

# Pesquisa e Ordenação de Dados

## Unidade 2:

### Ordenação Interna: Métodos Simples

# Método de Ordenação

- Um método de ordenação é um **algoritmo** que tem por objetivo dispor um conjunto de valores de tal modo que, após o processo de ordenação, os dados satisfaçam à propriedade  $V_1 \leq V_2 \leq V_3 \leq \dots \leq V_n$  (ordenação crescente) ou  $V_1 \geq V_2 \geq V_3 \geq \dots \geq V_n$  (ordenação decrescente).
- Um método de ordenação pode ser aplicado a qualquer conjunto de dados cujos elementos sejam **comparáveis** - ou seja, onde seja possível determinar uma ordem entre os elementos
  - numérica, alfabética, cronológica, ...

# Método de Ordenação

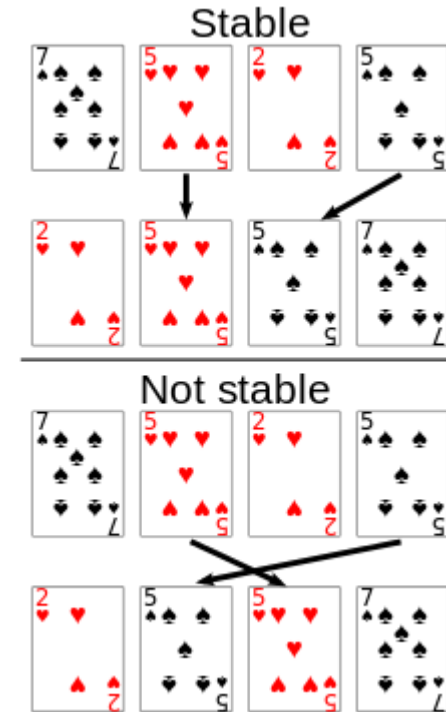
- Pode ser aplicado também a conjuntos de registros com vários campos
  - Neste caso, a ordenação incidirá sobre um dos campos, denominado **chave de ordenação**
    - ex: conjuntos de registros de alunos ordenados pelo campo número de matrícula

matricula: 5 nome: João cidade: Chapecó	matricula: 7 nome: Maria cidade: Chapecó	matricula: 8 nome: Pedro cidade: Xanxerê	matricula: 10 nome: Ana cidade: Chapecó	matricula: 11 nome: Pedro cidade: Xaxim	matricula: 15 nome: Bia cidade: Maravilha
---	--	--	---	---	---

- Sempre que um algoritmo de ordenação, após comparar duas chaves, necessitar trocá-las de posição, lembre que é necessário trocar também todos os dados relativos àquela chave.

# Características dos Métodos de Ordenação

- Estável x Não estável
  - Um método estável preserva a **ordem relativa** dos elementos que possuem o mesmo valor para a chave de ordenação.
  - Um método não estável não preserva esta ordem.
- *In place* (ou *in situ*)
  - Os valores são permutados **dentro da própria estrutura** original (vetor), com auxílio de algumas poucas variáveis escalares adicionais.



# Ordenação Interna

- **Baseada em comparações:**

- **Métodos simples:** implementações simples e fáceis de entender, porém, não tão eficientes
  - $O(n^2)$
  - Bubble Sort, Selection Sort, Insertion Sort
- **Métodos eficientes:** utilizam estratégias e/ou estruturas de dados mais sofisticadas
  - $O(n \log n)$
  - Merge Sort, Quick Sort, Heap Sort

- **Não baseada em comparações:**

- **Métodos por contagem**
  - $O(n)$
  - Counting Sort, Radix Sort, Bucket Sort

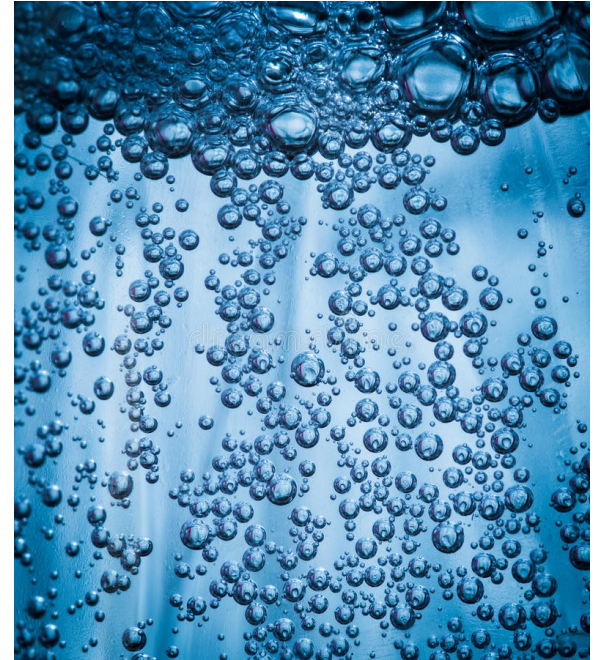
# Convenções de notação para a disciplina

- Vamos descrever os algoritmos de ordenação considerando o seguinte cenário:
  - A entrada é um vetor de números inteiros chamado **A** cujos elementos precisam ser ordenados sempre de forma **crescente**;
  - A saída é o mesmo vetor **A** com seus elementos na ordem especificada;
  - Sendo **n** o número de elementos, os índices deste vetor sempre iniciam em **0** e vão até **n-1**;
  - Os passos necessários para trocar dois elementos de posição não serão explicitados. Usaremos apenas a notação **troca(x, y)**;

# Pesquisa e Ordenação de Dados

Unidade 2.1:

**Bubble Sort**



# Bubble Sort

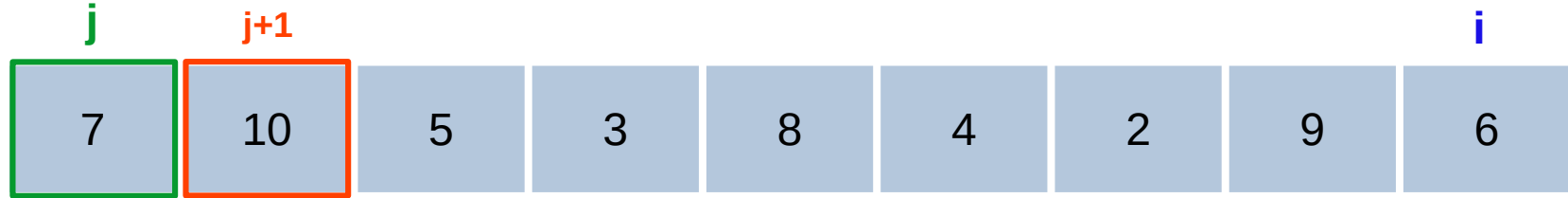
- Compara **pares de elementos adjacentes**; se o elemento da esquerda é maior do que o da direita, troca-os de posição (*swap*).
- Ordenação “**bolha**”
  - A cada iteração, os maiores elementos vão sendo deslocados para o final do vetor de forma contínua, passando por todas as posições no caminho
- Executa até  **$n - 1$**  iterações (dependendo da implementação)
- Cada iteração (ou varredura):
  - Percorre a lista a partir do início, comparando cada elemento com seu sucessor, trocando-os de posição se estiverem fora de ordem;
  - A iteração termina quando encontra um elemento que já foi posicionado (parte ordenada);
- Ao final da iteração  **$K$** , os  **$K$**  maiores elementos estarão nas  **$K$**  últimas posições, em ordem crescente.



# Bubble Sort

## Exemplo (1)

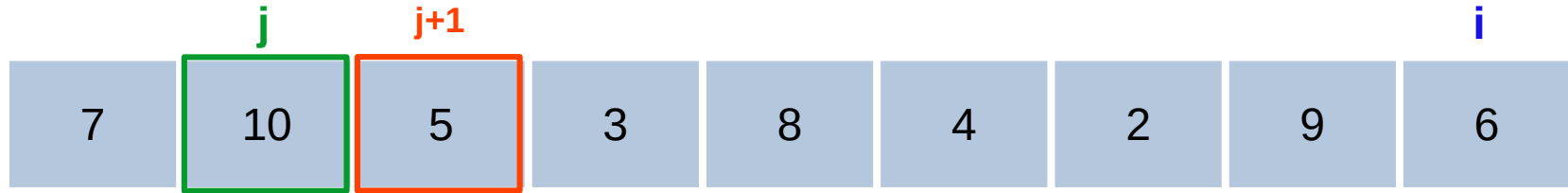
Iteração 1



# Bubble Sort

## Exemplo (2)

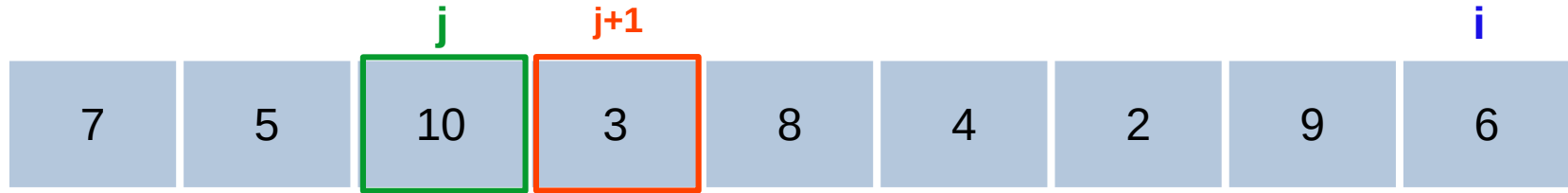
Iteração 1



# Bubble Sort

## Exemplo (3)

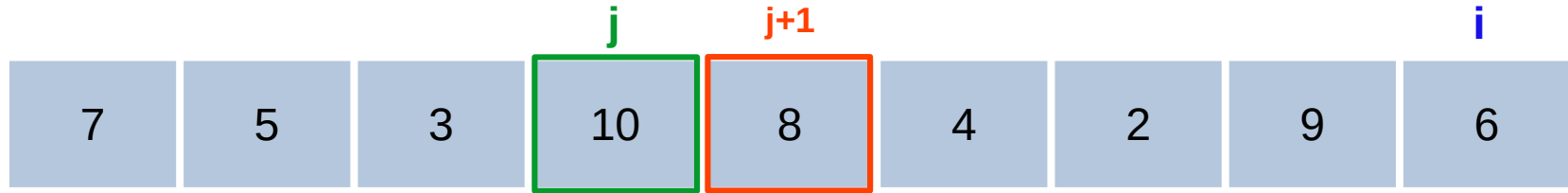
Iteração 1



# Bubble Sort

## Exemplo (4)

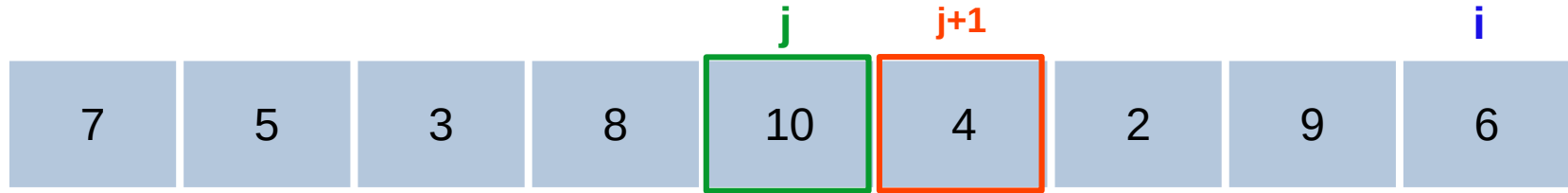
Iteração 1



# Bubble Sort

## Exemplo (5)

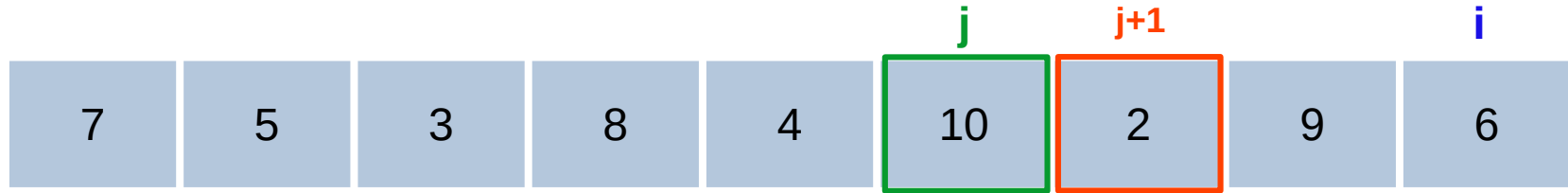
Iteração 1



# Bubble Sort

## Exemplo (6)

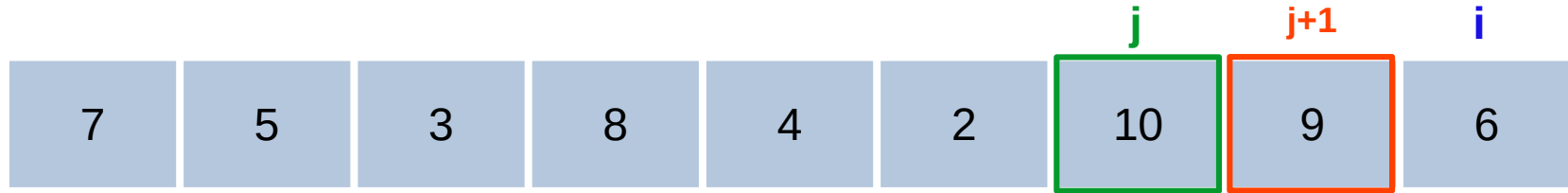
Iteração 1



# Bubble Sort

## Exemplo (7)

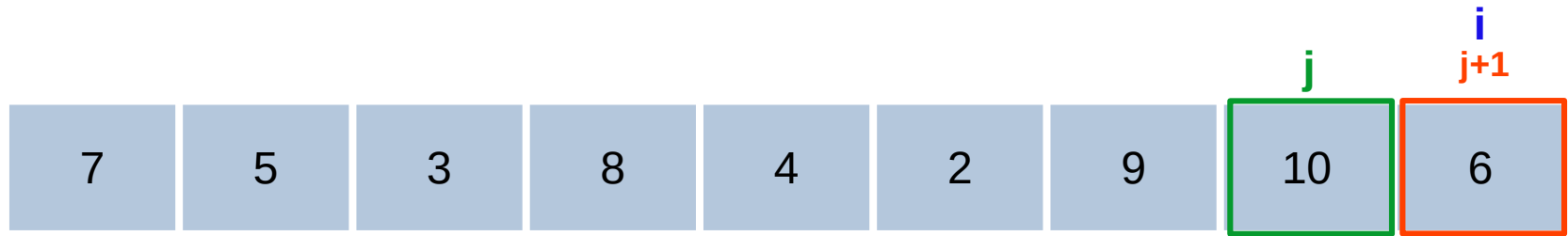
Iteração 1



# Bubble Sort

## Exemplo (8)

Iteração 1

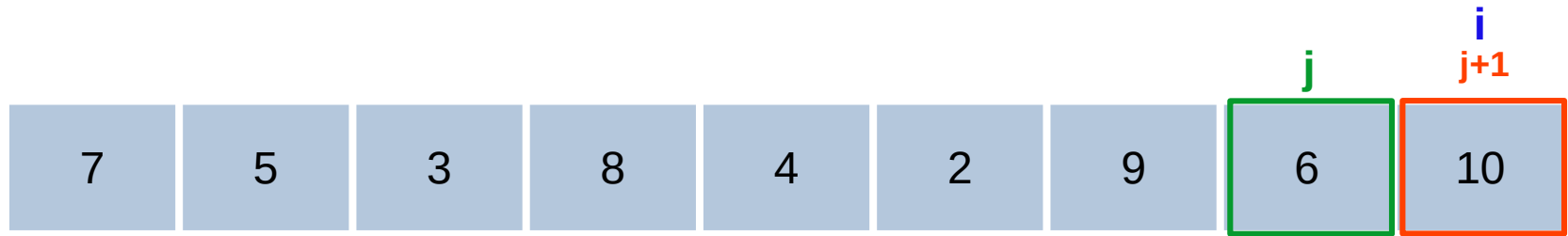




# Bubble Sort

## Exemplo (9)

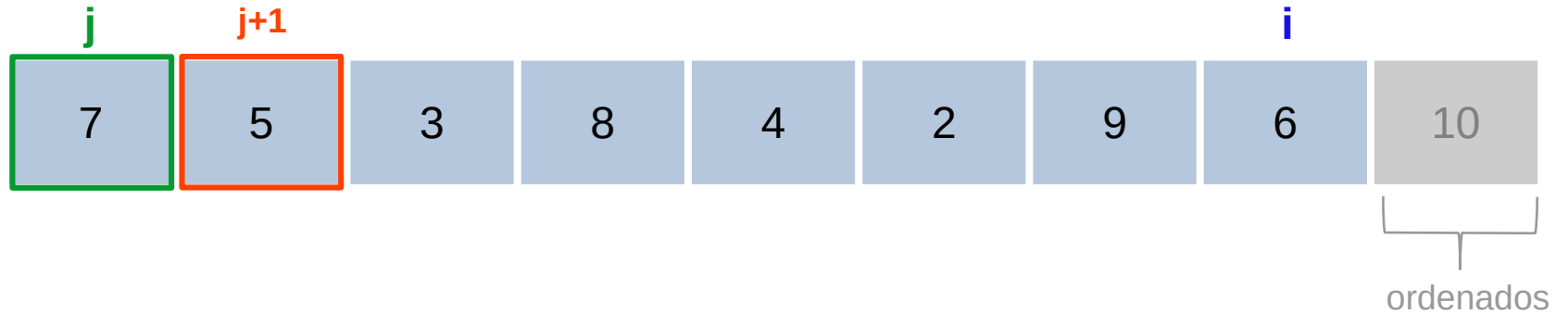
Iteração 1



# Bubble Sort

## Exemplo (10)

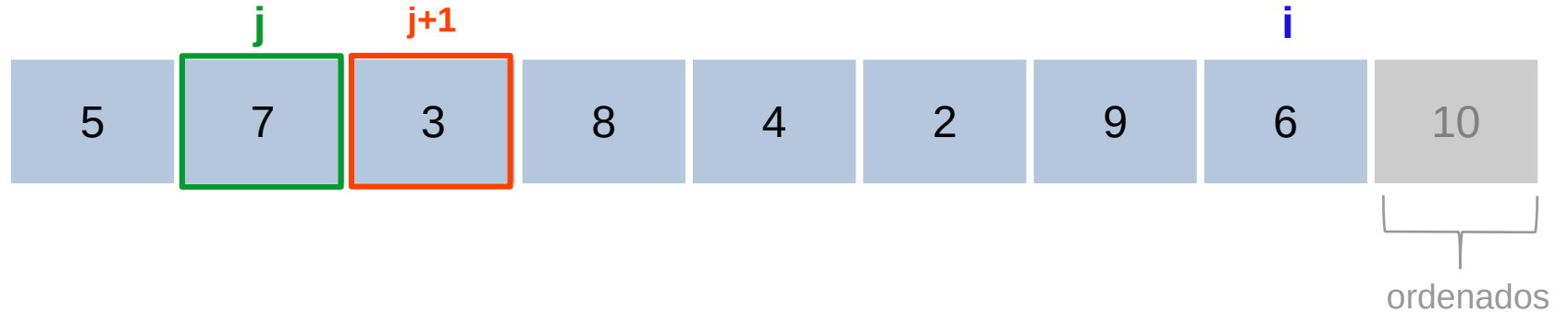
Iteração 2



# Bubble Sort

## Exemplo (11)

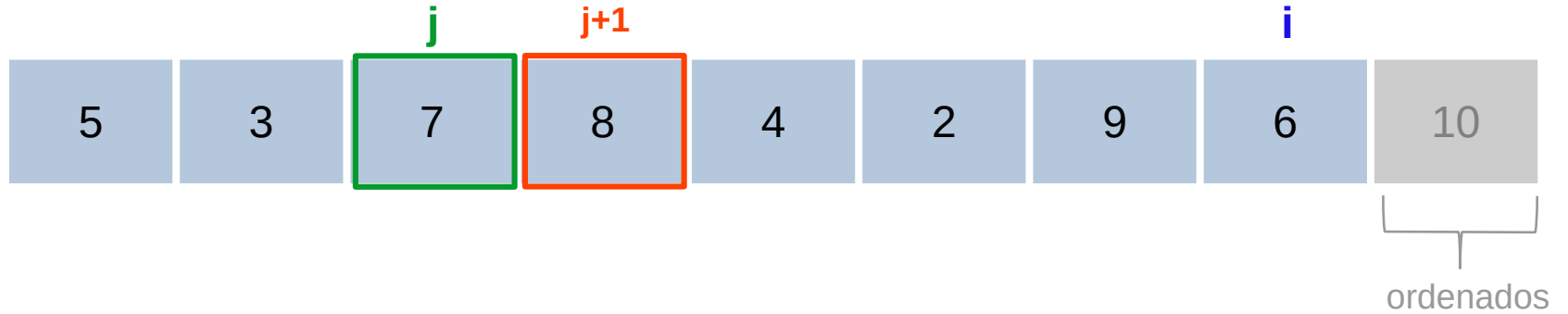
Iteração 2



# Bubble Sort

## Exemplo (12)

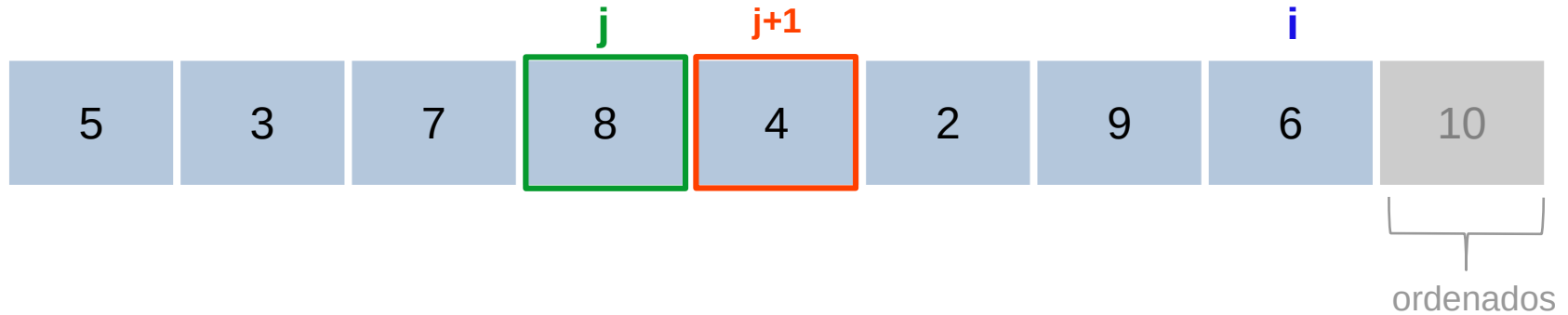
Iteração 2



# Bubble Sort

## Exemplo (13)

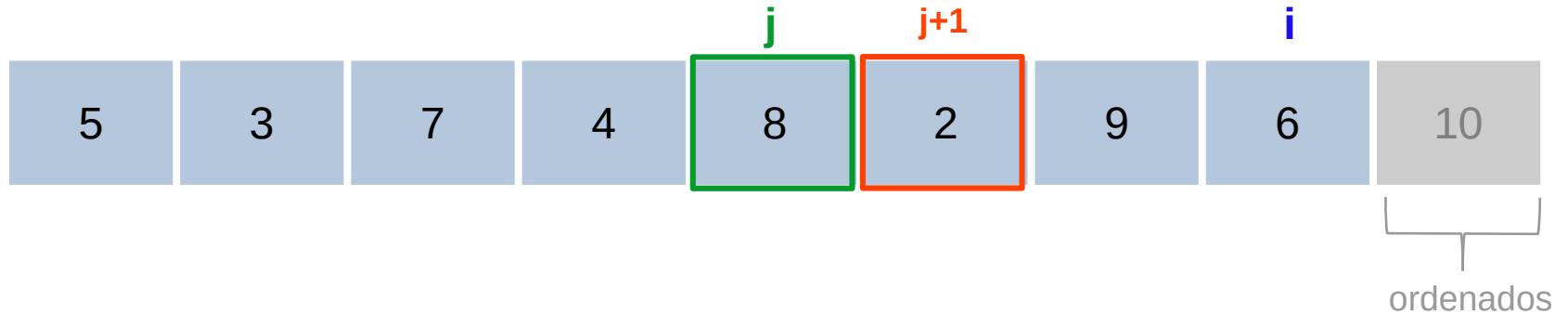
Iteração 2



# Bubble Sort

## Exemplo (14)

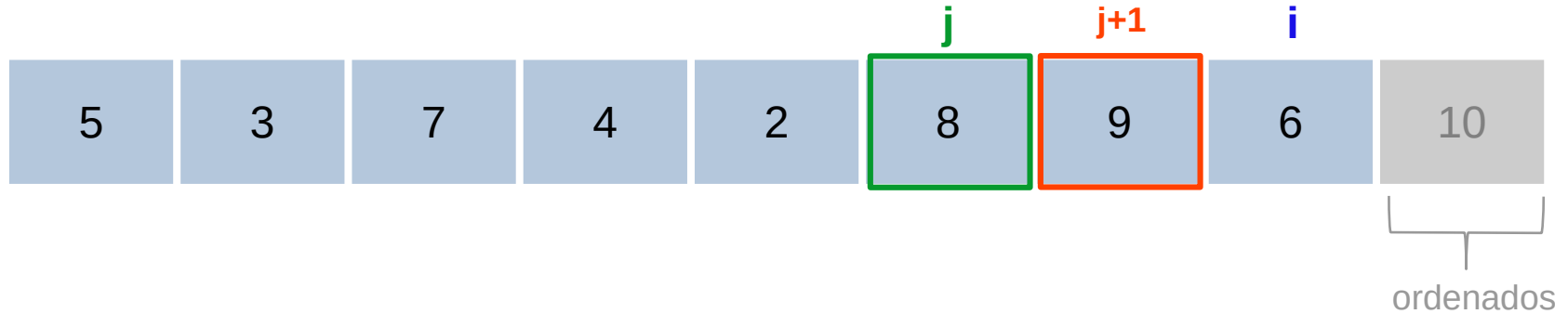
Iteração 2



# Bubble Sort

## Exemplo (15)

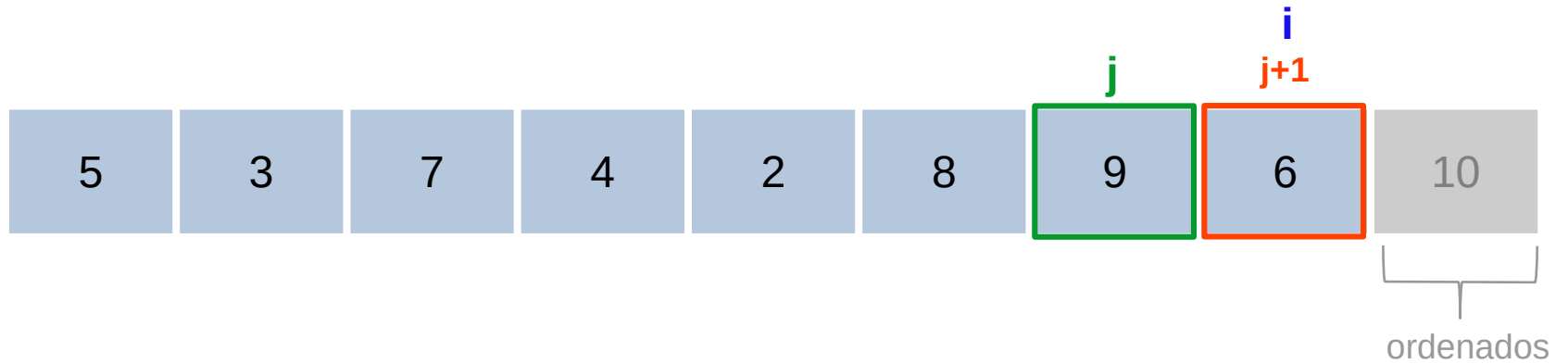
Iteração 2



# Bubble Sort

## Exemplo (16)

Iteração 2

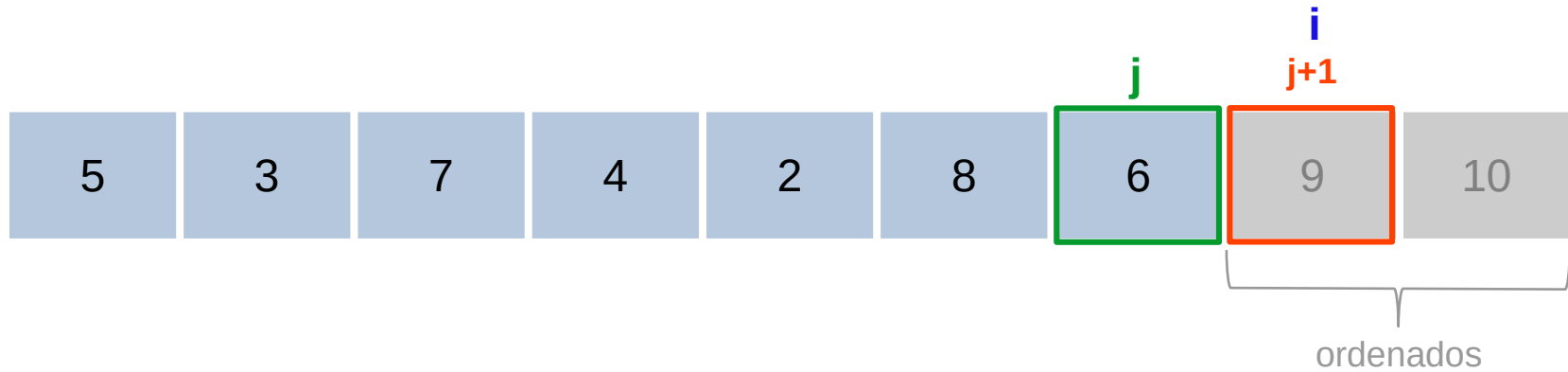




# Bubble Sort

## Exemplo (17)

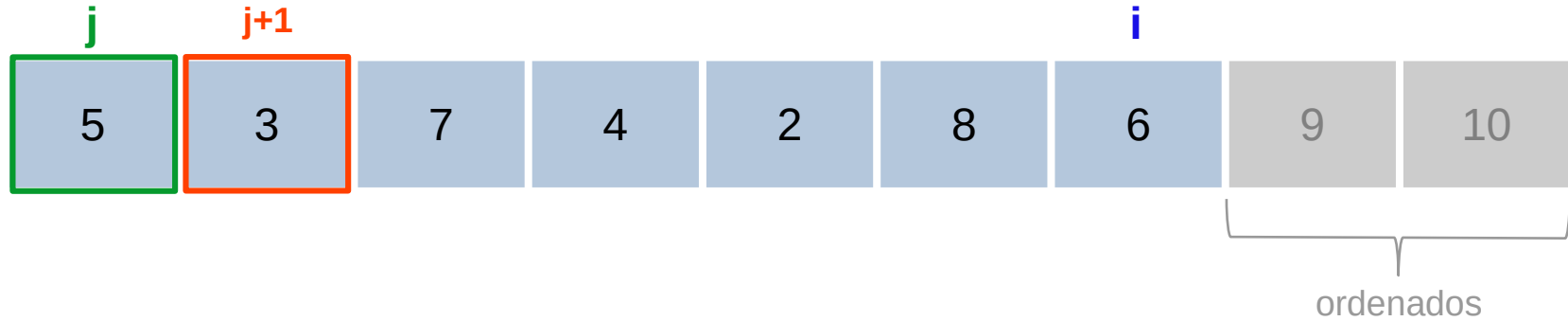
Iteração 2



# Bubble Sort

## Exemplo (18)

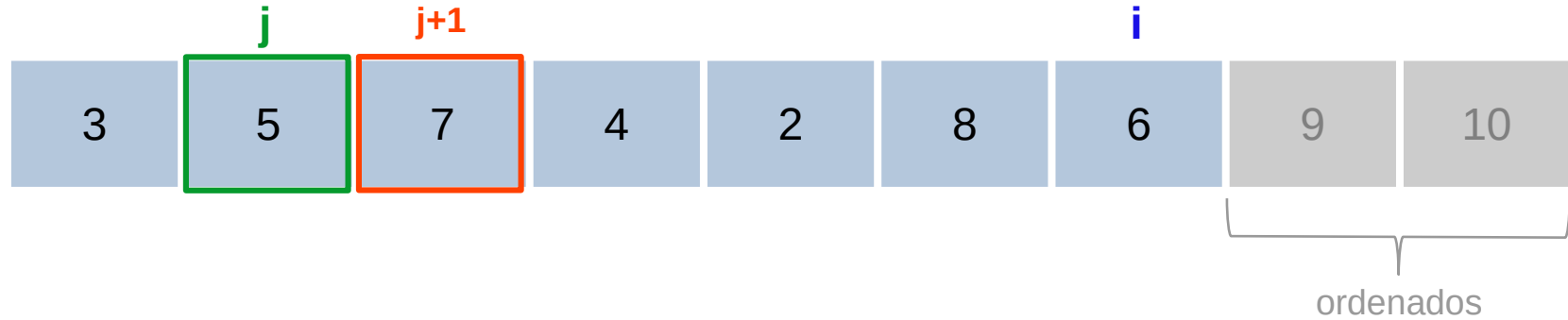
Iteração 3



# Bubble Sort

## Exemplo (19)

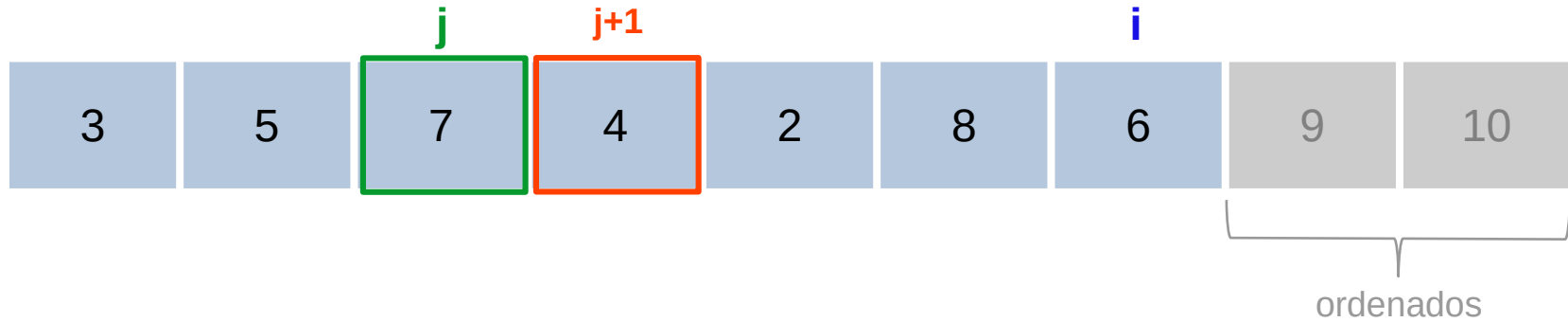
Iteração 3



# Bubble Sort

## Exemplo (20)

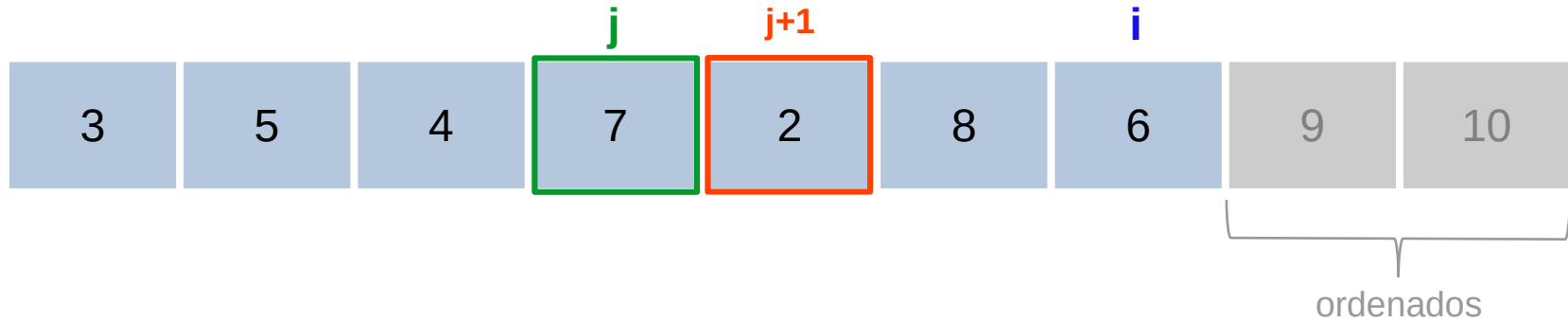
Iteração 3



# Bubble Sort

## Exemplo (21)

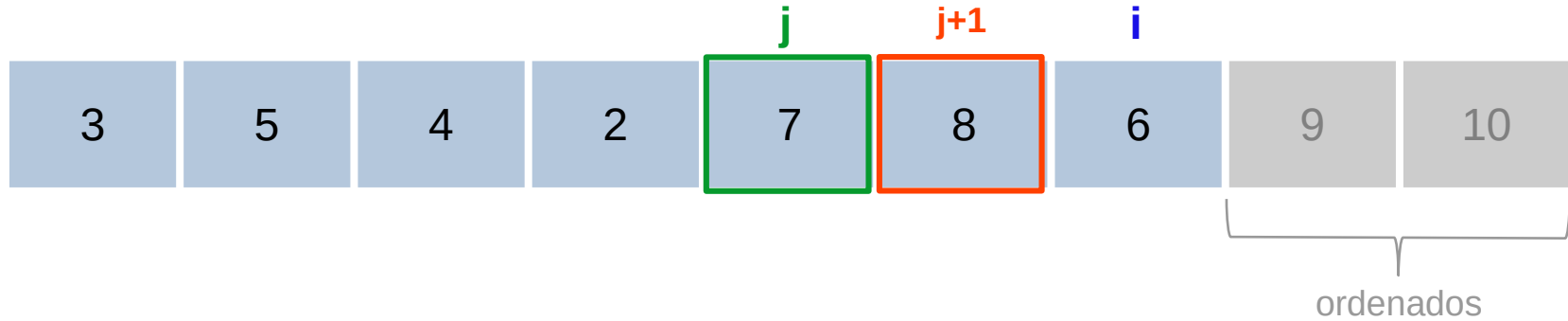
Iteração 3



# Bubble Sort

## Exemplo (22)

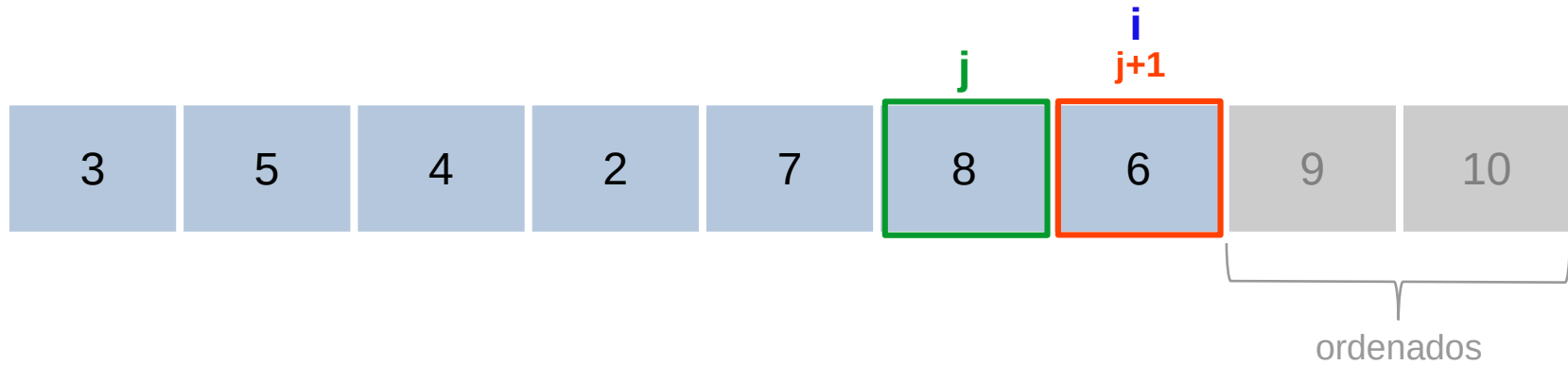
Iteração 3



# Bubble Sort

## Exemplo (23)

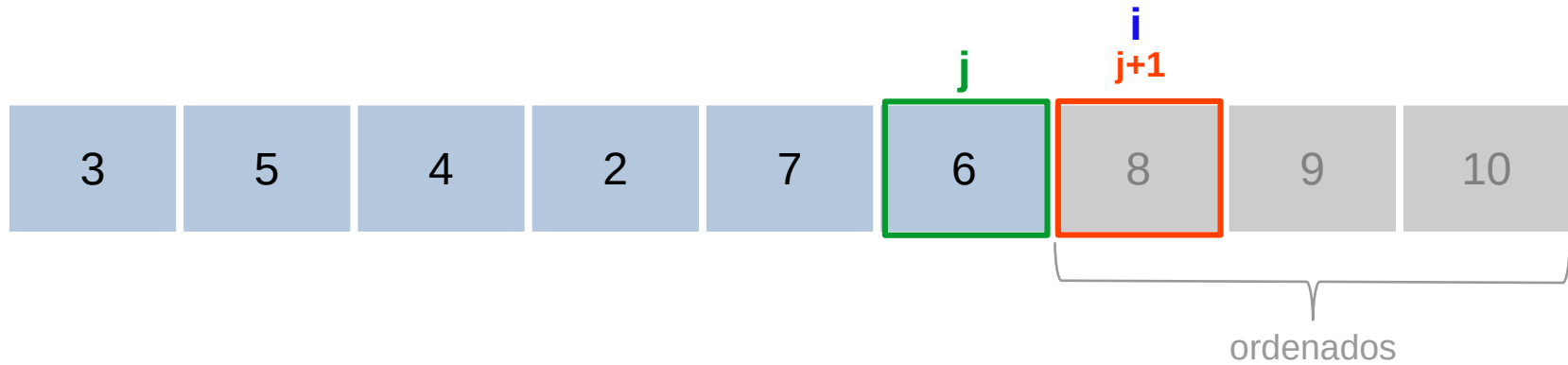
Iteração 3



# Bubble Sort

## Exemplo (24)

Iteração 3

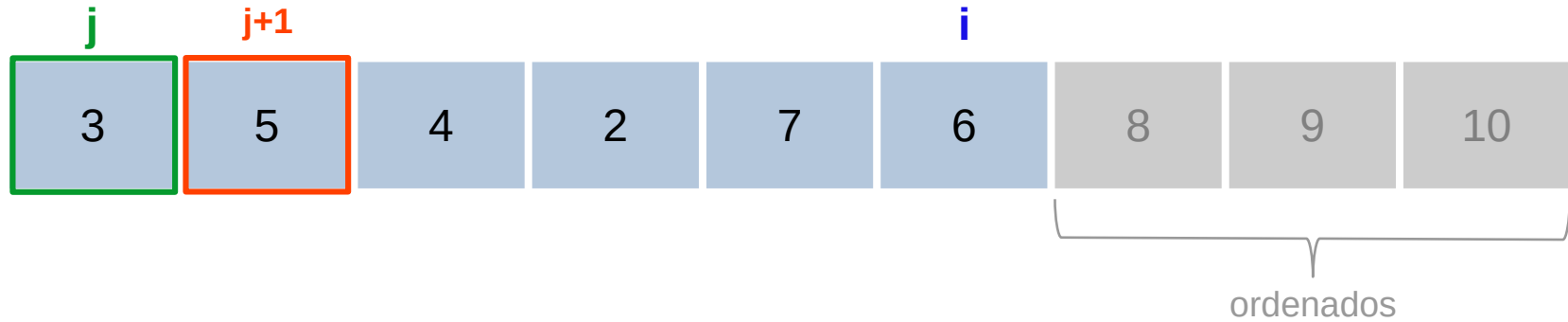




# Bubble Sort

## Exemplo (25)

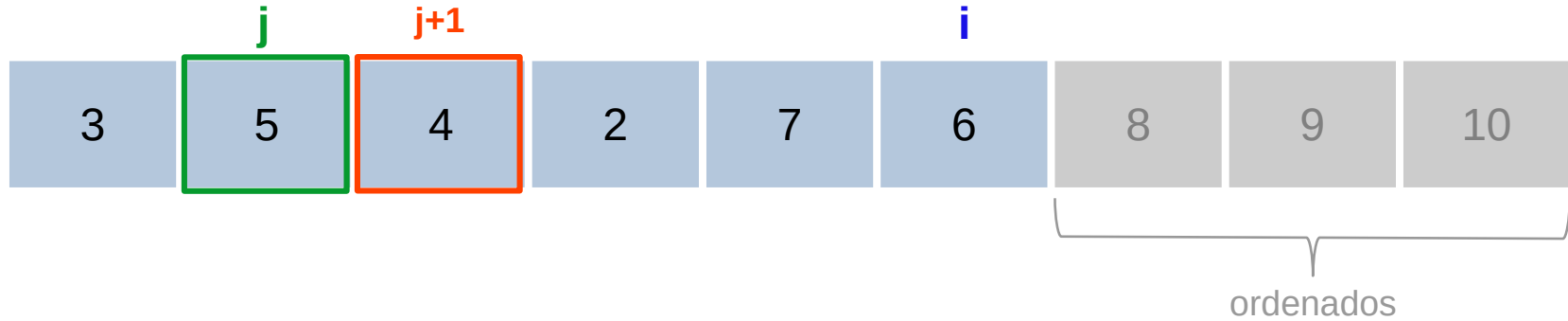
Iteração 4



# Bubble Sort

## Exemplo (26)

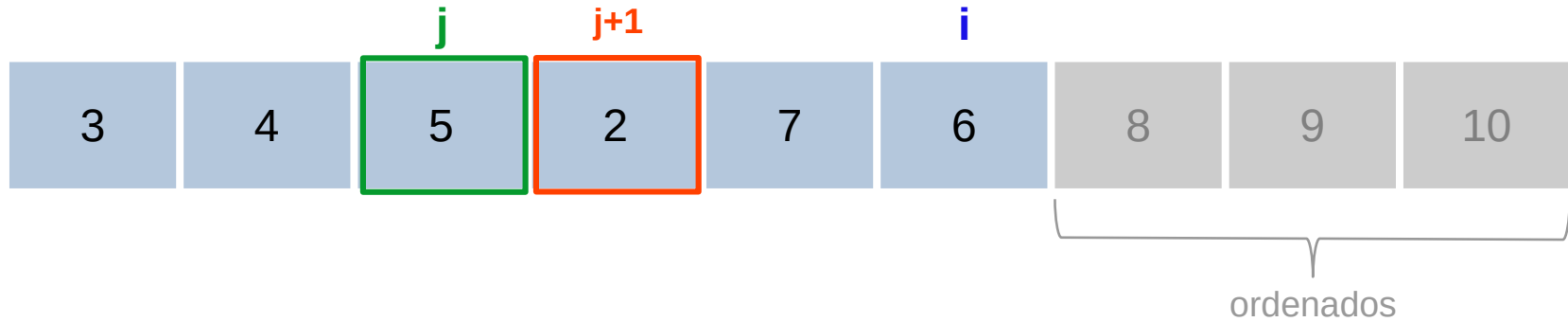
Iteração 4



# Bubble Sort

## Exemplo (27)

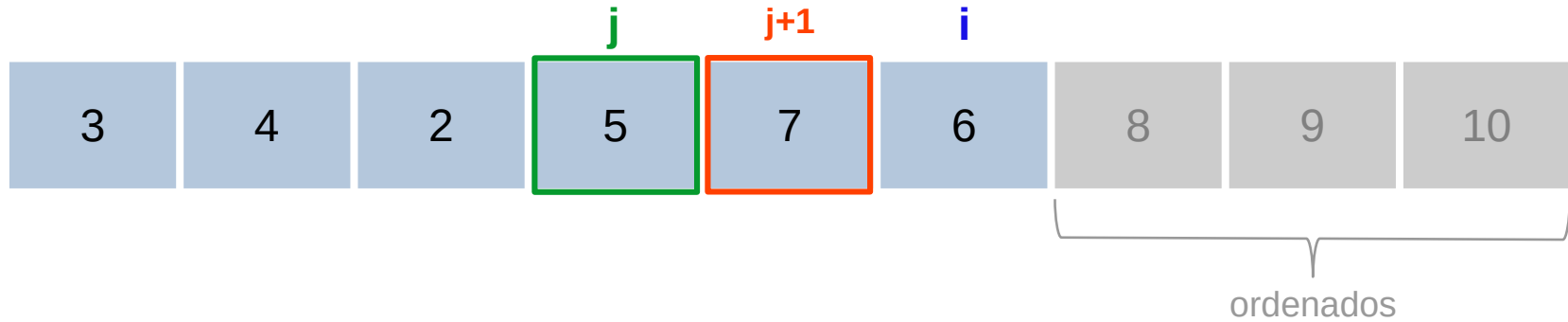
Iteração 4



# Bubble Sort

## Exemplo (28)

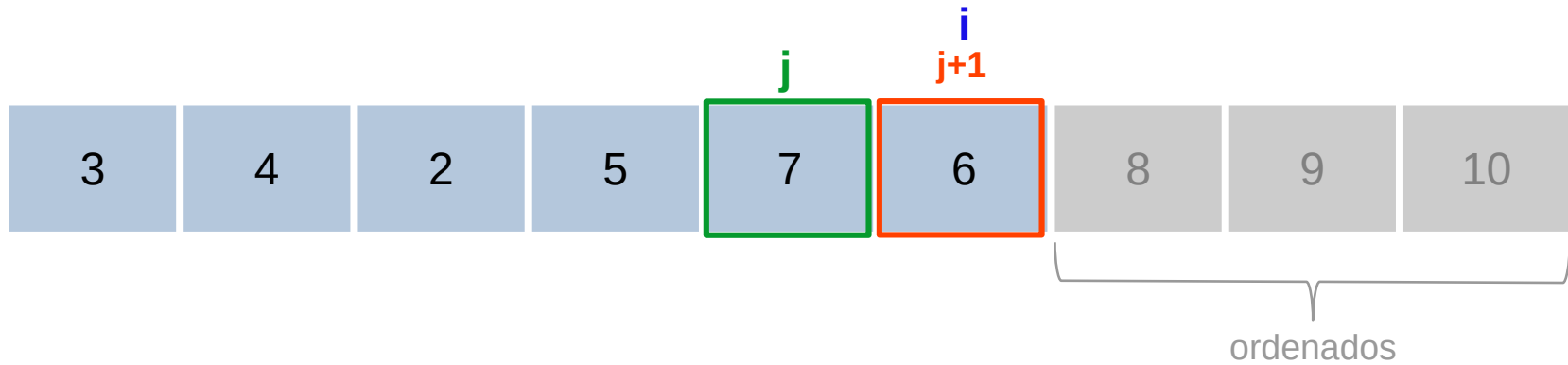
Iteração 4



# Bubble Sort

## Exemplo (29)

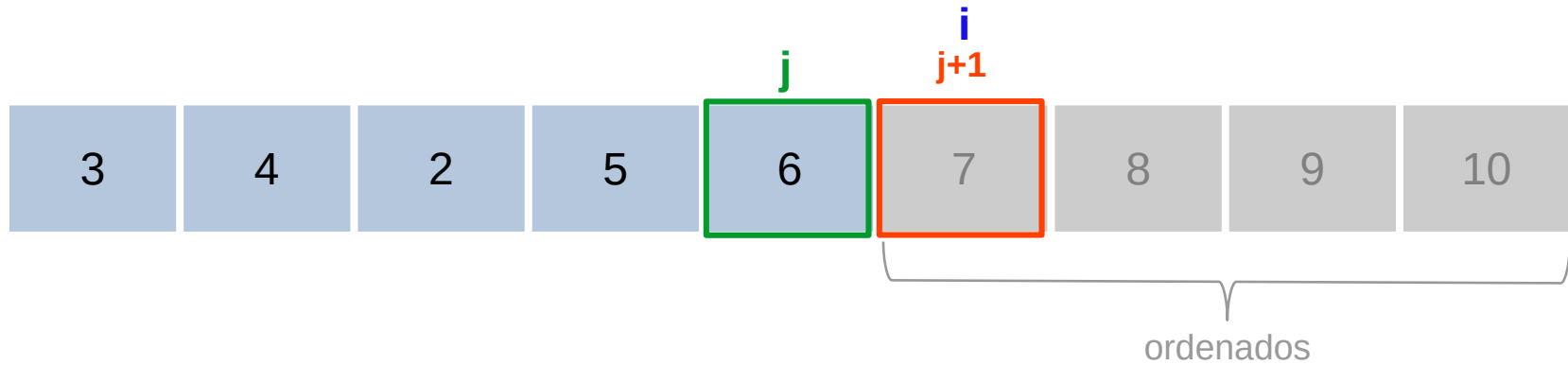
Iteração 4



# Bubble Sort

## Exemplo (30)

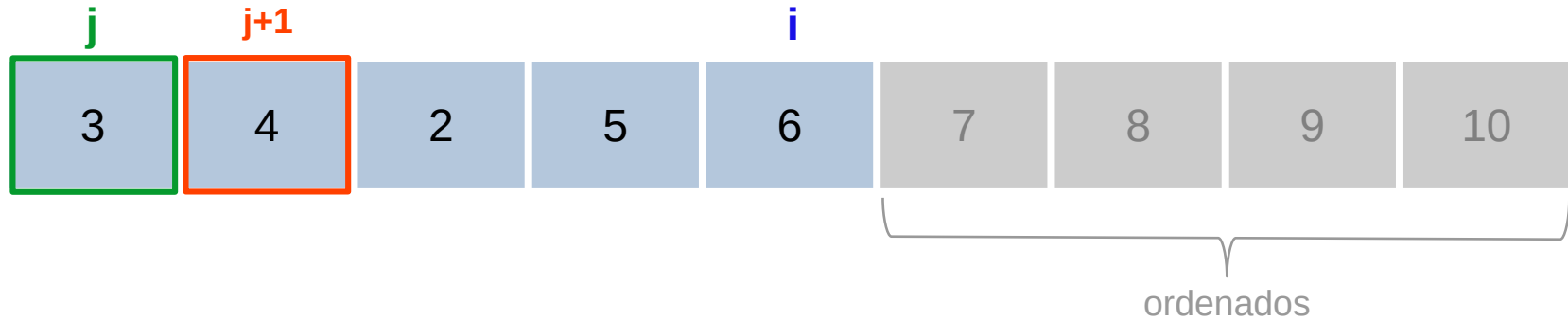
Iteração 4



# Bubble Sort

## Exemplo (31)

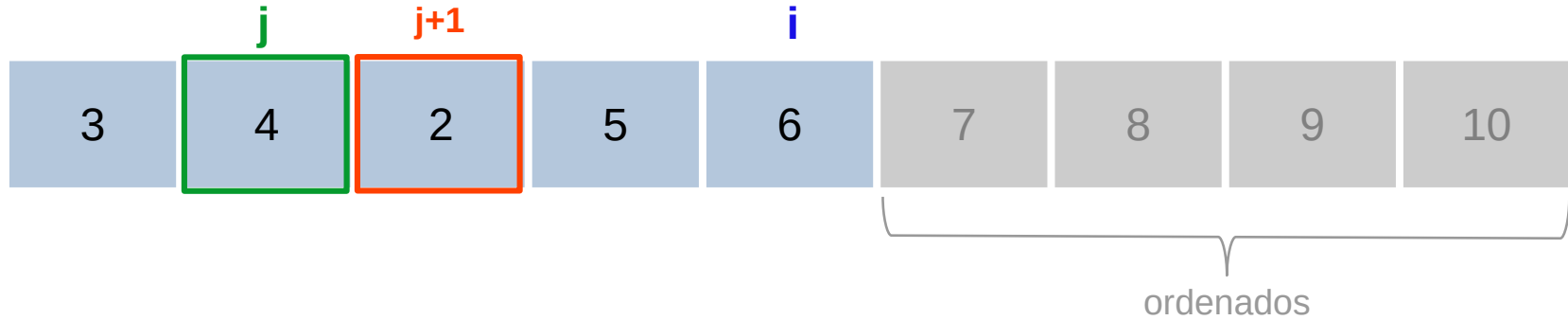
Iteração 5



# Bubble Sort

## Exemplo (32)

Iteração 5

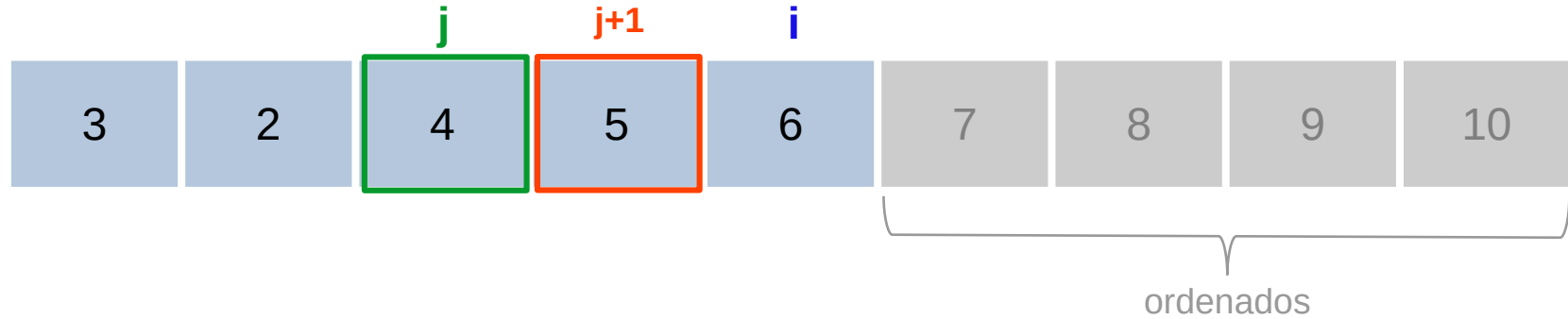




# Bubble Sort

## Exemplo (33)

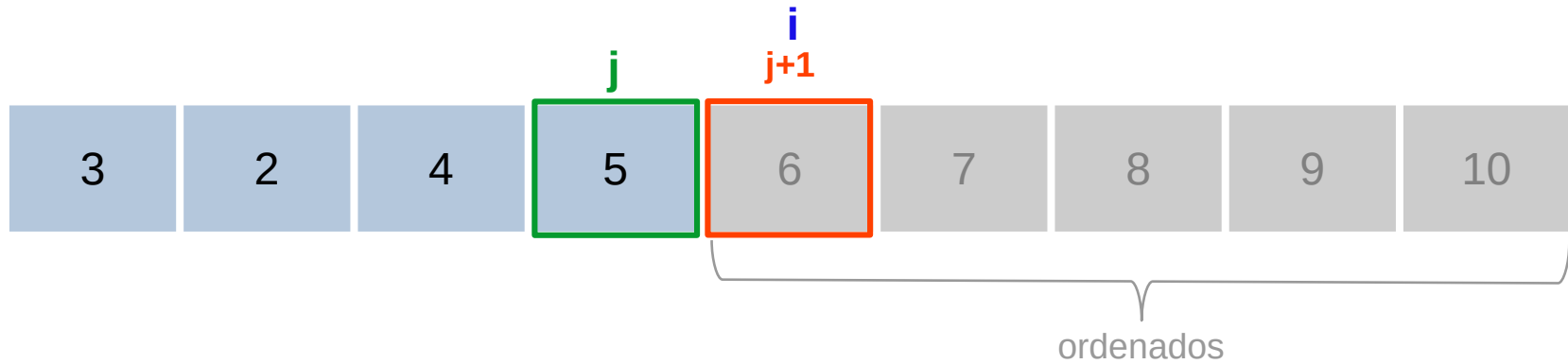
Iteração 5



# Bubble Sort

## Exemplo (34)

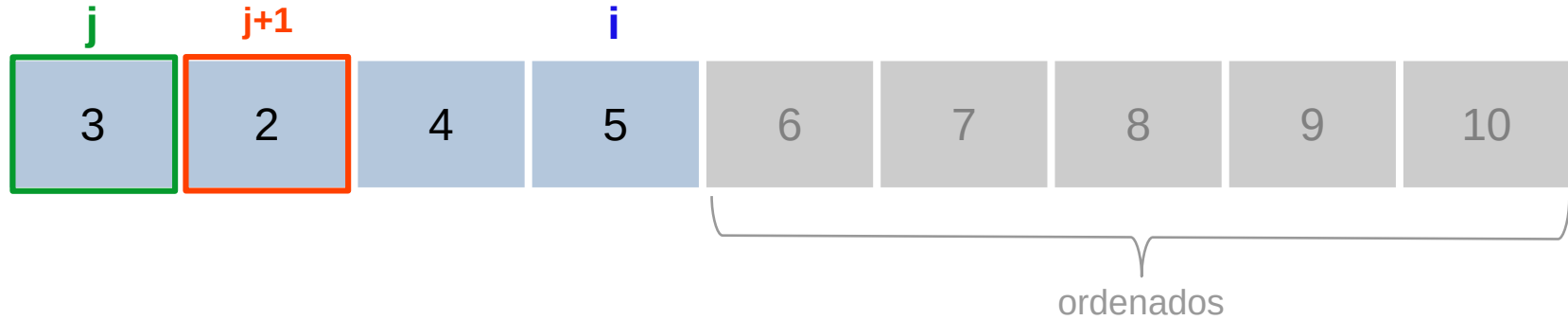
Iteração 5



# Bubble Sort

## Exemplo (35)

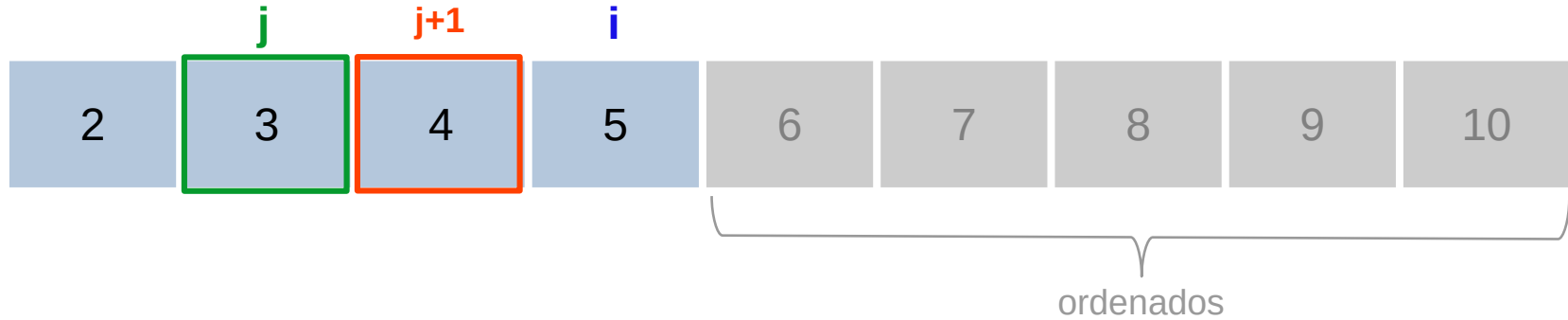
Iteração 6



# Bubble Sort

## Exemplo (36)

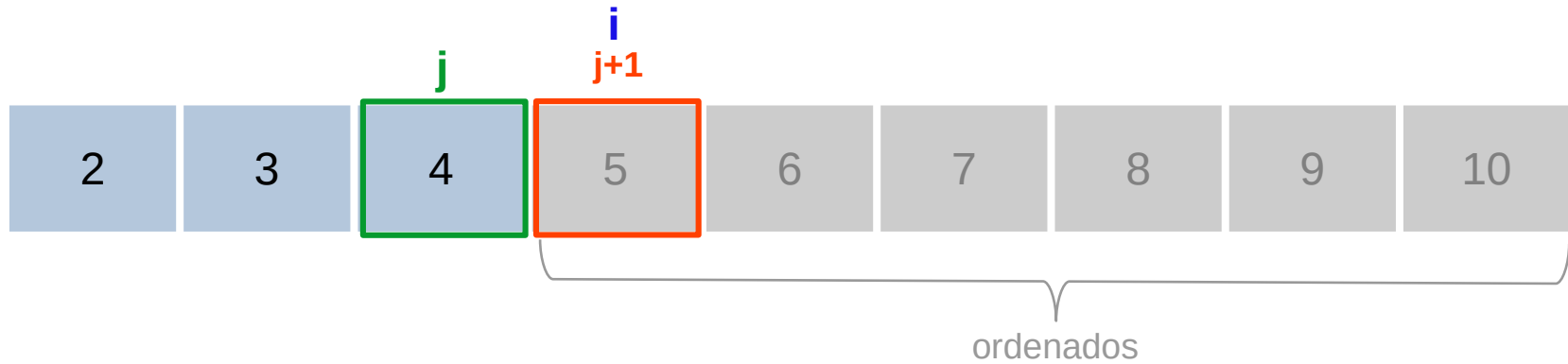
Iteração 6



# Bubble Sort

## Exemplo (37)

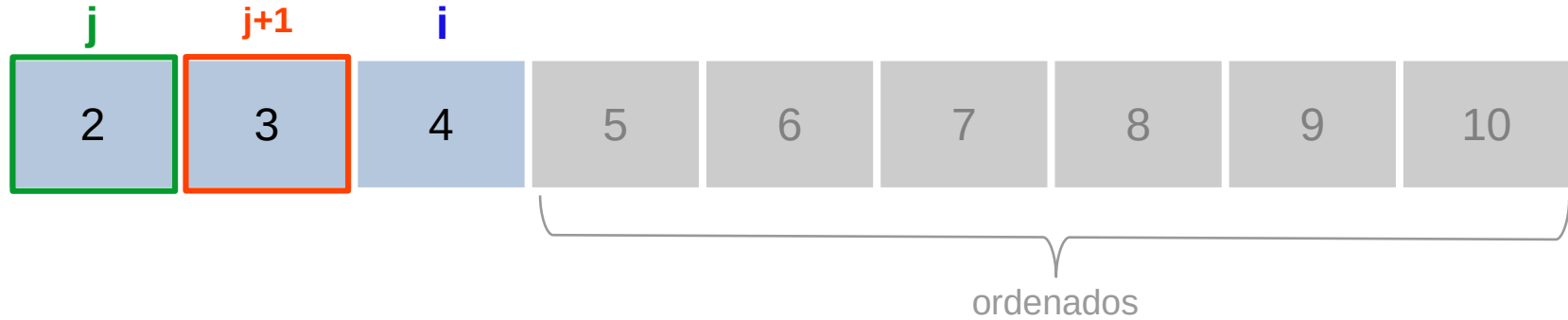
Iteração 6



# Bubble Sort

## Exemplo (38)

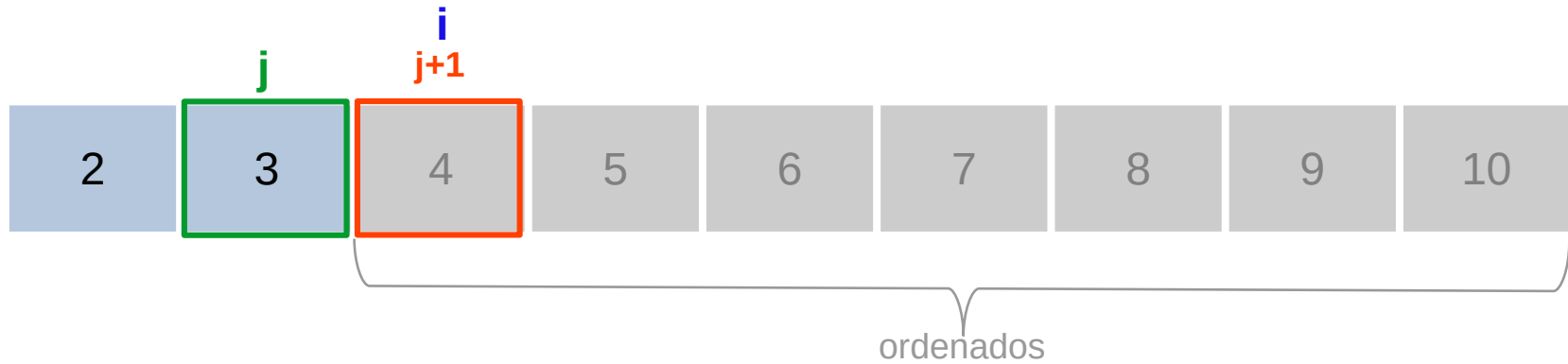
Iteração 7



# Bubble Sort

## Exemplo (39)

Iteração 7



# Bubble Sort

## Exemplo (40)

Iteração 8





# Bubble Sort

## Exemplo (41)



FIM

# Bubble Sort

## Pseudocódigo

Algoritmo Bubble

início

```
para i de n-1 até 1 faça
```

```
    para j de 0 até i-1 faça
```

```
        se  $A[j] > A[j+1]$  então
```

```
            troca( $A[j]$ ,  $A[j+1]$ )
```

```
        fimSe
```

```
    fimPara
```

```
fimPara
```

```
fimAlgoritmo
```

Iteração interna: compara  
elementos aos pares.  
j varia do início até o último  
elemento da parte não  
ordenada

Iteração do método.  
i controla o fim da parte não  
ordenada  
(varia da última posição até  
a segunda)

# Bubble Sort

## Outro Exemplo

- Observe este exemplo de ordenação usando o algoritmo anterior:

	0	1	2	3	4	5
Lista:	9	4	5	10	5	8
1ª it.	4	5	9	5	8	10
2ª it.	4	5	5	8	9	10
3ª it.	4	5	5	8	9	10
4ª it.	4	5	5	8	9	10
5ª it.	4	5	5	8	9	10

Neste ponto os dados já estão ordenados e o algoritmo não precisaria continuar.

Vamos melhorar este algoritmo para detectar se houve ou alguma troca durante a iteração.

Observe que o algoritmo é estável: 5 vinha antes de 5 e assim se manteve

# Bubble Sort – versão com Flag

## Pseudocódigo

Algoritmo Bubble com Flag

Início

para i de n-1 até 1 faça

flag = falso

para j de 0 até i-1 faça

se  $A[j] > A[j+1]$  então

troca( $A[j]$ ,  $A[j+1]$ )

flag = verdadeiro

fimSe

fimPara

se flag = falso então

interrompa

fimSe

fimPara

fimAlgoritmo

Flag = falso significa que a iteração anterior não resultou em nenhuma troca, ou seja, os elementos já estão em ordem.

# Bubble Sort

## Análise

- Algoritmo de ordem quadrática →  $O(n^2)$ 
  - não é recomendado para programas que precisem de agilidade e operem com quantidade elevada de dados.
- Número de comparações:  $(n^2 - n) / 2$
- Versão sem flag é sempre quadrática
- Versão com flag é sensível ao tipo de lista:
  - Melhor caso: se a lista já estiver ordenada, será feita apenas 1 iteração, então o tempo será  $O(n)$
  - Pior caso: menor elemento está na última posição, então o tempo será  $O(n^2)$
  - Caso médio:  $O(n^2)$

# Bubble Sort

## Análise

- Complexidade de espaço constante, pois utiliza apenas 3 variáveis adicionais (i, j e um auxiliar para a troca), sendo portanto um algoritmo *in place*.
- Algoritmo **estável**, pois a troca não ocorre quando os valores das chaves são iguais, mantendo assim sua ordem relativa.
- Os elementos se movem 1 posição por vez (não há “saltos”).
- Elementos que já estão em sua posição final são movidos para depois retornarem
  - Observe o que acontece com o elemento 9 nos slides 15 e 16.