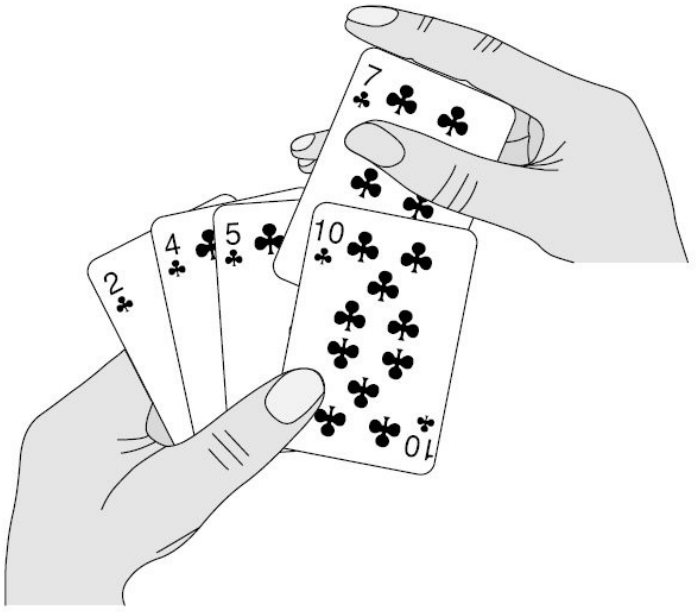
Pesquisa e Ordenação de Dados

Unidade 2.3:

Insertion Sort



Insertion Sort

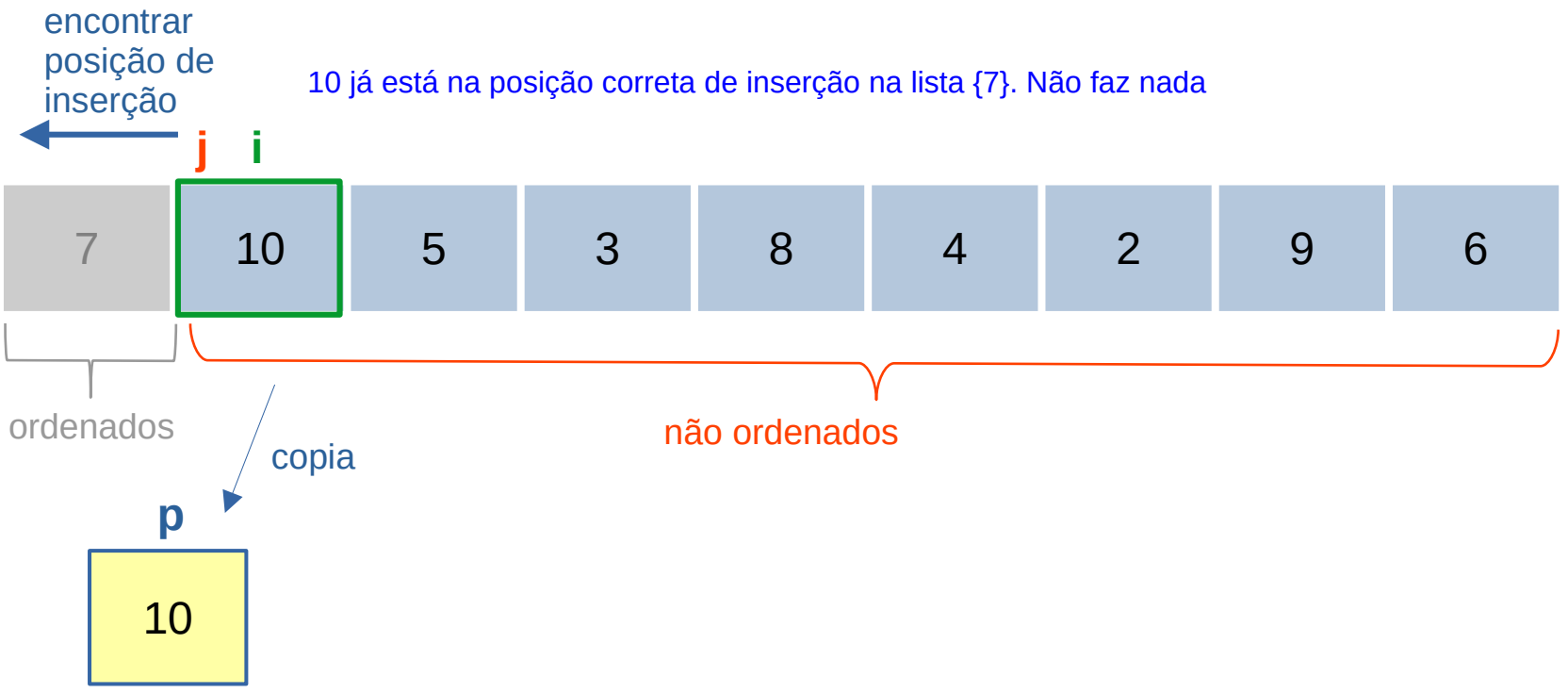
- Também chamado de **inserção direta**
- Analogia: Jogo de baralho
  - O jogador mantém um conjunto de cartas ordenadas na mão e, ao “pescar” uma nova carta, deve abrir espaço entre as demais para inseri-la na posição correta, de modo a manter a ordenação do conjunto.
- Percorre o vetor da esquerda para a direita e, conforme avança, insere cada elemento em sua posição correta na parte ordenada

Insertion Sort

- Divide a lista:
  - Parte ordenada, à esquerda (inclui inicialmente o primeiro elemento)
  - Parte não ordenada, à direita
- Executa **n-1** iterações
- Cada iteração:
  - Pega o primeiro elemento da parte não ordenada (vamos chamá-lo de **p**) e verifica qual seria sua posição correta de inserção na parte ordenada
    - Para encontrar a posição adequada de inserção, compara **p** com cada elemento à sua esquerda, deslocando este último 1 posição para a direita sempre que este for maior do que **p**
  - **p** então inserido à esquerda do último elemento movido.

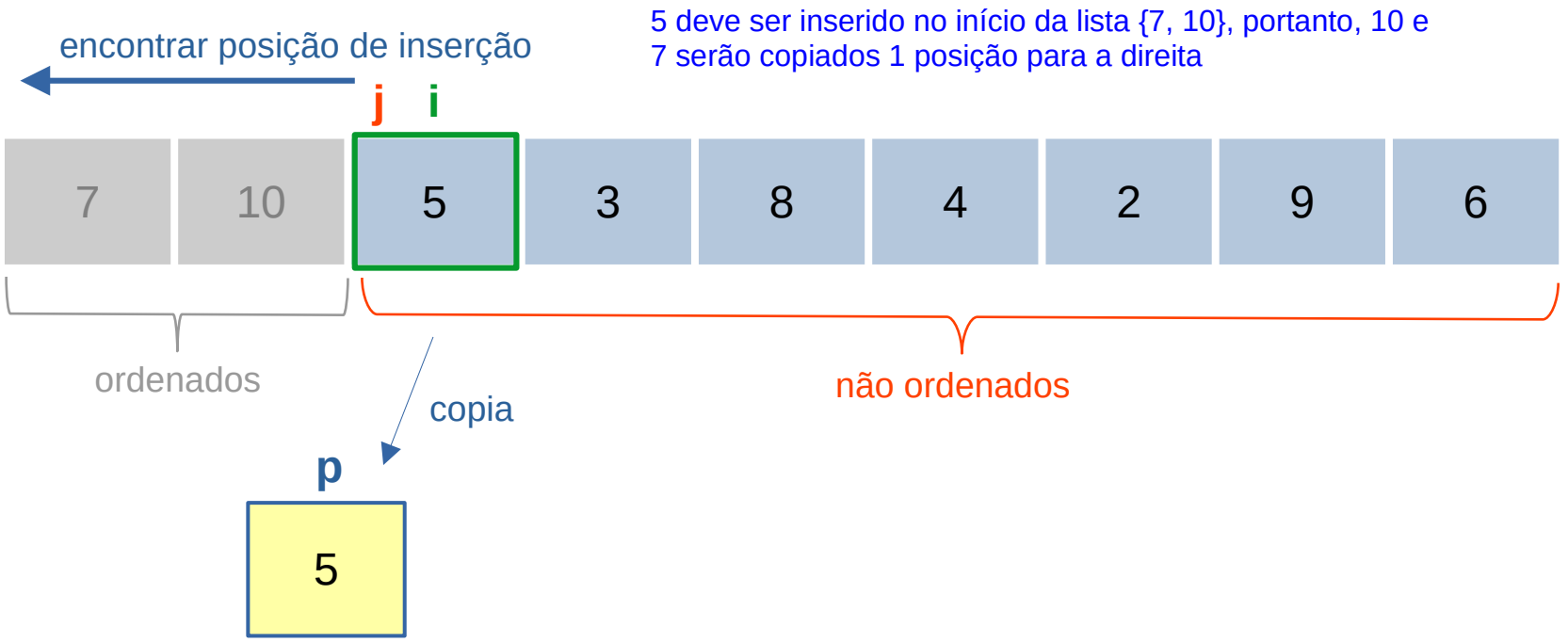
Insertion Sort
Exemplo (1)

Iteração 1



Insertion Sort
Exemplo (2)

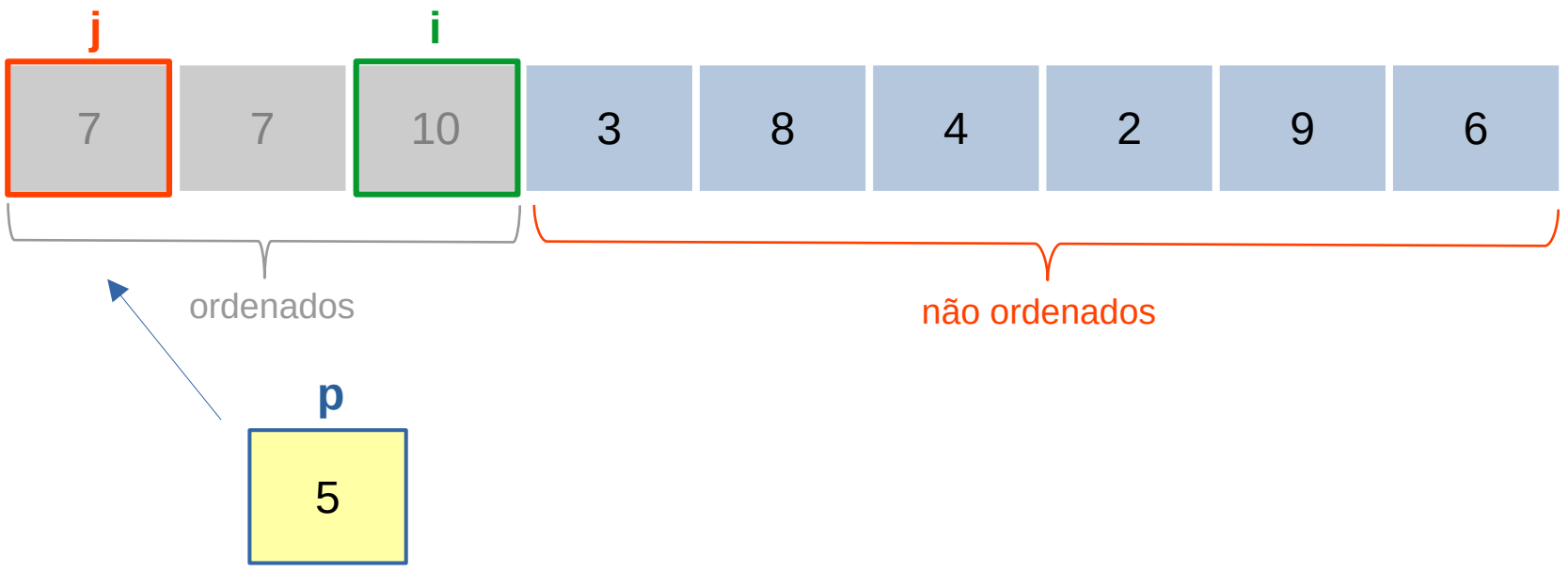
Iteração 2



Insertion Sort
Exemplo (3)

Iteração 2

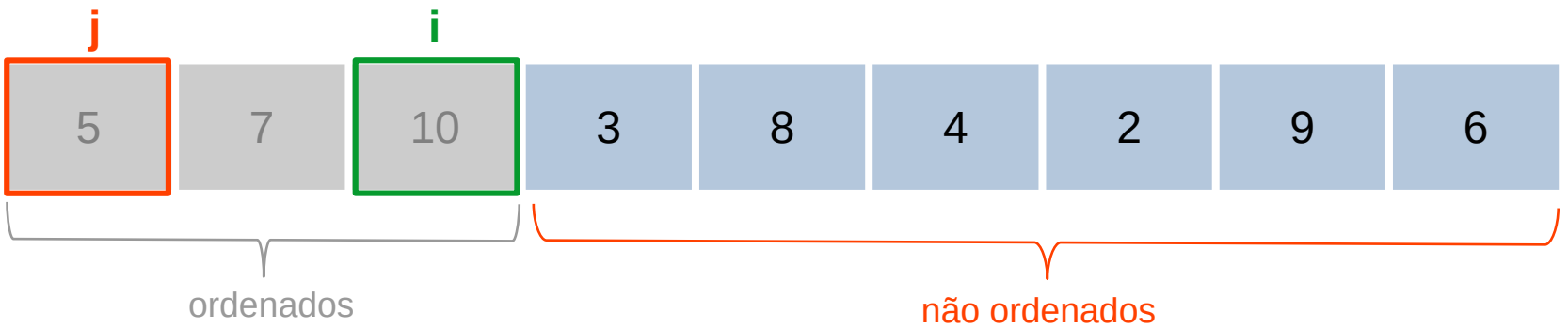
posição de inserção do 5



Insertion Sort

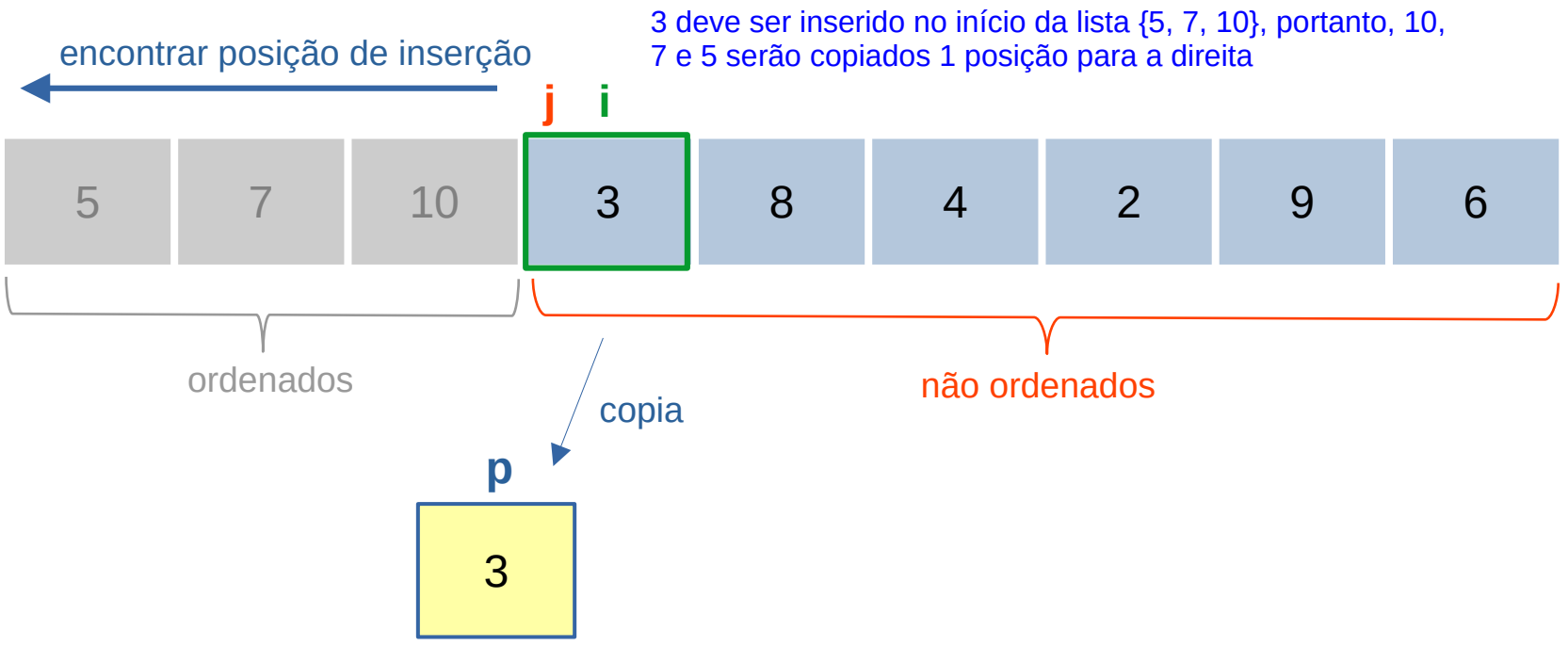
Exemplo (4)

Iteração 2



Insertion Sort
Exemplo (5)

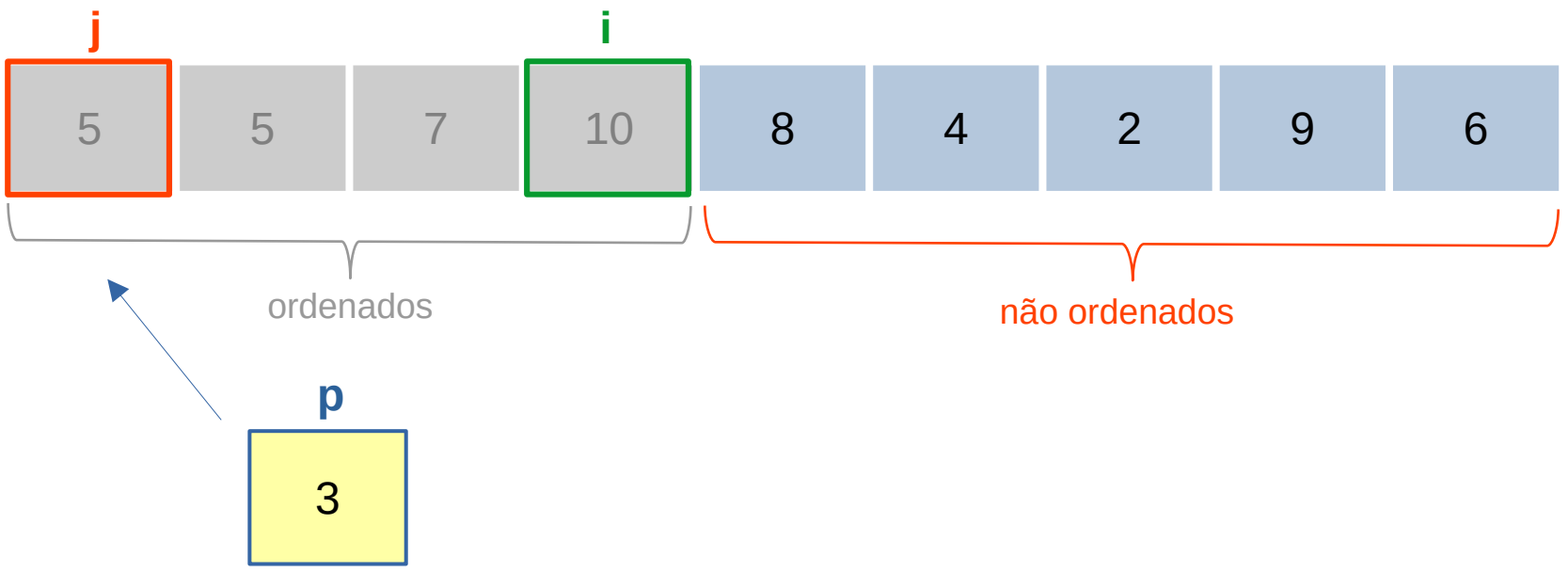
Iteração 3



Insertion Sort
Exemplo (6)

Iteração 3

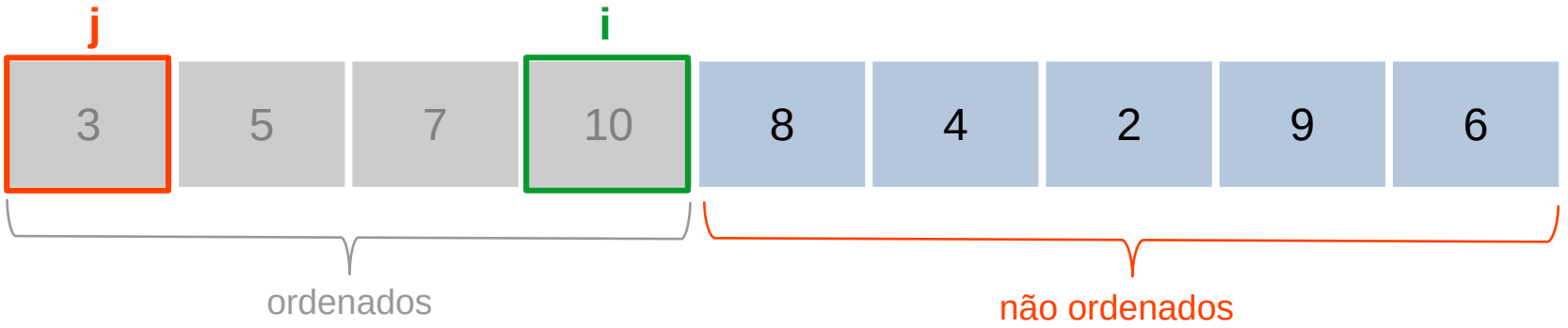
posição de inserção do 3



Insertion Sort

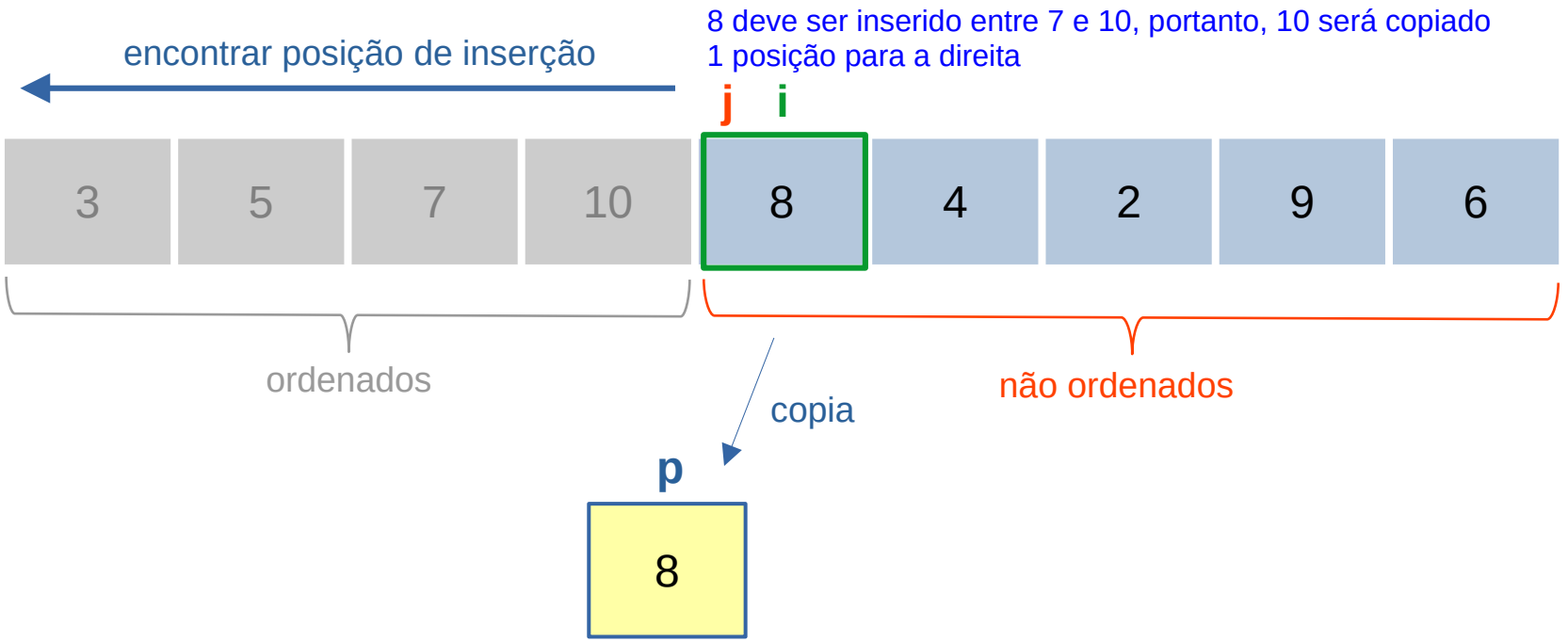
Exemplo (7)

Iteração 3



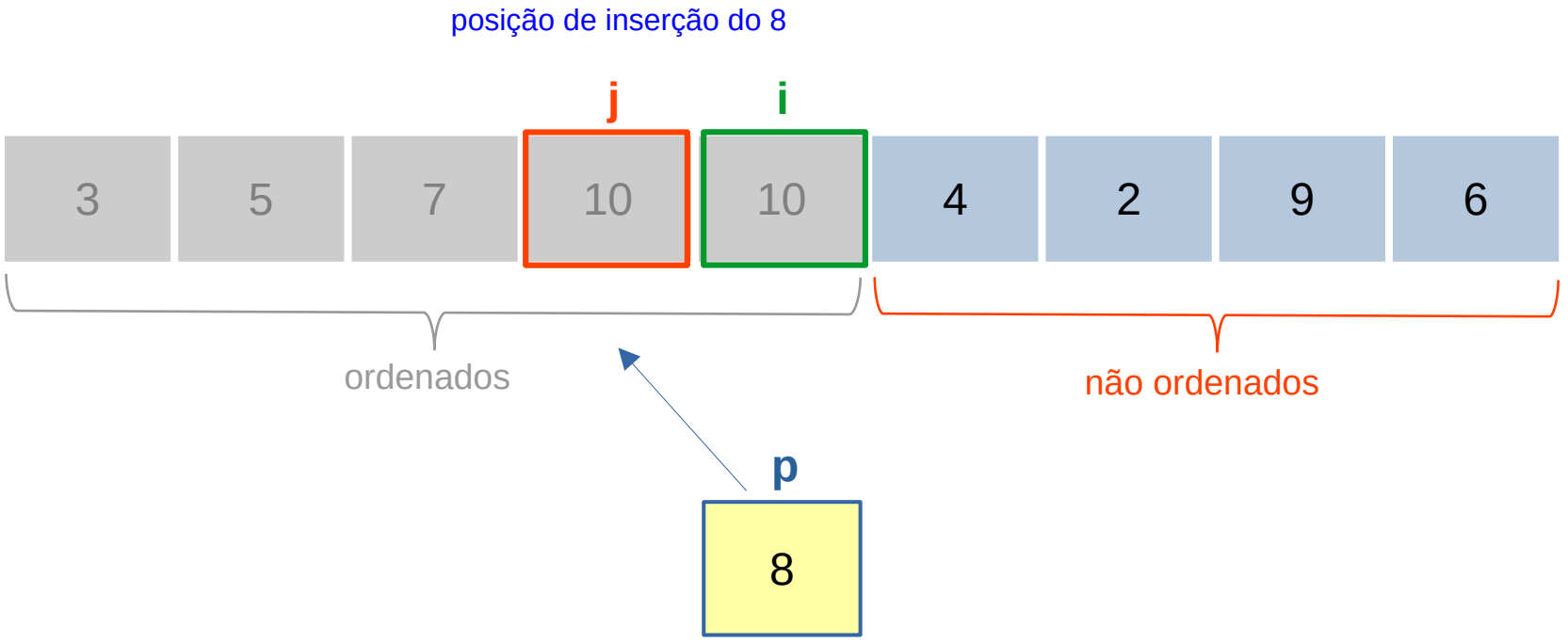
Insertion Sort
Exemplo (8)

Iteração 4



Insertion Sort
Exemplo (9)

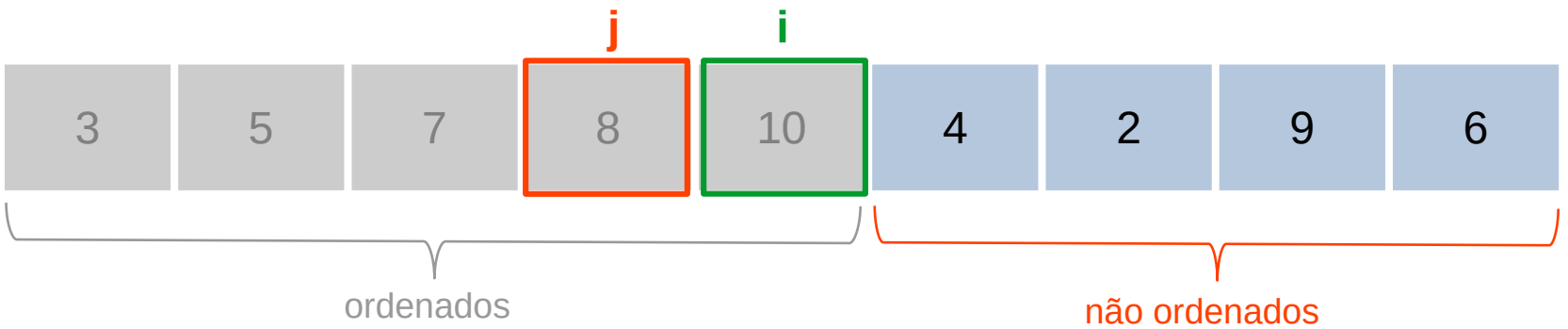
Iteração 4



Insertion Sort

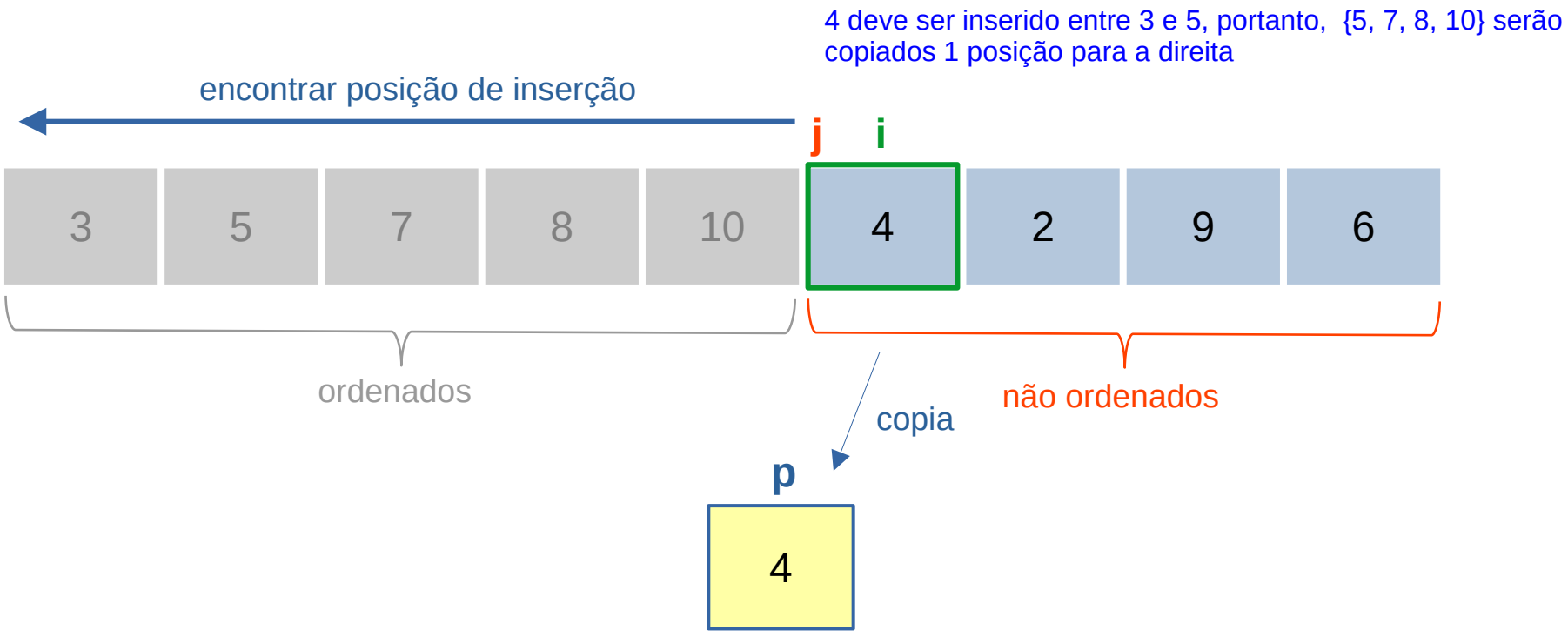
Exemplo (10)

Iteração 4



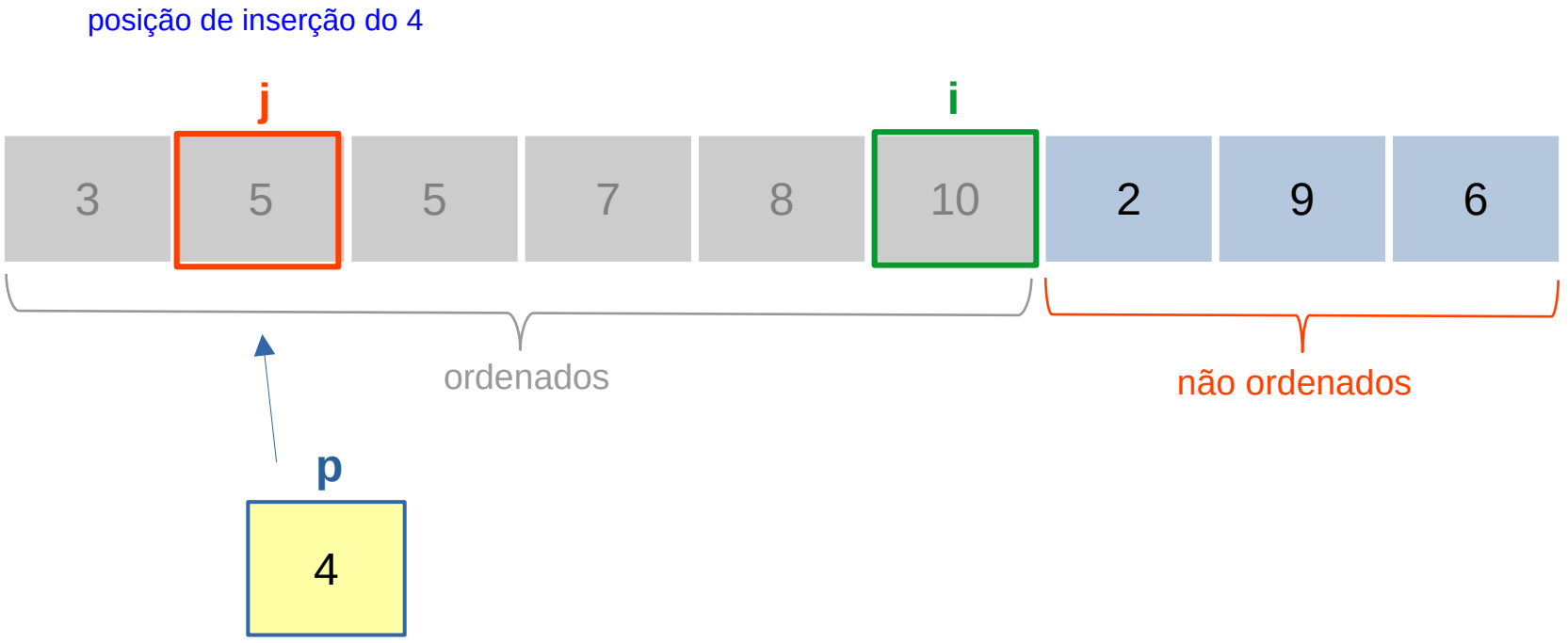
Insertion Sort
Exemplo (11)

Iteração 5



Insertion Sort
Exemplo (12)

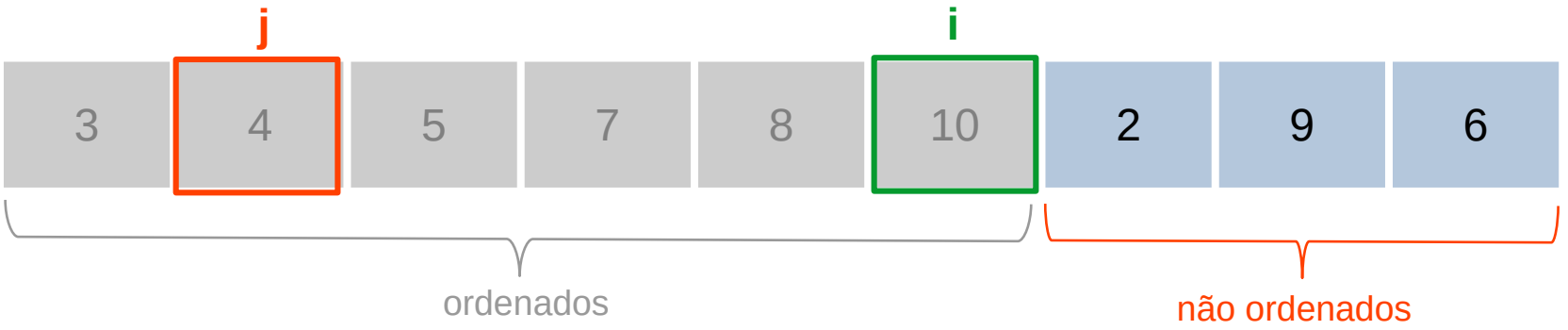
Iteração 5



Insertion Sort

Exemplo (13)

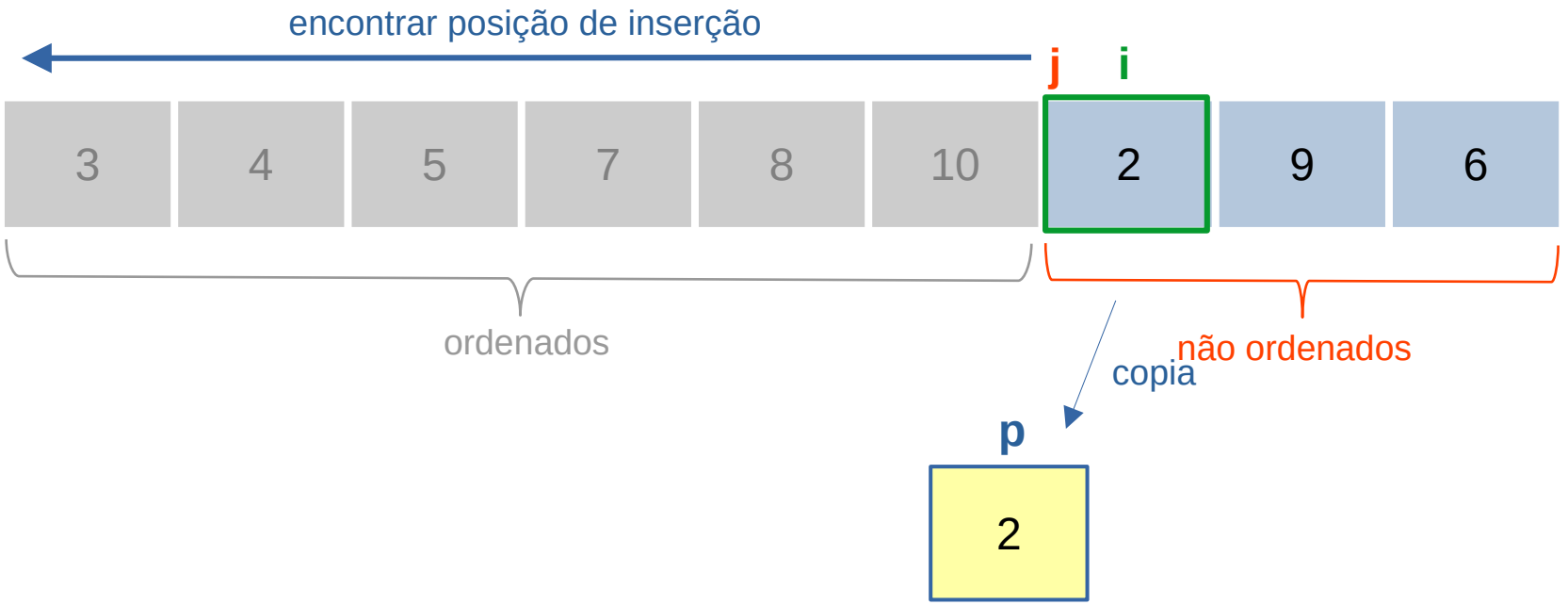
Iteração 5



Insertion Sort
Exemplo (14)

Iteração 6

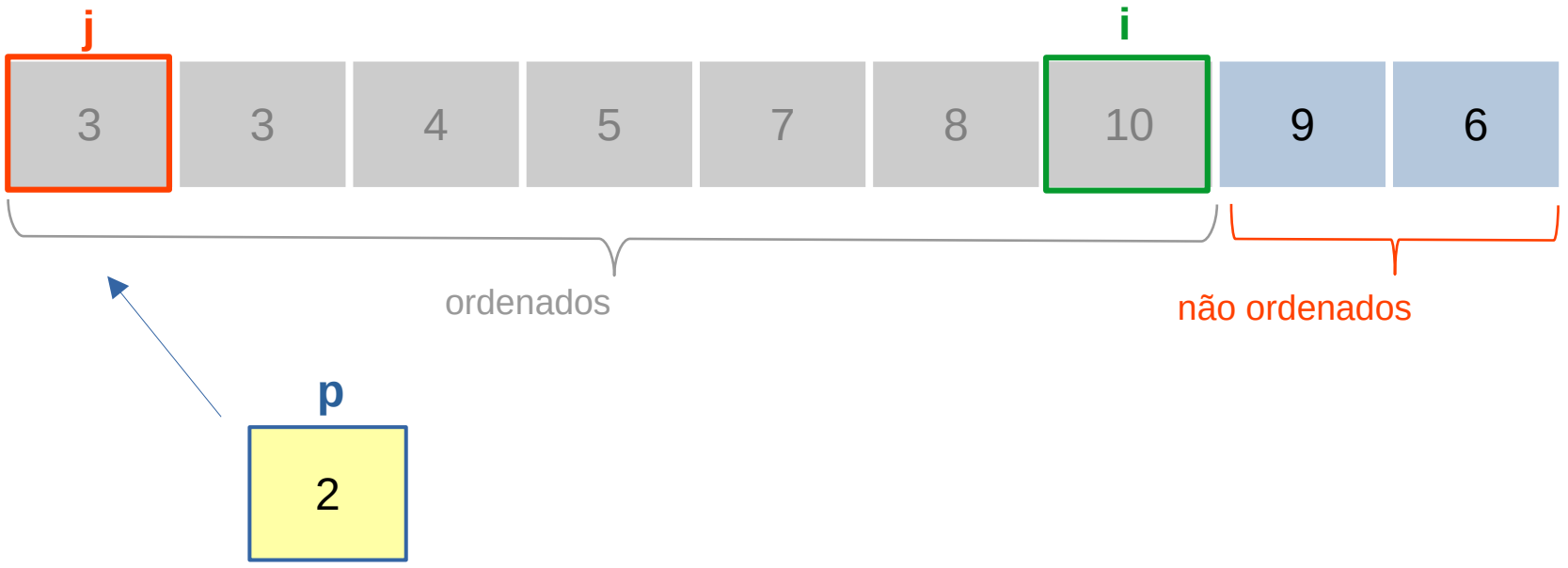
2 deve ser inserido no início da lista, portanto, todos os elementos serão copiados 1 posição para a direita



Insertion Sort
Exemplo (15)

Iteração 6

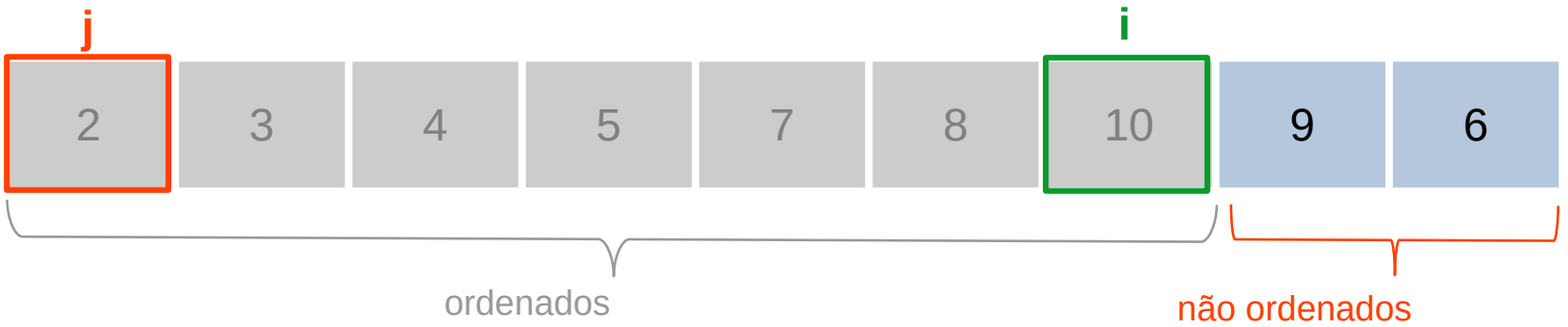
posição de inserção do 2



Insertion Sort

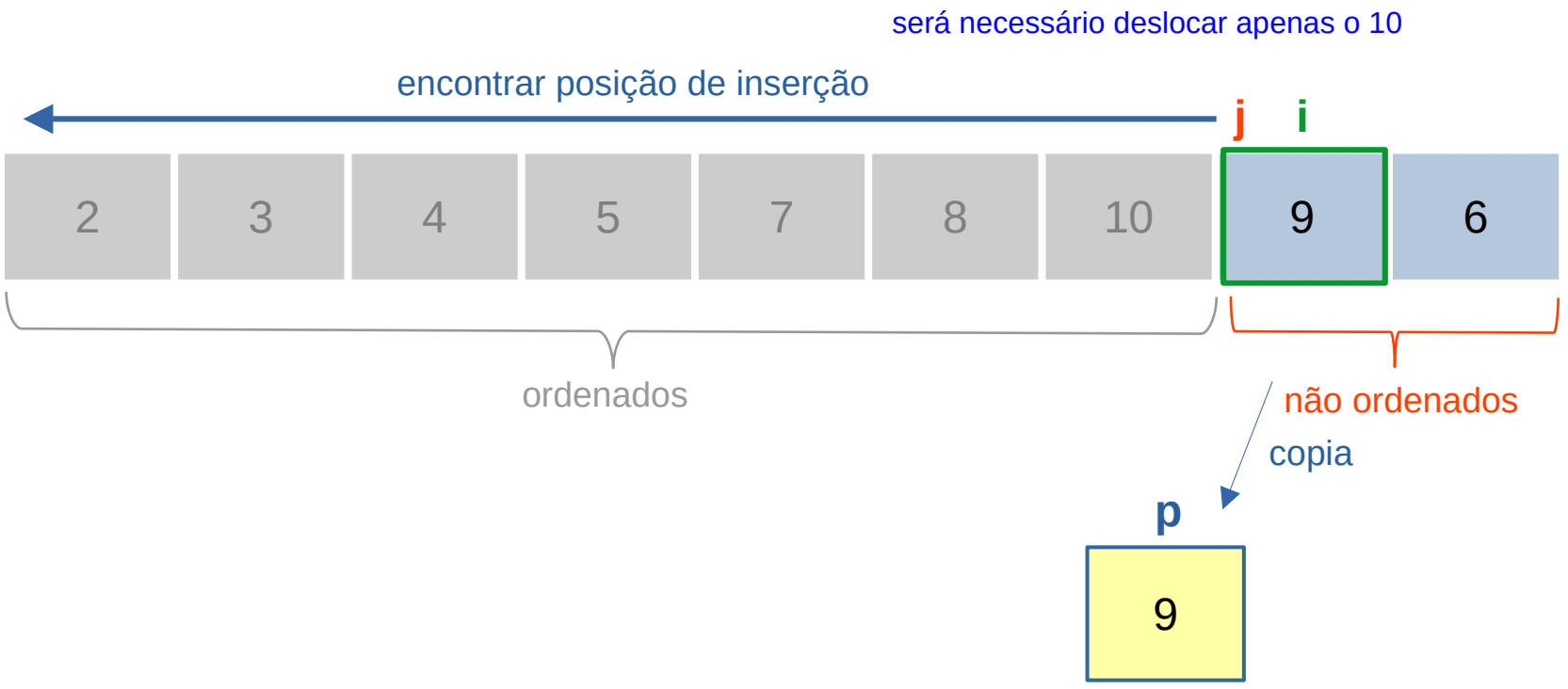
Exemplo (16)

Iteração 6



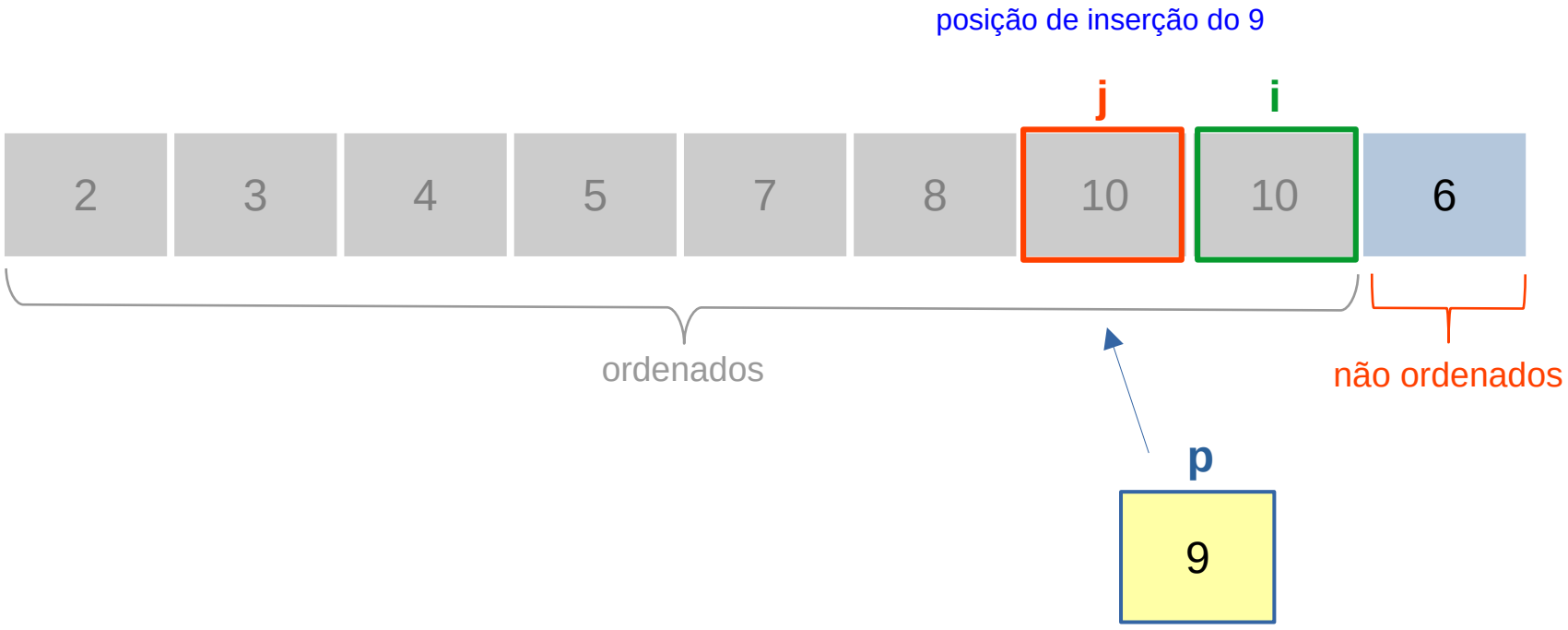
Insertion Sort
Exemplo (17)

Iteração 7



Insertion Sort
Exemplo (18)

Iteração 7



Insertion Sort

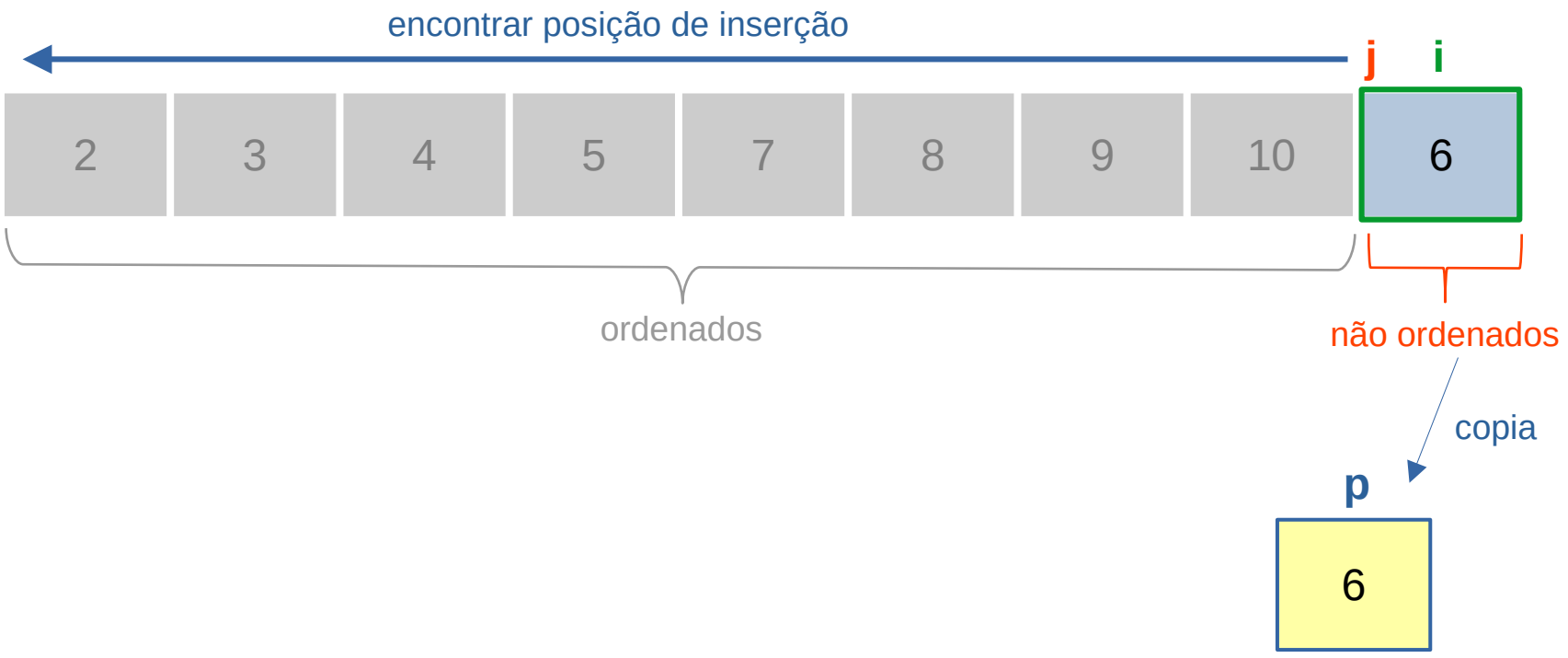
Exemplo (19)

Iteração 7

Insertion Sort
Exemplo (20)

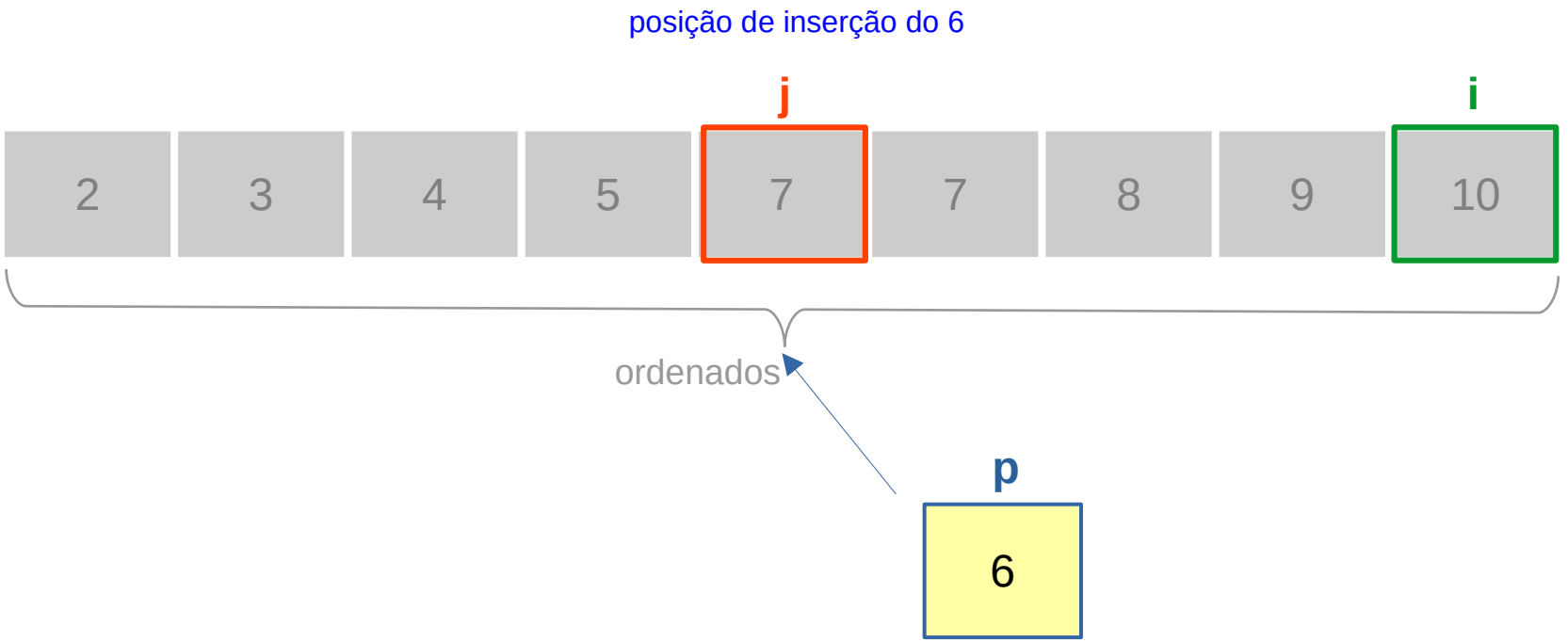
Iteração 8

será necessário deslocar {7, 8, 9, 10} 1 posição para a direita



Insertion Sort
Exemplo (21)

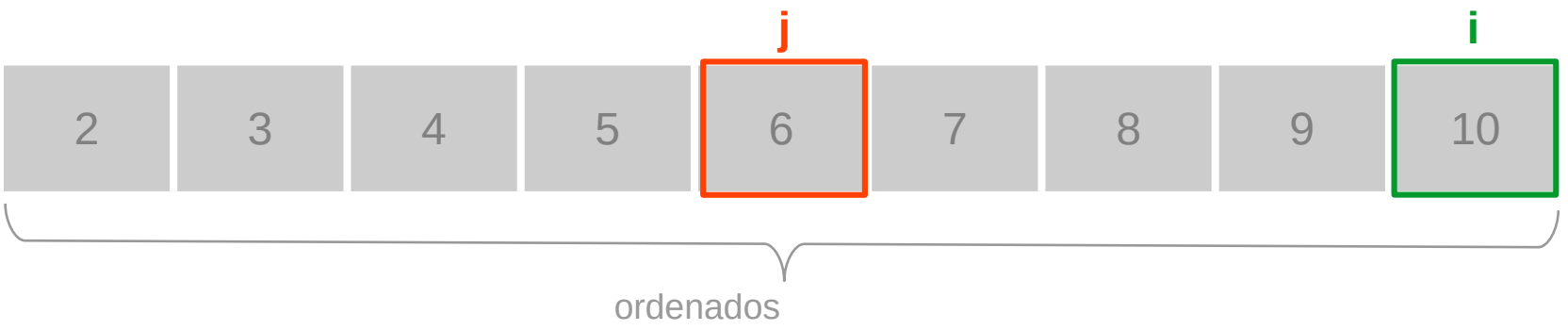
Iteração 8



Insertion Sort

Exemplo (22)

Iteração 8



Insertion Sort
Outro Exemplo

Legenda:
Rosa: elemento posicionado
Verde: **p** para a próxima iteração
Fundo azul: parte ordenada

- Lista original:

13	7	5	1	$\overline{7}$	4

1ª it.	**7**	13	**5**	1	$\overline{7}$	4
2ª it.	**5**	7	13	**1**	$\overline{7}$	4
3ª it.	**1**	5	7	13	**$\overline{7}$**	4
4ª it.	1	5	7	**$\overline{7}$**	13	**4**
5ª it.	1	**4**	5	7	$\overline{7}$	13

Observe que o método é estável: 7 vinha antes de $\overline{7}$ e assim se manteve

Insertion Sort
Pseudocódigo

```
Algoritmo Insertion
Início
    para i de 1 até n-1 faça
        p = A[i]
        para j de i até 1 e p < A[j-1] faça
            A[j] = A[j-1]
        fimPara
        A[j] = p
    fimPara
fimAlgoritmo
```

Primeiro da parte não ordenada

Iteração do método. Começa no segundo elemento e vai até o último

Iteração interna. Busca posição de inserção na parte ordenada.

Move o valor para a posição à sua direita (abre espaço)

p é colocado na posição encontrada

Insertion Sort
Análise

- Pior caso: lista inversamente ordenada → **$O(n^2)$**
  - A cada iteração, requer que todos os elementos sejam deslocados.
  - **$(n^2 - n) / 2$** comparações e trocas
  - Neste caso, é muito mais lento que o *Selection Sort*, pois executa o mesmo número de comparações, mas com mais trocas
- Melhor caso: lista já ordenada → **$O(n)$**
  - N-1 comparações e 0 trocas (não desloca nenhum elemento).
  - Neste caso, é muito mais rápido que o *Selection Sort*
- Caso médio: **$O(n^2)$**
  - requer na média que metade dos elementos seja movimentada
- Eficiente para listas quase ordenadas, quando for necessário adicionar poucos elementos a uma lista já ordenada
- *In place*
- Estável