

# Métodos de Ordenação

---

# Ordenação

---

- O problema de ordenação é um dos mais básicos em computação.
- Exemplos de aplicações:
  - Criação de rankings
  - Ordenação de listas
  - Otimizar sistemas de busca

# Ordenação

---

- Definição:
  - **Ordenação** é o processo de reorganizar um conjunto de dados em uma ordem ascendente ou descendente, segundo algum critério.
  - O objetivo da **ordenação** é facilitar a recuperação posterior de itens do conjunto ordenado.
  - A maioria dos **métodos de ordenação** é baseada em comparações das itens.

# Ordenação

---

- **Métodos de ordenação simples:**
  - Adequados para pequenos conjuntos de dados.
  - Lógica simples e de fácil implementação.
  - Tempo de execução maior.
- **Métodos de ordenação sofisticados:**
  - Adequados para grandes conjuntos de dados.
  - Lógica mais sofisticada e implementação mais trabalhosa.
  - Tempo de execução menor.

# Ordenação

---

- **Métodos de ordenação simples:**
  - Bubble Sort
  - Selection Sort
  - Insertion Sort
- **Métodos de ordenação sofisticados:**
  - Quicksort
  - Mergesort

# Ordenação

---

- Tempo de Execução:
  - A eficiência de tempo de execução é calculada pelo **número de operações críticas efetuadas.**
  - São consideradas **Operações Críticas:**
    - ✓ comparação de itens
    - ✓ troca de posição entre dois itens

# Ordenação

---

- Veremos alguns algoritmos e técnicas de ordenação.
  - Cada um apresenta diferentes soluções para o mesmo problema: **ordenar um conjunto de dados**.
- Por convenção, vamos considerar listas de números.
  - Porém, sabe-se que esses algoritmos podem ser aplicados para outros tipos de estruturas de dados sequenciais.

# Bubble Sort

*Ordenação por bolha*



# Bubble Sort

---

- Um dos algoritmos de ordenação mais lento
- Implementação simples
- Recomendado para conjuntos pequenos de dados
- Funciona através da comparação de itens dois a dois
  - Consiste em “Borbulhar” o maior item para o final da lista

# Bubble Sort

---

- Procedimento:
  - Compare o primeiro item com o segundo e troque-os se o primeiro for maior que o segundo
  - Compare o segundo item com o terceiro e troque-os se o segundo for maior que o terceiro
  - Compare o terceiro item com o quarto e troque-os se o terceiro for maior que o quarto
  - E assim sucessivamente ...
- Ao final desses passos, o maior item estará no final da lista!
  - Agora precisamos repetir todo esse processo  $N$  vezes.

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 1ª Iteração:  
[5, 3, 2, 1, 90, 6]      compara os dois primeiros itens

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 1ª Iteração:
  - [5, 3, 2, 1, 90, 6]
  - [3, 5, 2, 1, 90, 6]      5 é maior que 3, então troca

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 1ª Iteração:
  - [5, 3, 2, 1, 90, 6]
  - [3, 5, 2, 1, 90, 6]      compara com o próximo item

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]

- 1ª Iteração:

[5, 3, 2, 1, 90, 6]

[3, 5, 2, 1, 90, 6]

[3, 2, 5, 1, 90, 6]

5 é maior que 2, então troca

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 1ª Iteração:
  - [5, 3, 2, 1, 90, 6]
  - [3, 5, 2, 1, 90, 6]
  - [3, 2, 5, 1, 90, 6] compara com o próximo item



# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 1ª Iteração:
  - [5, 3, 2, 1, 90, 6]
  - [3, 5, 2, 1, 90, 6]
  - [3, 2, 5, 1, 90, 6]
  - [3, 2, 1, 5, 90, 6] 5 é maior que 1, então troca

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 1ª Iteração:
  - [5, 3, 2, 1, 90, 6]
  - [3, 5, 2, 1, 90, 6]
  - [3, 2, 5, 1, 90, 6]
  - [3, 2, 1, 5, 90, 6] compara com o próximo item

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 1ª Iteração:
  - [5, 3, 2, 1, 90, 6]
  - [3, 5, 2, 1, 90, 6]
  - [3, 2, 5, 1, 90, 6]
  - [3, 2, 1, 5, 90, 6]
  - [3, 2, 1, 5, 90, 6] 5 não é maior que 90, então não troca

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 1ª Iteração:
  - [5, 3, 2, 1, 90, 6]
  - [3, 5, 2, 1, 90, 6]
  - [3, 2, 5, 1, 90, 6]
  - [3, 2, 1, 5, 90, 6]
  - [3, 2, 1, 5, 90, 6] compara com o próximo item

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]

- 1ª Iteração:

[5, 3, 2, 1, 90, 6]

[3, 5, 2, 1, 90, 6]

[3, 2, 5, 1, 90, 6]

[3, 2, 1, 5, 90, 6]

[3, 2, 1, 5, 90, 6]

[3, 2, 1, 5, 6, 90] 90 é maior que 6, então troca

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 1ª Iteração:
  - [5, 3, 2, 1, 90, 6]
  - [3, 5, 2, 1, 90, 6]
  - [3, 2, 5, 1, 90, 6]
  - [3, 2, 1, 5, 90, 6]
  - [3, 2, 1, 5, 90, 6]
  - [3, 2, 1, 5, 6, 90] Ao final da 1ª iteração, o maior estará no final
- *Agora temos que repetir o processo novamente*

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 2ª Iteração:  
[3, 2, 1, 5, 6, 90] compara os dois primeiros itens

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 2ª Iteração:  
[3, 2, 1, 5, 6, 90]  
[2, 3, 1, 5, 6, 90] 3 é maior que 2, então troca



# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 2ª Iteração:  
[3, 2, 1, 5, 6, 90]  
[2, 3, 1, 5, 6, 90] compara com o próximo item

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 2ª Iteração:
  - [3, 2, 1, 5, 6, 90]
  - [2, 3, 1, 5, 6, 90]
  - [2, 1, 3, 5, 6, 90] 3 é maior que 1, então troca

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 2ª Iteração:
  - [3, 2, 1, 5, 6, 90]
  - [2, 3, 1, 5, 6, 90]
  - [2, 1, 3, 5, 6, 90] compara com o próximo item

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 2ª Iteração:
  - [3, 2, 1, 5, 6, 90]
  - [2, 3, 1, 5, 6, 90]
  - [2, 1, 3, 5, 6, 90]
  - [2, 1, 3, 5, 6, 90] 3 não é maior que 5, então não troca

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 2ª Iteração:
  - [3, 2, 1, 5, 6, 90]
  - [2, 3, 1, 5, 6, 90]
  - [2, 1, 3, 5, 6, 90]
  - [2, 1, 3, 5, 6, 90] compara com o próximo item

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 2ª Iteração:
  - [3, 2, 1, 5, 6, 90]
  - [2, 3, 1, 5, 6, 90]
  - [2, 1, 3, 5, 6, 90]
  - [2, 1, 3, 5, 6, 90] 5 não é maior que 6, então não troca

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 2ª Iteração:
  - [3, 2, 1, 5, 6, 90]
  - [2, 3, 1, 5, 6, 90]
  - [2, 1, 3, 5, 6, 90]
  - [2, 1, 3, 5, 6, 90] Ao final da 2ª iteração, o segundo maior estará no seu lugar
- *Agora temos que repetir o processo novamente*

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 3ª Iteração:  
[2, 1, 3, 5, 6, 90] compara os dois primeiros itens



# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 3ª Iteração:  
[2, 1, 3, 5, 6, 90]  
[1, 2, 3, 5, 6, 90] 2 é maior que 1, então troca

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 3ª Iteração:  
[2, 1, 3, 5, 6, 90]  
[1, 2, 3, 5, 6, 90] compara com o próximo item

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 3ª Iteração:
  - [2, 1, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90] 2 não é maior que 3, então não troca

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 3ª Iteração:
  - [2, 1, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90] compara com o próximo item

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 3ª Iteração:
  - [2, 1, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90] 3 não é maior que 5, então não troca

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 3ª Iteração:
  - [2, 1, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90] Ao final da 3ª iteração, o terceiro maior estará no seu lugar
- *Agora temos que repetir o processo novamente*

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 4ª Iteração:  
[1, 2, 3, 5, 6, 90] compara os dois primeiros itens

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 4ª Iteração:  
[1, 2, 3, 5, 6, 90]  
[1, 2, 3, 5, 6, 90] 1 não é maior que 2, então não troca



# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 4ª Iteração:  
[1, 2, 3, 5, 6, 90]  
[1, 2, 3, 5, 6, 90] compara com o próximo item

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 4ª Iteração:
  - [1, 2, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90] 2 não é maior que 3, então não troca

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 4ª Iteração:
  - [1, 2, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90] Ao final da 4ª iteração, o quarto maior estará no seu lugar
- *Agora temos que repetir o processo novamente*

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 5ª Iteração:  
[1, 2, 3, 5, 6, 90] compara os dois primeiros itens

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 5ª Iteração:  
[1, 2, 3, 5, 6, 90]  
[1, 2, 3, 5, 6, 90] 1 não é maior que 2, então não troca

# Bubble Sort

---

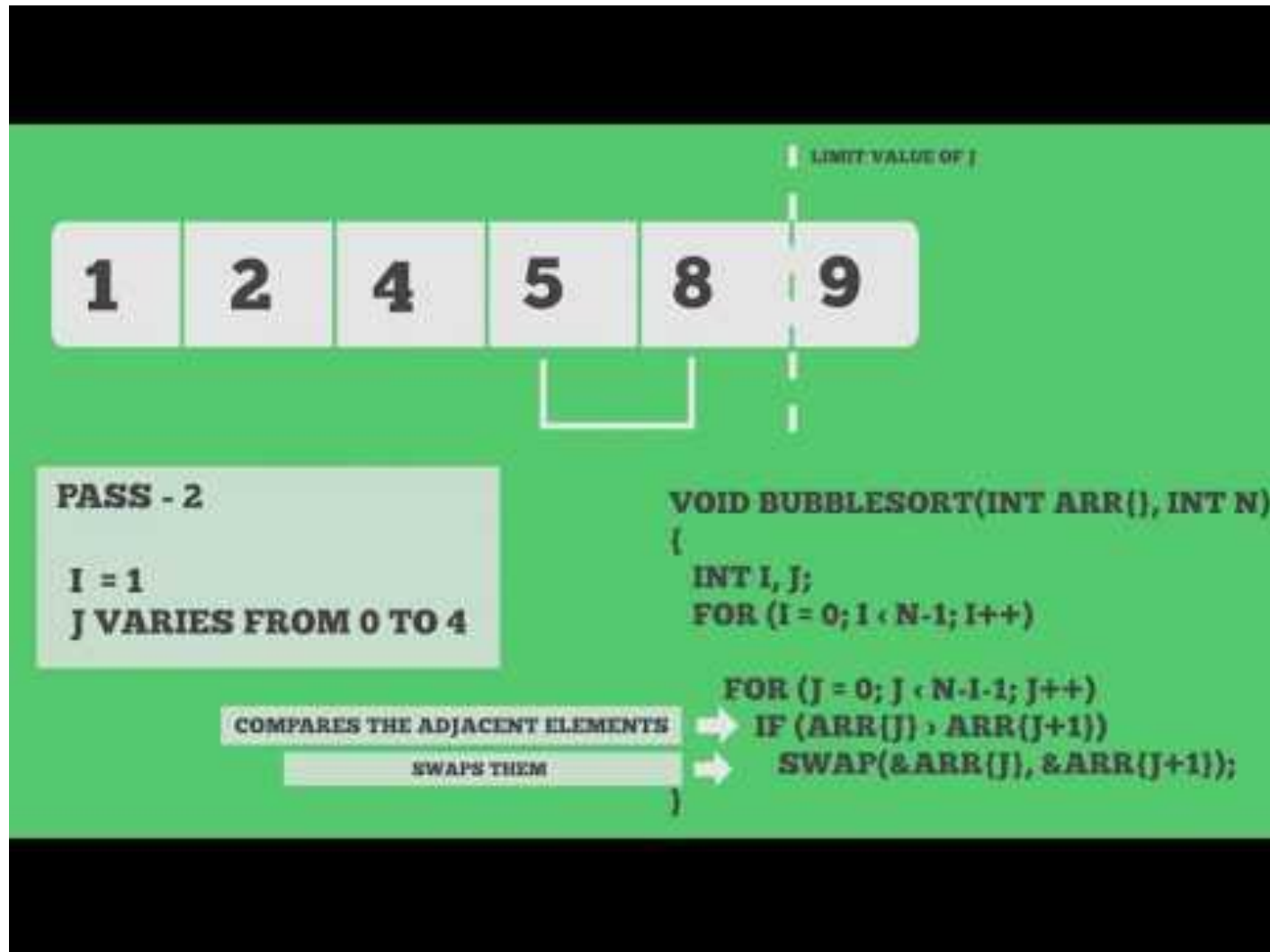
- Exemplo: [5, 3, 2, 1, 90, 6]
- 5ª Iteração:
  - [1, 2, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90] Ao final da 5ª iteração, o quinto maior estará no seu lugar

# Bubble Sort

---

- Exemplo: [5, 3, 2, 1, 90, 6]
- 5ª Iteração:
  - [1, 2, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90]
  - [1, 2, 3, 5, 6, 90] Ao final da 5ª iteração, o quinto maior estará no seu lugar
  - [1, 2, 3, 5, 6, 90] O primeiro item também já ficou em seu lugar, e portanto a lista está ordenada

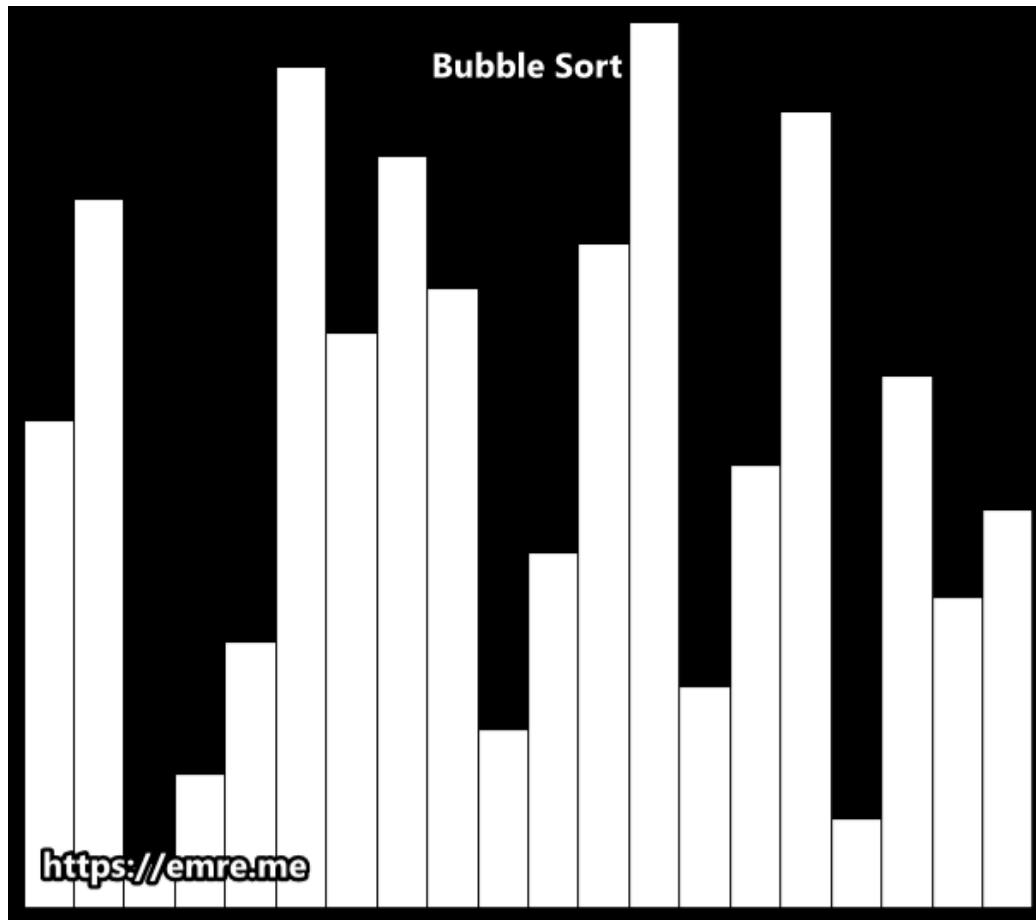
# Bubble Sort





# Bubble Sort

---



# Bubble Sort

---

- Implementação em Python

```
def bubble_sort(lista):  
    n = len(lista)  
    for i in range(0, n-1):  
        for j in range(0, n-1-i):  
            if lista[j] > lista[j+1]: # compara itens  
                aux = lista[j]        # faz a troca  
                lista[j] = lista[j+1]  
                lista[j+1] = aux
```

# Bubble Sort

---

- **Desempenho:**
  - **Melhor Caso:**
    - Quando a lista já se encontra ordenada
    - Realiza comparações, mas nenhuma troca ocorre
  - **Pior caso:**
    - Quando a lista se encontra na ordem inversa a desejada.
    - A cada varredura apenas um item será colocada em sua posição definitiva

# Selection Sort

*Ordenação por Seleção*

# Selection Sort

---

- Algoritmo de ordenação lento
- Implementação simples
- Recomendado para conjuntos pequenos de dados
- Procedimento:
  - Encontre o menor item da lista e troque com a primeira posição
  - Encontre o segundo menor item da lista e troque com a segunda posição
  - E assim por diante, até que a lista esteja ordenada

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

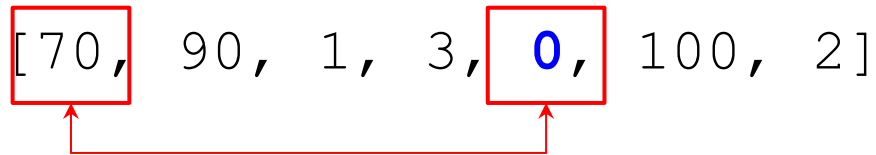
[70, 90, 1, 3, 0, 100, 2]

- 1ª iteração:
  - Encontramos o menor item: zero

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]



- 1ª iteração:
  - Encontramos o menor item: zero
  - Trocamos com a primeira posição: 70



# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 90, 1, 3, 70, 100, 2]

- 1ª iteração:
  - Encontramos o menor item: zero
  - Trocamos com a primeira posição: 70
  - O zero já está em sua posição correta

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 90, 1, 3, 70, 100, 2]

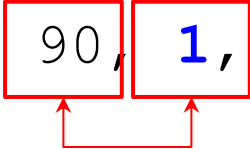
- 2ª iteração:
  - Encontramos o segundo menor item: 1

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 90, 1, 3, 70, 100, 2]



- 2ª iteração:
  - Encontramos o segundo menor item: 1
  - Trocamos com a segunda posição: 90

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 1, 90, 3, 70, 100, 2]

- 2ª iteração:
  - Encontramos o segundo menor item: 1
  - Trocamos com a segunda posição: 90
  - O 1 já está em sua posição correta

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 1, 90, 3, 70, 100, 2]

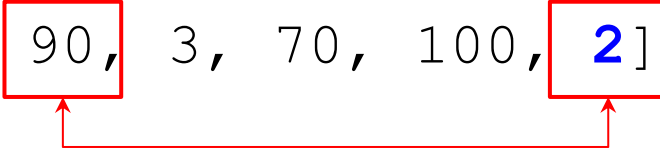
- 3ª iteração:
  - Encontramos o terceiro menor item: 2

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 1, 90, 3, 70, 100, 2]



- 3ª iteração:
  - Encontramos o terceiro menor item: 2
  - Trocamos com a terceira posição: 90

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 1, 2, 3, 70, 100, 90]

- 3ª iteração:
  - Encontramos o terceiro menor item: 2
  - Trocamos com a terceira posição: 90
  - O número 2 já está em sua posição

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 1, 2, 3, 70, 100, 90]

- 4ª iteração:
  - Encontramos o próximo menor item: 3

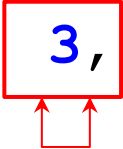


# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 1, 2, 3, 70, 100, 90]



- 4ª iteração:
  - Encontramos o próximo menor item: 3
  - O 3 já está no seu lugar, ele será trocado por ele mesmo

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 1, 2, 3, 70, 100, 90]

- 4ª iteração:
  - Encontramos o próximo menor item: 3
  - O 3 já está no seu lugar, ele será trocado por ele mesmo
  - O número 3 continua em sua posição

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 1, 2, 3, 70, 100, 90]

- 5ª iteração:
  - Encontramos o próximo menor item: 70

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 1, 2, 3, 70, 100, 90]

- 5ª iteração:
  - Encontramos o próximo menor item: 70
  - O 70 já está no seu lugar, ele será trocado por ele mesmo

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 1, 2, 3, 70, 100, 90]

- 5ª iteração:
  - Encontramos o próximo menor item: 70
  - O 70 já está no seu lugar, ele será trocado por ele mesmo
  - O número 70 continua em sua posição

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 1, 2, 3, 70, 100, 90]

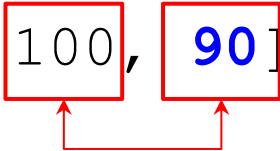
- 6ª iteração:
  - Encontramos o próximo menor item: 90
  - O 90 será trocado de posição
  - O número 90 já está em sua posição

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 1, 2, 3, 70, 100, 90]



- 6ª iteração:
  - Encontramos o próximo menor item: 90
  - O 90 será trocado de posição

# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 1, 2, 3, 70, 90, 100]

- 6ª iteração:
  - Encontramos o próximo menor item: 90
  - O 90 será trocado de posição
  - O número 90 já está em sua posição



# Selection Sort

---

- Exemplo: [70, 90, 1, 3, 0, 100, 2]

[0, 1, 2, 3, 70, 90, 100]

- Por fim, quando restar apenas o último número, ele já estará em sua posição correta.

# Selection Sort

---

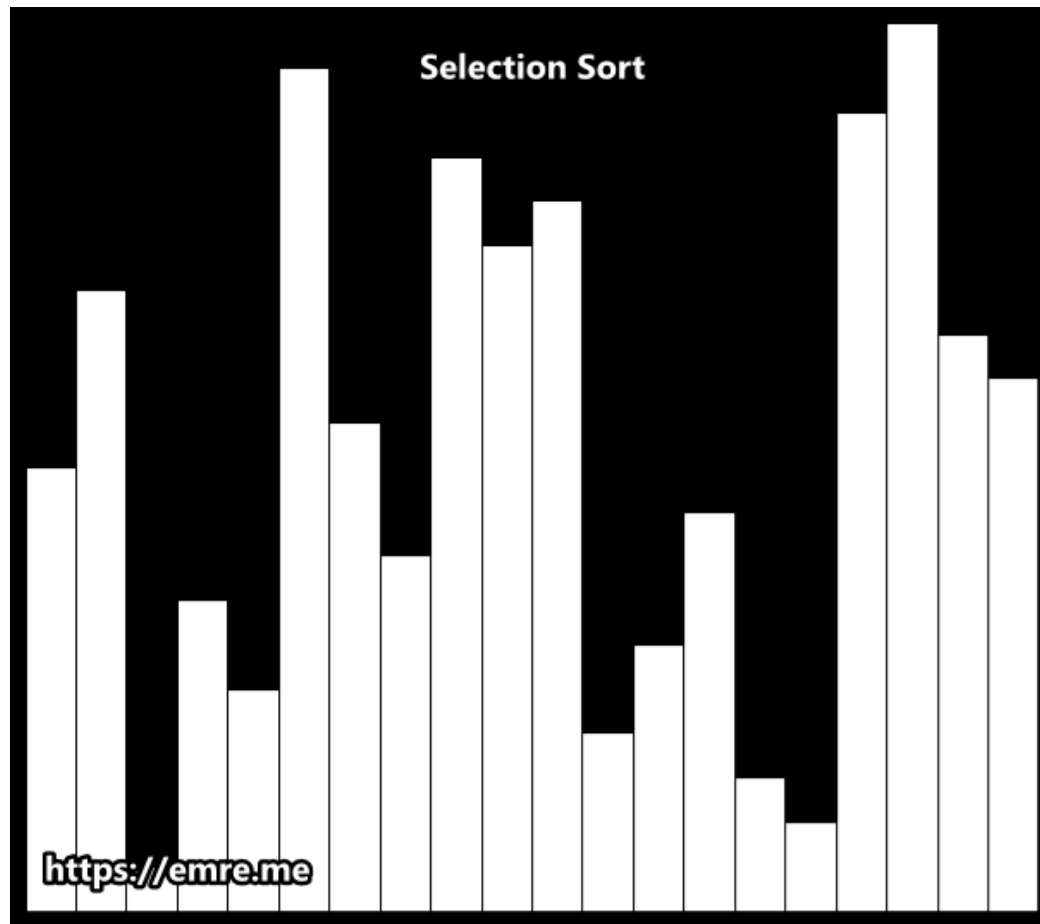
**SELECTION SORT**

GeeksforGeeks

A computer science portal for geeks

# Selection Sort

---



# Selection Sort

---

- Implementação em Python

```
def selection_sort(lista):  
    n = len(lista)  
    for i in range(0, n-1):  
        menor = i  
        for j in range(i + 1, n): # encontra índice do menor  
            if lista[j] < lista[menor]:  
                menor = j  
        aux = lista[i] # realiza a troca  
        lista[i] = lista[menor]  
        lista[menor] = aux
```

# Selection Sort

---

- Desempenho:
  - Para uma lista de tamanho  $N$ , precisamos encontrar o menor valor  **$N-1$  vezes**.
  - Essa operação acontece  **$N$  vezes**
  - Tempo total:  $(N - 1) * N$ , que é aproximadamente  $N^2$
  - Melhor que o *bubble sort*, pois faz menos trocas
  - Desvantagem: Se a lista inicial já estiver ordenada não teremos vantagem, pois o número de comparações continuará o mesmo.

# Insertion Sort

*Ordenação por inserção*

# Insertion Sort

---

- Algoritmo de ordenação lento
- Implementação simples
- Recomendado para conjuntos pequenos de dados
- Procedimento:
  - Comparar um item com os itens à sua esquerda e, se esse item for menor, trocar esses itens de posição
  - Essas comparações e trocas só devem parar quando o item for maior que o item à sua esquerda ou quando o item estiver na primeira posição da lista

# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]



# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]
- 1ª iteração: Começamos analisando o segundo item.

[70, 90, 3, 0, 100, 2]

# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]
- 1ª iteração: Começamos analisando o segundo item.

[70, 90, 3, 0, 100, 2] compara o segundo item com os anteriores

# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]
- 1ª iteração: Começamos analisando o segundo item.

[70, 90, 3, 0, 100, 2] compara o segundo item com os anteriores

[70, 90, 3, 0, 100, 2] 90 não é menor que 70, então não troca

# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]
- 2ª iteração: Agora analisamos o terceiro item.

[70, 90, 3, 0, 100, 2]

# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]
- 2ª iteração: Agora analisamos o terceiro item.

[70, 90, 3, 0, 100, 2] compara o terceiro item com os anteriores

# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]
- 2ª iteração: Agora analisamos o terceiro item.

[70, 90, 3, 0, 100, 2]

[70, 3, 90, 0, 100, 2] 3 é menor que 90, então troca

# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]
- 2ª iteração: Agora analisamos o terceiro item.

[70, 90, 3, 0, 100, 2]

[70, 3, 90, 0, 100, 2]

[3, 70, 90, 0, 100, 2] 3 é menor que 70, então troca

# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]
- 3ª iteração: Agora analisamos o quarto item.

[3, 70, 90, 0, 100, 2]



# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]
- 3ª iteração: Agora analisamos o quarto item.

[3, 70, 90, 0, 100, 2] compara o quarto item com os anteriores

# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]
- 3ª iteração: Agora analisamos o quarto item.

[3, 70, 90, 0, 100, 2]

[3, 70, 0, 90, 100, 2] zero é menor que 90, então troca

# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]
- 3ª iteração: Agora analisamos o quarto item.

[3, 70, 90, 0, 100, 2]

[3, 70, 0, 90, 100, 2]

[3, 0, 70, 90, 100, 2] zero é menor que 70, então troca

# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]
- 3ª iteração: Agora analisamos o quarto item.

[3, 70, 90, 0, 100, 2]

[3, 70, 0, 90, 100, 2]

[3, 0, 70, 90, 100, 2]

[0, 3, 70, 90, 100, 2] zero é menor que 3, então troca

# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]
- 4ª iteração: Agora analisamos o quinto item.

[0, 3, 70, 90, **100**, 2]

# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]
- 4ª iteração: Agora analisamos o quinto item.

[0, 3, 70, 90, 100, 2] compara o quinto item com os anteriores

# Insertion Sort

---

- Exemplo: [ 70 , 90 , 3 , 0 , 100 , 2 ]
- 4ª iteração: Agora analisamos o quinto item.

[ 0 , 3 , 70 , 90 , 100 , 2 ] compara o quinto item com os anteriores

[ 0 , 3 , 70 , 90 , 100 , 2 ] 100 não é menor que 90, então não troca

# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]
- 5ª iteração: Agora analisamos o sexto item.

[0, 3, 70, 90, 100, **2**]



# Insertion Sort

---

- Exemplo: [70, 90, 3, 0, 100, 2]
- 5ª iteração: Agora analisamos o sexto item.

[0, 3, 70, 90, 100, 2] compara o sexto item com os anteriores

# Insertion Sort

---

- Exemplo: [ 70 , 90 , 3 , 0 , 100 , 2 ]
- 5ª iteração: Agora analisamos o sexto item.

[ 0 , 3 , 70 , 90 , 100 , 2 ]

[ 0 , 3 , 70 , 90 , 2 , 100 ] 2 é menor que 100, então troca

# Insertion Sort

---

- Exemplo: [ 70 , 90 , 3 , 0 , 100 , 2 ]
- 5ª iteração: Agora analisamos o sexto item.

[ 0 , 3 , 70 , 90 , 100 , 2 ]

[ 0 , 3 , 70 , 90 , 2 , 100 ]

[ 0 , 3 , 70 , 2 , 90 , 100 ]    2 é menor que 90, então troca

# Insertion Sort

---

- Exemplo: [ 70 , 90 , 3 , 0 , 100 , 2 ]
- 5ª iteração: Agora analisamos o sexto item.

[ 0 , 3 , 70 , 90 , 100 , 2 ]

[ 0 , 3 , 70 , 90 , 2 , 100 ]

[ 0 , 3 , 70 , 2 , 90 , 100 ]

[ 0 , 3 , 2 , 70 , 90 , 100 ]    2 é menor que 70, então troca

# Insertion Sort

---

- Exemplo: [ 70 , 90 , 3 , 0 , 100 , 2 ]
- 5ª iteração: Agora analisamos o sexto item.

[ 0 , 3 , 70 , 90 , 100 , 2 ]

[ 0 , 3 , 70 , 90 , 2 , 100 ]

[ 0 , 3 , 70 , 2 , 90 , 100 ]

[ 0 , 3 , 2 , 70 , 90 , 100 ]

[ 0 , 2 , 3 , 70 , 90 , 100 ]    2 é menor que 3, então troca

# Insertion Sort

---

- Exemplo: [ 70 , 90 , 3 , 0 , 100 , 2 ]
- 5ª iteração: Agora analisamos o sexto item.

[ 0 , 3 , 70 , 90 , 100 , 2 ]

[ 0 , 3 , 70 , 90 , 2 , 100 ]

[ 0 , 3 , 70 , 2 , 90 , 100 ]

[ 0 , 3 , 2 , 70 , 90 , 100 ]

[ 0 , 2 , 3 , 70 , 90 , 100 ]

[ 0 , 2 , 3 , 70 , 90 , 100 ]    2 não é menor que 3, então não troca

# Insertion Sort

---

- Exemplo: [ 70 , 90 , 3 , 0 , 100 , 2 ]
- 5ª iteração: Agora analisamos o sexto item.

[ 0 , 3 , 70 , 90 , 100 , 2 ]

[ 0 , 3 , 70 , 90 , 2 , 100 ]

[ 0 , 3 , 70 , 2 , 90 , 100 ]

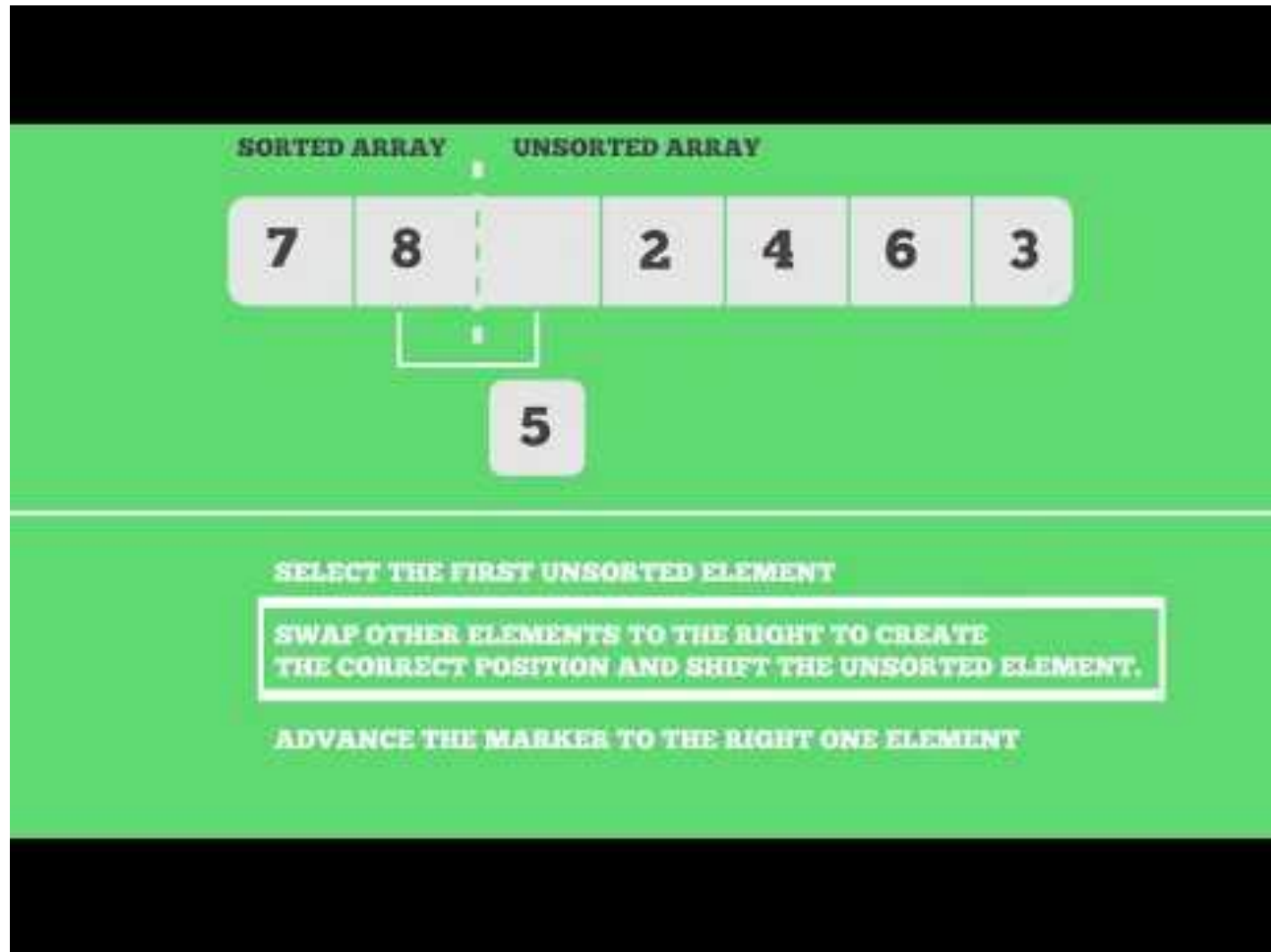
[ 0 , 3 , 2 , 70 , 90 , 100 ]

[ 0 , 2 , 3 , 70 , 90 , 100 ]

[ 0 , 2 , 3 , 70 , 90 , 100 ] 2 não é menor que 3, então não troca

Ao final, a lista está ordenada.

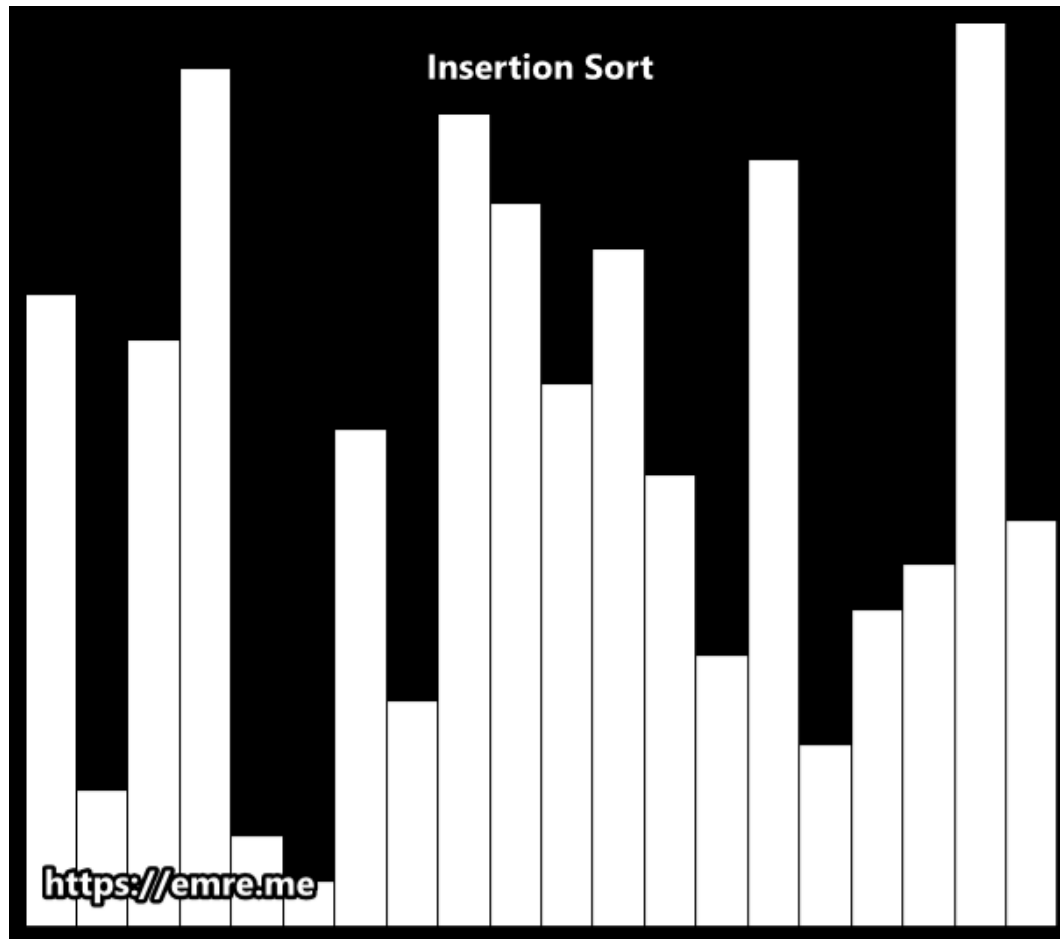
# Insertion Sort





# Insertion Sort

---



# Insertion Sort

---

- Implementação em Python

```
def insertion_sort(lista):  
    n = len(lista)  
    for i in range(1, n):  
        j = i  
        while j > 0 and lista[j] < lista[j-1]:  
            aux = lista[j]  
            lista[j] = lista[j-1]  
            lista[j-1] = aux  
            j -= 1
```

# Insertion Sort

---

- **Desempenho:**
  - **Melhor Caso:**
    - Ocorre quando a lista já está ordenada.
    - Todos já estão em suas devidas posições.
  - **Pior caso:**
    - Ocorre quando a lista está ordenada em ordem reversa
    - Para cada item será necessário percorrer a lista toda à esquerda, trocando os itens até encaixar o item na primeira posição.

# Comparação dos Métodos

---

- O Insertion Sort apresenta o melhor desempenho entre esses 3 algoritmos.
  - O Insertion Sort efetua menos comparações, pois nem sempre o item a ser inserido de forma ordenada deve ir até o início da lista.
  - Isso só acontece no pior dos casos, em que a lista está ordenada em ordem reversa.