ALGORITMOS ÓTIMOS DE ORDENAÇÃO

DCE792 - AEDs II (Prática)

Atualizado em: 28 de outubro de 2025

Iago Carvalho

Departamento de Ciência da Computação

ALGORITMO ÓTIMO

O problema de ordenação tem complexidade de $\Omega(n \log n)$ para um vetor de n posições

Desta forma, algoritmos de complexidade $O(n \log n)$ são dito serem algoritmos ótimos para o problema de ordenação

Aqui nós vamos ver dois destes algoritmos

- Merge sort
- Quick sort

MERGE SORT

MERGE SORT

Também conhecido como ordenação por intercalação

Algoritmo recursivo que utiliza a ideia de dividir para conquistar

- Um paradigma de projeto de algoritmos que vocês estudarão em AEDS 3
- Utiliza a premissa de que é mais fácil resolver múltiplos problemas de menor porte do que um problema muito grande

O Merge sort divide os dados em conjuntos cada vez menores

- Depois, recombina os conjuntos ordenando seus elementos

MERGE SORT

O Merge sort é um algoritmo estável

- Não altera a ordem de dados com mesmo valor

Entretanto, ele possui um gasto de memória maior do que outros algoritmos de ordenação

- Método recursivo
- Necessário criar uma cópia do vetor para cada chamada recursiva
  - Existe uma implementação avançada do Merge sort que utiliza um único vetor auxiliar grande durante toda a execução do método

MERGE SORT - FUNCIONAMENTO

1. Divide, recursivamente, o vetor em partes cada vez menores
   - Operação realizada até obter partes com um único elemento
2. Combina as sub-partes duas a duas de forma a obter uma parte maior
   - Processo denominado *merge*
   - Esse processo de combinação é realizado ordenando os elementos de cada parte
3. Processo de *merge* é repetido até que exista somente uma grande parte (um vetor)

MERGE SORT

```
def mergeSort(V, inicio, fim):
    if(inicio < fim):
        meio = int((inicio+fim)/2)
        mergeSort(V,inicio,meio)
        mergeSort(V,meio+1,fim)
        merge(V,inicio,meio,fim)
```

Chama a função para as 2 metades

Combina as 2 metades de forma ordenada

```
def merge(V, inicio, meio, fim):
    tamanho = fim-inicio+1
    p1 = inicio
    p2 = meio+1
    fim1 = False
    fim2 = False

    temp = [0 for i in range(tamanho)]

    for i in range(tamanho):
        if(not fim1 and not fim2):
            if(V[p1] < V[p2]):
                temp[i] = V[p1]
                p1 = p1 + 1
            else:
                temp[i] = V[p2]
                p2 = p2 + 1

        if(p1 > meio):
            fim1 = True

        if(p2 > fim):
            fim2 = True
        else:
            if(not fim1):
                temp[i] = V[p1]
                p1 = p1 + 1
            else:
                temp[i] = V[p2]
                p2 = p2 + 1

    k = inicio
    for j in range(tamanho):
        V[k] = temp[j]
        k = k + 1
```

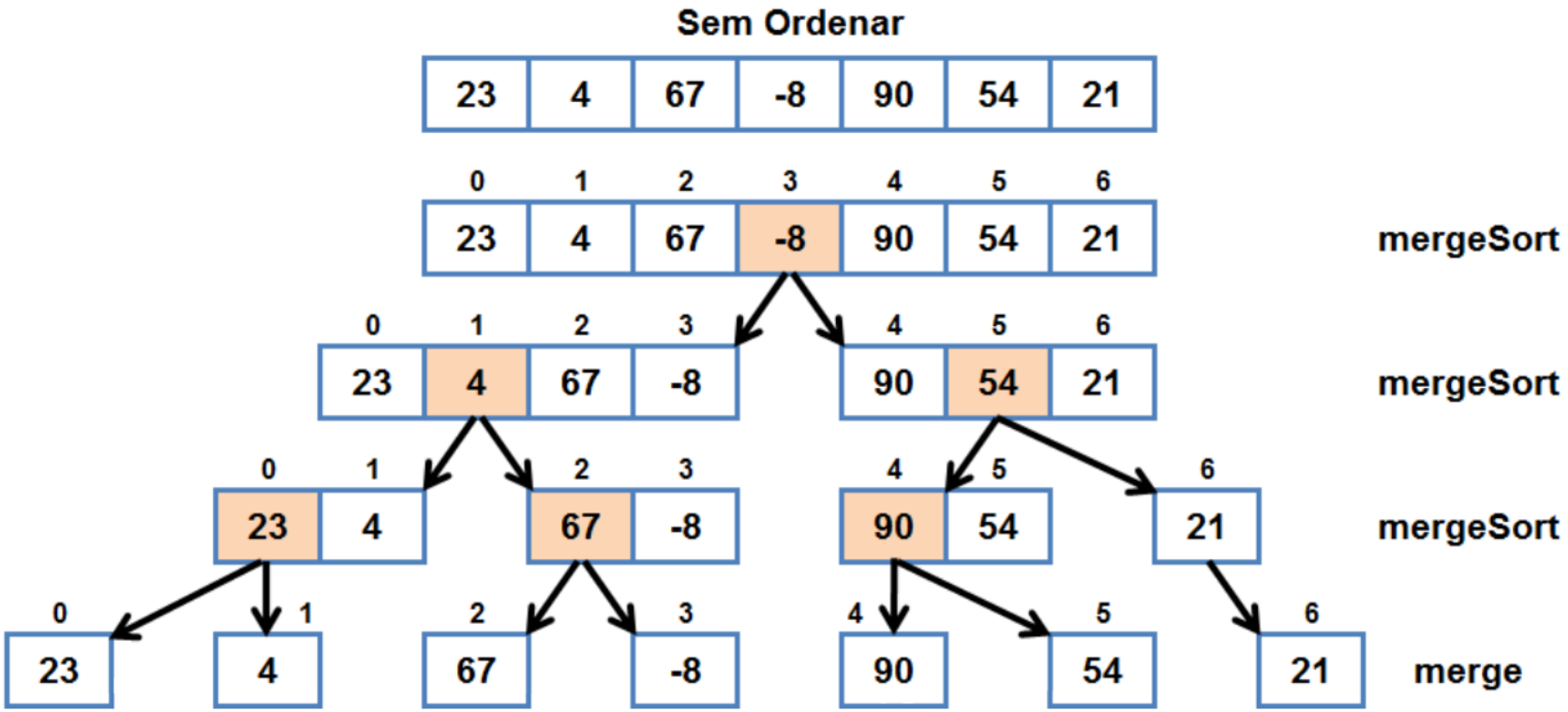
Combinar ordenando

Vetor acabou?

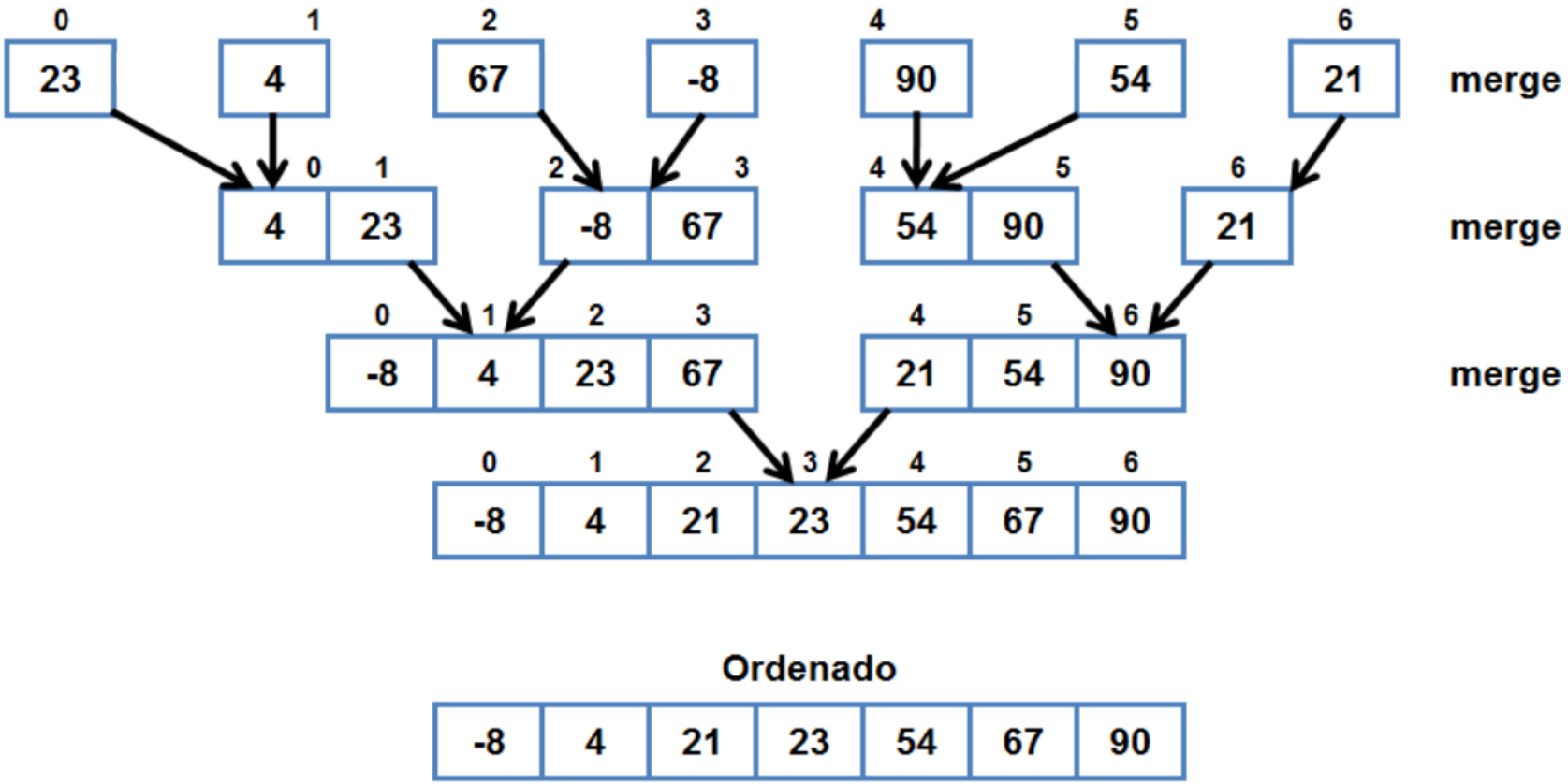
Copia o que sobrar

Copiar do auxiliar para o original

MERGE SORT


Sem Ordenar
23 4 67 -8 90 54 21
0 1 2 3 4 5 6
23 4 67 -8 90 54 21
mergeSort
0 1 2 3 4 5 6
23 4 67 -8 90 54 21
mergeSort
0 1 2 3 4 5 6
23 4 67 -8 90 54 21
mergeSort
0 1 2 3 4 5 6
23 4 67 -8 90 54 21
merge

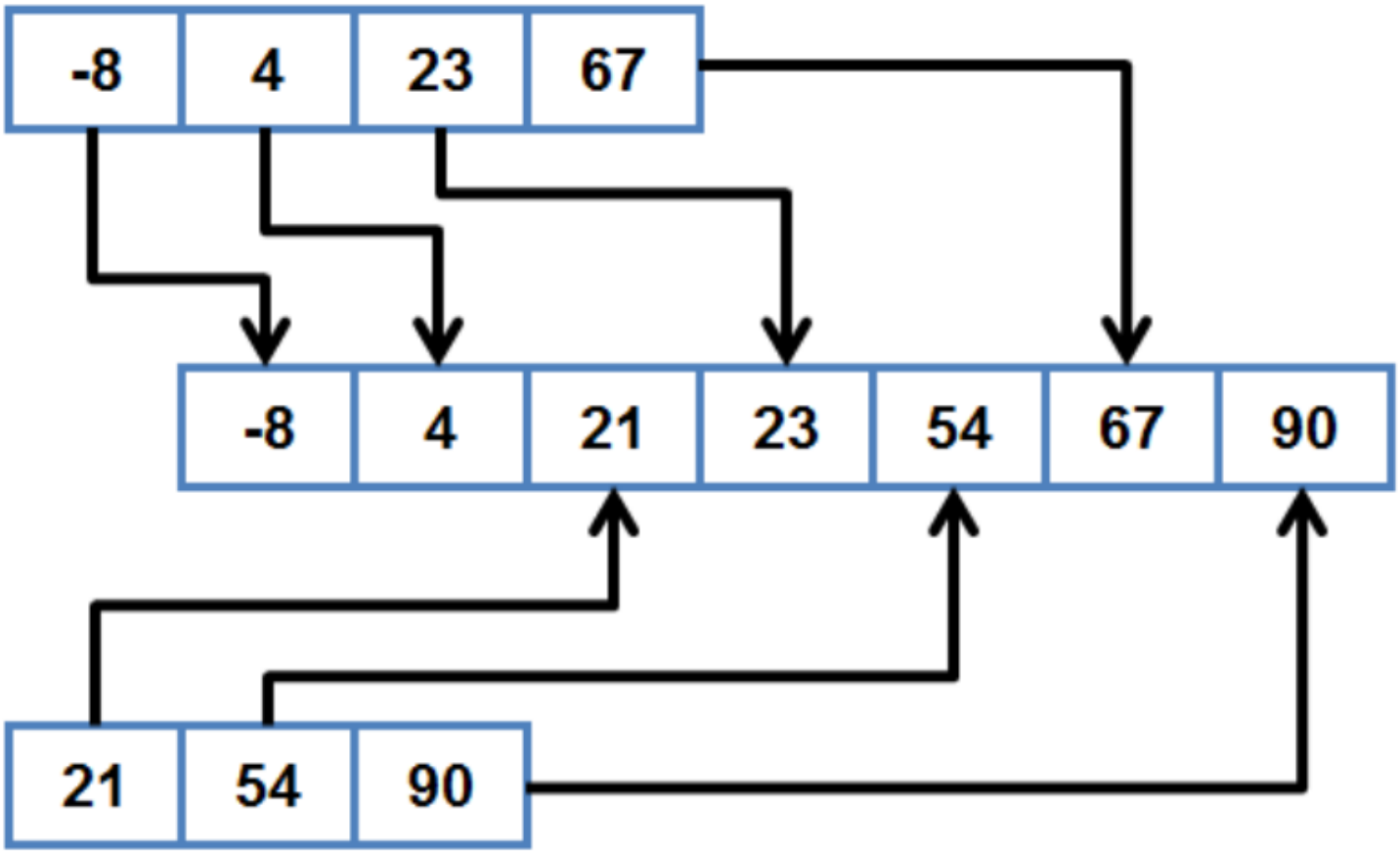
MERGE SORT


0 1 2 3 4 5 6
23 4 67 -8 90 54 21
merge
0 1 2 3 4 5 6
4 23 -8 67 54 90 21
merge
0 1 2 3 4 5 6
-8 4 23 67 21 54 90
merge
0 1 2 3 4 5 6
-8 4 21 23 54 67 90
Ordenado
-8 4 21 23 54 67 90

MERGE SORT - FUNCIONAMENTO

A função **merge** intercala os dados de forma ordenada em um vetor maior

- Para isto, ela utiliza um vetor auxiliar



QUICK SORT

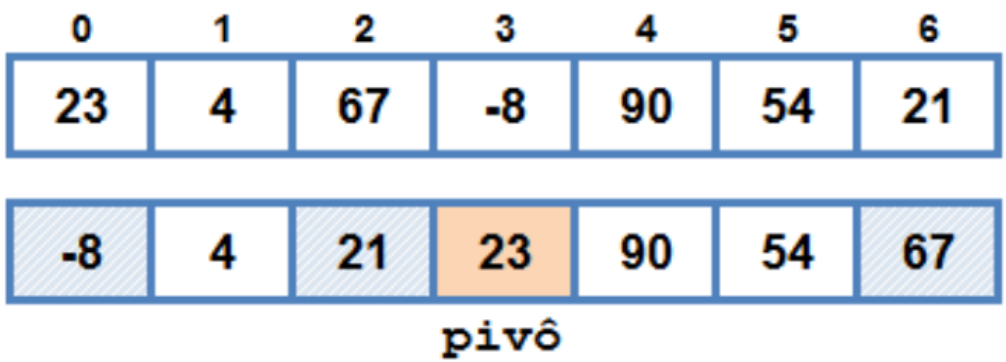
QUICK SORT

Talvez este seja o método de ordenação mais famoso e utilizado

- Também conhecido como ordenação por partição
- Outro algoritmo que utiliza a estratégia de dividr para conquistar

Consiste em ordenar o vetor utilizando um valor como **pivô**. Considerando um método de ordenação crescente, temos que

- Valores menor que o pivô ficam a sua esquerda
- Valores maiores que o pivô ficam a sua direita



QUICK SORT

O Quick sort **não** é um algoritmo estável

- Ele pode alterar a ordem de dados com mesmo valor

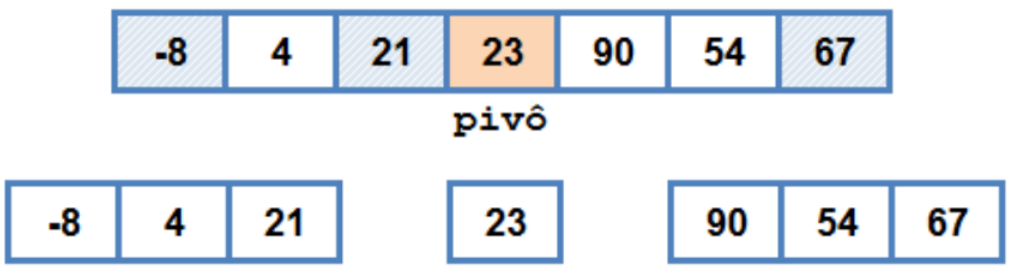
Não utiliza memória adicional

Na prática, ele é o mais rápido dos algoritmos de ordenação

- Entretanto, ele possui um pior caso de $O(n^2)$
- Esse pior caso acontece devido a uma má escolha do pivô
  - No pior caso, duas partições são criadas: uma com 0 elementos e outra com $n - 1$ elementos

QUICK SORT - FUNCIONAMENTO

1. Um elemento do vetor é escolhido como pivô
   - Valores menores que o pivô são inseridos a sua esquerda
   - Valores maiores que o pivô são inseridos a sua direita
2. Duas novas partições são criadas
   - Partição a esquerda do pivô
   - Partição a direita do pivô
3. Quick sort é aplicado recursivamente a cada partição
   - Procedimento é realizado até que cada partição contenha um único elemento



QUICK SORT

O Quick sort utiliza duas funções

1. quickSort
2. particiona

A função **quickSort** divide os dados em partições cada vez menores

```
def quickSort(V, inicio, fim):
    if(fim > inicio):
        pivo = particiona(V, inicio, fim)
        quickSort(V, inicio, pivo-1)
        quickSort(V, pivo+1, fim)
```

Separa os dados em 2 partições

Chama a função para as 2 metades

A FUNÇÃO PARTICIONA

A função **particiona** calcula o pivô e ordena os dados

```
def particiona(V, inicio, final):
    esq = inicio
    dir = final
    pivo = V[inicio]
    while(esq < dir):
        while(esq <= final and V[esq] <= pivo):
            esq = esq + 1

        while(dir >= 0 and V[dir] > pivo):
            dir = dir - 1

        if(esq < dir):
            aux = V[esq]
            V[esq] = V[dir]
            V[dir] = aux

    V[inicio] = V[dir]
    V[dir] = pivo
    return dir
```

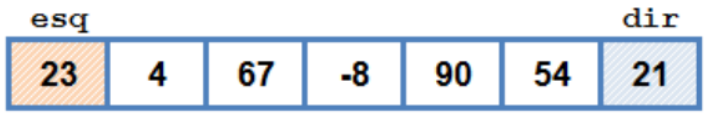
Avança posição da esquerda

Recua posição da direita

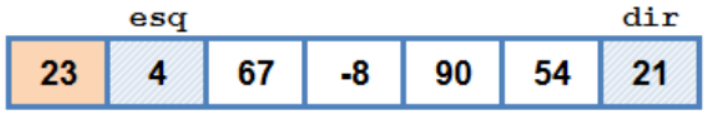
Trocar esq e dir

CALCULO DO PIVÔ

particiona(V,0,6)



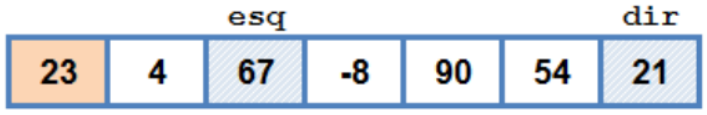
`esq <= pivo:`
`incrementa esq`



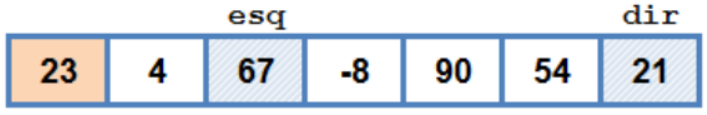
`esq <= pivo:`
`incrementa esq`

Primeira chamada

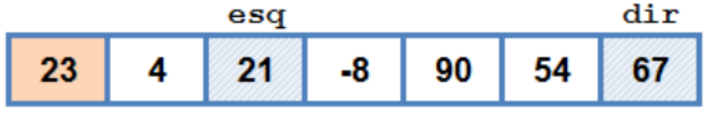
`while(esq < dir)`



`esq > pivo:`
`comparar dir`



`dir < pivo:`
`trocar esq e dir de lugar`

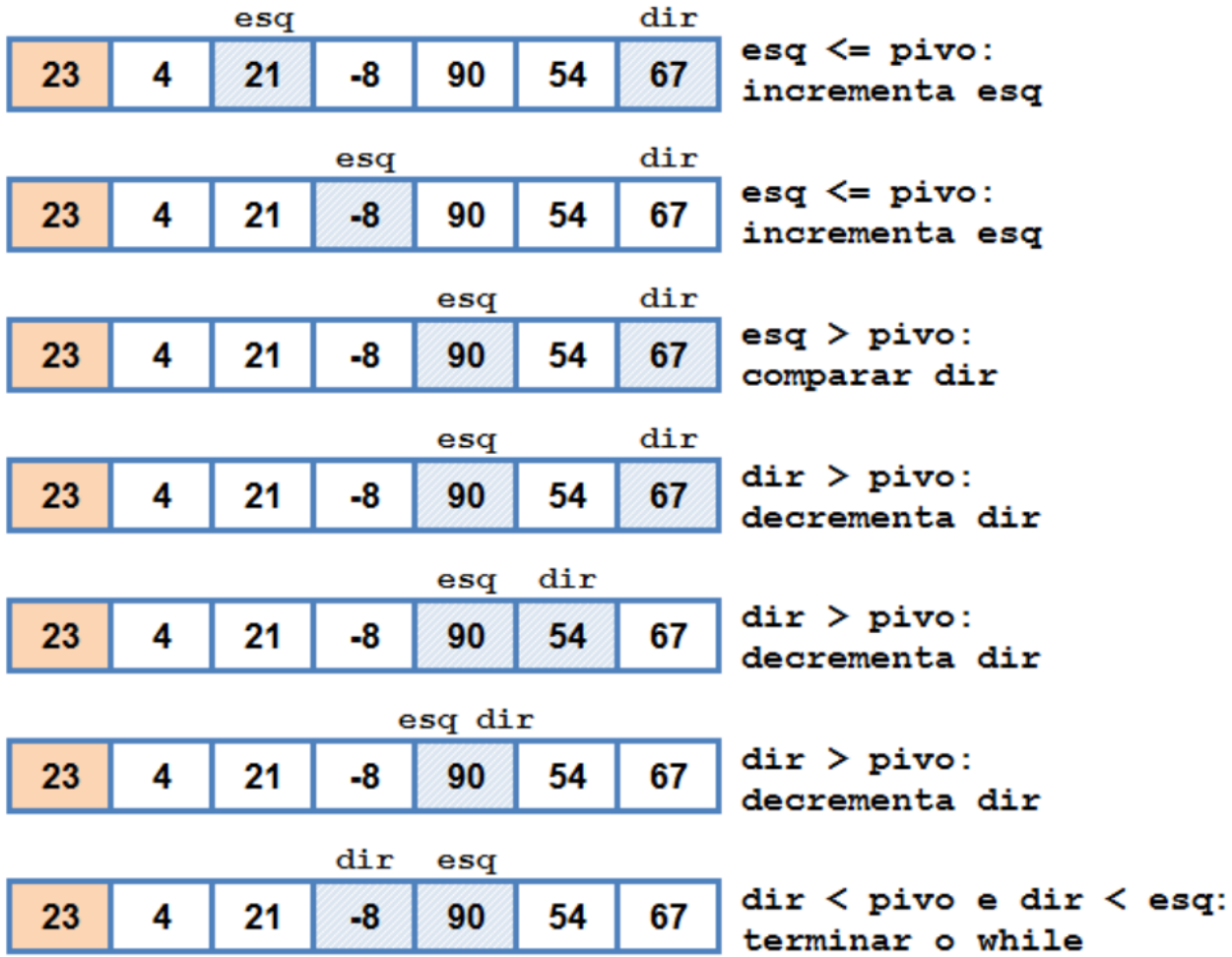


`esq < dir:`
`continua o while`

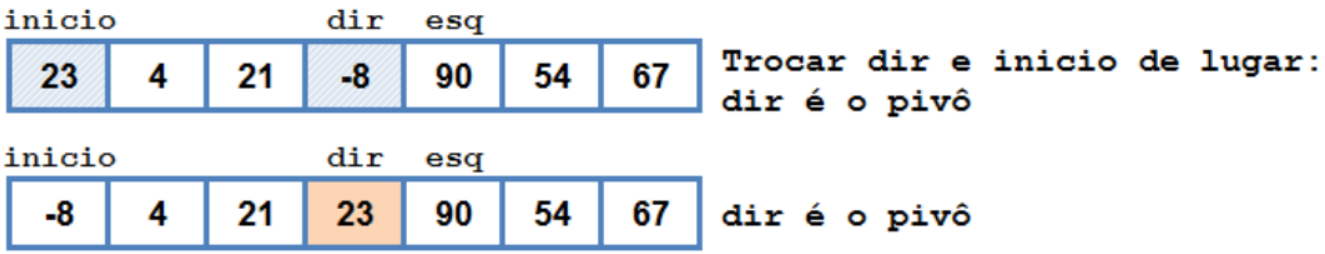
CALCULO DO PIVÔ

Segunda chamada

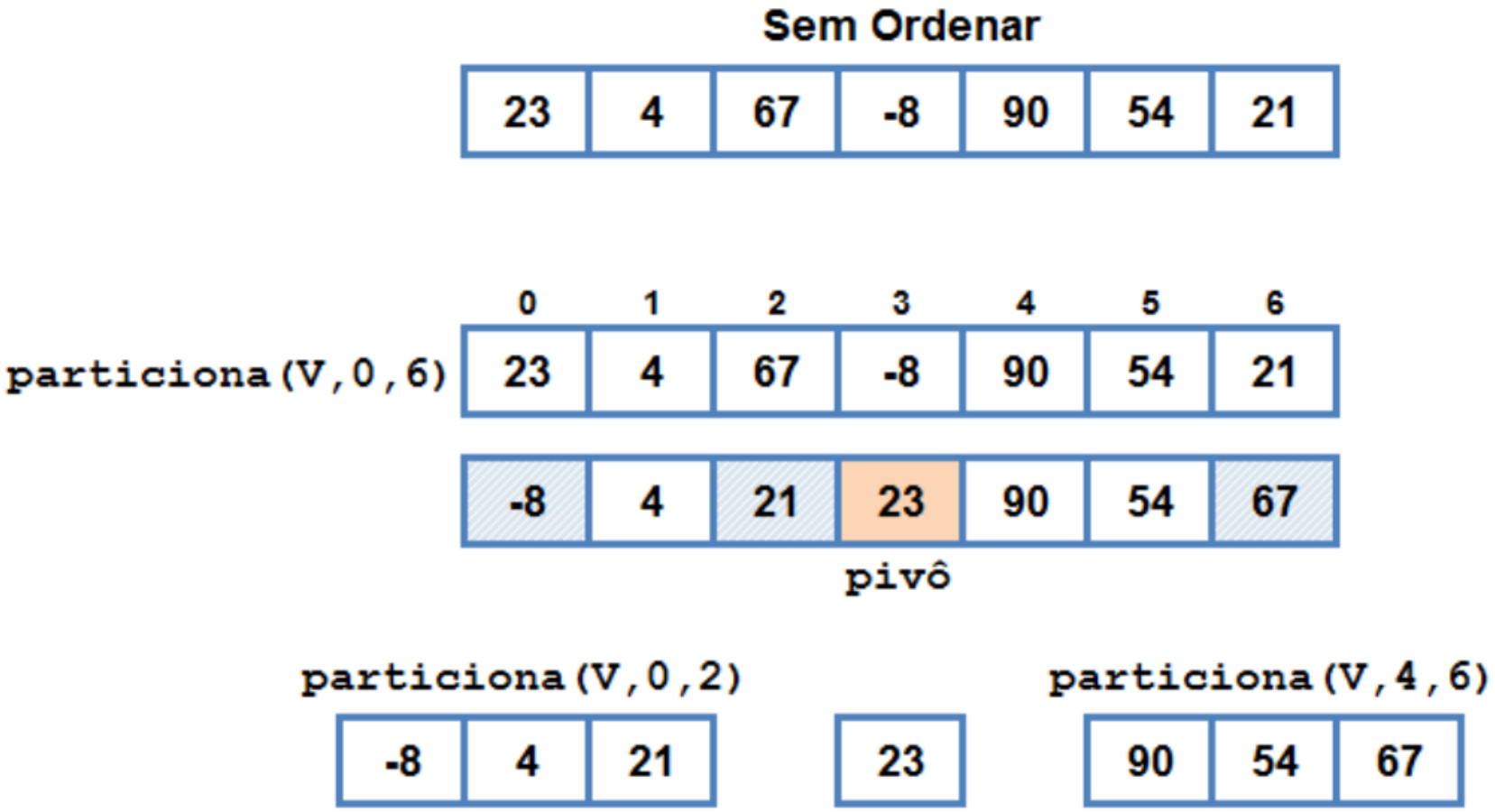
`while(esq < dir)`



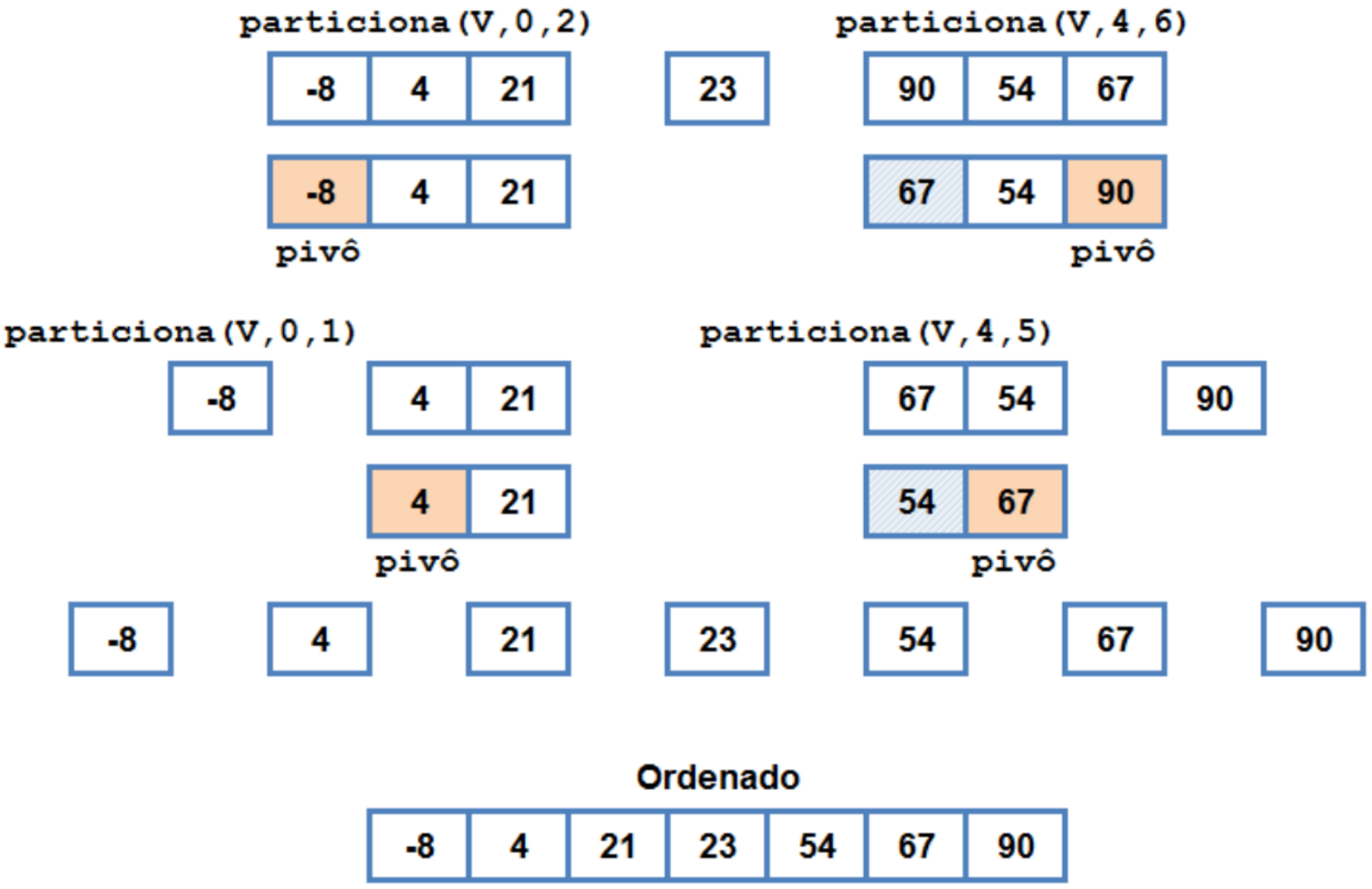
CALCULO DO PIVÔ



QUICK SORT


Sem Ordenar
23 4 67 -8 90 54 21
0 1 2 3 4 5 6
particiona(V,0,6)
23 4 67 -8 90 54 21
-8 4 21 23 90 54 67
pivô
particiona(V,0,2)
-8 4 21
23
particiona(V,4,6)
90 54 67

QUICK SORT


particiona(V,0,2)
-8
4
21
-8
4
21
pivô
23
particiona(V,4,6)
90
54
67
67
54
90
pivô
particiona(V,0,1)
-8
4
21
4
21
pivô
particiona(V,4,5)
67
54
90
54
67
pivô
-8
4
21
23
54
67
90
Ordenado
-8
4
21
23
54
67
90

ATIVIDADE PRÁTICA

ATIVIDADE

Deve-se implementar os dois algoritmos de ordenação

Código-base está disponível no Github ▸ Link

Os algoritmo pode ser visualizados em ▸ Link

PRÓXIMA AULA:

ALGORITMOS DE ORDENAÇÃO EM TEMPO LINEAR