INSTITUTO FEDERAL
DE EDUCAÇÃO, CIÊNCIA E TECNOLOGIA
São Paulo

Estrutura de Dados

Aula 5

GUSTAVO FORTUNATO PUGA

Retomando a Lista

- Forma simples de interligar os elementos de um conjunto.
- Agrupa informações referentes a um conjunto de elementos que se relacionam entre si de alguma forma.
- São úteis em aplicações tais como manipulação simbólica, gerência de memória, simulação e compiladores.
- Inúmeros tipos de dados podem ser representados por listas. Alguns exemplos de sistemas de informação são: informações sobre os funcionários de uma empresa, notas de alunos, itens de estoque, etc.

Retomando a Lista

- Estrutura em que as **operações inserir, retirar e localizar** são definidas.
- Itens da lista podem ser **acessados, inseridos ou retirados**.
- Podem **crescer ou diminuir** de tamanho durante a execução de um programa, de acordo com a demanda.
- Duas listas podem ser **concatenadas** para formar uma lista única, ou uma pode ser **partida** em duas ou mais listas.
- Podem ser adequadas quando não é possível prever a demanda por memória, permitindo a manipulação de quantidades imprevisíveis de dados, de formato também imprevisível.

Retomando a Lista

1) Adicionar um dado elemento no fim da Lista.

2) Adicionar um dado elemento em um dada posição.

3) Pegar o elemento de uma dada posição.

4) Remover o elemento de uma dada posição.

5) Verificar se um dado elemento está contido na Lista.

6) Informar a quantidade de elementos da Lista.

7) Verificar se a lista esta vazia

Generics

- Generics é uma funcionalidade incorporada ao Java a partir da versão 5.0
- Permite aos programadores escreverem métodos genéricos
  - Os parâmetros dos métodos, variáveis locais e o tipo de retorno podem ser definidos na chamada do método
  - Permite ao mesmo método ser invocado usando-se tipos distintos (sem precisar sobrescrevê-lo)
- Permite também a definição de classes genéricas
  - Os atributos da classe podem ser definidos no momento da instanciação do objeto
  - Recurso útil ao definir classes como estruturas de dados
- Generics em Java oferece os mesmos recursos dos Templates em C++

Generics

```
class Teste
{
    public static void imprimeVetor(double v[]){
        for ( double e : v ) System.out.printf(e + " ");
        System.out.println();
    }

    public static void main(String args[])
    {
        double[] arrayDouble = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6};
        System.out.println("Vetor de double: ");
        imprimeVetor(arrayDouble);

        int[] arrayInt = {1, 2, 3, 4, 5, 6};
        System.out.println("Vetor de inteiros: ");
        imprimeVetor(arrayInt);
    }
}
```

- Ao tentar compilar o código um erro é apresentado
- Embora o tipo double contenha o tipo int, uma referência para double nao pode referenciar um vetor de int
- Seria necessário ter 2 implementações de imprimeVetor

Generics

- **O problema se agravaria**, à medida em que fosse **necessário imprimir vetores de outros tipos de dados**

  - Para cada tipo, uma nova implementação
  - Cada implementação teria apenas o cabeçalho do método (e o tipo usado no for) diferente das demais

Generics

```
class Teste
{
    public static void imprimeVetor(double v[]){
        for ( double e : v ) System.out.printf(e + " ");
        System.out.println();
    }

    public static void main(String args[])
    {
        double[] arrayDouble = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6};
        System.out.println("Vetor de double: ");
        imprimeVetor(arrayDouble);

        int[] arrayInt = {1, 2, 3, 4, 5, 6};
        System.out.println("Vetor de inteiros: ");
        imprimeVetor(arrayInt);

        char[] arrayChar = {'E', 'C', 'O', '0', '3', '0'};
        System.out.println("Vetor de char: ");
        imprimeVetor(arrayChar);
    }
}
```

Generics

Este problema seria facilmente resolvido ao implementar um método genérico

```
class Teste
{
    public static < T > void imprimeVetor(T v[]){
        for ( T e : v ) System.out.printf(e + " ");
        System.out.println();
    }

    public static void main(String args[])
    {
        Double[] arrayDouble = {1.1, 2.2, 3.3, 4.4, 5.5, 6.6};
        System.out.println("Vetor de double: ");
        imprimeVetor(arrayDouble);

        Integer[] arrayInt = {1, 2, 3, 4, 5, 6};
        System.out.println("Vetor de inteiros: ");
        imprimeVetor(arrayInt);

        Character[] arrayChar = {'E', 'C', 'O', '0', '3', '0'};
        System.out.println("Vetor de char: ");
        imprimeVetor(arrayChar);
    }
}
```

- Define-se um ou mais tipos genéricos, que serão delimitados pelos símbolos < >
- Estes tipos são definidos na chamada do método

Generics

- Uma ressalva: métodos genéricos (e classes genéricas) podem ser definidos apenas para tipos referenciáveis

- Logo, não podem ser definidos para tipos primitivos {**byte, short, int, long, float, double, boolean, char**}

- Essa limitação é contornada usando-se as **classes empacotadoras de tipo**, que são uma alternativa oferecida por Java para tratar tipos primitivos como referenciáveis de forma transparente
  - {**Byte, Short, Integer, Long, Float, Double, Boolean, Character**}

Generics

Mas qual seria a diferença entre usar um método genérico e um método usando Object?

```
class Teste
{
    public static void imprimeVetor(Object v[]){
        for ( Object e : v ) System.out.printf(e + " ");
        System.out.println();
    }

    public static void main(String args[])
    {
        Double[] arrayDouble = {0.0, 6.6, 9.9, 4.4, 5.5, 2.2};
        System.out.println("Vetor de double: ");
        imprimeVetor(arrayDouble);

        Integer[] arrayInt = {10, 7, 13, 4, 9, 6};
        System.out.println("Vetor de inteiro: ");
        imprimeVetor(arrayInt);

        Character[] arrayChar = {'E', 'C', 'O', '0', '3', '0'};
        System.out.println("Vetor de char: ");
        imprimeVetor(arrayChar);
    }
}
```

Generics

Para este caso em específico, a diferença é NENHUMA!

```
class Teste
{
    public static void imprimeVetor(Object v[]){
        for ( Object e : v ) System.out.printf(e + " ");
        System.out.println();
    }

    public static void main(String args[])
    {
        Double[] arrayDouble = {0.0, 6.6, 9.9, 4.4, 5.5, 2.2};
        System.out.println("Vetor de double: ");
        imprimeVetor(arrayDouble);

        Integer[] arrayInt = {10, 7, 13, 4, 9, 6};
        System.out.println("Vetor de inteiro: ");
        imprimeVetor(arrayInt);

        Character[] arrayChar = {'E', 'C', 'O', '0', '3', '0'};
        System.out.println("Vetor de char: ");
        imprimeVetor(arrayChar);
    }
}
```

Generics

- Java implementa métodos e classes genéricas da seguinte forma
  - substitui os tipos genéricos por tipos que sejam de uma superclasse contenha todas as candidatas a serem usadas
  - realiza o casting de tipos (quando necessário) implicitamente
  - esse processo é conhecido como **erasure**
- No exemplo anterior, o tipo **Object** seria usado pois é o único tipo capaz de referenciar qualquer objeto que invoque o método

Generics

Transforme a lista em genérica

Generics

```
public class MinhaClasse<T> {
   T obj;
   public MinhaClasse(T obj) {
      this.obj = obj;
   }
   public void printar() {
      System.out.println(obj);
   }
}
```

Ordenação

Classificar dados, em ciência da computação, trata-se de colocar os elementos de uma dada sequência em uma certa ordem.

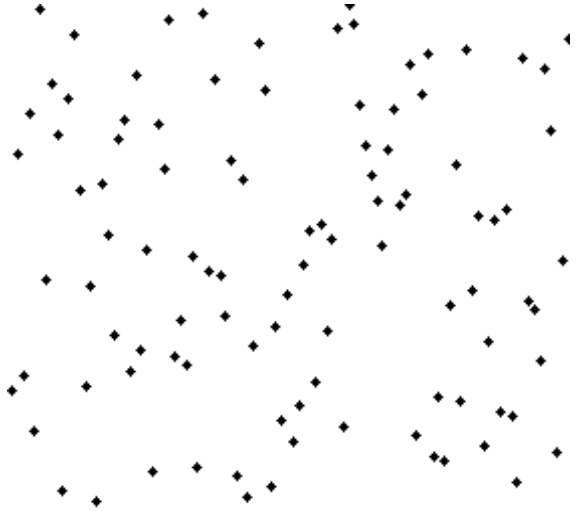
Pode-se ordenar:

- Em ordem crescente
- Em ordem decrescente

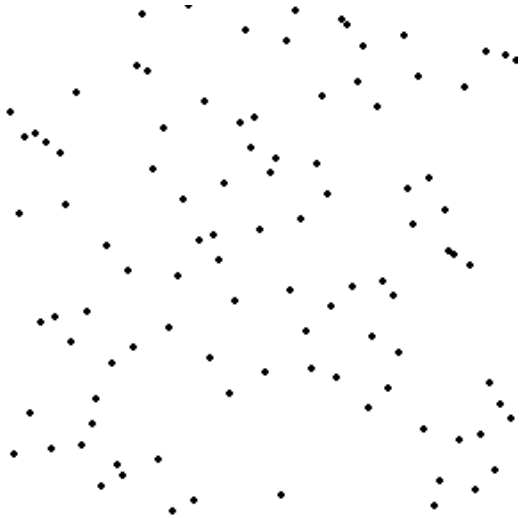
Ordenação

Existem vários algoritmos que realizam esta classificação:

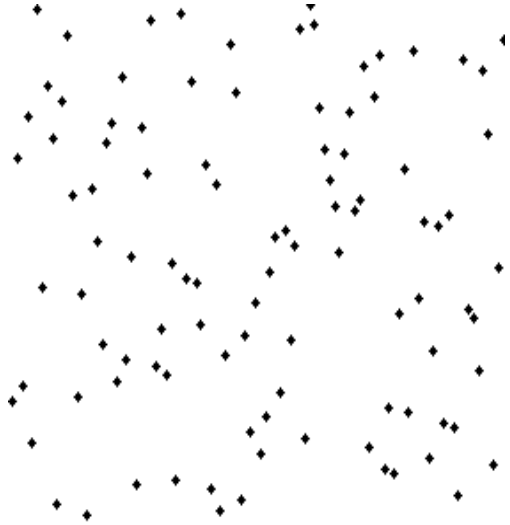
- *Bubble sort*



- *Selection sort*



- *Insertion sort*



Bubble sort

- O bubble sort, ou ordenação por flutuação, trata-se do algoritmo de ordenação dos mais simples.

- Nele se percorre o vetor diversas vezes, e, a cada passagem, faz o maior elemento da sequência se deslocar (“flutuar”) para o “fundo” do vetor.

- Essa movimentação lembra a forma como as bolhas em um tanque de água procuram seu próprio nível, e disso vem o nome do algoritmo.

Bubble sort

6 5 3 1 8 7 2 4

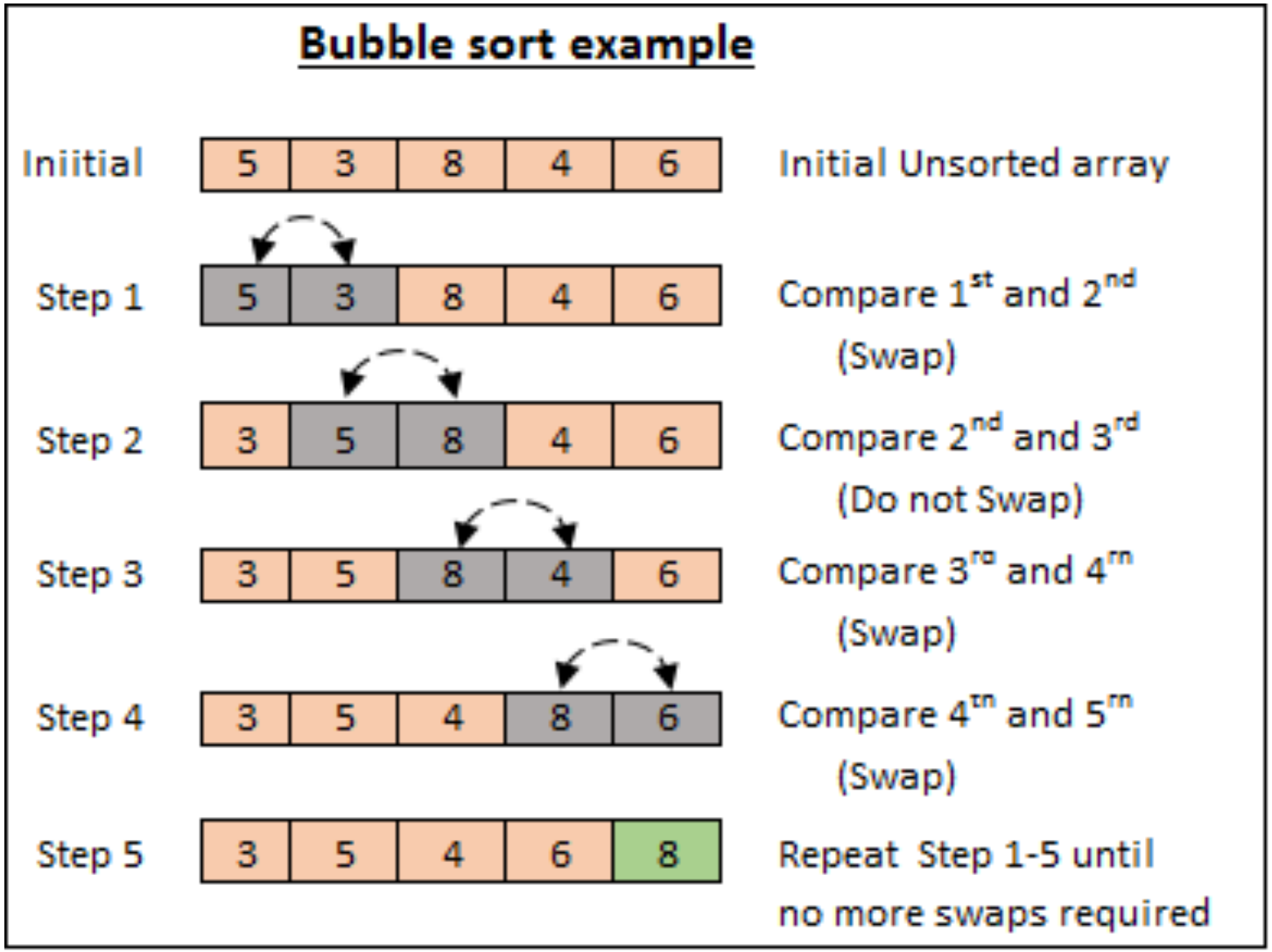
Funcionamento

1ª Etapa: O vetor é percorrido comparando cada elemento ao seu elemento adjacente (em pares).

2ª Etapa: Quando a regra de comparação é saciada, a permuta dos elementos ocorre.

3ª Etapa: Por fim, executa-se a primeira e a segunda etapa até que não existam elementos a serem permutados.

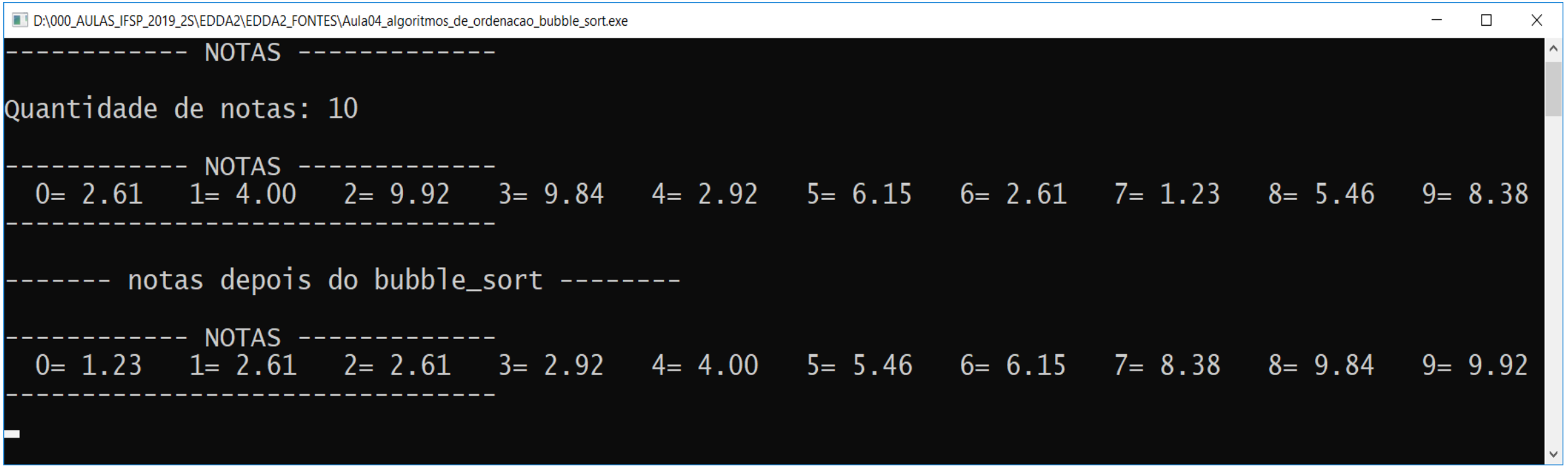
Bubble sort


Bubble sort example
Iniitial
5 3 8 4 6
Initial Unsorted array
Step 1
5 3 8 4 6
Compare 1st and 2nd
(Swap)
Step 2
3 5 8 4 6
Compare 2nd and 3rd
(Do not Swap)
Step 3
3 5 8 4 6
Compare 3rd and 4th
(Swap)
Step 4
3 5 4 8 6
Compare 4th and 5th
(Swap)
Step 5
3 5 4 6 8
Repeat Step 1-5 until
no more swaps required

Bubble sort

```
void bubble_sort(float * vetor)
{
  int i, j; float aux;
  for ( j = 0; j<TAMANHO_VETOR-1; j++)
  {
    for ( i = 0; i<TAMANHO_VETOR-1-j; i++)
    {
      printf ("\nComparando %.2f com %.2f ", vetor[i], vetor[i+1]);
      if ( vetor[i] > vetor[i+1])
      {
         printf ("->empurra %.2f para o fundo", vetor[i] );
         printf ("-> troca com %.2f", vetor[i+1]);
         aux      = vetor[i];
       vetor[i]           = vetor[i+1];
       vetor[i+1]  = aux;
      }
    }
    if (j<TAMANHO_VETOR-1)
       mostra_notas(vetor); /*exibe os valores das notas */
   }
}
```

Bubble sort

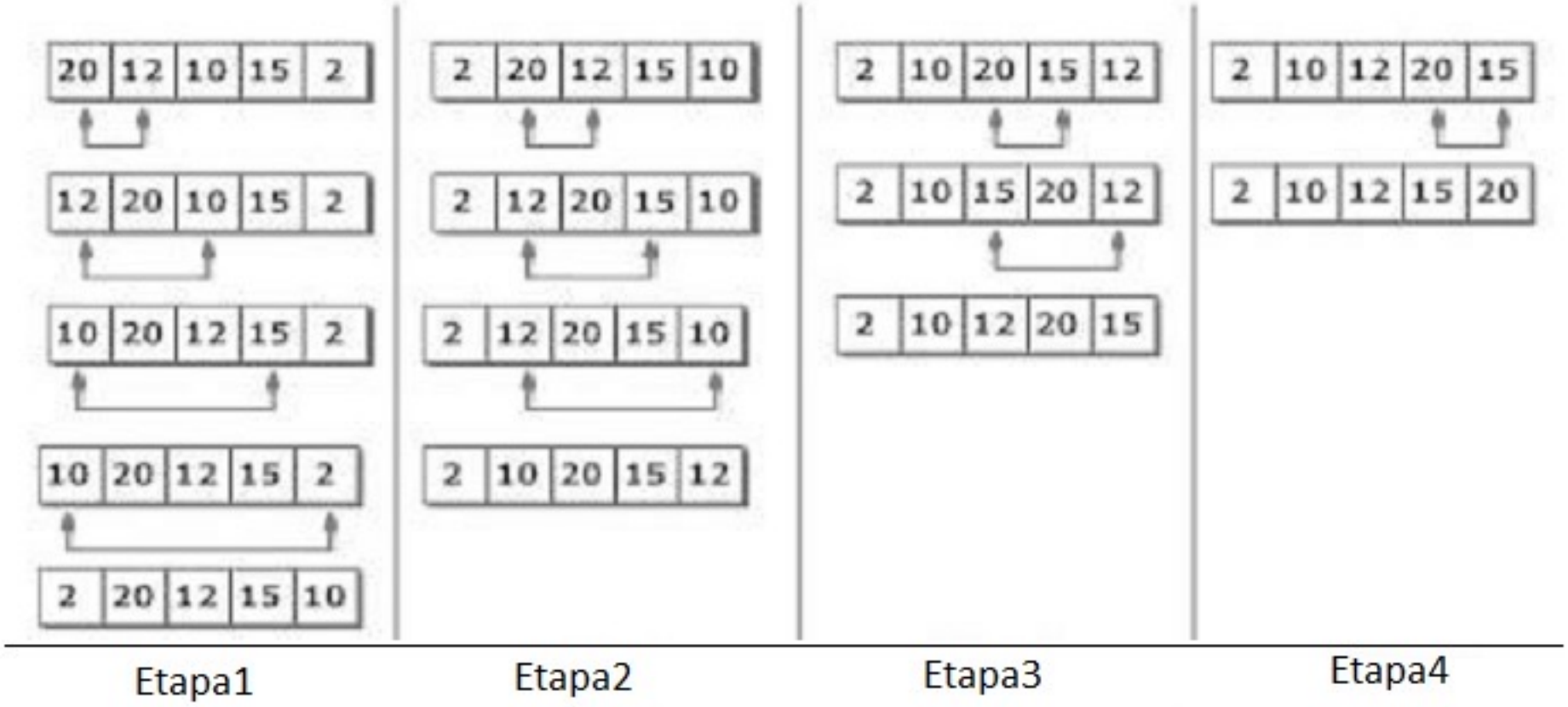

D:\000_AULAS_IFSP_2019_2S\EDDA2\EDDA2_FONTES\Aula04_algoritmos_de_ordenacao_bubble_sort.exe
---------- NOTAS ----------
Quantidade de notas: 10
---------- NOTAS ----------
0= 2.61 1= 4.00 2= 9.92 3= 9.84 4= 2.92 5= 6.15 6= 2.61 7= 1.23 8= 5.46 9= 8.38

------- notas depois do bubble_sort --------
---------- NOTAS ----------
0= 1.23 1= 2.61 2= 2.61 3= 2.92 4= 4.00 5= 5.46 6= 6.15 7= 8.38 8= 9.84 9= 9.92

Selection sort

- A ordenação por seleção (selection sort) é um algoritmo de ordenação que consiste em passar sempre o menor valor do vetor para a primeira posição (ou o maior dependendo da ordem requerida);
- Depois o de segundo menor valor para a segunda posição, e assim é feito sucessivamente com os n-1 elementos restantes, até os últimos dois elementos.

Selection sort



Selection sort

Funcionamento

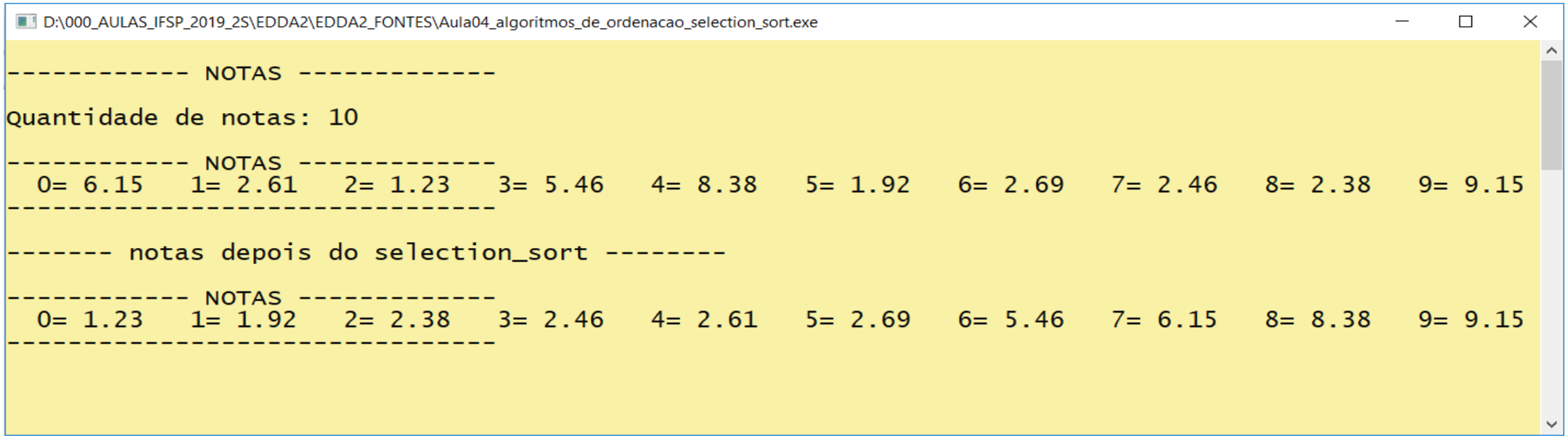
- O primeiro elemento é comparado com os demais a partir da sua direita;
- Quando encontrado um elemento menor, o elemento anteriormente selecionado ocupa a posição do menor elemento encontrado;
- Este elemento encontrado será o próximo elemento a ser selecionado;
- Caso não seja encontrado algum elemento menor que este selecionado, ele é colocado na posição do primeiro elemento considerado;
- E o próximo elemento à sua direita passa a ser o selecionado para fazer as comparações.
- Esse processo se repete até que os elementos estejam ordenados.

Selection sort

Selection sort

```
void selection_sort (float * vetor)
{
    int   pos_min, i, j;
    float           aux;
    /* Percorre todo o vetor até TAMANHO_VETOR-1,
       pois a última posição não precisa testar, pois já estará ordenada */
    for(i=0; i < TAMANHO_VETOR-1; i++)
    {
        pos_min = i; /* A posição do menor valor recebe a posição que está passando */
        /* Percorre o vetor da posição i+1 até o final */
        for (j=i+1; j < TAMANHO_VETOR; j++)
        {
          /* Testa se o elemento da posição que está passando
                é menor que o elemento daquela que tem o menor valor */
            if (vetor[j] < vetor[pos_min])
            {
                pos_min = j; /* pos_min recebe a posição do menor valor */
            }
        }
        /* Se a posição do menor for diferente da que está passando, ocorre a troca */
        if (pos_min != i)
        {
            aux            = vetor[i];
            vetor[i]       = vetor[pos_min];
            vetor[pos_min] = aux;
        }
    }
}
```

Selection sort



Insertion sort

Insertion Sort, trata-se do algoritmo que efetua a ordenação de uma estrutura (array, lista) construindo uma matriz ordenada inserindo um elemento de cada vez.

Insertion sort

Insertion Sort, trata-se do algoritmo que efetua a ordenação de uma estrutura (array, lista) construindo uma matriz ordenada inserindo um elemento de cada vez.

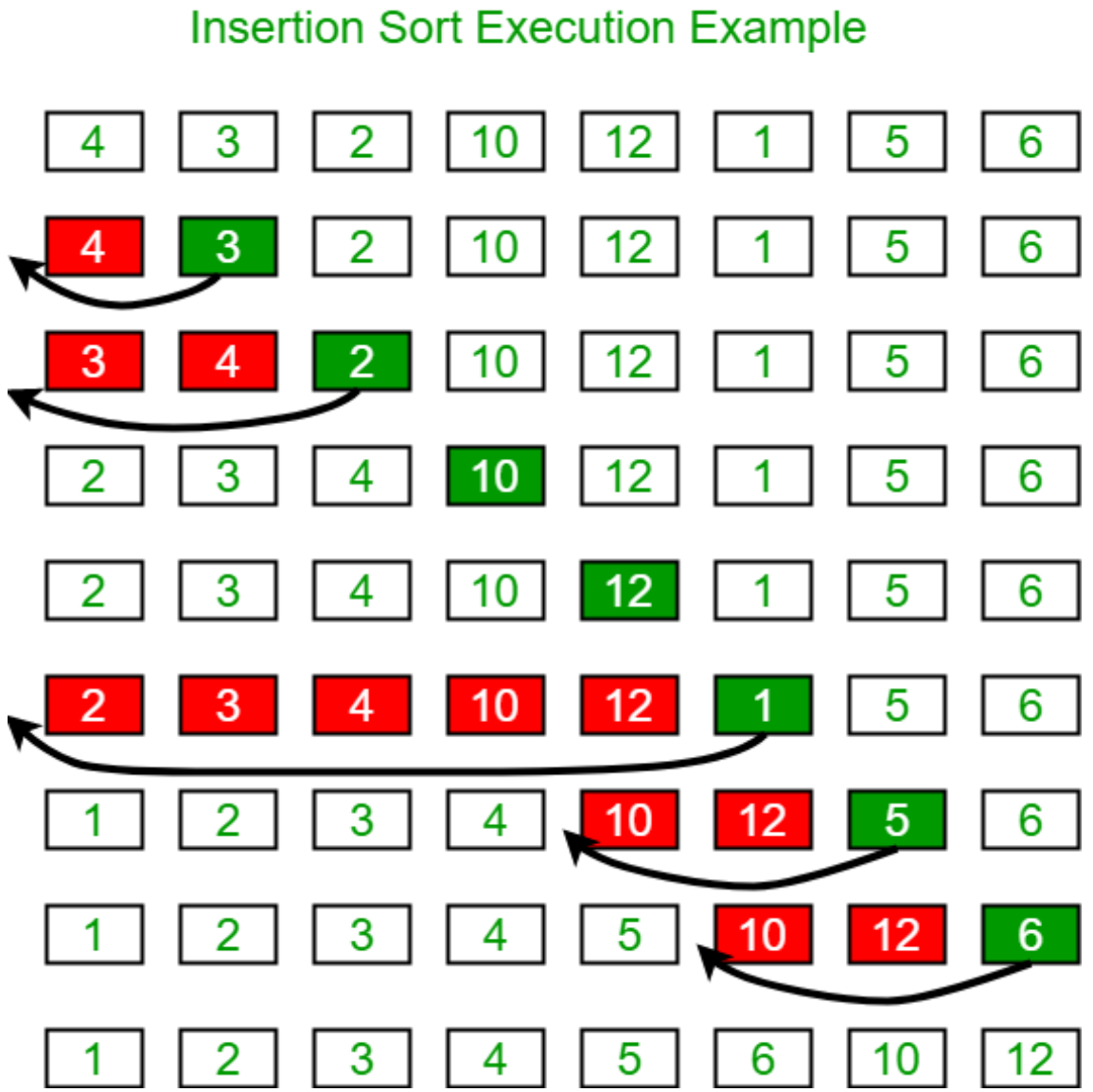
Insertion sort

Funcionamento

- O primeiro elemento a ser selecionado é o segundo.
- Se o elemento selecionado for menor que o elemento antes dele, o elemento selecionado é trocado com seu antecessor.
- Este processo é repetido até que o menor elemento seja “empurrado” para o início do vetor.

6 5 3 1 8 7 2 4

Insertion sort

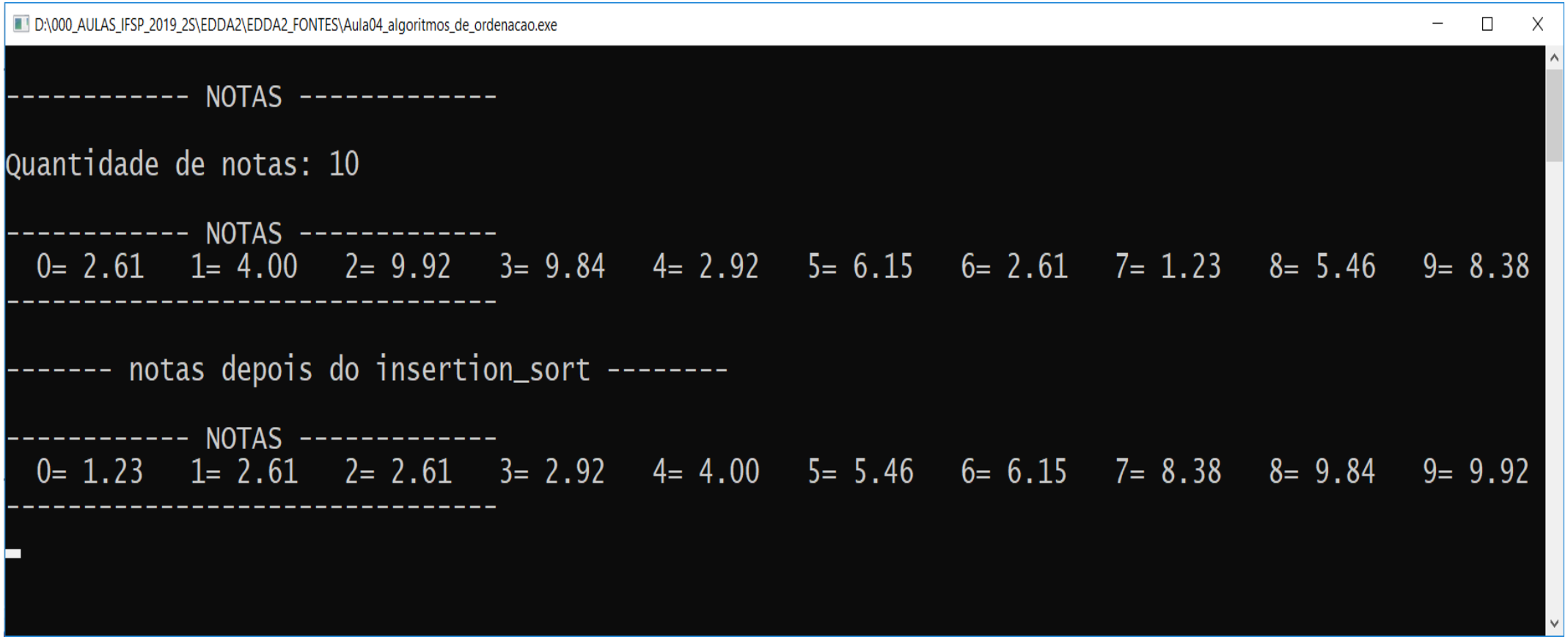


Insertion sort

```
void insertion_sort(float * vetor)
{
    float escolhido;
    int   anterior, i;
    for (i = 1; i < TAMANHO_VETOR; i++)
    {
       escolhido = vetor[i];
       anterior  = i - 1;

         while ( (anterior >= 0) && (vetor[anterior] > escolhido) )
       {
           vetor[anterior + 1] = vetor[anterior];
           anterior--;
       }
       vetor[anterior + 1] = escolhido;
    }
}
```

Insertion sort



Bibliografia

BARNES, David J.; KOLLING, Michael. **Programação orientada a objetos com Java**. 4. ed. São Paulo: Prentice Hall – Br, 2009.

Aditya Y. Bhargava. **Entendendo Algoritmos**. Novatec Editora. ISBN 9788575226629.

Algoritmos. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Cliford Stein. Campus.

Algorithms. Sanjoy Dasgupta, Christos Papadimitriou, Umesh Vazirani. McGraw Hill.

Concrete Mathematics: A Foundation for Computer Science (2nd Edition). Ronald L. Graham, Donald E. Knuth, Oren Patashnik. Addison Wesley.

DEITEL, Paul; DEITEL, Harvey. **Java: como programar**. 10. ed. São Paulo: Pearson, 2017. ISBN 9788543004792. Disponível em: https://ifsp.bv3.digitalpages.com.br/users/publications/9788543004792. Acesso em: 14 jun. 2019.

Helder da Rocha.. **Gerência de memória em Java.** Argonavis: 2005.

Sierra, Kathy. **Use A Cabeça Java**. S.l: Alta Books, 2009. ISBN: 8576081733

INSTITUTO FEDERAL
São Paulo

Câmpus
São Paulo

Obrigado!

Gustavo Fortunato Puga

gustavo.puga@ifsp.edu.br