

**SERVIÇO NACIONAL DE APRENDIZAGEM INDUSTRIAL**

**PWBE: Revisão inicial**

**Nicolas Vilela Barros da Costa Alves**

**Janeiro – 2025**

**Merge sort**

Basicamente um método usado para ordenar uma sequência de números. Este método consiste em dividir e conquistar, pois ao ter uma sequência de números você só tem aplicar certas separações e misturar os números para serem uma sequência novamente.

Considerando o array:

 [38, 27, 43, 3, 9, 82, 10]

*Passo 1: Dividir*

1. Divida o array em duas metades:
   * Esquerda: [38, 27, 43, 3]
   * Direita: [9, 82, 10]
   * Continue dividindo até que cada subarray tenha apenas um elemento:
     + Esquerda: [38, 27], [43, 3]
     + Direita: [9, 82], [10]

*Passo 2: Ordenar*

1. Ordene cada subarray:
   * [38, 27] se torna [27, 38]
   * [43, 3] se torna [3, 43]
   * [9, 82] se torna [9, 82]
   * [10] permanece [10]

*Passo 3: Intercalar*

Agora, intercale as metades ordenadas:

* + [27, 38] e [3, 43] se tornam [3, 27, 38, 43]
  + [9, 82] e [10] se tornam [9, 10, 82]
  + Finalmente, intercale as duas metades ordenadas:
    - [3, 27, 38, 43] e [9, 10, 82] se tornam [3, 9, 10, 27, 38, 43, 82]

Assim, o array ordenado é: [3, 9, 10, 27, 38, 43, 82].

**Insertion Sort**

Este método funciona com comparações

Considerando o array: [38, 27, 43, 3, 9, 82, 10]

*Passo 1: Iniciar com o segundo elemento (27)*

* O primeiro elemento (38) já está "ordenado". Agora, compare o segundo elemento (27) com o primeiro (38).
* Como 27 é menor que 38, troque as posições:
  + Array: [27, 38, 43, 3, 9, 82, 10]

*Passo 2: Incluir o próximo elemento (43)*

* Compare o terceiro elemento (43) com o anterior (38).
* Como 43 é maior que 38, ele já está na posição correta.

*Passo 3: Incluir o próximo elemento (3)*

* Compare o quarto elemento (3) com os anteriores.
* 3 é menor que 43, então troque 3 com 43.
* Depois, compare 3 com 38 e troque-os.
* Finalmente, compare 3 com 27 e troque-os.
  + Array: [3, 27, 38, 43, 9, 82, 10]

*Passo 4: Incluir o próximo elemento (9)*

* Compare o quinto elemento (9) com os anteriores.
* 9 é menor que 43, troque com 43.
* Compare 9 com 38, troque com 38.
* Compare 9 com 27, troque com 27.
* Finalmente, compare 9 com 3. Como 9 é maior que 3, ele vai na posição correta.
  + Array: [3, 9, 27, 38, 43, 82, 10]

*Passo 5: Incluir o próximo elemento (82)*

* 82 é maior que todos os elementos anteriores, então ele permanece no final.
  + Array: [3, 9, 27, 38, 43, 82, 10]

*Passo 6: Incluir o próximo elemento (10)*

* Compare 10 com 82, 43, 38, 27, 9, e 3.
* 10 vai para a posição correta entre 9 e 27.
  + Array: [3, 9, 10, 27, 38, 43, 82]

Agora o array está ordenado: [3, 9, 10, 27, 38, 43, 82].

**Bubble sort**

O **Bubble Sort** (ou **Ordenação por Troca**) é um algoritmo simples de ordenação que funciona repetidamente comparando elementos adjacentes e trocando-os se estiverem na ordem errada. Ele continua fazendo isso até que o array esteja ordenado. O algoritmo é chamado de "Bubble" (bolha) porque os maiores elementos "sobem" (ou "flutuam") para o final da lista, como bolhas de ar na água.

**Busca Linear**

É uma pesquisa que em base você examina cada registro sequencialmente. Sendo uma ótima opção para achar dados em um array pequeno. Um exemplo que podemos ter é este código em python:

def busca\_linear(lista, objetivo):

 for i in range(len(lista)):

        if lista[i] == objetivo:

            return i  # Retorna o índice do objetivo se encontrar

    return -1  # Retorna -1 se o objetivo não estiver na lista

lista = [3, 5, 7, 9, 11]

objetivo = 7

resultado = busca\_linear(lista, objetivo)

if resultado != -1:

    print(f"O elemento {objetivo} está no índice {resultado}.")

else:

    print(f"O elemento {objetivo} não está na lista.")

O objetivo deste código é achar o 7 dentro da lista, sendo uma lista pequena é um resultado rápido, porém com listas grandes o resultado não seria eficiente.

**Busca binária**

Outro método de busca, porém bem mais utilizado no mundo, bem mais eficiente comparado com a busca linear que tem seu principal problema achar um número em uma lista grande. Já este método consiste em dividir a lista em metade para reduzir o escopo da procura, isso até achar o item desejado. Mas para isso a lista precisa estar ordenada em ordem crescente.

Exemplo de código:

def busca\_linear(lista, objetivo):

    for i in range(len(lista)):

        if lista[i] == objetivo:

            return i  # Retorna o índice do objetivo se encontrar

    return -1  # Retorna -1 se o objetivo não estiver na lista

lista = [3, 5, 7, 9, 11]

objetivo = 7

resultado = busca\_linear(lista, objetivo)

if resultado != -1:

    print(f"O elemento {objetivo} está no índice {resultado}.")

else:

    print(f"O elemento {objetivo} não está na lista.")

**Complexidade de código**

De maneira mais abrangente este método busca saber o tempo e eficiência do código, não se importando na onde a aplicação está sendo implementada, ou seja, a potência do hardware ou do software. Este regulamento sempre examina a aplicação de maneira completa.