UFRGS
UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

Informática UFRGS

# INF01202
# Algoritmos e Programação
## Modalidade Ensino a Distância

## Linguagem C
### Tópico 10: Aplicações com arranjos unidimensionais

*Profas. Maria Aparecida M. Souto*
*Ivone M. Medero*
*Maria Aparecida Castro Livi*

Duas importantes operações sobre arranjos são a classificação e a pesquisa.

A importância dessas operações pode ser avaliada a partir de exemplos do mundo real.

Pode-se imaginar como seria difícil encontrar-se um nome no listão do vestibular ou em uma lista telefônica de uma grande cidade se essas listagens não estivessem classificadas.

E como seria trabalhoso encontrar-se material sobre um determinado assunto, se as informações nas várias mídias não estivessem de alguma forma organizadas.

# Classificação
# e
# pesquisa de dados

# em arranjos

# Classificação

Classificar um arranjo significa rearranjar seus elementos segundo uma ordem pré-determinada, ascendente ou descendente.

O principal objetivo da classificação é facilitar o posterior processo de ***pesquisa*** sobre o arranjo ordenado.

# Algoritmos de Classificação

Existem inúmeros algoritmos de classificação, com variada complexidade e eficiência.

A seguir são apresentados três métodos simples de classificação.

Os dois primeiros são bastante ineficientes, uma vez que são incapazes de detectar quando o conjunto de dados já se encontra ordenado.

O terceiro é o método conhecido como "Método da bolha" ou "*bubblesort*", mais otimizado.

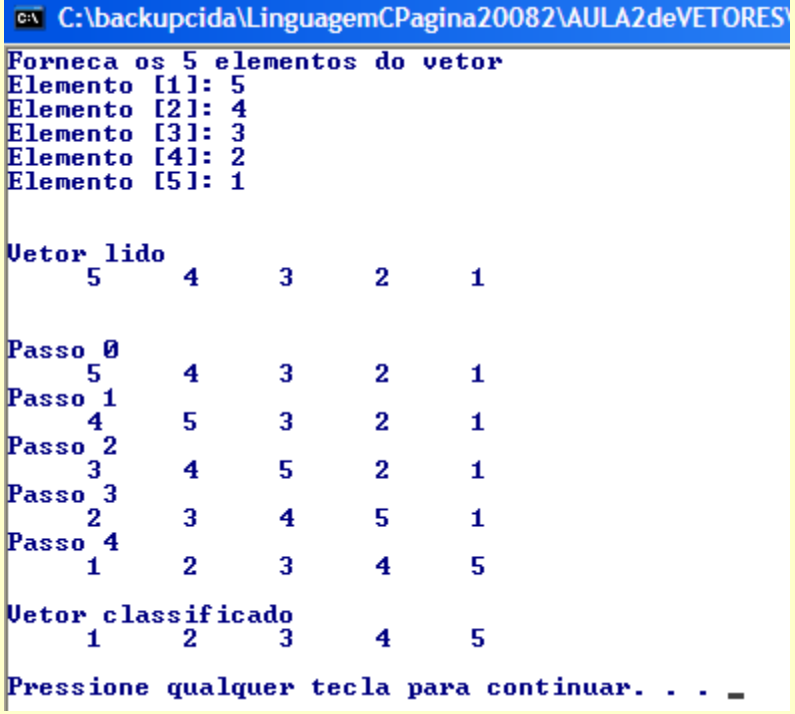
# Método de classificação 1: ordem crescente. *Ineficiente*

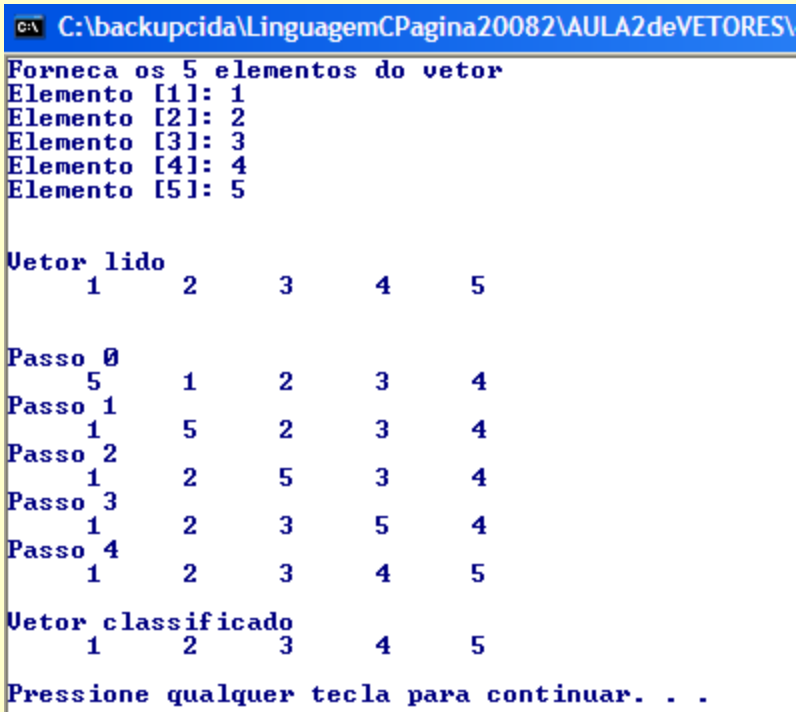
```
#include <stdio.h>
#include <stdlib.h>
#define MAXVET 5

int main (  )
 {
   int vet[MAXVET];
   int i, j, k, aux;
   system("color f1");
   //leitura do vetor
   printf("Forneca os %d elementos do vetor\n", MAXVET);
   for (i=0;i<MAXVET;i++)
    {
      printf("Elemento [%d]: ", i + 1);
      scanf("%d", &vet[i]);
    }
   printf("\n\nVetor lido\n");
   for (i=0;i<MAXVET;i++)
     printf("%6d", vet[i]);
   printf("\n\n");
```

**Estratégia:** são realizadas tantas varreduras do arranjo quantos são os seus elementos. Em cada varredura compara-se um elemento com todos os demais (exceto o próprio). Troca-se dois elementos de lugar sempre que necessário.

Cont.

```
...
//trecho da classificacao
for (i =0;i < MAXVET;i++)
  {
    for (j=0;j<MAXVET;j++)
      if (i!=j)
         if (vet[i] < vet[j])
            {
              aux = vet[i];
              vet[i] =vet[j];
              vet[j] = aux;
            }
  }
printf("\n\nVetor classificado\n");
for (i=0;i<MAXVET;i++)
  printf("%6d", vet[i]);
printf("\n\n");
system("pause");
return 0;
}
```



# Método de classificação 2: ordem crescente. *Ineficiente*

```
#include <stdio.h>
#include <stdlib.h>
#define MAXVET 5

int main (  )
 {
   int vet[MAXVET];
   int i, j, k, aux;
   system("color f1");
   //leitura do vetor
   printf("Forneca os %d elementos do vetor\n", MAXVET);
   for (i=0;i<MAXVET;i++)
    {
      printf("Elemento [%d]: ", i + 1);
      scanf("%d", &vet[i]);
    }
   printf("\n\nVetor lido\n");
   for (i=0;i<MAXVET;i++)
     printf("%6d", vet[i]);
   printf("\n\n");
   ...
```

**Estratégia:** são realizadas tantas varreduras do arranjo quantos são os seus elementos. Em cada varredura compara-se os elementos dois a dois Troca-se dois elementos de lugar sempre que necessário.

Cont.

```
...
//trecho da classificacao
for (i =0;i < MAXVET;i++)
  {
    for (j=0;j<MAXVET - 1;j++)
      if (vet[j] > vet[j+1])
         {
           aux = vet[j];
           vet[j] =vet[j+1];
           vet[j+1] = aux;
         }
  }
printf("\n\nVetor classificado\n");
for (i=0;i<MAXVET;i++)
  printf("%6d", vet[i]);
printf("\n\n");
system("pause");
return 0;
}
```

```
C:\backupcida\LinguagemCPagina20082\AULA2deVE
Forneca os 5 elementos do vetor
Elemento [1]: 1
Elemento [2]: 2
Elemento [3]: 3
Elemento [4]: 4
Elemento [5]: 5

Vetor lido
     1     2     3     4     5

Passo 0
     1     2     3     4     5
Passo 1
     1     2     3     4     5
Passo 2
     1     2     3     4     5
Passo 3
     1     2     3     4     5
Passo 4
     1     2     3     4     5

Vetor classificado
     1     2     3     4     5

Pressione qualquer tecla para continuar. . .
```

```
C:\backupcida\LinguagemCPagina20082\AULA2deVETOR
Forneca os 5 elementos do vetor
Elemento [1]: 5
Elemento [2]: 4
Elemento [3]: 3
Elemento [4]: 2
Elemento [5]: 1

Vetor lido
     5     4     3     2     1

Passo 0
     4     3     2     1     5
Passo 1
     3     2     1     4     5
Passo 2
     2     1     3     4     5
Passo 3
     1     2     3     4     5
Passo 4
     1     2     3     4     5

Vetor classificado
     1     2     3     4     5

Pressione qualquer tecla para continuar. . .
```

# O método da bolha (*bubblesort*)

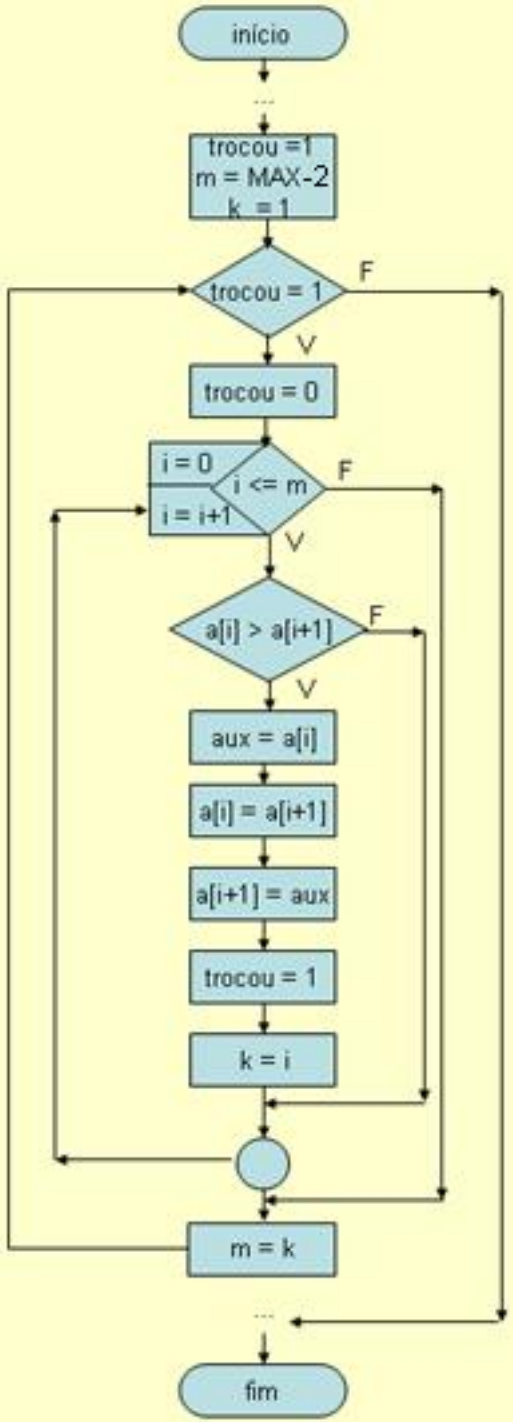
## Princípio de classificação

- Percorre-se o arranjo n vezes, sendo n um número menor ou igual ao número de elementos do arranjo.
- A cada vez que se percorre o arranjo, compara-se seus elementos dois a dois (os pares consecutivos ou adjacentes), trocando-os de posição se necessário.
- O número de posições do arranjo percorridas a cada iteração vai decrescendo à medida que o processo avança e que os valores vão sendo posicionados corretamente.
- Quando em uma dada passada não ocorre nenhuma troca entre elementos, o processo é encerrado.

# Bubblesort

Suponha que se deseja classificar em ordem crescente o seguinte arranjo de 5 elementos:
{28, 26, 30, 24, 25}

| | |
|---|---|
| **trocou** | 1 - houve troca durante uma varredura.<br>0 - não houve troca durante a varredura. |
| **m** | substitui MAX durante o processo de classificação e vai recuando de um (ou mais) a cada iteração. |
| **i** | variável de controle usada em *fors*. |
| **a [... ]** | arranjo a ser classificado. |
| **MAX** | número de elementos do vetor. |
| **aux** | variável auxiliar que viabiliza a troca de posição. |
| **k** | ao terminar uma varredura do arranjo, armazena a posição onde o ultimo valor trocado foi colocado. |




*Profas. Maria Aparecida M. Souto*
*Ivone M. Medero*
*Maria Aparecida Castro Livi*

**Suponha que se deseja classificar em ordem crescente o seguinte arranjo de 5 elementos: {28, 26, 30, 24, 25}**

## Primeira Iteração:

| | | | | | |
|---|---|---|---|---|---|
| 28 | 26 | 30 | 24 | 25 | compara par (28, 26) : troca |
| 26 | 28 | 30 | 24 | 25 | compara par (28, 30) : não troca |
| 26 | 28 | 30 | 24 | 25 | compara par (30, 24) : troca |
| 26 | 28 | 24 | 30 | 25 | compara par (30, 25) : troca |
| 26 | 28 | 24 | 25 | 30 | fim da primeira varredura |

(*Maior chave se encontra em sua posição definitiva!*)

**Todas as 5 posições do arranjo são processadas.**

**Suponha que se deseja classificar em ordem crescente o seguinte arranjo de 5 elementos: {28, 26, 30, 24, 25}**

## Segunda Iteração:

| | | | | | |
|---|---|---|---|---|---|
| 26 | 28 | 24 | 25 | 30 | compara par (26, 28) : não troca |
| 26 | 28 | 24 | 25 | 30 | compara par (28, 24) : troca |
| 26 | 24 | 28 | 25 | 30 | compara par (28, 25) : troca |
| 26 | 24 | 25 | 28 | 30 | (não precisa comparar) fim da segunda varredura |

**4 posições do arranjo são processadas.**

**Suponha que se deseja classificar em ordem crescente o seguinte arranjo de 5 elementos: {28, 26, 30, 24, 25}**

## Terceira Iteração:

| | | | | | |
|---|---|---|---|---|---|
| 26 | 24 | 25 | 28 | 30 | compara par (26, 24) : troca |
| 24 | 26 | 25 | 28 | 30 | compara par (26, 25) : troca |
| 24 | 25 | 26 | 28 | 30 | fim da terceira varredura |

Na próxima varredura (quarta varredura), nenhuma troca ocorrerá e a execução do algoritmo terminará.

**3 posições do arranjo são processadas.**

# O método da bolha (*bubblesort*): código

```
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
int main ( )
{
   int a [MAX], aux, i, m, k, trocou, passo = 0, j;
   system("color f1");
   printf ("\t>>> BUBBLESORT <<<\n");
   printf ("\t(em ordem crescente)\n\n");
   printf("Forneca os %d elementos do vetor\n", MAX);
   for (i=0;i<MAX;i++)
    {
      printf("Elemento [%d]: ", i + 1);
      scanf("%d", &a[i]);
    }
   printf("\nVetor lido\n");
   for (i=0;i<MAX;i++)
     printf("%6d", a[i]);
   printf("\n");
   ...
```

Cont.

# O método da bolha (*bubblesort*): código (cont.)

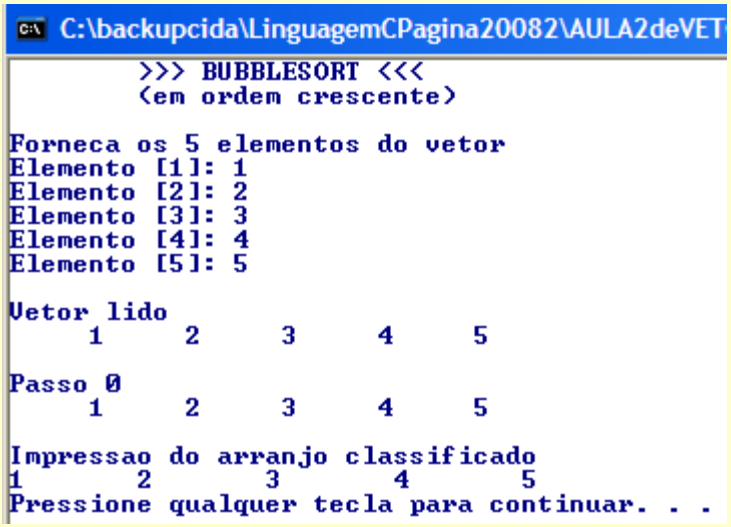
```
...
trocou = 1; m = (MAX-1); k = 1;
while (trocou)
      {
          trocou = 0;
          for (i = 0; i < m; i++)
                if  (a[ i ] >  a[ i + 1])
                {
                     aux = a [ i ];
                     a [ i ] = a [ i + 1];
                     a [ i + 1] = aux;
                     k = i;
                     trocou = 1;
                 }
           m = k;
           printf("\nPasso %d\n", passo);
           for (j=0;j<MAX;j++)
               printf("%6d", a[j]);
           passo++;
      }
...
```

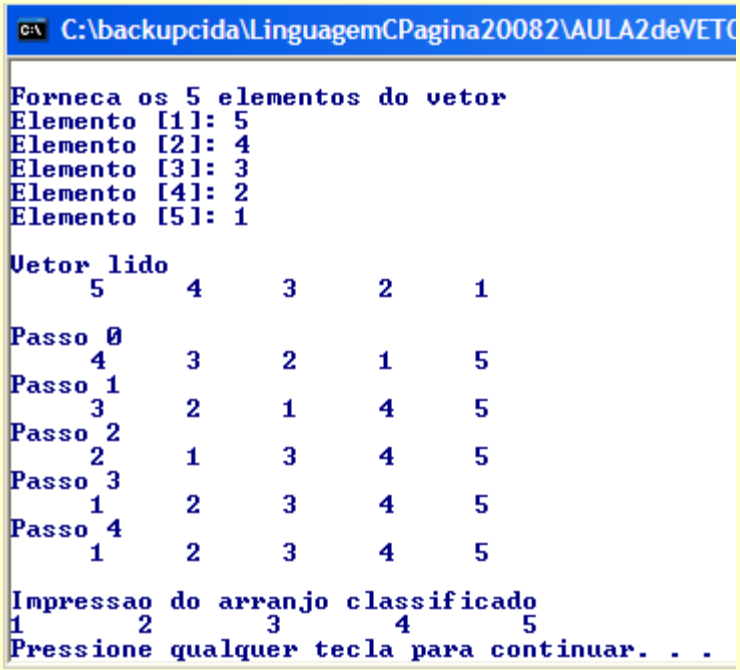
| | |
|---|---|
| **trocou** | **1** - houve troca durante uma varredura.<br>**0** - não houve troca durante a varredura. |
| **m** | substitui MAX durante o processo de classificação e vai recuando de um (ou mais) a cada iteração. |
| **i** | variável de controle usada nos *fors*. |
| **a [... ]** | arranjo a ser classificado. |
| **MAX** | número de elementos do vetor. |
| **aux** | variável auxiliar que viabiliza a troca de posição. |
| **k** | ao terminar uma varredura do arranjo, armazena a posição onde o ultimo valor trocado foi colocado. |

Cont.

# O método da bolha (*bubblesort*): código (cont.)

```
   ...
   printf ("\n\nImpressao do arranjo classificado\n");
   for (i = 0; i <= MAX - 1; i++)
         printf ("%d\t", a [ i ]);
   printf("\n");
   system ("pause");
   return 0;
}
```



# Pesquisa de Dados

- Os computadores são especialmente adequados para realizar consultas em uma grande quantidade de itens de dados armazenados em um *arranjo* ou tabela
- Por exemplo:
  - Pode ser necessário determinar se uma tabela contém um determinado item de dado...
- O processo de procurar um item de dado em uma tabela é chamado de ***pesquisa***

# Algoritmos de Pesquisa de Dados

- Duas técnicas bastante conhecidas, utilizadas para pesquisar dados em arranjos são:
  - Pesquisa Sequencial;
  - Pesquisa Binária.

# Pesquisa Sequencial

**Pré-condição:** não existe.

O arranjo pode conter valores ordenados ou desordenados.

## Técnica

Percorre-se o arranjo, posição a posição, desde seu início.

O processo de busca pára quando:

- o valor procurado é encontrado → pesquisa termina com sucesso ou
- o final do arranjo é atingido → o valor procurado não é encontrado.

# Pesquisa Sequencial

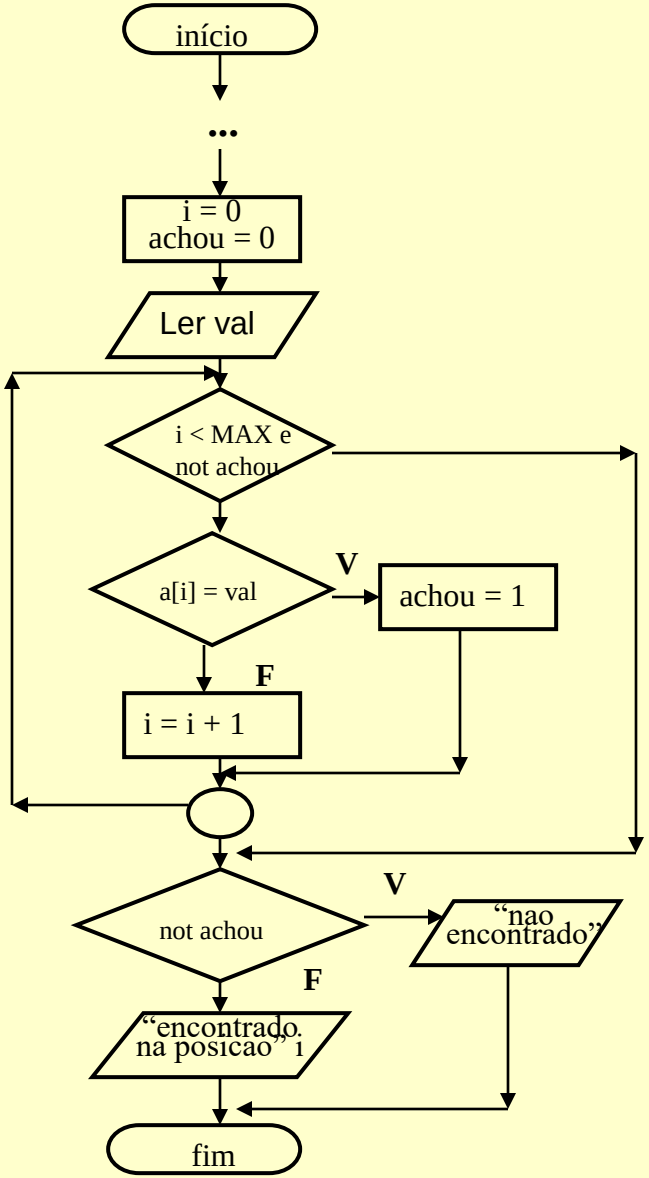
- Suponha que se deseje localizar os valores: 24 e 35 no arranjo abaixo:

| 28 | 26 | 30 | 24 | 25 |
|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 |

# Pesquisa Sequencial

Suponha que se deseja localizar um determinado valor em um arranjo de 5 elementos:
{28, 26, 30, 24, 25}

| | |
|---|---|
| **i** | variável de controle. Também identifica posição do elemento, se ele for encontrado. |
| **achou** | 1 - valor procurado é encontrado. |
| | 0 - valor procurado NÃO é encontrado. |
| **val** | valor que se deseja localizar. |
| **a [... ]** | arranjo com os valores. |



```
//pesquisa sequencial sobre o arranjo a
#include <stdio.h>
#include <stdlib.h>
#define MAX 5
int main ( )
{
   int a [ MAX ] = {28, 26, 30, 24, 25};
   int i, val, achou;
   printf ("\t>>> PESQUISA SEQUENCIAL <<<\n\n");
   printf ("Imprimindo o arranjo com os dados\n\n");
   for (i = 0; i < MAX; i++)
         printf ("%d\t", a [ i ]);
   printf ("\n\nForneca valor procurado: ");
   scanf ("%d", &val);
   ...
```

Cont.

```
    ...
    printf ("\nPesquisando . . .\n\n");
    i = 0; achou = 0;
    while (i < MAX && !achou)
       if (a [ i ] == val)
             achou = 1;
        else
             i++;
    if (!achou)
       printf ("Valor %d nao encontrado!!!\n\n", val);
    else
       printf ("O valor %d foi encontrado na posicao %d\n\n", val, i);
    system ("pause");
    return 0;
}
```

# Pesquisa Binária

**Pré-condição:**

o arranjo tem que estar ordenado.

**Técnica:**

A pesquisa binária, considerando que o arranjo está ordenado, divide sucessivamente o espaço de busca em metades até localizar o elemento buscado ou determinar que ele não existe no arranjo.

# Pesquisa Binária (cont.)

**Processo:**

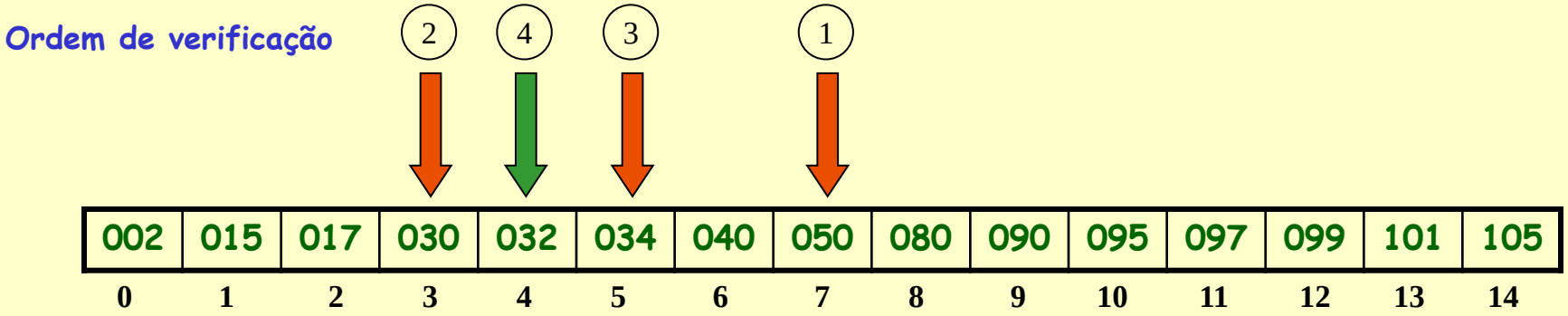
Inicia-se a pesquisa binária operando sobre o arranjo na sua totalidade. Determina-se seu ponto médio e compara-se o valor buscado com o existente no ponto médio.

Se a igualdade se verifica, a busca está encerrada, senão, determina-se em que metade (superior ou inferior) do arranjo o elemento buscado pode estar e ajusta-se os limites do espaço de busca para essa metade.

Determina-se novamente o ponto médio do novo espaço de busca e refaz-se o processo de comparação, com o ajuste do espaço de busca até que o valor buscado seja encontrado ou que se determine que ele não se encontra no arranjo.

# Pesquisa Binária

- Vamos supor que se deseja buscar o valor **32** no seguinte arranjo:



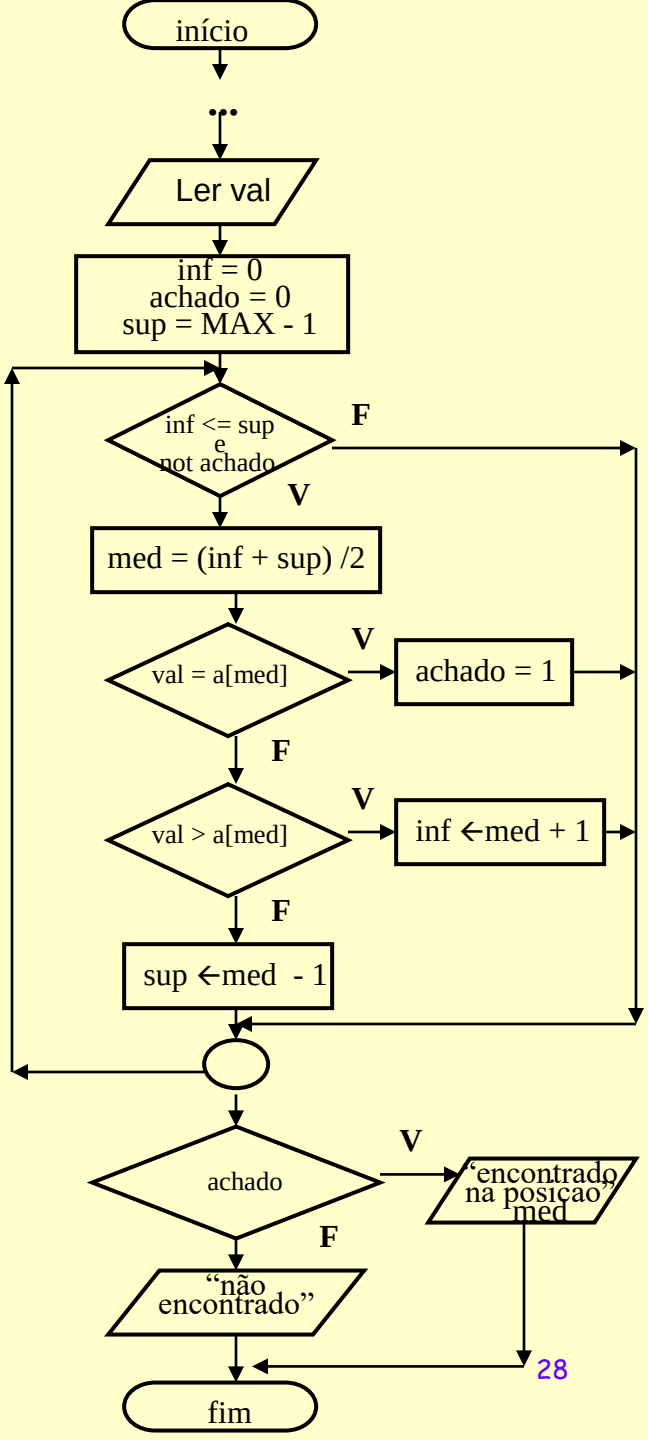

| | inf | sup | med |
|---|---|---|---|
| 1 | 0 | 14 | 7 |
| 2 | 0 | 7 | 3 |
| 3 | 3 | 7 | 5 |
| 4 | 3 | 5 | 4 |

**Achou!**

# Pesquisa Binária

Suponha que se deseja localizar um determinado valor em um arranjo de 15 elementos, conforme mostrado no slide anterior.

| | |
|---|---|
| **val** | **valor que se deseja localizar.** |
| **e** | **contém o endereço (ou posição no arranjo) onde o elemento foi localizado ou, -1 se o elemento não foi localizado.** |
| **inf** | **posição INFERIOR do intervalo sendo analisado.** |
| **sup** | **posição SUPERIOR do intervalo sendo analisado.** |
| **med** | **posição média do arranjo.Também identifica posição do elemento, se ele for encontrado.** |



```
//pesquisa binária sobre o arranjo a
#include <stdio.h>
#include <stdlib.h>
#define MAX 15
int main ( )
{
   int a[MAX] =
      {2, 15, 17, 30, 32, 34, 40, 50,
       80, 90, 95, 97, 99, 101, 105};
   int i, inf, sup, med, val, achado;
   system ("color f1");
   printf ("\t>>> PESQUISA BINARIA <<<\n\n");
   printf ("Imprimindo o arranjo classificado\n\n");
   for (i = 0; i < MAX; i++)
          printf ("%d\t", a[i]);
   printf ("\n\nForneca valor procurado: ");
   scanf ("%d", &val);
   ...
```

Cont.

```
   ...
   printf ("\nPesquisando . . .\n\n");
   inf = 0; sup = MAX - 1; achado = 0;
   while (inf <= sup && (!achado))
      {
         med = (inf + sup)/2;
         if (val == a [med] )
             achado = 1;
         else
             if (val > a [med])
                inf = med + 1;
             else
                sup = med - 1;
       }
   if (achado)
      printf ("Valor %d encontrado na posicao %d\n\n", val, med);
   else
      printf ("Valor %d nao encontrado !!!\n\n", val);
   system ("pause");
   return 0;
}
```