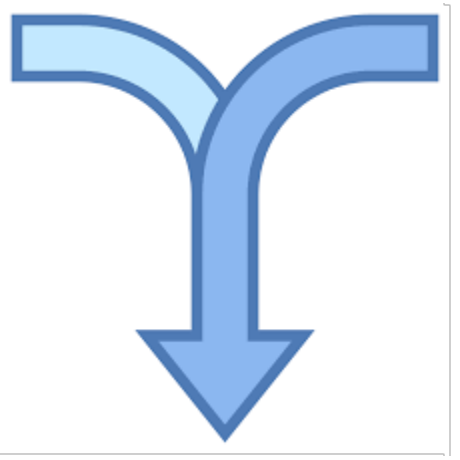
Pesquisa e Ordenação de Dados

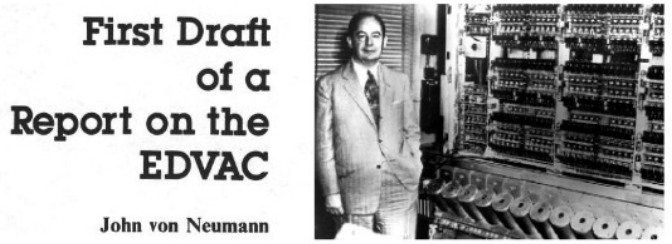
Unidade 2.4:

Merge Sort



Merge Sort

- Ordenação por **intercalação**
- Algoritmo do tipo **divisão e conquista**
  - *Divide and conquer*: técnica que consiste em decompor o problema a ser resolvido em instâncias cada vez menores do mesmo tipo de problema, resolver estas instâncias (em geral, recursivamente) e, por fim, combinar as soluções parciais para obter a solução do problema original.
- Método proposto por John Von Neumann em 1945



Merge Sort

- Ideia geral:
  - Dividir o vetor em duas partes;
  - Ordenar recursivamente cada parte;
  - Intercalar os dois segmentos ordenados, formando um novo segmento ordenado.

Cada segmento é ordenado usando o próprio merge sort

Fonte: Sedgewick, 2016.

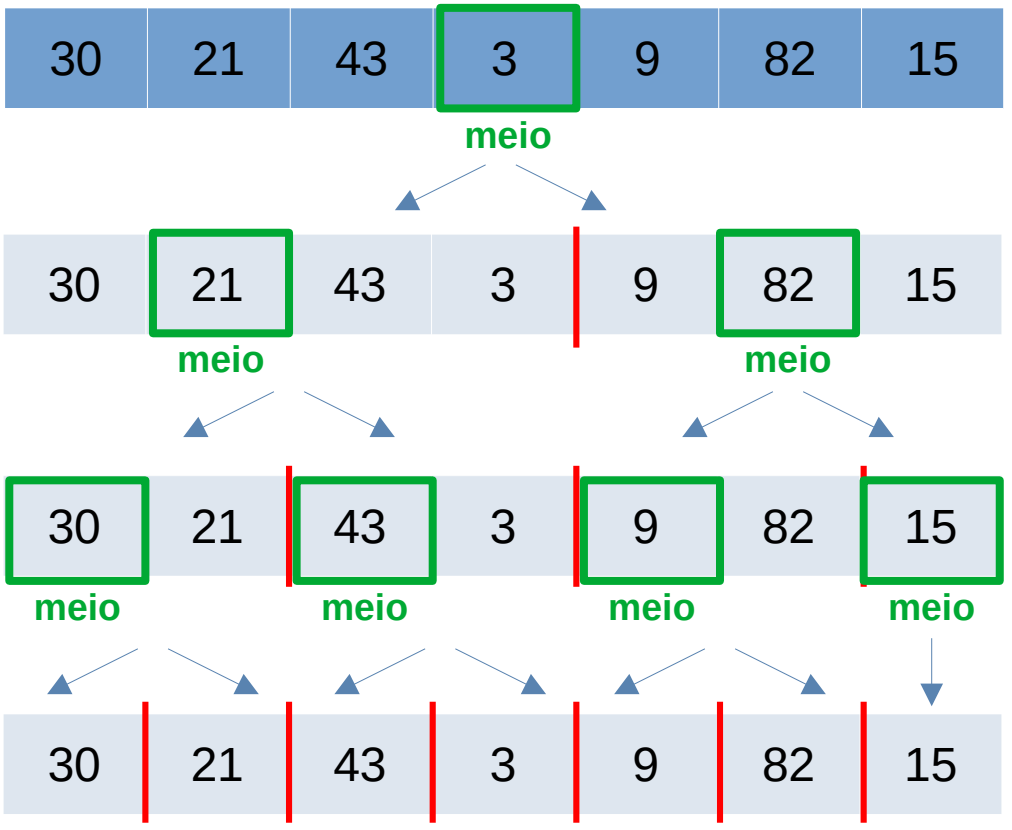
Merge Sort

- O funcionamento do algoritmo obedece aos seguintes passos:
  - Dividir:
    - Consiste em partir uma sequência em duas novas sequências;
      - Este passo é realizado recursivamente, iniciando com a divisão do vetor de **n** elementos em duas partes (perfeitamente iguais, se **n** é par, ou uma delas com 1 elemento a mais, se **n** for ímpar);
      - Então, cada uma das metades é novamente dividida em duas partes, e assim sucessivamente, até que não seja mais possível a divisão (ou seja, sobrem **n** conjuntos com 1 elemento cada);
  - Intercalar:
    - dadas 2 sequências ordenadas, juntá-las (*merge*) em uma única sequência ordenada.
      - sequências unitárias são, por definição, ordenadas
    - Este passo é aplicado a todos os pares de segmentos, até obter um único conjunto final ordenado.

Merge Sort

Como funciona - Divisão

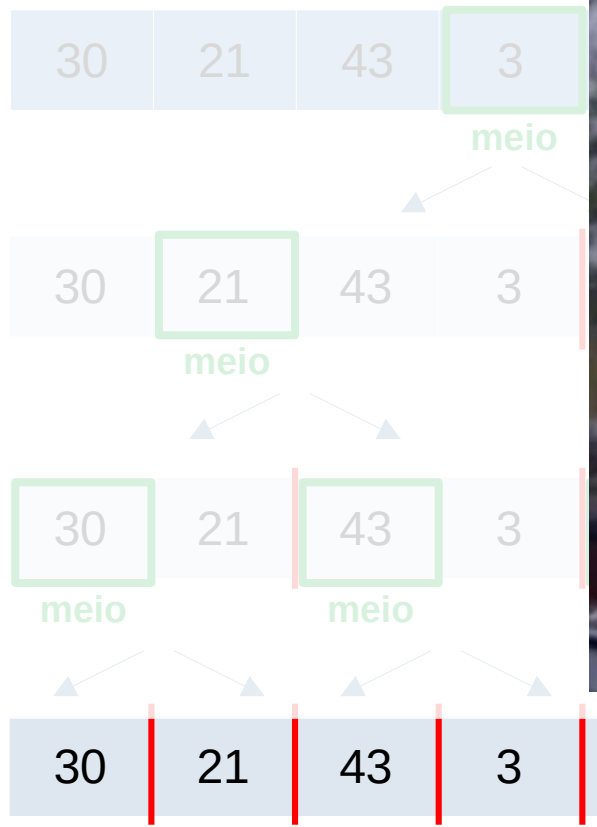
Divide recursivamente até chegar ao passo base: segmento unitário (ordenado por definição)

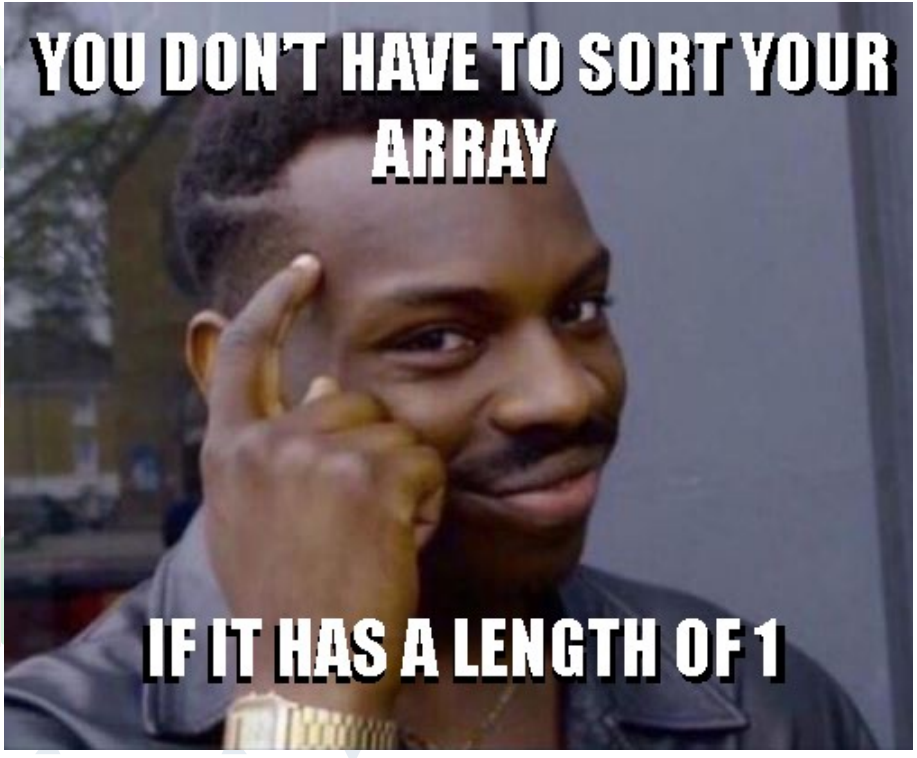


Merge Sort

Como funciona - Divisão

Divide recursivamente até chegar ao passo base: segmento unitário (ordenado por definição)

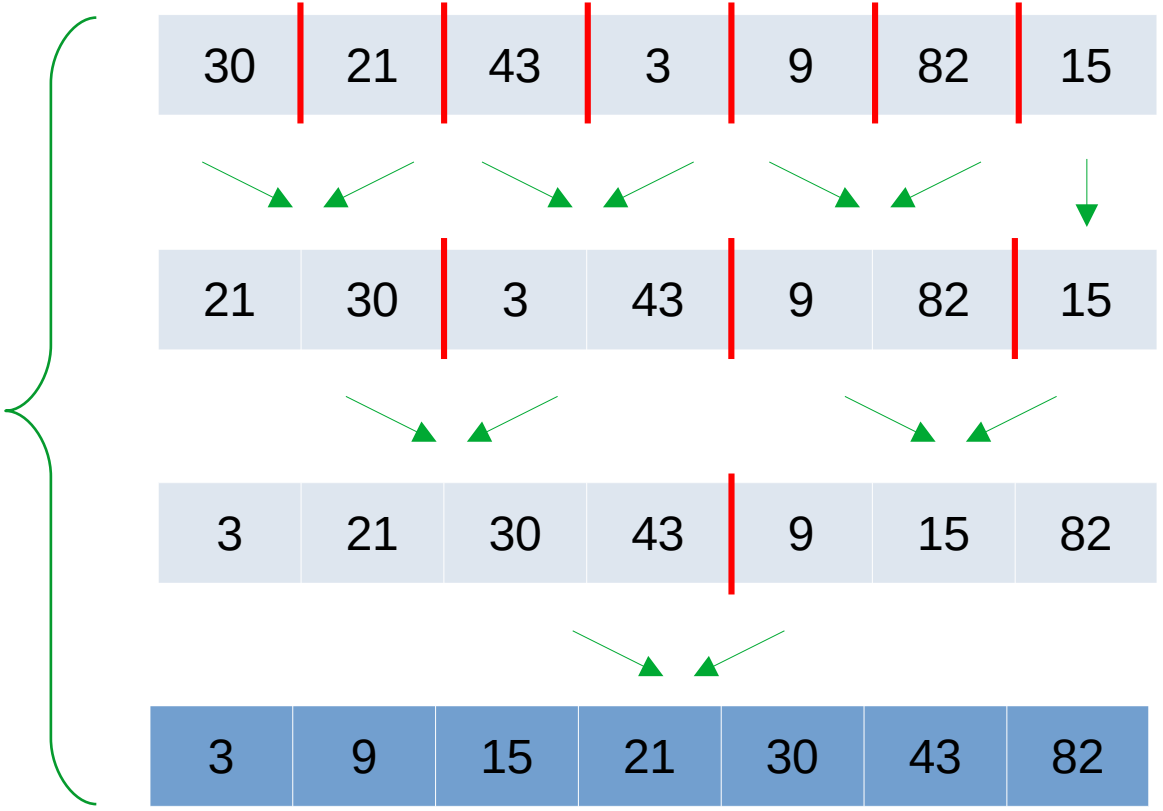




Merge Sort

Como funciona - Intercalação

Intercala 2 segmentos, formando um novo



Merge Sort
Implementação

- O algoritmo será implementado em duas partes:
  - **mergeSort**
    - Função principal que será chamada pelos demais programas
    - Responsável pela divisão do vetor e pela recursividade
    - Recebe como parâmetro o vetor, seu índice inicial e seu índice final
  - **intercala**
    - Função auxiliar responsável pela intercalação de dois segmentos ordenados, produzindo um novo segmento ordenado
    - Recebe como parâmetro o vetor e os limites de cada segmento (índices inicial, do meio e final)
    - Utiliza um vetor adicional para guardar os elementos já ordenados
    - Ao final, copia os dados do vetor auxiliar de volta para o respectivo segmento original

Merge Sort
Pseudocódigo

```
função mergeSort(A[], inicio, fim)
inicio
    se inicio < fim então
        meio ← (inicio + fim) / 2    /* divisão inteira */
        mergeSort(A, inicio, meio)
        mergeSort(A, meio + 1, fim)
        intercala(A, inicio, meio, fim)
    fimSe
fim
```

Segmento tem pelo menos 2 elementos

Chamadas recursivas

Merge Sort
Pseudocódigo

```
função intercala(A[], inicio, meio, fim)
inicio
    auxiliar ← alocação de espaço para (fim-inicio+1) elementos
    i ← inicio    /* i: posição atual no vetor da esquerda */
    j ← meio + 1  /* j: posição atual no vetor da direita */
    k = 0    /* k: posição atual no vetor auxiliar */
    enquanto i <= meio E j <= fim faça
        se A[i] <= A[j] entao
            auxiliar[k] ← A[i]
            i++
        senao
            auxiliar[k] ← A[j]
            j++
        fimSe
        k++
    fimEnquanto
```

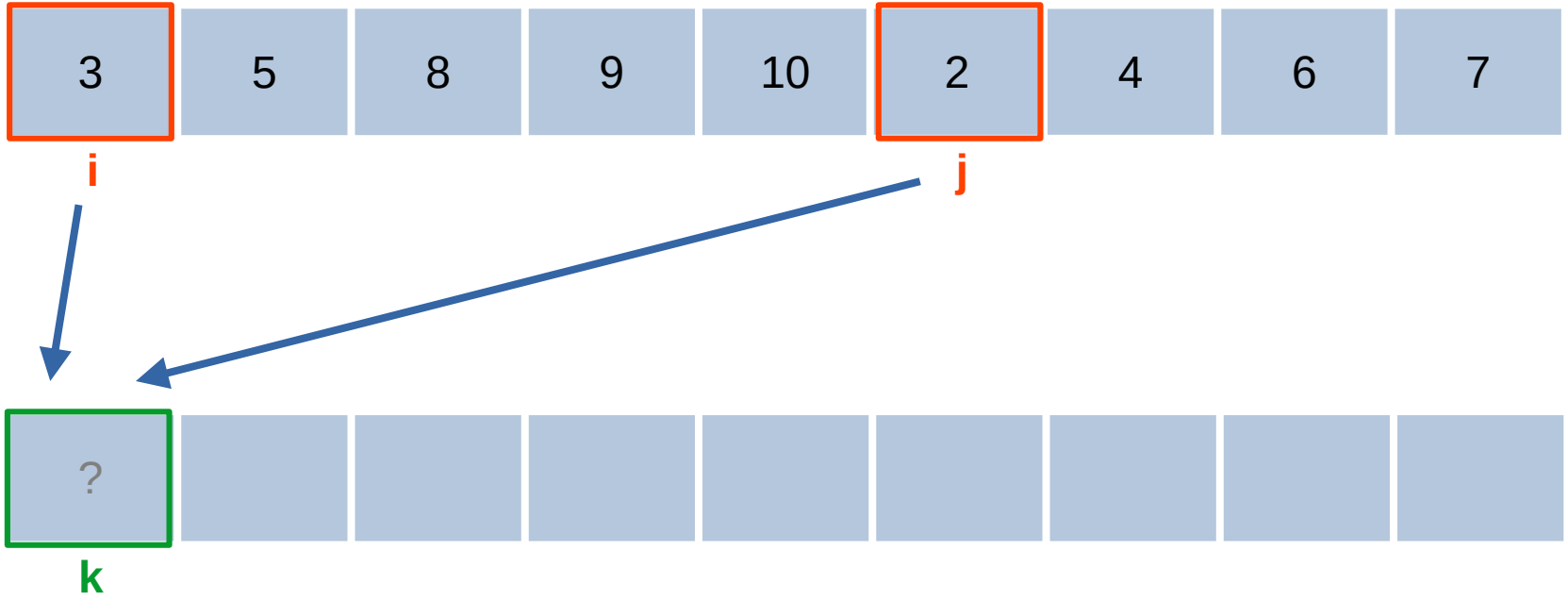
```
/* continuação */
enquanto i <= meio faca     /* ainda há elementos na primeira parte */
    auxiliar[k] ← A[i]
    k++
    i++
fimEnquanto

enquanto j <= fim faca      /* ainda há elementos na segunda parte */
    auxiliar[k] ← A[j]
    k++
    j++
fimEnquanto

para k de inicio ate fim faca    /* transferindo elementos de volta para o
vetor original */
    A[k] = auxiliar[k - inicio];
fimPara
Liberar espaço auxiliar
fim
```

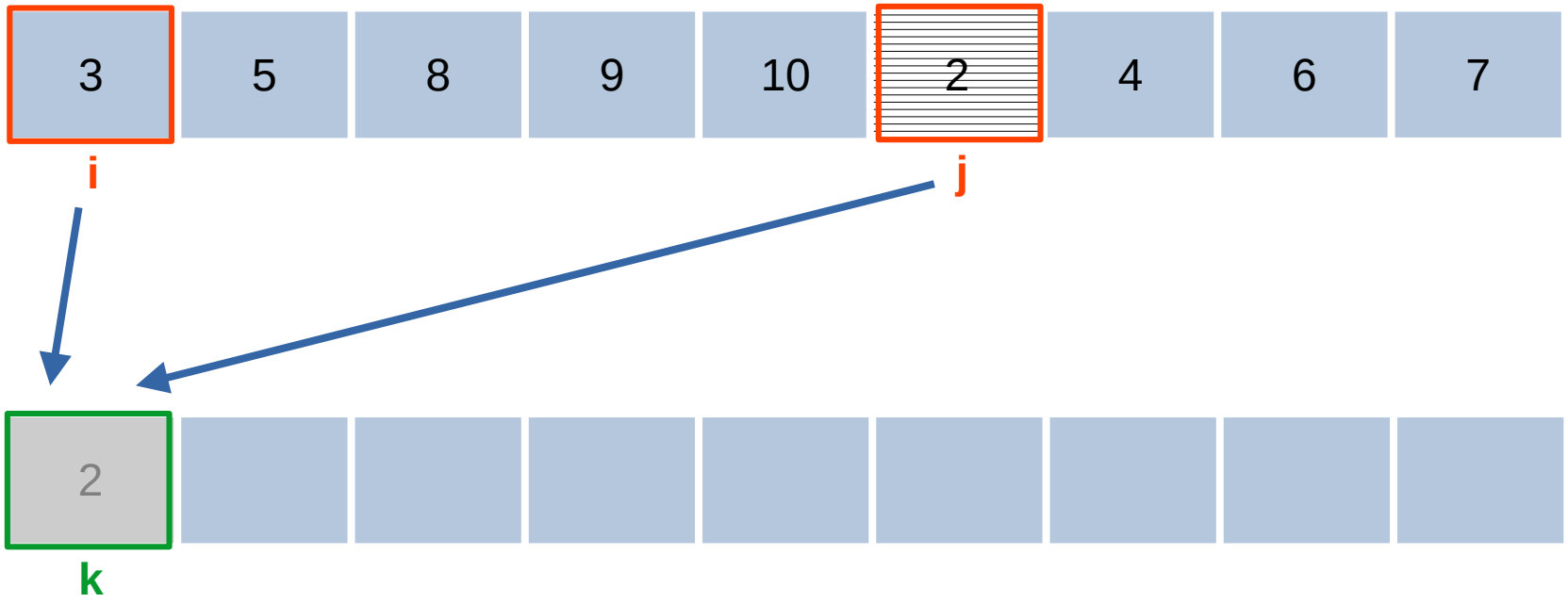
Merge Sort - Intercalação

Exemplo passo a passo (1)



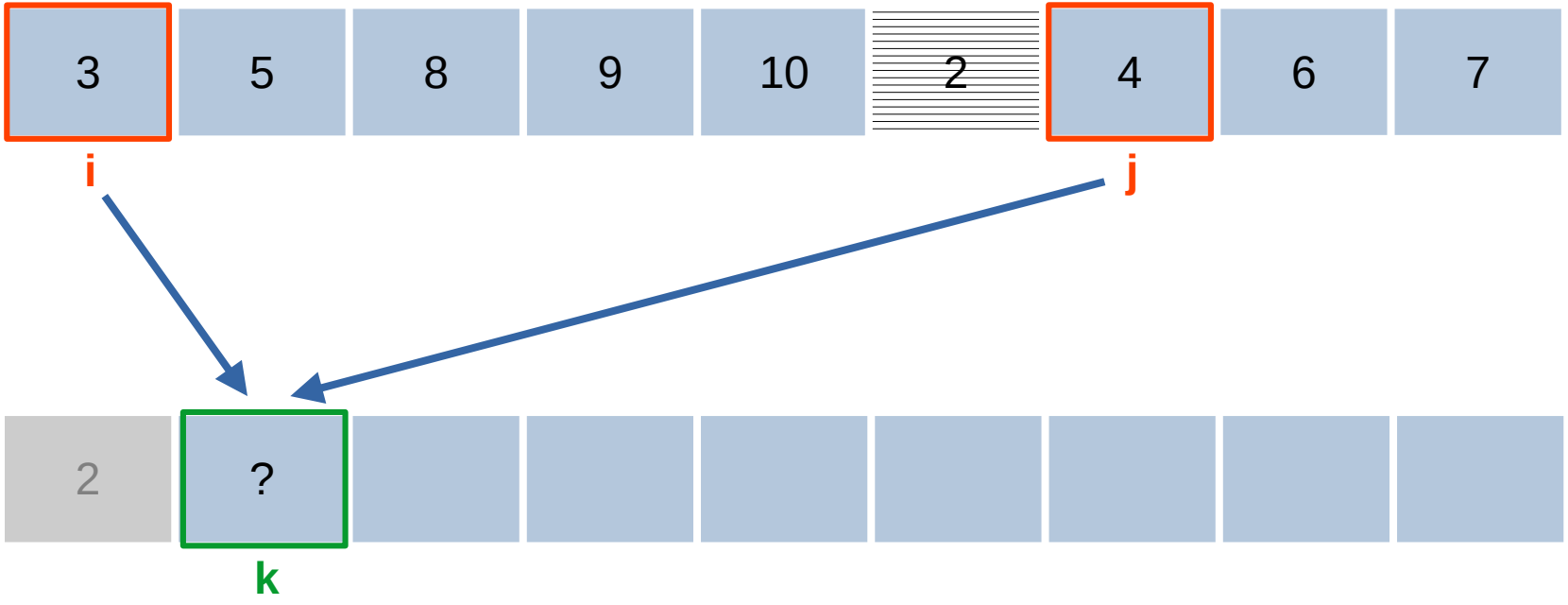
Merge Sort - Intercalação

Exemplo passo a passo (2)



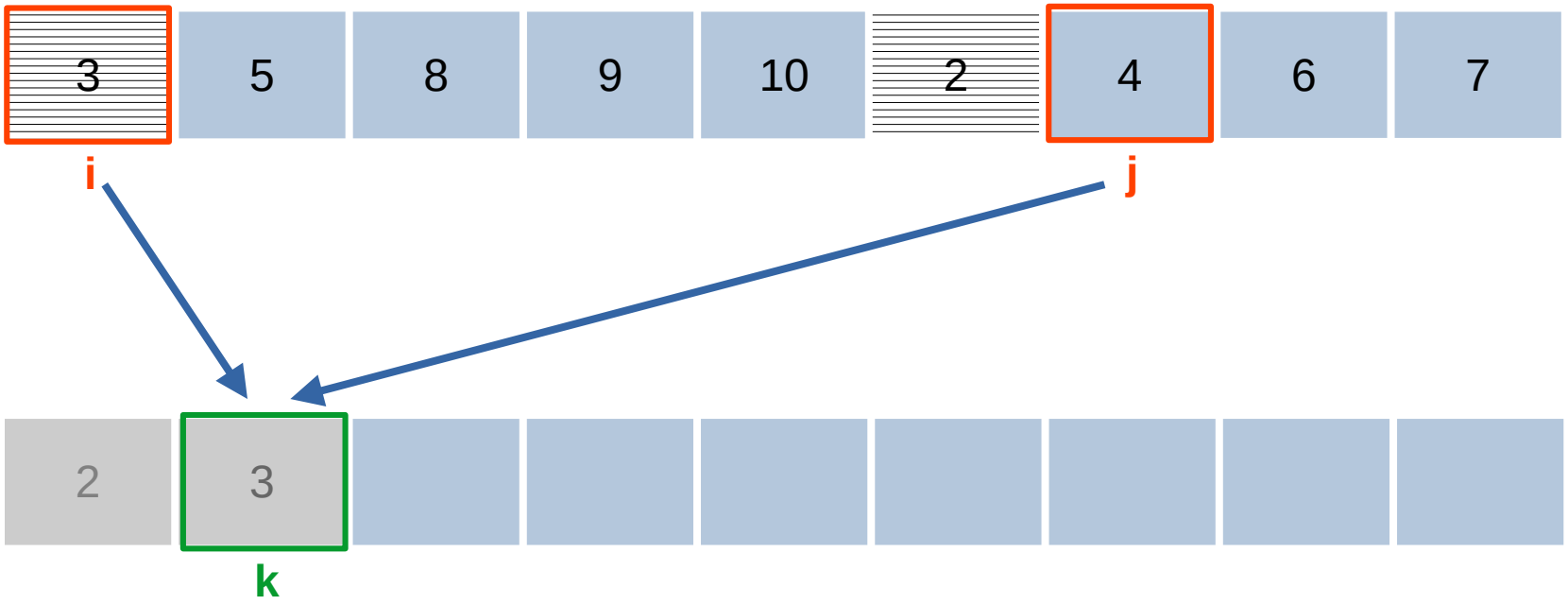
Merge Sort - Intercalação

Exemplo passo a passo (3)



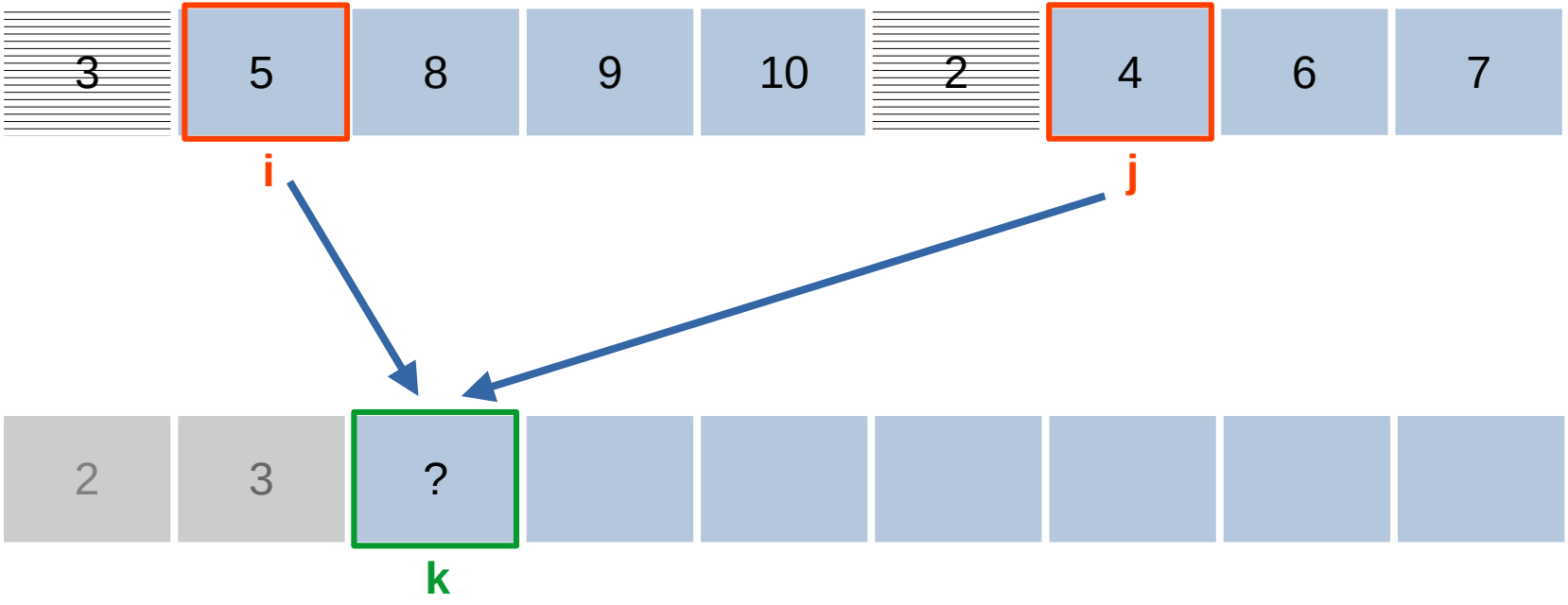
Merge Sort - Intercalação

Exemplo passo a passo (4)



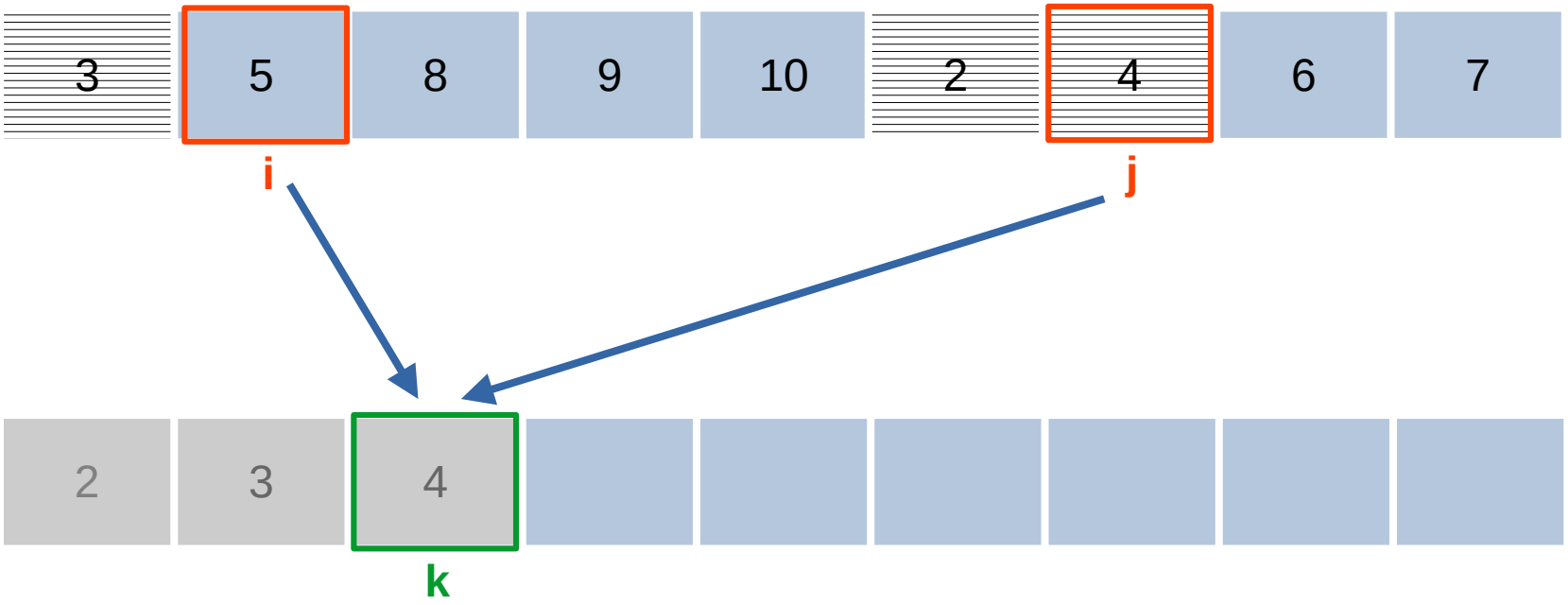
Merge Sort - Intercalação

Exemplo passo a passo (5)



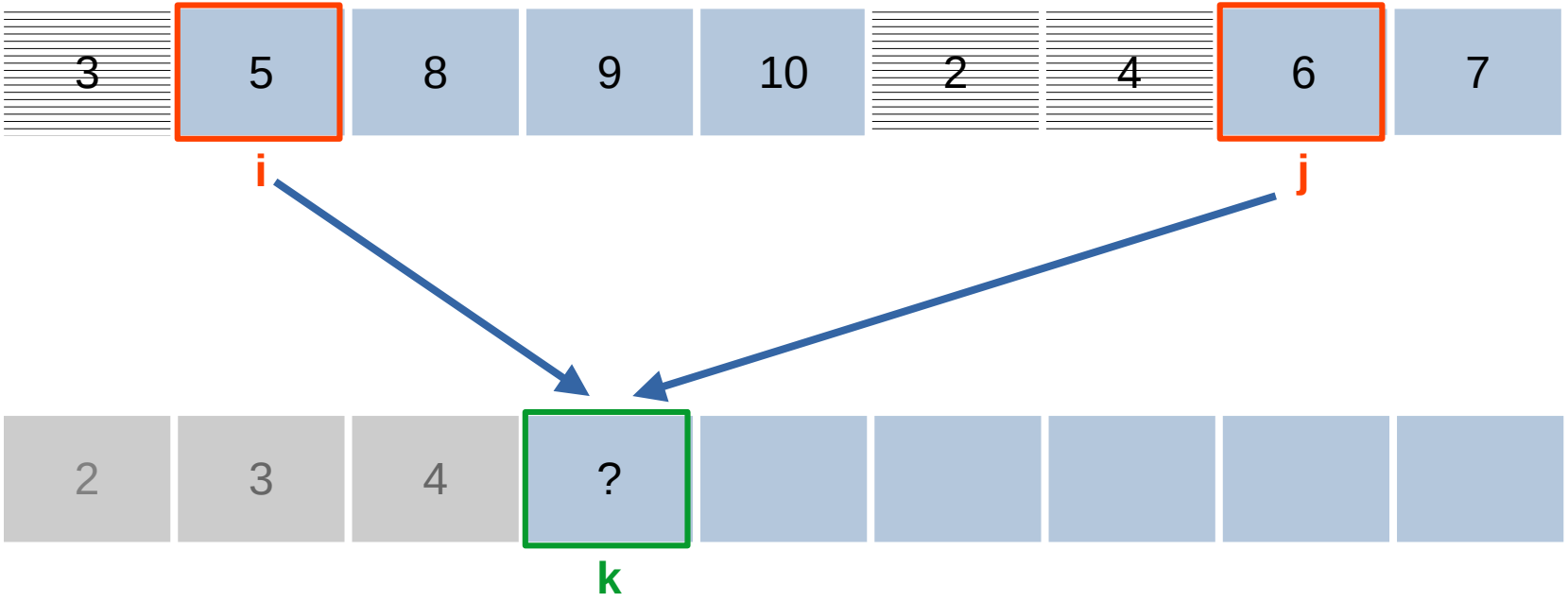
Merge Sort - Intercalação

Exemplo passo a passo (6)



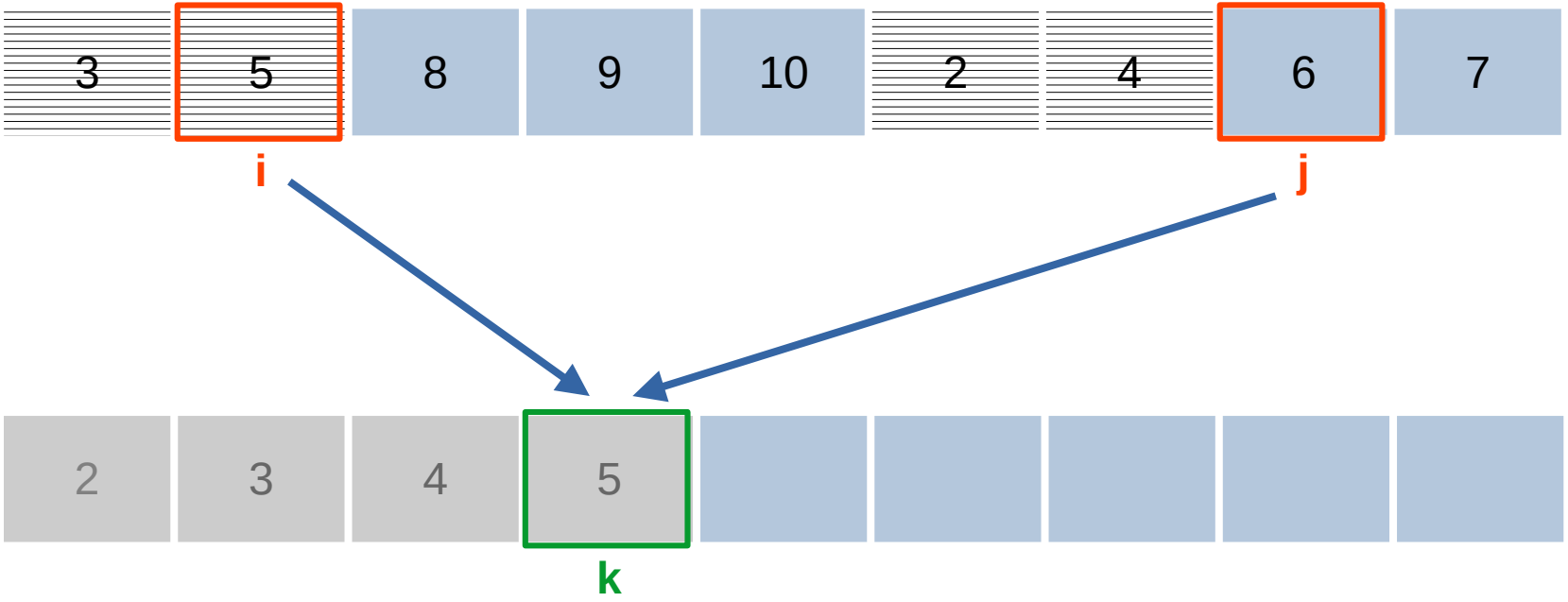
Merge Sort - Intercalação

Exemplo passo a passo (7)



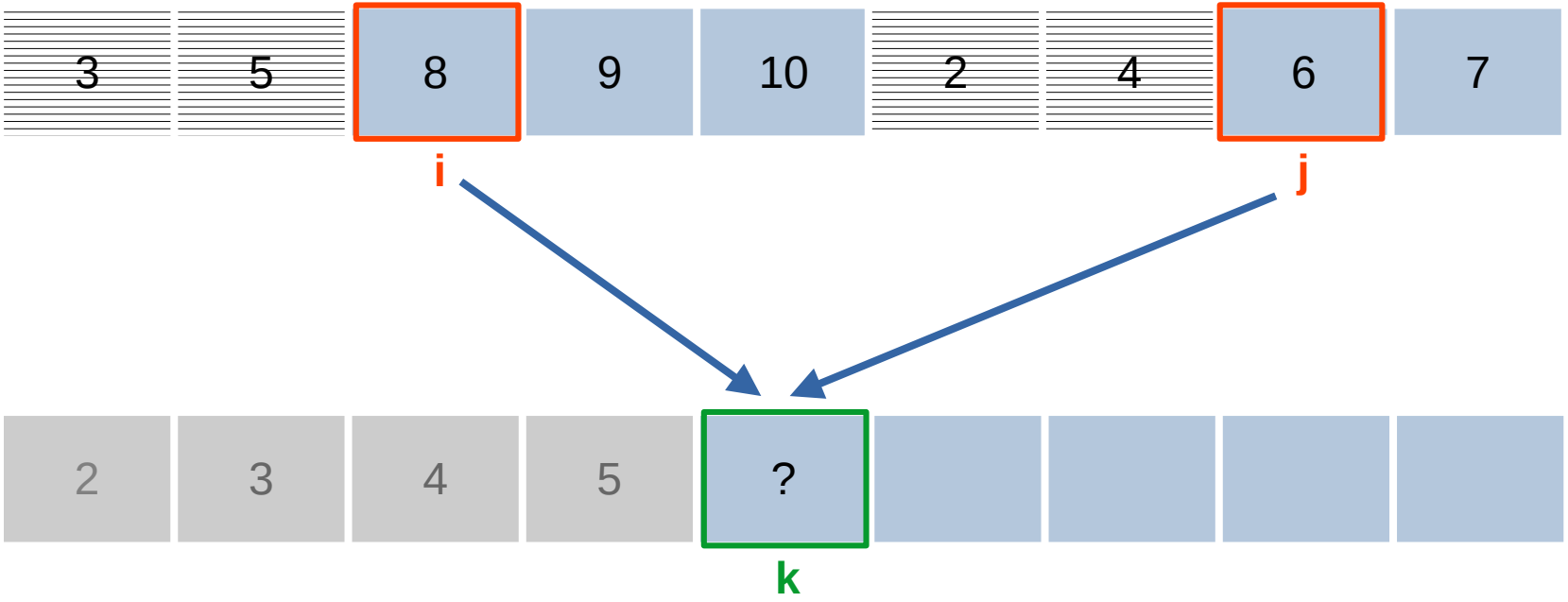
Merge Sort - Intercalação

Exemplo passo a passo (8)



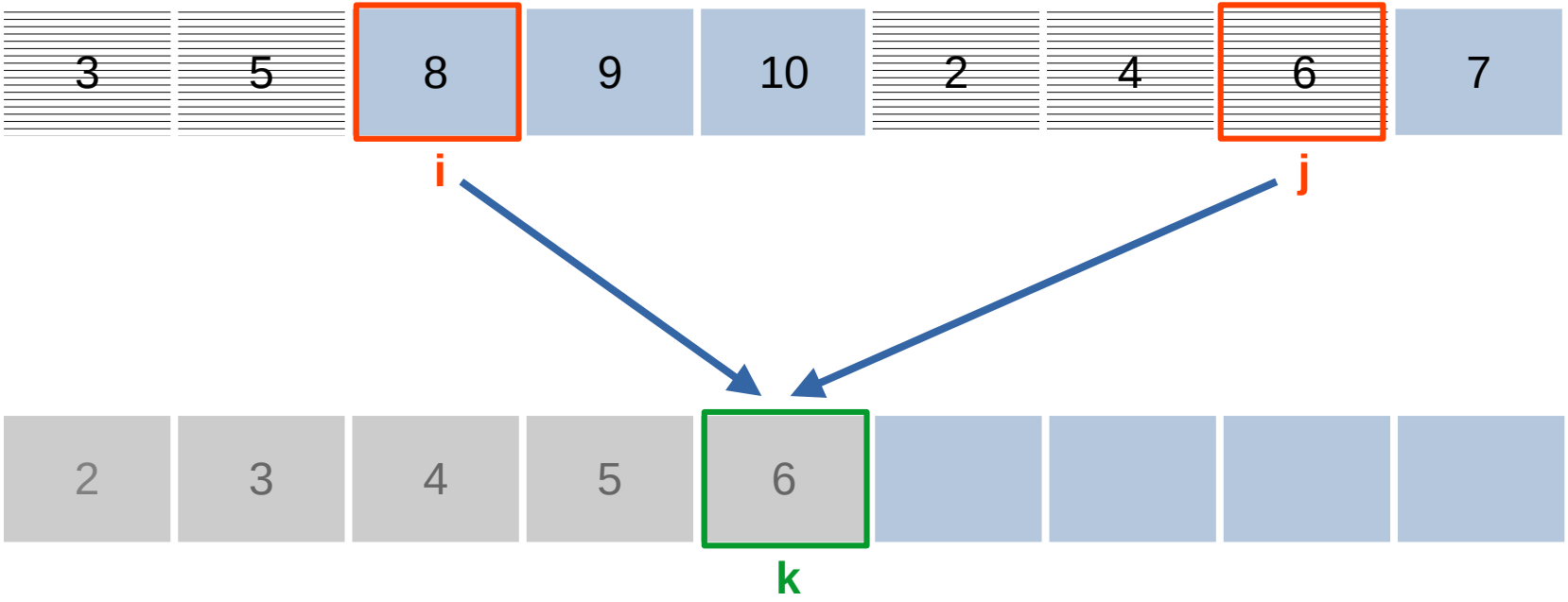
Merge Sort - Intercalação

Exemplo passo a passo (9)



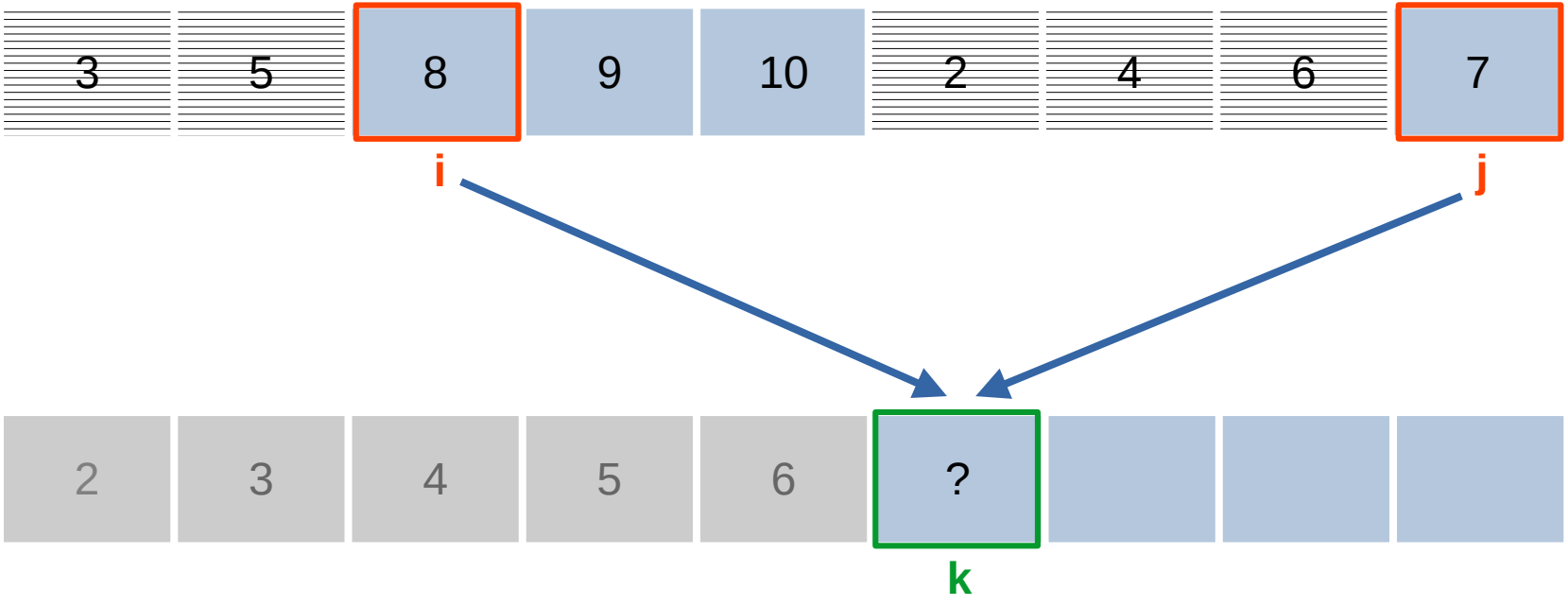
Merge Sort - Intercalação

Exemplo passo a passo (10)



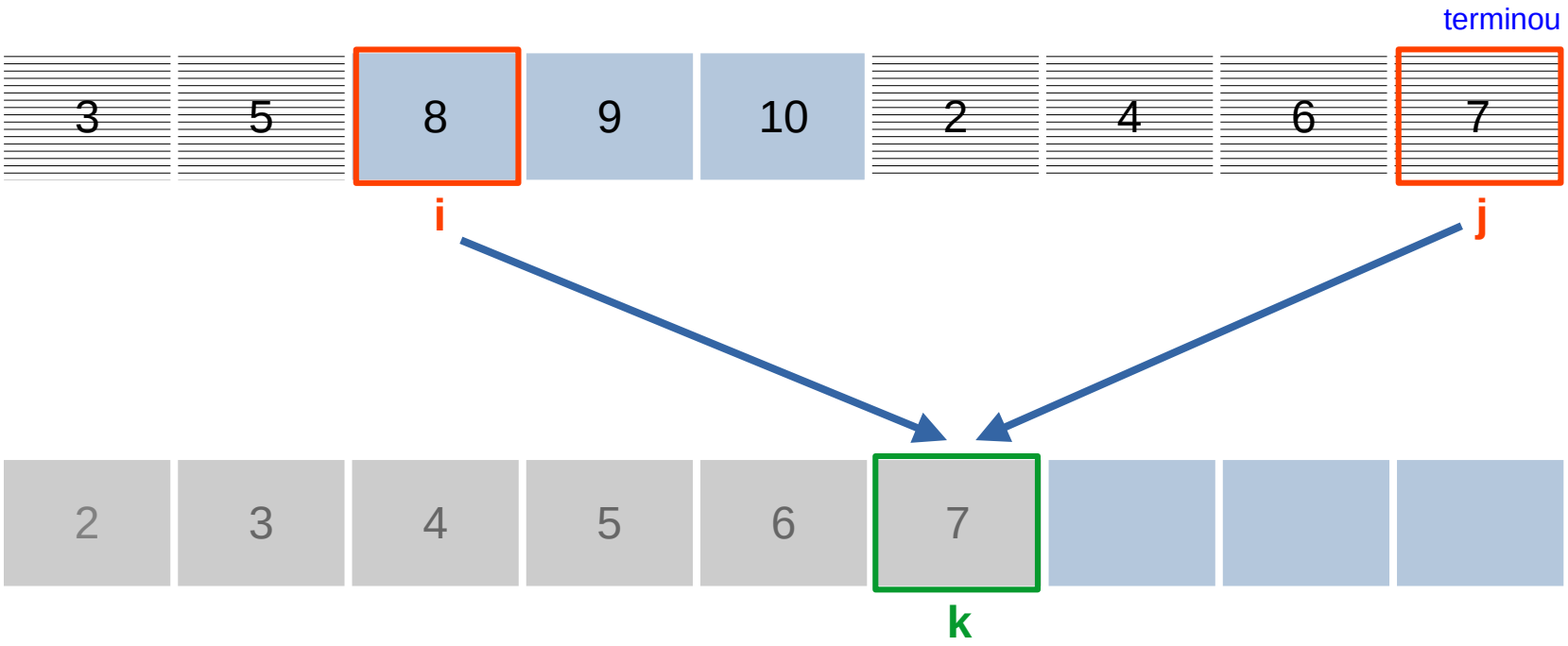
Merge Sort - Intercalação

Exemplo passo a passo (11)



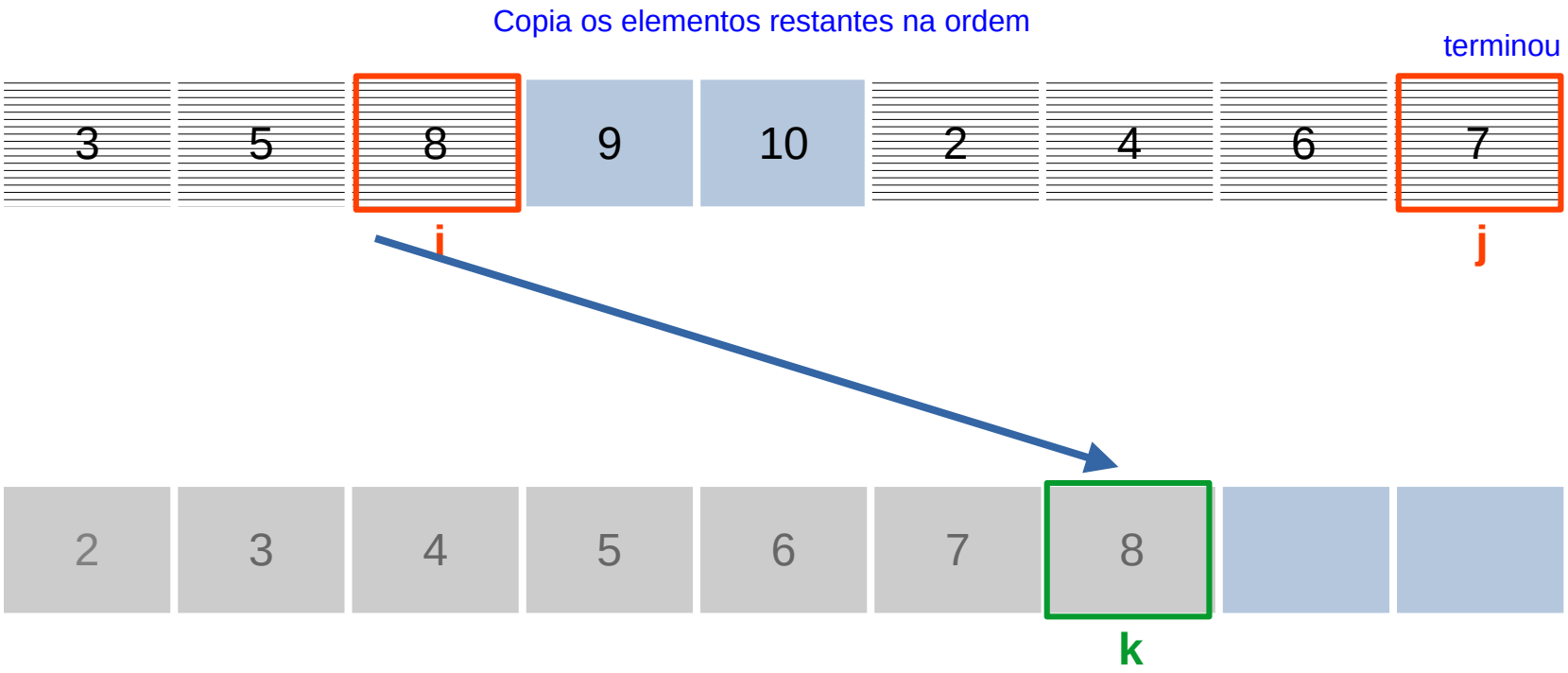
Merge Sort - Intercalação

Exemplo passo a passo (12)



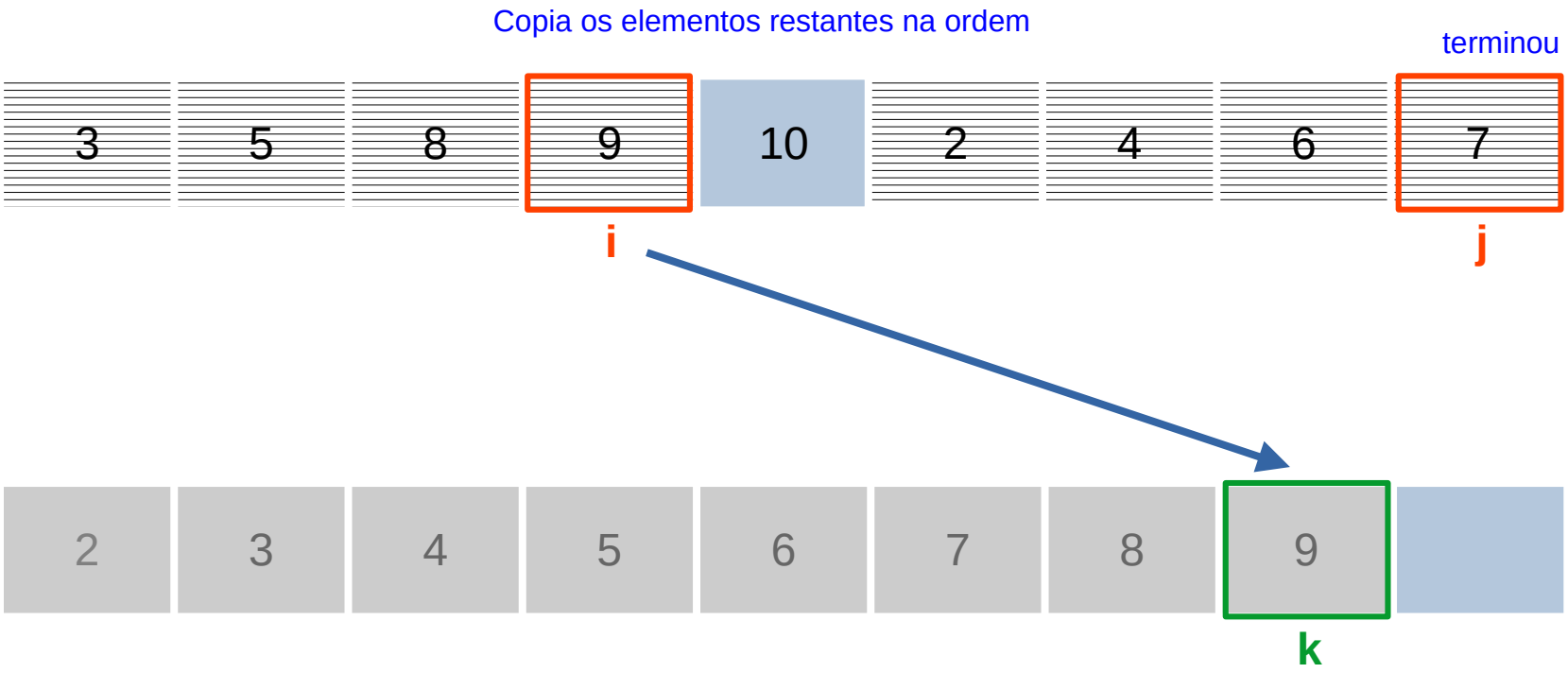
Merge Sort - Intercalação

Exemplo passo a passo (13)



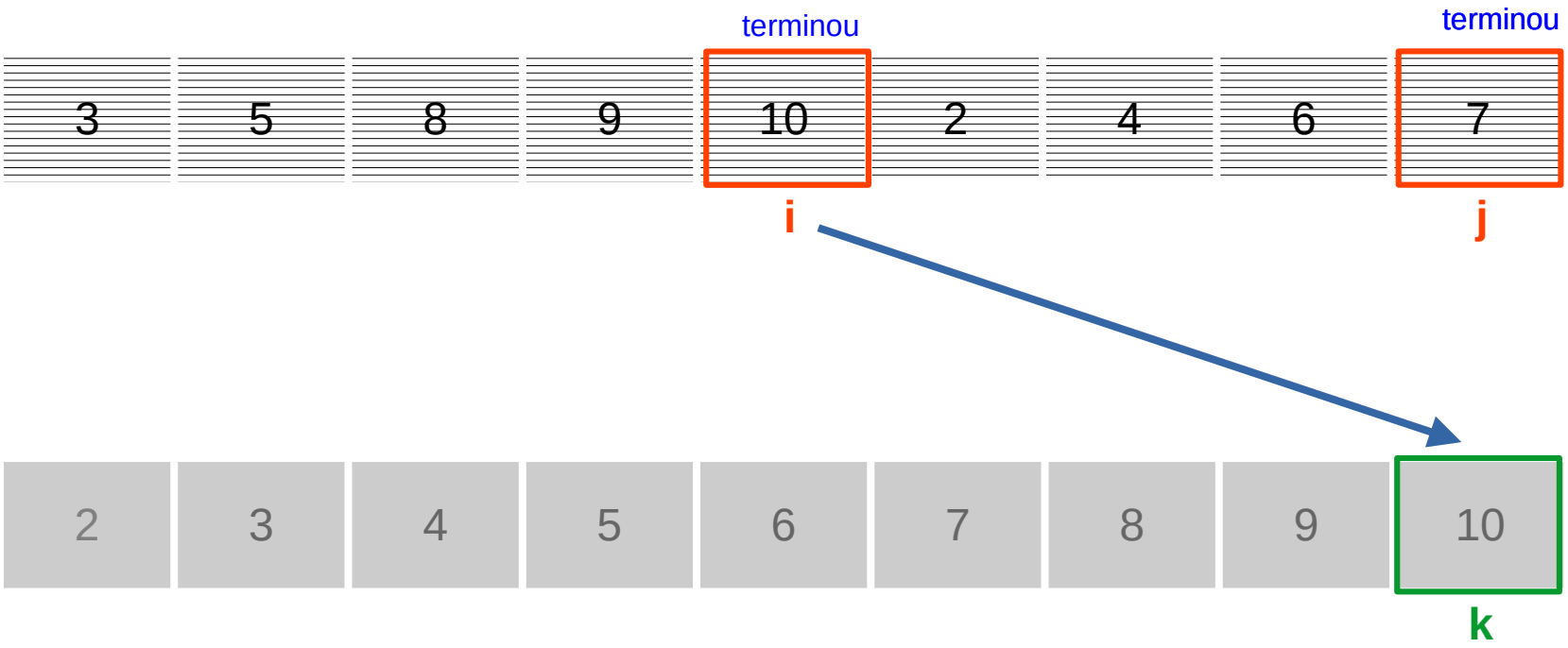
Merge Sort - Intercalação

Exemplo passo a passo (14)



Merge Sort - Intercalação

Exemplo passo a passo (15)



Merge Sort - Outro Exemplo

- Lista original:

23	17	8	15	9	12	19	7

	0	1	2	3	4	5	6	7
mergeSort(A, 0, 7)	23	17	8	**15**	9	12	19	7
mergeSort(A, 0, 3)	23	**17**	8	15	9	12	19	7
mergeSort(A, 0, 1)	**23**	17	8	15	9	12	19	7
mergeSort(A, 0, 0), mergeSort(A, 1, 1)	**23**	**17**	8	15	9	12	19	7
intercala(A, 0, 0, 1)	17	23	8	15	9	12	19	7
mergeSort(A, 2, 3)	17	23	**8**	15	9	12	19	7
mergeSort(A, 2, 2), mergeSort(A, 3, 3)	17	23	**8**	**15**	9	12	19	7
intercala(A, 2, 2, 3)	17	23	8	15	9	12	19	7
intercala(A, 0, 1, 3)	8	15	17	23	9	12	19	7
mergeSort(A, 4, 7)	8	15	17	23	9	**12**	19	7
mergeSort(A, 4, 5)	8	15	17	23	9	12	19	7
mergeSort(A, 4, 4), mergeSort(A, 5, 5)	8	15	17	23	**9**	**12**	19	7
intercala(A, 4, 4, 5)	8	15	17	23	9	12	19	7
mergeSort(A, 6, 7)	8	15	17	23	9	12	**19**	7
mergeSort(A, 6, 6), mergeSort(A, 7, 7)	8	15	17	23	9	12	**19**	**7**
intercala(A, 6, 6, 7)	8	15	17	23	9	12	7	19
intercala(A, 4, 5, 7)	8	15	17	23	7	9	12	19
intercala(A, 0, 3, 7)	7	8	9	12	15	17	19	23

Merge Sort
Análise

- O(n log n) em todos os casos
  - independentemente da ordenação inicial do conjunto de dados
- Complexidade de espaço O(n) → não é *in place*
  - A desvantagem deste algoritmo é precisar de uma lista (vetor) auxiliar para realizar a ordenação, ocasionando gasto extra de memória
- Estável

Merge Sort
Análise

Running time estimates:

- Laptop executes 10^8 compares/second.
- Supercomputer executes 10^{12} compares/second.

	insertion sort (N^2)			mergesort (N log N)		
computer	thousand	million	billion	thousand	million	billion
home	instant	2.8 hours	317 years	instant	1 second	18 min
super	instant	1 second	1 week	instant	instant	instant

Bottom line. Good algorithms are better than supercomputers.

Fonte: Sedgewick, 2016.