Aula 22 – CountingSort e RadixSort

Norton T. Roman & Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2023

Ordenação

- Vimos até agora algoritmos baseados apenas em comparações entre os elementos da entrada

Ordenação

- Vimos até agora algoritmos baseados apenas em comparações entre os elementos da entrada
  - Os chamados algoritmos de ordenação por comparação

Ordenação

- Vimos até agora algoritmos baseados apenas em comparações entre os elementos da entrada
  - Os chamados algoritmos de ordenação por comparação
- Vimos também que esse tipo de algoritmo tem limite $\Omega(n \log n)$

Ordenação

- Vimos até agora algoritmos baseados apenas em comparações entre os elementos da entrada
  - Os chamados algoritmos de ordenação por comparação
- Vimos também que esse tipo de algoritmo tem limite $\Omega(n \log n)$
- Seriam esses os únicos tipos possíveis?

Ordenação

- Vimos até agora algoritmos baseados apenas em comparações entre os elementos da entrada
  - Os chamados algoritmos de ordenação por comparação
- Vimos também que esse tipo de algoritmo tem limite $\Omega(n \log n)$
- Seriam esses os únicos tipos possíveis?
  - Veremos que não

Ordenação

- Vimos até agora algoritmos baseados apenas em comparações entre os elementos da entrada
  - Os chamados algoritmos de ordenação por comparação
- Vimos também que esse tipo de algoritmo tem limite $\Omega(n \log n)$
- Seriam esses os únicos tipos possíveis?
  - Veremos que não
- Se conseguirmos ter alguma informação que nos ajude, podemos fazer algoritmos melhores

Counting Sort

- Pressuposto:

Counting Sort

- Pressuposto:
  - O Counting Sort assume que cada um dos n elementos da entrada é um inteiro no intervalo de 0 a k, para algum inteiro k

Counting Sort

- Pressuposto:
  - O Counting Sort assume que cada um dos n elementos da entrada é um inteiro no intervalo de 0 a k, para algum inteiro k
- Ideia:

Counting Sort

- Pressuposto:
  - O Counting Sort assume que cada um dos n elementos da entrada é um inteiro no intervalo de 0 a k, para algum inteiro k
- Ideia:
  - Determinar, para cada elemento de entrada x, o número de elementos menores que x

Counting Sort

- Pressuposto:
  - O Counting Sort assume que cada um dos n elementos da entrada é um inteiro no intervalo de 0 a k, para algum inteiro k
- Ideia:
  - Determinar, para cada elemento de entrada x, o número de elementos menores que x
  - Usar essa informação para colocar x diretamente em sua posição no arranjo de saída

Counting Sort

- Pressuposto:
  - O Counting Sort assume que cada um dos n elementos da entrada é um inteiro no intervalo de 0 a k, para algum inteiro k
- Ideia:
  - Determinar, para cada elemento de entrada x, o número de elementos menores que x
  - Usar essa informação para colocar x diretamente em sua posição no arranjo de saída
  - Precisamos, no entanto, tomar cuidado para não incluirmos elementos de mesmo valor na mesma posição

Counting Sort

Implementação

Counting Sort

Implementação

- Assumimos que a entrada é um arranjo $A[1..n]$ (ou seja, conhecemos o tamanho de A de antemão)

Counting Sort

Implementação

- Assumimos que a entrada é um arranjo $A[1..n]$ (ou seja, conhecemos o tamanho de A de antemão)
- Precisaremos de 2 outros arranjos:

Counting Sort

Implementação

- Assumimos que a entrada é um arranjo $A[1..n]$ (ou seja, conhecemos o tamanho de A de antemão)
- Precisaremos de 2 outros arranjos:
  - $B[1..n]$: guardará a saída ordenada

Counting Sort

Implementação

- Assumimos que a entrada é um arranjo $A[1..n]$ (ou seja, conhecemos o tamanho de A de antemão)
- Precisaremos de 2 outros arranjos:
  - $B[1..n]$: guardará a saída ordenada
  - $C[0..k]$: para armazenar o número de ocorrências de cada elemento i de A $(0 \leq i \leq k)$

Counting Sort

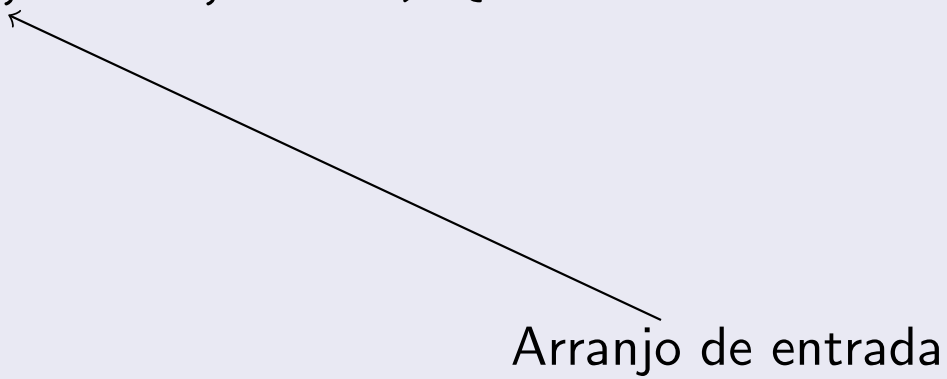
Implementação

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

Counting Sort

Implementação

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



Counting Sort

Implementação

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

Valor do maior elemento em A (ou algum outro limitante)

Counting Sort

Implementação

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

Fazemos `C[i]` conter o número de elementos em `A` que são iguais a `i`

Counting Sort

Implementação

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

Repare que os valores em A vão de 0 a k, assim como os índices de C (e que os índices estão naturalmente ordenados)

Counting Sort

Implementação

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

Agora fazemos C[i] conter o número de elementos em A que são $\leq i$

Counting Sort

Implementação

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

Colocamos cada elemento de A na posição correta em B

Counting Sort

Implementação

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

Cuidamos de não incluir novamente os repetidos

Counting Sort

Implementação

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

O -1 é para mapear a contagem ao índice em B

Counting Sort

Implementação

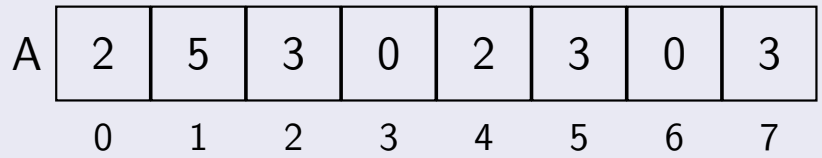
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n; j++)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

Copiamos para A os valores do arranjo ordenado que estão em B

Counting Sort

Implementação

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



Counting Sort

Implementação

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

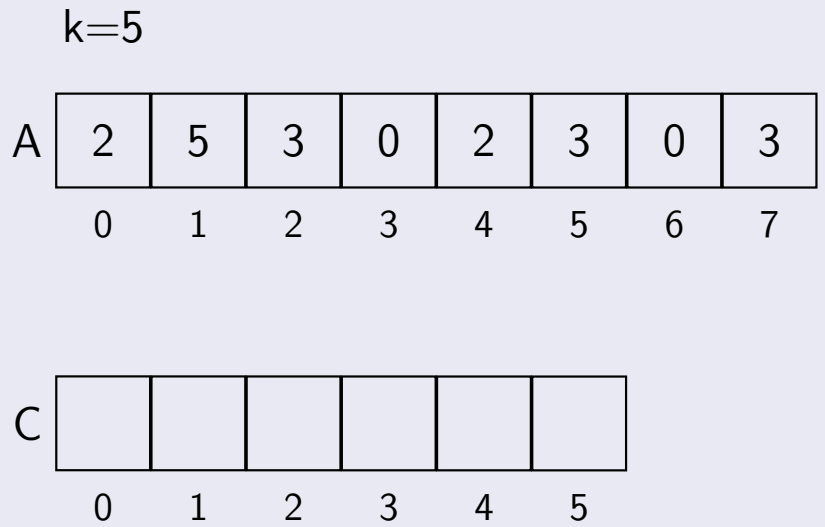
k=5

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

Counting Sort

Implementação

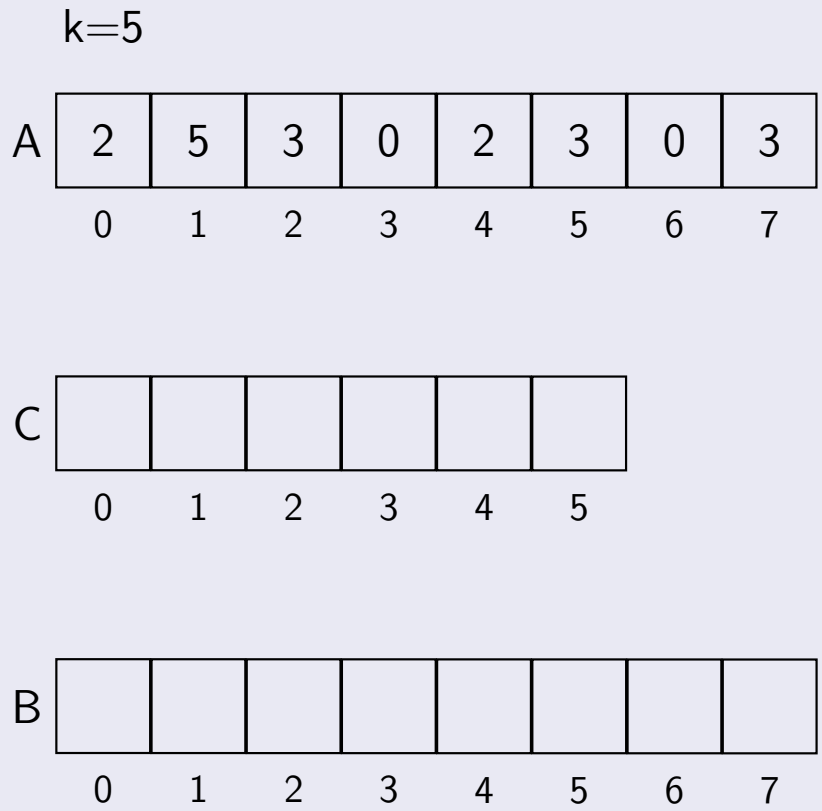
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

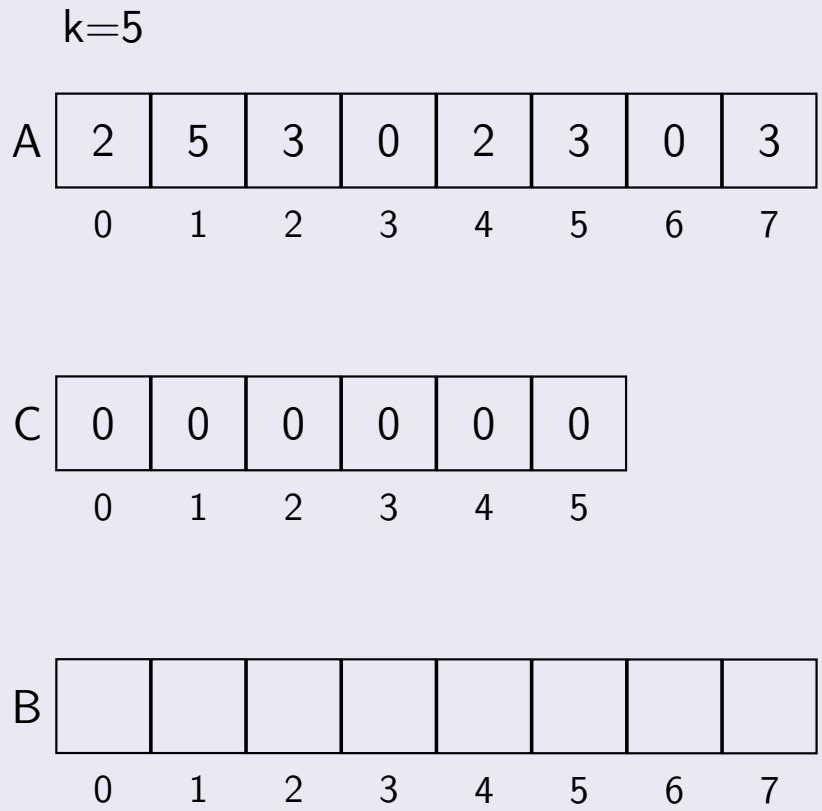
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

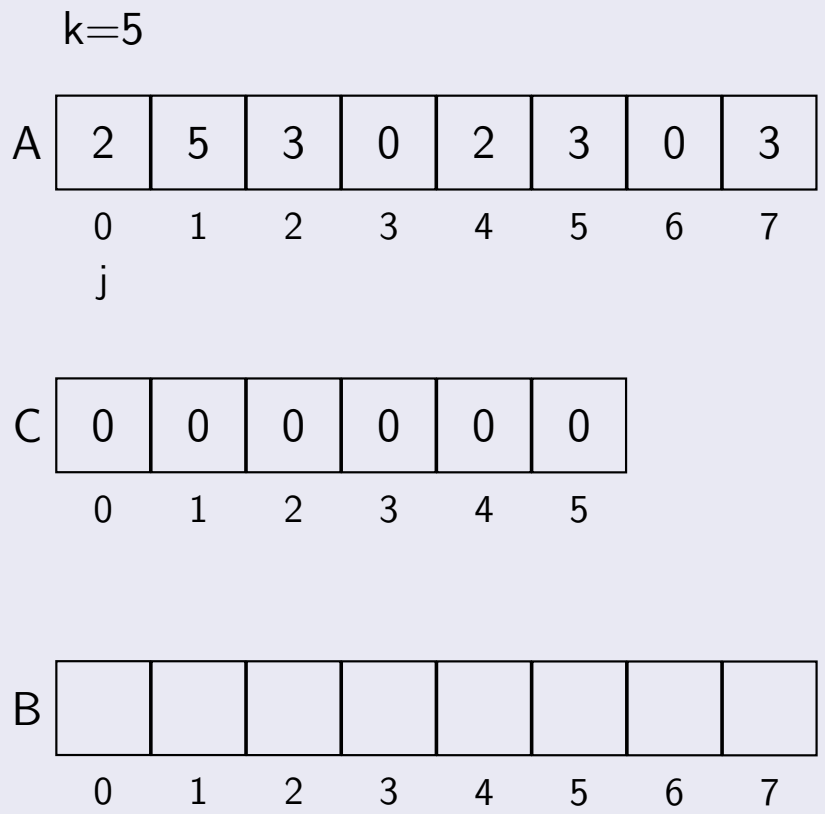
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

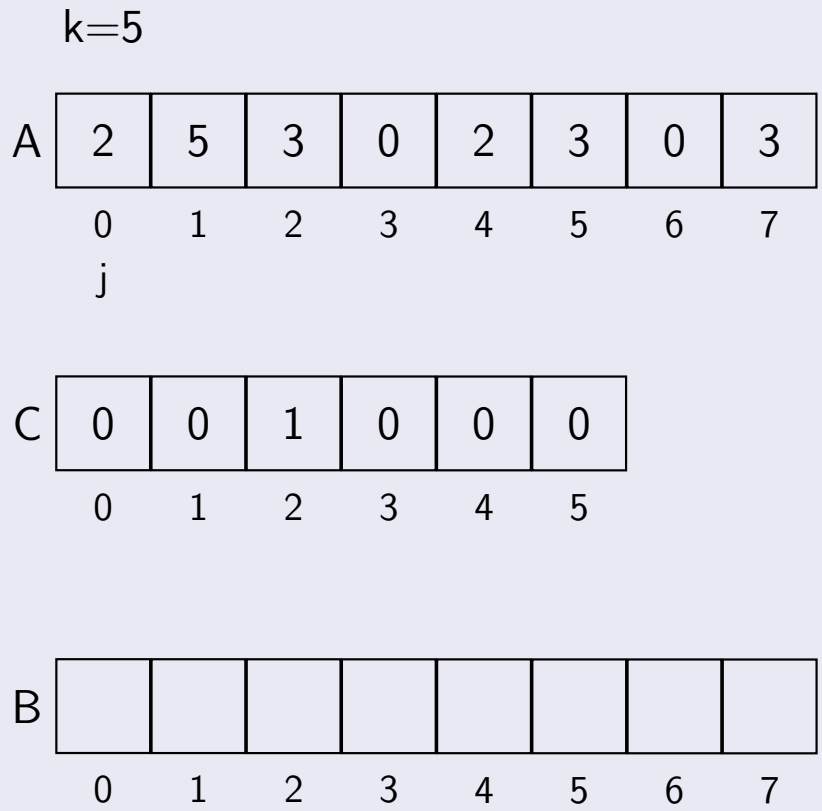
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

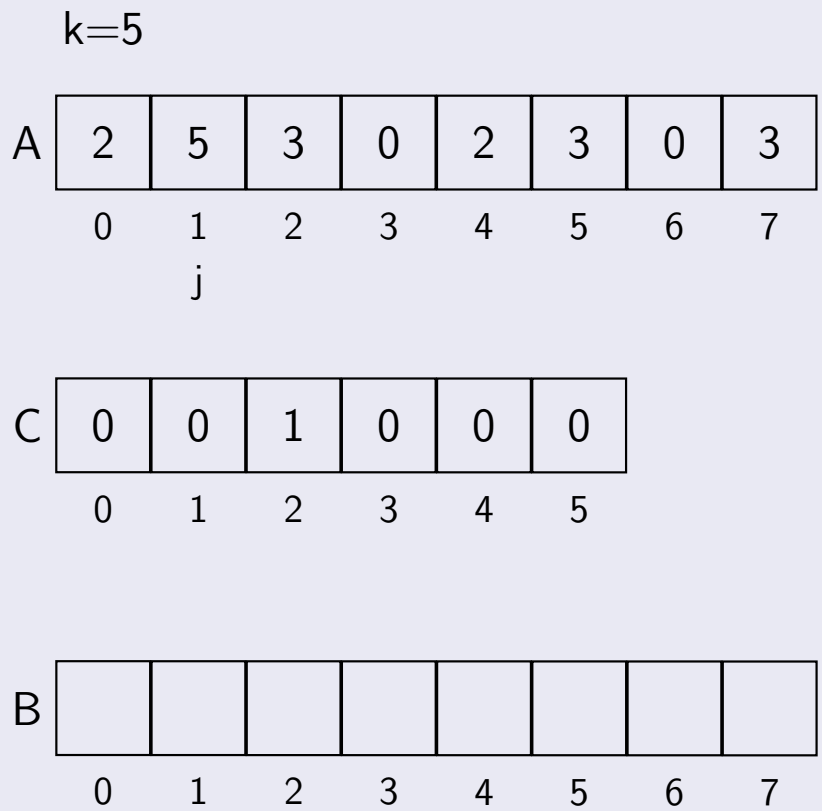
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

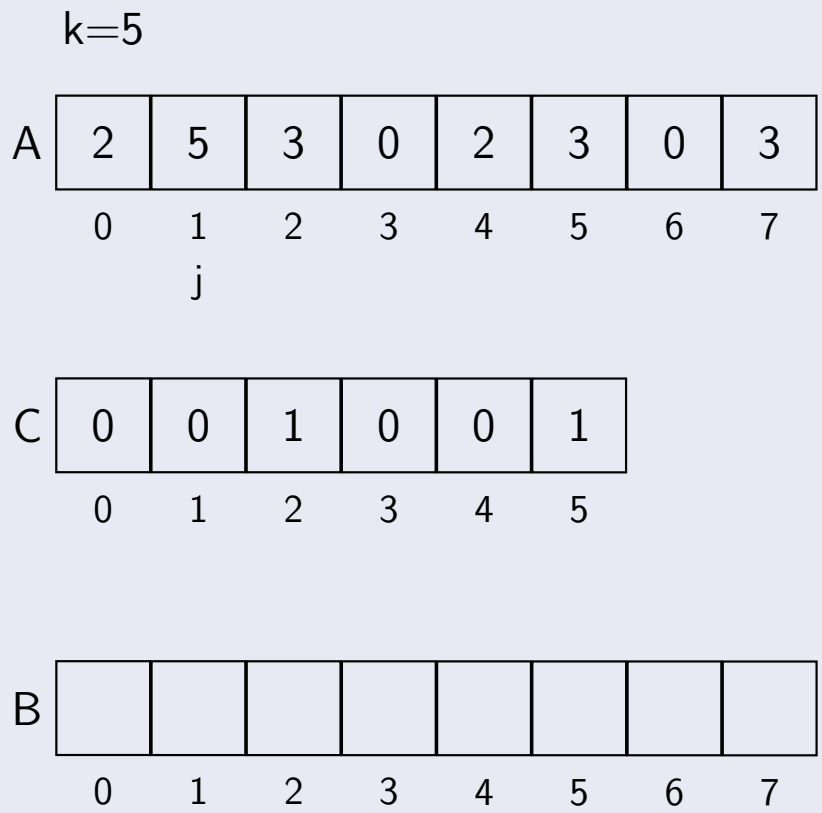
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

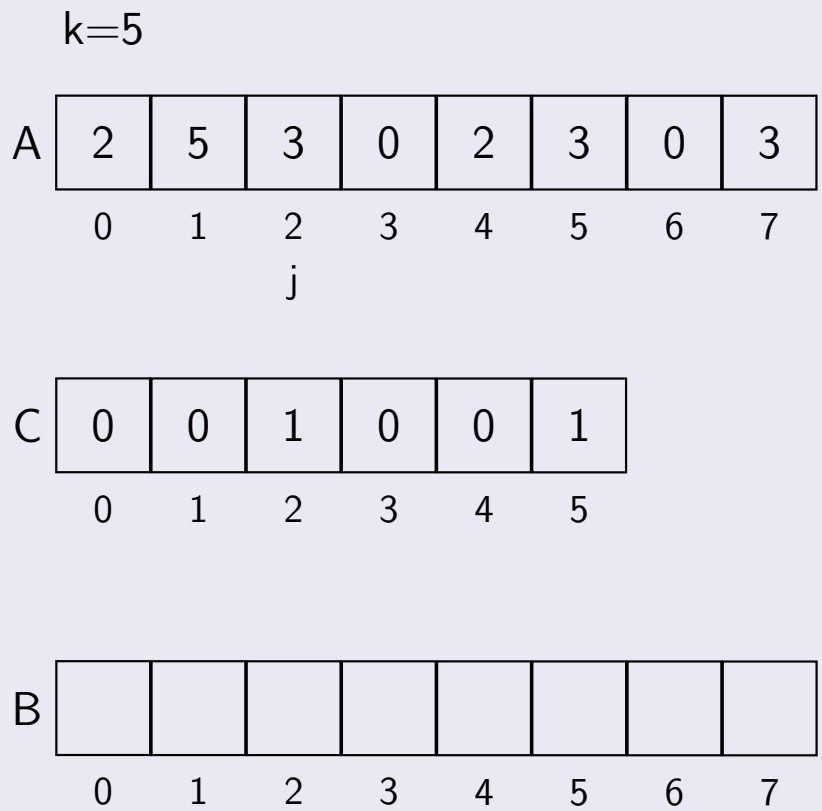
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

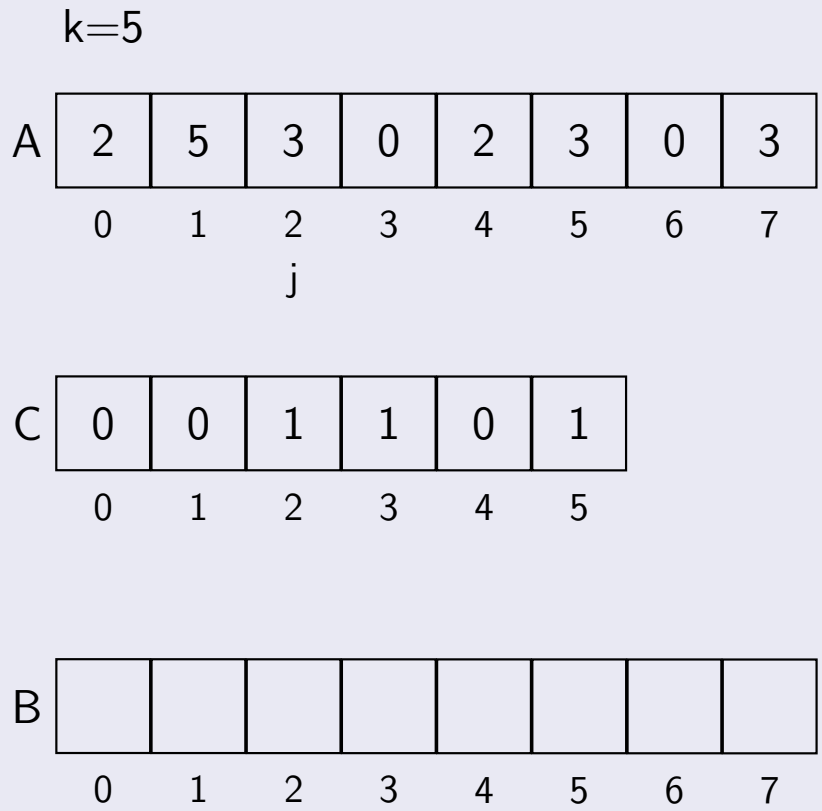
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

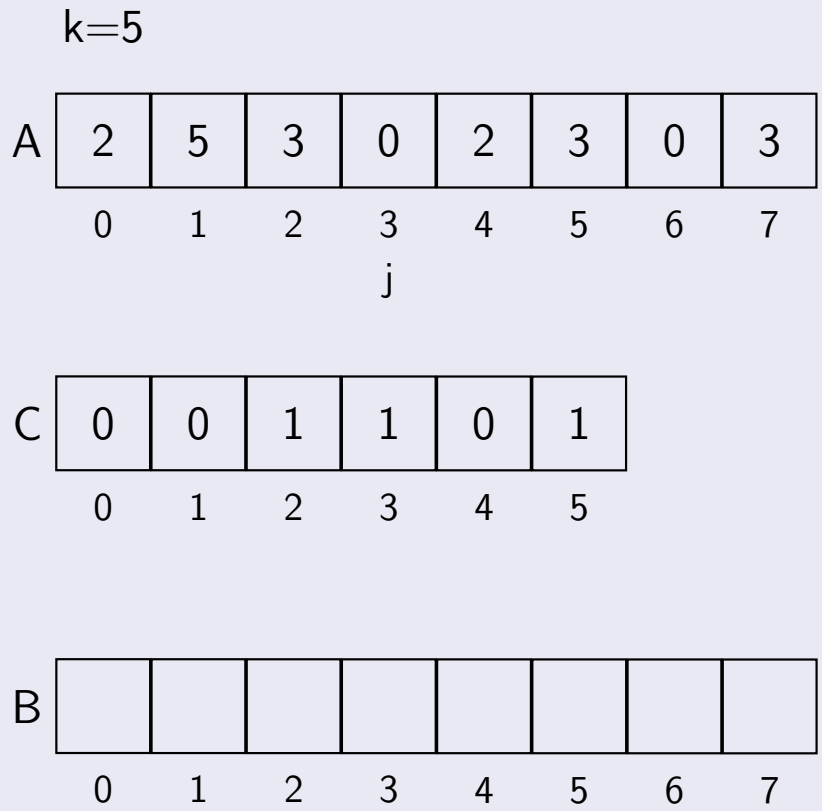
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

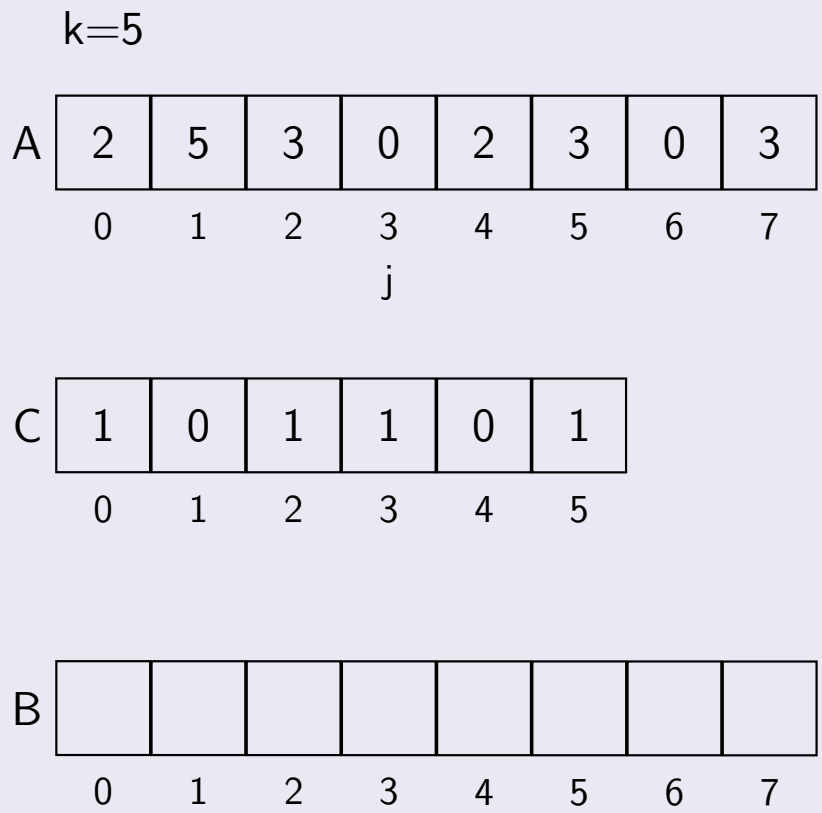
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

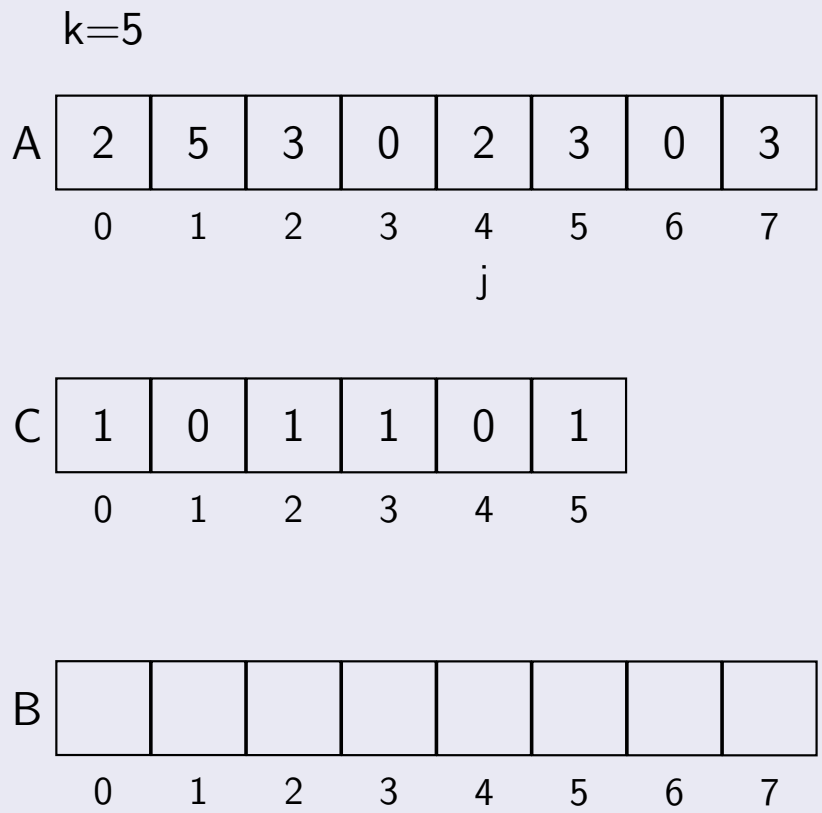
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

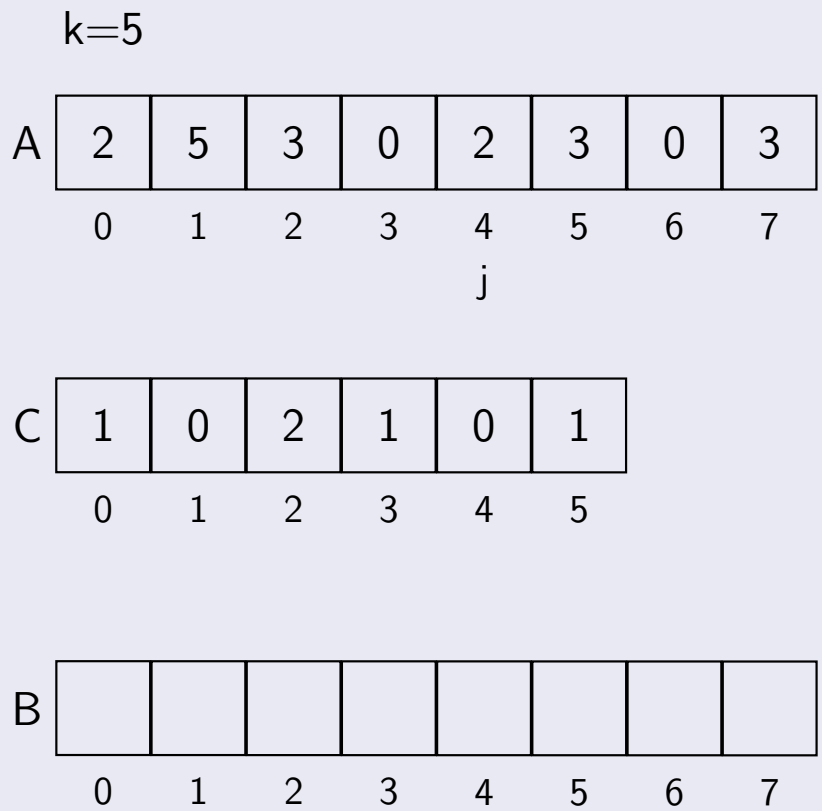
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

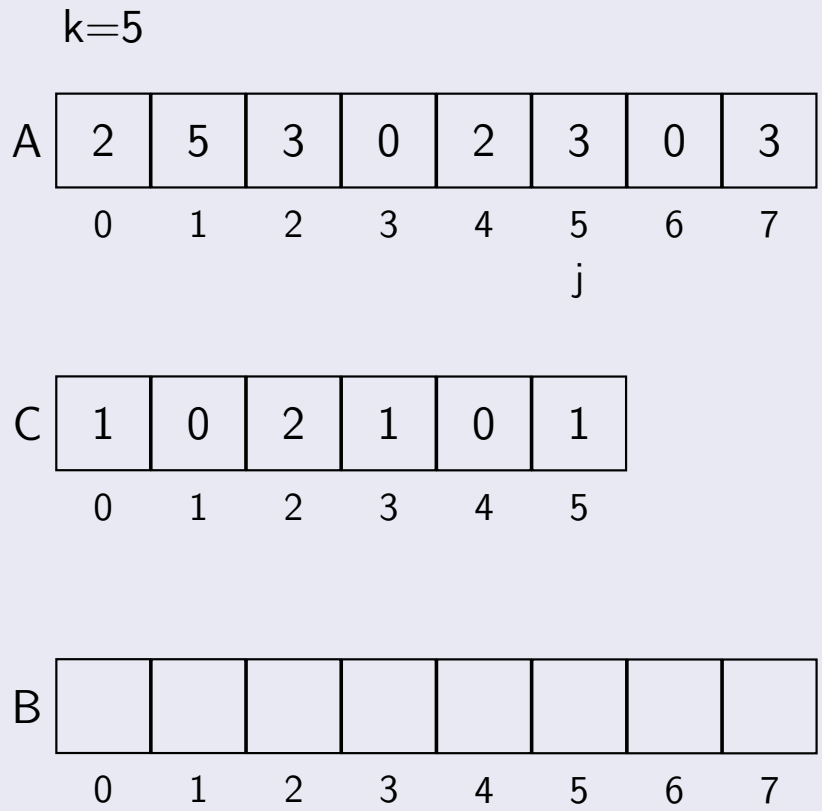
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

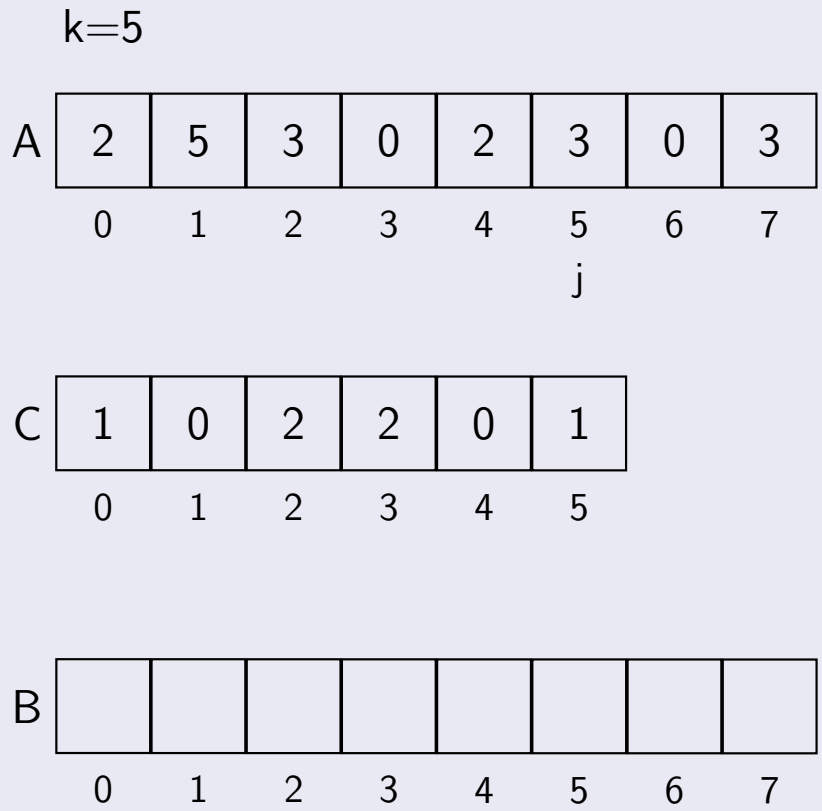
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

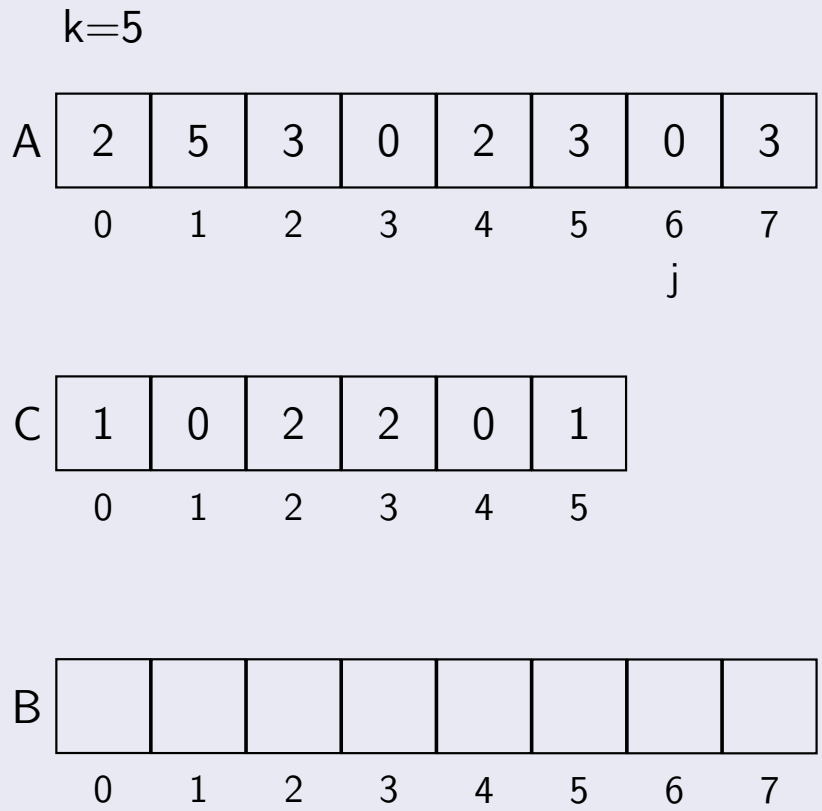
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

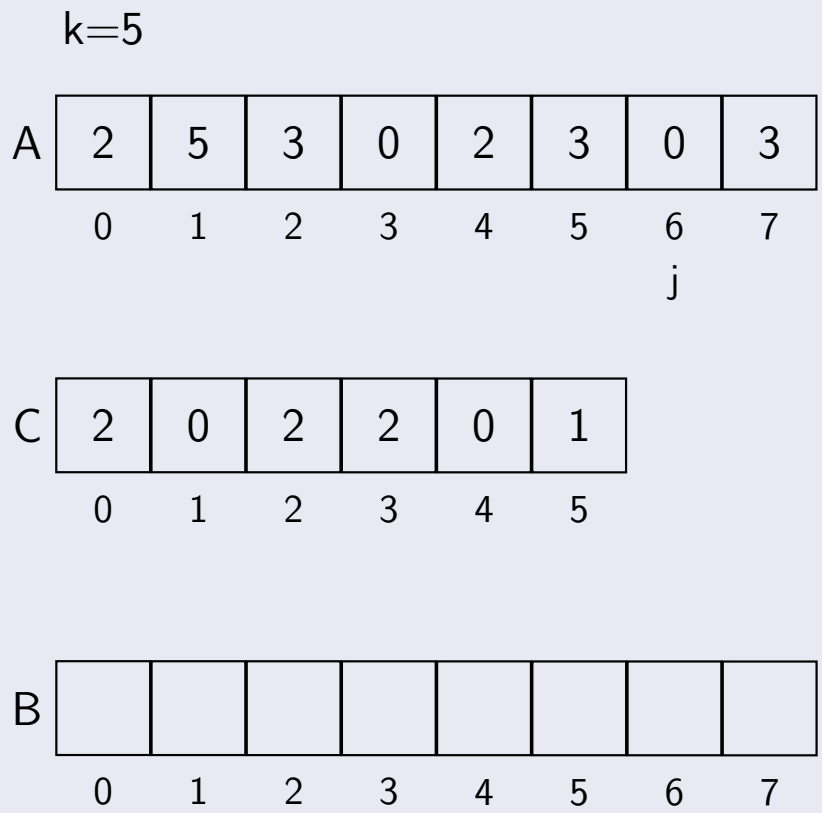
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

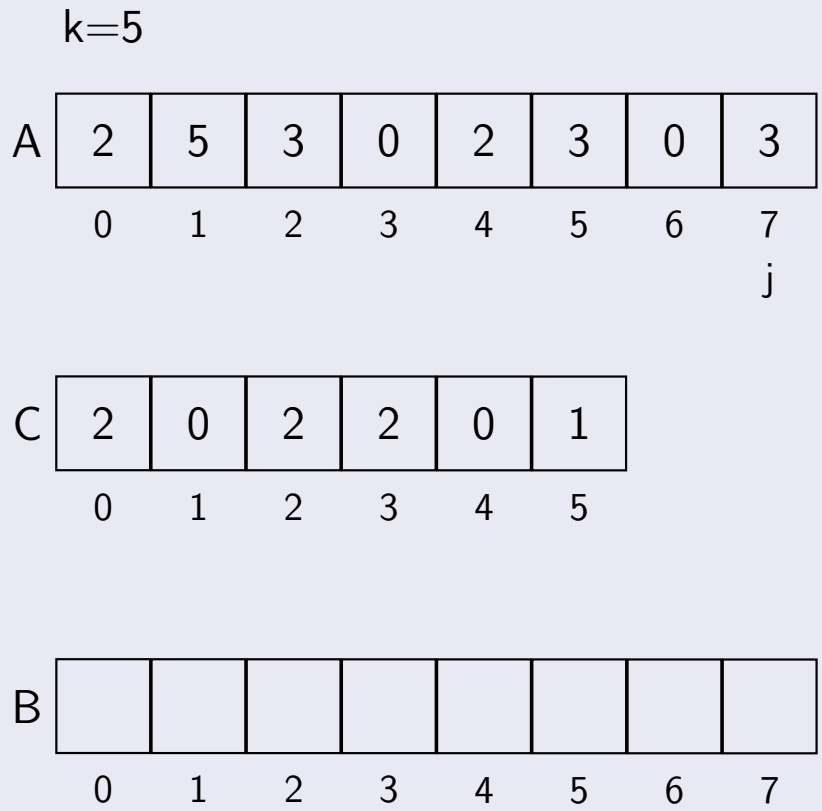
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

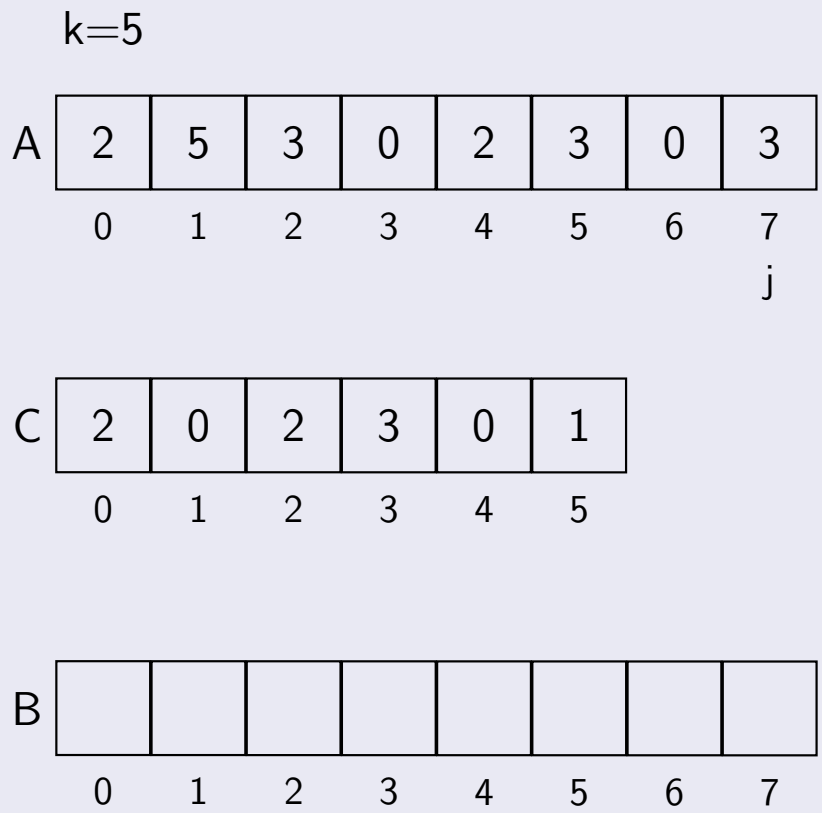
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

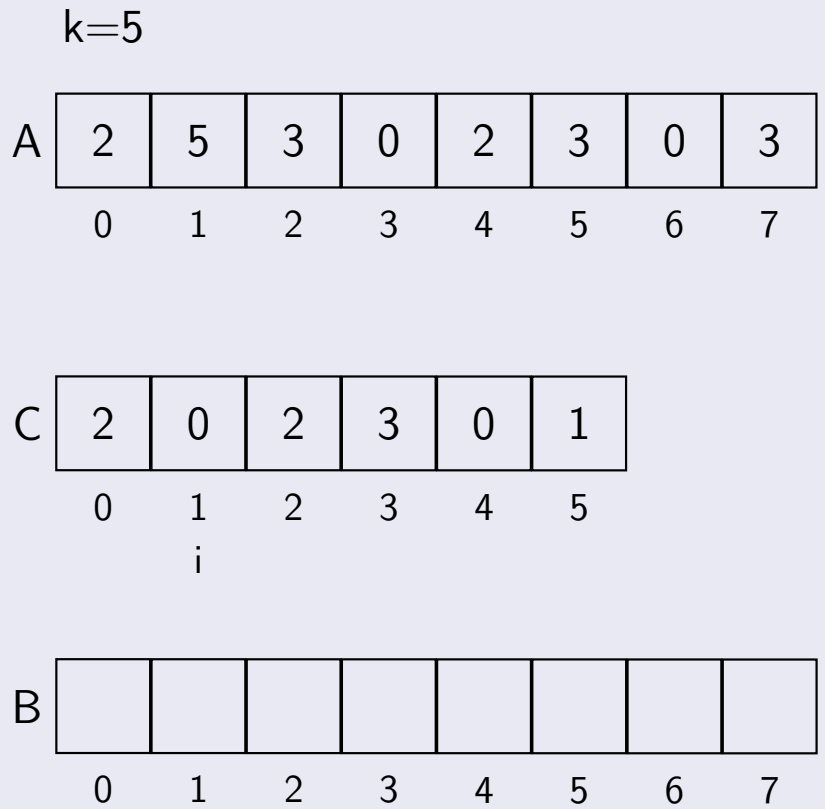
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

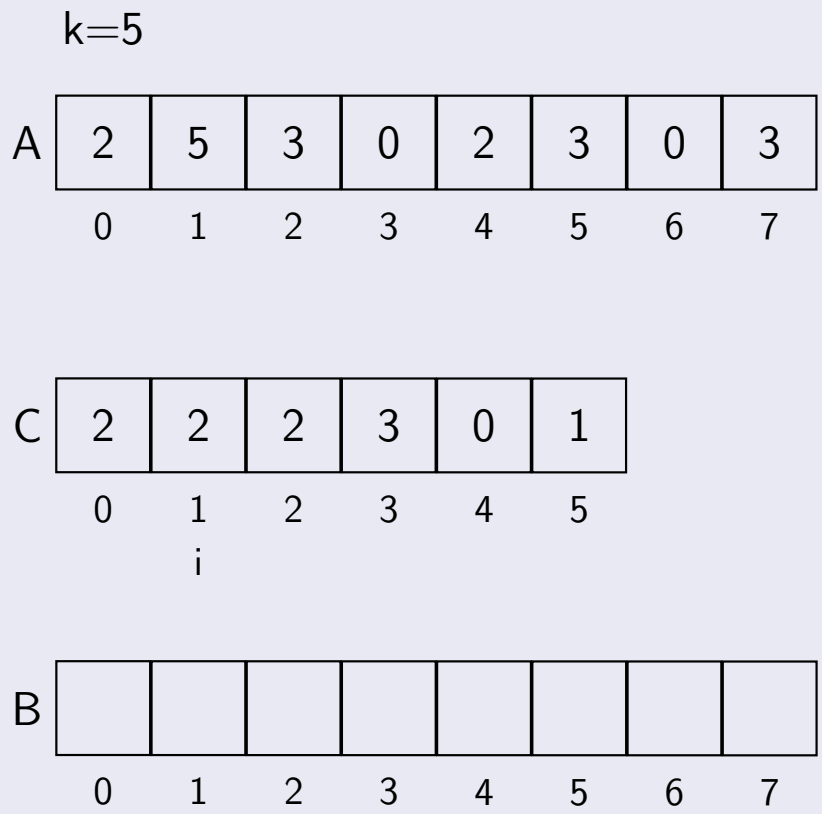
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

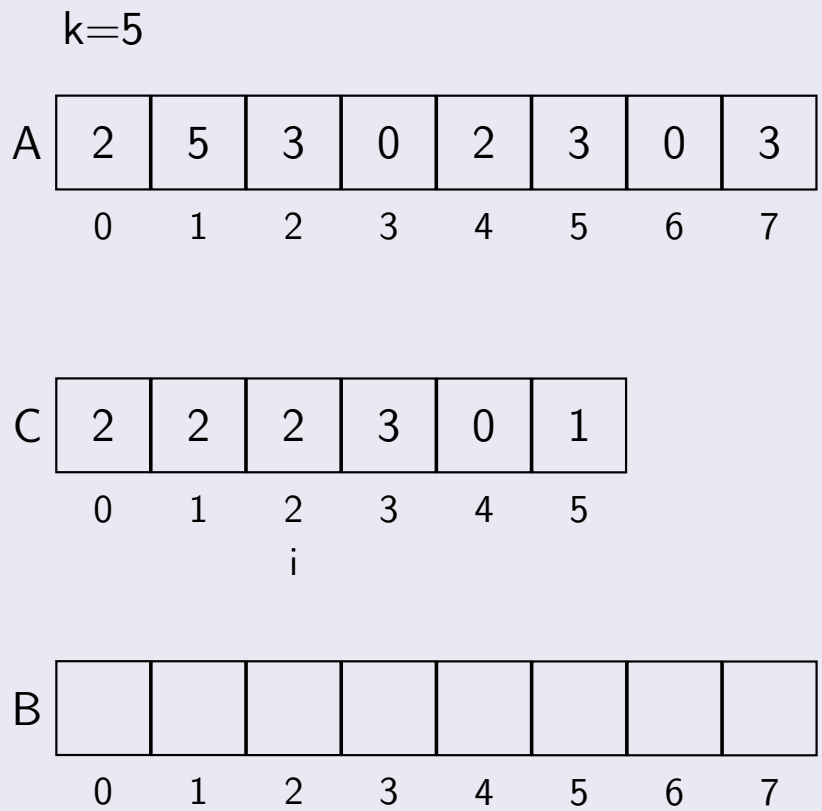
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

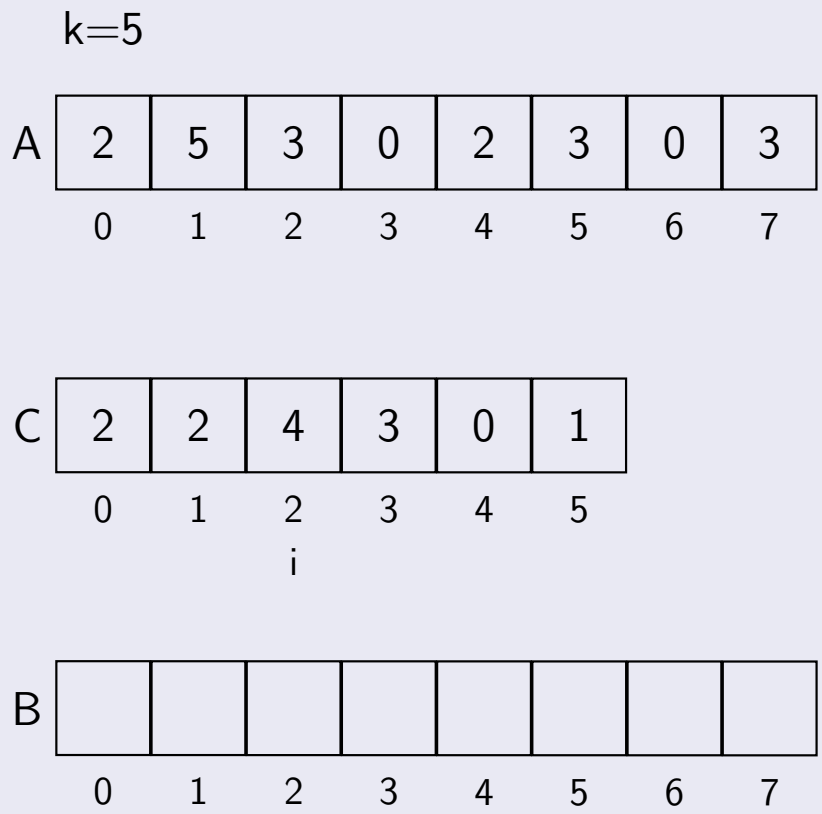
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

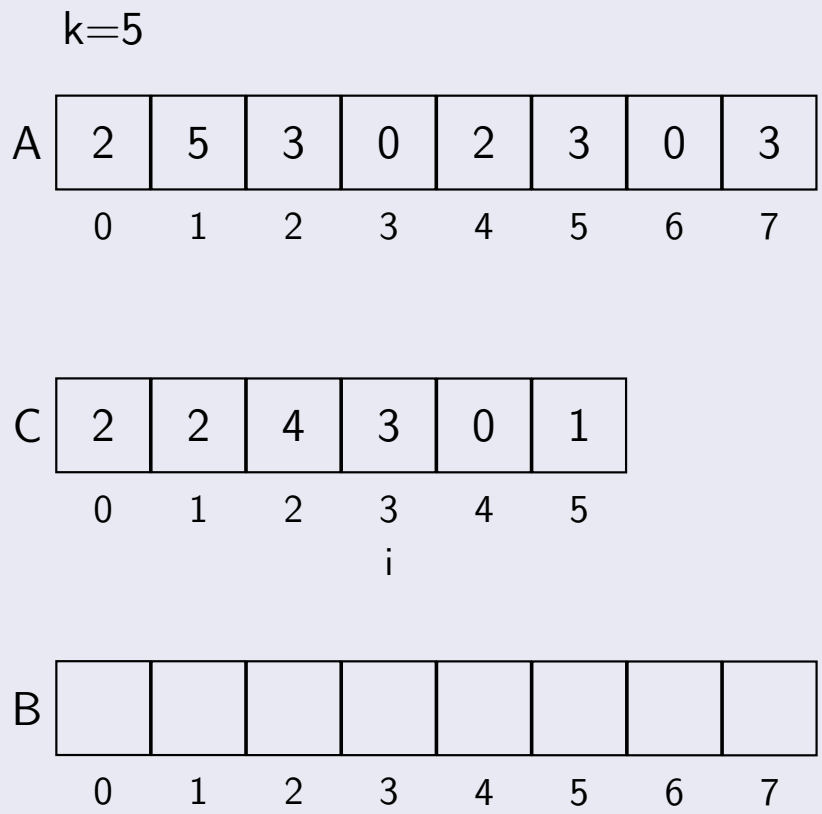
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

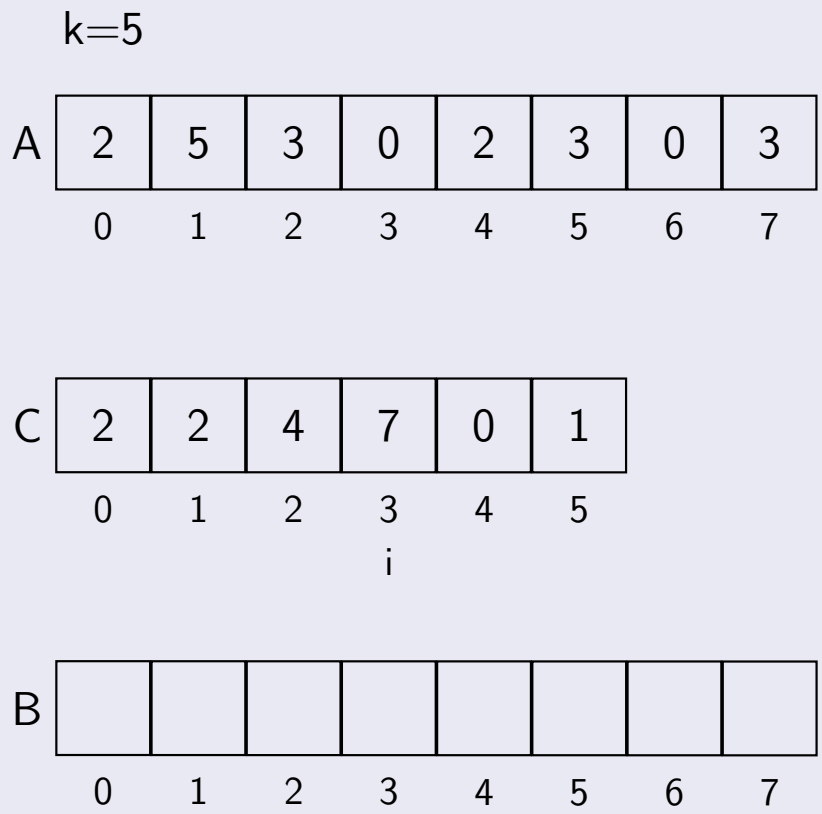
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

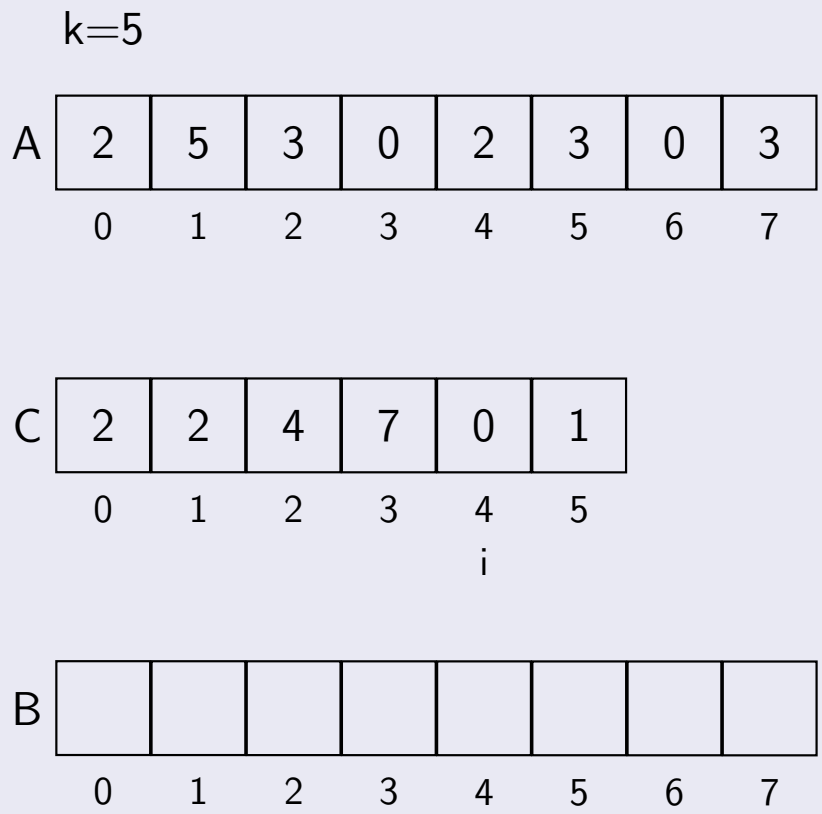
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

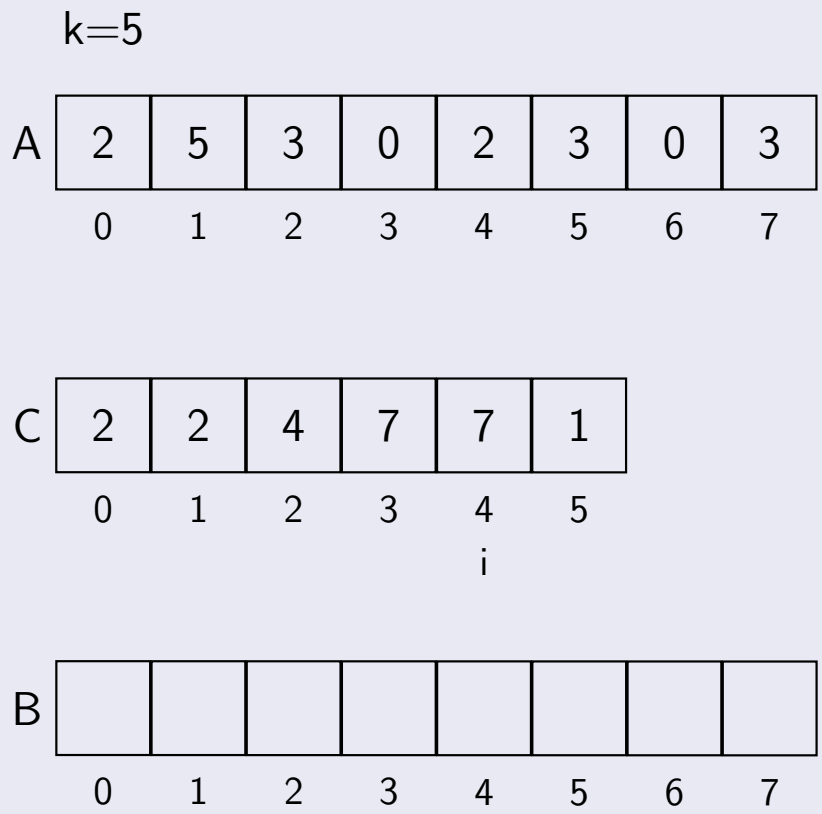
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

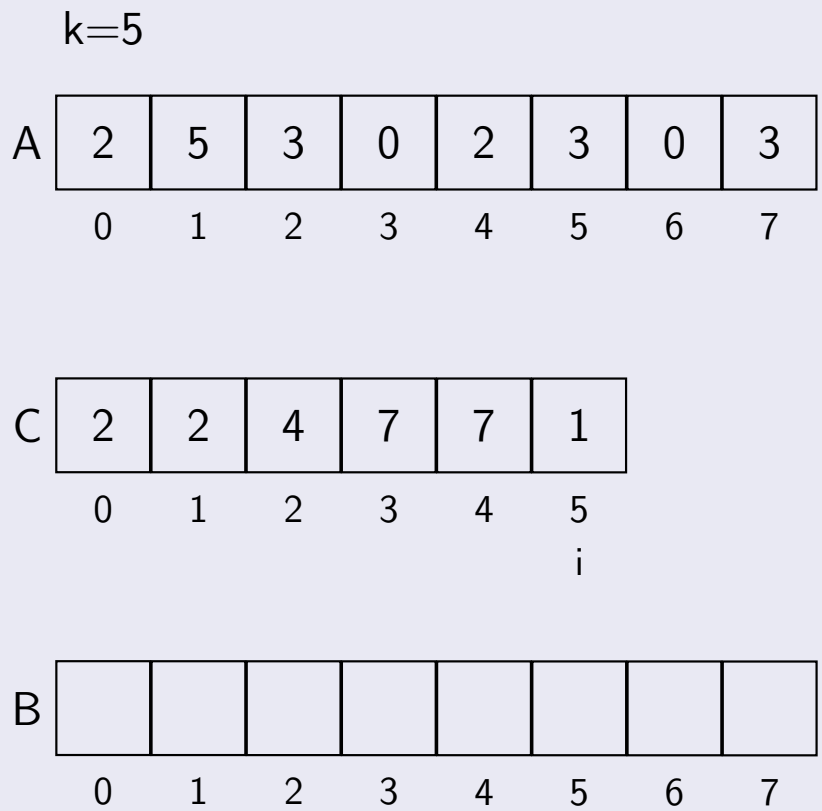
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

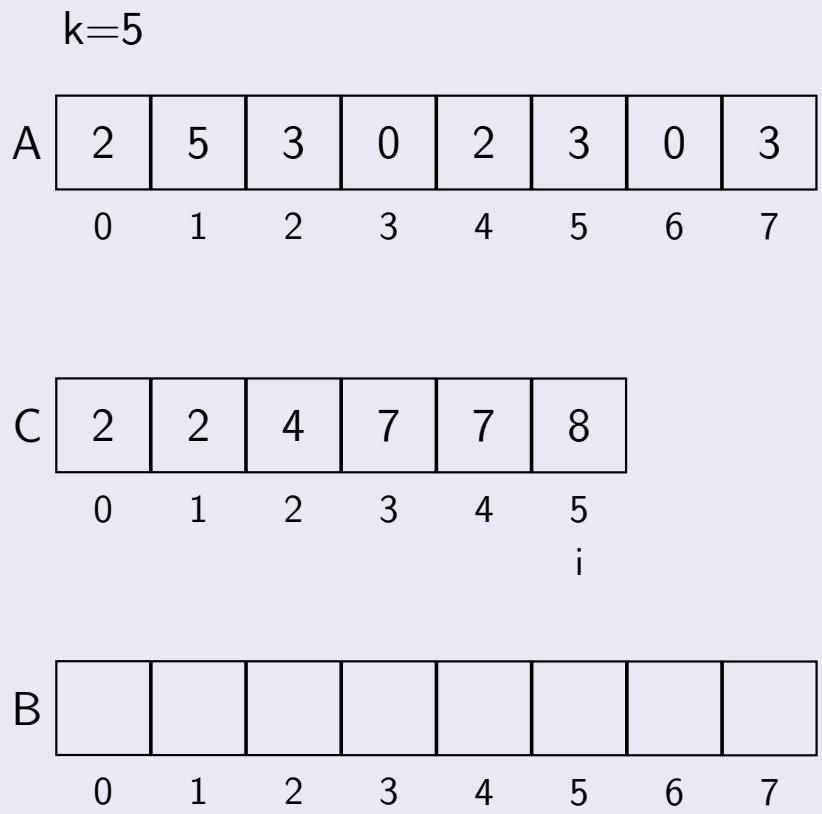
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

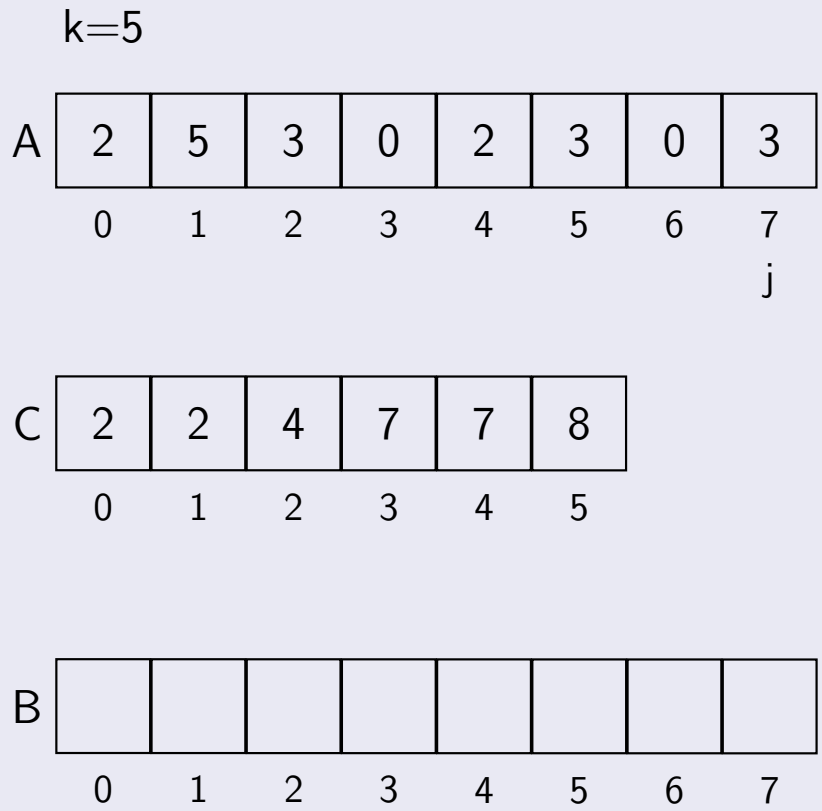
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

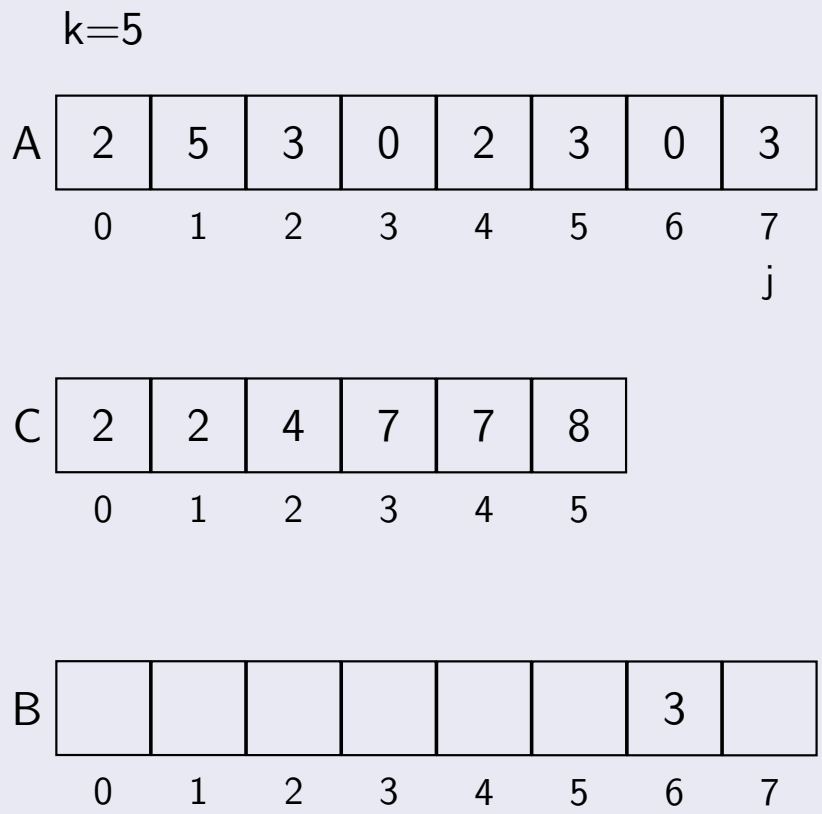
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

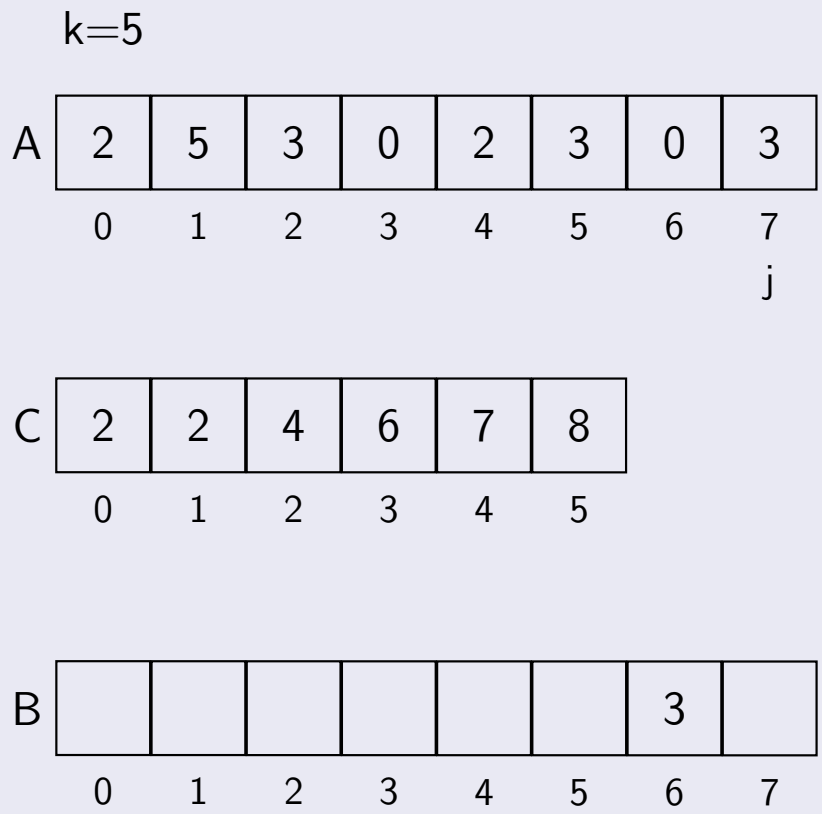
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

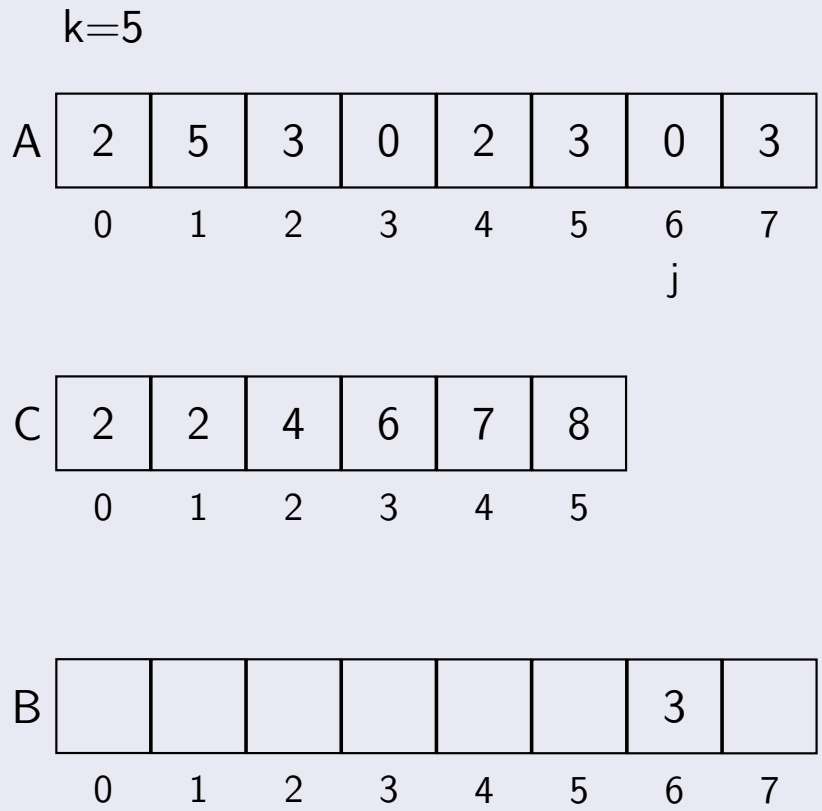
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

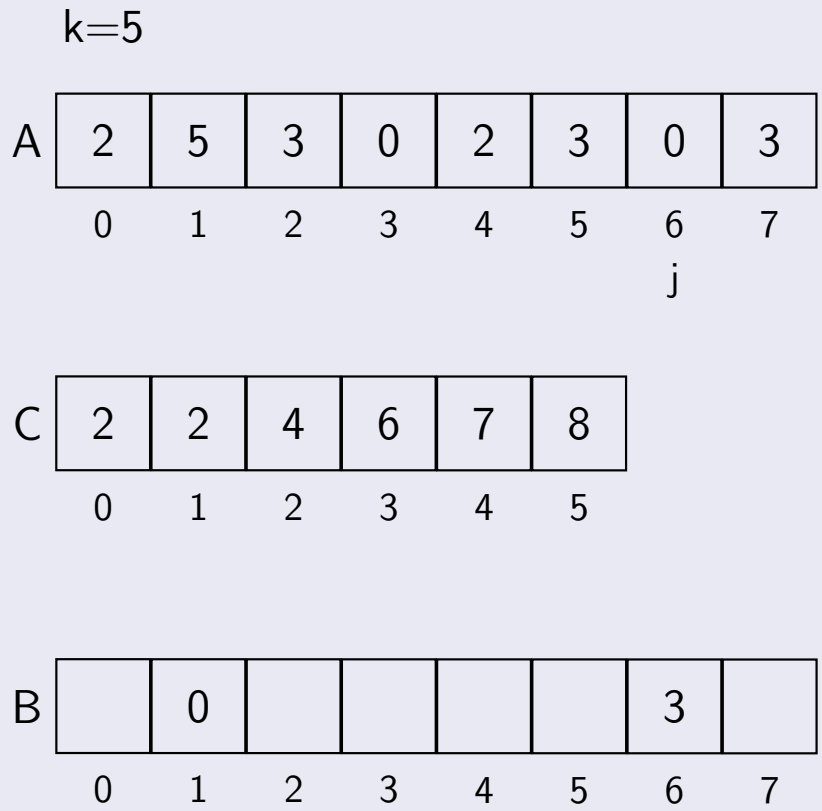
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

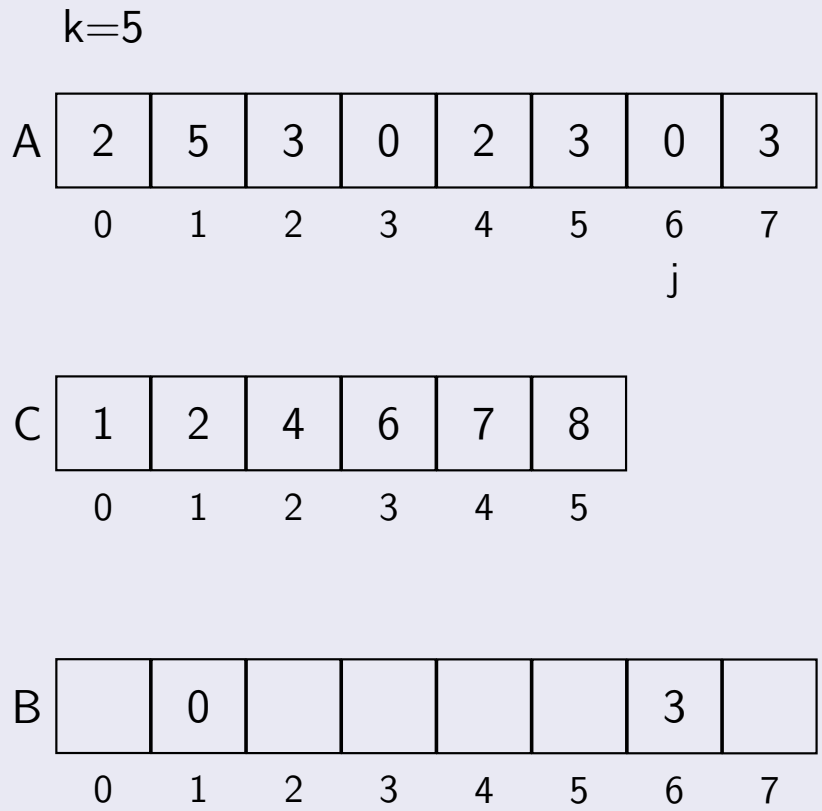
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

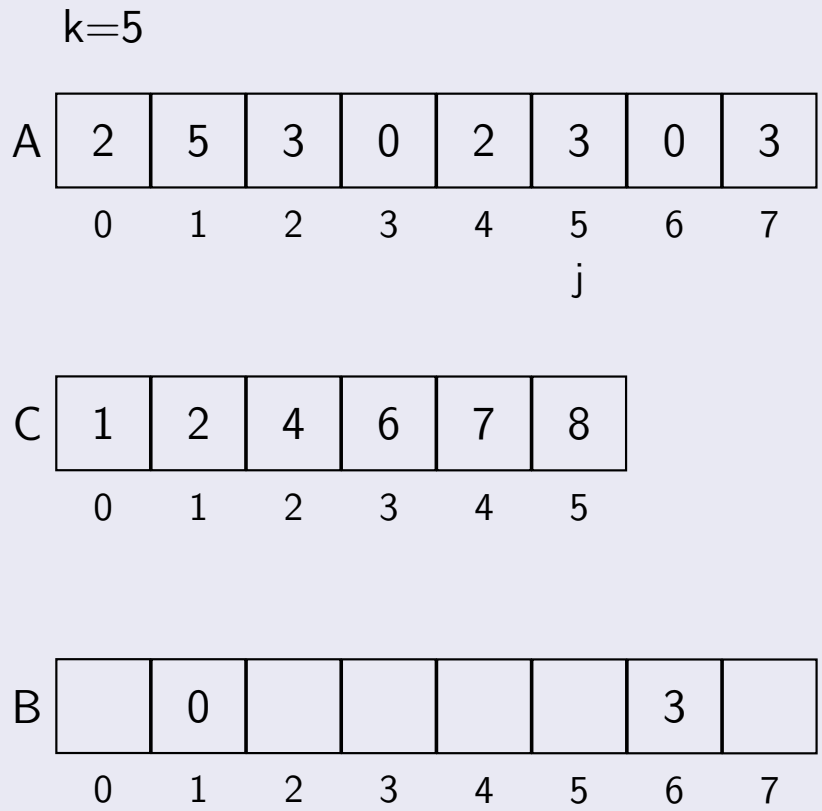
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

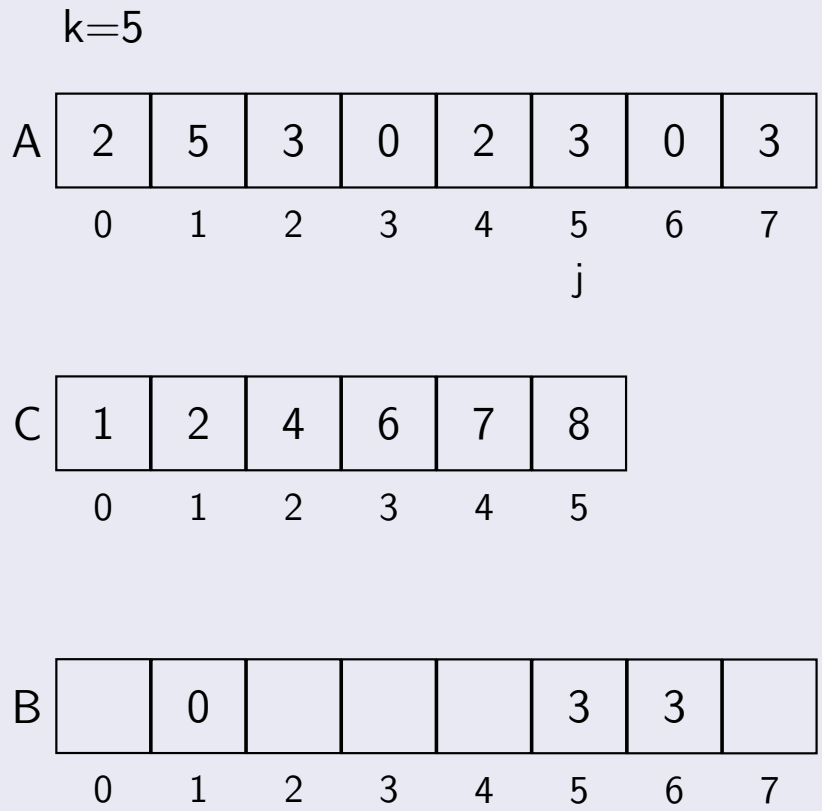
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

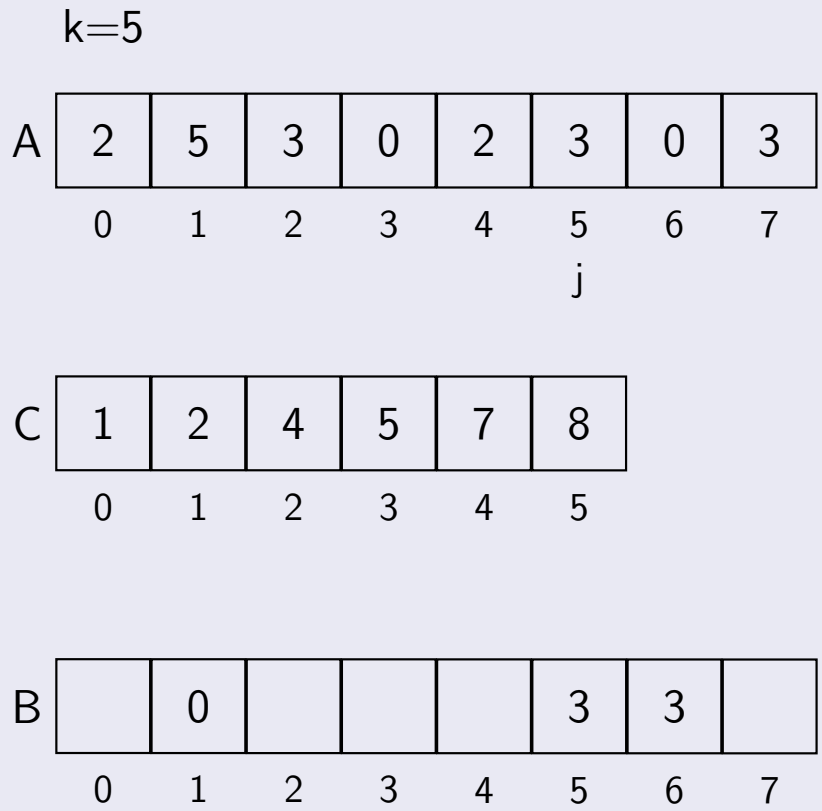
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

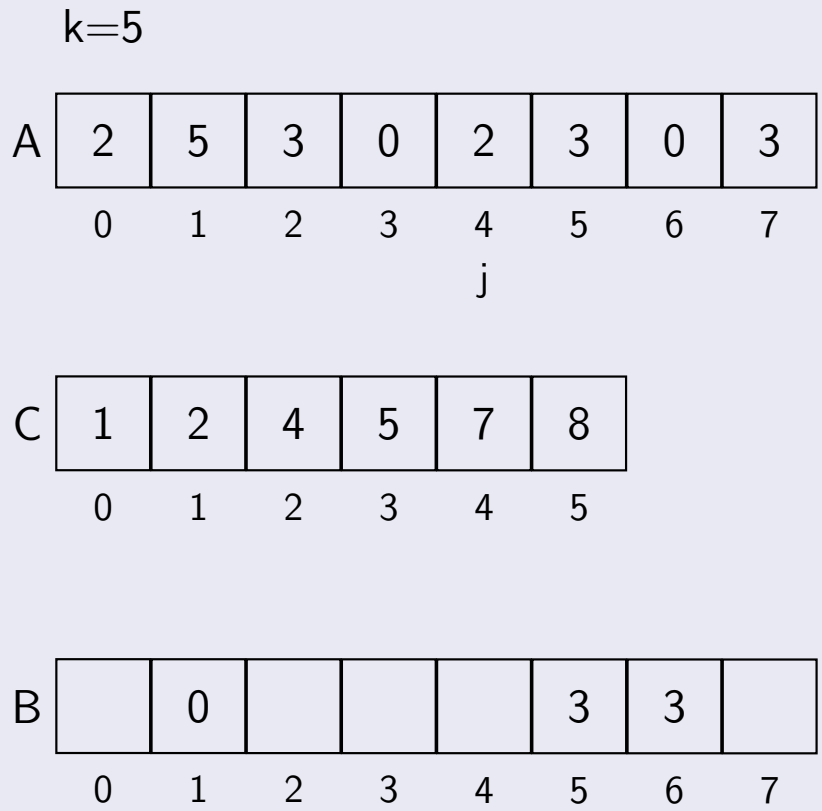
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

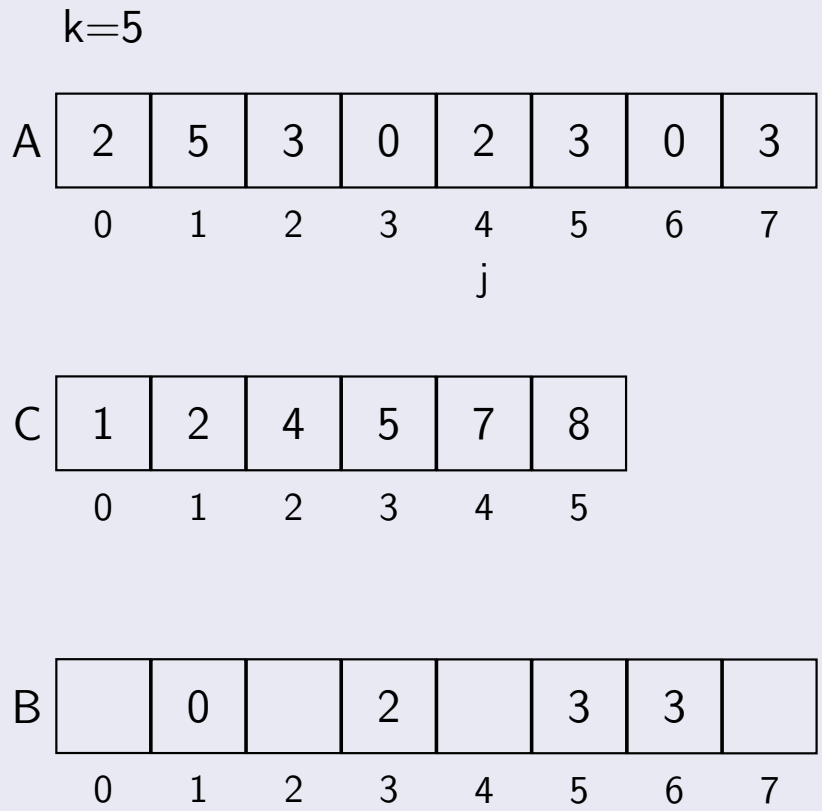
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

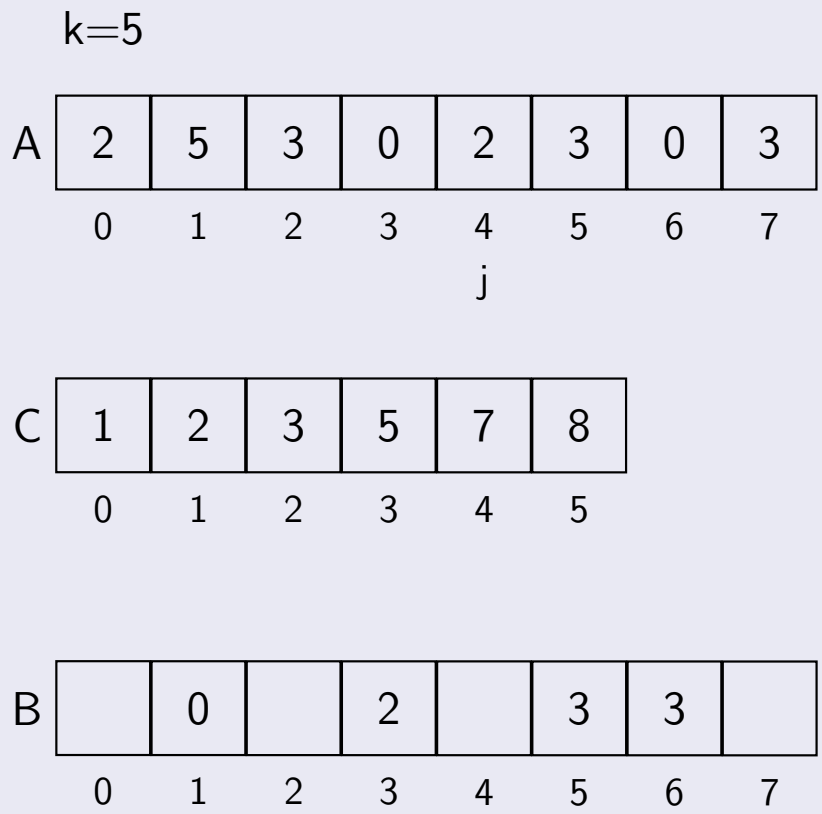
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

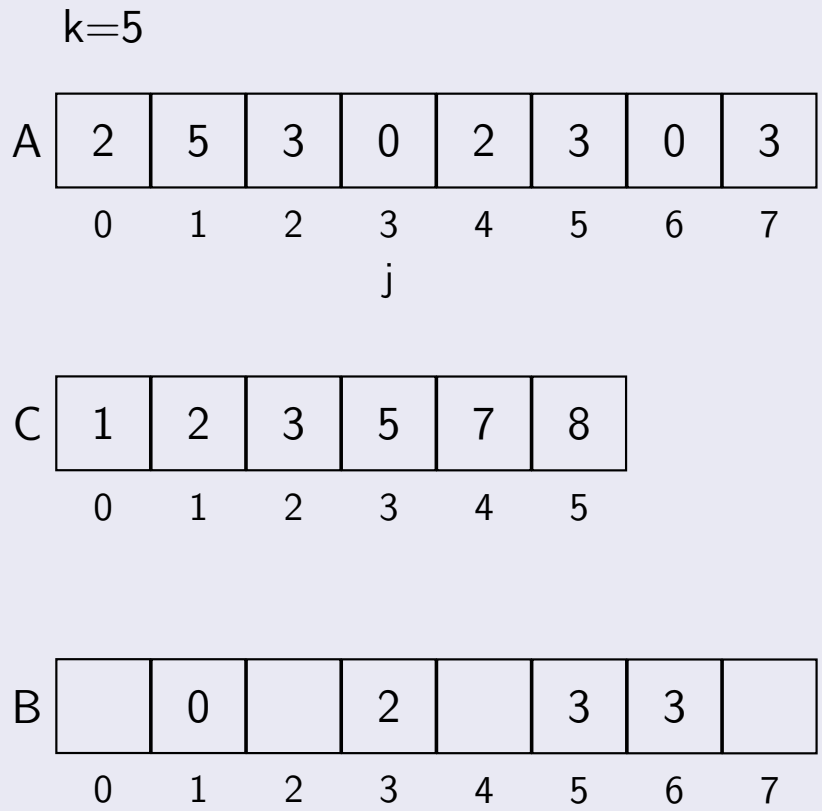
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

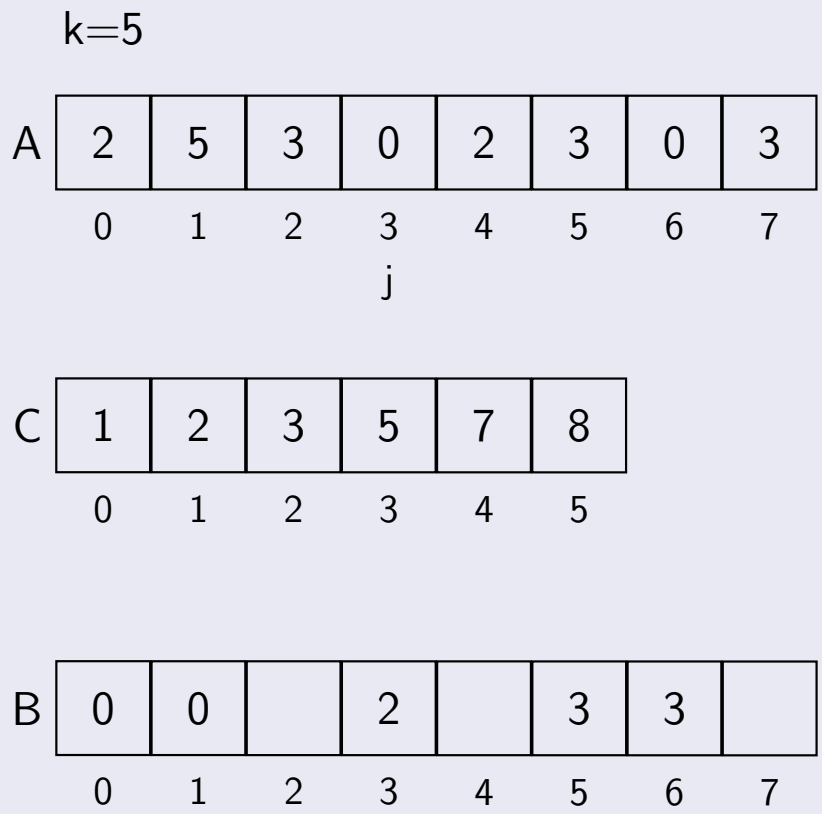
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

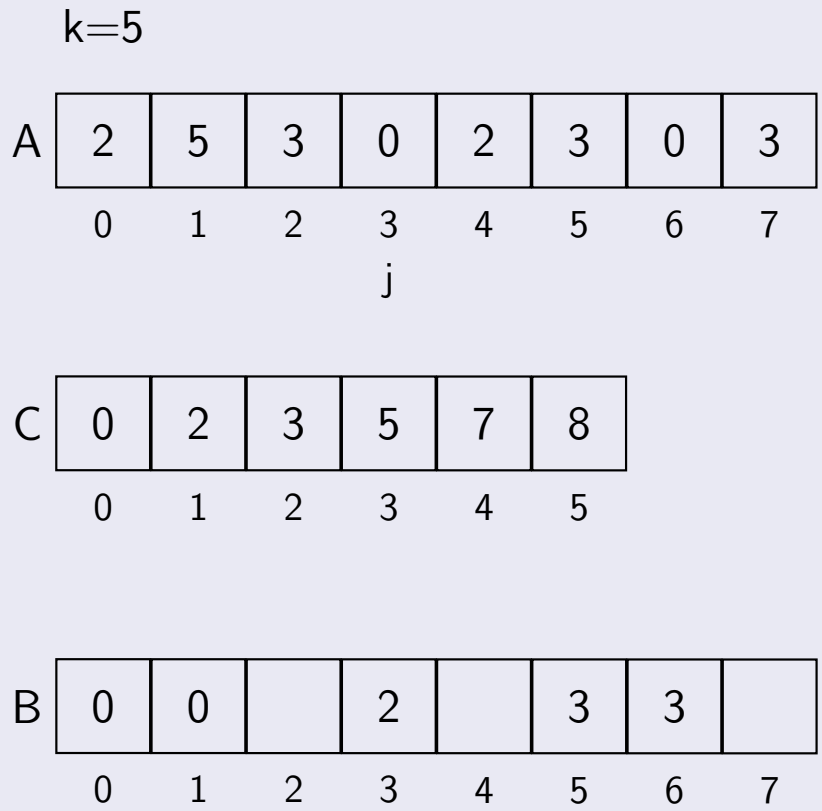
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

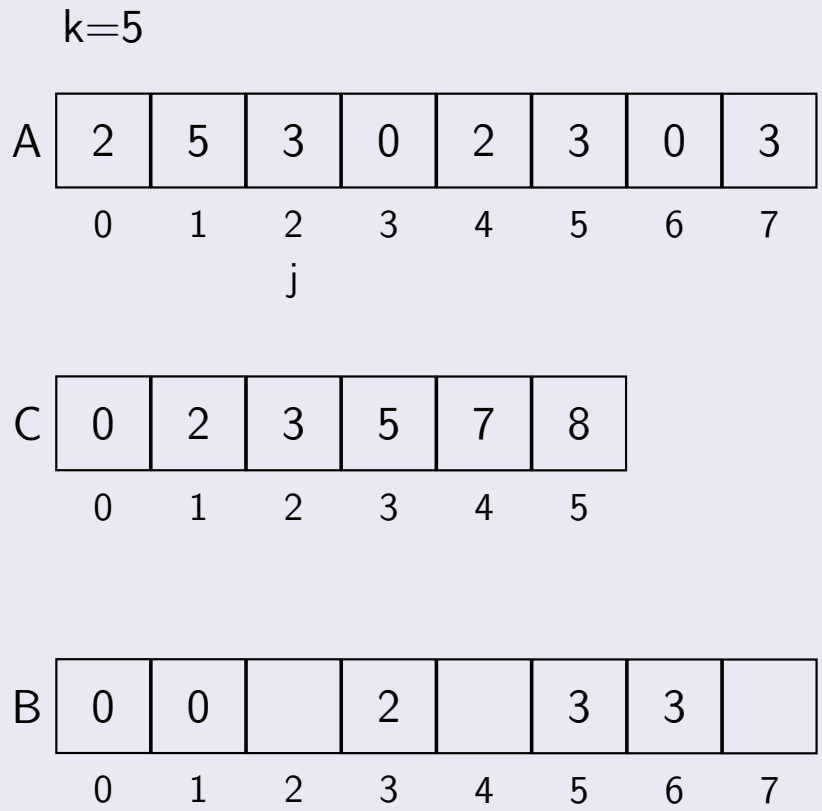
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

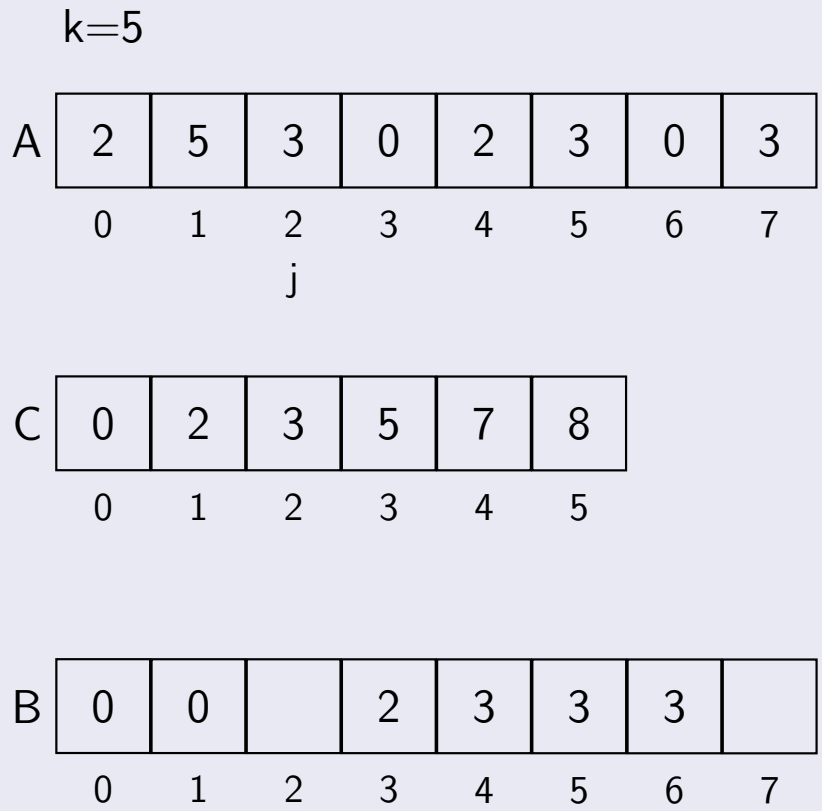
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

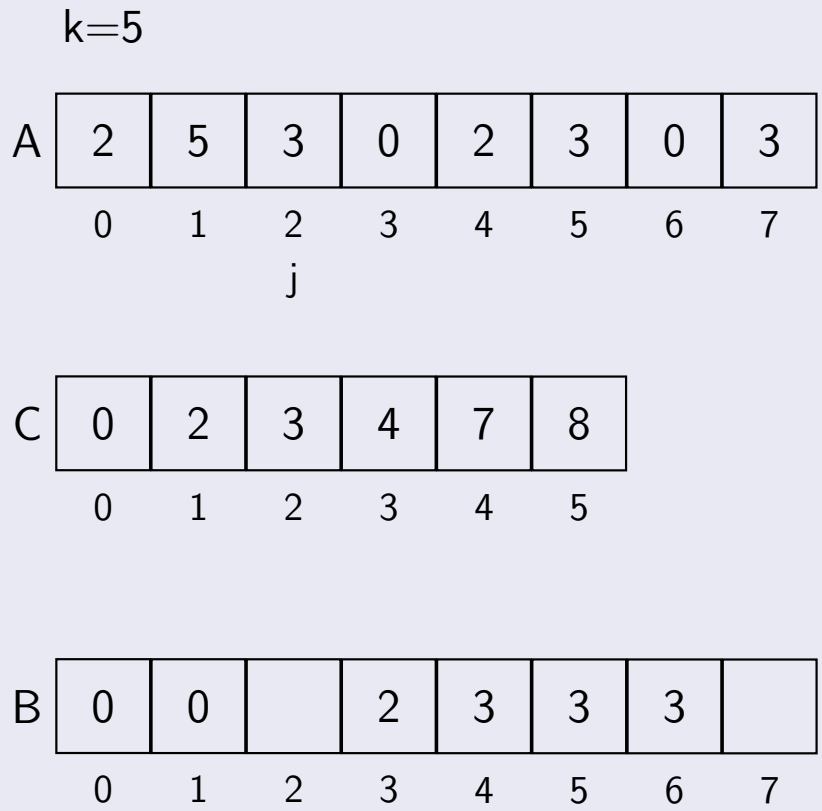
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

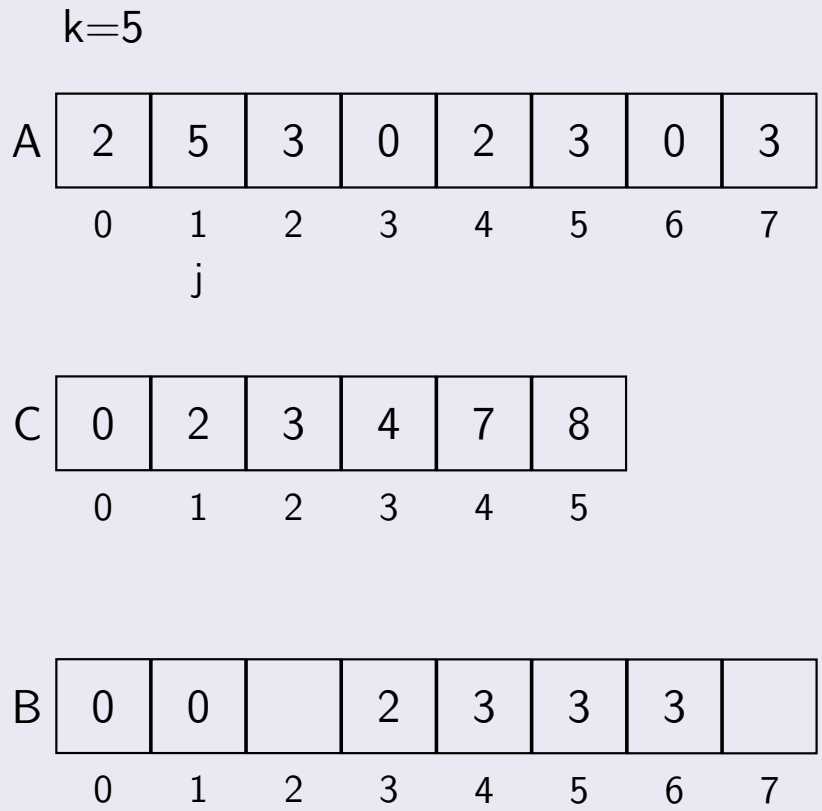
```
void countingSort(int A[], int k, int n) {
  int C[k+1];
  int B[n];
  int i, j;
  for (int i=0; i<=k; i++)
    C[i] = 0;
  for (j=0; j<n; j++)
    C[A[j]]++;
  for (i=1; i<=k; i++)
    C[i] = C[i] + C[i-1];
  for (j = n-1; j>=0; j--) {
    B[C[A[j]]-1] = A[j];
    C[A[j]]--;
  }
  for (j=0; j<n;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



Counting Sort

Implementação

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



Counting Sort

Implementação

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

k=5

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

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

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

Counting Sort

Implementação

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

k=5

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

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

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

Counting Sort

Implementação

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

k=5

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

j

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

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

Counting Sort

Implementação

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

k=5

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

j

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

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

Counting Sort

Implementação

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

k=5

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

j

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

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

Counting Sort

Implementação

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

k=5

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

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

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

Counting Sort

Implementação

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

k=5

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

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

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

Counting Sort

Complexidade

- E qual a complexidade disso?

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

Counting Sort

Complexidade

- E qual a complexidade disso?
  - $O(k)$

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

Counting Sort

Complexidade

- E qual a complexidade disso?
  - $O(k)$

    $+O(n)$

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

Counting Sort

Complexidade

- E qual a complexidade disso?
  - $O(k)$

    $+O(n)$

    $+O(k)$

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

Counting Sort

Complexidade

- E qual a complexidade disso?
  - $O(k)$

    $+O(n)$

    $+O(k)$

    $+O(n)$

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

Counting Sort

Complexidade

- E qual a complexidade disso?
  - $O(k)$

    $+O(n)$

    $+O(k)$

    $+O(n)$

    $+O(n)$

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

Counting Sort

Complexidade

- E qual a complexidade disso?
  - $O(k)$
    $+O(n)$
    $+O(k)$
    $+O(n)$
    $+O(n)$
  - $O(n+k)$

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

Counting Sort

Complexidade

- $O(n + k)$

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

Counting Sort

Complexidade

- $O(n + k)$
- Se $k \in O(n)$, então Counting Sort roda em $O(n)$

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

Counting Sort

Complexidade

- $O(n + k)$
- Se $k \in O(n)$, então Counting Sort roda em $O(n)$
  - $k \in O(n)$ significa que k **é constante** ou, no máximo, **linearmente proporcional a** n

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

Counting Sort

Complexidade

- $O(n + k)$
- Se $k \in O(n)$, então Counting Sort roda em $O(n)$
  - $k \in O(n)$ significa que k **é constante** ou, no máximo, **linearmente proporcional a** n
  - Essa é a limitação do algoritmo

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

Counting Sort

Complexidade

- Mas o limite inferior para ordenação não era $\Omega(n \log n)$?

Counting Sort

Complexidade

- Mas o limite inferior para ordenação não era $\Omega(n \log n)$?
  - Sim, mas para algoritmos baseados em comparações entre elementos do arranjo

Counting Sort

Complexidade

- Mas o limite inferior para ordenação não era $\Omega(n \log n)$?
  - Sim, mas para algoritmos baseados em comparações entre elementos do arranjo
- Nenhuma comparação entre elementos do arranjo ocorre no Counting Sort

Counting Sort

Complexidade

- Mas o limite inferior para ordenação não era $\Omega(n \log n)$?
  - Sim, mas para algoritmos baseados em comparações entre elementos do arranjo
- Nenhuma comparação entre elementos do arranjo ocorre no Counting Sort
  - Em vez disso, ele usa esses valores para indexar o arranjo `C`

Counting Sort

Complexidade

- Mas o limite inferior para ordenação não era $\Omega(n \log n)$?
  - Sim, mas para algoritmos baseados em comparações entre elementos do arranjo
- Nenhuma comparação entre elementos do arranjo ocorre no Counting Sort
  - Em vez disso, ele usa esses valores para indexar o arranjo `C`
- Por isso o limite $\Omega(n \log n)$ não se aplica

Counting Sort

Ordenação Estável

- Uma importante propriedade do Counting Sort é que ele é **estável**

Counting Sort

Ordenação Estável

- Uma importante propriedade do Counting Sort é que ele é **estável**
  - Ou seja, valores iguais aparecem no arranjo de saída na mesma ordem em que estavam no de entrada

Counting Sort

Ordenação Estável

- Uma importante propriedade do Counting Sort é que ele é **estável**
  - Ou seja, valores iguais aparecem no arranjo de saída na mesma ordem em que estavam no de entrada
- Isso é importante quando cada dado carrega consigo outros dados satélites

Counting Sort

Ordenação Estável

- Uma importante propriedade do Counting Sort é que ele é **estável**
  - Ou seja, valores iguais aparecem no arranjo de saída na mesma ordem em que estavam no de entrada
- Isso é importante quando cada dado carrega consigo outros dados satélites
  - Que dependem da ordem do dado em relação aos que têm o mesmo valor no arranjo

Counting Sort

Ordenação Estável

- Uma importante propriedade do Counting Sort é que ele é **estável**
  - Ou seja, valores iguais aparecem no arranjo de saída na mesma ordem em que estavam no de entrada
- Isso é importante quando cada dado carrega consigo outros dados satélites
  - Que dependem da ordem do dado em relação aos que têm o mesmo valor no arranjo
- Mais adiante veremos um exemplo de onde isso é importante...

Radix Sort

Radix Sort

- Pressuposto:

Radix Sort

- Pressuposto:
  - O Radix Sort assume que cada um dos n elementos da entrada é um inteiro de, no máximo, d dígitos (d constante)

Radix Sort

- Pressuposto:
  - O Radix Sort assume que cada um dos n elementos da entrada é um inteiro de, no máximo, d dígitos (d constante)
  - Onde o dígito 1 é o menos significativo (direita) e o dígito d o mais significativo (esquerda)

Radix Sort

- Pressuposto:
  - O Radix Sort assume que cada um dos n elementos da entrada é um inteiro de, no máximo, d dígitos (d constante)
  - Onde o dígito 1 é o menos significativo (direita) e o dígito d o mais significativo (esquerda)
  - Ex: CEP – inteiro de 8 dígitos

Radix Sort

- Pressuposto:
  - O Radix Sort assume que cada um dos n elementos da entrada é um inteiro de, no máximo, d dígitos (d constante)
  - Onde o dígito 1 é o menos significativo (direita) e o dígito d o mais significativo (esquerda)
  - Ex: CEP – inteiro de 8 dígitos
- Note que cada elemento da entrada ter um comprimento máximo constante o torna independente de n

Radix Sort

- Pressuposto:
  - O Radix Sort assume que cada um dos n elementos da entrada é um inteiro de, no máximo, d dígitos (d constante)
  - Onde o dígito 1 é o menos significativo (direita) e o dígito d o mais significativo (esquerda)
  - Ex: CEP – inteiro de 8 dígitos
- Note que cada elemento da entrada ter um comprimento máximo constante o torna independente de n
  - Independente do tamanho da própria entrada

Radix Sort

Funcionamento

Radix Sort

Funcionamento

1. Ordene os valores primeiro pelo dígito menos significativo

Radix Sort

Funcionamento

1. Ordene os valores primeiro pelo dígito menos significativo
2. Repita a operação para o segundo dígito menos significativo

Radix Sort

Funcionamento

1. Ordene os valores primeiro pelo dígito menos significativo
2. Repita a operação para o segundo dígito menos significativo
3. Continue até ter passado por todos os dígitos

Radix Sort

Funcionamento

1. Ordene os valores primeiro pelo dígito menos significativo
2. Repita a operação para o segundo dígito menos significativo
3. Continue até ter passado por todos os dígitos

329
457
657
839
436
720
355

Radix Sort

Funcionamento

1. Ordene os valores primeiro pelo dígito menos significativo
2. Repita a operação para o segundo dígito menos significativo
3. Continue até ter passado por todos os dígitos

329	720
457	355
657	436
839	457
436	657
720	329
355	839

Radix Sort

Funcionamento

1. Ordene os valores primeiro pelo dígito menos significativo
2. Repita a operação para o segundo dígito menos significativo
3. Continue até ter passado por todos os dígitos

329	720	720
457	355	329
657	436	436
839	457	839
436	657	355
720	329	457
355	839	657

Radix Sort

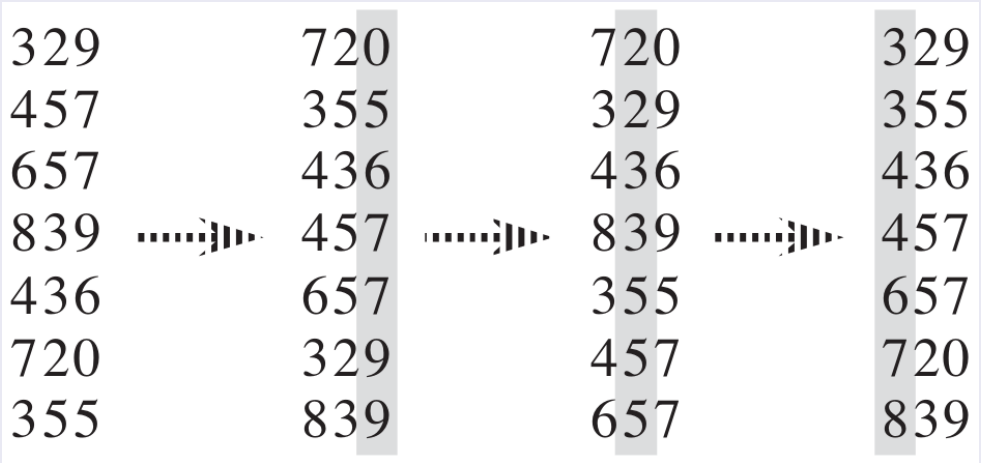
Funcionamento

1. Ordene os valores primeiro pelo dígito menos significativo
2. Repita a operação para o segundo dígito menos significativo
3. Continue até ter passado por todos os dígitos

329	720	720	329
457	355	329	355
657	436	436	436
839	457	839	457
436	657	355	657
720	329	457	720
355	839	657	839

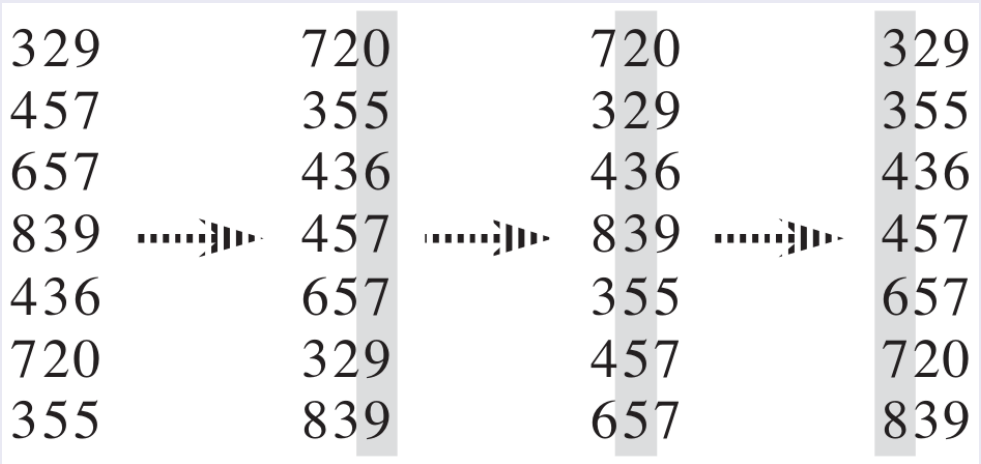
Radix Sort

Funcionamento



Radix Sort

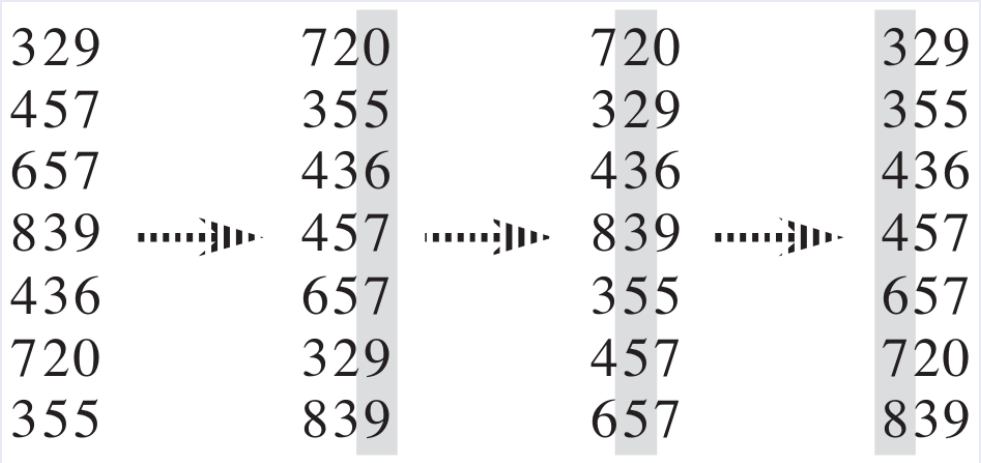
Funcionamento



- Para que isso funcione, é essencial que a ordenação dos dígitos seja estável

Radix Sort

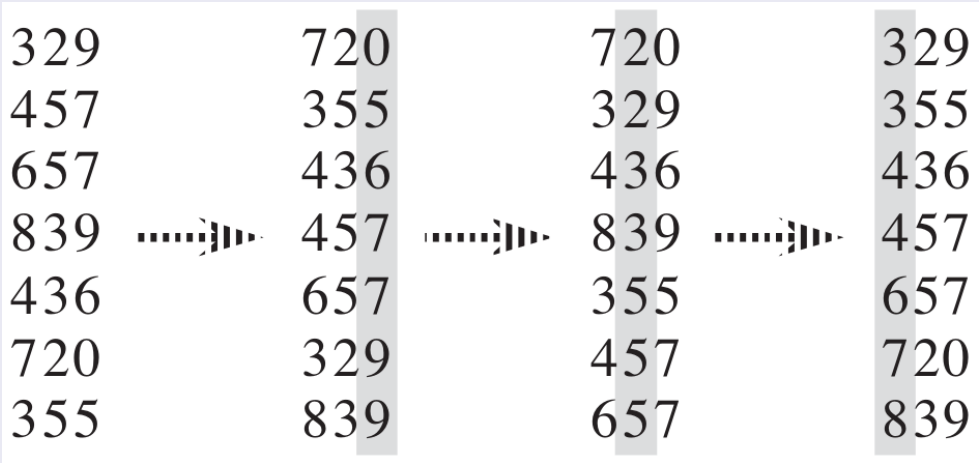
Funcionamento



- Para que isso funcione, é essencial que a ordenação dos dígitos seja estável
  - De modo a manter a ordem dos dígitos menos significativos já ordenados

Radix Sort

Funcionamento



- Para que isso funcione, é essencial que a ordenação dos dígitos seja estável
  - De modo a manter a ordem dos dígitos menos significativos já ordenados
  - Ex: Counting Sort e Merge Sort (se implementado para tal)

Radix Sort

Algoritmo

```
radixSort(A, d, n):
  para i = 1 até d:
    use um método estável para ordenar o
          arranjo A de tamanho n no dígito i
```

[a] Já que o Counting Sort estará ordenando apenas um dígito de cada vez, sua restrição (para k) é mais "branda".

Radix Sort

Algoritmo

```
radixSort(A, d, n):
  para i = 1 até d:
    use um método estável para ordenar o
          arranjo A de tamanho n no dígito i
```

- E qual a complexidade disso?

[a] Já que o Counting Sort estará ordenando apenas um dígito de cada vez, sua restrição (para k) é mais "branda".

Radix Sort

Algoritmo

```
radixSort(A, d, n):
  para i = 1 até d:
    use um método estável para ordenar o
          arranjo A de tamanho n no dígito i
```

- E qual a complexidade disso?
  - d

[a] Já que o Counting Sort estará ordenando apenas um dígito de cada vez, sua restrição (para k) é mais "branda".

Radix Sort

Algoritmo

```
radixSort(A, d, n):
  para i = 1 até d:
    use um método estável para ordenar o
          arranjo A de tamanho n no dígito i
```

- E qual a complexidade disso?
  - $d \times O(f(n))$, onde $O(f(n))$ é a complexidade do algoritmo de ordenação usado

[a] Já que o Counting Sort estará ordenando apenas um dígito de cada vez, sua restrição (para k) é mais "branda".

Radix Sort

Algoritmo

```
radixSort(A, d, n):
  para i = 1 até d:
    use um método estável para ordenar o
          arranjo A de tamanho n no dígito i
```

- E qual a complexidade disso?
  - $d \times O(f(n))$, onde $O(f(n))$ é a complexidade do algoritmo de ordenação usado
  - Se $d \in O(1)$ e o algoritmo auxiliar for o Counting Sort, em sua versão linear[a], então $O(f(n)) = O(n)$, e o Radix Sort será $O(n)$

[a] Já que o Counting Sort estará ordenando apenas um dígito de cada vez, sua restrição (para k) é mais "branda".

Radix Sort - ordenando nomes

Código

```
typedef struct {
  int id;
  char* nome;
} PESSOA;
```

Radix Sort - ordenando nomes

Código

```
typedef struct {
  int id;
  char* nome;
} PESSOA;


void radixSort(PESSOA A[], int d, int n){
  int x;
  for (x=d-1; x>=0; x--)
    countingSort2(A, x, n);
}
```

Radix Sort - ordenando nomes

Código

```
typedef struct {
  int id;
  char* nome;
} PESSOA;


void radixSort(PESSOA A[], int d, int n){
  int x;
  for (x=d-1; x>=0; x--)
    countingSort2(A, x, n);
}
```

Radix Sort - ordenando nomes

Código

```
typedef struct {
  int id;
  char* nome;
} PESSOA;


void radixSort(PESSOA A[], int d, int n){
  int x;
  for (x=d-1; x>=0; x--)
    countingSort2(A, x, n);
}
```

Radix Sort - ordenando nomes

Código

```
void countingSort2(PESSOA A[], int x, int n) {
  int i, j, k;
  k = 128;
  int C[k+1];
  PESSOA B[n];
  for (i=0; i<=k; i++) C[i] = 0;
  for (j=0; j<n; j++) C[A[j].nome[x]]++;
  for (i=1; i<=k; i++) C[i] = C[i] + C[i-1];
  for (int j = n-1; j>=0; j--) {
    B[C[A[j].nome[x]]-1] = A[j];
    C[A[j].nome[x]]--;
  }
  for (j=0; j<n; j++) A[j] = B[j];
}
```

Radix Sort - ordenando nomes

Código

```
void countingSort2(PESSOA A[], int x, int n) {
  int i, j, k;
  k = 128;
  int C[k+1];
  PESSOA B[n];
  for (i=0; i<=k; i++) C[i] = 0;
  for (j=0; j<n; j++) C[A[j].nome[x]]++;
  for (i=1; i<=k; i++) C[i] = C[i] + C[i-1];
  for (int j = n-1; j>=0; j--) {
    B[C[A[j].nome[x]]-1] = A[j];
    C[A[j].nome[x]]--;
  }
  for (j=0; j<n; j++) A[j] = B[j];
}
```

Radix Sort - ordenando nomes

Código

```
int main(){
  PESSOA pessoas[7];
  pessoas[0] = novaPessoa(1, "Luciano Digi   \0");
  pessoas[1] = novaPessoa(2, "Luciano Antonio\0");
  pessoas[2] = novaPessoa(3, "Eduardo TumTum \0");
  pessoas[3] = novaPessoa(4, "Eduardo Antonio\0");
  pessoas[4] = novaPessoa(5, "Norton Trevisan\0");
  pessoas[5] = novaPessoa(6, "Norton Roman   \0");
  pessoas[6] = novaPessoa(7, "Luciano Antonio\0");

  exibirPessoas(pessoas,7);
  radixSort(pessoas,16, 7);
  exibirPessoas(pessoas,7);

  return 0;
}
```

Radix Sort - ordenando nomes

Resultado

Conjunto inicial:

```
[1] Luciano Digi
[2] Luciano Antonio
[3] Eduardo TumTum
[4] Eduardo Antonio
[5] Norton Trevisan
[6] Norton Roman
[7] Luciano Antonio
```

Conjunto ordenado:

```
[4] Eduardo Antonio
[3] Eduardo TumTum
[2] Luciano Antonio
[7] Luciano Antonio
[1] Luciano Digi
[6] Norton Roman
[5] Norton Trevisan
```

Radix Sort - ordenando nomes

Resultado

Conjunto inicial:

```
[1] Luciano Digi
[2] Luciano Antonio
[3] Eduardo TumTum
[4] Eduardo Antonio
[5] Norton Trevisan
[6] Norton Roman
[7] Luciano Antonio
```

Conjunto ordenado:

```
[4] Eduardo Antonio
[3] Eduardo TumTum
[2] Luciano Antonio
[7] Luciano Antonio
[1] Luciano Digi
[6] Norton Roman
[5] Norton Trevisan
```

Radix Sort

Observações

- Antigamente usado nas máquinas mecânicas de ordenar cartas

Radix Sort

Observações

- Antigamente usado nas máquinas mecânicas de ordenar cartas
- Hoje, por vezes usado para ordenar registros de informação que possuem múltiplas chaves

Radix Sort

Observações

- Antigamente usado nas máquinas mecânicas de ordenar cartas
- Hoje, por vezes usado para ordenar registros de informação que possuem múltiplas chaves
  - Ex: datas

Radix Sort

Observações

- Antigamente usado nas máquinas mecânicas de ordenar cartas
- Hoje, por vezes usado para ordenar registros de informação que possuem múltiplas chaves
  - Ex: datas
    - Podemos usar um algoritmo de ordenação cuja função de comparação retorne a maior de duas dadas, comparando ano e, em caso de empate, mês e, em caso de empate, dia

Radix Sort

Observações

- Antigamente usado nas máquinas mecânicas de ordenar cartas
- Hoje, por vezes usado para ordenar registros de informação que possuem múltiplas chaves
  - Ex: datas
    - Podemos usar um algoritmo de ordenação cuja função de comparação retorne a maior de duas dadas, comparando ano e, em caso de empate, mês e, em caso de empate, dia
    - Ou ordenar a informação três vezes com um método estável de ordenação: primeiro no dia, em seguida no mês, e então no ano

Radix Sort

Observações

- Altamente dependente do método usado para ordenar os dígitos

Radix Sort

Observações

- Altamente dependente do método usado para ordenar os dígitos
- Com impacto direto em sua complexidade

Radix Sort

Observações

- Altamente dependente do método usado para ordenar os dígitos
- Com impacto direto em sua complexidade
- Se usado com Counting sort:

Radix Sort

Observações

- Altamente dependente do método usado para ordenar os dígitos
- Com impacto direto em sua complexidade
- Se usado com Counting sort:
  - Tem complexidade linear (se $d \in O(1)$)

Radix Sort

Observações

- Altamente dependente do método usado para ordenar os dígitos
- Com impacto direto em sua complexidade
- Se usado com Counting sort:
  - Tem complexidade linear (se $d \in O(1)$)
  - Não é *in-place*, criando diversos arranjos auxiliares na memória

Radix Sort

Discussão

- Desejamos ordenar um conjunto de $\mathbf{2^{20}}$ **números** de **64 bits**.

Radix Sort

Discussão

- Desejamos ordenar um conjunto de $\mathbf{2^{20}}$ **números** de **64 bits**.
- Qual algoritmo apresentaria um melhor desempenho: *Merge Sort* ou *Radix Sort*?

Radix Sort

Discussão

- Desejamos ordenar um conjunto de $\mathbf{2^{20}}$ **números** de **64 bits**.
- Qual algoritmo apresentaria um melhor desempenho: *Merge Sort* ou *Radix Sort*?
- Vamos considerar que o Merge Sort fará $\mathbf{n} * \mathbf{logn}$ operações e que o Radix Sort fará $\mathbf{d} * \mathbf{n}$ operações.

Radix Sort

Discussão

- Desejamos ordenar um conjunto de $\mathbf{2^{20}}$ **números** de **64 bits**.
- Qual algoritmo apresentaria um melhor desempenho: *Merge Sort* ou *Radix Sort*?
- Vamos considerar que o Merge Sort fará $\mathbf{n} * \mathbf{logn}$ operações e que o Radix Sort fará $\mathbf{d} * \mathbf{n}$ operações.
  - Merge Sort: $2^{20} * log_2 2^{20} = 20 * 2^{20} = 20.971.520$ operações

Radix Sort

Discussão

- Desejamos ordenar um conjunto de $\mathbf{2^{20}}$ **números** de **64 bits**.
- Qual algoritmo apresentaria um melhor desempenho: *Merge Sort* ou *Radix Sort*?
- Vamos considerar que o Merge Sort fará $\mathbf{n} * \mathbf{logn}$ operações e que o Radix Sort fará $\mathbf{d} * \mathbf{n}$ operações.
  - Merge Sort: $2^{20} * log_2 2^{20} = 20 * 2^{20} =$ 20.971.520 operações
  - Radix Sort: $64 * 2^{20} =$ 67.108.864 operações

Radix Sort

Discussão

- Quando os algoritmos empatariam?

Radix Sort

Discussão

- Quando os algoritmos empatariam?
- Empatariam se quisemos ordenar um conjunto de $\mathbf{2^{64}}$ **números** de 64 bits.

Radix Sort

Discussão

- Quando os algoritmos empatariam?
- Empatariam se quisemos ordenar um conjunto de $\mathbf{2^{64}}$ **números** de 64 bits.
- Seriam **18.446.744.073.709.600.000 números**, isto é: 18,4 exa números (18.446 peta números) de 64 bits.

Radix Sort

Discussão

- Quando os algoritmos empatariam?
- Empatariam se quisemos ordenar um conjunto de $\mathbf{2^{64}}$ **números** de 64 bits.
- Seriam **18.446.744.073.709.600.000 números**, isto é: 18,4 exa números (18.446 peta números) de 64 bits.
- Precisaríamos de **14.757 peta bytes** de memória principal só para termos o arranjo original na memória.

Referências

- Cormen, Thomas H., Leiserson, Charles E., Rivest, Ronald L., Stein, Clifford. Introduction to Algorithms. 2a ed. MIT Press, 2001.

- `https://www.geeksforgeeks.org/radix-sort/`

Aula 22 – CountingSort e RadixSort

Norton T. Roman & Luciano A. Digiampietri
digiampietri@usp.br
@digiampietri

2023