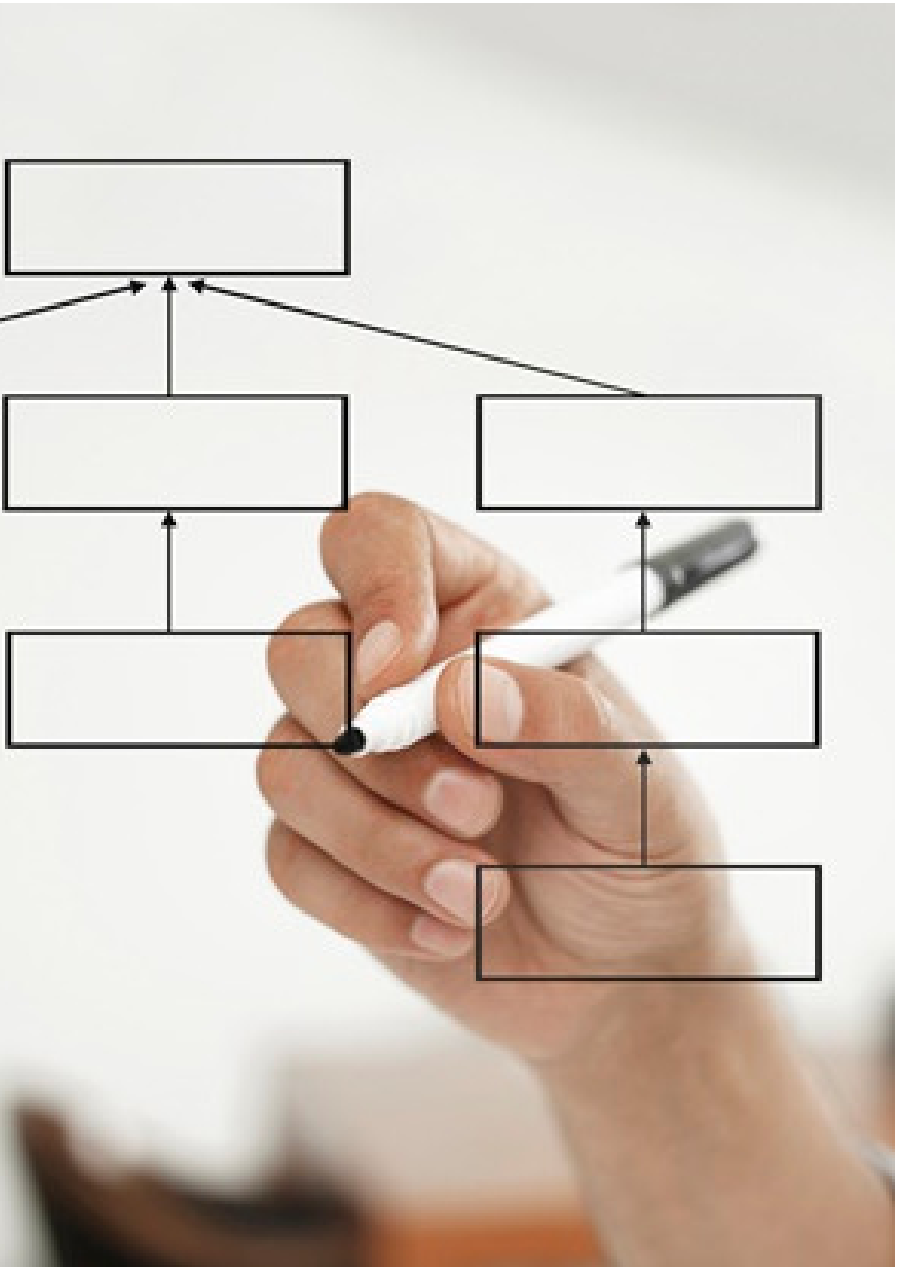


# Estrutura de Dados I

CÁSSIO CAPUCHO PEÇANHA – 07

PARTE 2

# Ordenação por Mistura

## (MergeSort)

22/09/2025

# MergeSort

- Também conhecido como ordenação por intercalação
  - Algoritmo recursivo que usa a idéia de dividir para conquistar para ordenar os dados
    - Parte do princípio de que é mais fácil ordenar um conjunto com poucos dados do que um com muitos
  - O algoritmo divide os dados em conjuntos cada vez menores para depois ordená-los e combina-los por meio de intercalação (merge)

# MergeSort

Ideia básica: Dividir e Conquistar;

Divide, recursivamente, o conjunto de dados até que cada subconjunto possua 1 elemento;

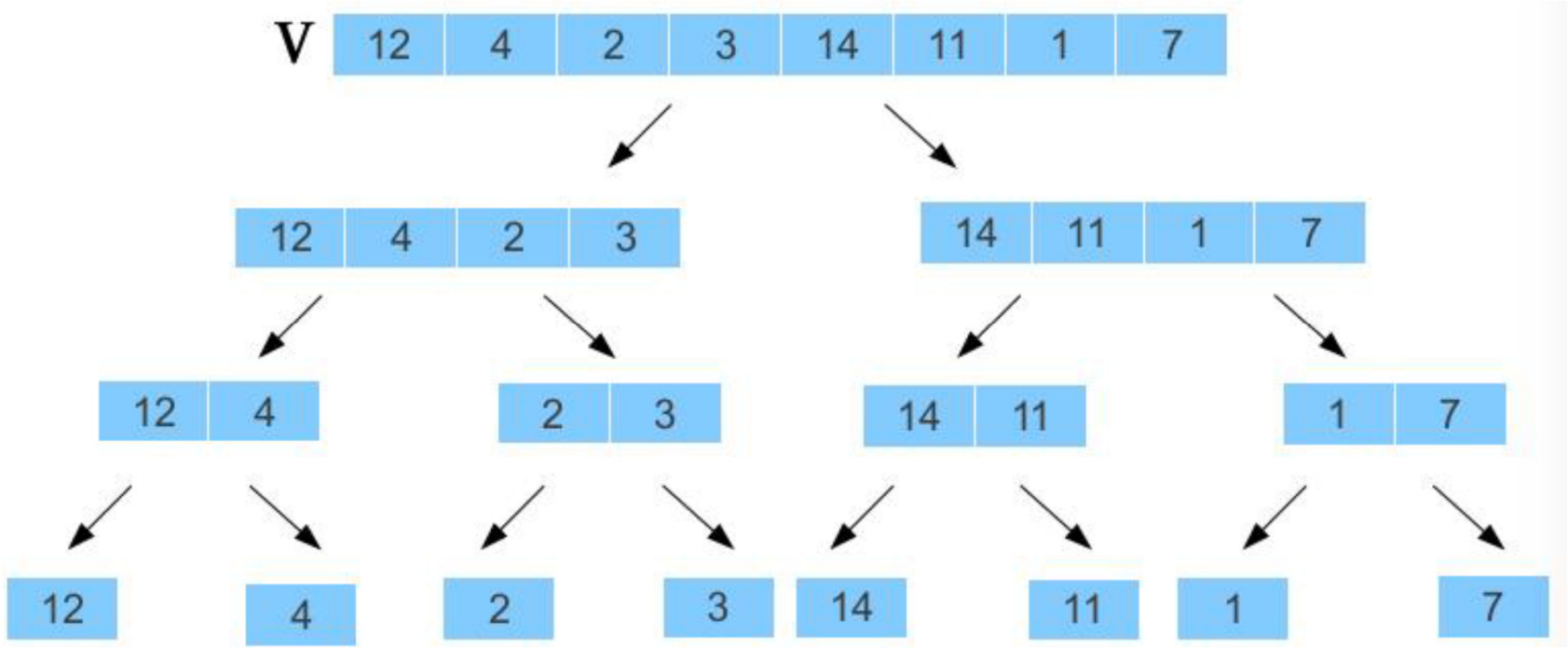
Combina 2 subconjuntos de forma a obter 1 conjunto maior e ordenado

Esse processo se repete até que exista apenas 1 conjunto.

# MergeSort - Funcionamento

- Divide, recursivamente, o array em duas partes
- Continua até cada parte ter apenas um elemento
- Em seguida, combina dois array de forma a obter um array maior e ordenado
- A combinação é feita intercalando os elementos de acordo com o sentido da ordenação (crescente ou decrescente)
- Este processo se repete até que exista apenas um array

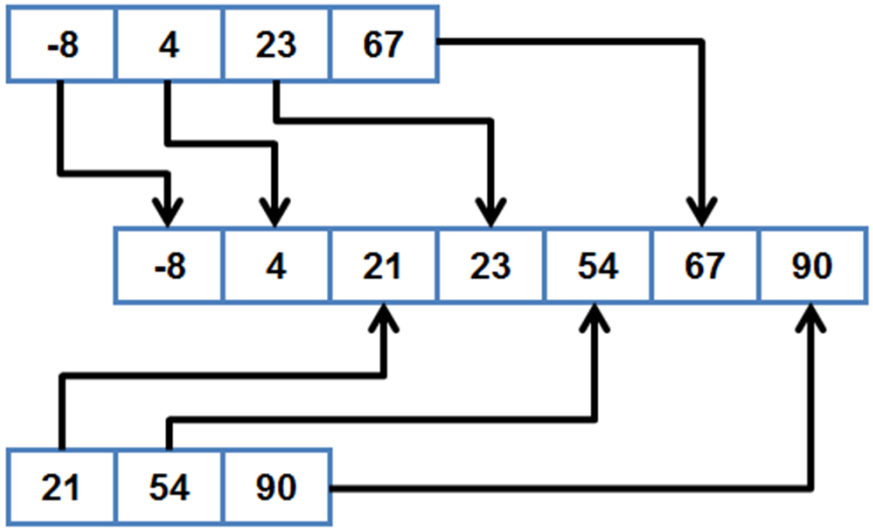
# MergeSort - Exemplo



# Algoritmo Merge Sort

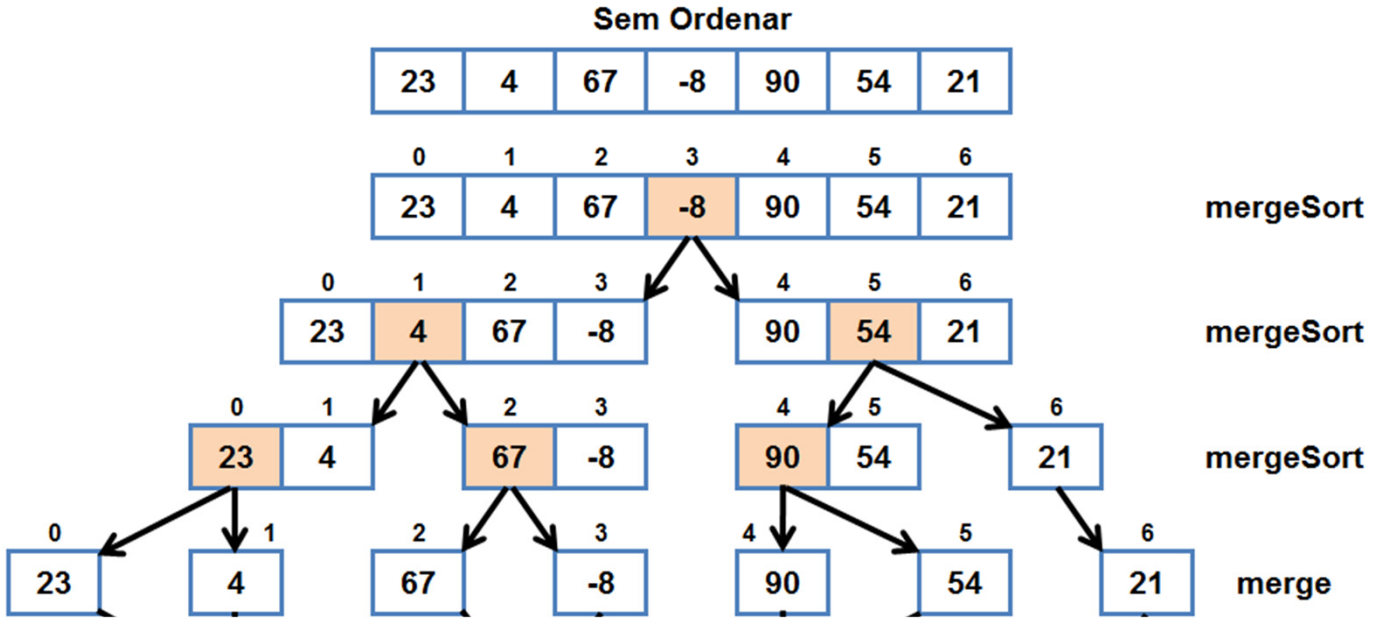
Passo a passo: função merge

- Intercala os dados de forma ordenada em um array maior
- Utiliza um array auxiliar



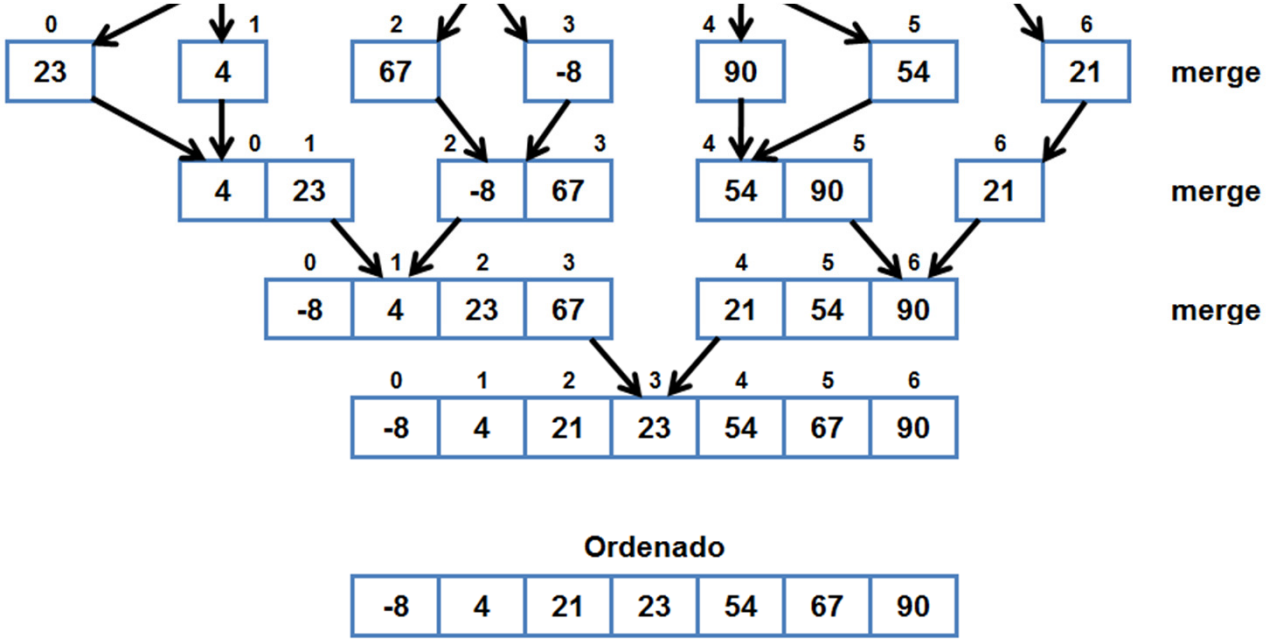
# Algoritmo Merge Sort

- Passo a passo
  - Divide o array até ter **N** arrays de 1 elemento cada



# Algoritmo Merge Sort

- Passo a passo
  - Intercala os arrays até obter um único array de **N** elementos



# Algoritmo Merge Sort

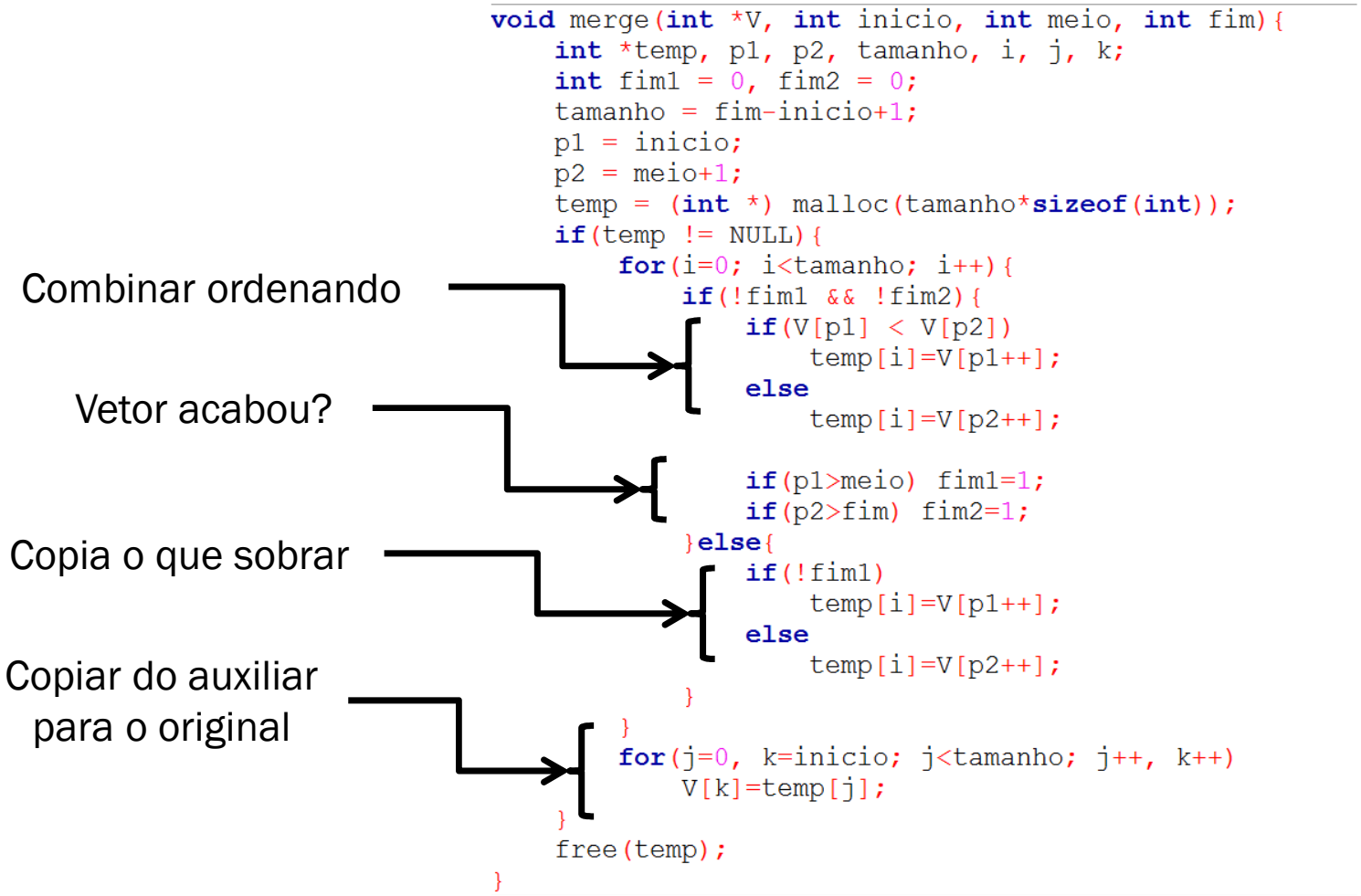
- Algoritmo usa 2 funções
  - mergeSort : divide os dados em arrays cada vez menores
  - merge: intercala os dados de forma ordenada em um array maior

```
23
24  void mergeSort(int *V, int inicio, int fim){
25      int meio;
26      if(inicio < fim){
27          meio = floor((inicio+fim)/2);
28          mergeSort(V,inicio,meio);
29          mergeSort(V,meio+1,fim);
30          merge(V,inicio,meio,fim);
31      }
32  }
```

Chama a função para as 2 metades

Combina as 2 metades de forma ordenada

# Algoritmo Merge Sort




```
void merge(int *V, int inicio, int meio, int fim){
    int *temp, p1, p2, tamanho, i, j, k;
    int fim1 = 0, fim2 = 0;
    tamanho = fim-inicio+1;
    p1 = inicio;
    p2 = meio+1;
    temp = (int *) malloc(tamanho*sizeof(int));
    if(temp != NULL){
        for(i=0; i<tamanho; i++){
            if(!fim1 && !fim2){
                if(V[p1] < V[p2])
                    temp[i]=V[p1++];
                else
                    temp[i]=V[p2++];

                if(p1>meio) fim1=1;
                if(p2>fim) fim2=1;
            }else{
                if(!fim1)
                    temp[i]=V[p1++];
                else
                    temp[i]=V[p2++];
            }
        }
        for(j=0, k=inicio; j<tamanho; j++, k++)
            V[k]=temp[j];
    }
    free(temp);
}
```

# MergeSort - Exercício

1. Determine o melhor e o pior caso para o MergeSort? Explique.
2. O MergeSort é estável? Explique.

# MergeSort – Exercício (Explicação)

- Método de ordenação com tempo:
  - Melhor caso: n * $\log_2 n$;
  - Pior caso: n * $\log_2 n$;
- Pode ser adaptado para ordenação de arquivos externos (memória secundária).
- Utiliza mais memória para poder ordenar (vetor auxiliar).
- O MergeSort é estável: não altera a ordem de dados iguais.

# Ordenação por Troca de Partições

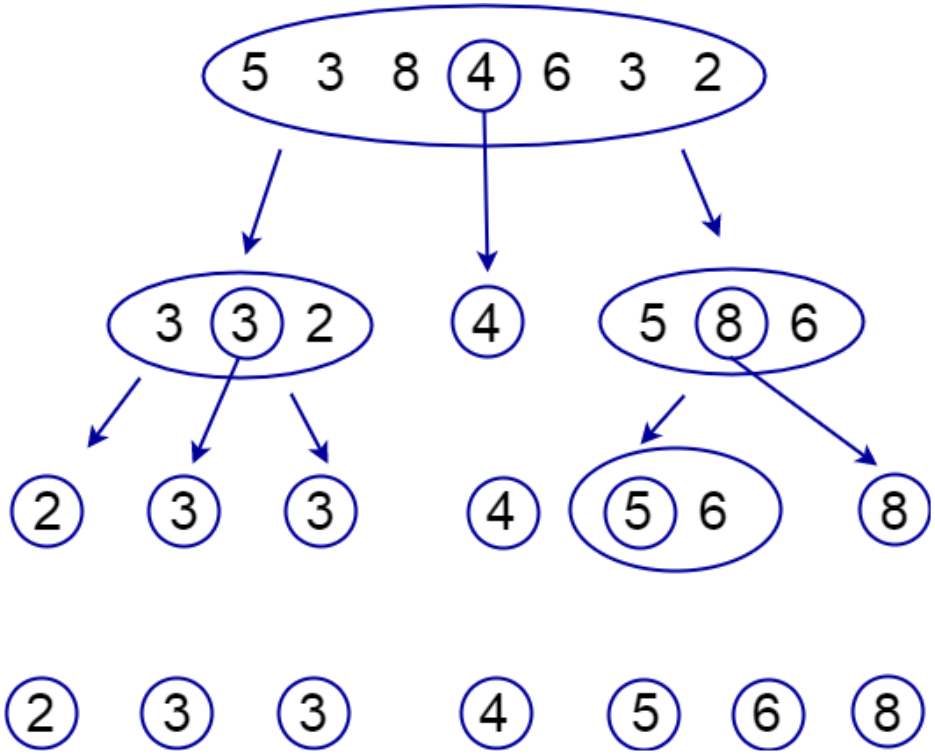
## (QuickSort)

22/09/2025

# QuickSort

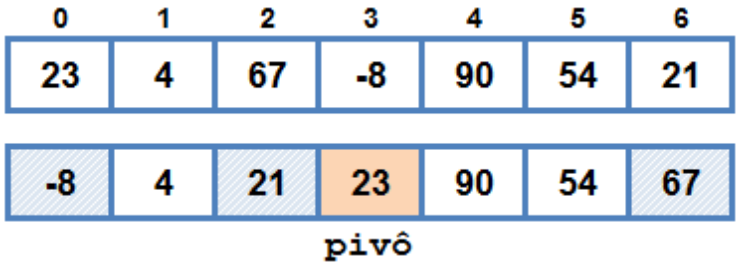
- É o algoritmo de ordenação interna mais rápido que se conhece para uma ampla variedade de situações;
- Provavelmente é o mais utilizado;
- Ideia básica: Dividir e Conquistar;
- Um elemento é escolhido como pivô;
- “Particionar”: os dados são rearranjados (valores menores do que o pivô são colocados antes dele e os maiores, depois);
- Recursivamente ordena as 2 partições;

# QuickSort



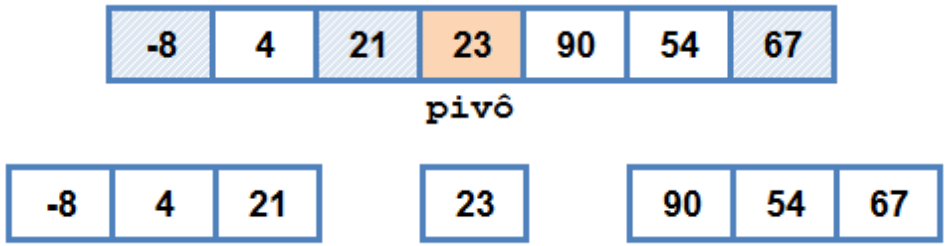
# Algoritmo Quick Sort

- Problema da separação
  - Em inglês, *partition subproblem*
  - Consiste em rearranjar o array usando um valor como **pivô**
    - Valores menores do que o **pivô** ficam a esquerda
    - Valores maiores do que o **pivô** ficam a direita



# Algoritmo Quick Sort

- Funcionamento
  - Um elemento é escolhido como pivô
  - Valores menores do que o pivô são colocados antes dele e os maiores, depois
    - Supondo o pivô na posição **X**, esse processo cria duas partições: **[0,...,X-1]** e **[X+1,...,N-1]**.
  - Aplicar recursivamente a cada partição
    - Até que cada partição contenha um único elemento

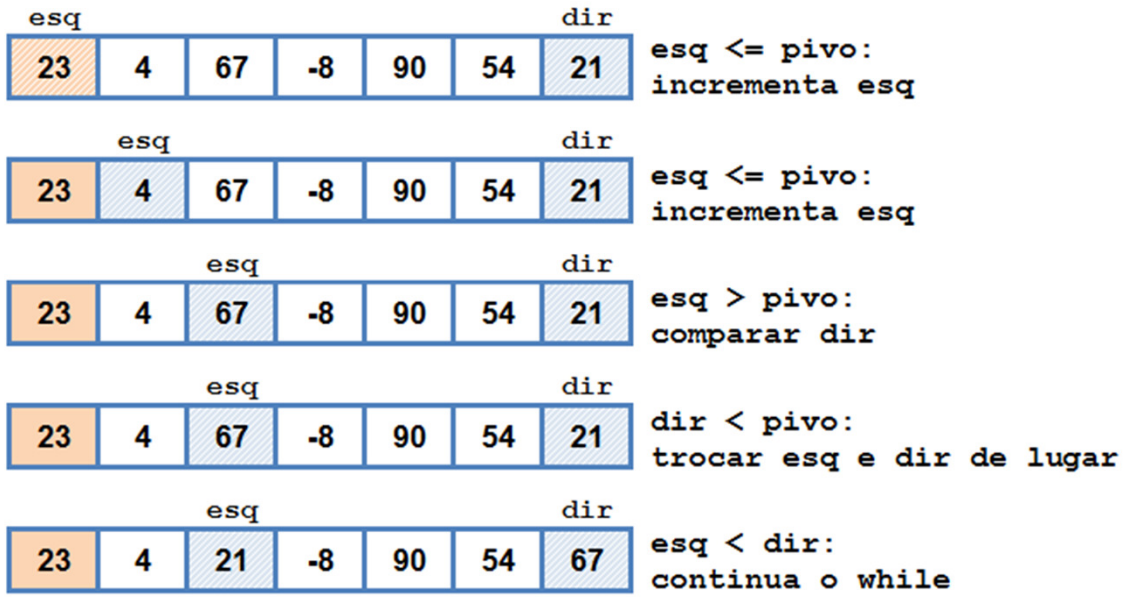


# Quick Sort - Passo a passo: função particiona

**particiona(V,0,6)**

**Primeira chamada**

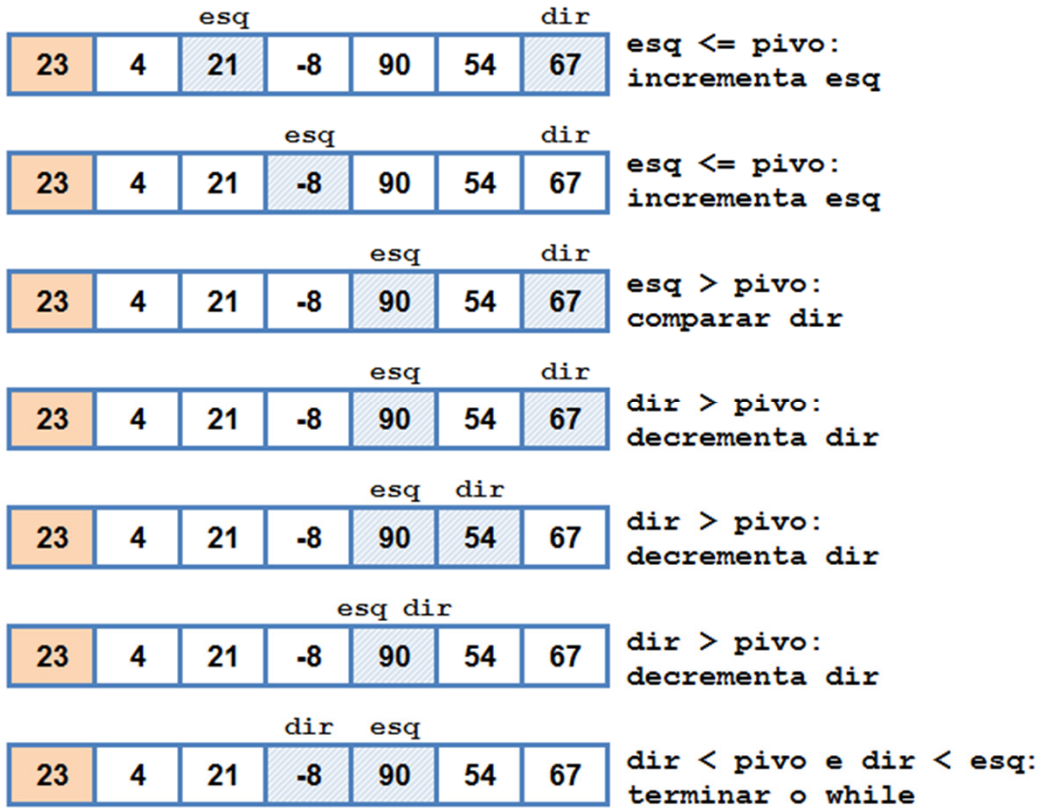
`while(esq < dir)`



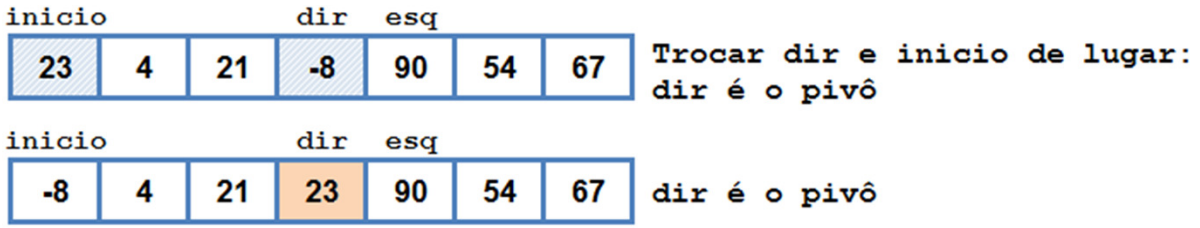
# Quick Sort - Passo a passo: função particiona

**Segunda chamada**

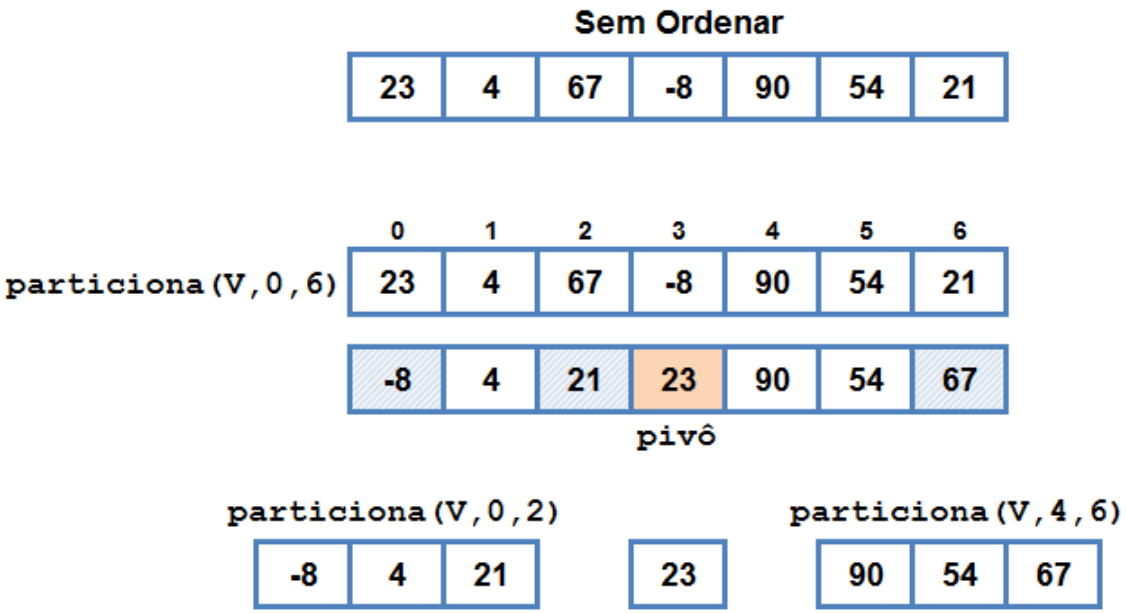
`while(esq < dir)`



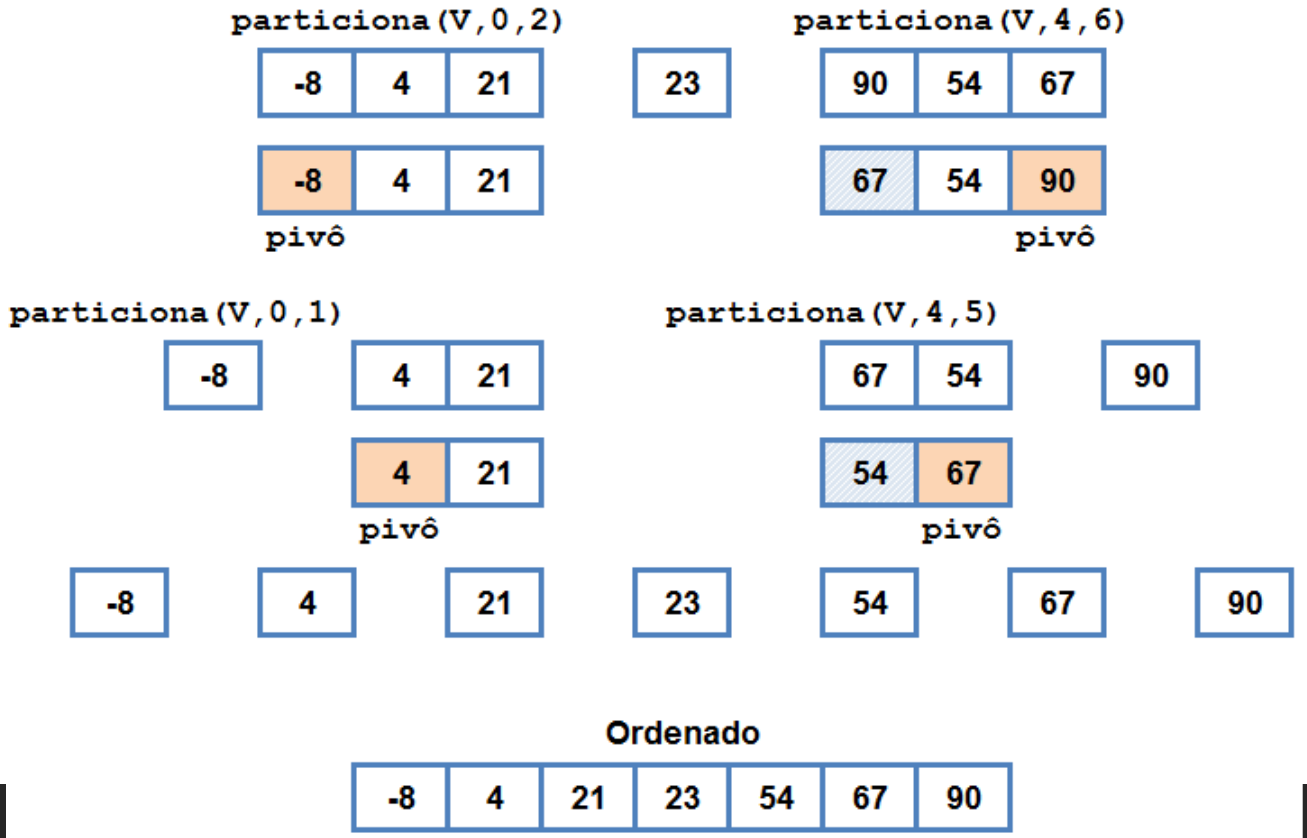
# Quick Sort - Passo a passo: função particiona



# Algoritmo Quick Sort



# Algoritmo Quick Sort



# QuickSort - Exercício

1. Explique como funciona o código do QuickSort.
2. Determine o melhor e o pior caso para o QuickSort? Explique com suas palavras.
3. O QuickSort é estável? Explique com suas palavras.

**Envio de arquivo em .c.txt no blog.

# QuickSort – Exercício (Explicação)

- Melhor Caso: O (N Log N)
  - Esta situação ocorre quando cada partição divide o arquivo em duas partes iguais.
- Pior Caso: O ($N^2$)
  - O pior caso ocorre quando, sistematicamente, o pivô é escolhido como sendo um dos extremos de um arquivo já ordenado.
  - Isto faz com que o procedimento Ordena seja chamado recursivamente n vezes, eliminando apenas um item em cada chamada.
  - O pior caso pode ser evitado empregando pequenas modificações no algoritmo.
  - Para isso basta escolher três itens quaisquer do vetor e usar a mediana dos três como pivô.
- Instável
- Desvantagem: Como escolher o pivô?