UNIFCV
CENTRO UNIVERSITÁRIO CIDADE VERDE

Métodos de ordenação

Prof. Me. Leonardo G. Catharin

prof_leonardo@unifcv.edu.br

Por que estudar métodos de ordenação?

- O problema da ordenação é um dos principais problemas estudados na área de algoritmos
  - Por vezes, ordenar é uma tarefa inerente a uma aplicação. Exemplos:
  - Extratos de clientes (cheques ordenados por seus números).
  - Lista telefônica.
- Algoritmos de ordenação aplicam diversas técnicas estudadas e são utilizados como base para demonstração de limites.

Definição formal

- Problema de ordenação.

> **Entrada:** Uma sequência de n elementos (a_1, a_2, ..., a_n)**.**
>
> **Saída:** Uma permutação (reordenação) (a'_1, a'_2, ..., a'_n) da sequência de entrada, tal que, $a'_1 \leq a'_2 \leq ... \leq a'_n$.

- Dessa forma, dada uma sequência de entrada como (10, 3, 4, 5, 2), um algoritmo de ordenação retorna como saída a sequência (2, 3, 4, 5, 10) (considerando uma ordem ascendente).

Métodos de ordenação

- **Algoritmos** desenvolvidos para rearranjar uma sequência de registros em uma ordem ascendente ou descendente.
- O objetivo principal de ordenar é facilitar uma posterior recuperação dos registros (exemplo: lista telefônica).
- Existem **diversos algoritmos** para ordenar registros.

Classificação de métodos de ordenação

- Podem ser classificados em **dois grandes grupos**.
- **Ordenação interna:** a sequência (arranjo) a ser reordenada se encontra completamente na memória principal.
- **Ordenação externa:** toda ou parte da sequência (arranjo) a ser reordenada se encontra em algum dispositivo de armazenamento não volátil (disco rígido por exemplo).
- A principal diferença é que em um método de ordenação interna, qualquer registro do arranjo pode ser acessado instantaneamente.

Classificação de métodos de ordenação

- Podem ainda ser divididos em métodos baseados em comparações e métodos baseados em distribuição.

Comparação	Distribuição
Uso de chave para ordenação	Uso do princípio de distribuição
Exemplos: Insertion Sort, Bubble Sort, Quick Sort, entre outros.	**Exemplos:** Radixsort, bucketsort, entre outros.

Classificação de métodos de ordenação

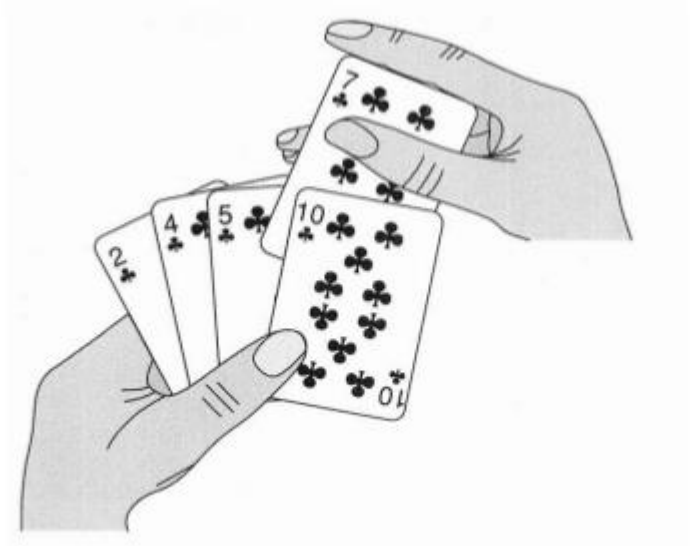
- Classificados em métodos **simples** e métodos **eficientes**.
- Os métodos simples são adequados para pequenos arranjos, enquanto que os métodos eficientes são mais utilizados para maiores volumes de dados.
- Métodos simples, em geral, tem complexidade quadrática **$O(n^2)$**.
- Métodos eficientes, em geral, tem complexidade **$O(n \log n)$**.

Métodos simples

- Ordenação por inserção (Insertion Sort).
- Ordenação por seleção (Selection Sort).
- Ordenação por troca (Bubble Sort).

Ordenação por inserção O(n²)

- Eficiente para ordenar arranjos pequenos.



Adaptado de Cormen *et al*. (2001)

Ordenação por inserção - algoritmo

INSERTION-SORT(A)

1.for j = 2 **to** A.length

2. **do** *key* = A[j]

3. i = j - 1

4. **while** i > 0 and A[i] > *key*

5. **do** A[i + 1] = A[i]

6. i = i - 1

7. A[i + 1] = *key*

Ordenação por inserção - algoritmo

1 ...					n
5	2	4	6	1	3

INSERTION-SORT(A)

1.for j = 2 **to** A.length

2. **do** *key* = A[j]

3. i = j - 1

4. **while** i > 0 and A[i] > *key*

5. **do** A[i + 1] = A[i]

6. i = i - 1

7. A[i + 1] = *key*

Ordenação por inserção - algoritmo

INSERTION-SORT(A)

1.for j = 2 **to** A.length

2. **do** *key* = A[j]

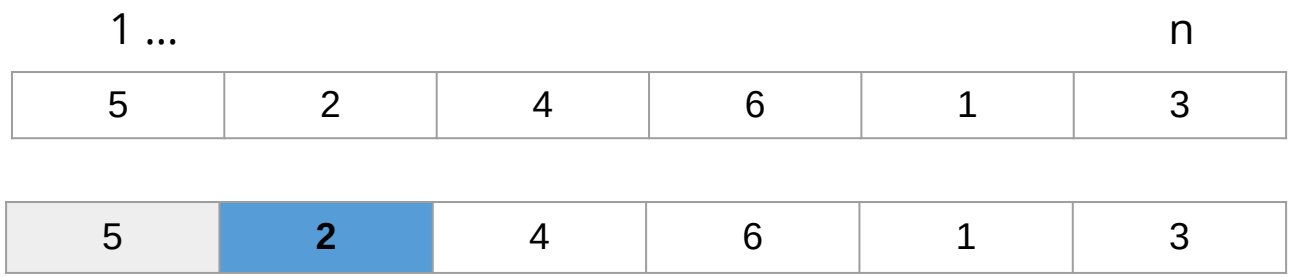
3. i = j - 1

4. **while** i > 0 and A[i] > *key*

5. **do** A[i + 1] = A[i]

6. i = i - 1

7. A[i + 1] = *key*



Ordenação por inserção - algoritmo

INSERTION-SORT(A)

1.for j = 2 **to** A.length

2. **do** *key* = A[j]

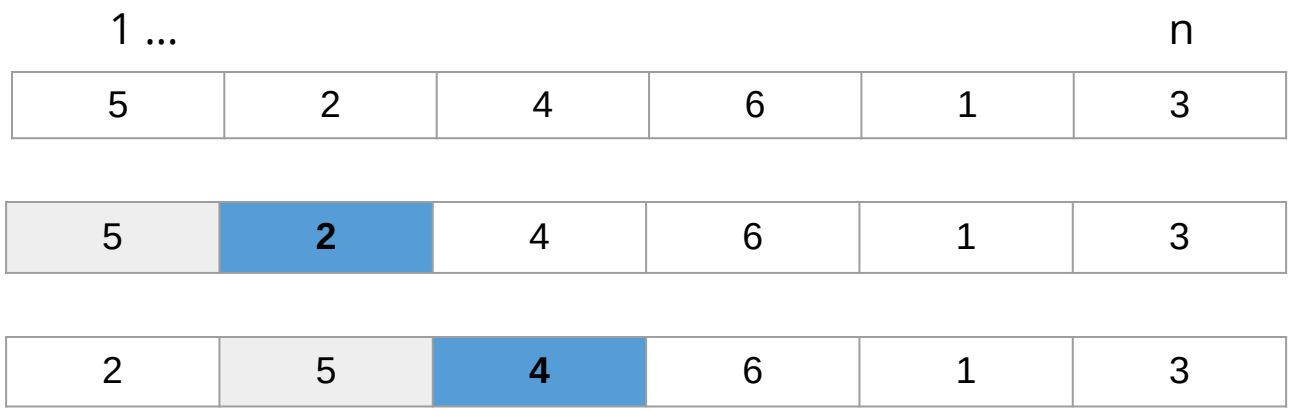
3. i = j - 1

4. **while** i > 0 and A[i] > *key*

5. **do** A[i + 1] = A[i]

6. i = i - 1

7. A[i + 1] = *key*



Ordenação por inserção - algoritmo

INSERTION-SORT(A)

1.for j = 2 **to** A.length

2. **do** *key* = A[j]

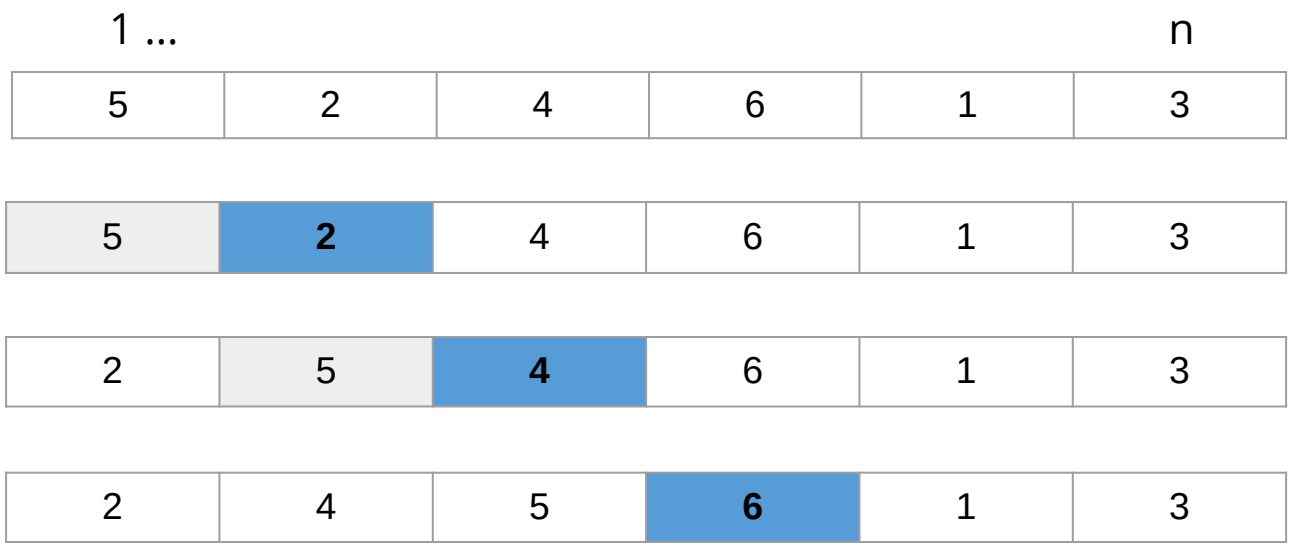
3. i = j - 1

4. **while** i > 0 and A[i] > *key*

5. **do** A[i + 1] = A[i]

6. i = i - 1

7. A[i + 1] = *key*



Ordenação por inserção - algoritmo

INSERTION-SORT(A)

1.for j = 2 **to** A.length

2. **do** *key* = A[j]

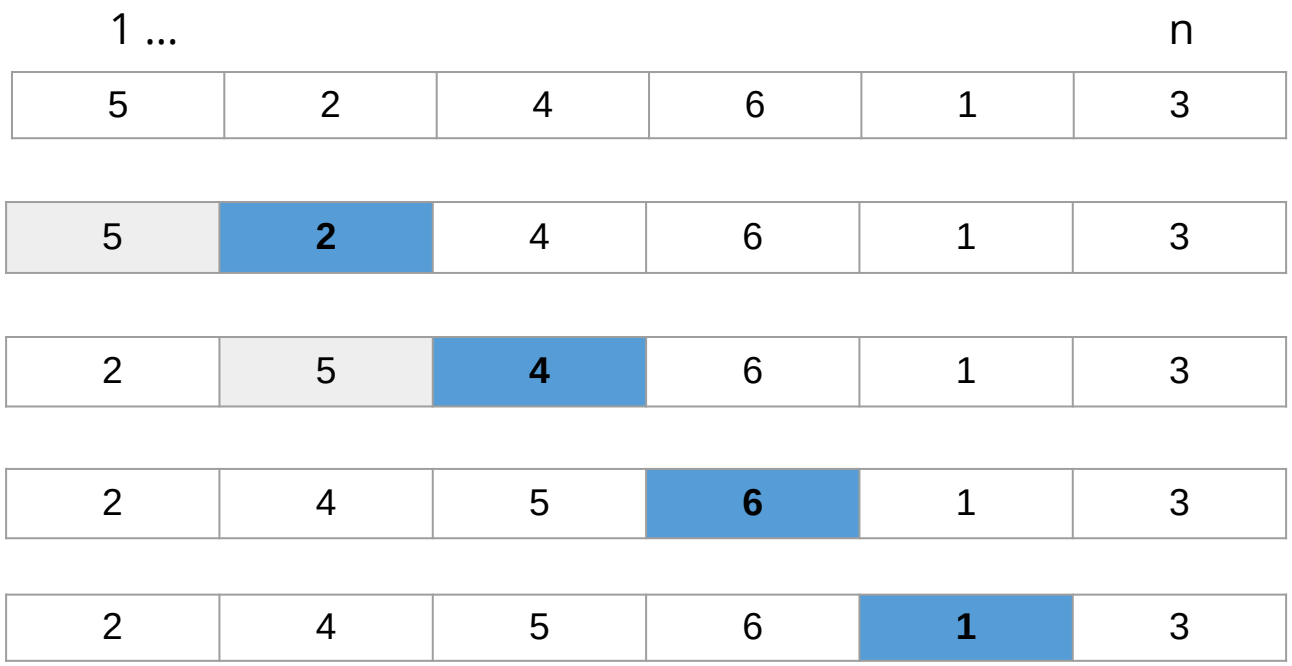
3. i = j - 1

4. **while** i > 0 and A[i] > *key*

5. **do** A[i + 1] = A[i]

6. i = i - 1

7. A[i + 1] = *key*



Ordenação por inserção - algoritmo

INSERTION-SORT(A)

1.for j = 2 **to** A.length

2. **do** *key* = A[j]

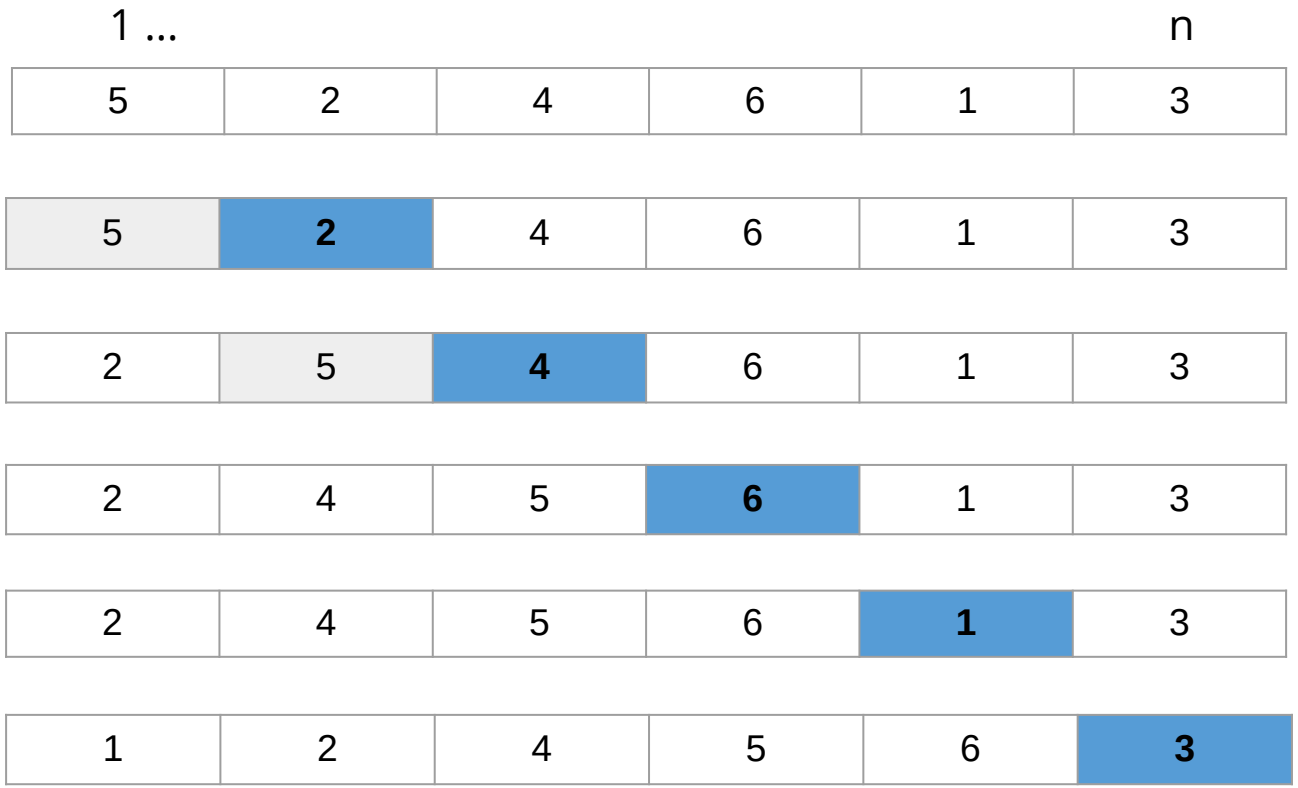
3. i = j - 1

4. **while** i > 0 and A[i] > *key*

5. **do** A[i + 1] = A[i]

6. i = i - 1

7. A[i + 1] = *key*



Ordenação por inserção - algoritmo

INSERTION-SORT(A)

1.for j = 2 **to** A.length

2. **do** *key* = A[j]

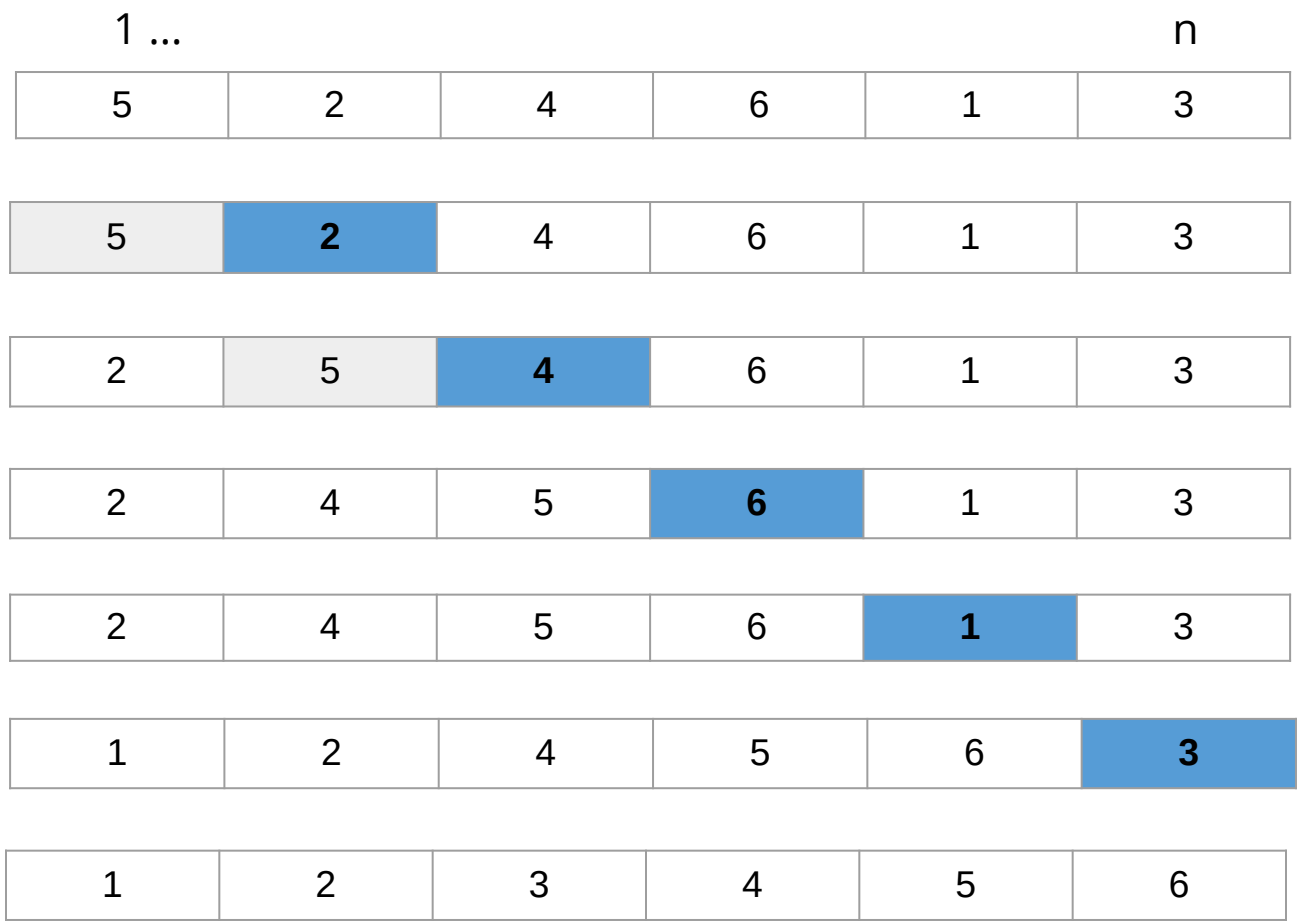
3. i = j - 1

4. **while** i > 0 and A[i] > *key*

5. **do** A[i + 1] = A[i]

6. i = i - 1

7. A[i + 1] = *key*



Exemplo

- Implementação do **Insertion Sort.**

Ordenação por seleção – O(n²)

- Encontrar o menor elemento do arranjo e inseri-lo na primeira posição do arranjo.

- Realizar a mesma operação para todos os elementos.

Ordenação por inserção - algoritmo

SELECTION-SORT(A)

1. **for** i = 1 **to** n – 1 **do**
2. *min* = i
3. **for** j = i + 1 **to** n **do**
4. **if** A[j] < A[min]
5. **do** min = j
6. x = A[min]
7. A[min] = A[i]
8. A[i] = x

Ordenação por inserção - algoritmo

1 …					n
5	2	4	6	1	3

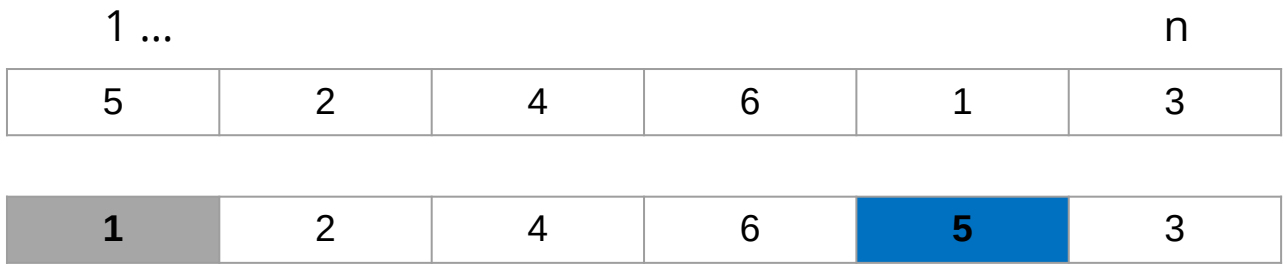
SELECTION-SORT(A)

1. **for** i = 1 **to** n – 1 **do**
2. *min* = i
3. **for** j = i + 1 **to** n **do**
4. **if** A[j] < A[min]
5. **do** min = j
6. x = A[min]
7. A[min] = A[i]
8. A[i] = x

Ordenação por inserção - algoritmo

SELECTION-SORT(A)

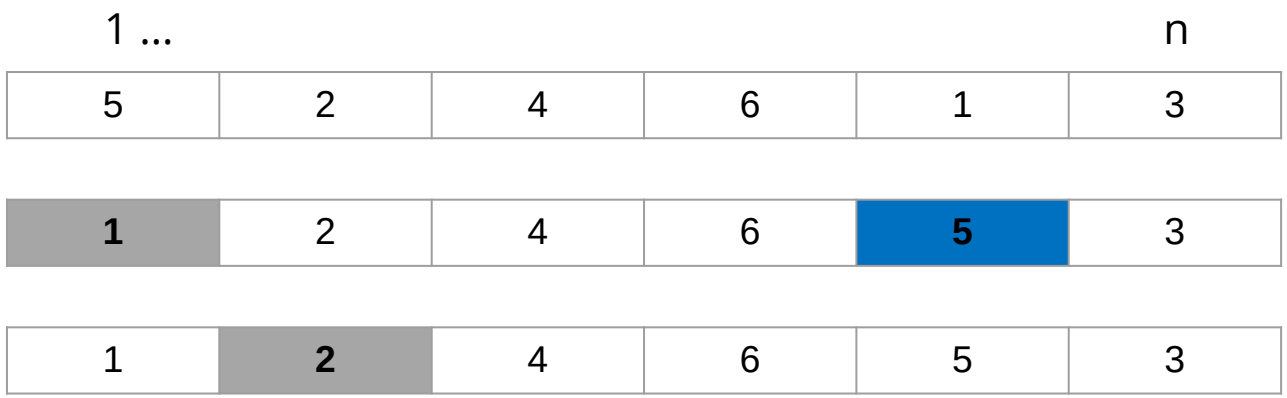
1. **for** i = 1 **to** n – 1 **do**
2. *min* = i
3. **for** j = i + 1 **to** n **do**
4. **if** A[j] < A[min]
5. **do** min = j
6. x = A[min]
7. A[min] = A[i]
8. A[i] = x



Ordenação por inserção - algoritmo

SELECTION-SORT(A)

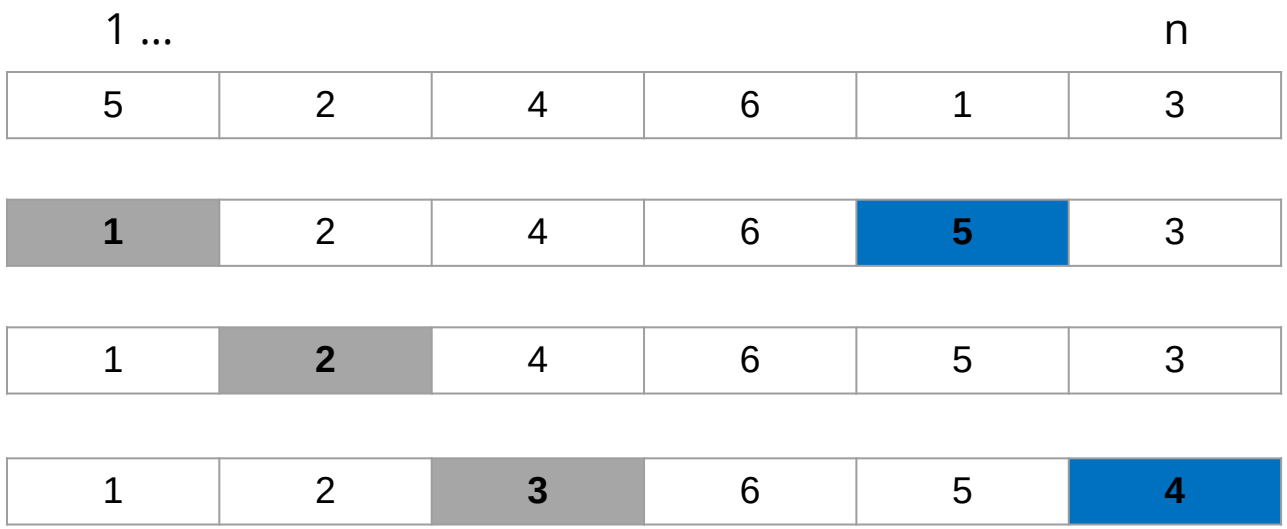
1. **for** i = 1 **to** n – 1 **do**
2. *min* = i
3. **for** j = i + 1 **to** n **do**
4. **if** A[j] < A[min]
5. **do** min = j
6. x = A[min]
7. A[min] = A[i]
8. A[i] = x



Ordenação por inserção - algoritmo

SELECTION-SORT(A)

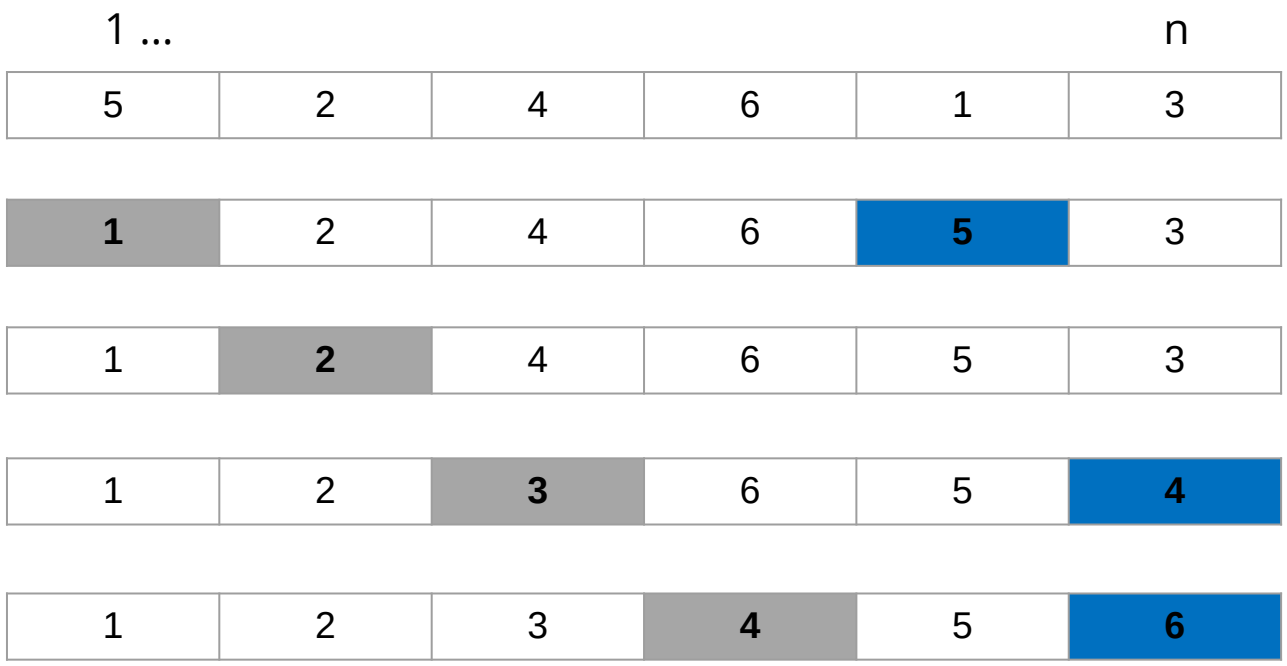
1. **for** i = 1 **to** n – 1 **do**
2. *min* = i
3. **for** j = i + 1 **to** n **do**
4. **if** A[j] < A[min]
5. **do** min = j
6. x = A[min]
7. A[min] = A[i]
8. A[i] = x



Ordenação por inserção - algoritmo

SELECTION-SORT(A)

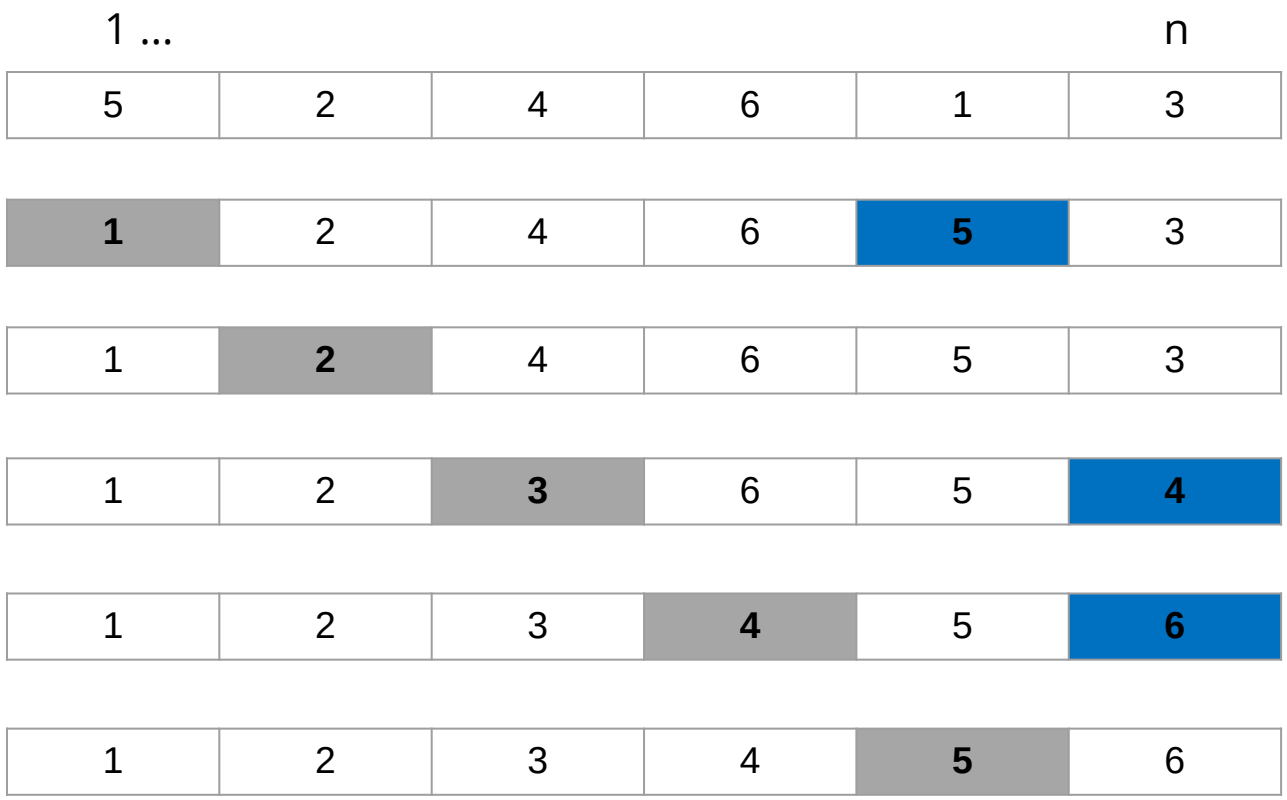
1. **for** i = 1 **to** n – 1 **do**
2. *min* = i
3. **for** j = i + 1 **to** n **do**
4. **if** A[j] < A[min]
5. **do** min = j
6. x = A[min]
7. A[min] = A[i]
8. A[i] = x



Ordenação por inserção - algoritmo

SELECTION-SORT(A)

1. **for** i = 1 **to** n – 1 **do**
2. *min* = i
3. **for** j = i + 1 **to** n **do**
4. **if** A[j] < A[min]
5. **do** min = j
6. x = A[min]
7. A[min] = A[i]
8. A[i] = x



Exemplo

- Implementação do **Selection Sort.**

Ordenação por troca – O(n²)

- Permutação sistemática, par a par, entre os elementos do arranjo.

- Estratégia bolha: **borbulhar** o maior elemento para o fim da lista.

Ordenação por troca - algoritmo

BUBBLE-SORT(A)

```
1. for i = 1 to n – 1 do
2.     for j = 1 to n – 1 - i do
3.         if A[j] > A[j+1] do
4.             swap(A[j], A[j + 1])
```

Ordenação por troca - algoritmo

BUBBLE-SORT(A)

1. **for** i = 1 **to** n – 1 **do**
2. **for** j = 1 **to** n – 1 - i **do**
3. **if** A[j] > A[j+1] **do**
4. swap(A[j], A[j + 1])

1 ...					n
5	2	4	6	1	3

Ordenação por troca - algoritmo

BUBBLE-SORT(A)

1. **for** i = 1 **to** n – 1 **do**
2. **for** j = 1 **to** n – 1 - i **do**
3. **if** A[j] > A[j+1] **do**
4. swap(A[j], A[j + 1])

1 ...					n
5	2	4	6	1	3

2	4	5	1	3	**6**

Ordenação por troca - algoritmo

BUBBLE-SORT(A)

1. **for** i = 1 **to** n – 1 **do**
2. **for** j = 1 **to** n – 1 - i **do**
3. **if** A[j] > A[j+1] **do**
4. swap(A[j], A[j + 1])

1 ...					n
5	2	4	6	1	3
2	4	5	1	3	**6**
2	4	1	3	**5**	**6**

Ordenação por troca - algoritmo

BUBBLE-SORT(A)

1. **for** i = 1 **to** n – 1 **do**
2. **for** j = 1 **to** n – 1 - i **do**
3. **if** A[j] > A[j+1] **do**
4. swap(A[j], A[j + 1])

1 ...					n
5	2	4	6	1	3
2	4	5	1	3	**6**
2	4	1	3	**5**	**6**
2	1	3	**4**	**5**	**6**

Ordenação por troca - algoritmo

BUBBLE-SORT(A)

1. **for** i = 1 **to** n – 1 **do**
2. **for** j = 1 **to** n – 1 - i **do**
3. **if** A[j] > A[j+1] **do**
4. swap(A[j], A[j + 1])

1 ...					n
5	2	4	6	1	3
2	4	5	1	3	**6**
2	4	1	3	**5**	**6**
2	1	3	**4**	**5**	**6**
1	2	**3**	**4**	**5**	**6**

Ordenação por troca - algoritmo

BUBBLE-SORT(A)

1. **for** i = 1 **to** n – 1 **do**
2. **for** j = 1 **to** n – 1 - i **do**
3. **if** A[j] > A[j+1] **do**
4. swap(A[j], A[j + 1])

1 ...					n
5	2	4	6	1	3
2	4	5	1	3	**6**
2	4	1	3	**5**	**6**
2	1	3	**4**	**5**	**6**
1	2	**3**	**4**	**5**	**6**
1	**2**	**3**	**4**	**5**	**6**

Ordenação por troca - algoritmo

BUBBLE-SORT(A)

1. **for** i = 1 **to** n – 1 **do**
2. **for** j = 1 **to** n – 1 - i **do**
3. **if** A[j] > A[j+1] **do**
4. swap(A[j], A[j + 1])

1 ...					n
5	2	4	6	1	3
2	4	5	1	3	**6**
2	4	1	3	**5**	**6**
2	1	3	**4**	**5**	**6**
1	2	**3**	**4**	**5**	**6**
1	**2**	**3**	**4**	**5**	**6**
1	**2**	**3**	**4**	**5**	**6**

Exemplo

- Implementação do **Bubble Sort.**

Complexidade

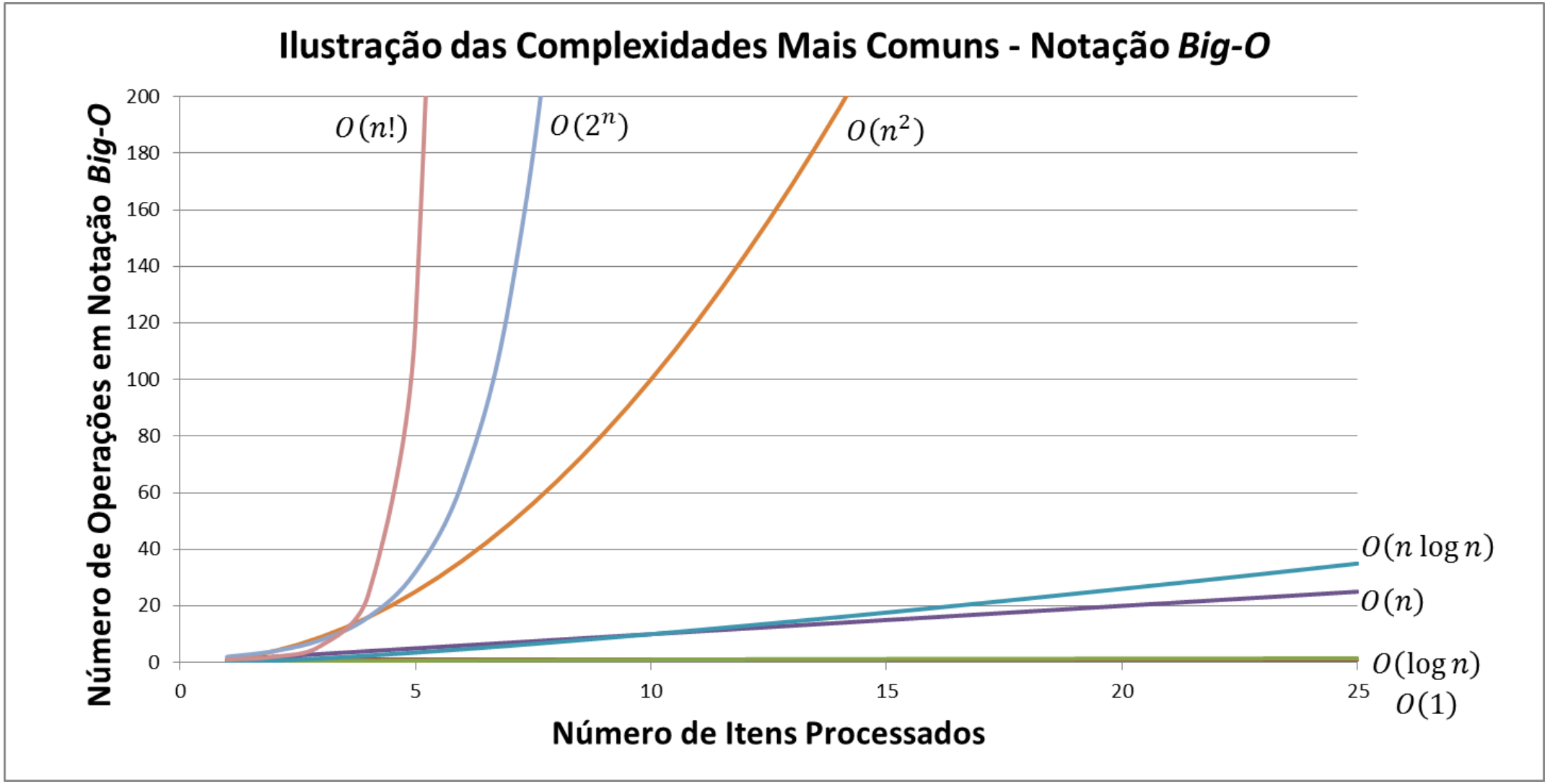


Imagem disponível em: *https://pt.stackoverflow.com/questions/56836/defini%C3%A7%C3%A3o-da-nota%C3%A7%C3%A3o-big-o*

Complexidade

BUBBLE-SORT(A)

1. **for** i = 1 **to** n – 1 **do** **(n - 1)**
2. **for** j = 1 **to** n – 1 - i **do** **(n – 1 - i)**
3. **if** A[j] > A[j+1] **do** **(1)**
4. swap(A[j], A[j + 1]) **(1)**

(n - 1).(n – 1 - i).(1).(1)

(n ~~- 1~~).(n ~~– 1 - i).(1).(1)~~

$n.n = n^2$

Limite superior do algoritmo é **$O(n^2)$**.

Significa dizer que o *Bubble Sort* leva no máximo n^2 para ser executado.

Métodos Eficientes

- Ordenação rápida (Quick Sort).

QuickSort

- Baseado no paradigma de dividir e conquistar.
- Tempo de execução esperado **Θ (n lg n)**, porém no pior caso leva **Θ (n²)**.
- Devido a sua média, o QuickSort é constantemente indicado como a melhor opção prática.
- O pior caso desse algoritmo quando os subarranjos são divididos em tamanhos 0 e n-1, ou seja, completamente desbalanceados.

Ordenação rápida - algoritmo

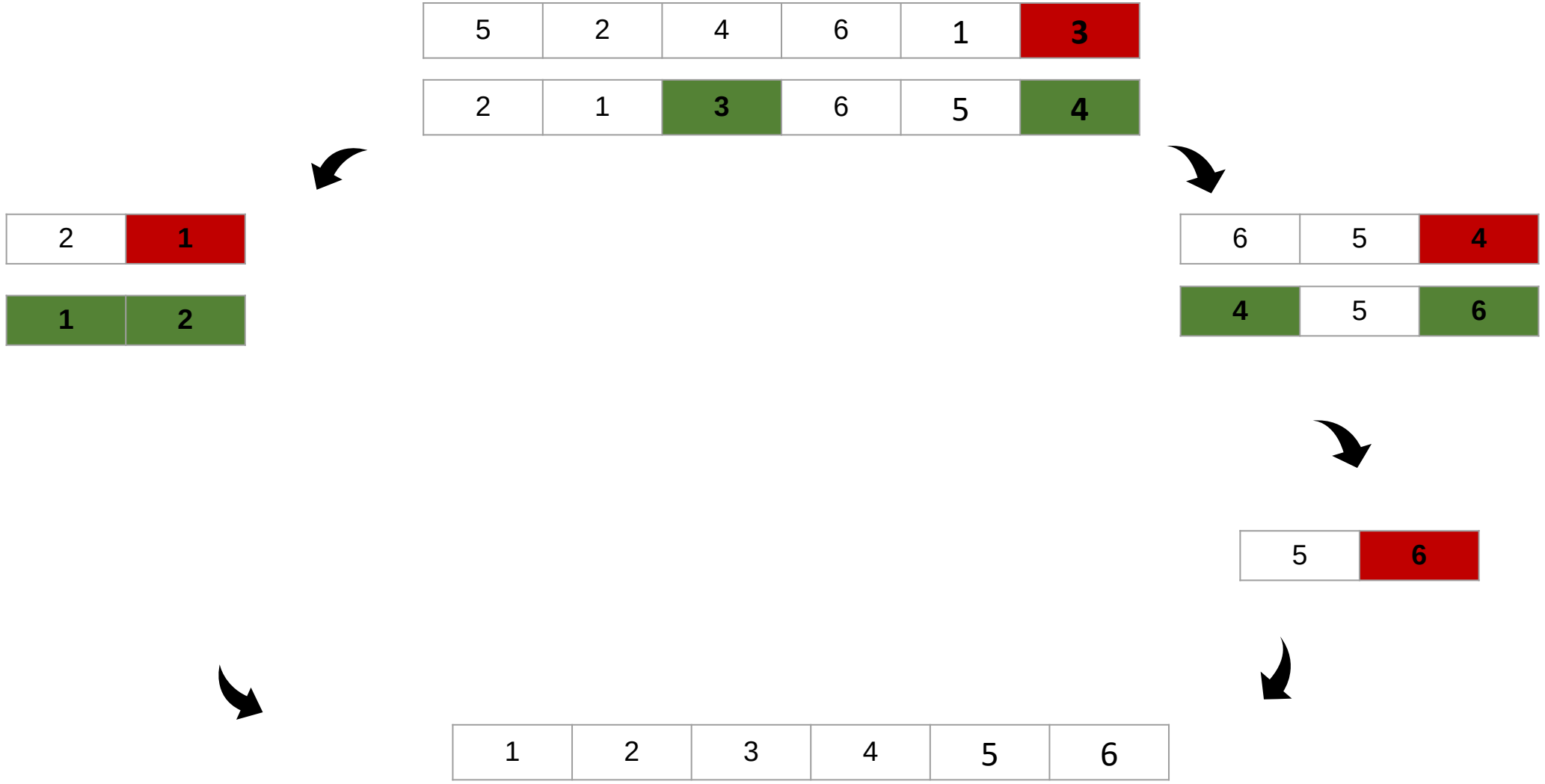
QUICKSORT(A)

1. **if** p < r
2. **do** q = PARTITION (A, p, r) **/* p = posição inicial e r = tamanho do arranjo*/**
3. QUICKSORT(A, p, q - 1)
4. QUICKSORT(A, q + 1, r)

PARTITION(A, p, r)

1. x = A[r] **// seleção do pivô**
2. i = p – 1
3. **for** j = p **to** r – 1 **do**
4. **if** A[j] ≤ x **do**
5. i = i + 1
6. swap(A[i], A[j])
7. swap(A[i + 1], A[r])
8. **return** i + 1

Ordenação rápida - algoritmo



Referências

- CORMEN, Thomas H. et al. Introduction to algorithms second edition. 2001.

- ZIVIANI, Nivio et al. Projeto de algoritmos: com implementações em Pascal e C. Thomson, 2004.