# Algoritmos de ordenação e busca

[**Algoritmos de ordenação e busca**](#_i1h9pgxcvwi8)

**1**

[**Percorre o array procurando o menor elemento a partir da posição atual e troca o elemento atual com o menor elemento encontrado.**](#_v745ej9fd6e2) 1

[K-ésimo Maior Elemento: Dado um array de números e um valor k, encontre o k-ésimo maior elemento no array.](#_8bko2puzngfp) 2

[**Contagem de Inversões: Dado um array, determine o número de inversões, ou seja, quantos pares de elementos estão fora de ordem.**](#_b9hjvndyayal) 2

[Fusão de Intervalos: Dado um conjunto de intervalos, implemente um algoritmo para fundir intervalos sobrepostos.](#_a2b13271mv29) 3

[**Ordenação Estável: Peça para implementar um algoritmo de ordenação estável, onde a ordem relativa dos elementos iguais é preservada.**](#_uzb60kfuqjta) 4

[Ordenação por Contagem: Implemente um algoritmo de ordenação por contagem para ordenar um array de números inteiros dentro de um intervalo específico.](#_mu3wngcgkcih) 5

[**Busca Binária em Matriz: Dada uma matriz ordenada (linha e coluna), implemente um algoritmo de busca binária para encontrar um elemento específico.**](#_kk97lplg0st0) 6

[Encontrar Elemento Faltante: Dado um array contendo números de 1 a N, com um número faltando, implemente um algoritmo para encontrar o número ausente.](#_2grx1hj5yt9k) 7

[**Busca por Ponto Fixo: Dado um array de números distintos em ordem crescente, encontre um índice i em que o elemento no índice i seja igual a i.**](#_h306rsa7fjwg) 8

# 

## Percorre o array procurando o menor elemento a partir da posição atual e troca o elemento atual com o menor elemento encontrado.

Digite os números separados por espaço: 1 9 8 6 2 4 5 3

Array ordenado: [1, 2, 3, 4, 5, 6, 8, 9]

def selection\_sort(array):

n = len(array)

for i in range(n):

min\_index = i

for j in range(i+1, n):

if array[j] < array[min\_index]:

min\_index = j

array[i], array[min\_index] = array[min\_index], array[i]

# Solicita a entrada do usuário para o array de números

entrada = input("Digite os números separados por espaço: ")

numeros = list(map(int, entrada.split()))

# Chama a função de ordenação

selection\_sort(numeros)

# Exibe o array ordenado

print("Array ordenado:", numeros)

## K-ésimo Maior Elemento: Dado um array de números e um valor k, encontre o k-ésimo maior elemento no array.

Digite os números separados por espaço: 1 8 6 5 7 8 1 2 5 6 98

Digite o valor de k para encontrar o k-ésimo maior elemento: 5

O 5-ésimo maior elemento é: 6

def kth\_largest\_element(arr, k):

if k <= 0 or k > len(arr):

return None

arr.sort()

return arr[-k]

# Exemplo de uso

array = [3, 1, 7, 4, 2, 5, 6]

k = 3

print("O", k, "º maior elemento é:", kth\_largest\_element(array, k)) # Saída: 5

## Contagem de Inversões: Dado um array, determine o número de inversões, ou seja, quantos pares de elementos estão fora de ordem.

Digite os números separados por espaço: 3 2 1

Número de inversões: 3

def merge\_and\_count(arr, left, mid, right):

count = 0

left\_arr = arr[left:mid+1]

right\_arr = arr[mid+1:right+1]

i = j = 0

k = left

while i < len(left\_arr) and j < len(right\_arr):

if left\_arr[i] <= right\_arr[j]:

arr[k] = left\_arr[i]

i += 1

else:

arr[k] = right\_arr[j]

j += 1

count += (mid - i + 1) # Contabiliza inversões

k += 1

while i < len(left\_arr):

arr[k] = left\_arr[i]

i += 1

k += 1

while j < len(right\_arr):

arr[k] = right\_arr[j]

j += 1

k += 1

return count

def merge\_sort\_and\_count(arr, left, right):

count = 0

if left < right:

mid = (left + right) // 2

count += merge\_sort\_and\_count(arr, left, mid)

count += merge\_sort\_and\_count(arr, mid + 1, right)

count += merge\_and\_count(arr, left, mid, right)

return count

def count\_inversions(arr):

return merge\_sort\_and\_count(arr, 0, len(arr) - 1)

# Solicita a entrada do usuário para o array de números

entrada = input("Digite os números separados por espaço: ")

numeros = list(map(int, entrada.split()))

# Calcula o número de inversões no array usando merge sort

numero\_inversoes = count\_inversions(numeros)

# Exibe o número de inversões

print("Número de inversões:", numero\_inversoes)

## Fusão de Intervalos: Dado um conjunto de intervalos, implemente um algoritmo para fundir intervalos sobrepostos.

Digite o número de intervalos: 3

Digite o 1º intervalo (formato: início fim): 1 5

Digite o 2º intervalo (formato: início fim): 2 9

Digite o 3º intervalo (formato: início fim): 6 7

SAIDA Intervalos fundidos: [(1, 9)]

def merge\_intervals(intervals):

if not intervals:

return []

intervals.sort(key=lambda x: x[0])

merged\_intervals = [intervals[0]]

for interval in intervals[1:]:

if interval[0] <= merged\_intervals[-1][1]:

merged\_intervals[-1] = (merged\_intervals[-1][0], max(merged\_intervals[-1][1], interval[1]))

else:

merged\_intervals.append(interval)

return merged\_intervals

# Solicita a entrada do usuário para os intervalos

num\_intervalos = int(input("Digite o número de intervalos: "))

intervals = []

for i in range(num\_intervalos):

inicio, fim = map(int, input(f"Digite o {i+1}º intervalo (formato: início fim): ").split())

intervals.append((inicio, fim))

# Chama a função para fundir os intervalos sobrepostos

merged\_intervals = merge\_intervals(intervals)

# Exibe os intervalos fundidos

print("Intervalos fundidos:", merged\_intervals)

## Ordenação Estável: Peça para implementar um algoritmo de ordenação estável, onde a ordem relativa dos elementos iguais é preservada.

Digite os números separados por espaço: 1 5 9 8 7 5 6 2

Array ordenado: [1, 2, 5, 5, 6, 7, 8, 9]

def merge\_sort(arr):

if len(arr) <= 1:

return arr

mid = len(arr) // 2

left\_half = arr[:mid]

right\_half = arr[mid:]

left\_half = merge\_sort(left\_half)

right\_half = merge\_sort(right\_half)

return merge(left\_half, right\_half)

def merge(left, right):

merged = []

i = j = 0

while i < len(left) and j < len(right):

if left[i] <= right[j]:

merged.append(left[i])

i += 1

else:

merged.append(right[j])

j += 1

merged.extend(left[i:])

merged.extend(right[j:])

return merged

# Solicita a entrada do usuário para o array de números

entrada = input("Digite os números separados por espaço: ")

numeros = list(map(int, entrada.split()))

# Chama a função de ordenação estável (Merge Sort)

numeros\_ordenados = merge\_sort(numeros)

# Exibe o array ordenado

print("Array ordenado:", numeros\_ordenados)

## Ordenação por Contagem: Implemente um algoritmo de ordenação por contagem para ordenar um array de números inteiros dentro de um intervalo específico.

Digite os números separados por espaço: 1 2 5 7 1 2 5 4 6 3 5 2

Array ordenado: [1, 1, 2, 2, 2, 3, 4, 5, 5, 5, 6, 7]

def counting\_sort(arr):

# Encontra o valor máximo e mínimo no array

max\_value = max(arr)

min\_value = min(arr)

# Calcula o intervalo (range) entre o valor mínimo e máximo

interval = max\_value - min\_value + 1

# Inicializa o array de contagem com zeros

count\_array = [0] \* interval

# Conta a ocorrência de cada elemento no array original

for num in arr:

count\_array[num - min\_value] += 1

# Reconstrói o array ordenado usando as contagens

sorted\_array = []

for i, count in enumerate(count\_array):

sorted\_array.extend([i + min\_value] \* count)

return sorted\_array

# Solicita a entrada do usuário para o array de números

entrada = input("Digite os números separados por espaço: ")

numeros = list(map(int, entrada.split()))

# Chama a função de ordenação por contagem

numeros\_ordenados = counting\_sort(numeros)

# Exibe o array ordenado

print("Array ordenado:", numeros\_ordenados)

## Busca Binária em Matriz: Dada uma matriz ordenada (linha e coluna), implemente um algoritmo de busca binária para encontrar um elemento específico.

False

def binary\_search\_matrix(matrix, target):

if not matrix or not matrix[0]:

return False

rows = len(matrix)

cols = len(matrix[0])

# Começar na esquina superior direita da matriz

row = 0

col = cols - 1

while row < rows and col >= 0:

if matrix[row][col] == target:

return True

elif matrix[row][col] < target:

# Se o elemento atual for menor que o alvo, vá para a próxima linha

row += 1

else:

# Se o elemento atual for maior que o alvo, vá para a coluna anterior

col -= 1

return False

# Exemplo de uso

matrix = [

[1, 3, 5, 7],

[2, 4, 6, 8],

[3, 5, 7, 9]

]

target = 6

print(binary\_search\_matrix(matrix, target)) # Saída: True

## Encontrar Elemento Faltante: Dado um array contendo números de 1 a N, com um número faltando, implemente um algoritmo para encontrar o número ausente.

Digite os números separados por espaço (de 1 a N-1): 1 5 8 6 2 3 4

O número ausente é: 7

def find\_missing\_number(nums):

n = len(nums) + 1

expected\_sum = (n \* (n + 1)) // 2

actual\_sum = sum(nums)

missing\_number = expected\_sum - actual\_sum

return missing\_number

# Solicita a entrada do usuário para o array de números

entrada = input("Digite os números separados por espaço (de 1 a N-1): ")

numeros = list(map(int, entrada.split()))

# Chama a função para encontrar o número ausente

numero\_ausente = find\_missing\_number(numeros)

# Exibe o número ausente

print("O número ausente é:", numero\_ausente)

## Busca por Ponto Fixo: Dado um array de números distintos em ordem crescente, encontre um índice i em que o elemento no índice i seja igual a i.

Digite os números separados por espaço: 0 5 8 88 8 8

O ponto fixo está no índice 0.

def fixed\_point\_search(arr):

left, right = 0, len(arr) - 1

while left <= right:

mid = (left + right) // 2

if arr[mid] == mid:

return mid

elif arr[mid] < mid:

left = mid + 1

else:

right = mid - 1

return -1

# Solicita a entrada do usuário para o array de números

entrada = input("Digite os números separados por espaço: ")

numeros = list(map(int, entrada.split()))

# Chama a função para buscar o ponto fixo

ponto\_fixo = fixed\_point\_search(numeros)

# Exibe o resultado da busca

if ponto\_fixo != -1:

print(f"O ponto fixo está no índice {ponto\_fixo}.")

else:

print("Não foi encontrado ponto fixo no array.")