ENGENHARIA INFORMÁTICA E DE COMPUTADORES

Algoritmos e Estruturas de Dados

(parte 9 – Ordenação em Tempo Linear)

2° Semestre 2022/2023
Instituto Superior de Engenharia de Lisboa
Paula Graça

ORDENAÇÃO DE CUSTO LINEAR

- Counting Sort
  - É um algoritmo de ordenação baseado em contagem de elementos dentro de um intervalo conhecido de valores
    - Comparision Counting Sort
    - Distribution Counting Sort

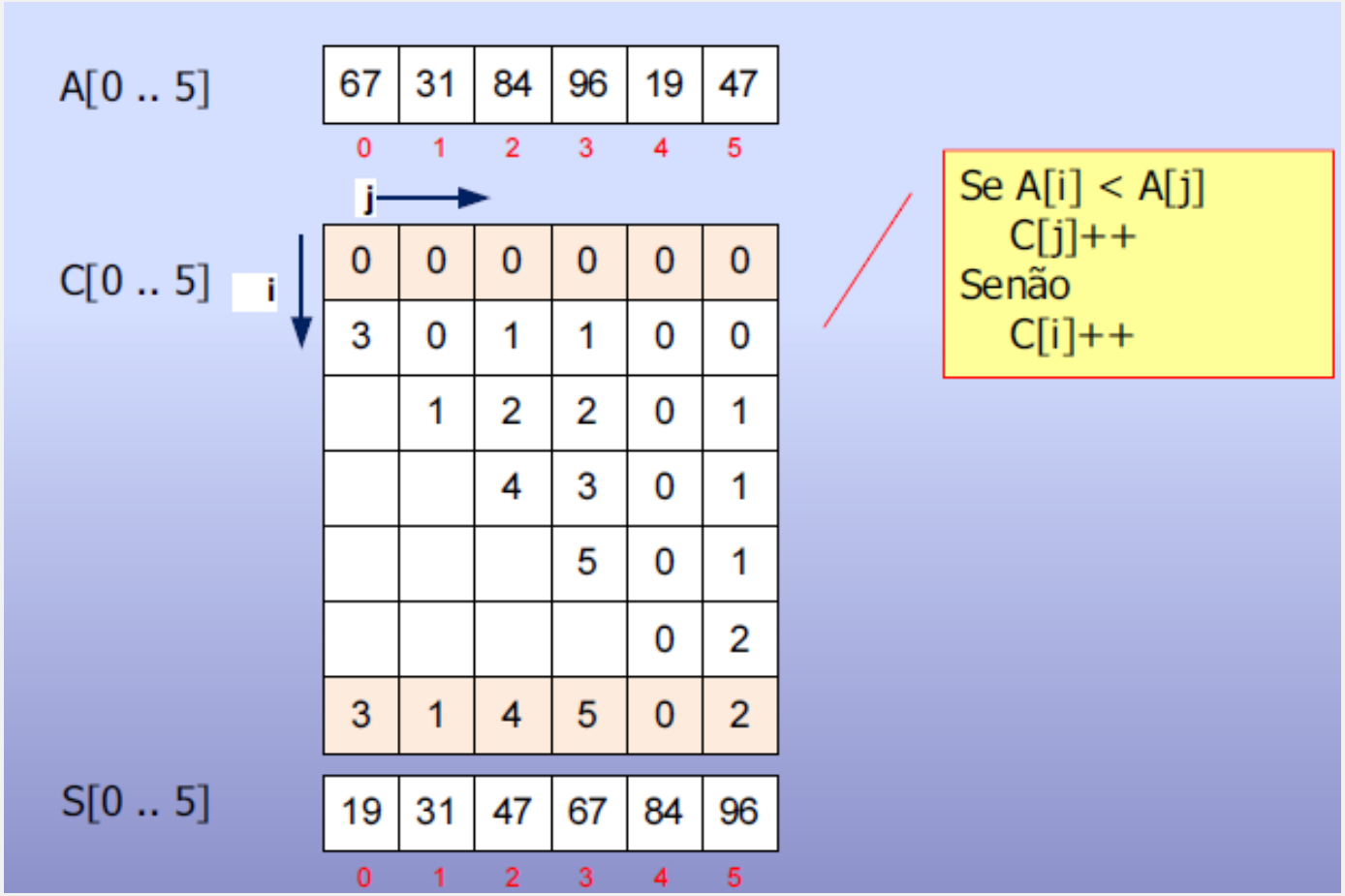
- Radix Sort
  - É um algoritmo de ordenação em tempo linear adequado para valores inteiros ou *strings* de dimensão fixa. Usam espaço adicional
    - LSD (Least Significant Digit) - Utiliza o algoritmo Distribution Counting Sort para a distribuição de cada elemento a ordenar
    - MSD (Most Significant Digit)

ORDENAÇÃO DE CUSTO LINEAR

- Comparision Counting Sort
  - Para cada elemento do conjunto a ordenar, contam-se todos os valores menores que ele próprio, registando-se o resultado numa tabela adicional
  - A contagem indica a posição de cada elemento no conjunto ordenado
    - Ex: Se contador=3 (existem 3 elementos menores), o elemento a arrumar fica na 4ª posição
  - Todos os elementos são copiados para o conjunto final ordenado, ocupando a posição indicada pela contagem

ORDENAÇÃO DE CUSTO LINEAR

- Comparision Counting Sort



ORDENAÇÃO DE CUSTO LINEAR

- Comparision Counting Sort

$T(n) = O(n^2)$

Este algoritmo mostra o princípio da ordenação por contagem, mas executa em tempo quadrático. Pode melhorar.

```
ComparisionCountingSort(A, n)
    for i = 0 to n – 1
        C[i] = 0                    // inicializa os contadores
    for i = 0 to n – 2              // elementos a ordenar
        for j = i + 1 to n – 1      // contagem dos menores
            if A[i] < A[j]
                C[j] = C[j] + 1
            else C[i] = C[i] + 1
    for i = 0 to n – 1
        S[C[i]] = A[i]              // arruma na posição indicada pelo
    return S                        // contador
```

ORDENAÇÃO DE CUSTO LINEAR

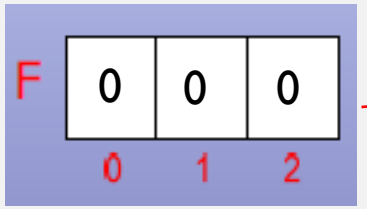
- **Distribution Counting Sort**
  - Tira vantagem do facto de serem conhecidos os valores do conjunto a ordenar, ou seja, é conhecido o seu intervalo de valores [ℓower .. $u$pper]

  - Considerando o *array* A[0 .. n-1]

13	11	12	13	12	12

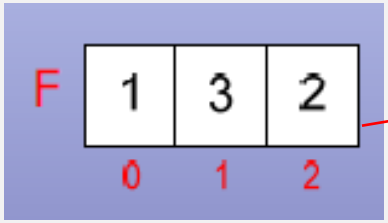
  - Os elementos pertencem ao conjunto {11, 12, 13}

$\ell = 11 \quad u = 13$



Dimensão do *array* de frequências:
n = u – ℓ + 1 = 3

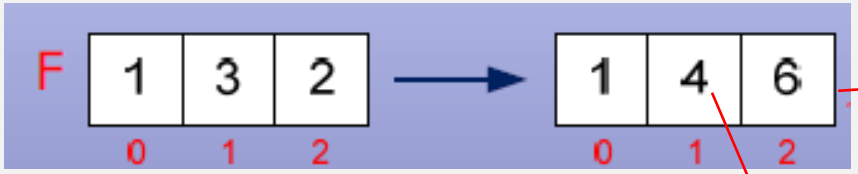
  - O *array* A é percorrido para que seja feita a contagem da frequência de cada elemento:



A frequência de cada elemento **A[i]** está em:
F[A[i] – ℓ]

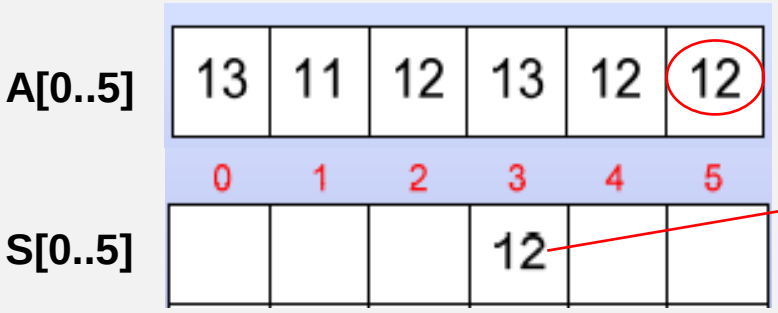
ORDENAÇÃO DE CUSTO LINEAR

- Calcula-se a distribuição dos elementos do *array* A, reutilizando o *array* de frequências F
- A distribuição calculada em F indica qual a posição final dos elementos no *array* ordenado



F[0] = F [0]
F[1] = F [0] + F [1]
F[2] = F [1] + F [2]

- Finalmente, copiam-se os elementos do *array* A (iniciando no último), para as posições finais ordenadas no *array* S, de acordo com as distribuições calculadas em F



Cada elemento **A[i]** é arrumado em:
S [F[A [i] – ℓ] - 1]

ORDENAÇÃO DE CUSTO LINEAR

- Cada valor da distribuição F[i], indica a posição da última ocorrência do elemento colocado no *array* ordenado S
- Começando em A[5] = 12, a distribuição calculada F[1] = 4, indica que o elemento vai para a 4ª posição S[3]

A[0 .. 5]	13	11	12	13	12	12
	0	1	2	3	4	5

F[0 .. 2]

	0	1	2
A[5] = 12	1	4	6
A[4] = 12	1	3	6
A[3] = 13	1	2	6
A[2] = 12	1	2	5
A[1] = 11	1	1	5
A[0] = 13	0	1	5

S[0 .. 5]

0	1	2	3	4	5
			12		
		12	12		
		12	12		13
	12	12	12		13
11	12	12	12		13
11	12	12	12	13	13

ORDENAÇÃO DE CUSTO LINEAR

- Distribution Counting Sort – Algoritmo

```
DistributionCountingSort(A, n)
    for j = 0 to u – ℓ                  // inicializa os contadores
        F[j] = 0
    for i = 0 to n – 1
        F[A[i]-ℓ] = F[A[i]-ℓ] + 1       // calcula as frequências
    for j = 1 to u – ℓ
        F[j] = F[j –1] + F[j]           // calcula as distribuições
    for i = n – 1 downto 0              // reutilizando F
        j = A[i] – ℓ
        S[F[j] – 1] = A[i]
        F[j] = F[j] – 1
    return S
```

ORDENAÇÃO DE CUSTO LINEAR

- Distribution Counting Sort – Análise

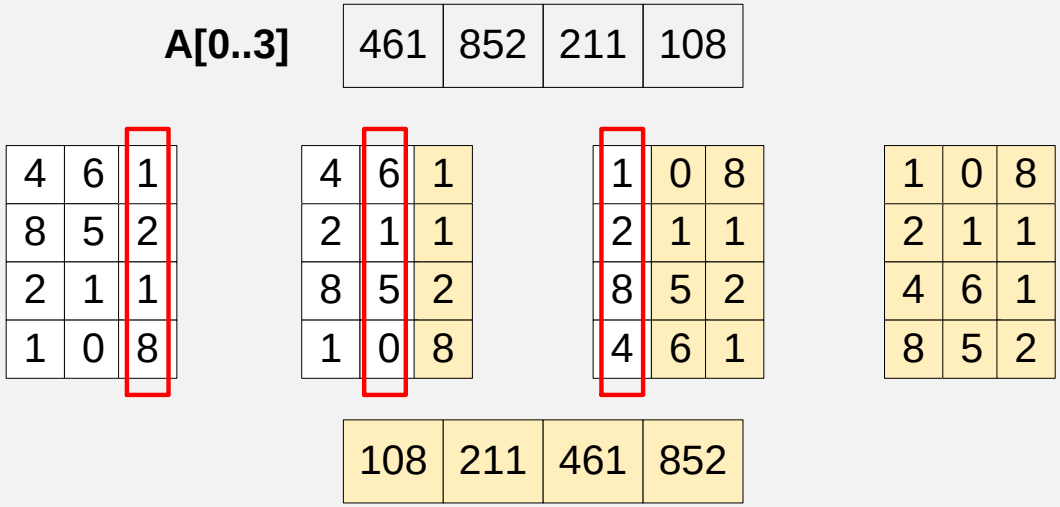
  - $T(n) = \Theta(n+k)$ sendo $k = u - l + 1$

    $= \Theta(n)$ se $k = O(n)$

ORDENAÇÃO DE CUSTO LINEAR

- Radix Sort (LSD)
  - Ordena os elementos do conjunto, arrumando em cada passagem, um dígito de cada elemento, iniciando no digito menos significativo até ao mais significativo
  - Para ordenar cada grupo de dígitos, utiliza-se o Distribution Counting Sort

  sendo ℓower = 0 e $u$pper = 9

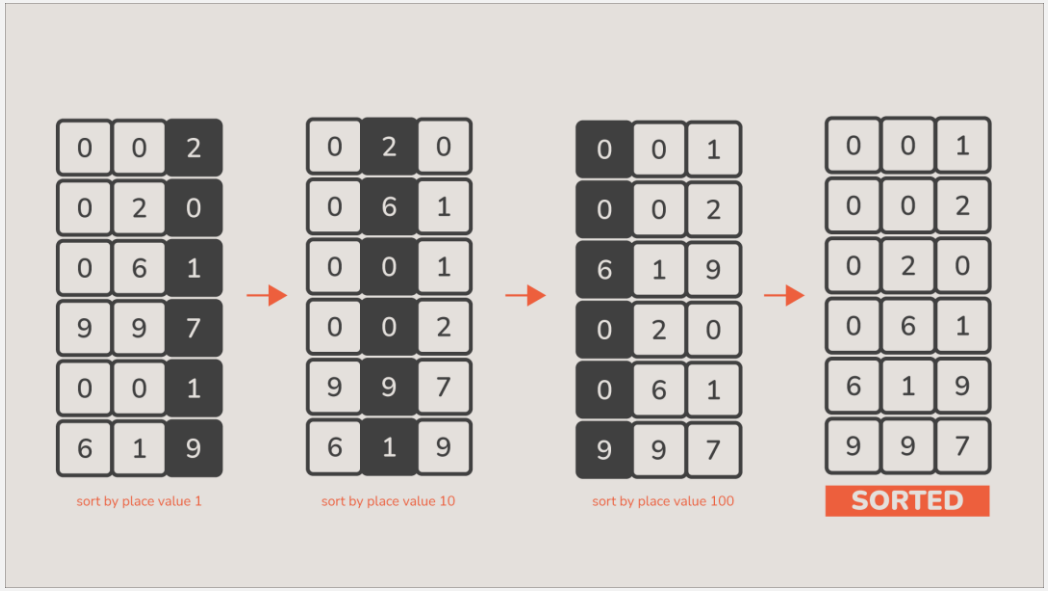


ORDENAÇÃO EM CUSTO LINEAR

- Mais exemplos
  - Ordenação de palavras com a mesma dimensão
  - Ordenação de valores com um número diferente de digitos

input	sorted result
4PGC938	1ICK750
2IYE230	1ICK750
3CIO720	1OHV845
1ICK750	1OHV845
1OHV845	1OHV845
4JZY524	2IYE230
1ICK750	2RLA629
3CIO720	2RLA629
1OHV845	3ATW723
1OHV845	3CIO720
2RLA629	3CIO720
2RLA629	4JZY524
3ATW723	4PGC938

keys are all the same length



ORDENAÇÃO DE CUSTO LINEAR

- Radix Sort – Análise
  - $T(n) = \Theta(n+k)$ para cada digito

    $= \Theta(dn)$ existem d (dígitos) iterações

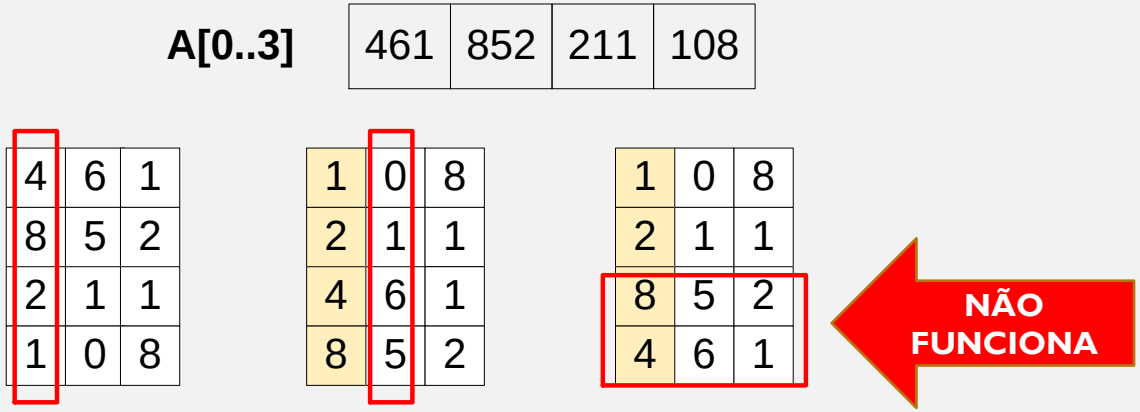
    se $k = O(n)$

- Radix Sort - Algoritmo

```
RadixSort(A, d)
    for i = 1 to d      // d = número de dígitos
        do use DistributionCountingSort to sort array A on digit i
    return S
```

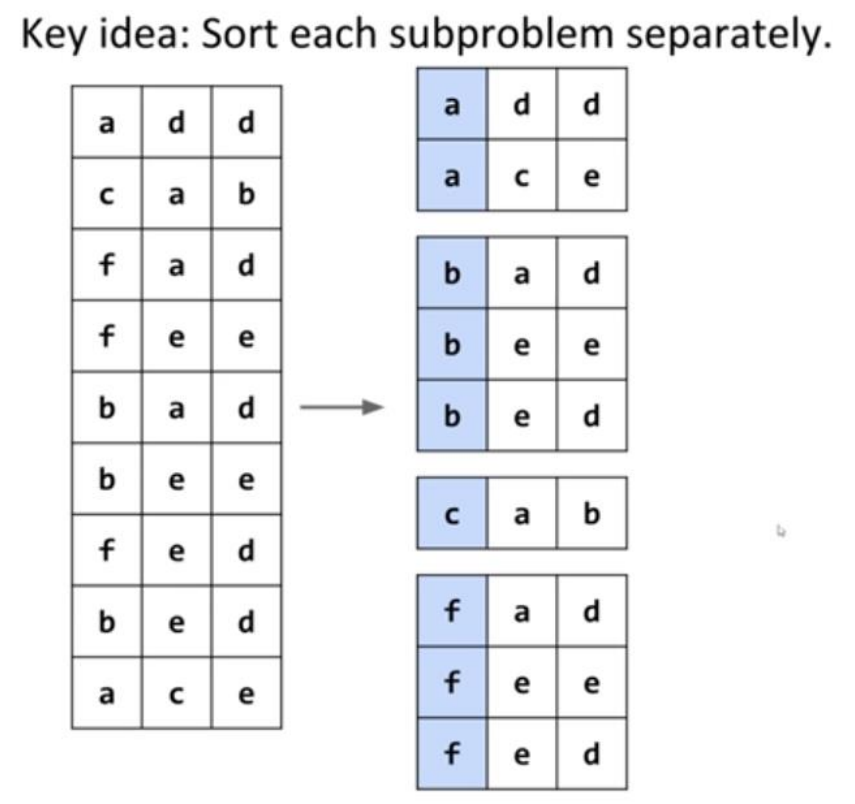
ORDENAÇÃO DE CUSTO LINEAR

- Radix Sort (MSD)
  - Ordenação dos elementos do conjunto, através da arrumação por cada um dos seus dígitos, iniciando no dígito mais significativo
  - Usando o mesmo método de LSD, mas iniciando no dígito mais significativo



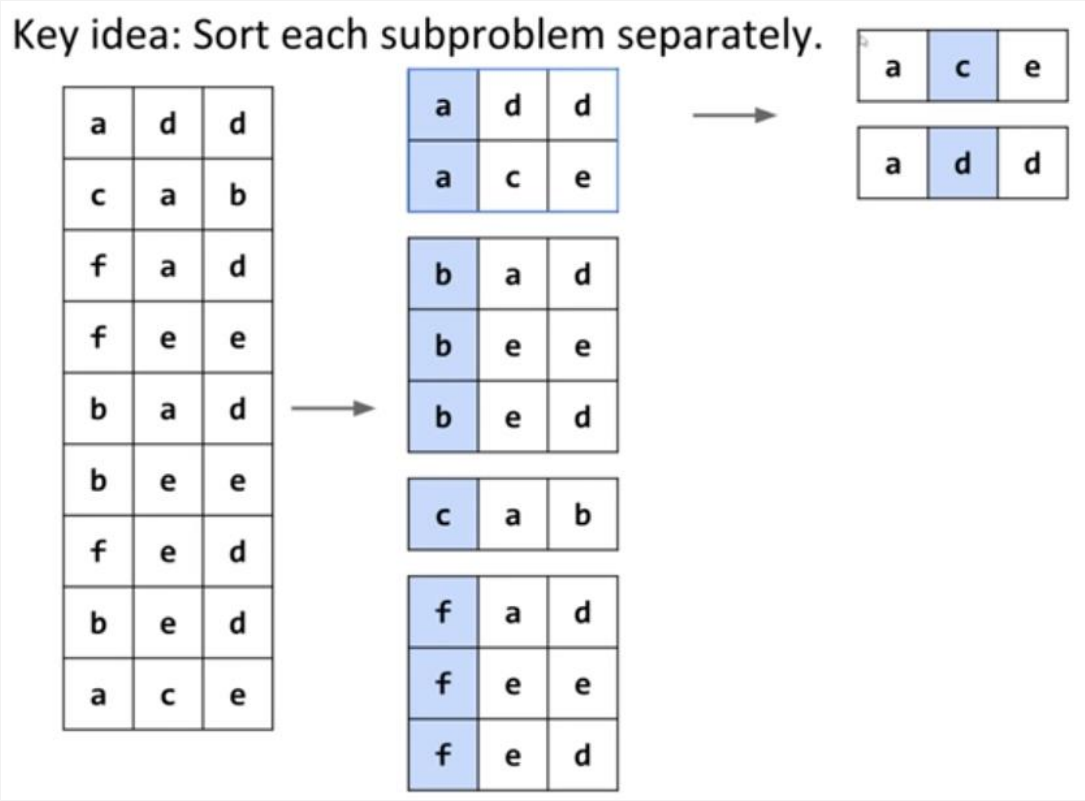
ORDENAÇÃO DE CUSTO LINEAR

- Radix Sort (MSD)



ALGORITMOS DE CUSTO LINEAR

- Radix Sort (MSD)



ALGORITMOS DE CUSTO LINEAR

- Radix Sort (MSD)