Análise e Projeto de Algoritmos

Aula 2 –Algoritmos de ordenação iterativos

Prof. Bruno Bruck

Centro de Informática
UFPB

2018.1

Sumário

- Algoritmos de ordenação
- Insertion Sort (Ordenação por inserção)
  - Ideia + pseudocódigo
  - Exemplos
  - Análise assintótica
- Selection Sort (Ordenação por seleção)
  - Ideia + pseudocódigo
  - Exemplos
  - Análise assintótica

Algoritmos de ordenação

Algoritmos de ordenação

- Ordenação
  - Processo de rearranjar um conjunto de objetos em uma ordem
    - Ascendente ou decrescente
  - Visa facilitar a recuperação posterior de itens do conjunto ordenado
    - Ex.: Busca binária
    - Imagine tentar encontrar a pessoa com maior nota em uma lista não ordenada do ENEM (e sem busca)

Algoritmos de ordenação

- A maioria dos algoritmos de ordenação
  - Baseados em comparações de elementos
    - A > B / A < B / A = B
  - Entretanto existem outros métodos de ordenação que não realizam comparações!
    - Veremos alguns ao longo do curso

Algoritmos de ordenação

- Existem diversos algoritmos de ordenação
- Como escolher o melhor?
  - Depende muito do contexto prático
- Fatores a considerar
  - Complexidade assintótica
  - Tamanho da entrada (n = número de elementos)
  - Complexidade da implementação
  - Restrições de hardware (ex.: Arduino, Raspberry Pi Zero)

Algoritmos de ordenação

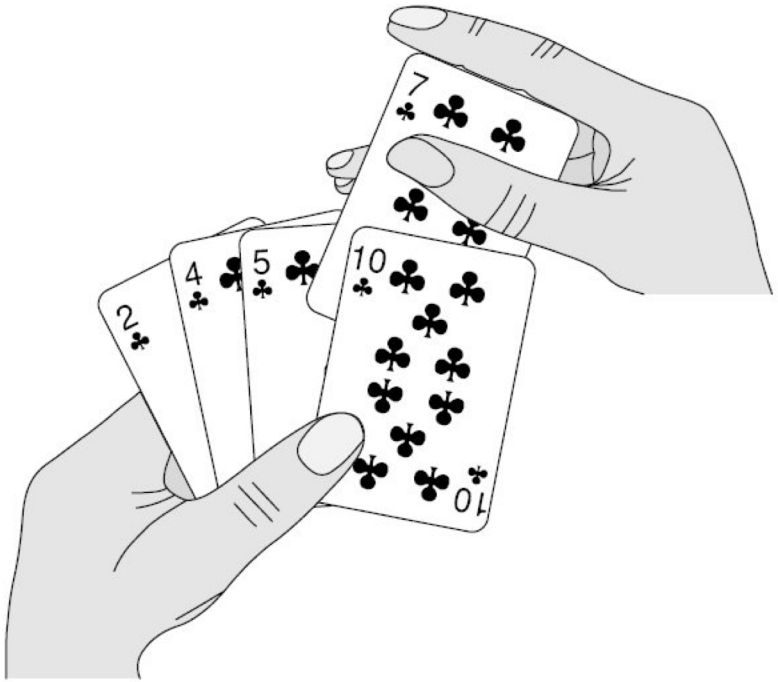
- Algoritmos mais simples
  - Adequados para **pequenos** conjuntos de dados
  - Complexidade assintótica **maior**
  - Implementação mais **fácil** e códigos menores

- Algoritmos eficientes
  - Adequados para conjuntos de dados **maiores**
  - Complexidade assintótica **menor**
  - Implementação mais complexa

Insertion Sort

Insertion Sort

- Ideia básica
  - Similar ao modo em que cartas de baralho são ordenadas na mão de um jogador



Insertion Sort

- Ideia básica
  - Imagine um baralho de cartas virado com a face para baixo
  - Inicialmente a mão do jogador está vazia
  - Uma a uma, cartas do baralho são reveladas e adicionadas à mão do jogador
    - Inseridas na posição correta (ordenada)
    - Feito comparando-se cada nova carta com as cartas na mão do jogador, da direta para a esquerda

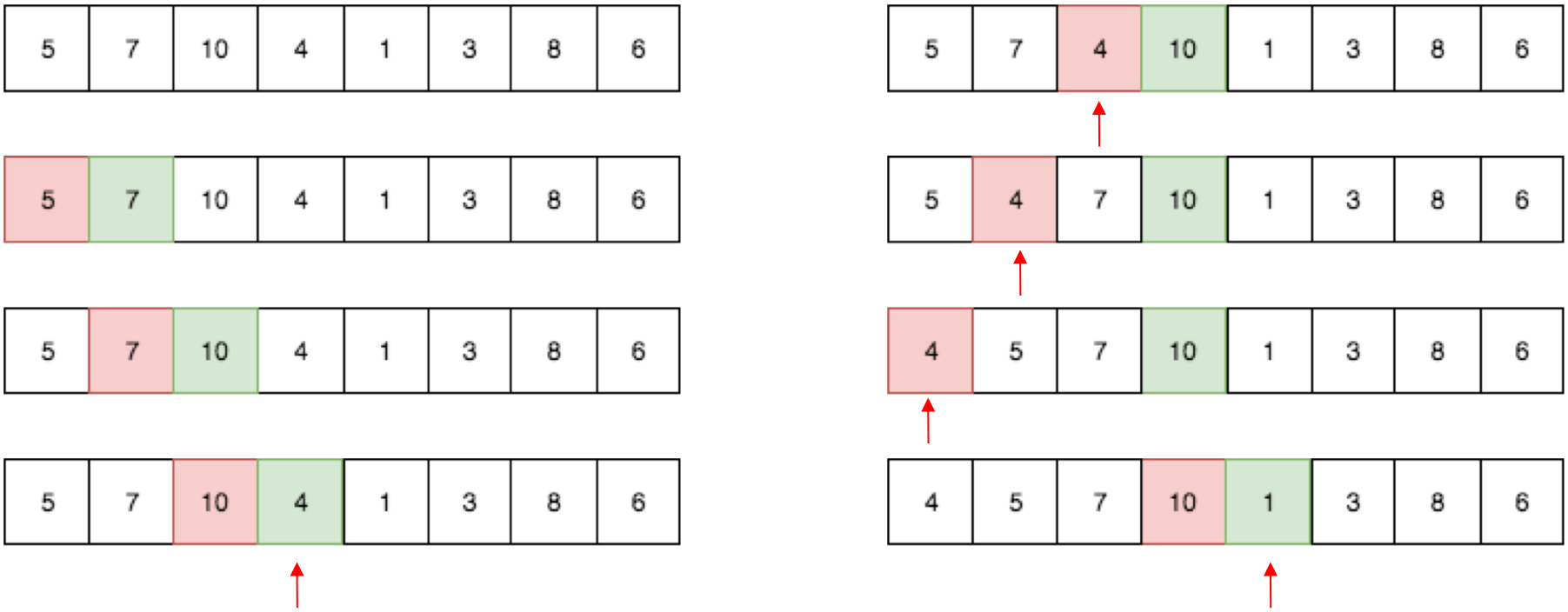
Insertion Sort

- Psedocódigo

```
InsertionSort(A, n)
1.    pivo ← null
2.    for i ← 1 to n – 1 do
3.        pivo ← A[i]
4.        j ← i - 1
5.        while j ≥ 0 and A[j] > pivo do
6.            A[j+1] ← A[j]
7.            j ← j - 1
8.        A[j+1] ← pivo
9.    return A
```

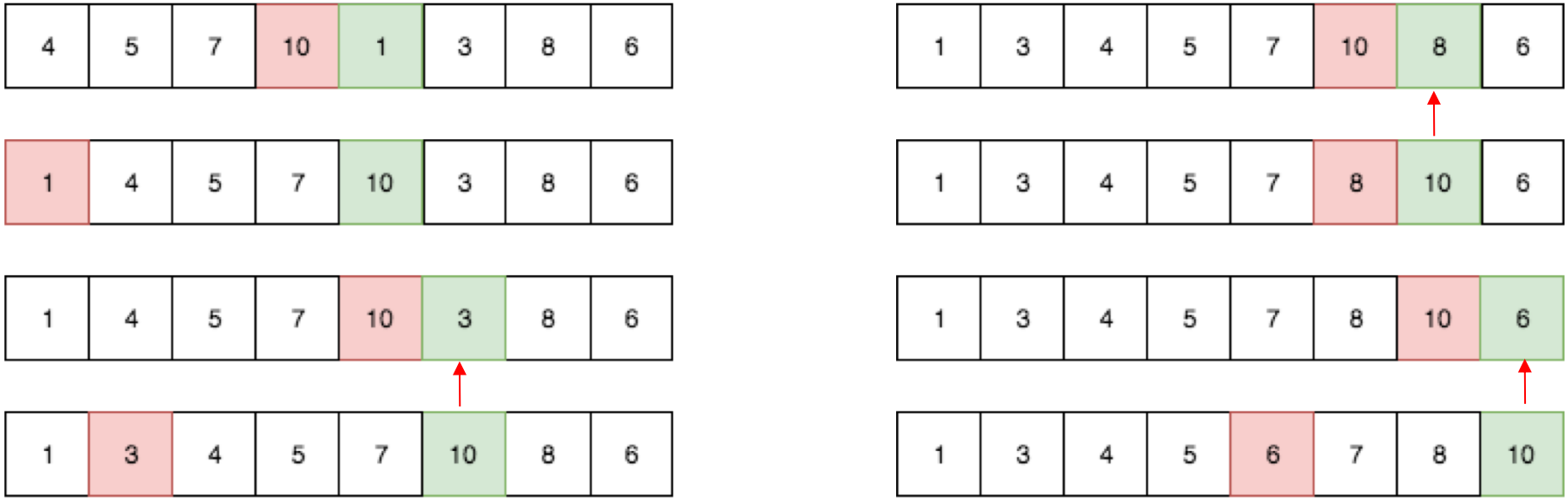
Insertion Sort

- Exemplo funcionamento



Insertion Sort

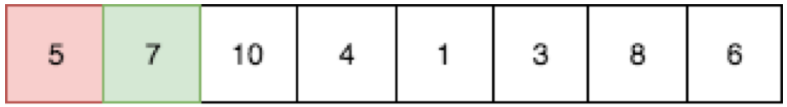
- Exemplo funcionamento



Ordenado!!

Insertion Sort

- Qual o **melhor** caso?




```
InsertionSort(A, n)
1.    pivo ← null
2.    for i ← 1 to n – 1 do
3.        pivo ← A[i]
4.        j ← i - 1
5.        while j ≥ 0 and A[j] > pivo do
6.            A[j+1] ← A[j]
7.            j ← j - 1
8.        A[j+1] ← pivo
9.    return A
```

Número de execuções baseado somente em n!

Número de execuções baseado na distribuição dos valores no array!!

Insertion Sort

- Qual o **melhor** caso?
  - Array já ordenado: [1, 3, 4, 5, 6, 7, 8, 10]
    - Não executa nenhuma operação de troca

```
InsertionSort(A, n)
1.    pivo ← null
2.    for i ← 1 to n – 1 do
3.        pivo ← A[i]
4.        j ← i - 1
5.        while j ≥ 0 and A[j] > pivo do
6.            A[j+1] ← A[j]
7.            j ← j - 1
8.        A[j+1] ← pivo
9.    return A
```

Número de execuções baseado somente em n!

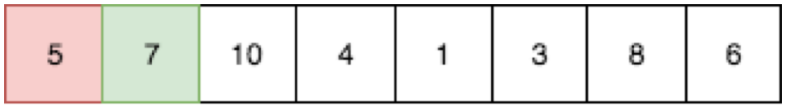
(n -1) vezes

$O(n)$

$c = 1, n_0 = 2$

Insertion Sort

- Qual o **pior** caso?




```
InsertionSort(A, n)
1.    pivo ← null
2.    for i ← 1 to n – 1 do
3.        pivo ← A[i]
4.        j ← i - 1
5.        while j ≥ 0 and A[j] > pivo do
6.            A[j+1] ← A[j]
7.            j ← j - 1
8.        A[j+1] ← pivo
9.    return A
```

Número de execuções baseado somente em n!

Número de execuções baseado na distribuição dos valores no array!!

Insertion Sort

- Qual o pior caso?
  - Vetor ordenado ao contrário! [10, 8, 7, 6, 5, 4, 3, 1]
    - Maior número de execuções do loop da linha 5.

```
InsertionSort(A, n)
1.    pivo ← null
2.    for i ← 1 to n – 1 do
3.        pivo ← A[i]
4.        j ← i - 1
5.        while j ≥ 0 and A[j] > pivo do
6.            A[j+1] ← A[j]
7.            j ← j - 1
8.        A[j+1] ← pivo
9.    return A
```

Número de execuções baseado somente em n!

Número de execuções baseado na distribuição dos valores no array!!

Insertion Sort

- Qual o pior caso?
  - Vetor ordenado ao contrário! [10, 8, 7, 6, 5, 4, 3, 1]
    - Maior número de execuções do loop da linha 5.

```
InsertionSort(A, n)
1.    pivo ← null
2.    for i ← 1 to n – 1 do
3.        pivo ← A[i]
4.        j ← i - 1
5.        while j ≥ 0 and A[j] > pivo do
6.            A[j+1] ← A[j]
7.            j ← j - 1
8.        A[j+1] ← pivo
9.    return A
```

Quantas vezes é executado?

Insertion Sort

- Qual o pior caso?
  - Vetor ordenado ao contrário! [10, 8, 7, 6, 5, 4, 3, 1]
    - Maior número de execuções do loop da linha 5.

```
InsertionSort(A, n)
1.    pivo ← null
2.    for i ← 1 to n – 1 do
3.        pivo ← A[i]
4.        j ← i - 1
5.        while j ≥ 0 and A[j] > pivo do
6.            A[j+1] ← A[j]
7.            j ← j - 1
8.        A[j+1] ← pivo
9.    return A
```

Quantas vezes é executado? **n - 1**

Insertion Sort

- Qual o pior caso?
  - Vetor ordenado ao contrário! [10, 8, 7, 6, 5, 4, 3, 1]
    - Maior número de execuções do loop da linha 5.

```
InsertionSort(A, n)
1.    pivo ← null
2.    for i ← 1 to n – 1 do
3.        pivo ← A[i]
4.        j ← i - 1
5.        while j ≥ 0 and A[j] > pivo do
6.            A[j+1] ← A[j]
7.            j ← j - 1
8.        A[j+1] ← pivo
9.    return A
```

Quantas vezes é executado?

1 + 2 + 3 + … + (n-1)

Insertion Sort

- Qual o pior caso?
  - Vetor ordenado ao contrário! [10, 8, 7, 6, 5, 4, 3, 1]
    - Maior número de execuções do loop da linha 5.

```
InsertionSort(A, n)
1.    pivo ← null
2.    for i ← 1 to n – 1 do
3.        pivo ← A[i]
4.        j ← i - 1
5.        while j ≥ 0 and A[j] > pivo do
6.            A[j+1] ← A[j]
7.            j ← j - 1
8.        A[j+1] ← pivo
9.    return A
```

Quantas vezes é executado?

$1 + 2 + 3 + \ldots + (n\text{-}1) = \sum_{j=1}^{n-1} j$

Insertion Sort

- Qual o pior caso?
  - Vetor ordenado ao contrário! [10, 8, 7, 6, 5, 4, 3, 1]
    - Maior número de execuções do loop da linha 5.

```
InsertionSort(A, n)
1.    pivo ← null
2.    for i ← 1 to n – 1 do
3.        pivo ← A[i]
4.        j ← i - 1
5.        while j ≥ 0 and A[j] > pivo do
6.            A[j+1] ← A[j]
7.            j ← j - 1
8.        A[j+1] ← pivo
9.    return A
```

Quantas vezes é executado?

$$1 + 2 + 3 + \ldots + (n\text{-}1) = \sum_{j=1}^{n-1} j = \frac{n\,(n-1)}{2}$$

Insertion Sort

- Qual o pior caso?
  - Vetor ordenado ao contrário! [10, 8, 7, 6, 5, 4, 3, 1]
    - Maior número de execuções do loop da linha 5.
      - O algoritmo executa todas as operações de trocas possíveis

```
InsertionSort(A, n)
1.    pivo ← null
2.    for i ← 1 to n – 1 do
3.        pivo ← A[i]
4.        j ← i - 1
5.        while j ≥ 0 and A[j] > pivo do
6.            A[j+1] ← A[j]
7.            j ← j - 1
8.        A[j+1] ← pivo
9.    return A
```

Quantas vezes é executado?

$O(n^2)$

$c = 1, n_0 = 2$

Selection Sort

Selection Sort

- Ideia básica
  - Ordena o vetor da esquerda para a direita
  - A cada iteração o menor elemento ainda não analisado é movido para sua posição final

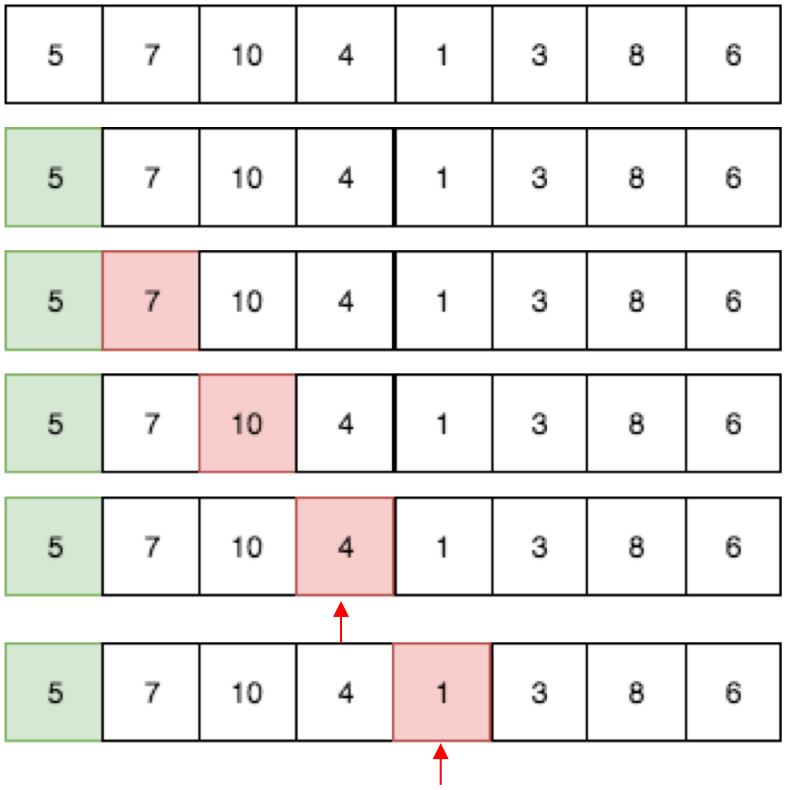
Selection Sort

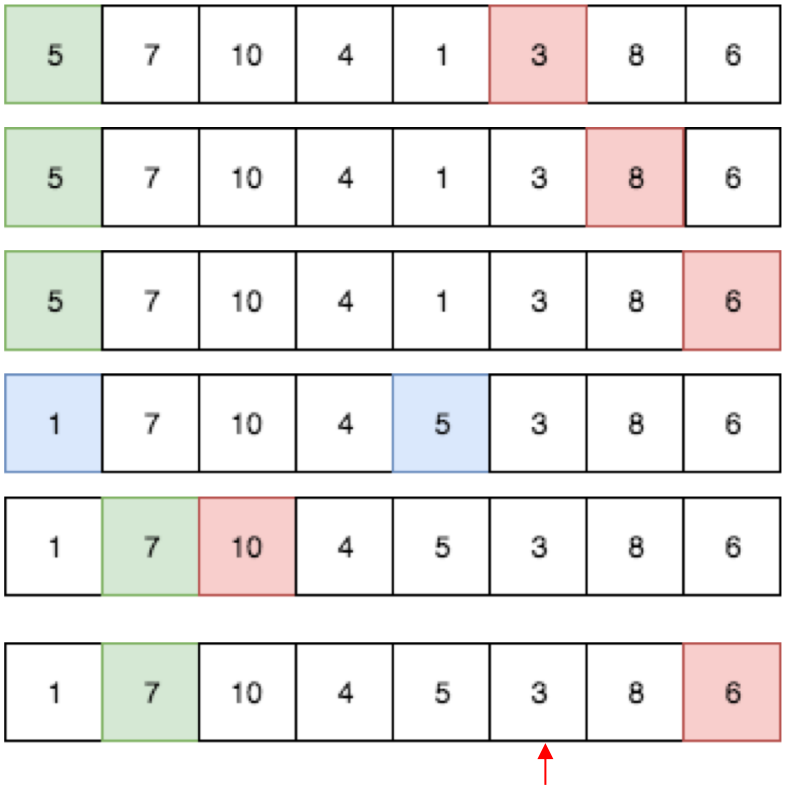
- Psedocódigo

```
SelectionSort(A, n)
1.    for i ← 0 to n – 1 do
2.        i_min ← i
3.        for j ← i+1 to n – 1 do
4.            if A[j] < A[i_min] then
5.                vi_min ← j
6.        if A[i] ≠ A[i_min] then
7.            temp ← A[i]
8.            A[i] ← A[i_min]
9.            A[i_min] ← temp
10.   return A
```

Selection Sort

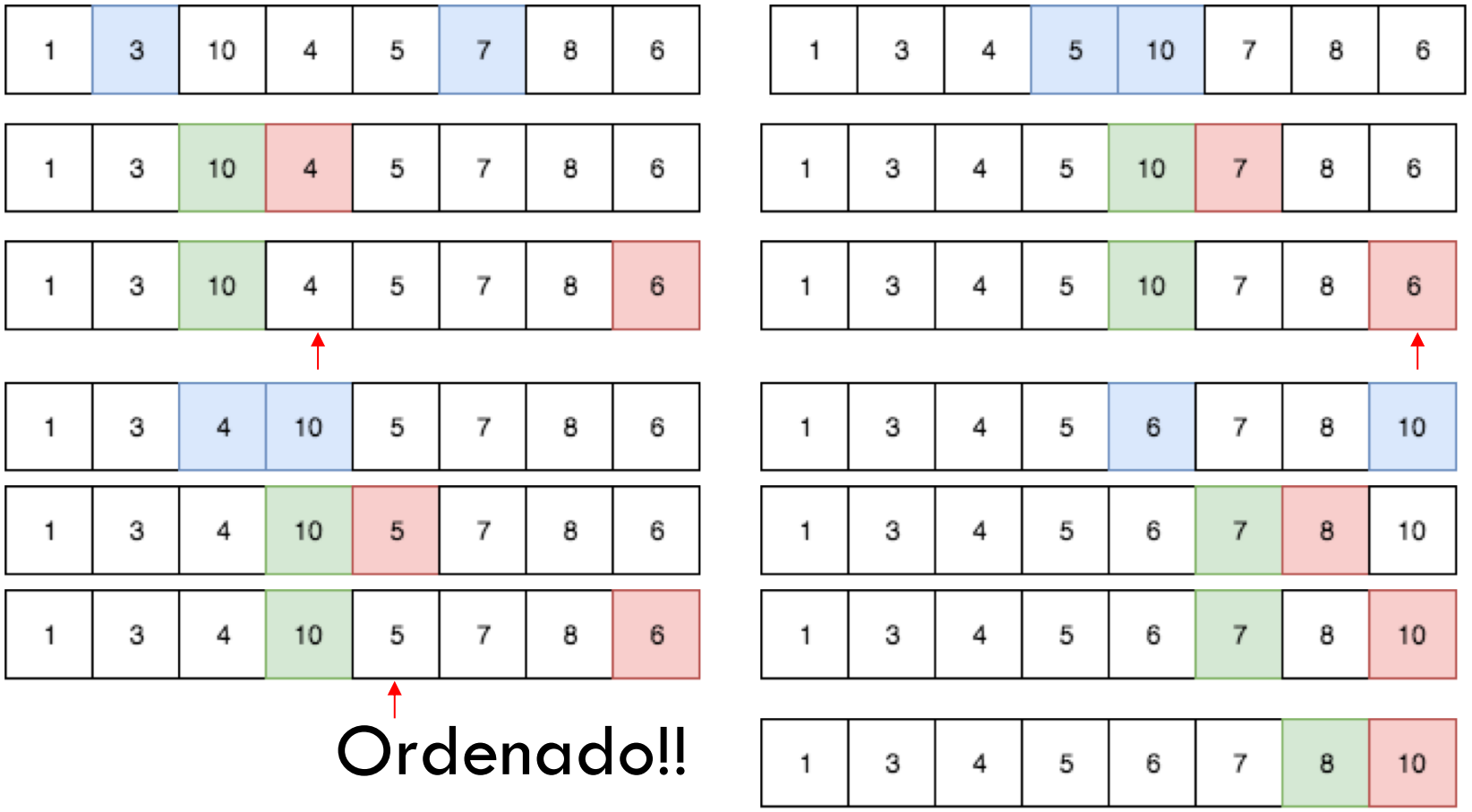
- Exemplo funcionamento





Selection Sort

- Exemplo funcionamento



Selection Sort

- Qual o **melhor** caso?


```
SelectionSort(A, n)
1.    for i ← 0 to n – 1 do
2.        i_min ← i
3.        for j ← i+1 to n – 1 do
4.            if A[j] < A[i_min] then
5.                vi_min ← j
6.        if A[i] ≠ A[i_min] then
7.            temp ← A[i]
8.            A[i] ← A[i_min]
9.            A[i_min] ← temp
10.   return A
```

Número de execuções baseado somente em n!

Número de execuções baseado na distribuição dos valores no array!!

Selection Sort

- Qual o **melhor** caso?
  - Independente do conteúdo do array, os loops das linhas 1. e 3.

```
SelectionSort(A, n)
1.    for i ← 0 to n – 1 do
2.        i_min ← i
3.        for j ← i+1 to n – 1 do
4.            if A[j] < A[i_min] then
5.                vi_min ← j
6.        if A[i] ≠ A[i_min] then
7.            temp ← A[i]
8.            A[i] ← A[i_min]
9.            A[i_min] ← temp
10.   return A
```

Número de execuções baseado somente em n!

Número de execuções baseado na distribuição dos valores no array!!

Selection Sort

- Qual o **melhor** caso?
  - O número de execuções da linha 5. varia de acordo com a distribuição dos elementos do vetor
    - Mas a diferença é somente por um fator constante

```
SelectionSort(A, n)
1.    for i ← 0 to n – 1 do
2.        i_min ← i
3.        for j ← i+1 to n – 1 do
4.            if A[j] < A[i_min] then
5.                vi_min ← j
6.        if A[i] ≠ A[i_min] then
7.            temp ← A[i]
8.            A[i] ← A[i_min]
9.            A[i_min] ← temp
10.   return A
```

Número de execuções baseado somente em n!

Número de execuções baseado na distribuição dos valores no array!!

Selection Sort

- Qual o **melhor** caso?
  - Quantas vezes os loops são executados?

```
SelectionSort(A, n)
1.    for i ← 0 to n – 1 do
2.        i_min ← i
3.        for j ← i+1 to n – 1 do
4.            if A[j] < A[i_min] then
5.                vi_min ← j
6.        if A[i] ≠ A[i_min] then
7.            temp ← A[i]
8.            A[i] ← A[i_min]
9.            A[i_min] ← temp
10.   return A
```

Número de execuções baseado somente em n!

Loop linha 3. = (n-1) + (n-2) + … + 1 =

$$= \frac{(n-1)\,n}{2}$$

$$O(n^2)$$

Selection Sort

- Qual o **pior** caso?
  - Como o número de execuções do loop da linha 3. não depende da distribuição dos elementos do array...

```
SelectionSort(A, n)
1.    for i ← 0 to n – 1 do
2.        i_min ← i
3.        for j ← i+1 to n – 1 do
4.            if A[j] < A[i_min] then
5.                vi_min ← j
6.        if A[i] ≠ A[i_min] then
7.            temp ← A[i]
8.            A[i] ← A[i_min]
9.            A[i_min] ← temp
10.   return A
```

Número de execuções baseado somente em n!

Loop linha 3. = (n-1) + (n-2) + … + 1 =

$$= \frac{(n-1)\,n}{2}$$

$$O(n^2)$$