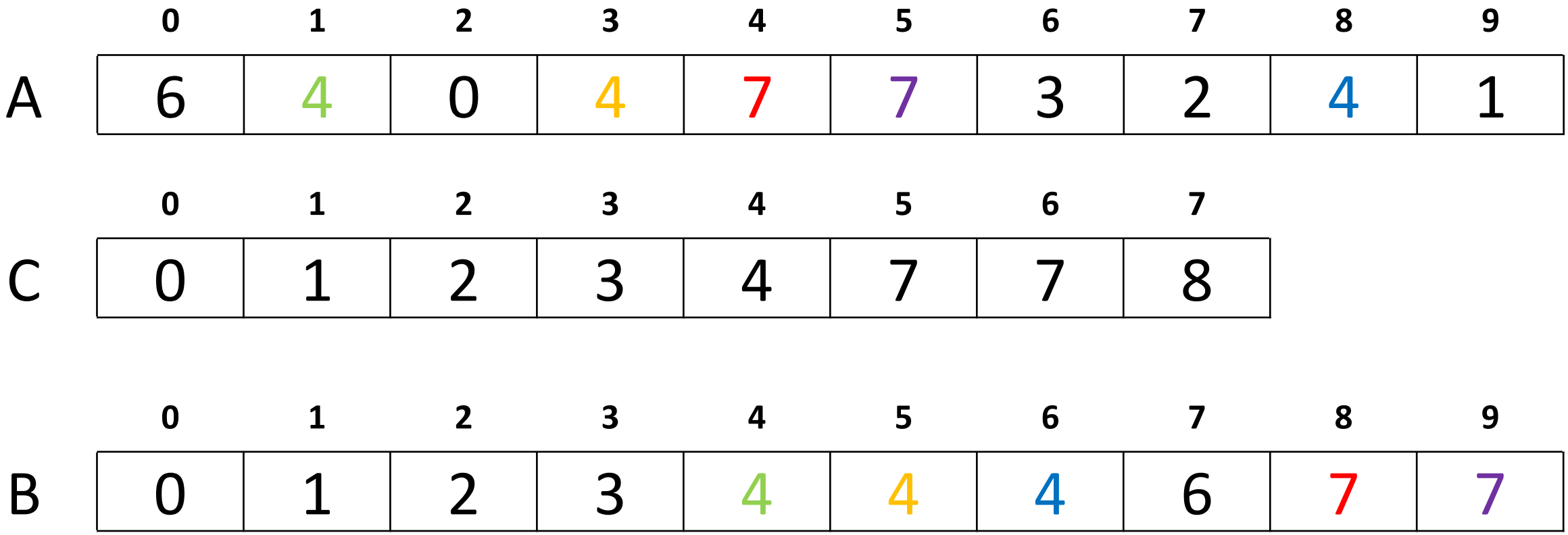
```
for (int j=n-1;j>=0;j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
}
```

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

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

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

Arranjo Ordenado



Propriedades

- O algoritmo é *in-place*?

Propriedades

- O algoritmo é *in-place*?
  - Não – ele precisa de arranjo(s) auxiliar(es) para realizar a ordenação.

Propriedades

- O algoritmo é *in-place*?
  - Não – ele precisa de arranjo(s) auxiliar(es) para realizar a ordenação.

- O algoritmo é estável?

Propriedades

- O algoritmo é *in-place*?
  - Não – ele precisa de arranjo(s) auxiliar(es) para realizar a ordenação.

- O algoritmo é estável?
  - Sim – a ordem relativa dos valores repetidos foi mantida.

Complexidade Assintótica

Counting Sort não ordena utilizando comparações entre elementos do arranjo.

Assim, calcularemos seu custo com base no número de atribuições de valores nos arranjos criados.

Complexidade Assintótica

```
int[] countingSort(int A[], int k){
      int n = A.length;
      int[] B = new int[n];
      int[] C = new int[k];
      for (int i=0;i<k;i++) C[i] = 0;
      for (int j=0;j<n;j++) C[A[j]]++;
      for (int i=1;i<k;i++) C[i] += C[i-1];
      for (int j=n-1;j>=0;j--) {
            B[C[A[j]]-1] = A[j];
            C[A[j]]--;
      }
      return B;
}
```

Atribuições:
0
0
0
k
n
k-1
n
n
Total:
3*n + 2*k – 1

CountingSort(n) ϵ Θ(n+k)

Complexidade Assintótica

O que significa: **CountingSort(n) ϵ Θ(n+k)** ?

Complexidade Assintótica

O que significa: **CountingSort(n) ϵ Θ(n+k)** ?

Significa que a complexidade não é dada simplesmente pela quantidade de números (n), mas também pelo valor desses números.

Assim, CountingSort(n) ϵ Θ(n) **se e somente se k ϵ O(n)**.

Complexidade Assintótica

Como foi dito, se k ϵ O(n) então CountingSort(n) ϵ Θ(n).

Porém, vimos que não é possível ordenar um conjunto de n elementos usando comparações com complexidade inferior n*log n. Isto é alguma inconsistência?

Complexidade Assintótica

Como foi dito, se k ϵ O(n) então CountingSort(n) ϵ Θ(n).

Porém, vimos que não é possível ordenar um conjunto de n elementos usando comparações com complexidade inferior n*log n. Isto é alguma inconsistência?

Não, pois não é possível ordenar **usando comparações** com complexidade inferior a n*log n, porém o Couting Sort ordena **sem usar comparações** entre os valores dos elementos do arranjo.