Estruturas de Dados

Prof. Rodrigo Martins
rodrigo.martins@francomontoro.com.br

Cronograma da Aula

- Ordenação para otimização de busca
  - Importância da Ordenação
  - Eficiência
- Tipos de Ordenação
  - Bubble Sort
- Exemplos
- Exercícios

Ordenação para otimização de busca

- A ordenação é uma técnica muito utilizada para otimizar a busca em estruturas de dados em que os elementos estão armazenados de forma desordenada.

- Isso ocorre porque a ordenação permite que os elementos estejam organizados de forma que a busca seja mais rápida e eficiente.

Ordenação para otimização de busca

- Por exemplo, considere uma lista com milhões de nomes de pessoas.
- Se a lista não estiver ordenada, a busca por um nome específico pode ser muito demorada, pois é necessário percorrer toda a lista para encontrá-lo.
- No entanto, se a lista estiver ordenada em ordem alfabética, é possível usar algoritmos de busca mais eficientes, como a busca binária, que é muito mais rápida e eficiente do que a busca sequencial em listas desordenadas.
- Além disso, a ordenação também pode ser utilizada para identificar rapidamente o maior e o menor valor de um conjunto de dados.
- Isso é útil em muitos problemas, como encontrar o menor e o maior preço de um produto em um catálogo, ou a menor e a maior temperatura em uma série histórica de dados.

Ordenação para otimização de busca

- Existem vários algoritmos de ordenação disponíveis em C++, cada um com suas próprias vantagens e desvantagens em relação à complexidade de tempo e espaço. Alguns dos algoritmos de ordenação mais populares são:

  - Ordenação por seleção
  - Ordenação por inserção
  - Ordenação por bolha
  - Ordenação por merge
  - Ordenação rápida (quicksort)
  - Ordenação por heap (heapsort)

- Cada algoritmo de ordenação tem suas próprias características e é adequado para diferentes tipos de dados e situações.
- A escolha do algoritmo certo dependerá do problema em questão, dos dados a serem ordenados e das restrições de desempenho.

Bubble Sort

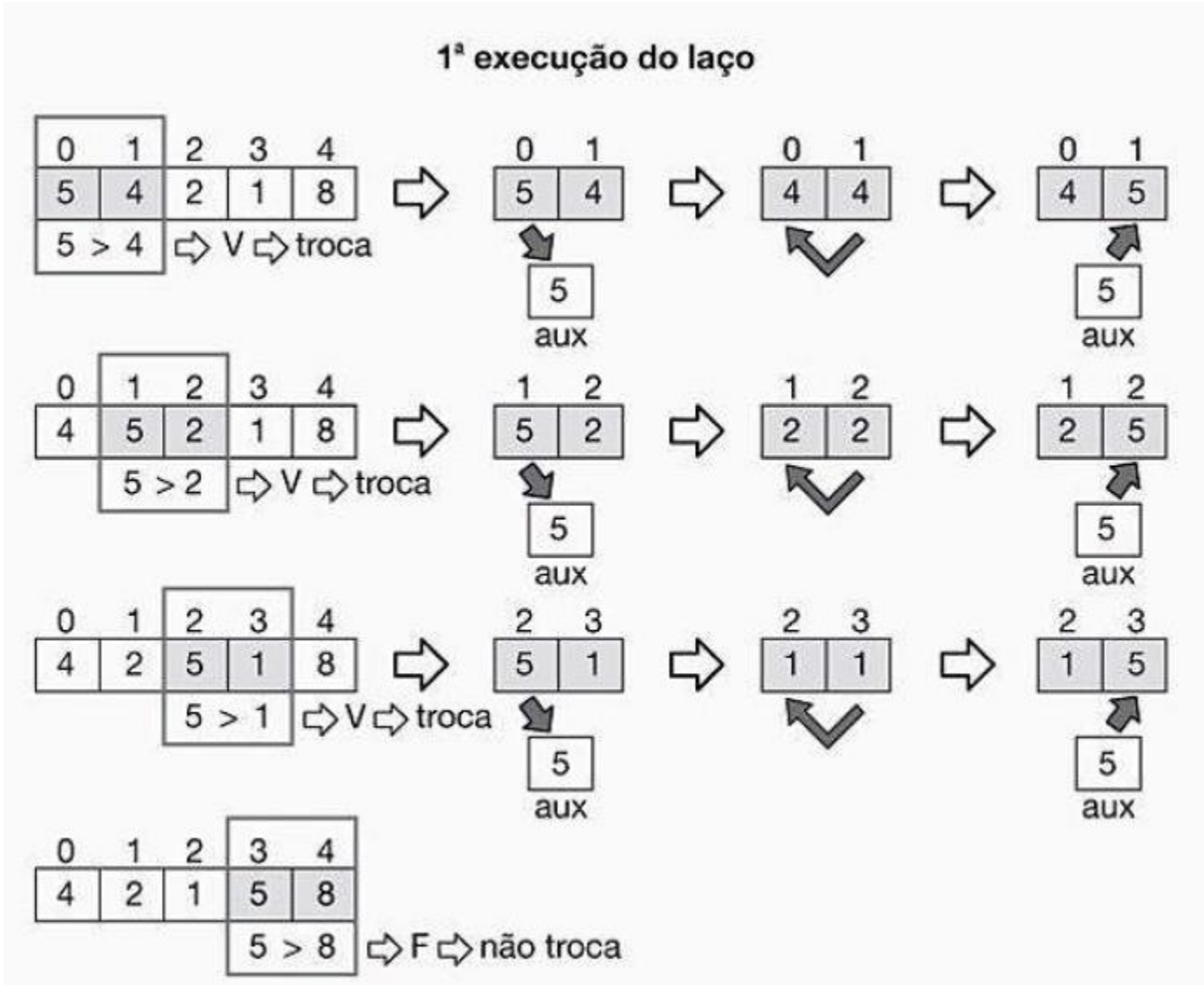
- **Bubble sort** é o algoritmo mais simples, mas o menos eficientes.

- No algoritmo (versão 1) cada elemento da posição i será comparado com o elemento da posição i + 1, ou seja, um elemento da posição 2 será comparado com o elemento da posição 3.

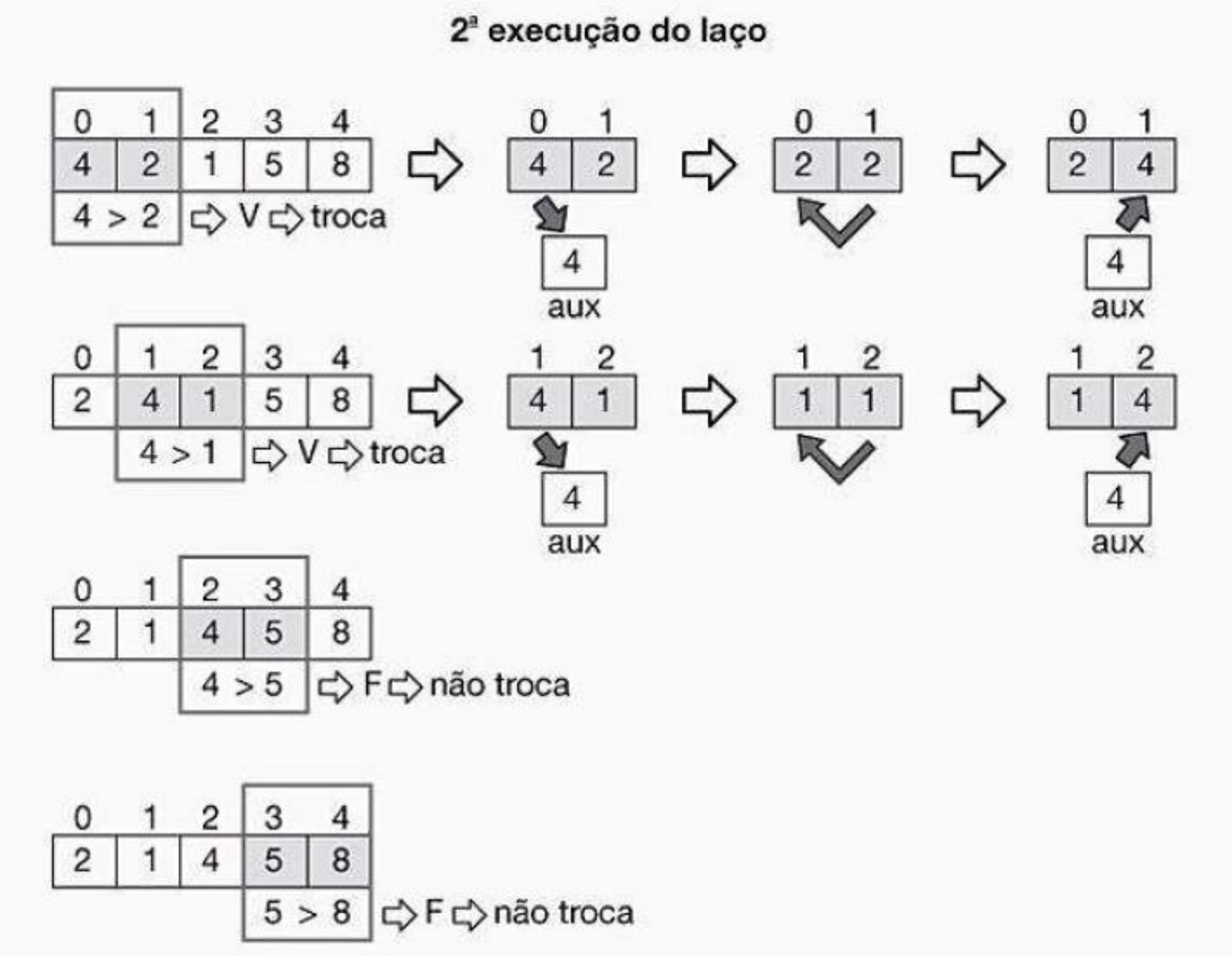
- Caso o elemento da posição 2 for maior que o da posição 3, eles trocam de lugar e assim sucessivamente.

- Por causa dessa forma de execução, o vetor terá que ser percorrido quantas vezes que for necessária, tornando o algoritmo ineficiente para listas muito grandes.

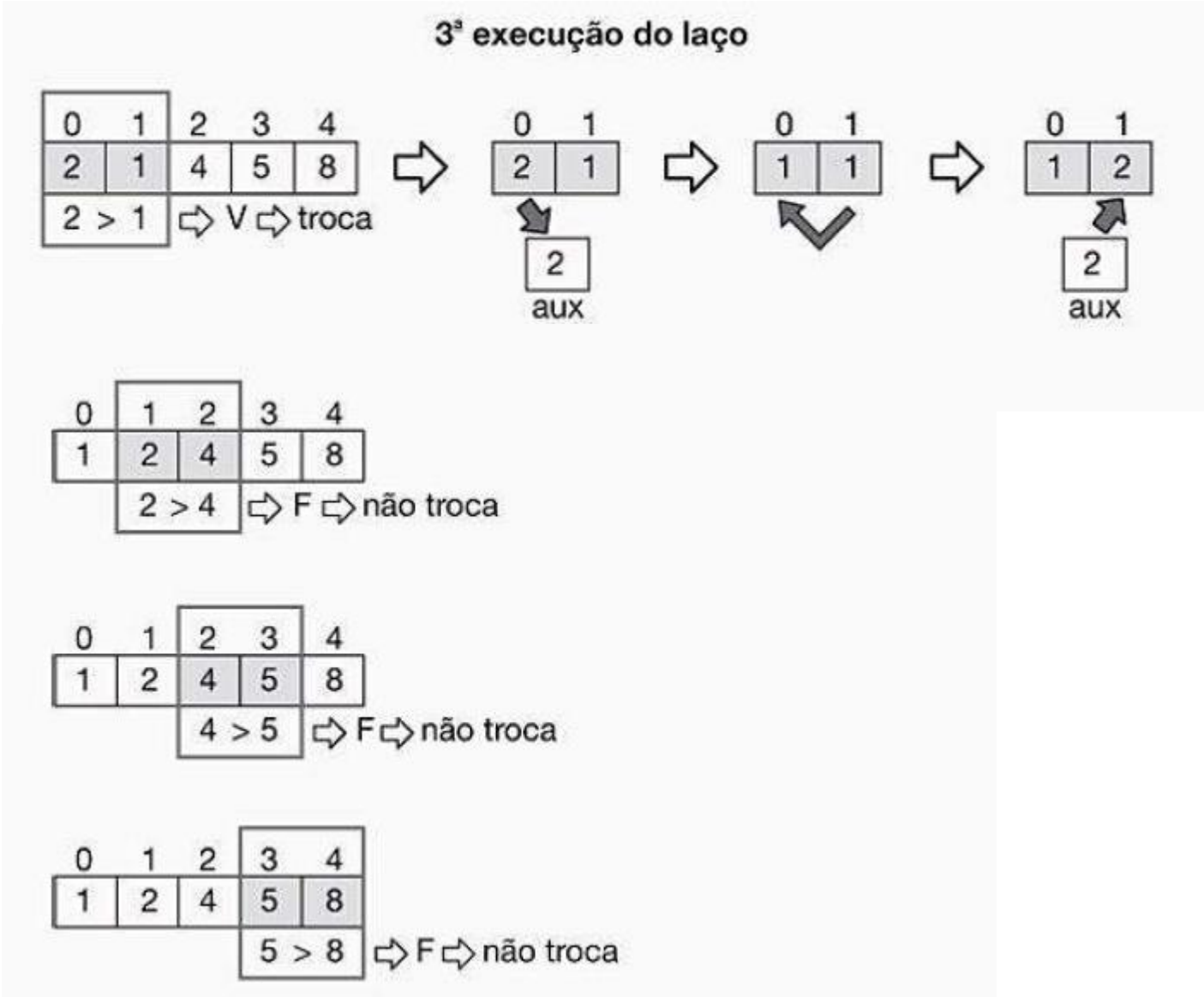
Bubble Sort


1ª execução do laço
0 1 2 3 4
5 4 2 1 8
5 > 4 V troca
5
aux
0 1
5 4
4 4
4 5
0 1 2 3 4
4 5 2 1 8
5 > 2 V troca
1 2
5 2
2 2
2 5
0 1 2 3 4
4 2 5 1 8
5 > 1 V troca
2 3
5 1
1 1
1 5
0 1 2 3 4
4 2 1 5 8
5 > 8 F não troca

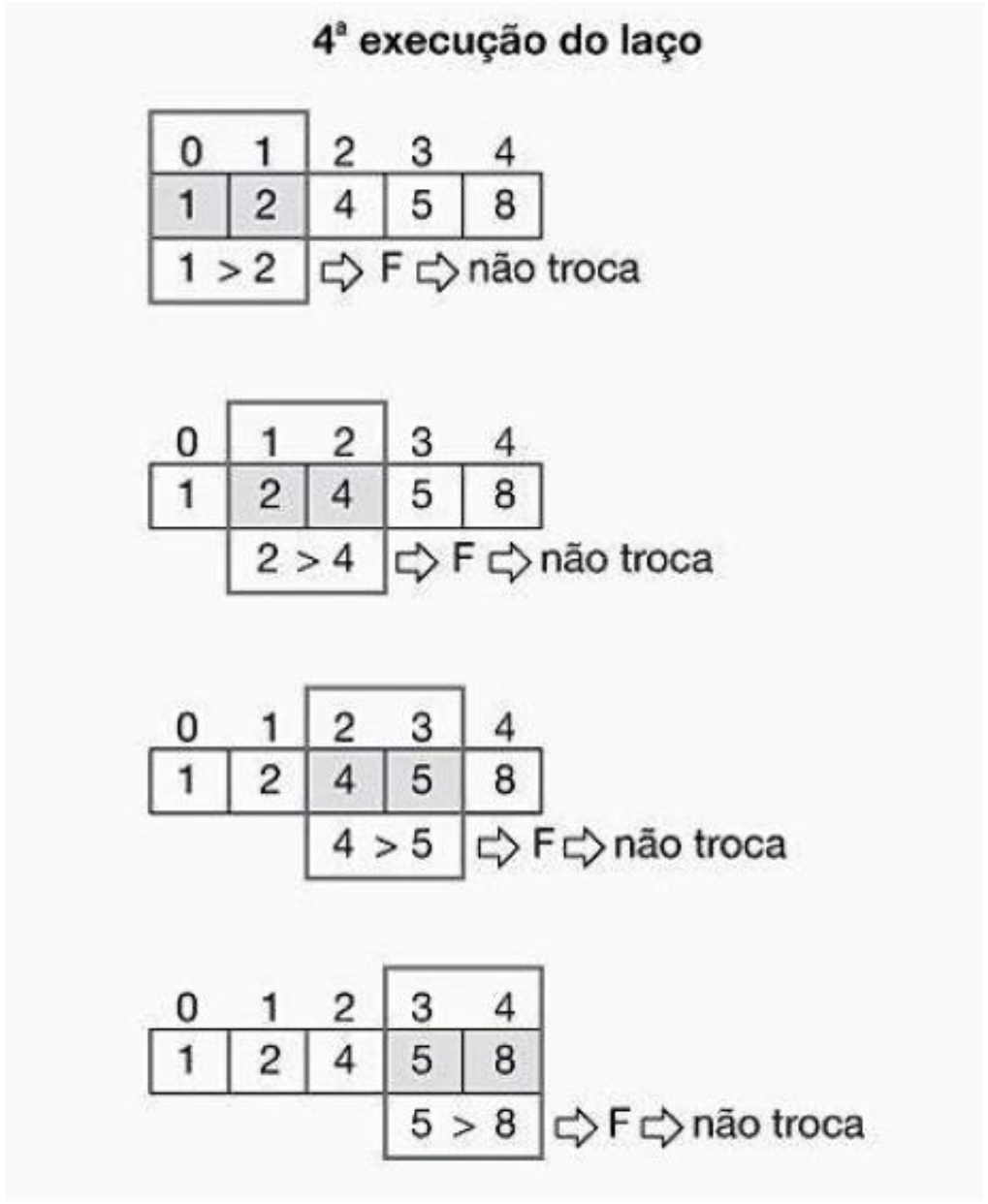
Bubble Sort


2ª execução do laço
0 1 2 3 4
4 2 1 5 8
4 > 2
V
troca
4
aux
0 1
4 2
0 1
2 2
0 1
2 4
4
aux
0 1 2 3 4
2 4 1 5 8
4 > 1
V
troca
1 2
4 1
1 2
1 1
1 2
1 4
4
aux
0 1 2 3 4
2 1 4 5 8
4 > 5
F
não troca
0 1 2 3 4
2 1 4 5 8
5 > 8
F
não troca

Bubble Sort


3ª execução do laço
0 1 2 3 4
2 1 4 5 8
2 > 1
V
troca
0 1
2 1
2
aux
0 1
1 1
0 1
1 2
2
aux
0 1 2 3 4
1 2 4 5 8
2 > 4
F
não troca
0 1 2 3 4
1 2 4 5 8
4 > 5
F
não troca
0 1 2 3 4
1 2 4 5 8
5 > 8
F
não troca

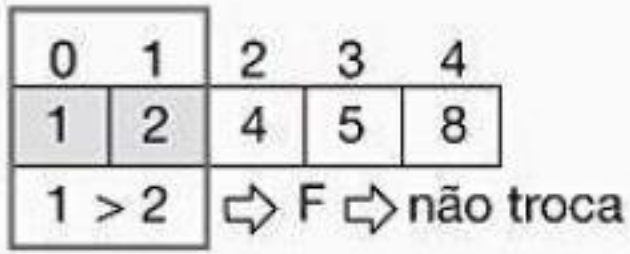
Bubble Sort

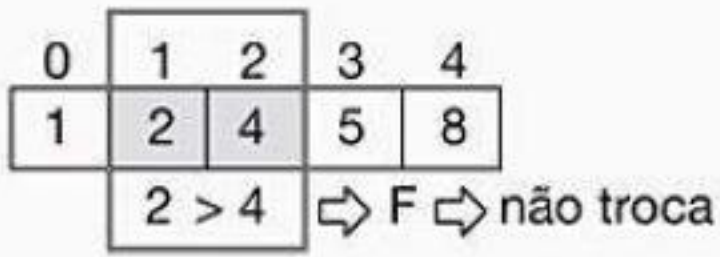

4ª execução do laço
0 1 2 3 4
1 2 4 5 8
1 > 2 ⇨ F ⇨ não troca
0 1 2 3 4
1 2 4 5 8
2 > 4 ⇨ F ⇨ não troca
0 1 2 3 4
1 2 4 5 8
4 > 5 ⇨ F ⇨ não troca
0 1 2 3 4
1 2 4 5 8
5 > 8 ⇨ F ⇨ não troca

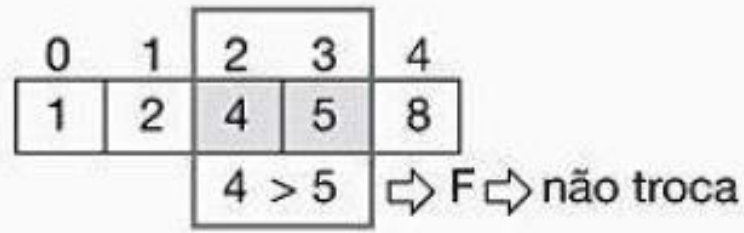
Bubble Sort

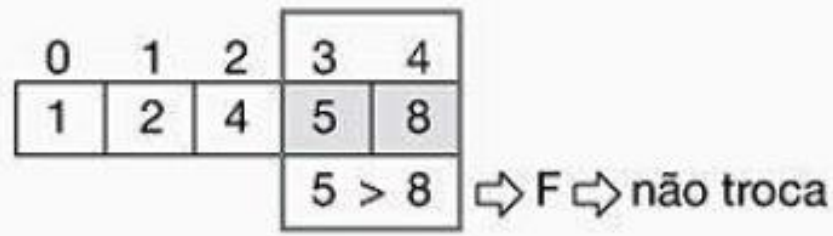
5ª execução do laço

Apesar de o vetor já estar ordenado, mais uma execução do laço será realizada.









Bubble Sort – Versão 1

bubbleSort.cpp

```
1  #include <iostream>
2  #include <stdlib.h>
3  #include <string>
4
5  #define TAM 10
6
7  using namespace std;
8
9  void imprimeVetor(int vetor[]){
10     int i;
11     cout << endl;
12     for (i = 0; i < TAM; i++){
13         cout << " |" << vetor[i] << "| ";
14     }
15 }
16
```

```
17  void bubbleSort(int vetor[TAM]){
18      int x, y, aux;
19
20      // valor da esquerda sendo analizado
21      for(x = 0; x < TAM; x++){
22          //valor da direita sendo analizado
23          for (y = x + 1; y < TAM; y++){
24              //análise se é necessário a troca
25              if (vetor[x] > vetor[y]){
26                  aux = vetor[x];
27                  vetor[x] = vetor[y];
28                  vetor[y] = aux;
29              }
30          }
31      }
32  }
33
34  int main(){
35
36      int vetor[TAM] = {10,9,8,7,6,5,4,3,2,1};
37
38      imprimeVetor(vetor);
39      cout << endl;
40
41      bubbleSort(vetor);
42      imprimeVetor(vetor);
43
44  }
```

Bubble Sort – Versão 1

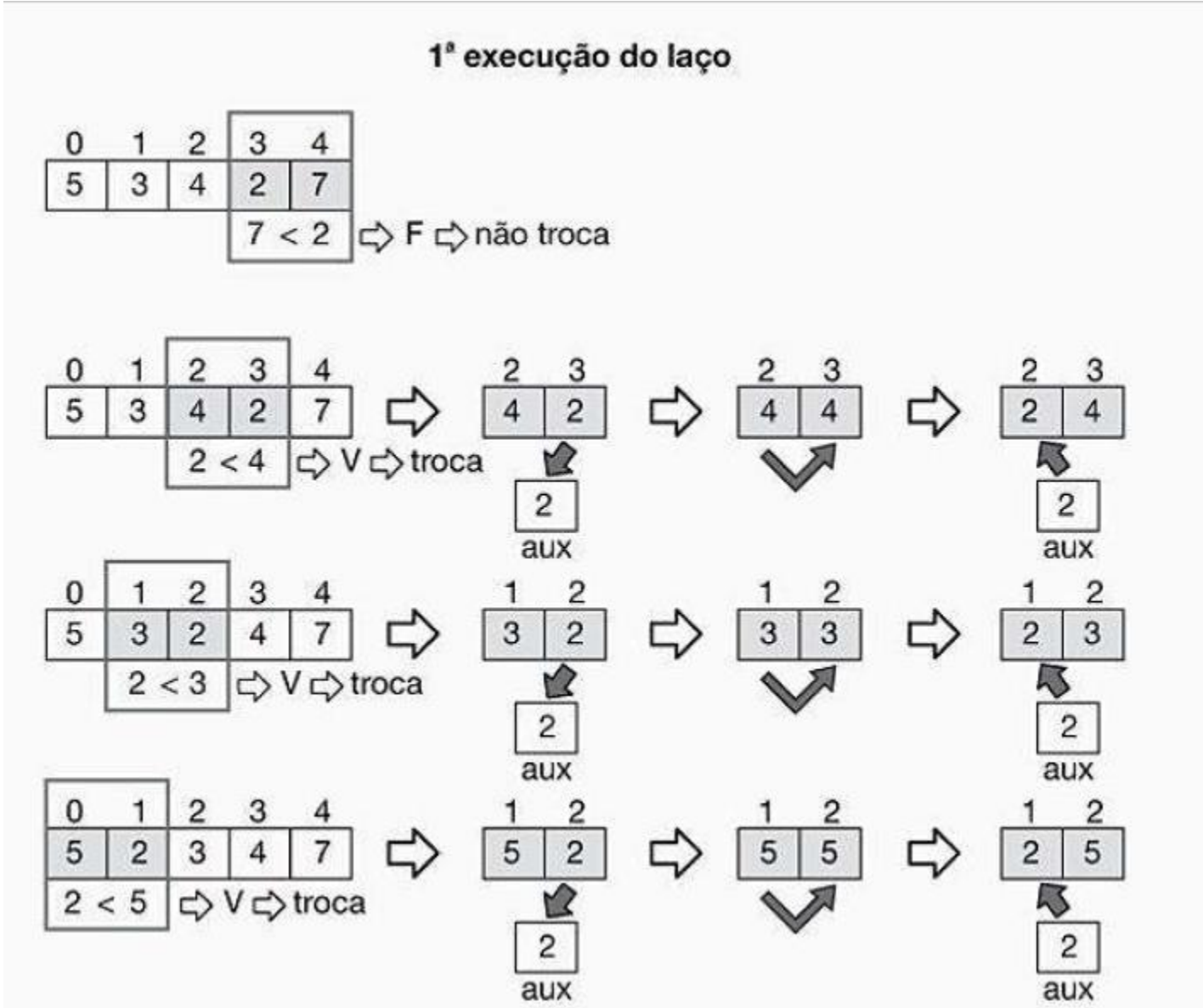
- Em alguns algoritmos de ordenação o fator relevante que determina seu tempo de execução é o número de comparações utilizadas.
- Considerando que o algoritmo seja implementado para um vetor com 5 posições, por exemplo, verifica-se que o número de iterações do primeiro laço é 5.
- O segundo laço possui 4 iterações, mas como está interno ao primeiro será executado 20 vezes (5 x 4).
- Neste algoritmo não há situações melhores ou piores.
- Qualquer que seja o vetor de entrada, o algoritmo se comportará da mesma maneira, realizando todas as comparações, mesmo que desnecessárias.

Bubble Sort - 2ª Versão

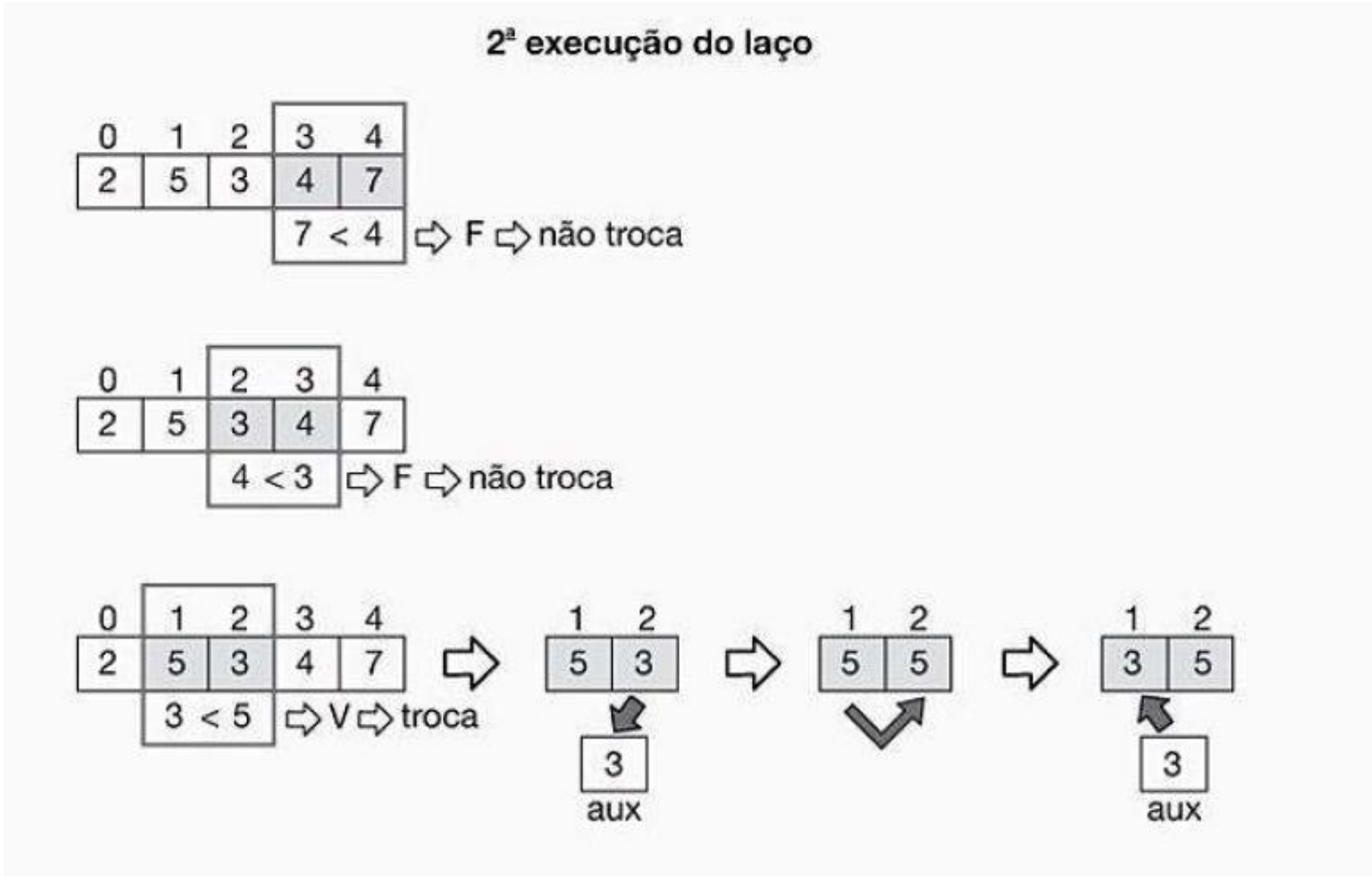
- Neste algoritmo de ordenação serão efetuadas comparações entre os dados armazenados em um vetor de tamanho n.

- Cada elemento de posição i será comparado com o de posição i – 1, e quando a ordenação procurada (crescente ou decrescente) é encontrada, uma troca de posições entre os elementos é feita.

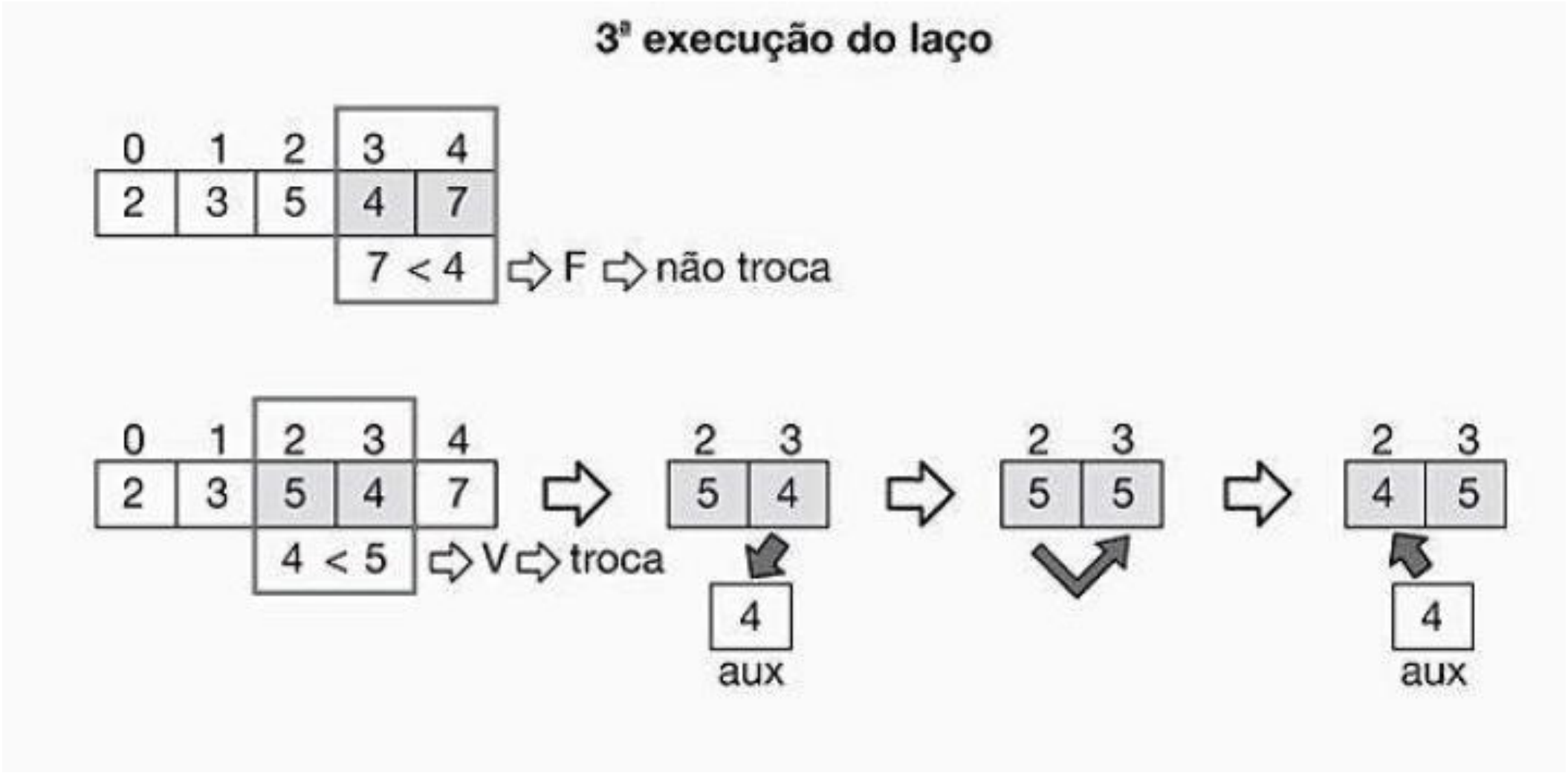
Bubble Sort - 2ª Versão



Bubble Sort - 2ª Versão



Bubble Sort - 2ª Versão



Bubble Sort - 2ª Versão

4ª execução do laço

Apesar de o vetor já estar ordenado, mais uma execução do laço será realizada.

0	1	2	3	4
2	3	4	5	7

7 < 5 ⇨ F ⇨ não troca

Bubble Sort - 2ª Versão

*bubbleSort2.cpp

```
1  #include <iostream>
2
3  using namespace std;
4
5  void imprimeVetor(int x[])
6  {
7
8      for (int i = 0; i <= 4; i++)
9      {
10         cout << i + 1 << " numero: " << x[i] << endl;
11     }
12
13 }
14
```

Bubble Sort - 2ª Versão

```
15  void bubbleSort2(int x[])
16  {
17      int aux = 0;
18      //ordenando de forma crescente
19      //laço com a quantidade de elementos do vetor - 1
20      for (int j = 1; j <= 4; j++)
21      {
22          //laço que percorre da última posição
23          //até posição j do vetor
24          for (int i = 4; i >= j; i--)
25          {
26              if (x[i] < x[i - 1])
27              {
28                  aux = x[i];
29                  x[i] = x[i - 1];
30                  x[i - 1] = aux;
31              }
32          }
33      }
34
35  }
36
```

Bubble Sort - 2ª Versão

```
36
37  int main(int argc, char** argv)
38  {
39      int x[5];
40
41      //carregando os números no vetor
42      for (int i = 0; i <= 4; i++)
43      {
44          cout << "Digite o numero" << endl;
45          cin >> x[i];
46      }
47
48      bubbleSort2(x);
49      imprimeVetor(x);
50
51      return 0;
52  }
53
```

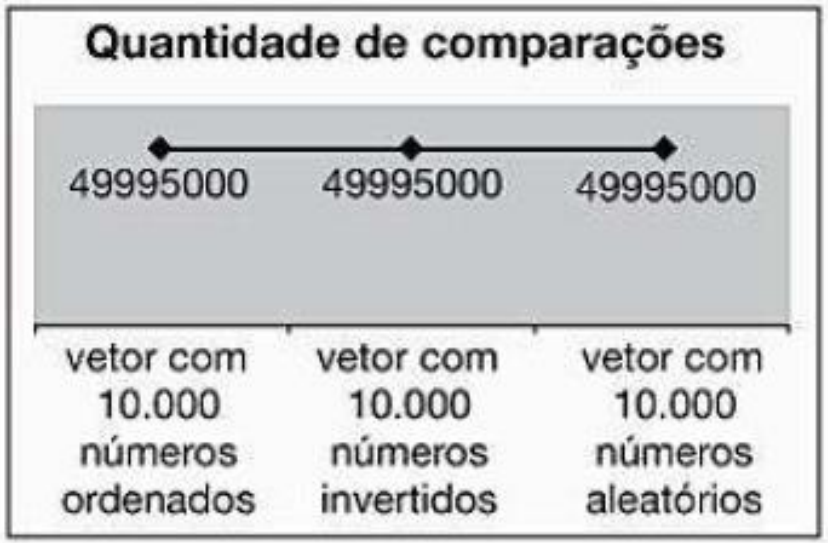
Bubble Sort - 2ª Versão

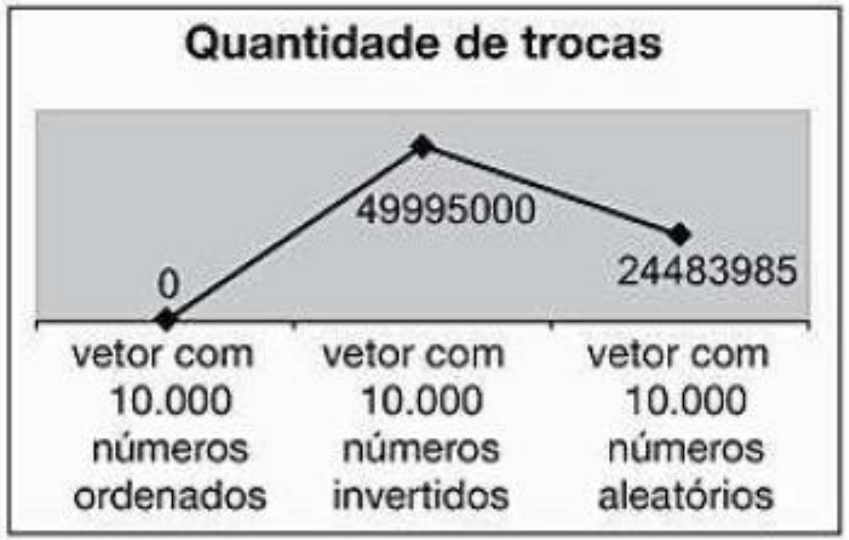
- Nesta versão, para um vetor de tamanho 5, por exemplo, o número de comparações realizadas, que é interna será (4 + 3 + 2 + 1) = 10.

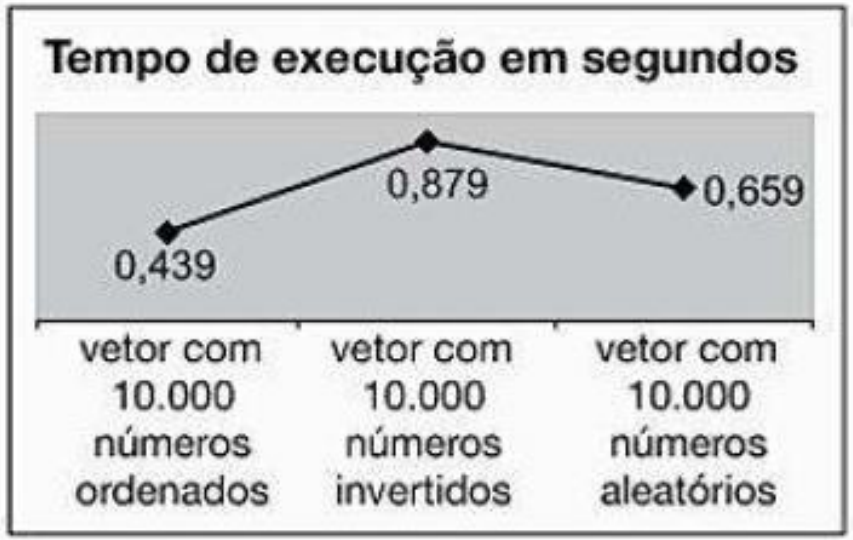
- Para qualquer vetor de entrada, o algoritmo se comporta da mesma maneira.

- No próximo slide é apresentado o gráfico de desempenho dessa versão.

Bubble Sort - 2ª Versão





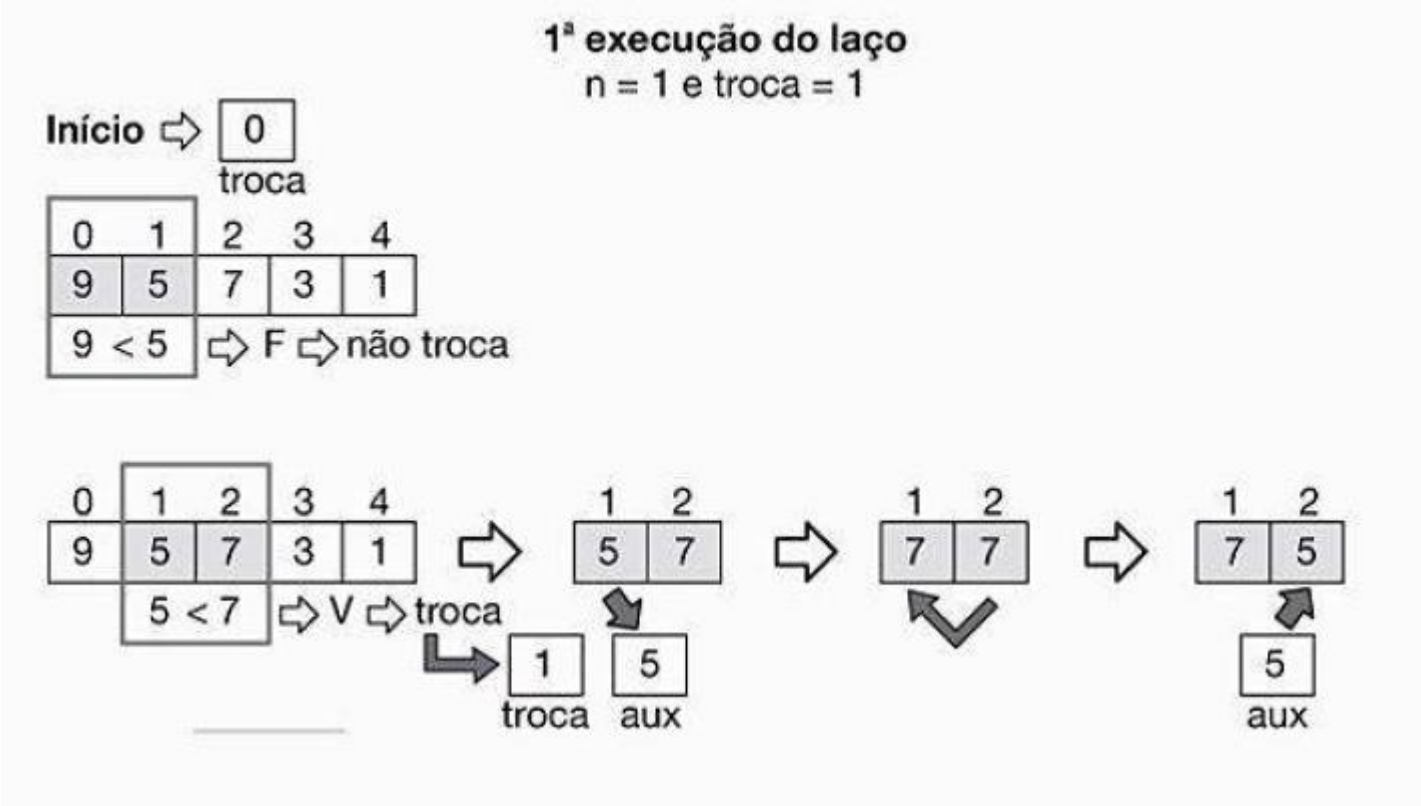


Bubble Sort - 3ª Versão

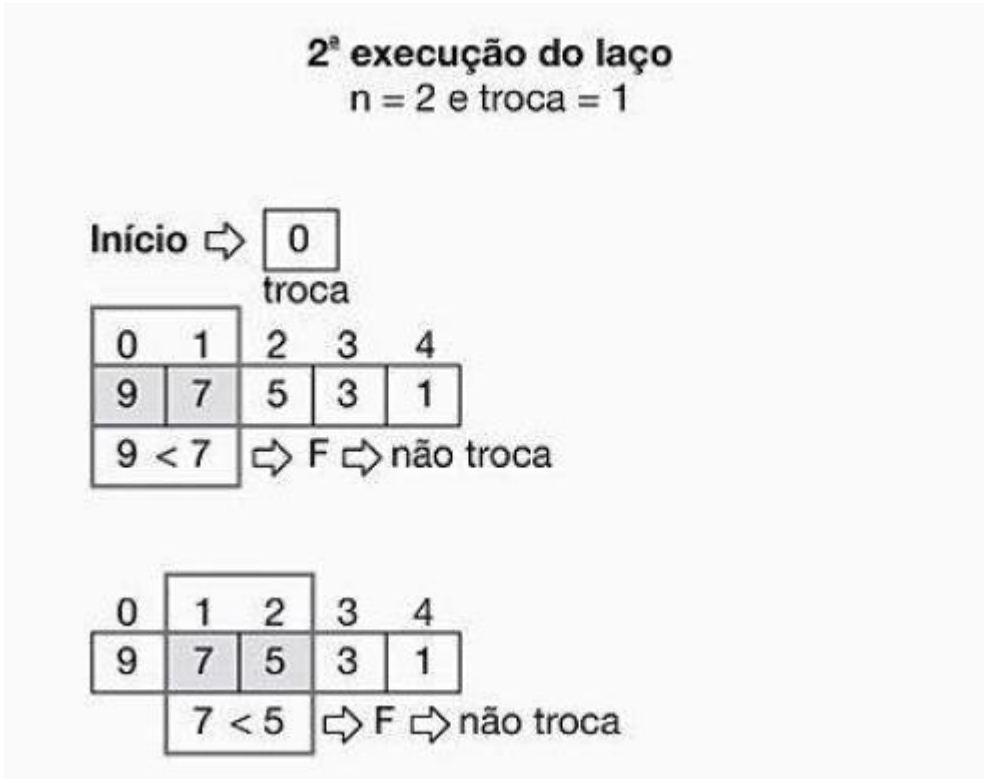
- Nesta versão, serão efetuadas comparações entre os dados armazenados em um vetor de tamanho n.

- Cada elemento de posição i será comparado com o deposição i + 1 e, quando a ordenação que se busca (crescente ou decrescente) é encontrada, uma troca de posições entre os dados é feita.

Bubble Sort - 3ª Versão

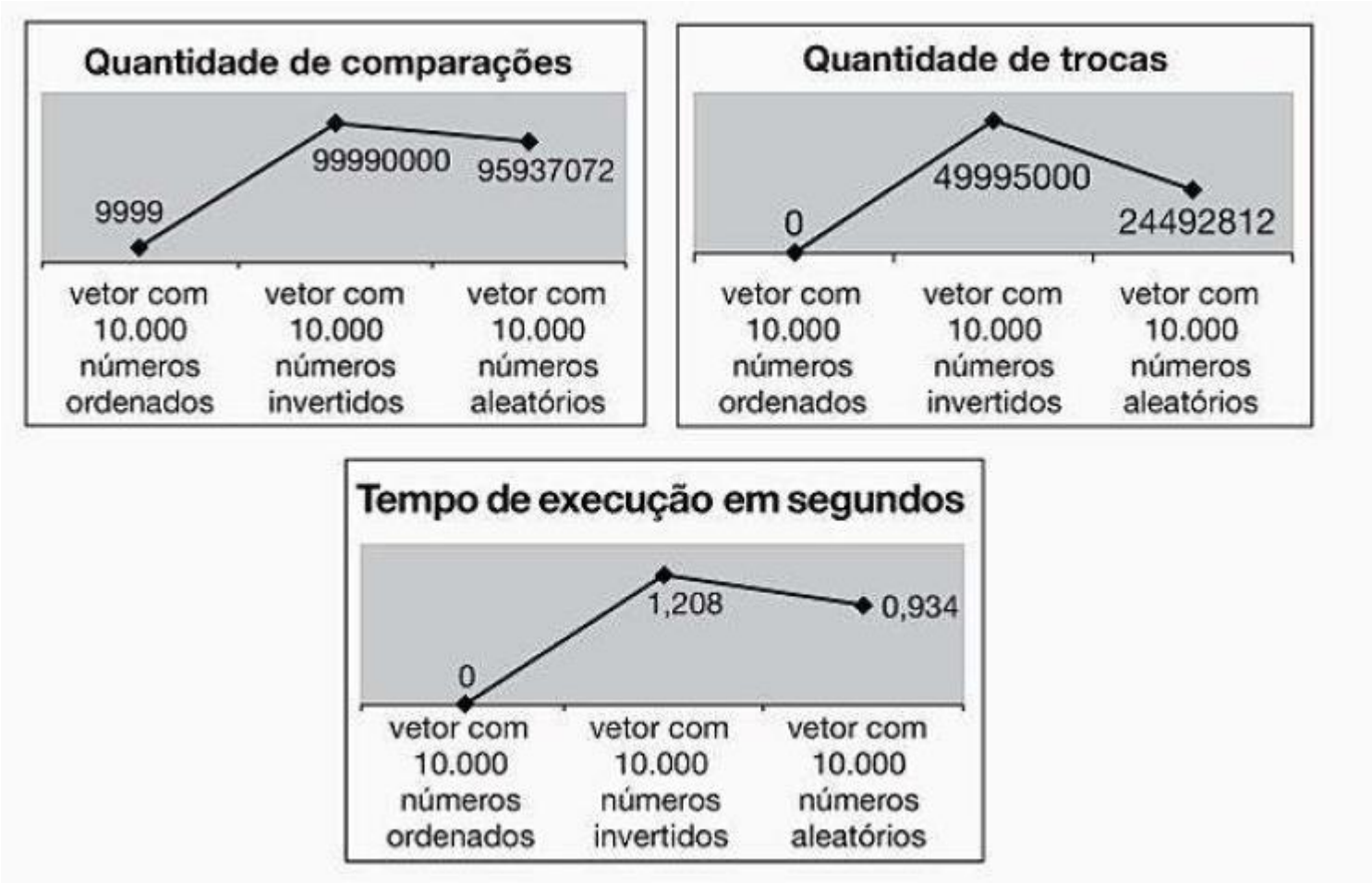


Bubble Sort - 3ª Versão



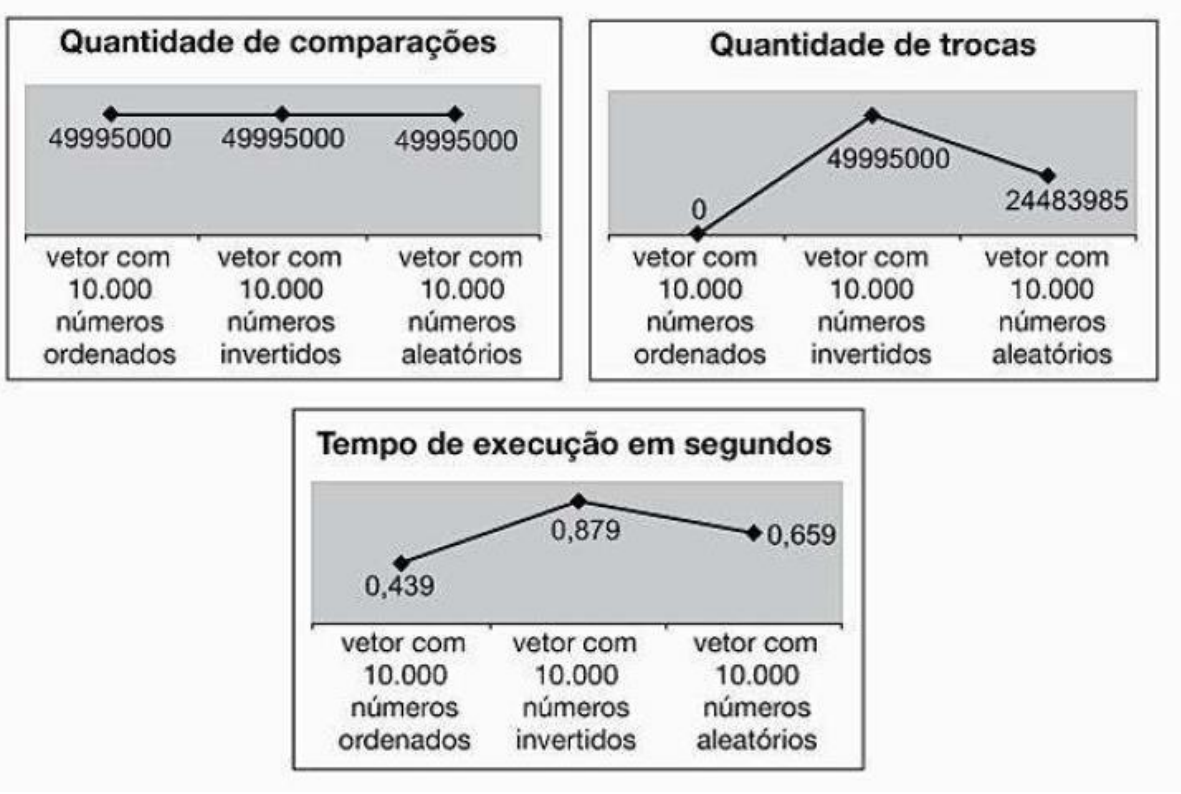
Bubble Sort - 3ª Versão

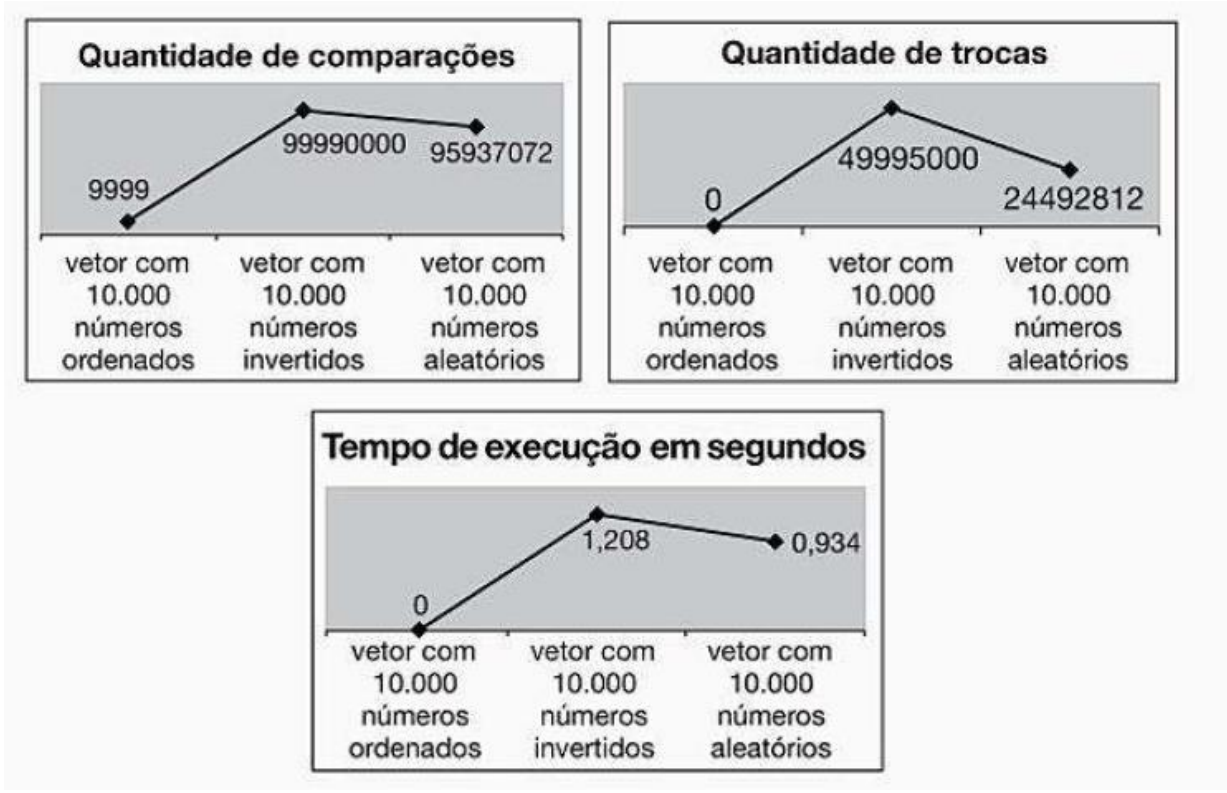
- Gráfico de desempenho da 3ª Versão



Bubble Sort

- Comparativo 2ª e da 3ª Versão





Exercícios - Bubble Sort - 3ª Versão

- Desenvolva você mesmo a 3ª Versão do algoritmo de ordenação Bubble Sort, com o objetivo de ordenar os dados de forma decrescente.

Referência desta aula

- https://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261
- https://www.devmedia.com.br/algoritmos-de-ordenacao/2622
- https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao/
- http://www.cplusplus.com/reference/
- Ascencio, A. F. G. (2010). Estruturas De Dados: ALGORITMOS, ANÁLISE DA COMPLEXIDADE E IMPLEMENTAÇÃO. Brasil: PEARSON BRASIL.

Obrigado

Rodrigo