ALGORITMOS DE ORDENAÇÃO EM TEMPO LINEAR

DCE792 - AEDs II (Prática)

Atualizado em: 28 de outubro de 2024

Iago Carvalho

Departamento de Ciência da Computação

COMPARAÇÃO VS. CONTAGEM

Até o momento, todos os algoritmos de ordenação estudados trabalhavam com a comparação de elements

- Item i é menor ou maior que j?

Entretanto, existe outra classe de algoritmos de ordenação baseada em contagem

- Algoritmos de tempo linear
- Complexidade $O(n + k)$
  - n é o número de itens a serem ordenados
  - k é o valor do maior item a ser ordenado

Como a complexidade depende de um parâmetro adicional k da entrada, estes algoritmos podem ser melhores ou piores que os que estudamos até o momento

ALGORITMOS BASEADOS EM CONTAGEM

Hoje nós vamos estudar 3 algoritmos baseados em contagem

- Eles são facilmente aplicados para ordenar valores inteiros
- Existem variações para números reais e *strings*

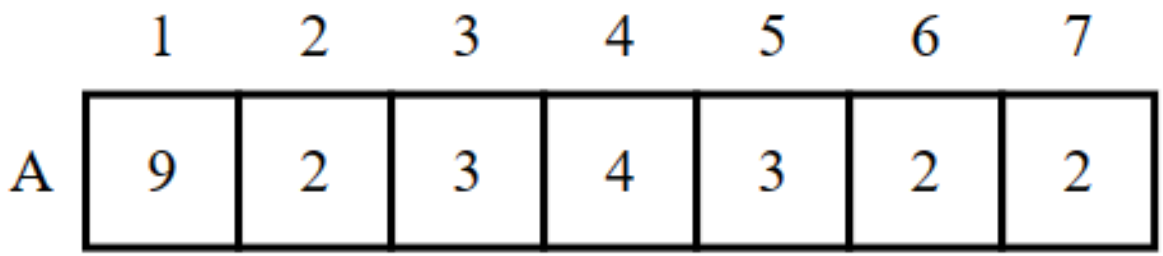
1. Counting sort
2. Bucket sort
3. Radix sort

COUTING SORT

COUNTING SORT

Este algoritmo recebe como entrada um vetor A de n inteiros

- Os inteiros variam entre 1 e k

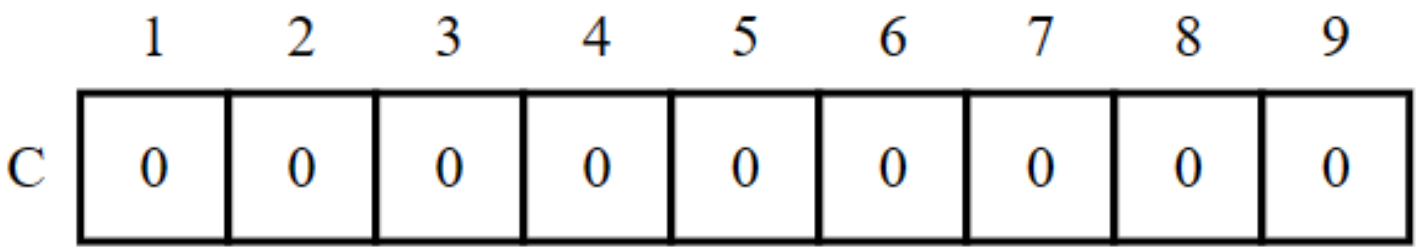


$n = 7, k = 9$

COUTING SORT

O Counting sort utiliza um vetor adicional C de k posições

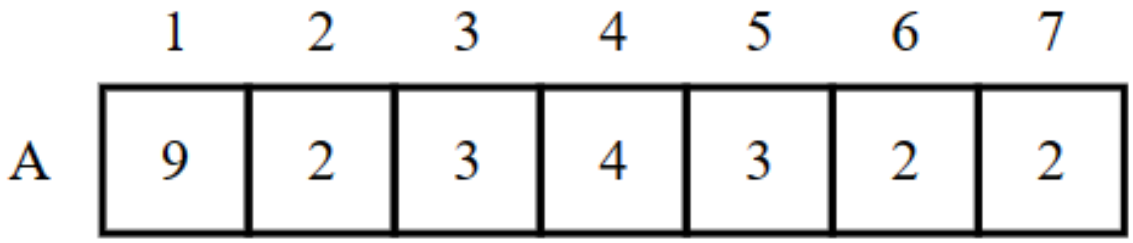
- Vetor todo inicializado com zeros

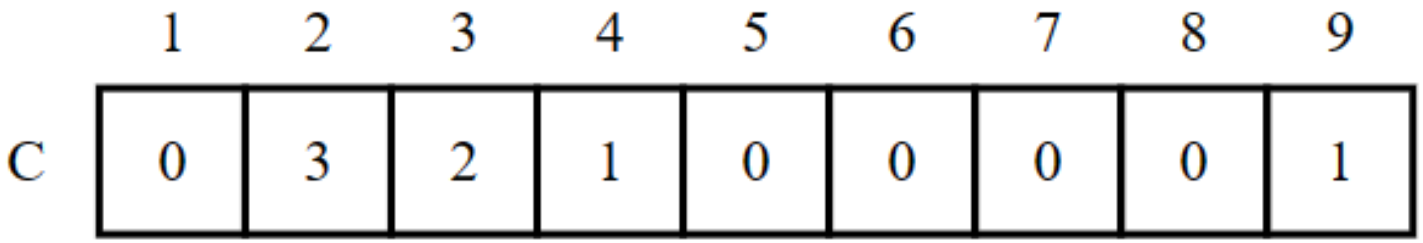


COUNTING SORT

O Counting sort intera sobre o vetor *A* contando o número de elementos iguais

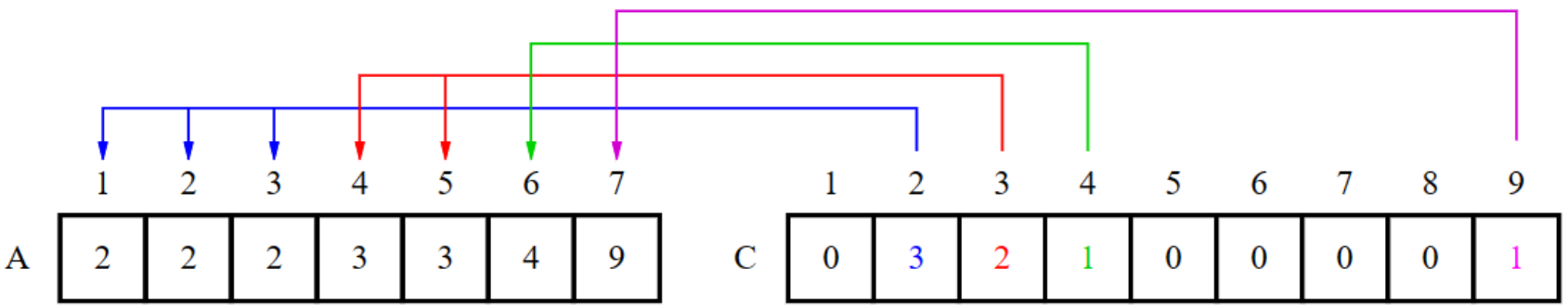
- Salva este número no vetor *C*





COUNTING SORT

Por fim, o Counting sort itera sobre o vetor C e reconstrói o vetor A de maneira ordenada



COUNTING SORT - CONSIDERAÇÕES

O Counting sort só ordena números inteiros

- Imagine contar todos os números reais existentes...

Ele pode requerer um grande espaço de memória

- O que acontece se $A = \{3, 9, 8, 5, 2, 1, 100000\}$?

Complexidade do algoritmo varia de acordo com k

- Se $k = n^3$, então a complexidade do Counting sort é $O(n^3)$

BUCKET SORT

BUCKET SORT

O Bucket sort é um segundo algoritmo baseado em contagem

- Utiliza menos memória adicional que o Counting sort

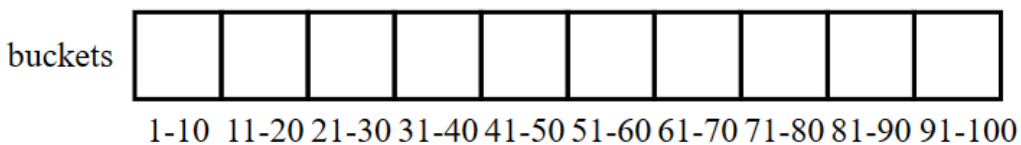
Ele é uma variação do Counting sort útil quando o vetor de entrada segue uma distribuição próxima a uniforme

Enquanto o Counting sort conta os diferentes elementos, o Bucket sort agrupa elementos de valor similar

BUCKET SORT

Vamos supor uma entrada com $|A| = 20$ e $k = 100$

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A	61	42	67	27	17	75	56	93	76	46	63	55	70	59	98	9	7	67	95	90

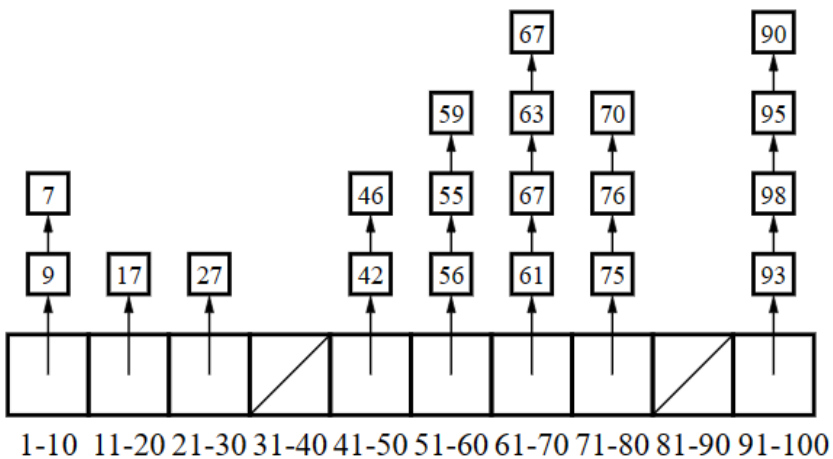


BUCKET SORT

O Bucket sort faz uma contagem similar ao Counting sort

- Entretanto, ao invés de contar exatamente quantos elementos iguais existem, ele agrupa os elementos nos *buckets*
- Cada *bucket* é uma lista linear

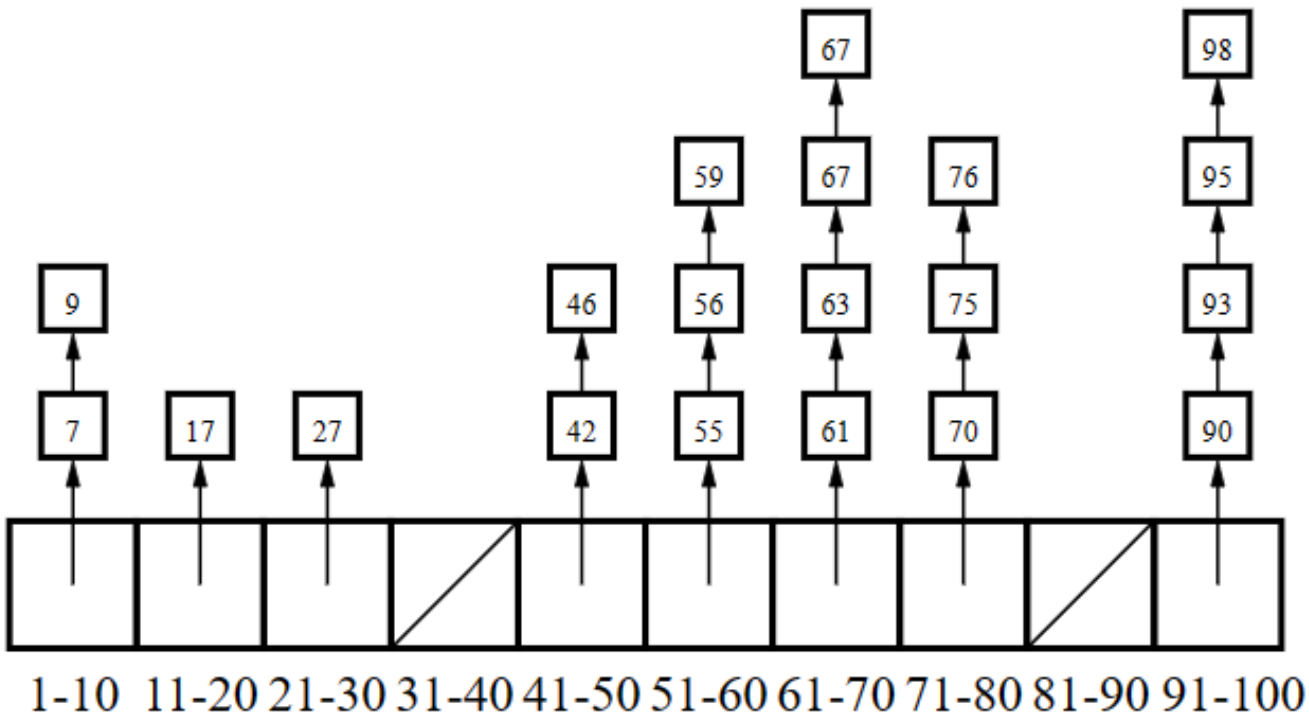
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
A	61	42	67	27	17	75	56	93	76	46	63	55	70	59	98	9	7	67	95	90



BUCKET SORT

O próximo passo então é ordenar cada uma destas listas lineares individualmente

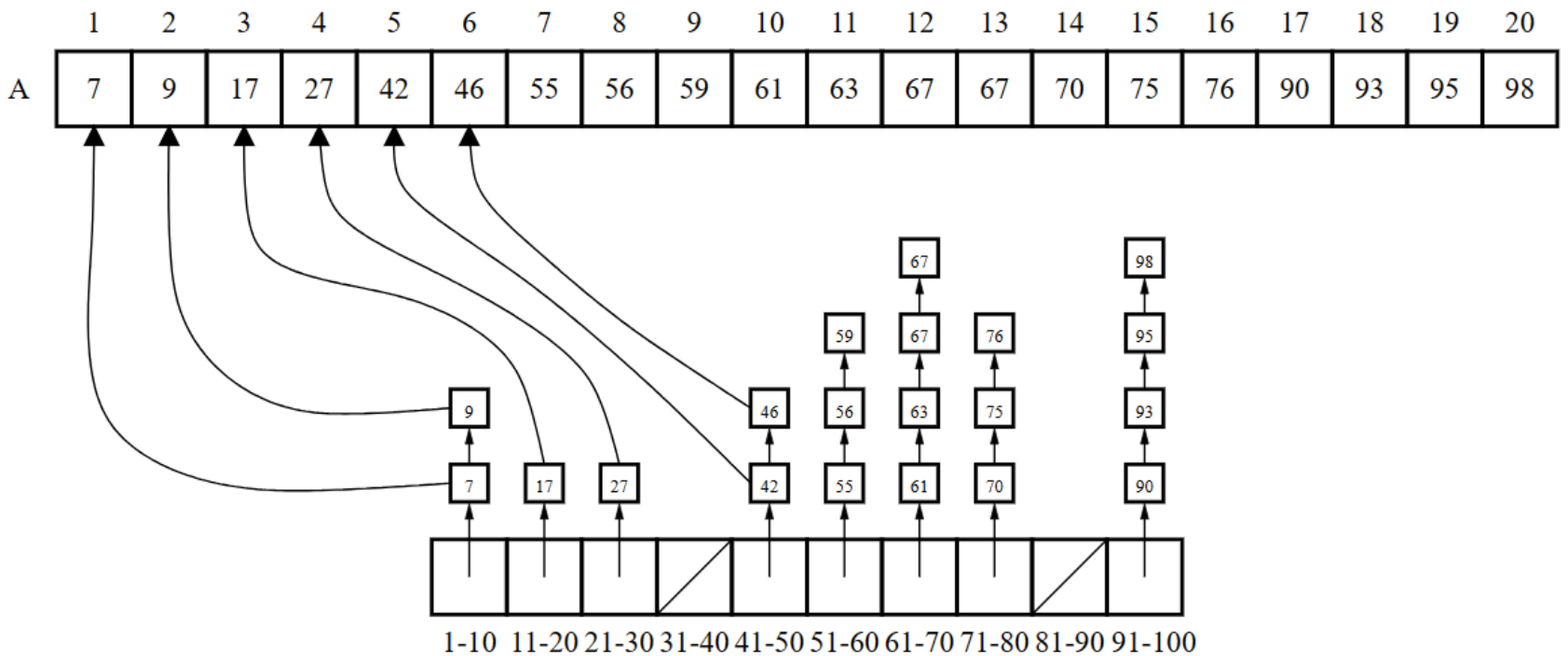
- Qualquer algoritmo de ordenação pode ser utilizado



BUCKET SORT

Por fim, o Bucket sort percorre os *buckets* para ordenar o vetor *A*

- Similar a estratégia do Counting sort



BUCKET SORT - CONSIDERAÇÕES

Assim como o Counting sort, o Bounting sort só ordena números inteiros

- Imagine contar todos os números reais existentes...

Ele é muito bom quando os elementos estão uniformemente distribuídos

- Na vida real, isso praticamente nunca acontece

Complexidade do algoritmo varia de acordo com k, com o algoritmo de ordenação utilizado e com o número de *buckets*

- Normalmente é utilizado o Insertion sort $\left(O(n^2)\right)$
- Outros algoritmos não lidam bem com listas encadeadas

RADIX SORT

RADIX SORT

O Counting sort utiliza uma grande quantidade de memória adicional.

O Bucket sort depende da distribuição dos dados de entrada e do número de *buckets*, o que pode levar a complexidade de $O(n^2)$

O Radix sort tenta encontrar um meio termo entre ambos os algoritmos

RADIX SORT

O Radix sort também é baseado em *buckets*

Entretanto, aqui nós temos um *bucket* para cada dígito do valor de entrada

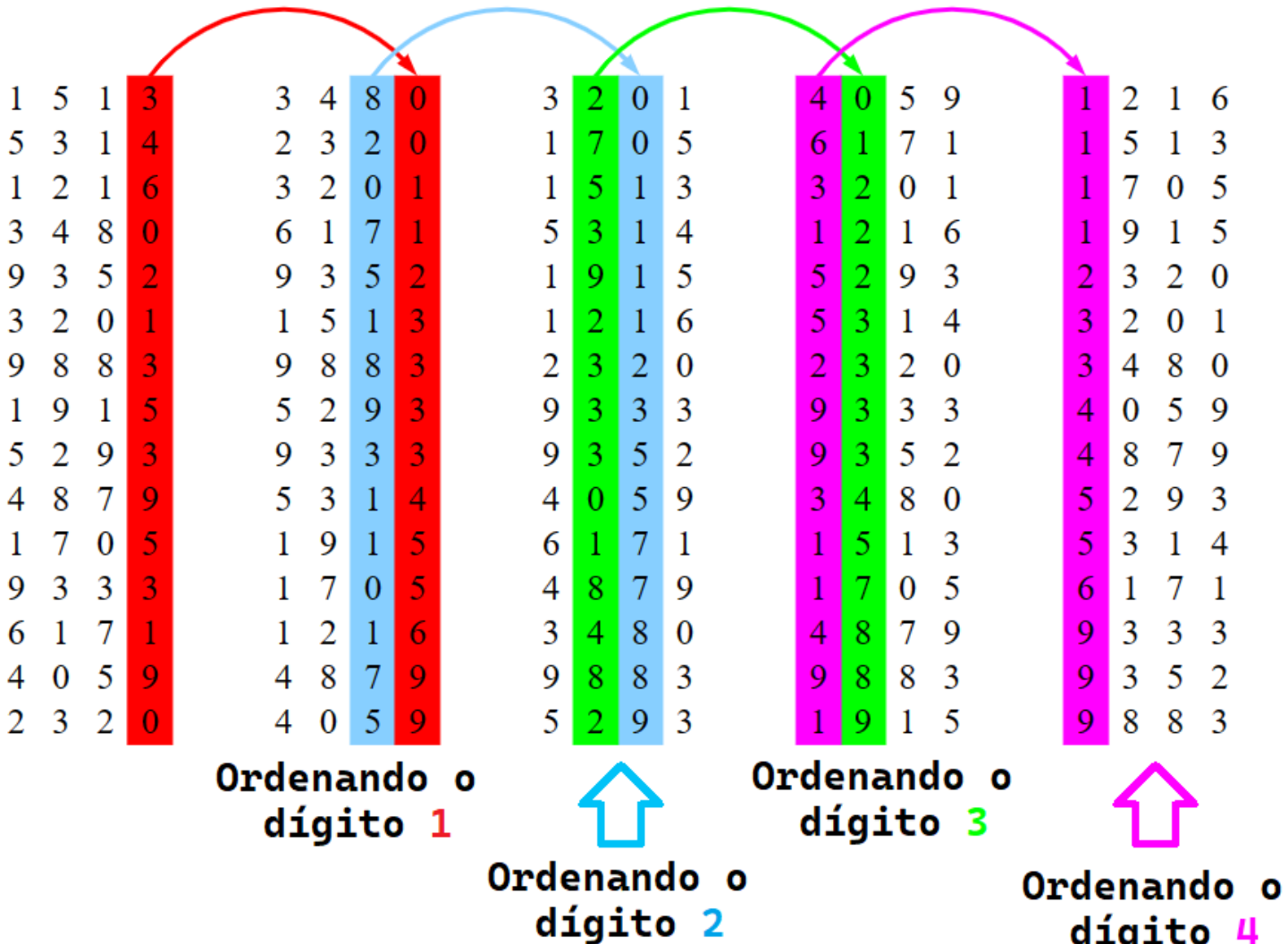
- Cada dígito é inserido em um *bucket*
- O número inteiro *5429* tem 4 dígitos
- A palavra *futebol* tem 7 dígitos
- O número real *7,53* tem 3 dígitos

A ideia é ordenar cada dígito separadamente

- Do menos significativo para o mais significativo
- Ou o contrário

RADIX SORT

Considere o Radix sort aplicado para ordenar inteiros de 4 dígitos - do menos significativo para o mais significativo



RADIX SORT

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

Como estamos trabalhando com inteiros, podemos fazer a ordenação utilizando o Counting sort

- Temos $k = 10$
- Complexidade do Radix sort é $O(nd)$
  - d é o número de dígitos

Este algoritmo pode ser adaptado para ordenar *strings*

- O tamanho k de cada *bucket* irá mudar

ATIVIDADE PRÁTICA

ATIVIDADE

Deve-se implementar os três algoritmos de ordenação para inteiros

Desta vez, sem código base

Lembre-se: um deles será necessário para nosso terceiro trabalho prático

PRÓXIMA AULA:

LISTAS DE PRIORIDADE