

# Algoritmos e Estruturas de Dados

---

## Vetores: Ordenação

Prof. Maiquel de Brito  
[maiquel.b@ufsc.br](mailto:maiquel.b@ufsc.br)

# Ordenação de Vetores

**Problema:** dado um vetor ( $v$ ) com **N** elementos, rearranjar esses elementos por ordem crescente.

- Entrada: vetor com elementos a serem ordenados
- Saída: mesmo vetor com elementos na ordem especificada
- Ordenação:
  - Pode ser aplicado a qualquer dado com ordem bem definida (inclusive *structs*)
  - Ordenação de *structs*: a ordenação é baseada em um ou mais campos

# Algoritmos de Ordenação

Ordenação por inserção (*insertion sort*)

Ordenação por seleção (*selection sort*)

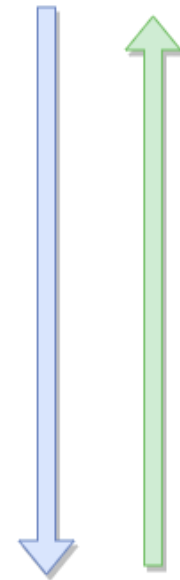
*BubbleSort*

*ShellSort*

*MergeSort*

*QuickSort*

*HeapSort*



Codificação simples



Complexidade (custo) computacional no pior caso

Nome - algoritmos estudados na disciplina

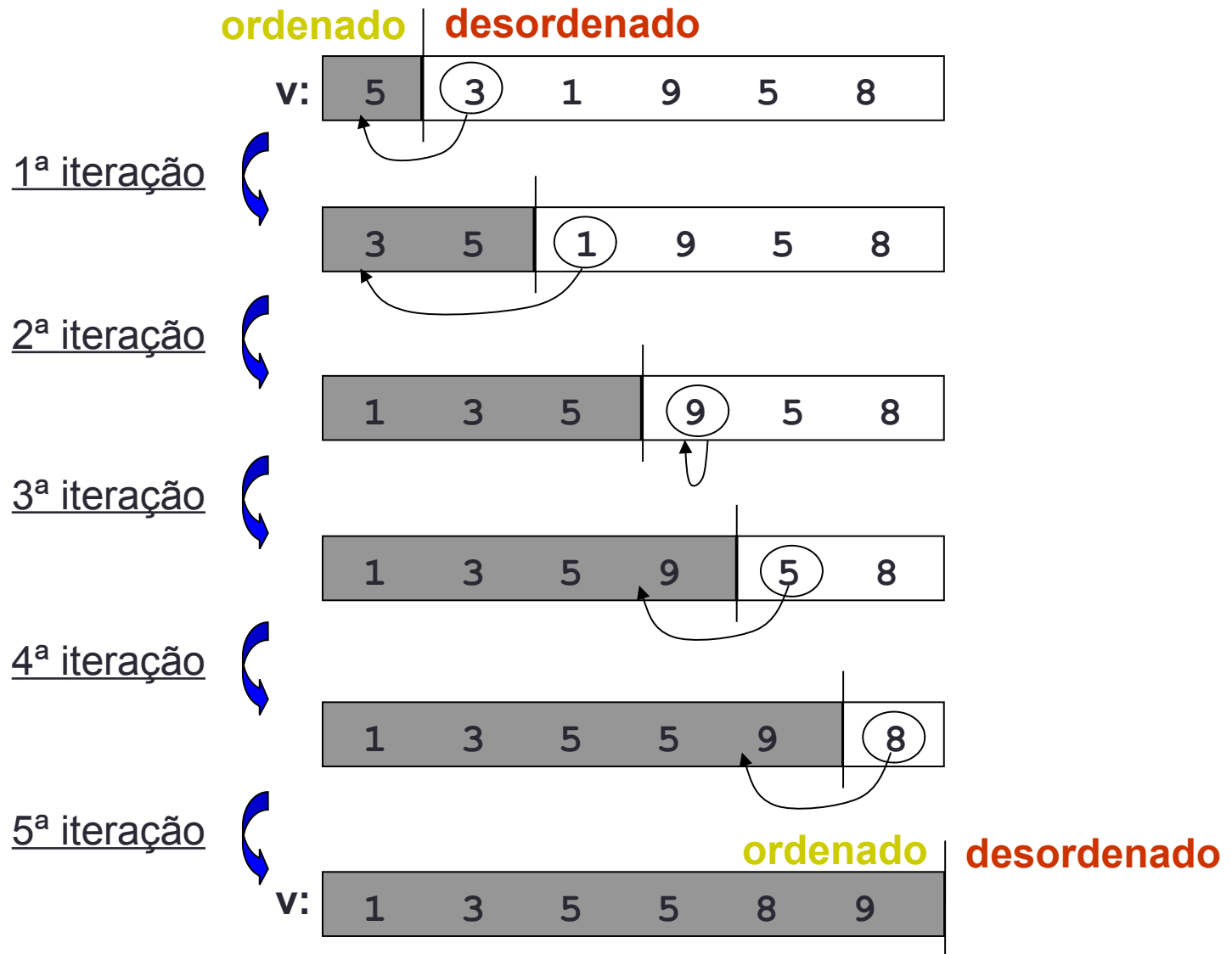
# Ordenação por Inserção

Percorre-se um vetor de elementos da esquerda para a direita e à medida que avança vai deixando os elementos mais à esquerda ordenados

## Algoritmo

- Considera-se o vetor dividido em dois sub-vetores (esquerdo e direito), com o da esquerda ordenado e o da direita desordenado
- Começa-se com um elemento apenas no sub-vetor da esquerda
- Move-se um elemento de cada vez do sub-vetor da direita para o sub-vetor da esquerda, inserindo-o na posição correta de forma a manter o sub-vetor da esquerda ordenado
- Termina-se quando o sub-vetor da direita fica vazio

# Ordenação por Inserção



# Ordenação por Inserção (pseudocódigo)

Entrada: vetor  $v$  com  $n$  elementos (i.e.  $v[0..n-1]$ )

```
para i de 1 até n-1{
    Atual = v[i]
    j = i
    enquanto (j > 0) & (v[j-1] > atual){
        v[j] = v[j-1]
        j = j - 1
    }
    v[j] = atual
}
```

# Ordenação por Inserção - Análise

Pior caso: vetor em ordem inversa:

cada elemento da parte não ordenada tem que ser trocado com todos os elementos da parte ordenada

Para um vetor de  $n$  elementos:

- 2º elemento faz uma troca;
- 3º elemento faz duas trocas;
- . . .
- $n^{\circ}$  elemento faz  $n-1$  troca

$$\sum_{i=1}^{n-1} i = (n-1) \times (n/2) = \frac{(n^2 - n)}{2} = O(n^2)$$

# Ordenação por Seleção

Estratégia: seleciona o menor elemento do vetor, depois o segundo menor, depois o terceiro menor, e assim por diante

Em cada etapa F:

- Procura-se (sequencialmente) a posição M com o menor elemento guardado nas posições de F a N;
- Troca-se o valor guardado na posição F com o valor guardado na posição M (excepto se M for igual a F)

	8
	5
	2
	6
	9
	3
	1
	4
	0
	7



# Ordenação por Seleção (vetor na vertical)

Índice	início	passo 1	passo 2	passo 3	passo 4	passo 5	passo 6	passo 7
0	7	7	2	2	2	2	2	2
1	21	21	21	7	7	7	7	7
2	10	10	10	10	10	10	10	10
3	15	15	15	15	15	11	11	11
4	2	2	7	21	21	21	13	13
5	13	13	13	13	13	13	21	15
6	11	11	11	11	11	15	15	21

The diagram illustrates the selection sort algorithm on a vertical vector. The table shows the state of the array after each pass. The 'início' column represents the current position being filled. The 'passo' columns show the state of the array after each pass. Red numbers indicate the current minimum element being selected. Grey shading highlights the elements that have been placed in their final sorted positions. Arrows show the movement of the minimum element to its final position.

# Ordenação por Seleção (pseudocódigo)

Entrada: vetor  $v$  com  $n$  elementos (i.e.  $v[0..n-1]$ )

```
para i de 0 to N-2{
    menor = i
    para j de i+1 to N-1{
        if  $v[j] < v[menor]$ 
            menor = j
    }
    troca( $v[i], v[menor]$ )
}
```

# Ordenação por Seleção - Análise

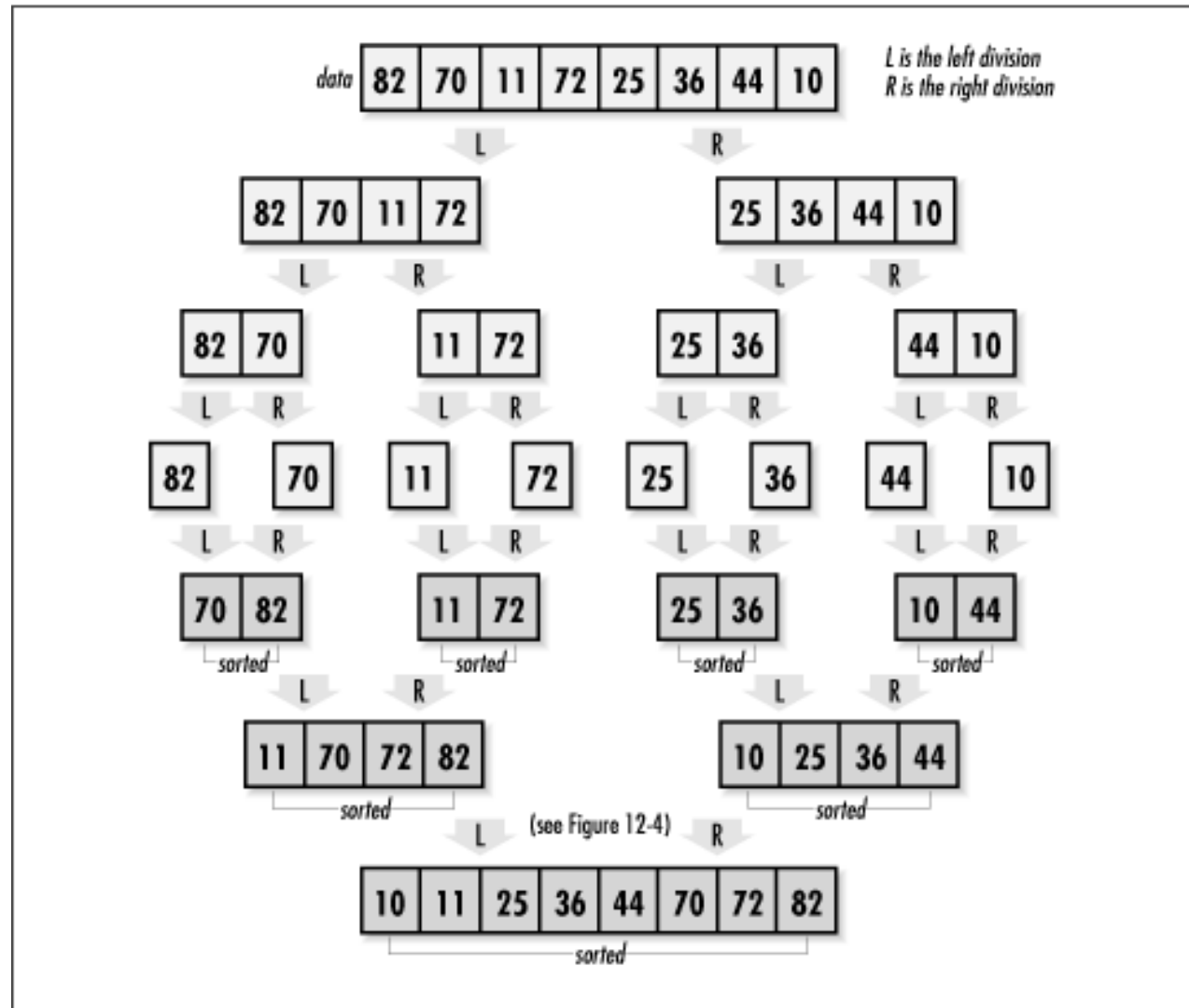
Cada posição é comparada com todas as posições seguintes

Para um vetor de  $n$  elementos:

- 1º elemento faz  $n-1$  comparações;
- 2º elemento faz  $n-2$  comparações;
- . . .
- $(n-1)^\circ$  elemento faz 1 comparação

$$\sum_{i=1}^{n-1} i = (n-1) \times (n/2) = \frac{(n^2 - n)}{2} = O(n^2)$$

# Merge Sort



# HeapSort

Em um *max heap* de  $n$  elementos, troca-se o 1º e o  $n$ º elementos.

O 1º elemento é o maior de todos e deve ocupar a última posição em um vetor ordenado.

O  $n$ º elemento pode ser descartado. O max-heap fica com  $n-1$  elementos.

Todos os elementos do novo vetor formam um *max-heap*, exceto o primeiro

Aplica-se *max-heapify*( $H, 1$ ) para posicionar o primeiro elemento adequadamente.

Exemplo:  $v = [16, 14, 10, 8, 7, 9, 3, 2, 4, 1]$

Complexidade:  $O(n \log_2 n)$ :

$n$  trocas; para cada troca, um *max-heapify* com complexidade  $O(n \log_2 n)$

C  
o  
m  
p  
a  
r  
a  
t  
i  
v  
o

	Worst case	Average case	Best case	Extra space	Stable;
<a href="#"><u>BubbleSort</u></a>	$O(n^2)$	$O(n^2)?$	$O(n)$	$O(1)$	yes
<a href="#"><u>SelectionSort</u></a>	$O(n^2)$	$O(n^2)$	$O(n^2)$	$O(1)$	No (i
<a href="#"><u>InsertionSort</u></a>	$O(n^2)$	$O(n^2)$	$O(n)$	$O(1)$	yes
<a href="#"><u>BitonicSort</u></a>	$O(n \log^2 n)$	$O(n \log^2 n)?$	?	$O(1)?$	?
<a href="#"><u>ShellSort</u></a>	$O(n^2)$	$O(n \log n)?$	$O(n)$	$O(1)$	no
<a href="#"><u>QuickSort</u></a>	$O(n^2)$	$O(n \log n)$	$O(n \log n)$	$O(\log n)$	no
<a href="#"><u>HeapSort</u></a>	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(1)$	no
<a href="#"><u>SmoothSort</u></a>	$O(n \log n)$	$O(n \log n)?$	$O(n)$	$O(1)$	no
<a href="#"><u>MergeSort</u></a>	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$	$O(n)$	yes
<a href="#"><u>TimSort</u></a>	$O(n \log n)$	$O(n \log n)?$	$O(n)$	$O(n)$	yes
<a href="#"><u>CountingSort</u></a>	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(n+k)$	yes
<a href="#"><u>RadixSort</u></a>	$O(n+k)$	$O(n+k)$	$O(n+k)$	$O(n+k)$	yes
<a href="#"><u>BucketSort</u></a>	$O(n^2)$	$O(n+k)$	?????	$O(n \cdot k)$ or $O(n+k)$	?
<a href="#"><u>BogoSort</u></a>	unbounded	$O(n!)$	$O(n)$	$O(1)$	no
<a href="#"><u>SlowSort</u></a>	$O(n^{(\log n)})$	$O(n^{(\log n)})$	$O(n^{(\log n)})$	$O(1)$	yes
<a href="#"><u>QuantumBogoSort</u></a>	$O(1)$	$O(1)$	$O(1)$	$O(0)$	no

# Simuladores de Algoritmos de Ordenação

<https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html>

<https://www.toptal.com/developers/sorting-algorithms>

<https://visualgo.net/bn/sorting>