

# Programação 1

## Aula 10

Valeri Skliarov, Prof. Catedrático

Email: [skl@ua.pt](mailto:skl@ua.pt)

URL: <http://sweet.ua.pt/skl/>

Departamento de Eletrónica, Telecomunicações e Informática  
Universidade de Aveiro

<http://elearning.ua.pt/>

## Objetivos:

1. Aplicações praticas na área de programação (exemplos de pesquisa e ordenação de dados).
2. Operações básicas de pesquisa e ordenação.
3. Estilo de programação e avaliação de algoritmos.
4. Algoritmos de ordenação de dados (sequencial, por flutuação (*bubble*), merge, *radix*, *quick*).
5. Redes de ordenação (*even-odd transition*, *even-odd merge*, *bitonic merge*, *insertion*).
6. Pesquisa de dados
7. Exemplos (veja nomes de algoritmos sublinhados)

# Aplicações praticas na área de programação (pesquisa e ordenação de dados)

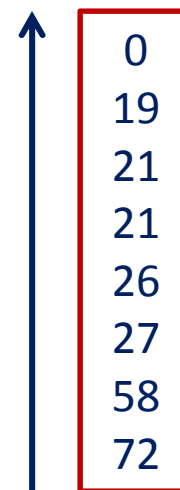
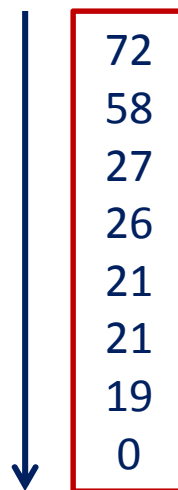
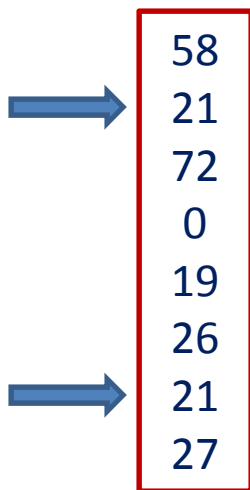
## Ordenação

Tarefa: para um dado conjunto de dados ordenar estes dados

por ordem decrescente

OU

por ordem crescente



Dados podem ter tipos diferentes e podem ser repetidos

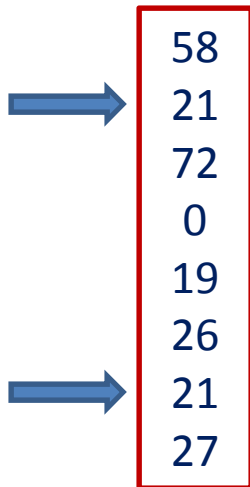
# Aplicações praticas na área de programação (pesquisa e ordenação de dados)

## Pesquisa

Tarefa: para um dado conjunto de dados encontrar dados com valores necessários (ex.: mínimo e máximo)

Mínimo

Máximo



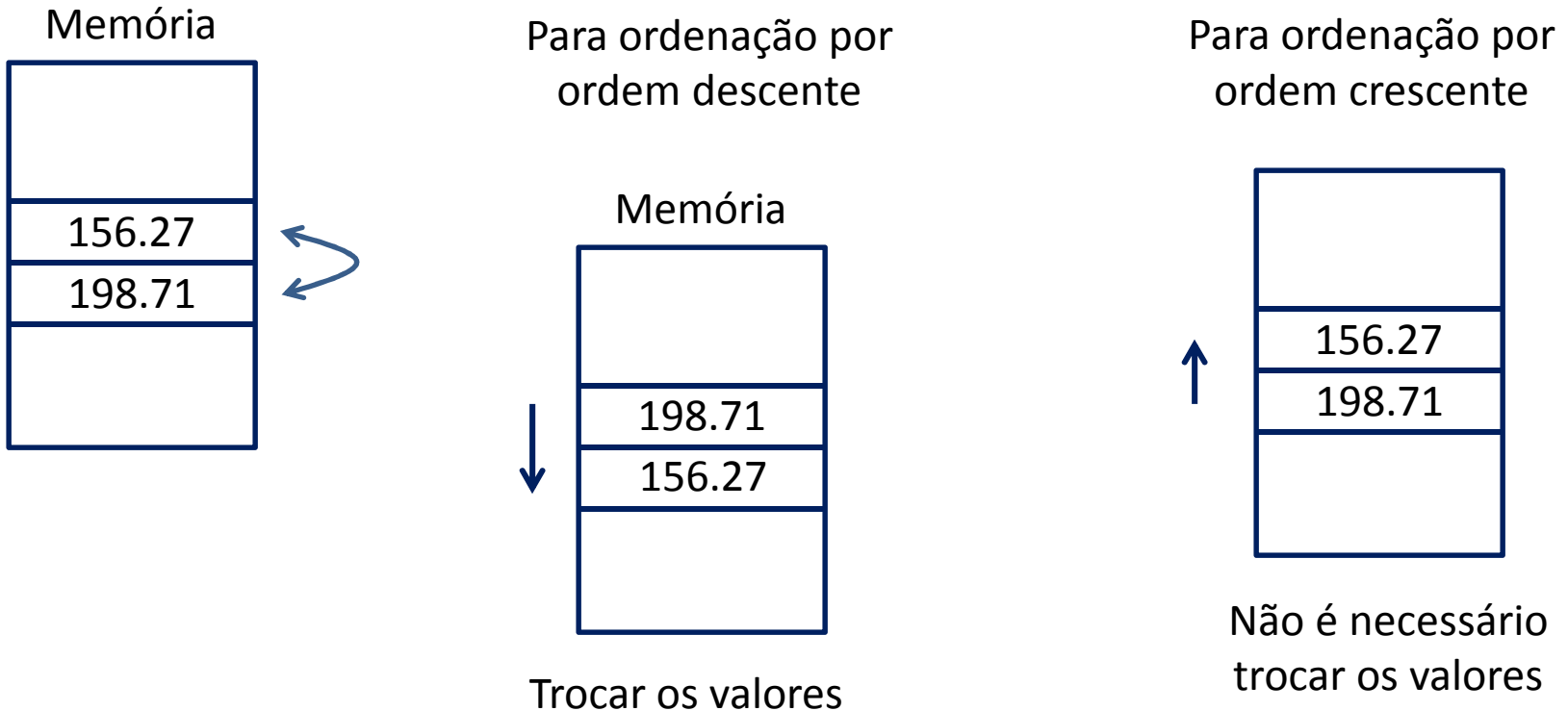
0

72

Dados podem ter tipos diferentes e podem ser repetidos

## Operações básicas de pesquisa e ordenação

### Comparação e troca de dados

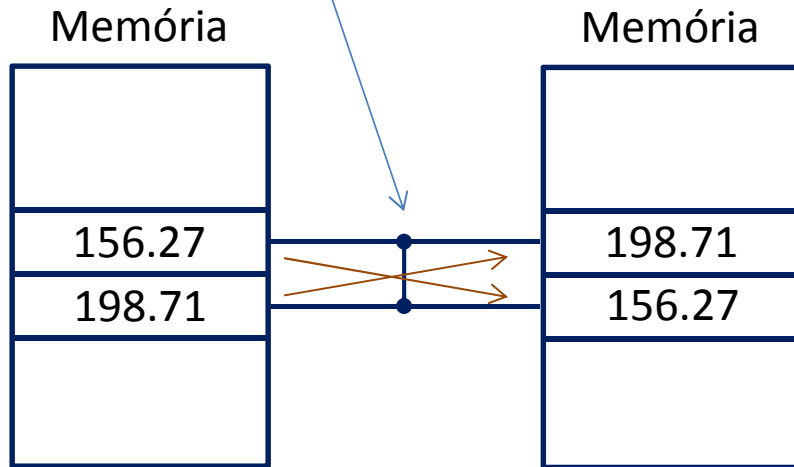


Por isso devemos trocar ou não trocar valores

## Operações básicas de pesquisa e ordenação

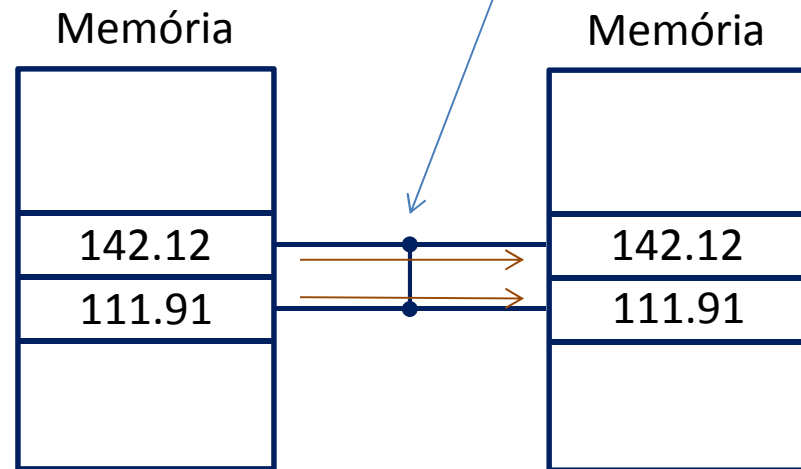
### Comparação e troca de dados

Comparação e troca



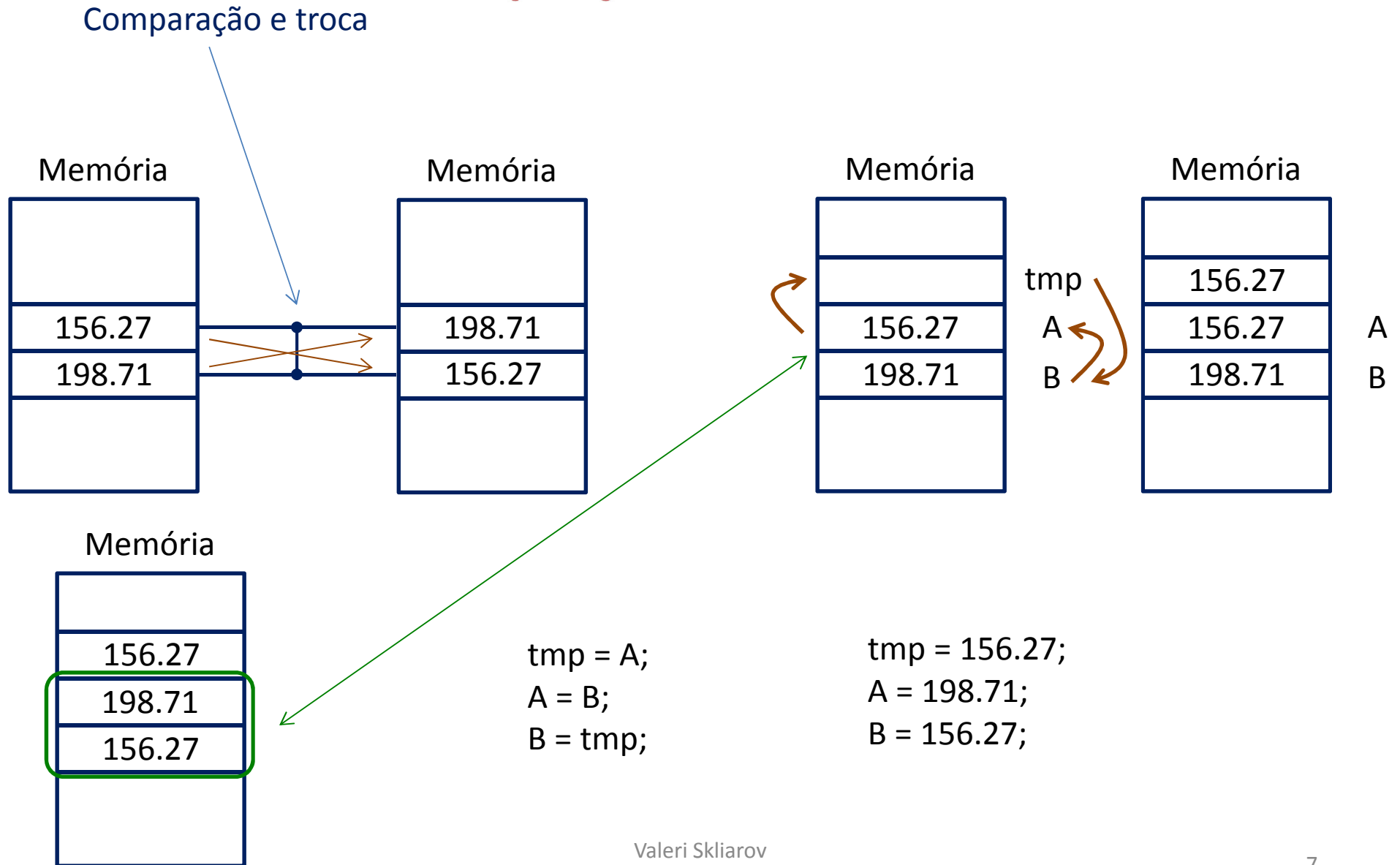
Expressão de troca  
graficamente:

Comparação e troca



## Operações básicas de pesquisa e ordenação

### Comparação e troca de dados



## Estilo de programação

1. Criar funções para operações repetentes. Isto permite simplificar a utilização de operações.

Memória

156.27
198.71
156.27

```
tmp = A;  
A = B;  
B = tmp;
```

Tarefa: descrever uma função que recebe dois argumentos, troca valores dos argumentos e devolve argumentos com valores trocados.

- a) Não é permitido devolver dois valores.
- b) Soluções possíveis:
  - i. Utilizar variáveis globais (geralmente não é bom).
  - ii. Utilizar elementos de um *array* através de argumento – referência para o *array* (geralmente tal estilo é bom e em muitas situações é o melhor).
  - iii. Utilizar elementos de um objeto do tipo *class* (geralmente tal estilo é bom mas para algumas situações não é o melhor).



## Utilização de variáveis globais

Exemplo:

Memória

156.27
198.71

```
tmp = A;  
A = B;  
B = tmp;
```

Deve tentar evitar usar funções deste tipo.

Este estilo pode ser usado mas é melhor tentar evitar.

Em muitas situações particulares tais funções podem afetar os resultados de avaliação.

```
public class UseOfGlobalVariables {  
    static double A, B;  
    public static void main (String args[]) {  
        A = 156.27; B = 198.71;  
        System.out.printf("A = %f, B = %f\n", A,B);  
        comparar_trocar();  
        System.out.printf("A = %f, B = %f\n", A,B);  
    }  
    public static void comparar_trocar() {  
        if (A < B)  
        { double tmp = A;  
          A = B;  
          B = tmp; }  
    }  
}
```

Valor de retorno  
é do tipo **void**

Função não tem  
argumentos

## Utilização de variáveis globais

Exemplo:

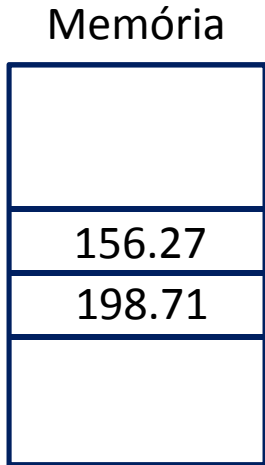
```
public class UseOfGlobalVariables {  
    static double A, B;  
    public static void main (String args[]) {  
        A = 156.27; B = 198.71;  
        System.out.printf("A = %f, B = %f\n", A,B);  
        comparar_trocar();  
        System.out.printf("A = %f, B = %f\n", A,B);  
    }  
    public static void comparar_trocar() {  
        if (A < B)  
        { double tmp = A;  
          A = B;  
          B = tmp; }  
    }  
}
```

Os resultados de execução

```
A = 156.270000, B = 198.710000  
A = 198.710000, B = 156.270000  
Press any key to continue . . .
```

## Utilização de elementos de um *array* através de argumento – referência para o *array*

Exemplo:



tmp = A;  
A = B;  
B = tmp;

Em muitas situações este estilo é muito bom

Este estilo é apropriado para ordenação e pesquisa

É importante que os elementos de um *array* declarado fora da função podem ser alterados dentro da função através de referência para o *array*

```
public class UseOfArrayElements {  
    public static void main (String args[]) {  
        double array[] = { 66.234, 156.27, 198.71, 172.1 };  
        System.out.printf("array[1] = %f, array[2] = %f\n",  
                           array[1],array[2]);  
        comparar_trocar(array,1,2);  
        System.out.printf("array[1] = %f, array[2] = %f\n",  
                           array[1],array[2]);  
    }  
    public static void comparar_trocar(double a[],  
                                       int indice1, int indice2) {  
        if (a[indice1] < a[indice2])  
        { double tmp = a[indice1];  
          a[indice1] = a[indice2];  
          a[indice2] = tmp; }  
    }  
}
```

Valor de retorno  
é do tipo **void**

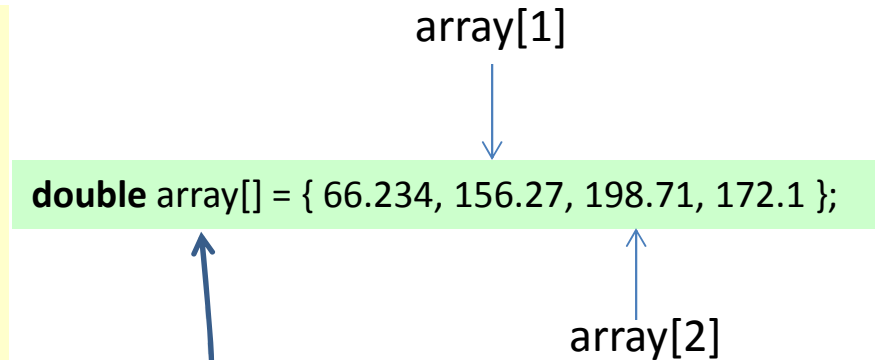
Função tem três argumentos:

1. *Array*.
2. Índice do primeiro elemento do *array*.
3. Índice do segundo elemento do *array*.

## Utilização de elementos de um *array* através de argumento – referência para o *array*

Exemplo:

```
public class UseOfArrayElements {  
    public static void main (String args[]) {  
        double array[] = { 66.234, 156.27, 198.71, 172.1 };  
        System.out.printf("array[1] = %f, array[2] = %f\n",  
                           array[1],array[2]);  
        comparar_trocar(array,1,2);  
        System.out.printf("array[1] = %f, array[2] = %f\n",  
                           array[1],array[2]);  
    }  
    public static void comparar_trocar(double a[],  
                                       int indice1, int indice2) {  
        if (a[indice1] < a[indice2])  
        { double tmp = a[indice1];  
          a[indice1] = a[indice2];  
          a[indice2] = tmp; }  
    }  
}
```



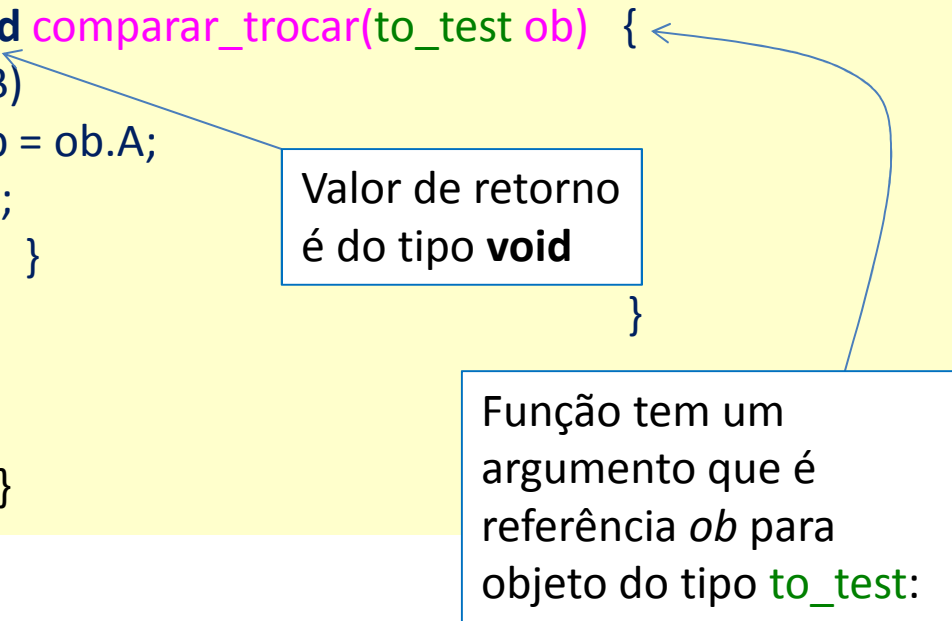
Os resultados de execução

```
array[1] = 156.270000, array[2] = 198.710000  
array[1] = 198.710000, array[2] = 156.270000  
Press any key to continue . . .
```

## Utilização de elementos de um objeto do tipo *class*

Exemplo:

```
public class UseOfObjectElements {  
    public static void main (String args[]) {  
        to_test class_object = new to_test();  
        class_object.A = 156.27;  
        class_object.B = 198.71;  
        System.out.printf("class_object.A = %f, class_object.B = %f\n",  
                           class_object.A, class_object.B);  
        comparar_trocar(class_object);  
        System.out.printf("class_object.A = %f, class_object.B = %f\n",  
                           class_object.A, class_object.B);  
    }  
    public static void comparar_trocar(to_test ob) {  
        if (ob.A < ob.B)  
        { double tmp = ob.A;  
          ob.A = ob.B;  
          ob.B = tmp; }  
    }  
}  
class to_test  
{ double A, B; }
```



Geralmente este estilo é bom para muitas tarefas práticas

Um estilo (por exemplo, este ou anterior) pode ser escolhido analisando representação de dados

Quando os dados são representados num *array*, estilo anterior é melhor

Quando os dados são representados num objeto este estilo é melhor

Conclusão: geralmente o estilo anterior é preferível para ordenação e pesquisa

## Utilização de elementos de um objeto do tipo *class*

Exemplo:

```
public class UseOfObjectElements {  
    public static void main (String args[]) {  
        to_test class_object = new to_test();  
        class_object.A = 156.27; ←  
        class_object.B = 198.71; ←  
        System.out.printf("class_object.A = %f, class_object.B = %f\n",  
                           class_object.A, class_object.B);  
        comparar_trocar(class_object);  
        System.out.printf("class_object.A = %f, class_object.B = %f\n",  
                           class_object.A, class_object.B);  
    }  
    public static void comparar_trocar(to_test ob) {  
        if (ob.A < ob.B)  
        { double tmp = ob.A;  
          ob.A = ob.B;  
          ob.B = tmp; }  
    }  
}  
class to_test  
{ double A, B; }
```

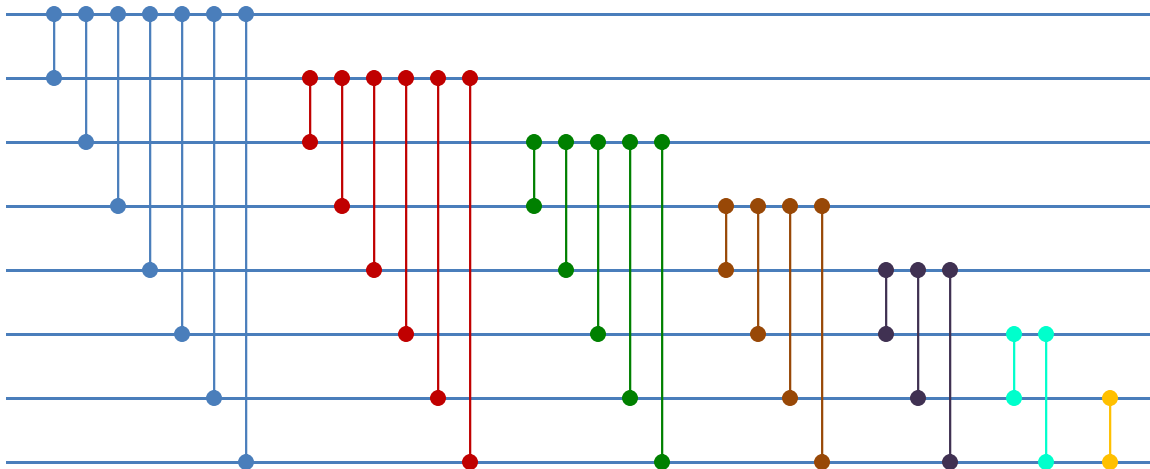
Os resultados de execução

```
class_object.A = 156.270000, class_object.B = 198.710000  
class_object.A = 198.710000, class_object.B = 156.270000  
Press any key to continue . . . _
```

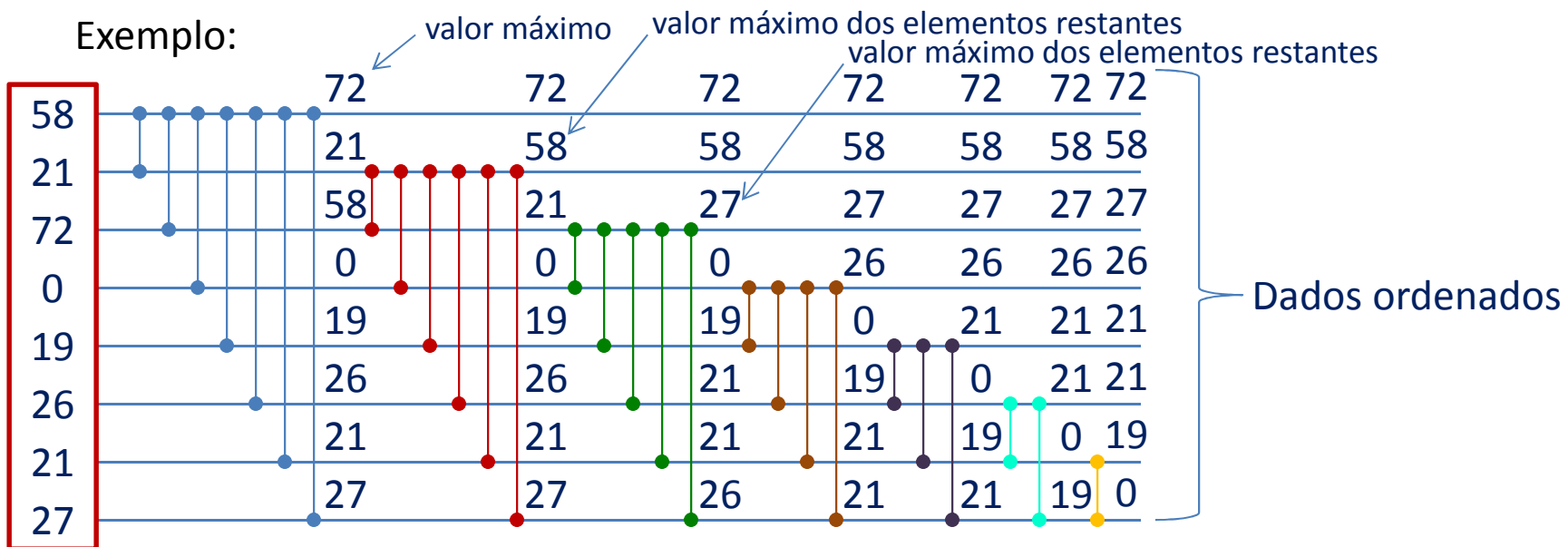
# Algoritmos de ordenação de dados (ordenação sequencial)

## Descrição gráfica do algoritmo

Descrição:

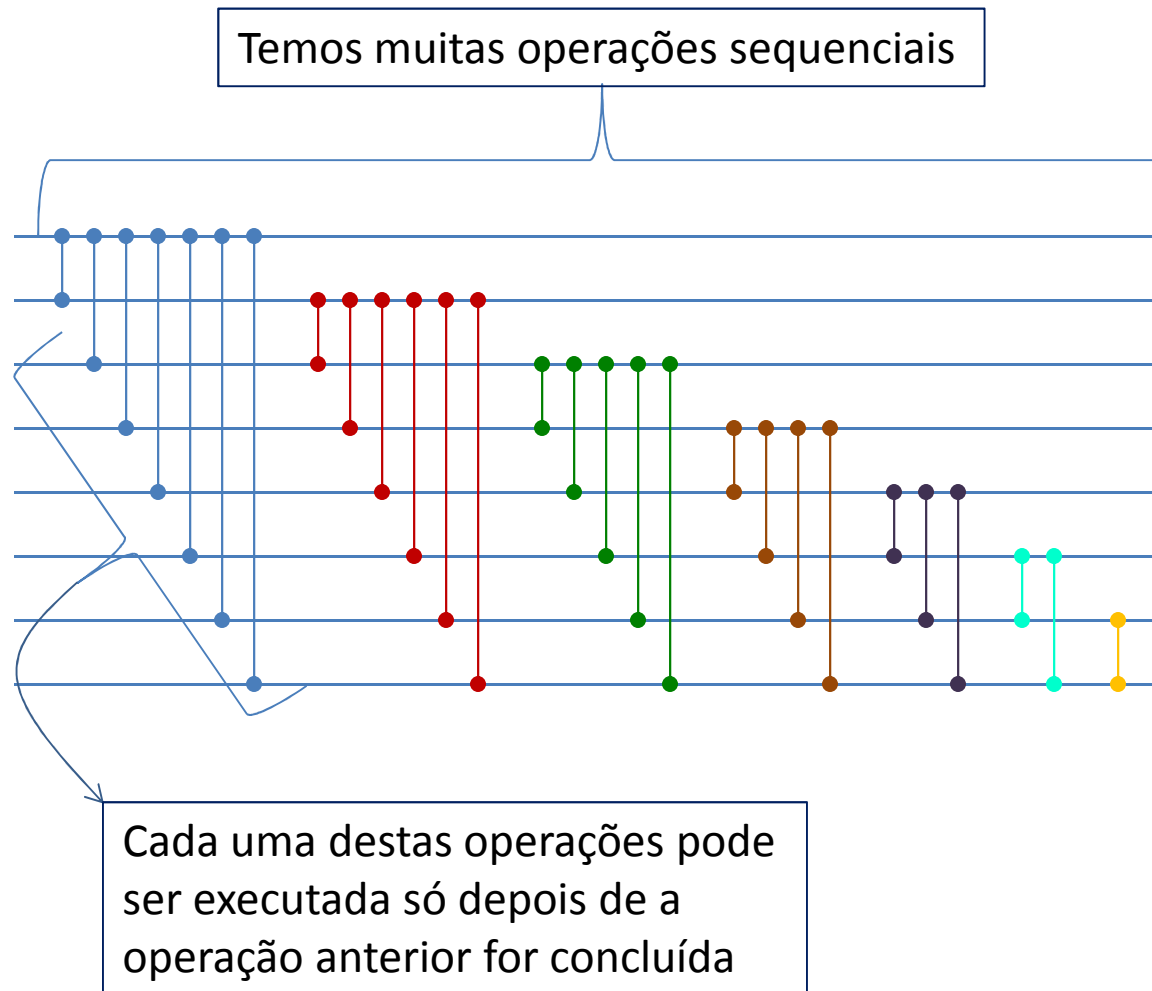


Exemplo:



# Algoritmos de ordenação de dados (ordenação sequencial)

## Análise do algoritmo



Por isso o algoritmo deve ser bastante lento. Isto não é bom



# Algoritmos de ordenação de dados (ordenação sequencial)

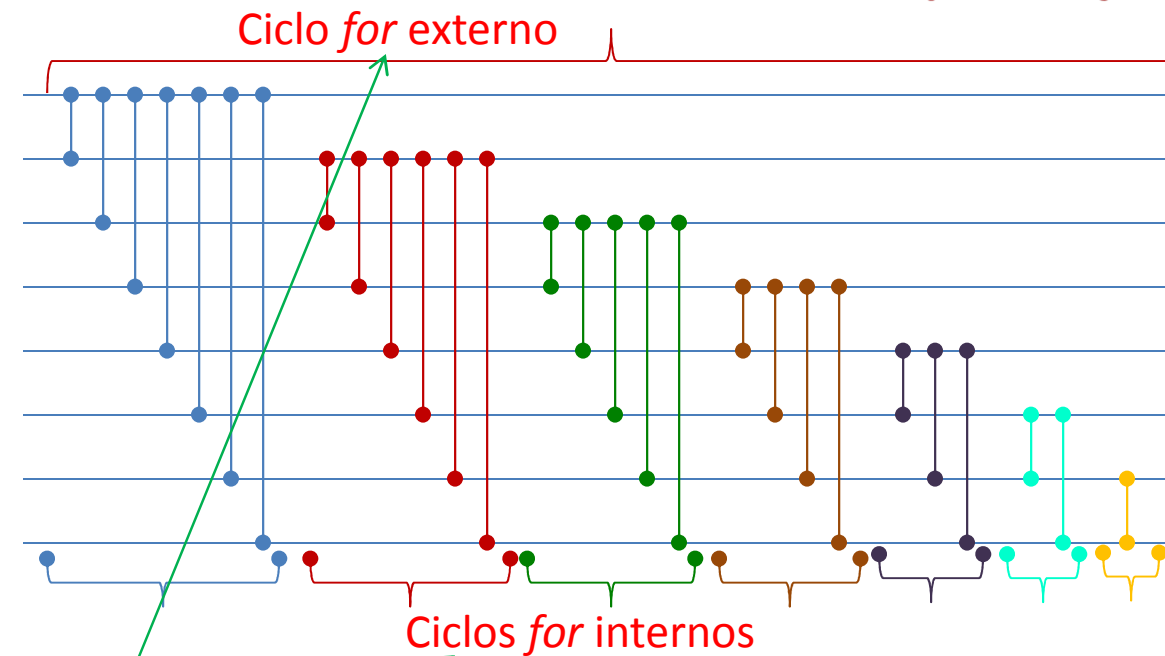
## Implementação

## Estilo de programação

1. O código é compreensível.
2. O código é compacto.
3. O código é baseado nas funções anteriores e isto é bom.

## Função anterior

```
public static void comparar_trocar(int a[],  
                                   int indice1, int indice2) {  
    if (a[indice1] < a[indice2])  
    { int tmp = a[indice1];  
      a[indice1] = a[indice2];  
      a[indice2] = tmp; }  
}
```



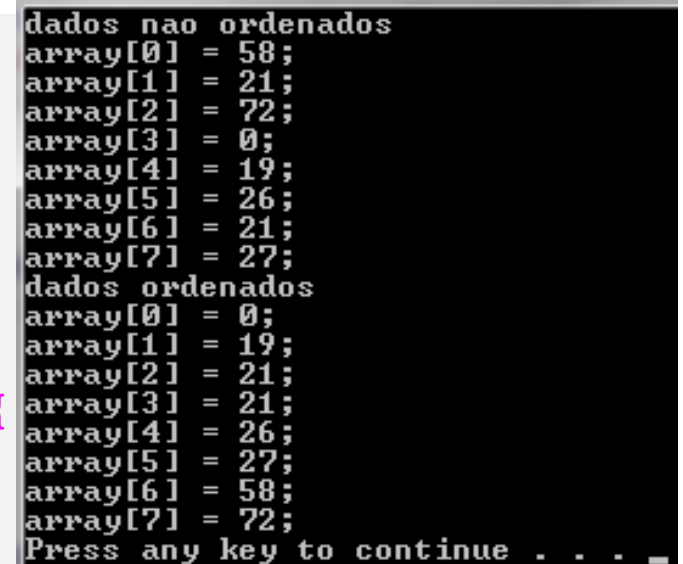
```
public static void sortCrescSeq(int[] num_array) {  
    for(int i = 0; i < num_array.length-1; i++)  
        for(int j = i + 1; j < num_array.length; j++)  
            comparar_trocar(num_array, j, i);  
}
```

# Algoritmos de ordenação de dados (ordenação sequencial)

## Programa completo

```
public class SequentialSorting {
public static void main (String args[]) {
    int a[] = {58,21,72,0,19,26,21,27};
    String dao = "dados nao ordenados", dor = "dados ordenados";
    System.out.println(dao);
    print(a);
    sortCrescSeq(a);
    System.out.println(dor);
    print(a);
}
public static void sortCrescSeq(int[] num_array) {
    for(int i = 0; i < num_array.length-1; i++)
        for(int j = i + 1; j < num_array.length; j++)
            comparar_trocar(num_array, j, i);
}
public static void comparar_trocar(int a[], int indice1, int indice2) {
    if (a[indice1] < a[indice2])
    { int tmp = a[indice1];
      a[indice1] = a[indice2];
      a[indice2] = tmp; }
}
public static void print(int array[])
{
    for(int i = 0; i < array.length; i++)
        System.out.printf("array[%d] = %d;\n", i, array[i]);
}
```

## Os resultados de execução



```
dados nao ordenados
array[0] = 58;
array[1] = 21;
array[2] = 72;
array[3] = 0;
array[4] = 19;
array[5] = 26;
array[6] = 21;
array[7] = 27;
dados ordenados
array[0] = 0;
array[1] = 19;
array[2] = 21;
array[3] = 21;
array[4] = 26;
array[5] = 27;
array[6] = 58;
array[7] = 72;
Press any key to continue . . . _
```

# Algoritmos de ordenação de dados (ordenação sequencial)

## Avaliação de algoritmos

### 1. Geração aleatória de dados:

```
import java.util.*; import java.io.*;
public class merge_sort_random_to_file {
    static Scanner read = new Scanner(System.in);
    static Random rand = new Random();
    static final int N = 128;
    public static void main (String args[]) throws IOException {
        int a[] = new int[N];
        for(int i = 0; i < a.length; i++)
            a[i] = rand.nextInt(1000);
        // .....
    }
}
```

**N** é o número de dados que devem ser gerados aleatoriamente

Vamos reservar memória para *array* **a** com **N** elementos

Geração aleatória de dados

### 2. Medir o tempo de execução:

```
public class MedirTempo {
    public static void main (String args[]) {
        long time=System.nanoTime();
        // chamada da função ou conjunto de instruções
        long time_end=System.nanoTime();
        System.out.printf("measured time (in ns): %d\n",time_end-time);
        System.out.printf("measured time (in ms): %.3f\n",(double)(time_end-time)/1000000.);
    }
}
```

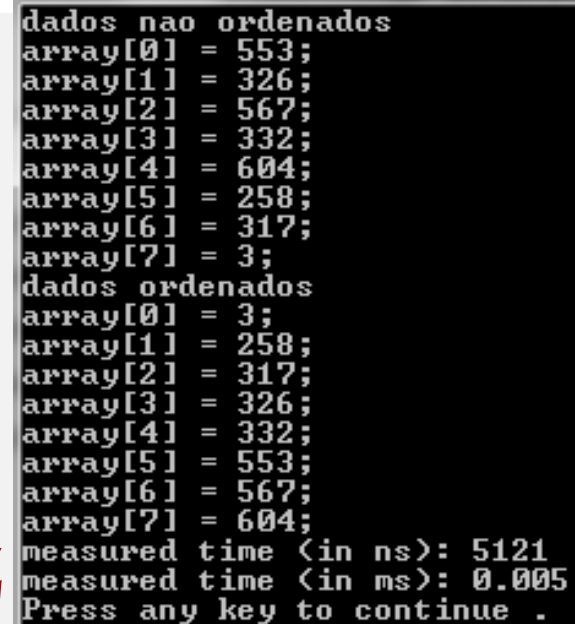
Medir o tempo de execução deste bloco

## Algoritmos de ordenação de dados (ordenação sequencial)

### Programa completo

```
import java.util.*;
public class SequentialSortingRandomAndTime {
    static Random rand = new Random();
    static final int N = 8;
    public static void main (String args[]) {
        int a[] = new int[N];
        for(int i = 0; i < a.length; i++)
            a[i] = rand.nextInt(1000);
        String dao = "dados nao ordenados", dor = "dados ordenados";
        System.out.println(dao);
        print(a);
        long time=System.nanoTime();
        sortCrescSeq(a);
        long time_end=System.nanoTime();
        System.out.println(dor);
        print(a);
        System.out.printf("measured time (in ns): %d\n",time_end-time);
        System.out.printf("measured time (in ms): %.3f\n",(double)(time_end-time)/1000000.);
    }
    public static void sortCrescSeq(int[] num_array) {
        // .....
        public static void comparar_trocar(int a[], int indice1, int indice2) {
            // .....
        }
        public static void print(int array[])
        // .....
    }
}
```

### Os resultados de execução



```
dados nao ordenados
array[0] = 553;
array[1] = 326;
array[2] = 567;
array[3] = 332;
array[4] = 604;
array[5] = 258;
array[6] = 317;
array[7] = 3;
dados ordenados
array[0] = 3;
array[1] = 258;
array[2] = 317;
array[3] = 326;
array[4] = 332;
array[5] = 553;
array[6] = 567;
array[7] = 604;
measured time (in ns): 5121
measured time (in ms): 0.005
Press any key to continue . . . _
```

Two red arrows point from the text "measured time (in ns): 5121" and "measured time (in ms): 0.005" in the code block to the corresponding lines in the execution output screenshot.

# Algoritmos de ordenação de dados (ordenação sequencial)

## Alteração de ordem

Ordem crescente:

```
public static void sortCrescSeq(int[] num_array) {  
    for(int i = 0; i < num_array.length-1; i++)  
        for(int j = i + 1; j < num_array.length; j++)  
            comparar_trocar(num_array, j, i);  
}
```



Os resultados de execução

```
array[0] = 3;  
array[1] = 30;  
array[2] = 98;  
array[3] = 192;  
array[4] = 587;  
array[5] = 753;  
array[6] = 824;  
array[7] = 845;  
measured time <in ns>: 5121  
measured time <in ms>: 0.005  
Press any key to continue . . .
```

Ordem decrescente:

```
public static void sortDecresSeq(int[] num_array) {  
    for(int i = 0; i < num_array.length-1; i++)  
        for(int j = i + 1; j < num_array.length; j++)  
            comparar_trocar(num_array, i, j);  
}
```



Os resultados de execução

```
dados ordenados  
array[0] = 955;  
array[1] = 935;  
array[2] = 913;  
array[3] = 886;  
array[4] = 596;  
array[5] = 452;  
array[6] = 307;  
array[7] = 303;  
measured time <in ns>: 4820  
measured time <in ms>: 0.005  
Press any key to continue . . . _
```

## Algoritmos de ordenação de dados (ordenação sequencial)

Vamos agora executar o mesmo código para: **static final int N = 99;**

### Os resultados de execução

```
array[67] = 419;  
array[68] = 418;  
array[69] = 393;  
array[70] = 381;  
array[71] = 333;  
array[72] = 316;  
array[73] = 302;  
array[74] = 299;  
array[75] = 278;  
array[76] = 269;  
array[77] = 265;  
array[78] = 256;  
array[79] = 253;  
array[80] = 242;  
array[81] = 183;  
array[82] = 144;  
array[83] = 132;  
array[84] = 130;  
array[85] = 130;  
array[86] = 113;  
array[87] = 99;  
array[88] = 76;  
array[89] = 72;  
array[90] = 58;  
array[91] = 57;  
array[92] = 43;  
array[93] = 40;  
array[94] = 34;  
array[95] = 33;  
array[96] = 18;  
array[97] = 12;  
array[98] = 2;  
measured time (in ns): 179551  
measured time (in ms): 0.180  
Press any key to continue . . . _
```

### Estilo de programação

1. O código deve ser facilmente alterado para vários conjuntos de dados.
2. Basta alterar os valores em algumas linhas é o mesmo programa pode ser usado para outros conjuntos de dados.
3. Parametrização é muito importante na programação.

**Exemplo:** **static final int N = 99;**

## Exemplo:

```
import java.util.*;
import java.io.*;
public class WriteToFileTrivial
{
    static Scanner kb = new Scanner(System.in);
    public static void main(String[] args) throws IOException
    {
        String name_of_file, line_of_text;
        System.out.print("Nome do ficheiro: ");
        name_of_file = kb.nextLine();
        File my_file = new File(name_of_file);
        PrintWriter pw = new PrintWriter(my_file);

        for(;;) {
            System.out.print("Linha para escrever: ");
            line_of_text = kb.nextLine();
            if (line_of_text.compareToIgnoreCase("End") == 0) break;
            pw.println(line_of_text);
        }
        pw.close();
    }
}
```

Para nome do ficheiro

Para linhas de texto

throws IOException

Sempre deve  
inserir esta linha

Introduzir o nome do ficheiro

Declarar objeto my\_file do tipo File

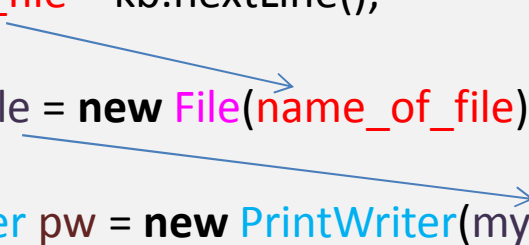
Declarar objeto pw do tipo PrintWriter  
para escrever dados no ficheiro my\_file

Escrever  
dados no  
ficheiro


Fechar o ficheiro

*Como escrever  
resultados no ficheiro  
my\_file*

```
// .....  
import java.io.*;  
// .....  
public static void main(String[] args) throws IOException  
{ String name_of_file;  
  System.out.print("Nome do ficheiro: ");  
  name_of_file = kb.nextLine();  
  
  File my_file = new File(name_of_file);  
  
  PrintWriter pw = new PrintWriter(my_file);  
  
  !!! // .....  
  pw.close();  
  // .....
```



```
public static void print(int array[]) throws IOException {  
  for(int i = 0; i < array.length; i++)  
    System.out.printf("array[%d] = %d;\n",i,array[i]);  
}
```



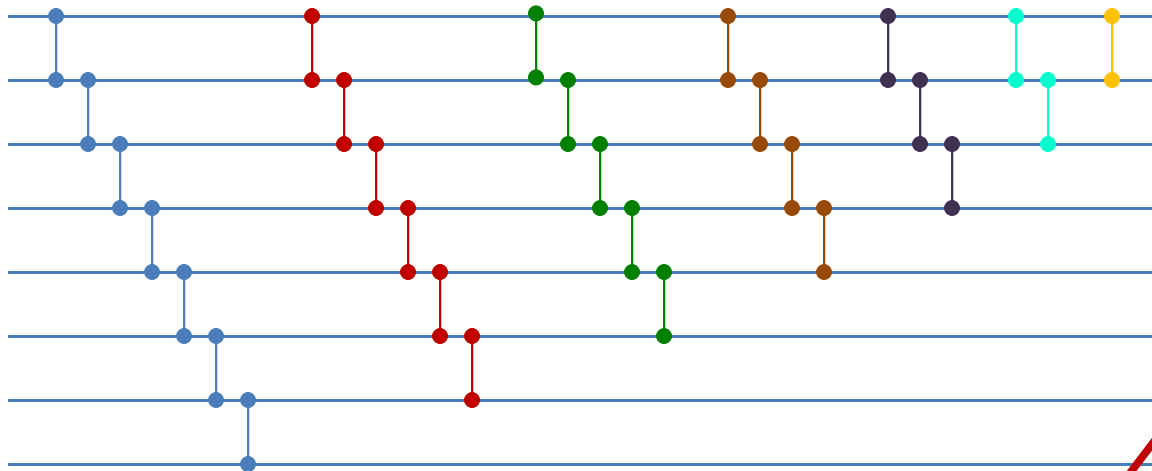
```
pw.printf("array[%d] = %d;\n",i,array[i]);
```



# Algoritmos de ordenação de dados (ordenação por flutuação: bubble sort)

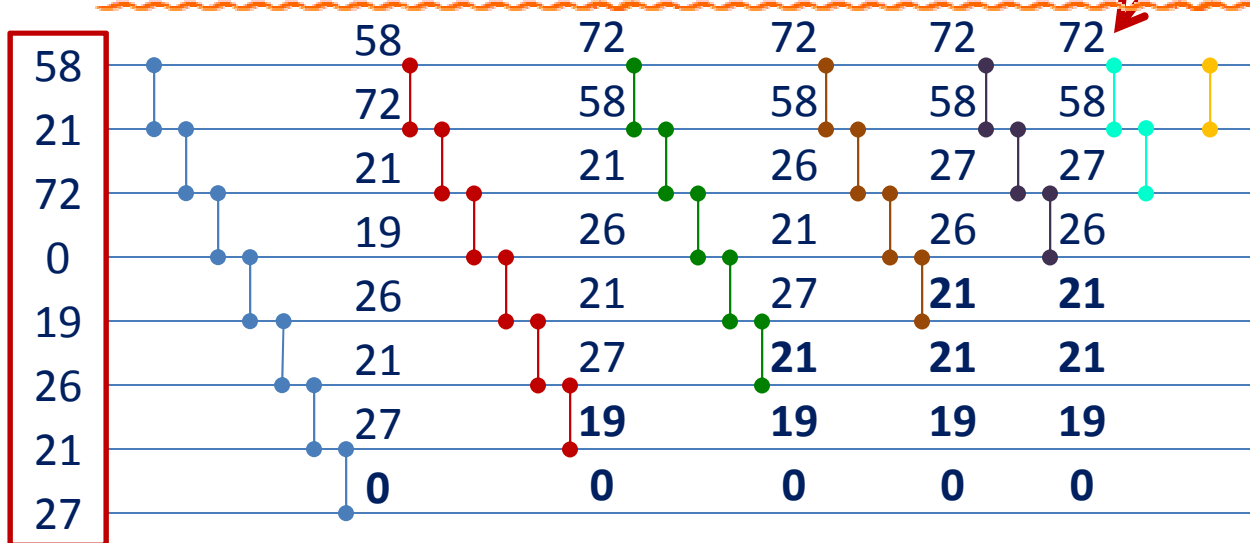
## Descrição gráfica do algoritmo

Descrição:



Não temos  
nenhuma troca  
significa que os  
dados já estão  
ordenados

Exemplo:



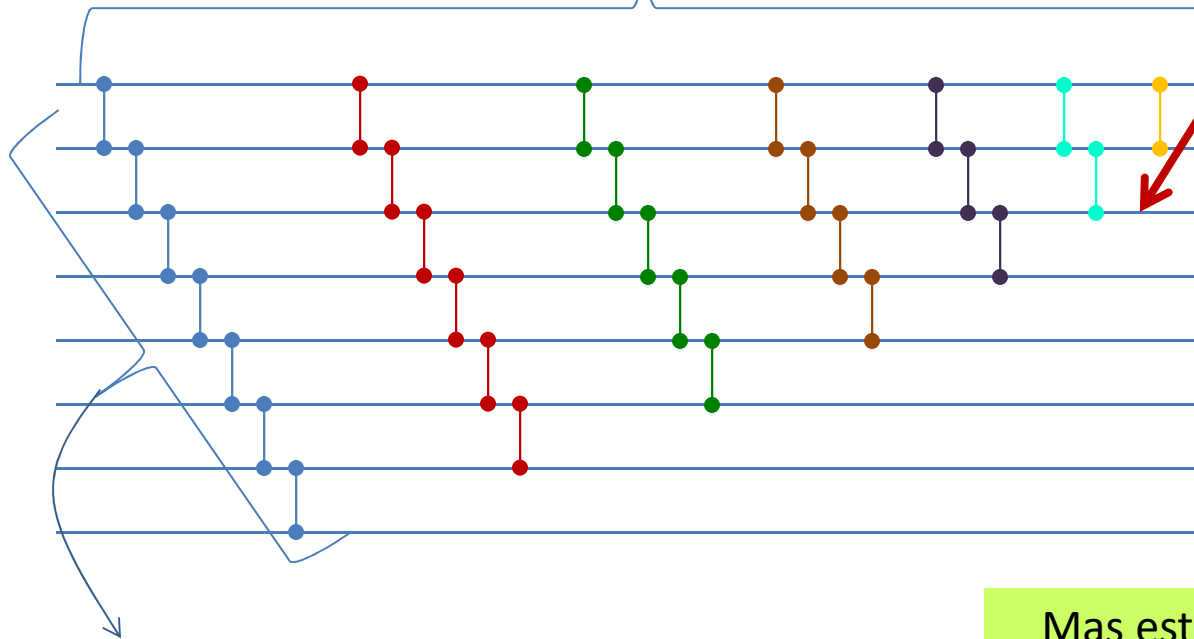
Dados ordenados

Valores mais pequenas vão para baixo

# Algoritmos de ordenação de dados (ordenação por flutuação)

## Análise do algoritmo

Temos muitas operações sequenciais



Cada uma destas operações pode ser executada só depois de a operação anterior for concluída

Não temos nenhuma troca significa que os dados já estão ordenados

Por isso o algoritmo deve ser bastante lento. Isto não é bom

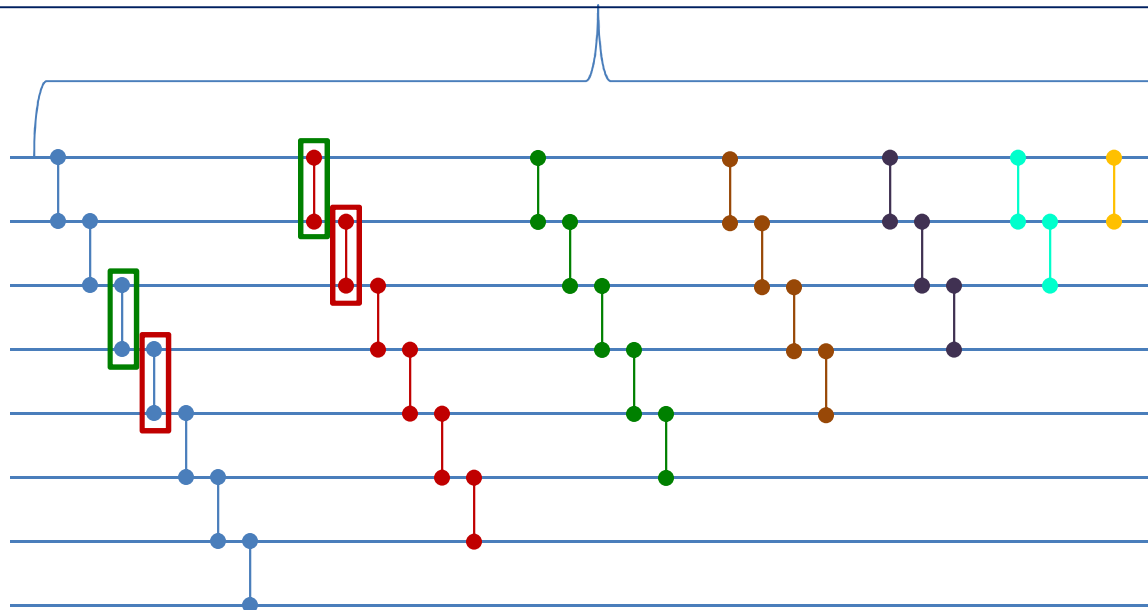
Mas este algoritmo deve ser mais rápido que o algoritmo sequencial porque

**Então podemos obter menor número de passos sequenciais**

# Algoritmos de ordenação de dados (ordenação por flutuação)

## Análise do algoritmo

Podemos implementar algum paralelismo aqui mas este tópico é um tópico avançado



Por exemplo, estas operações podem ser executadas em paralelo

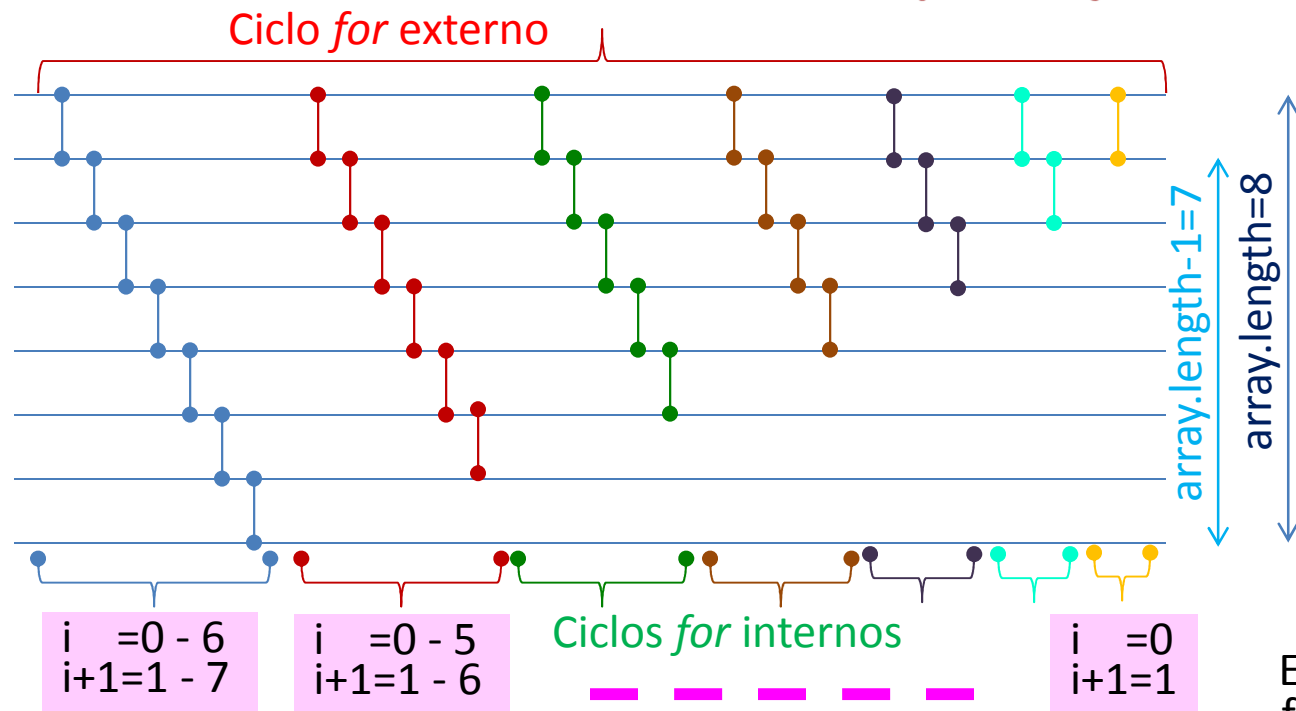
Estes operações também podem ser executadas em paralelo

**Podemos tirar partido de paralelismo em sistemas computacionais avançados**

**Então podemos reduzir o número de passos**

# Algoritmos de ordenação de dados (ordenação por flutuação)

## Implementação



## Estilo de programação

1. O código é compreensível.
2. O código é compacto.
3. O código é baseado nas funções anteriores.

```
public static void sortRiseUpBubble(int[] num_array) {  
    for(int bubble = 1; bubble < num_array.length; bubble++)  
        for(int i = 0; i < num_array.length-bubble; i++)  
            comparar_trocar(num_array,i,i+1);  
}
```

Esta verificação ainda não foi implementada. Vamos implementar tal verificação nos próximos passos

Não temos  
nenhuma troca  
significa que os  
dados já estão  
ordenados

# Algoritmos de ordenação de dados (ordenação por flutuação)

Para ter certeza que os resultados estão corretos

## Verificação de resultados simples (não está completa)

```
public static boolean verifyRiseDown(int array[])
{
    for(int i = 0; i < array.length-1; i++)
        if (array[i] > array[i+1]) return false;
    return true;
}

public static boolean verifyRiseUp(int array[])
{
    for(int i = 0; i < array.length-1; i++)
        if (array[i] < array[i+1]) return false;
    return true;
}
```

Estilo de programação

Verificação de resultados é muito importante.

**Simple**s porque não verifica todos os erros potenciais, por exemplo, o erro seguinte não vai ser detetado:

Array inicial: {**56**,24,33,**54**}

Array final (assumimos que é ordenado): **56**,**56**,33,24

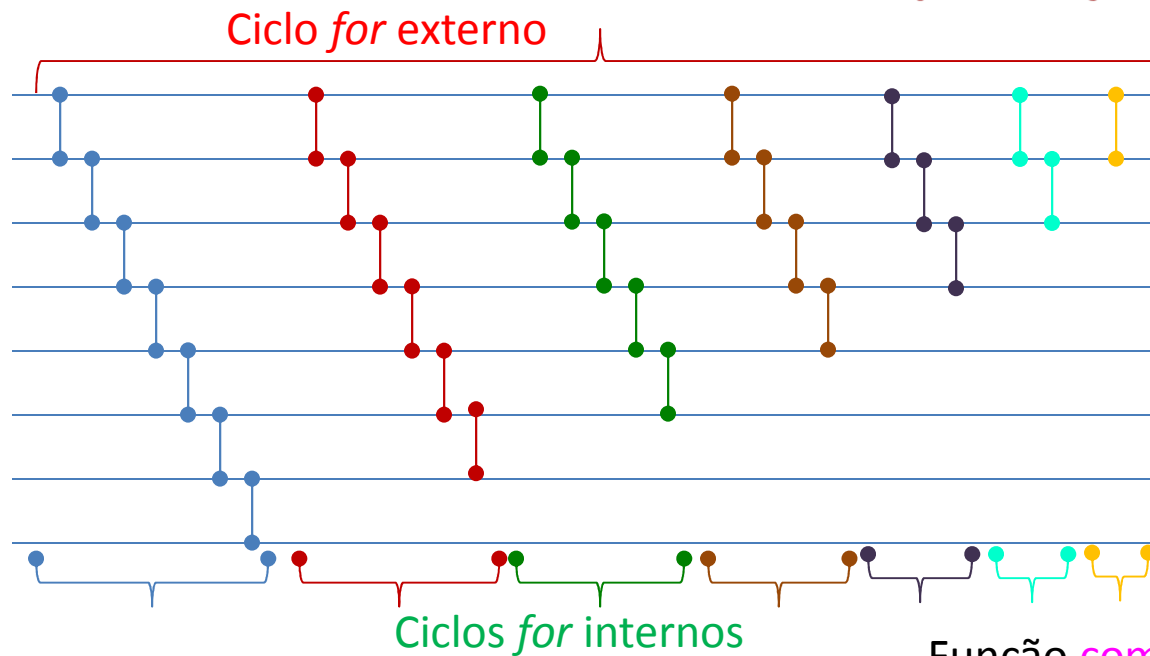
Temos o erro porque o valor **54** desapareceu e o valor **56** foi duplicado.

# Algoritmos de ordenação de dados (ordenação por flutuação)

## Implementação

## Estilo de programação

1. O código é compreensível.
2. O código é compacto.
3. O código é baseado nas funções anteriores (embora um pouco alteradas).



Função **comparar\_trocarR** foi um pouco alterada

```
public static void sortRiseUpBubbleAlternative(int[] num_array) {  
    boolean trocas;  
    for(int bubble = 1; bubble < num_array.length; bubble++)  
    { trocas = false;  
        for(int i = 0; i < num_array.length-bubble; i++)  
            if( comparar_trocarR(num_array,i,i+1) ) trocas = true;  
        if(trocas == false) break;  
    }  
}
```

```
public static boolean comparar_trocarR(int a[],  
                                       int indice1, int indice2) {  
    if (a[indice1] < a[indice2])  
    { int tmp = a[indice1];  
      a[indice1] = a[indice2];  
      a[indice2] = tmp;  
      return true; }  
    return false;  
}
```

Não temos  
nenhuma troca  
significa que os  
dados já estão  
ordenados

## Algoritmos de ordenação de dados (ordenação por flutuação)

### Programa completo

```
import java.util.*;

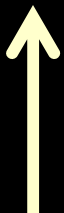
public class BubbleSortingRandomAndTime {
    static Random rand = new Random();
    static final int N = 8;

    public static void main (String args[]) {
        int a[] = new int[N];
        for(int i = 0; i < a.length; i++) a[i] = rand.nextInt(1000);
        String dao = "dados nao ordenados", dor = "dados ordenados";
        System.out.println(dao); print(a);
        long time=System.nanoTime();
        sortRiseUpBubbleAlternative(a); // sortRiseUpBubble(a);
        long time_end=System.nanoTime();
        if (!verifyRiseUp(a)) System.out.println("The results are wrong");
        else System.out.println("The results are probably correct");
        System.out.println(dor); print(a);
        System.out.printf("measured time (in ns): %d\n",time_end-time);
        System.out.printf("measured time (in ms): %.3f\n",(double)(time_end-time)/1000000.); }

    public static void sortRiseDown(int[] num_array) { // .....
    public static void sortRiseUpBubble(int[] num_array) { // .....
    public static void sortRiseUpBubbleAlternative(int[] num_array) { // .....
    public static void comparar_trocar(int a[], int indice1, int indice2) { // .....
    public static boolean comparar_trocarR(int a[], int indice1, int indice2) { // .....
    public static void print(int array[]) // .....
    public static boolean verifyRiseUp(int array[]) // .....
}
```

### Os resultados de execução

```
dados nao ordenados
array[0] = 910;
array[1] = 430;
array[2] = 72;
array[3] = 975;
array[4] = 939;
array[5] = 364;
array[6] = 385;
array[7] = 730;
The results are probably correct
dados ordenados
array[0] = 975;
array[1] = 939;
array[2] = 910;
array[3] = 730;
array[4] = 430;
array[5] = 385;
array[6] = 364;
array[7] = 72;
measured time <in ns>: 5122
measured time <in ms>: 0.005
Press any key to continue . . .
```



## Estilo de programação

Vamos comparar eficiência de dois programas: sem verificação de troca (`sortRiseUpBubble`) e com verificação (`sortRiseUpBubbleAlternative`):

```
public static void sortRiseUpBubble(int[] num_array) {  
    for(int bubble = 1; bubble < num_array.length; bubble++)  
        for(int i = 0; i < num_array.length-bubble; i++)  
            comparar_trocar(num_array, i, i+1);  
}
```

```
public static void sortRiseUpBubbleAlternative(int[] num_array)  
{  
    boolean trocas;  
    for(int bubble = 1; bubble < num_array.length; bubble++)  
    { trocas = false;  
        for(int i = 0; i < num_array.length-bubble; i++)  
            if( comparar_trocarR(num_array, i, i+1) ) trocas = true;  
        if(trocas == false) break;  
    }  
}
```

Não temos  
nenhuma troca  
significa que os  
dados já estão  
ordenados



## Estilo de programação

Vamos comparar eficiência de dois programas: sem verificação de troca (`sortRiseUpBubble`) e com verificação (`sortRiseUpBubbleAlternative`):

```
import java.util.*;
public class CompareBubbleWith_and_WithoutVerification {
    static Random rand = new Random();
    static final int N = 1024*16; // ordenação de 16384 dados
    static final int repeat = 1000;
    public static void main (String args[]) {
        int a[] = new int[N];
        long time, time_end, time1000ex=0;
        for(int k=0; k<repeat; k++)
        {
            for(int i = 0; i < a.length; i++)
                a[i] = rand.nextInt(Integer.MAX_VALUE);
            time =System.nanoTime();
            sortRiseUpBubble(a);
            time_end=System.nanoTime();
            time1000ex += (time_end - time);
        }
        System.out.printf("average time BUBBLE without verification: in ms: ");
        System.out.printf("%.3f\n",(((double)(time1000ex)/1000000.)/repeat));
        time1000ex=0;
        for(int k=0; k<repeat; k++)
        {
            for(int i = 0; i < a.length; i++)
                a[i] = rand.nextInt(Integer.MAX_VALUE);
            time =System.nanoTime();
            sortRiseUpBubbleAlternative(a);
            time_end=System.nanoTime();
            time1000ex += (time_end - time);
        }
        System.out.printf("average time BUBBLE with verification: in ms: ");
        System.out.printf("%.3f\n",(((double)(time1000ex)/1000000.)/repeat));
    }
}
```

O tempo foi medido em mil execuções e depois a média foi calculada.

1. O tempo de execução de `sortRiseUpBubble` (i.e. sem verificação de troca) é melhor que o tempo de execução de `sortRiseUpBubbleAlternative`.
2. Então, não vale a pena fazer o programa mais complicado.

```
average time BUBBLE without verification: in ms: 283.526
average time BUBBLE with verification: in ms: 365.137
Press any key to continue . . . _
```

## Estilo de programação

Agora dois programas para ordenação podem ser escolhidos:

```
public static void sortRiseUpBubble(int[] num_array) {  
    for(int bubble = 1; bubble < num_array.length; bubble++)  
        for(int i = 0; i < num_array.length-bubble; i++)  
            comparar_trocar(num_array, i, i+1);  
}
```

```
public static void sortDecresSeq(int[] num_array) {  
    for(int i = 0; i < num_array.length-1; i++)  
        for(int j = i + 1; j < num_array.length; j++)  
            comparar_trocar(num_array, i, j);  
}
```

Vamos comparar estes programas

## Estilo de programação

```
import java.util.*;
public class CompareBubbleWith_and_WithoutVerification {
    static Random rand = new Random();
    static final int N = 1024*16; // ordenação de 16384 dados
    static final int repeat = 1000;
    public static void main (String args[]) {
        int a[] = new int[N];
        long time, time_end, time1000ex=0;
        for(int k=0; k<repeat; k++)
        {
            for(int i = 0; i < a.length; i++)
                a[i] = rand.nextInt(Integer.MAX_VALUE);
            time =System.nanoTime();
            sortRiseUpBubble(a);
            time_end=System.nanoTime();
            time1000ex += (time_end - time);
        }
        System.out.printf("average time BUBBLE without verification: in ms: ");
        System.out.printf("%.3f\n",((double)(time1000ex)/1000000.)/repeat);
        time1000ex=0;
        for(int k=0; k<repeat; k++)
        {
            for(int i = 0; i < a.length; i++)
                a[i] = rand.nextInt(Integer.MAX_VALUE);
            time =System.nanoTime();
            sortDecresSeq(a);
            time_end=System.nanoTime();
            time1000ex += (time_end - time);
        }
        System.out.printf("average time BUBBLE with verification: in ms: ");
        System.out.printf("%.3f\n",((double)(time1000ex)/1000000.)/repeat);
    }
}
```

Os melhores resultados foram obtidos para ordenação sequencial

Podemos esperar resultados melhores de ordenação por flutuação quando aplicarmos o paralelismo (tópicos mais avançados)

```
average time BUBBLE without verification: in ms: 281.919
average time SEQUENTIAL: in ms: 263.329
Press any key to continue . . . _
```

## Estilo de programação

1. O tempo de execução para programas com funções pode ser maior que o tempo de execução de programas semelhantes sem funções.
2. Mas funções permitem preparar programas mais claros e compreensíveis.
3. Funções permitem também reutilizar o mesmo código e este estilo é muito bom na programação.
4. Utilização de funções pode ser vista como um estilo bom na programação.

## Ordenação por flutuação vs. ordenação sequencial vs. ordenação merge

### MERGE - Fusão

```
import java.util.*;
public class CompareBubbleSeq_and_MERGE {
    static Random rand = new Random();
    static final int N = 1024*16;
    static final int repeat = 1000;
    static final int nWords = 1;
```

```
public static void main (String args[]) {
    int a[] = new int[N];
    long time, time_end, time1000ex=0;
```

```
    int mergeSize = (int)Math.pow(2,nWords);
    for(int k=0; k<repeat; k++)
    {
        for(int i = 0; i < a.length; i++)    a[i] = rand.nextInt(Integer.MAX_VALUE);
        time =System.nanoTime();
        sortRiseUpBubble(a);
        time_end=System.nanoTime();    time1000ex += (time_end - time);    }
    System.out.printf("average time BUBBLE without verification: in ms: ");
    System.out.printf("%.3f\n",((double)(time1000ex)/1000000.)/repeat);
```

```
    time1000ex=0;
    for(int k=0; k<repeat; k++)
    {
        for(int i = 0; i < a.length; i++)    a[i] = rand.nextInt(Integer.MAX_VALUE);
        time =System.nanoTime();
        sortDecresSeq(a);
        time_end=System.nanoTime();    time1000ex += (time_end - time);    }
    System.out.printf("average time SEQUENTIAL: in ms: ");
    System.out.printf("%.3f\n",((double)(time1000ex)/1000000.)/repeat);
```

```
    time1000ex=0;
    for(int k=0; k<repeat; k++)
    {
        for(int i = 0; i < a.length; i++)    a[i] = rand.nextInt(Integer.MAX_VALUE);
        time =System.nanoTime();
        merge_final(a, mergeSize);
        time_end=System.nanoTime();    time1000ex += (time_end - time);    }
    System.out.printf("average time MERGE: in ms: ");
    System.out.printf("%.3f\n",((double)(time1000ex)/1000000.)/repeat);
}
```

```
average time BUBBLE without verification: in ms: 282.833
average time SEQUENTIAL: in ms: 259.648
average time MERGE: in ms: 1.858
Press any key to continue . . . _
```

A ordenação MERGE é 139 vezes mais rápida que a ordenação sequencial e 152 vezes mais rápida que a ordenação por flutuação

Vamos fazer alterações seguintes

```
static final int N = 1024*32;
// .....
for(int k=0; k<repeat; k++)
{
    for(int i = 0; i < a.length; i++)
    a[i] = rand.nextInt((Integer.MAX_VALUE));
    // .....
```

```
average time BUBBLE without verification: in ms: 1146.367
average time SEQUENTIAL: in ms: 1153.756
average time MERGE: in ms: 3.734
Press any key to continue . . .
```

## Implementação de funções para ordenação MERGE (Fusão)

```
public static void merge_final(int a[], int mergeSize)
{
    for (int i = nWords; i < nWords * N; i *= 2)
        for (int j = 0; j < nWords * N; j += i * 2)
        {
            if ((nWords * N - j) > 2 * i) mergeSize = 2 * i;
            else if ((nWords * N - j) > i) mergeSize = nWords * N - j;
            else continue;
            merge(a, j, i, mergeSize);
        }
}

public static void merge(int vec[], int beg, int mid, int vecSize)
{
    int i=0, j=mid, k=0;
    int tmp[] = new int[vecSize];
    while (i < mid && j < vecSize)
        if (vec[i+beg] >= vec[j+beg]) tmp[k++] = vec[beg+i++];
        else tmp[k++] = vec[beg+j++];

    if (i == mid)
        for(int l = k; l < vecSize; l++) tmp[l] = vec[l+beg];
    else
        for(int l = k; l < vecSize; l++) tmp[l] = vec[l-mid+beg];
    for(int ii = 0; ii < vecSize; ii++)
        vec[ii+beg] = tmp[ii];
}
```

# Aplicações praticas na área de programação (pesquisa e ordenação de dados)

## Pesquisa sequencial

A pesquisa compara um dado valor sequencialmente com todos os valores no conjunto até encontrar o valor pretendido ou até atingirmos o último elemento (que significa que não encontramos o valor pretendido)

```
public static int PesquisaSequencial(int seq[],int valor ) {  
    for(int i = 0 ; i < seq.length; i++)  
        if(seq[i] == valor) return i;  
    return -1;  
}
```

Os resultados

```
o numero ? 87  
O numero está na pos 5  
Press any key to continue . . . _
```

## Programa completo

```
import java.util.*;  
public class SequentialSearch {  
    public static Scanner sc = new Scanner(System.in);  
    public static void main (String args[]) {  
        int a[] = { 56,21,267,345,12,87,267};  
        int ind;  
        System.out.print("o numero ? ");  
        System.out.println(((ind=PesquisaSequencial(a, sc.nextInt())) < 0) ?  
            "O numero não existe" :  
            "O numero está na pos " + ind);  
    }  
}
```

Operação ternária

# Aplicações praticas na área de programação (pesquisa e ordenação de dados)

## Pesquisa binária

```
import java.util.*;
public class SequentialSearch {
    public static Scanner sc = new Scanner(System.in);
    public static void main (String args[]) {
        int a[] = { 17,87,100,199,267,300,321}; // o array deve ser ordenado
        int ind;
        System.out.print("o numero ? ");
        System.out.println(((ind=PesquisaBinaria(a, sc.nextInt())) < 0) ?
            "O numero não existe" :
            "O numero está na pos " + ind);
    }
    public static int PesquisaBinaria(int seq[], int valor){
        int inicio = 0, fim = seq.length, meio;
        for (;inicio <= fim;)
        { meio = (fim+inicio)/2;
            if(seq[meio] == valor) return meio;
            else if (seq[meio] > valor) fim = meio - 1;
            else inicio = meio + 1;
        }
        return -1;
    }
}
```

### Programa completo

```
o numero ? 300
O numero está na pos 5
Press any key to continue . . . _
```



# Sumario

1. O próprio estilo de programação é importante.
2. Deve perceber todos os detalhes dum algoritmo antes de o implementar em software
3. Eliminar iterações de ciclos não necessárias.
4. Não introduzir variáveis não necessárias.
5. Usar funções propriamente.
6. Usar argumentos e valores de retorno propriamente.
7. Comparar a eficiência dos programas antes de concluir se os programas são bons.
8. Reutilização!!!

# Importante para aulas:

## Sumário (aula 9)

### Leitura

1 String

```
File my_file = new File(name_of_file);
Scanner read_from_file = new Scanner(my_file);
while(read_from_file.hasNextLine())
    System.out.println(read_from_file.nextLine());
```

### Escrita

1 String

```
File my_file = new File(name_of_file);
PrintWriter pw = new PrintWriter(my_file);
pw.println(line_of_text);
```

Usar semelhante ao **System.out**.


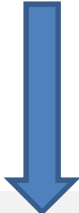
### Verificação

```
if (!my_file.exists()) {
    System.out.println("ERROR: input file " + nameIn + " does not exist!");
    System.exit(2);
}
if (my_file.isDirectory()) {
    System.out.println("ERROR: input file " + nameIn + " is a directory!");
    System.exit(3);
}
if (!my_file.canRead()) {
    System.out.println("ERROR: cannot read from input file " + nameIn + "!");
    System.exit(4);
}
```


Fechar objetos.

Ex.: `read_from_file.close();`  
`pw.close();`

# Importante para aulas:



```
import java.io.*;  
public static void main(String[] args) throws IOException  
  
...close();
```



```
public static void print(int array[]) throws IOException {  
    for(int i = 0; i < array.length; i++)  
        pw.printf("array[%d] = %d;\n",i,array[i]);  
}
```

```
import java.util.*;
import java.io.*;
public class SortStudentsByDifferentFields {
    static Scanner read = new Scanner(System.in);
    static final int N = 7; // numero de alunos maximo numa turma
    public static void main (String args[]) throws IOException {
        String line[] = new String[N];
        String names_of_students[] = new String[N];
        int n_mecs[] = new int[N];
        double notas[] = new double[N];
        -----
    }
}
```

Nuno Ferreira	3671	15	23
Ana Patricia	4511	13	22
Alexandre Carvalho	3499	16	30
Pedro Silva	7619	10	19
Ricardo reis	9921	17	27
Claudia Silva	5676	14	20
Carla Pereira	4691	16	28



Sort by names:

Alexandre Carvalho	3499	16	30
Ana Patricia	4511	13	22
Carla Pereira	4691	16	28
Claudia Silva	5676	14	20
Nuno Ferreira	3671	15	23
Pedro Silva	7619	10	19
Ricardo reis	9921	17	27

Sort by n\_mec:

Alexandre Carvalho	3499	16	30
Nuno Ferreira	3671	15	23
Ana Patricia	4511	13	22
Carla Pereira	4691	16	28
Claudia Silva	5676	14	20
Pedro Silva	7619	10	19
Ricardo reis	9921	17	27

Sort by grades:

Pedro Silva	7619	10	19
Ana Patricia	4511	13	22
Claudia Silva	5676	14	20
Nuno Ferreira	3671	15	23
Alexandre Carvalho	3499	16	30
Carla Pereira	4691	16	28
Ricardo reis	9921	17	27

Sort by age:

Pedro Silva	7619	10	19
Claudia Silva	5676	14	20
Ana Patricia	4511	13	22
Nuno Ferreira	3671	15	23
Ricardo reis	9921	17	27

# e-learning

```

import java.util.*;

public class ex1101 {

    static Scanner read = new Scanner(System.in);

    public static void main (String args[]) {
        int num_array[] = new int[50];
        int op = 0;
        do{
            op = printMenu();
            switch(op) {
                case 1:
                    num_array = readSeq();
                    num_array = removeZeros(num_array);
                    break;
                case 2:
                    printSeq(num_array);
                    break;
                case 3:
                    printMax(num_array);
                    break;
                case 4:
                    printMin(num_array);
                    break;
                case 5:
                    printMed(num_array);
                    break;
                case 6:
                    printIfEven(num_array);
                    break;
                case 7:
                    sortCrescSeq(num_array);
                    printSeq(num_array);
                    break;
                case 8:
                    sortDecresFlut(num_array);
                    printSeq(num_array);
                    break;
                case 9:
                    searchSeq(num_array);
                    break;
                default:
                    break;
            }
        } while (op != -1);
    }
}

```

---

```

import java.util.*;
import java.io.*;

//*****
// Please pay attention to correct use of operations over
// data in files and internal data in the program, namely:
// execute operation 1 if you want to verify data from operation :
//*****

public class MeteoStationOpt {
    static Scanner read = new Scanner(System.in);
    static final String file_name = "data.txt";
    static final int samples = 31;
    static final String meteodata_fields[] = { "Temperature", "Humid", "Wind", "Pressure", "Clouds" };
    static final String types[] = { "sort", "min max", "media" };
    // statistics object below is common for Meteostation and may be used in other programs
    // here (although this is questionable and programmer-dependent)
    static final results statistics = new results();

    public static void main (String args[]) throws IOException {
        meteodata md[] = null;
        String s[] = { "Estacao meteorologica:",
                       "  1 - Ler ficheiro de dados",
                       "  2 - Acrescentar medida",

```

# e-learning