9002 — Aula 27
Algoritmos e Programação de Computadores

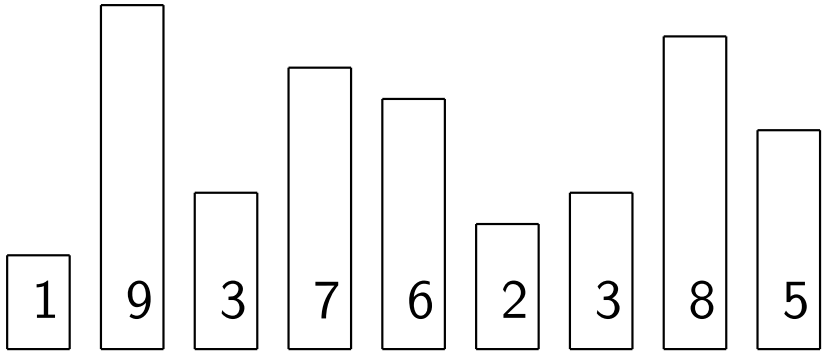
Instituto de Engenharia – UFMT

Segundo Semestre de 2014

26 de janeiro de 2015

Roteiro

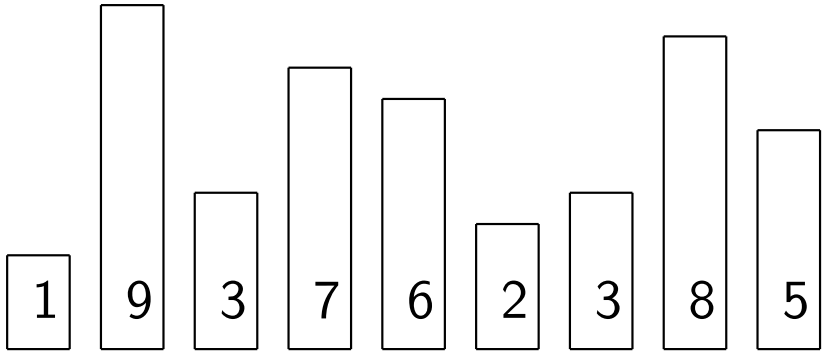
Introdução



Problema 1

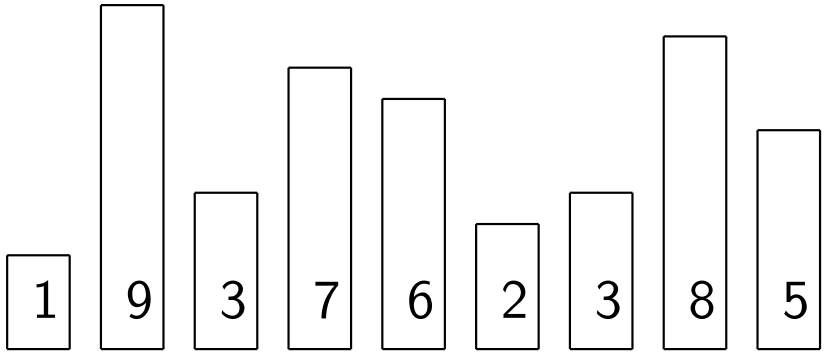
Suponha que temos um vetor desordenado com 10 números. Como fazer com que números *pequenos* (menores que 5) fiquem antes dos números *grandes* (maiores que 5)?

Considere a função

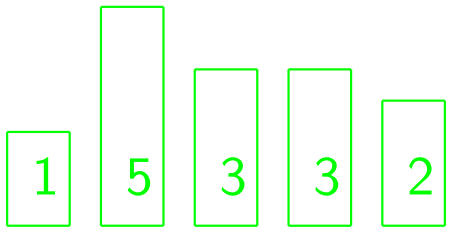


- `int particionar(int vetor[], int ini, int fim)`

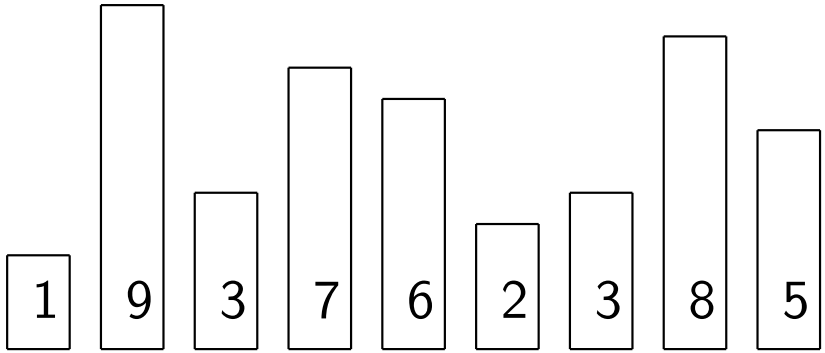
Considere a função



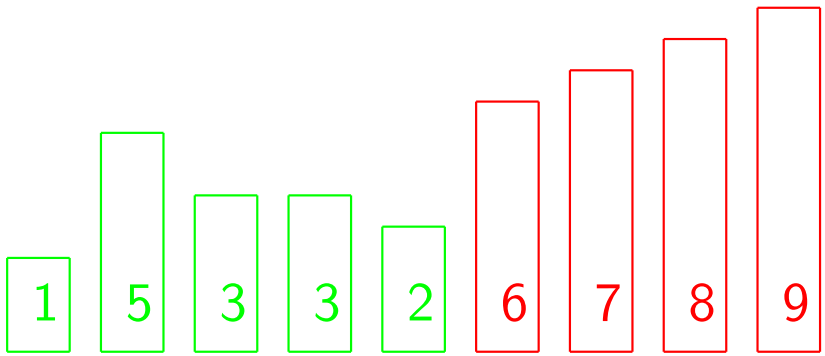
- `int particionar(int vetor[], int ini, int fim)`
  - a primeira parte do vetor contém elementos **"pequenos"**
  - a segunda parte do vetor contém elementos **"grandes"**



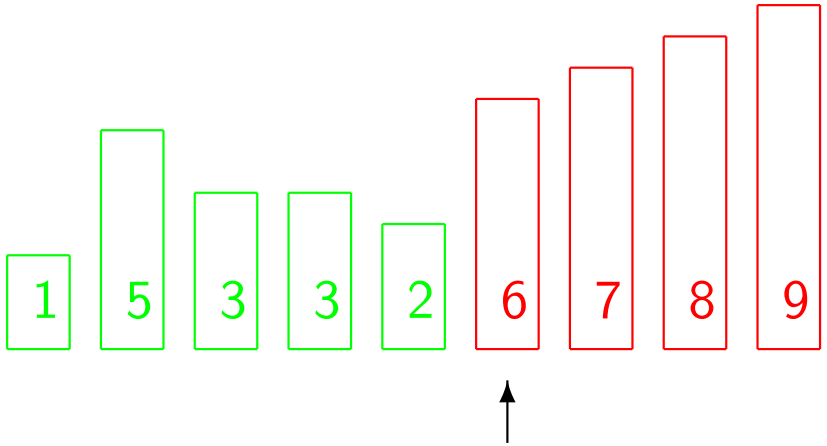
Considere a função



- `int particionar(int vetor[], int ini, int fim)`
  - a primeira parte do vetor contém elementos **"pequenos"**
  - a segunda parte do vetor contém elementos **"grandes"**



Combinando



Problema 2

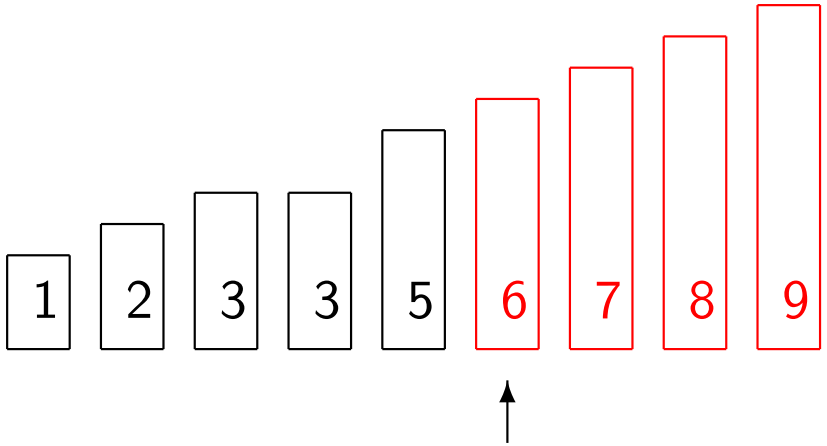
Suponha que o subvetor

- da posição **pos** a **fim**: contenha apenas elementos grandes
- da posição **ini** a **pos - 1**: contenha apenas elementos pequenos

Como ordenar?

- Ordenamos recursivamente o primeiro subvetor
- Depois o segundo subvetor

Combinando



Problema 2

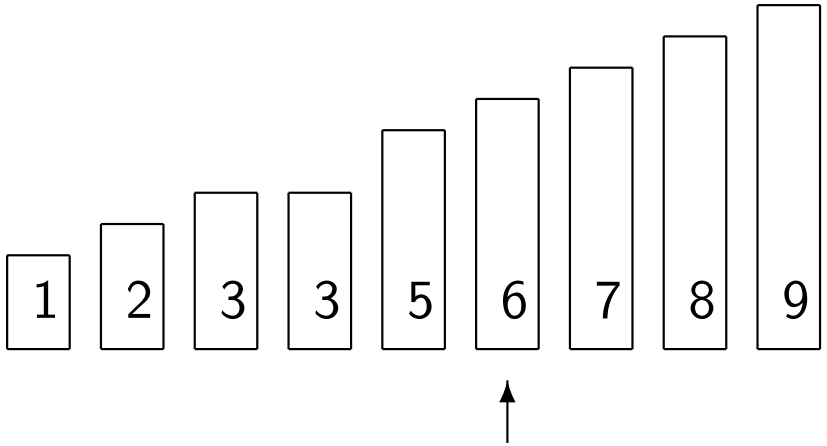
Suponha que o subvetor

- da posição **pos** a **fim**: contenha apenas elementos grandes
- da posição **ini** a **pos - 1**: contenha apenas elementos pequenos

Como ordenar?

- Ordenamos recursivamente o primeiro subvetor
- Depois o segundo subvetor

Combinando



Problema 2

Suponha que o subvetor

- da posição **pos** a **fim**: contenha apenas elementos grandes
- da posição **ini** a **pos - 1**: contenha apenas elementos pequenos

Como ordenar?

- Ordenamos recursivamente o primeiro subvetor
- Depois o segundo subvetor

Divisão e conquista novamente

Quick Sort

- **Divisão:** Separamos elementos pequenos e grandes
- **Conquista:** Ordenamos cada subvetor

QuickSort

```
void quick_sort(int vetor[], int ini, int fim) {
    int pos;

    if (ini < fim){
        pos = particionar(vetor, ini, fim);

        quick_sort(vetor, ini, pos - 1);
        quick_sort(vetor, pos, fim);
    }
}
```

Divisão e conquista novamente

Quick Sort

- **Divisão:** Separamos elementos pequenos e grandes
- **Conquista:** Ordenamos cada subvetor

QuickSort

```
void quick_sort(int vetor[], int ini, int fim) {
    int pos;

    if (ini < fim){
        pos = particionar(vetor, ini, fim);

        quick_sort(vetor, ini, pos - 1);
        quick_sort(vetor, pos, fim);
    }
}
```

Divisão e conquista novamente

Quick Sort

- **Divisão:** Separamos elementos pequenos e grandes
- **Conquista:** Ordenamos cada subvetor

QuickSort

```
void quick_sort(int vetor[], int ini, int fim) {
    int pos;

    if (ini < fim){
        pos = particionar(vetor, ini, fim);

        quick_sort(vetor, ini, pos - 1);
        quick_sort(vetor, pos, fim);
    }
}
```

Divisão e conquista novamente

Quick Sort

- **Divisão:** Separamos elementos pequenos e grandes
- **Conquista:** Ordenamos cada subvetor

QuickSort

```
void quick_sort(int vetor[], int ini, int fim) {
    int pos;

    if (ini < fim){
        pos = particionar(vetor, ini, fim);

        quick_sort(vetor, ini, pos - 1);
        quick_sort(vetor, pos, fim);
    }
}
```

Como particionar um vetor?

Ideia

- Escolhemos um valor pivô
- Separamos o vetor em duas partes:
  1. **primeira**: apenas elementos menores ou iguais ao pivô
  2. **segunda**: apenas elementos maiores que o pivô

Algoritmo

1. Obtemos o valor do pivô:
   - escolhemos sempre o valor do último elemento
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas

Como particionar um vetor?

Ideia

- Escolhemos um valor pivô
- Separamos o vetor em duas partes:
  1. **primeira**: apenas elementos menores ou iguais ao pivô
  2. **segunda**: apenas elementos maiores que o pivô

Algoritmo

1. Obtemos o valor do pivô:
   - escolhemos sempre o valor do último elemento
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas

Como particionar um vetor?

Ideia

- Escolhemos um valor pivô
- Separamos o vetor em duas partes:
  1. **primeira**: apenas elementos menores ou iguais ao pivô
  2. **segunda**: apenas elementos maiores que o pivô

Algoritmo

1. Obtemos o valor do pivô:
   - escolhemos sempre o valor do último elemento
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas

Como particionar um vetor?

Ideia

- Escolhemos um valor pivô
- Separamos o vetor em duas partes:
  1. **primeira**: apenas elementos menores ou iguais ao pivô
  2. **segunda**: apenas elementos maiores que o pivô

Algoritmo

1. Obtemos o valor do pivô:
   - escolhemos sempre o valor do último elemento
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas

Como particionar um vetor?

Ideia

- Escolhemos um valor pivô
- Separamos o vetor em duas partes:
  1. **primeira**: apenas elementos menores ou iguais ao pivô
  2. **segunda**: apenas elementos maiores que o pivô

Algoritmo

1. Obtemos o valor do pivô:
   - escolhemos sempre o valor do último elemento
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas

Como particionar um vetor?

Ideia

- Escolhemos um valor pivô
- Separamos o vetor em duas partes:
  1. **primeira**: apenas elementos menores ou iguais ao pivô
  2. **segunda**: apenas elementos maiores que o pivô

Algoritmo

1. Obtemos o valor do pivô:
   - escolhemos sempre o valor do último elemento
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas

Como particionar um vetor?

Ideia

- Escolhemos um valor pivô
- Separamos o vetor em duas partes:
  1. **primeira**: apenas elementos menores ou iguais ao pivô
  2. **segunda**: apenas elementos maiores que o pivô

Algoritmo

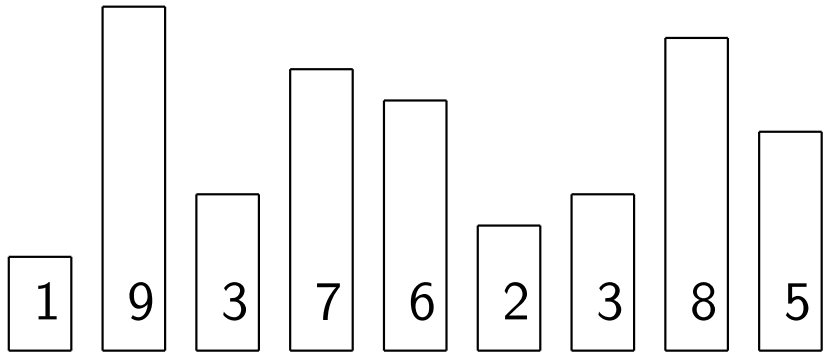
1. Obtemos o valor do pivô:
   - escolhemos sempre o valor do último elemento
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas

Algoritmo de particionamento

Particionar vetor

```
int particionar(int vetor[], int ini, int fim) {
    int pivo;

    pivo = vetor[fim];

    while (ini < fim) {
        while (ini < fim && vetor[ini] <= pivo)
            ini++;

        while (ini < fim && vetor[fim] > pivo)
            fim--;

        troca(&vetor[ini], &vetor[fim]);
    }

    return ini; // ini é a posição do primeiro elemento grande
}
```

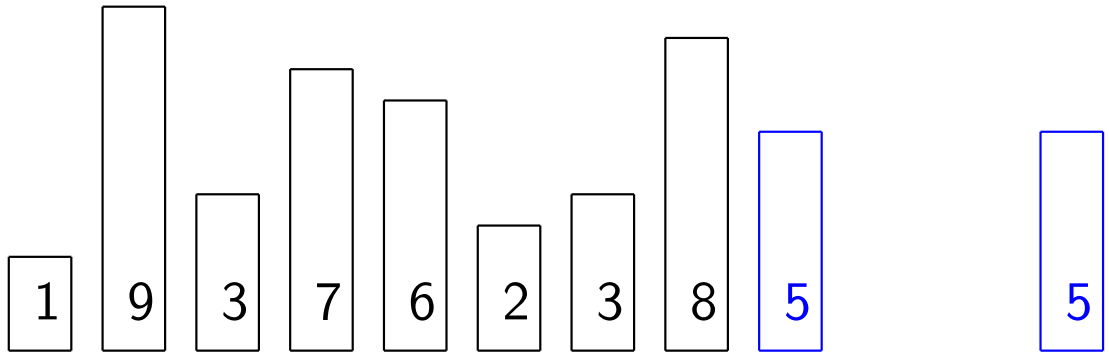
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - da frente para trás em busca de elementos menores que o pivô
   - de trás para a frente em busca de elementos maiores que o pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

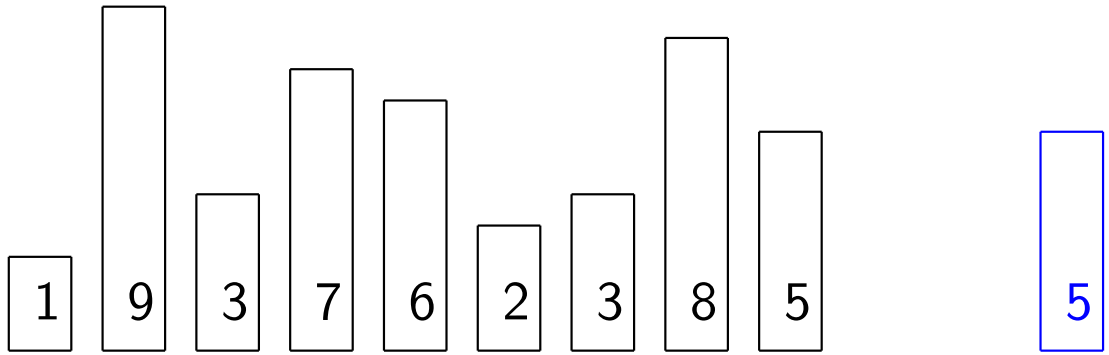
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - da frente para trás em busca de elemento menor que o pivô
   - de trás para a frente em busca de elemento maior que o pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

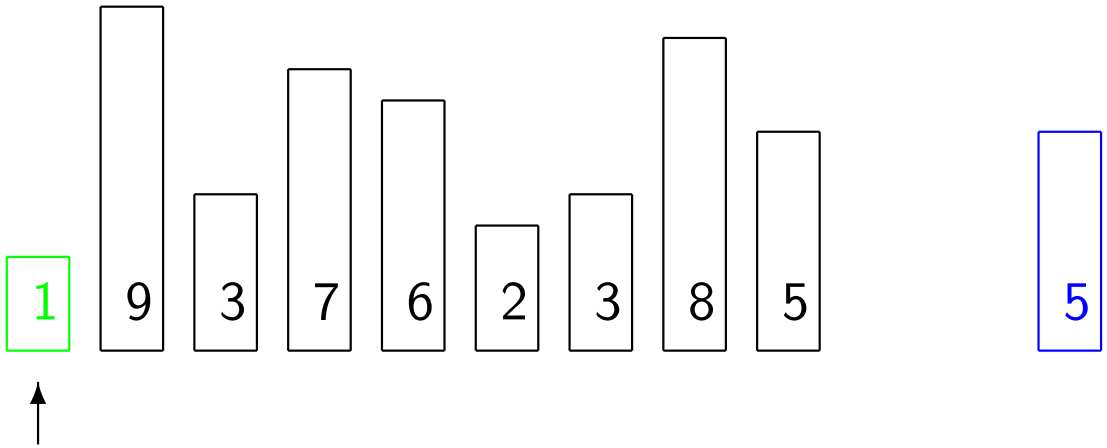
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

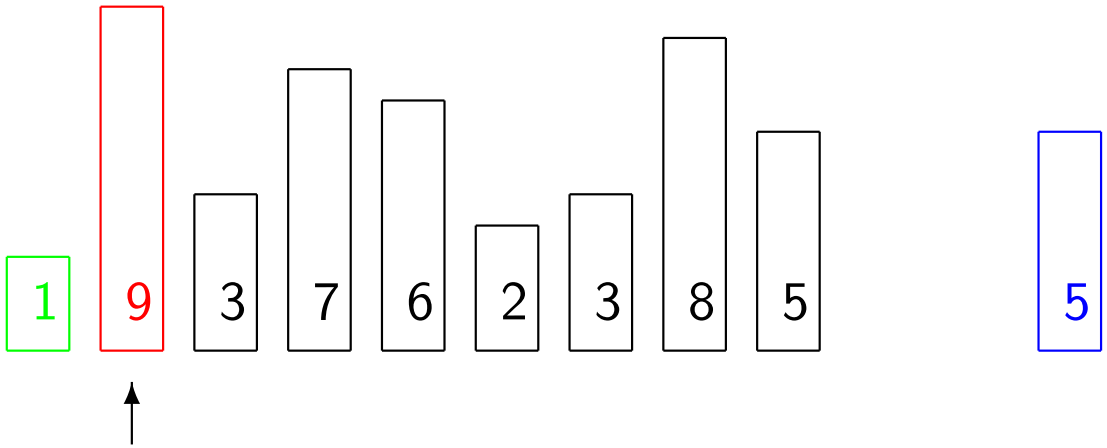
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

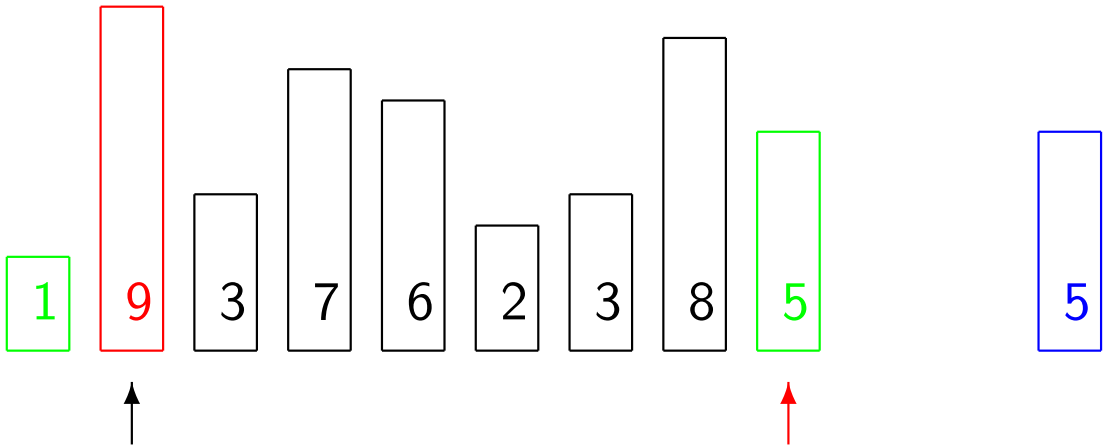
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

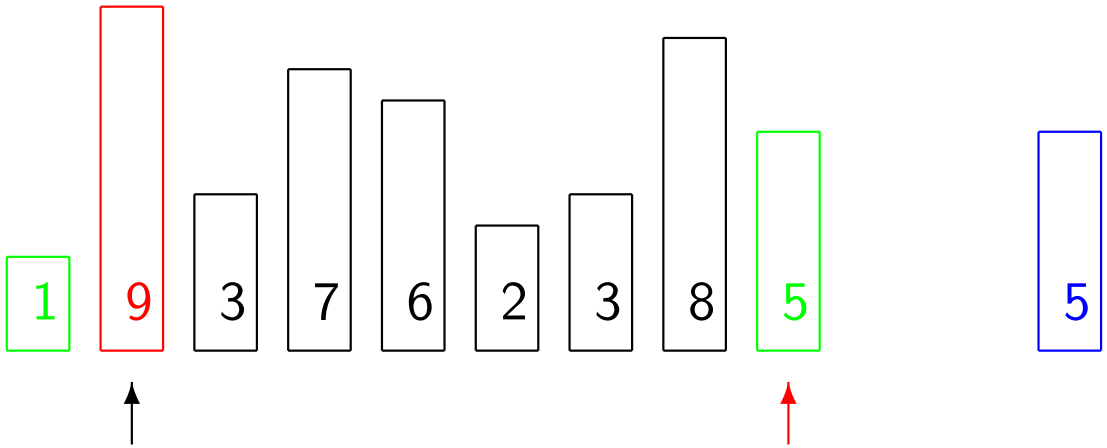
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

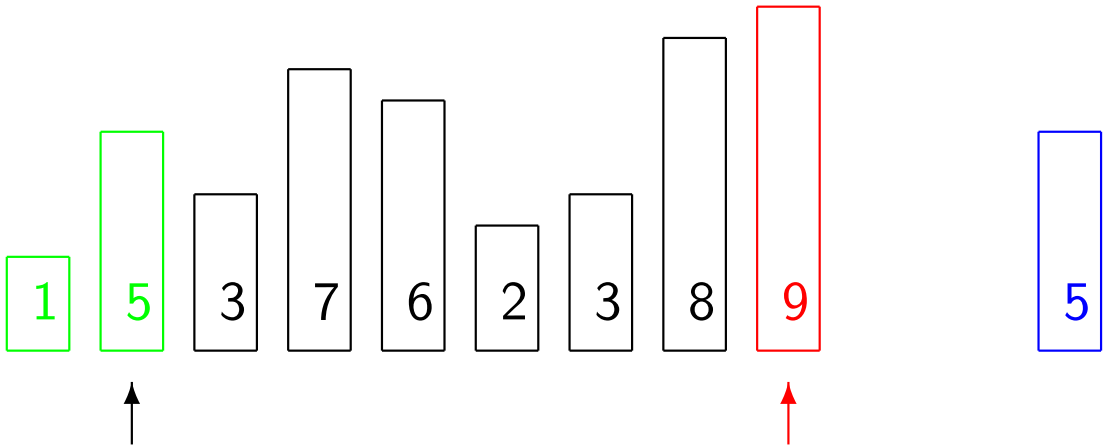
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

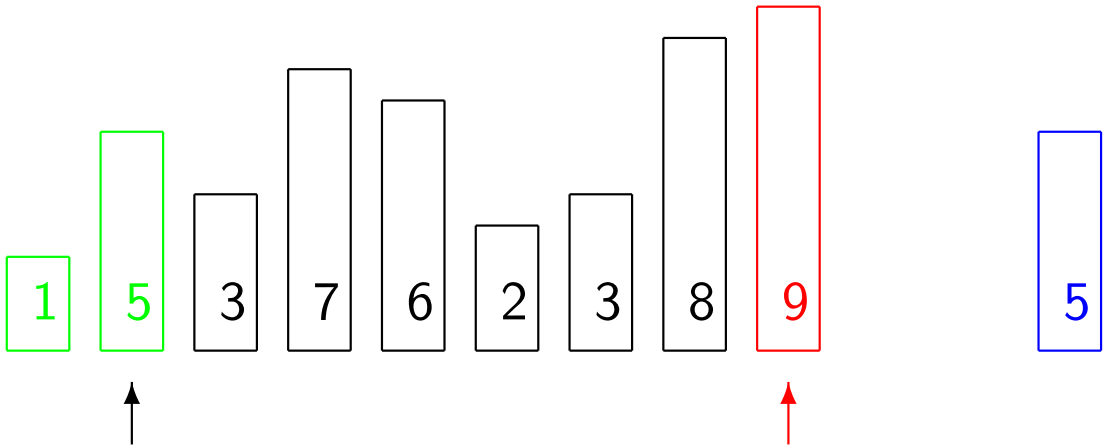
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

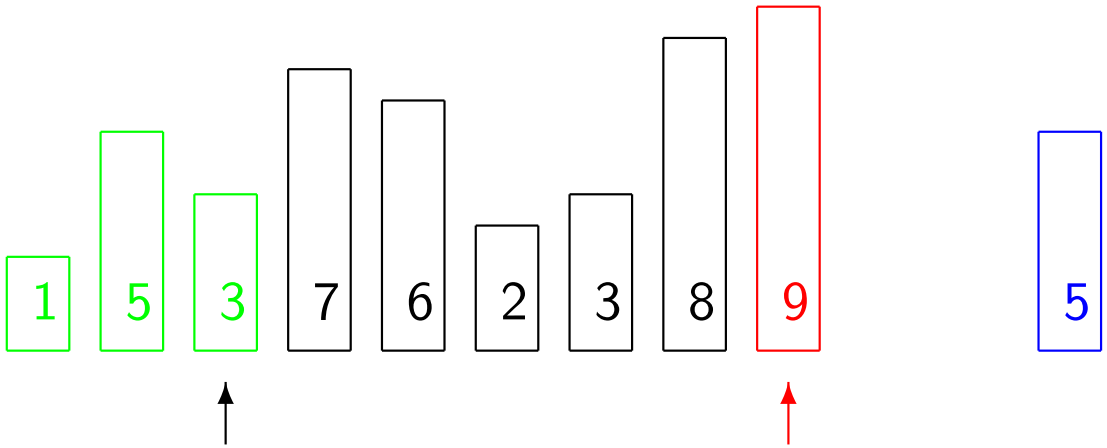
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

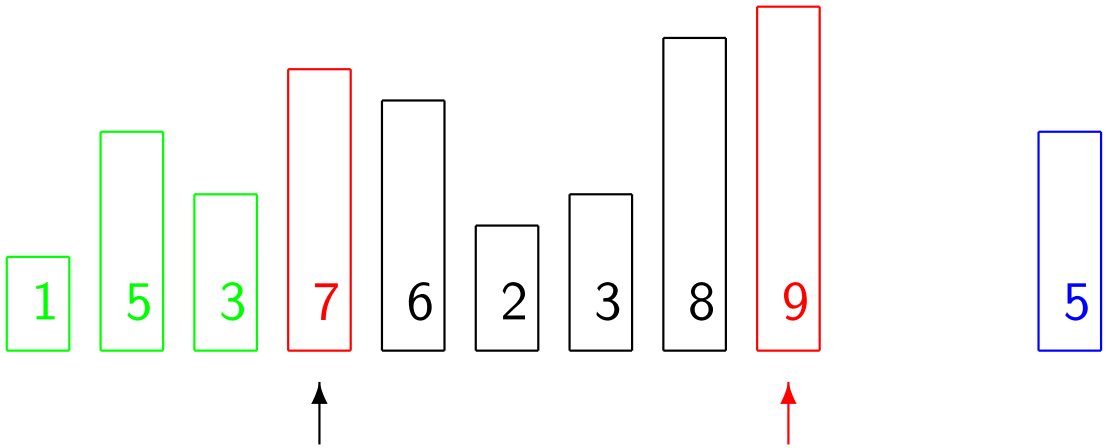
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

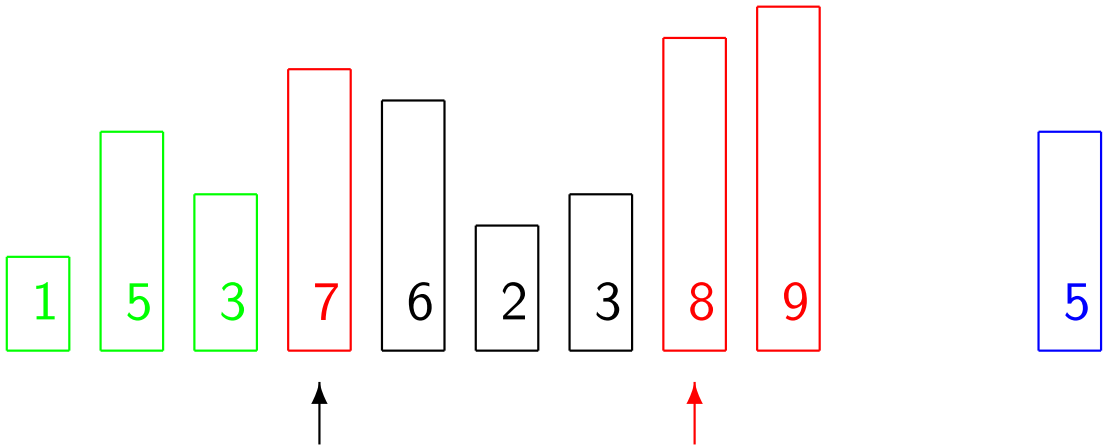
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

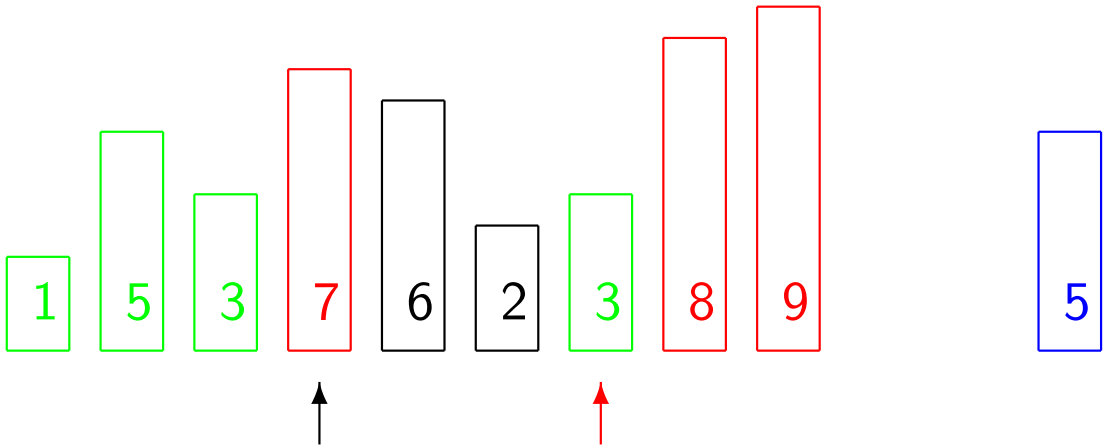
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

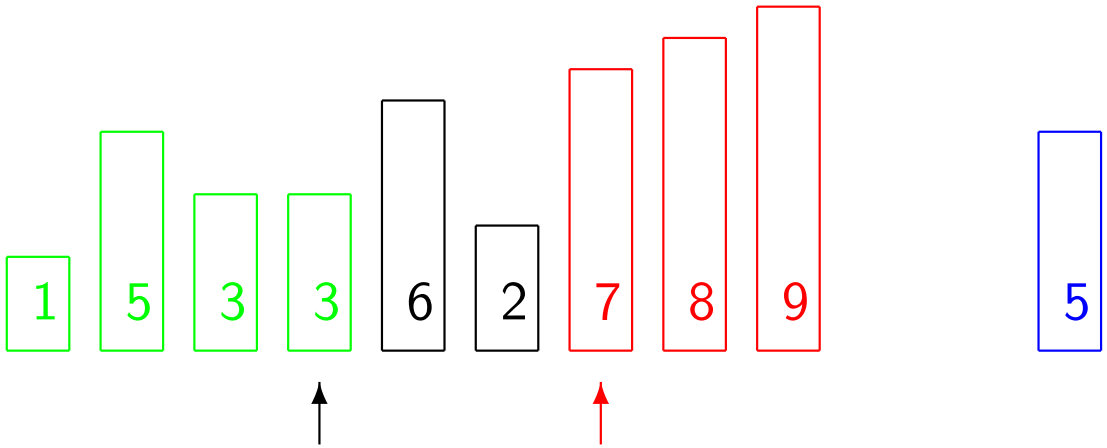
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

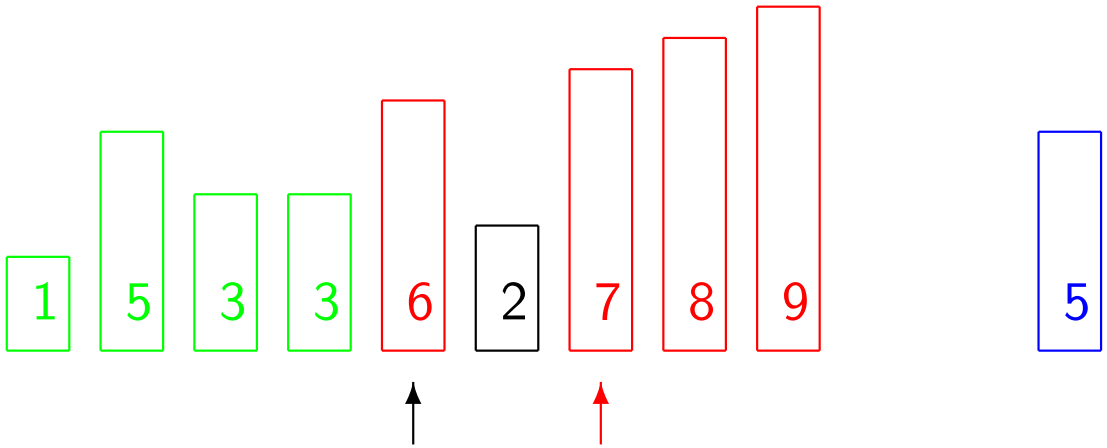
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

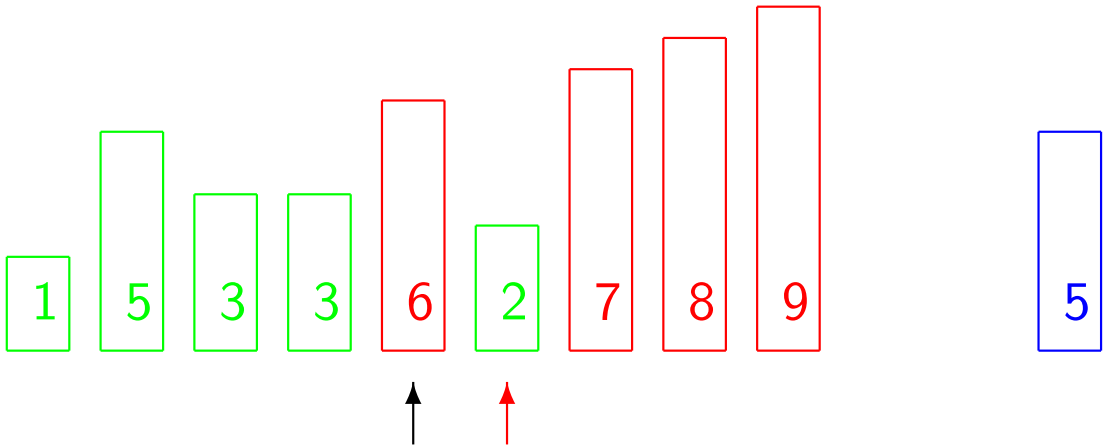
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

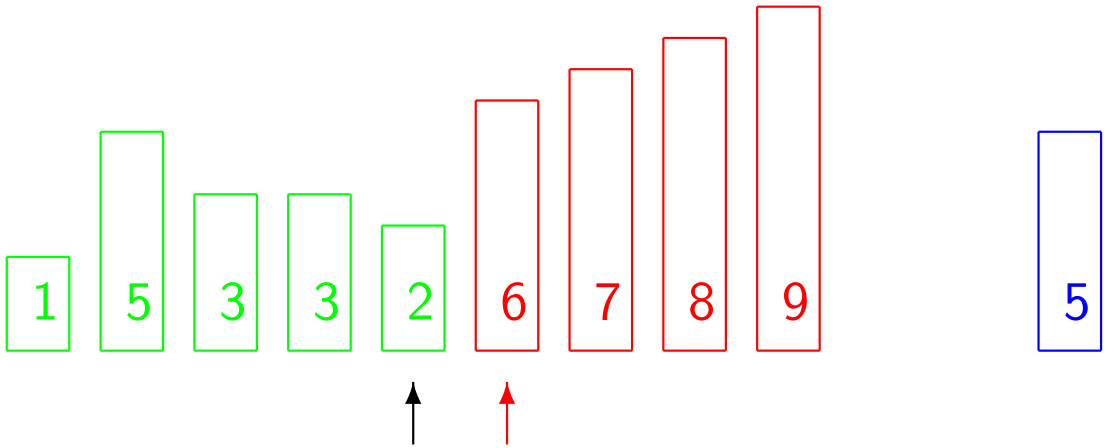
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

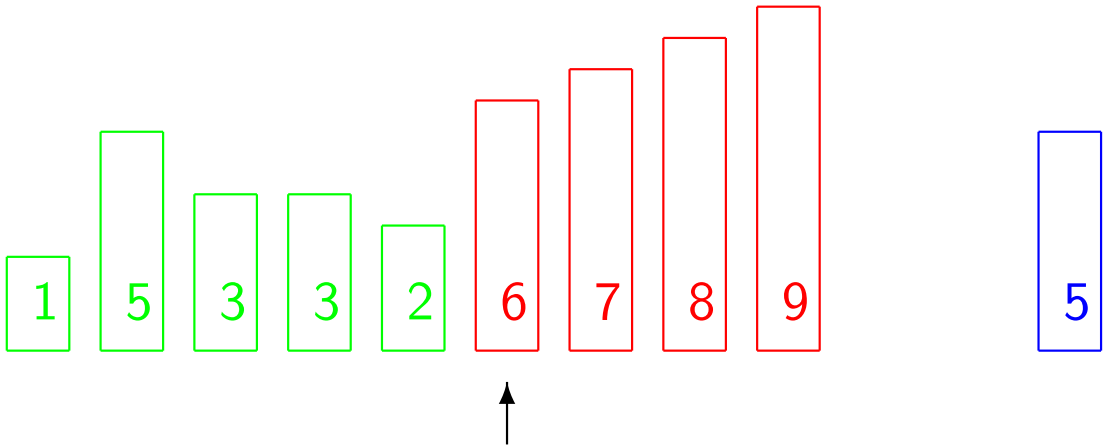
Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

Particionamento



Algoritmo

1. Obtemos o valor do pivô:
2. Procuramos elementos fora de ordem:
   - **do início ao fim**: em busca de elementos maiores que o pivô
   - **do fim ao início**: em busca de elementos menores ou iguais ao pivô
3. Trocamos os elementos em posições erradas
4. Continuamos passo 2 até índices se encontrarem

Exercício

1. Aplique o algoritmo de particionamento sobre o vetor $(13, 19, 9, 5, 12, 21, 7, 4, 11, 2, 6, 6)$ com pivô igual a 6.
2. Modifique o algoritmo QuickSort para ordenar vetores em ordem decrescente.