ALGORITMOS E PROGRAMAÇÃO DE COMPUTADORES

Disciplina: 113476

Profa. Carla Denise Castanho

Universidade de Brasília – UnB
Instituto de Ciências Exatas – IE
Departamento de Ciência da Computação – CIC

7. ORDENAÇÃO E PESQUISA

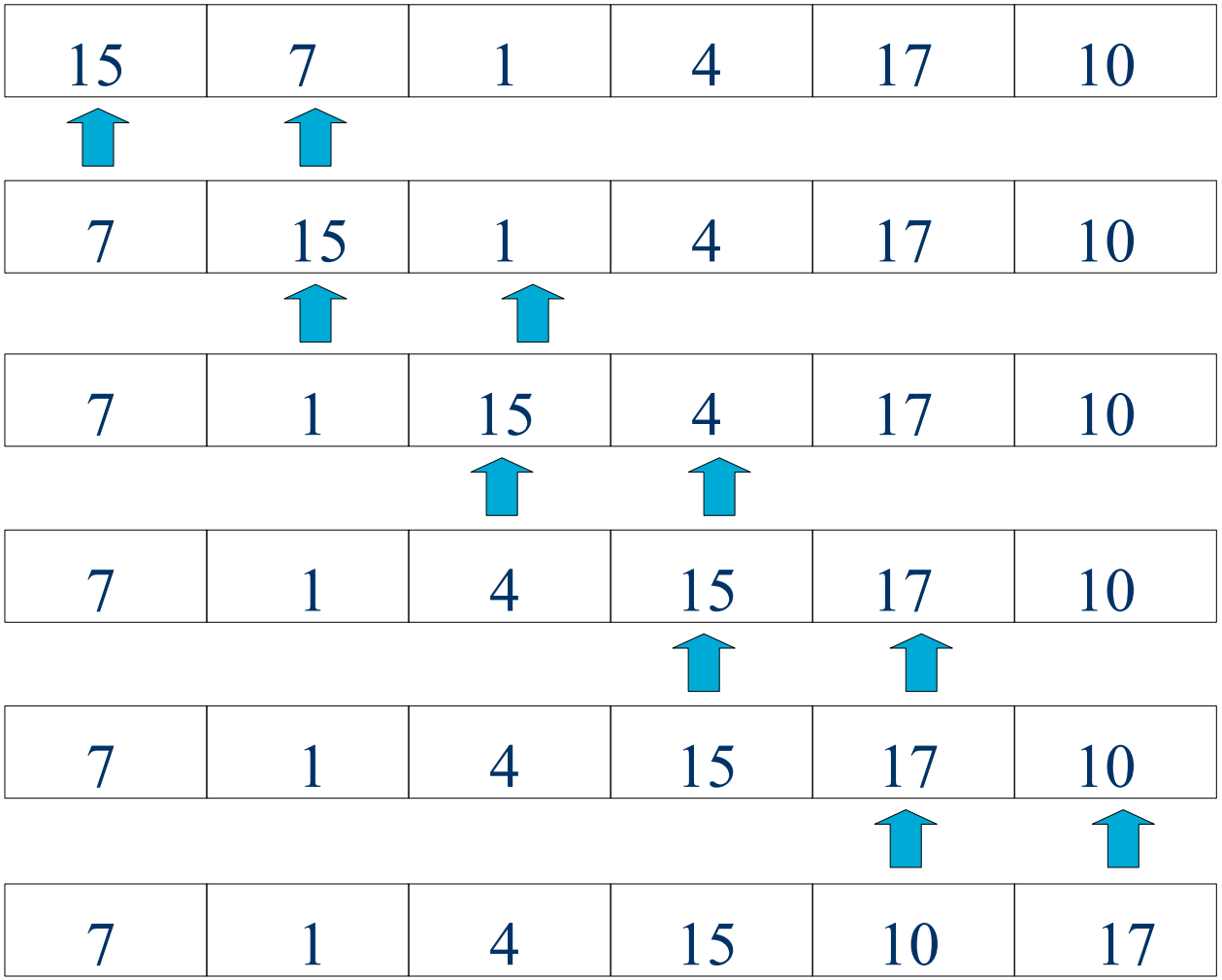
Ordenação e Pesquisa

- Ordenação:
  - A ordenação consiste em dispor elementos em uma certa sequência, seguindo algum critério para isso. Por exemplo, a ordenação alfabética para dados literais, ou crescente e decrescente para dados numéricos.
  - Para isso temos muitos métodos conhecidos, dentre eles o InsertionSort, ShellSort, BubbleSort, HeapSort, MergeSort , QuickSort, etc...
  - Neste capítulo trataremos especificamente do método **BubbleSort**.
- Pesquisa:
  - Já a pesquisa consiste na verificação da existência de um valor dentro de um conjunto de dados.
  - Trataremos neste capítulo da **pesquisa sequencial** e da **pesquisa binária.**

Ordenação

- O método **BubbleSort** não é de forma alguma o mais eficiente para classificar uma grande quantidade de dados, mas ele é muito simples, e o utilizaremos por motivos didáticos.
- Este método consiste em percorrer todo o vetor, comparando, em cada posição, os elementos vizinhos entre si. Caso estejam fora da ordem (conforme o critério em questão), eles trocam de posição. Procede-se assim até o final do vetor.
- O algoritmo repete esse laço até que o vetor seja percorrido e não haja mais nenhuma troca de posição entre os elementos consecutivos do vetor.
- Uma ilustração do método e o algoritmo é apresentado a seguir.

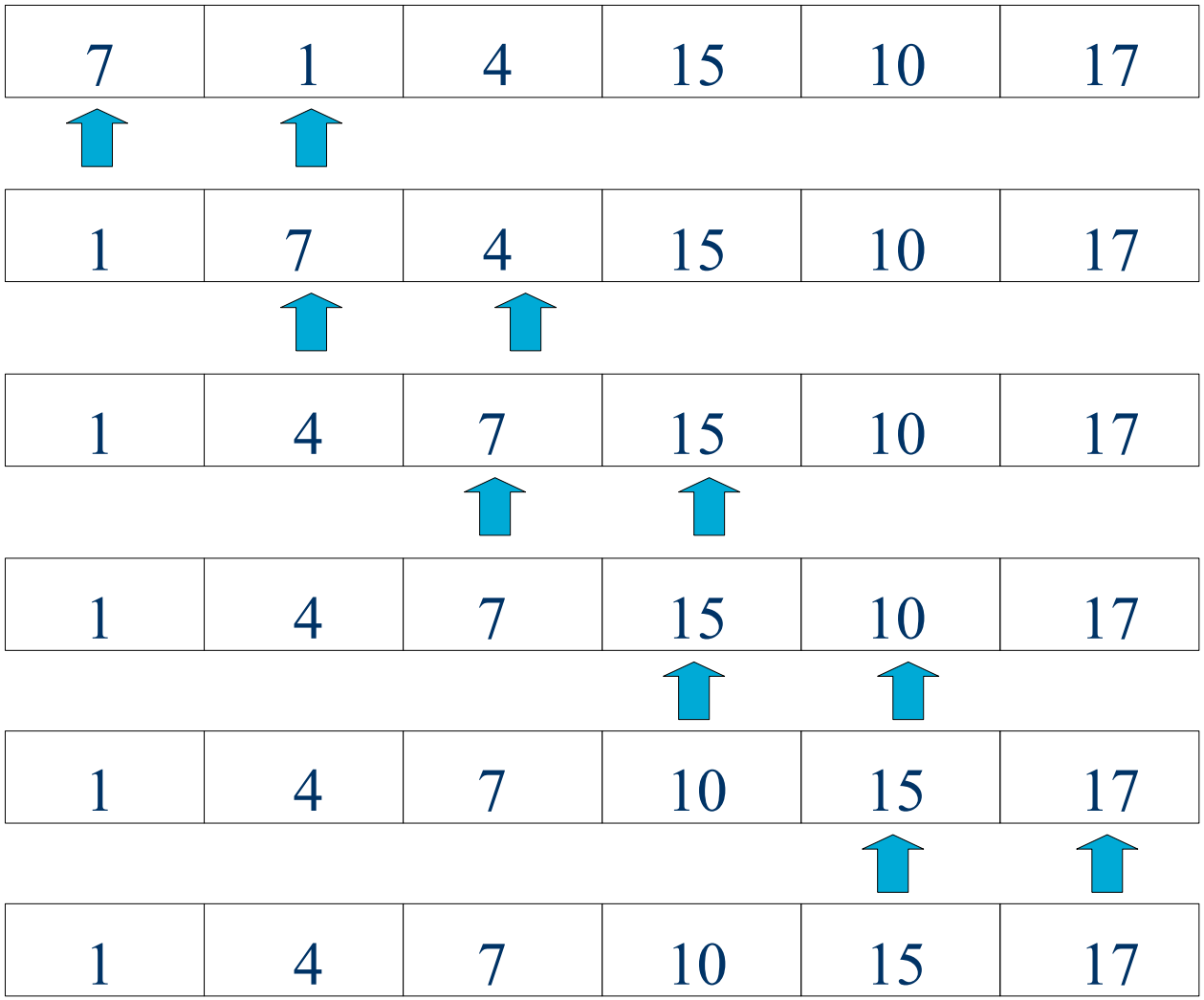
BubbleSort


15 7 1 4 17 10
7 15 1 4 17 10
7 1 15 4 17 10
7 1 4 15 17 10
7 1 4 15 17 10
7 1 4 15 10 17

BubbleSort

- O vetor está agora próximo de uma ordenação, mas ainda não da ordenação desejada.
- Isso indica que devemos repetir o processo mais vezes até que o vetor esteja ordenado.
- Executando mais uma vez o trecho de algoritmo...

BubbleSort


7 1 4 15 10 17
1 7 4 15 10 17
1 4 7 15 10 17
1 4 7 15 10 17
1 4 7 10 15 17
1 4 7 10 15 17

BubbleSort

- O número máximo de execuções do trecho do algoritmo para que o vetor fique ordenado é N-1 vezes, onde N é o número de elementos do vetor.

Parte do algoritmo de ordenação - troca dos elementos do vetor. Para um exemplo onde N = 6.

```
Para k ← 0 até 4 faça
    Para i ← 0 até 4 faça
        Se vetor[i] > vetor[i + 1] então
            aux ← vetor[i];
            vetor[i] ← vetor[i + 1];
            vetor[i + 1] ← aux;
        Fim-Se
    Fim-Para
Fim-Para
```

BubbleSort

- O número máximo de execuções do trecho do algoritmo para que o vetor fique ordenado é N-1 vezes, onde N é o número de elementos do vetor.

Parte do algoritmo de ordenação - troca dos elementos do vetor. Para um exemplo onde N = 6.

```
Para k ← 0 até 4 faça
    Para i ← 0 até 4 faça
        Se vetor[i] > vetor[i + 1] então
            aux ← vetor[i];
            vetor[i] ← vetor[i + 1];
            vetor[i + 1] ← aux;
        Fim-Se
    Fim-Para
Fim-Para
```

Troca vetor[i] e vetor[i+1]

BubbleSort

- O número máximo de execuções do trecho do algoritmo para que o vetor fique ordenado é N-1 vezes, onde N é o número de elementos do vetor.

Parte do algoritmo de ordenação - troca dos elementos do vetor. Para um exemplo onde N = 6.

```
Para k ← 0 até 4 faça
    Para i ← 0 até 4 faça
        Se vetor[i] > vetor[i + 1] então
            aux ← vetor[i];
            vetor[i] ← vetor[i + 1];
            vetor[i + 1] ← aux;
        Fim-Se
    Fim-Para
Fim-Para
```

Troca vetor[i] e vetor[i+1]

Note que a variável i do loop de repetição só deve ir até 4, pois no final do loop ele testa o elemento i com o i+1, ou seja, o penúltimo com o último. Se o i for até 5, ele vai testar o último elemento com o próximo que não existe, e ocorrerá um ERRO!

BubbleSort

- É sempre necessário repetir o passo N-1 vezes?
  - No exemplo apresentado em apenas duas execuções do algoritmo o vetor já estava ordenado!
- Como controlar o número de vezes?
  - Se o vetor já estiver ordenado, não precisa repetir o passo mais uma vez. Como saber?
  - Se não houve trocas entre os elementos do vetor ao executar o trecho do algoritmo, então está ordenado.

BubbleSort

Algoritmo de Ordenação (em ordem crescente) - BubbleSort

```
Algoritmo BubbleSort
Variáveis
    vetor : vetor [6] de inteiros
    i, aux, fez_troca : inteiro
Início
    Para i ← 0 até 5 faça
        Escreva ("Informe o elemento ", i + 1, " do vetor:")
        Leia (vetor[i])
    Fim-Para
    fez_troca ← 1
    Enquanto (fez_troca = 1) faça
        fez_troca ← 0
        Para i ← 0 até 4 faça
            Se vetor[i] > vetor[i + 1] então
                aux ← vetor[i]
                vetor[i] ← vetor[i + 1]
                vetor[i + 1] ← aux
                fez_troca ← 1
            Fim-Se
        Fim-Para
    Fim-Enquanto
    Escreva ("Vetor ordenado:")
    Para i ← 0 até 5 faça
        Escreva (vetor[i])
    Fim-Para
Fim
```

BubbleSort

Algoritmo de Ordenação (em ordem crescente) - BubbleSort

```
Algoritmo BubbleSort
Variáveis
    vetor : vetor [6] de inteiros
    i, aux, fez_troca : inteiro
Início
    Para i ← 0 até 5 faça
        Escreva ("Informe o elemento ", i + 1, " do vetor:")
        Leia (vetor[i])
    Fim-Para
    fez_troca ← 1
    Enquanto (fez_troca = 1) faça
        fez_troca ← 0
        Para i ← 0 até 4 faça
            Se vetor[i] > vetor[i + 1] então
                aux ← vetor[i]
                vetor[i] ← vetor[i + 1]
                vetor[i + 1] ← aux
                fez_troca ← 1
            Fim-Se
        Fim-Para
    Fim-Enquanto
    Escreva ("Vetor ordenado:")
    Para i ← 0 até 5 faça
        Escreva (vetor[i])
    Fim-Para
Fim
```

O loop mais externo só termina quando não houve nenhuma troca, *i.e.*, fez_troca = 0. Nesse caso, o vetor já está ordenado.

Quando necessário, troca vetor[i] e vetor[i+1]. Sempre que fizer qualquer troca, atribui 1 à variável fez_troca.

BubbleSort

Programa em C do exemplo anterior

```c
#include <stdio.h>

int main () {
    int vetor[6], i, aux, fez_troca;
    for (i = 0; i < 6; i++) {
        printf("Informe o elemento %d do vetor:\n", i + 1);
        scanf("%d", &vetor[i]);
    }
    fez_troca = 1;
    while (fez_troca) {
        fez_troca = 0;
        for (i = 0; i < 5; i++) {
            if (vetor[i] > vetor[i + 1]) {
                aux = vetor[i];
                vetor[i] = vetor[i + 1];
                vetor[i + 1] = aux;
                fez_troca = 1;
            }
        }
    }
    printf("Vetor ordenado:\n");
    for (i = 0; i < 6; i++) {
        printf("%d\n", vetor[i]);
    }
    return 0;
}
```

Pesquisa Sequencial

- Se um conjunto não está ordenado segundo algum critério, o método mais simples de se buscar um elemento é a pesquisa **sequencial** ou **linear**.
- Verificamos sequencialmente (ou seja, um após o outro) cada elemento. Se encontramos o valor desejado, então a pesquisa foi **bem sucedida**.
- Caso todos os elementos do conjunto sejam verificados e o elemento desejado não esteja entre eles, dizemos que a pesquisa foi **mal sucedida**.

Pesquisa Sequencial

Lê um conjunto de 10 números e faz uma pesquisa sequencial.

```
Algoritmo PesquisaSequencial
Variáveis
    conjunto : vetor [10] de inteiros
    i, valor, achou : inteiro
Início
    Para i ← 0 até 9 faça
        Escreva ("Informe o elemento ", i + 1, " do conjunto:")
        Leia (conjunto[i])
    Fim-Para
    Escreva ("Informe o valor a ser procurado:")
    Leia (valor)
    i ← 0
    achou ← 0
    Enquanto (i < 10) e (achou = 0) faça
        Se conjunto[i] = valor então
            achou ← 1
        Senão
            i ← i + 1
        Fim-Se
    Fim-Enquanto
    Se achou = 1 então
        Escreva (valor, " foi encontrado na posição ", i + 1, ".")
    Senão
        Escreva (valor, " não foi encontrado.")
    Fim-Se
Fim
```

Pesquisa Sequencial

Programa em C do exemplo anterior

```c
#include <stdio.h>

int main () {
    int conjunto[10], i, valor, achou;

    for (i = 0; i < 10; i++) {
        printf("Informe o elemento %d do conjunto:\n", i + 1);
        scanf("%d", &conjunto[i]);
    }
    printf("Informe o valor a ser procurado:\n");
    scanf("%d", &valor);

    i = 0;
    achou = 0;
    while ((i < 10) && (!achou)) {
        if (conjunto[i] == valor)
            achou = 1;
        else
            i++;
    }
    if (achou)
        printf("%d encontrado na posicao %d.\n", valor, i + 1);
    else
        printf("%d nao encontrado.\n", valor);

    return 0;
}
```

Pesquisa Binária

- É fácil ver que, quanto maior o vetor, menos eficaz a pesquisa linear se torna, pois:
  - no **pior caso**, teremos que pesquisar **todos os elementos**;
  - e **na média**, teremos que pesquisar **pelo menos a metade** deles para encontrar o elemento desejado.
- Se tivermos um vetor **ordenado**, podemos usar a **pesquisa binária**.
- Ela consegue eliminar sempre a **metade** dos elementos do vetor **em cada comparação**.

Pesquisa Binária

- A pesquisa binária utiliza a técnica de "**dividir e conquistar**":
  - primeiro, testamos se o valor é igual ao elemento que está no **meio** do vetor;
  - se este elemento (do meio) for **maior** que o valor procurado, passamos a buscar apenas na **primeira metade do vetor**;
  - caso contrário (o elemento do meio é **menor**), buscamos apenas na **segunda metade do vetor**.
- Esse procedimento é repetido até que o elemento seja encontrado ou não haja mais elementos a testar.

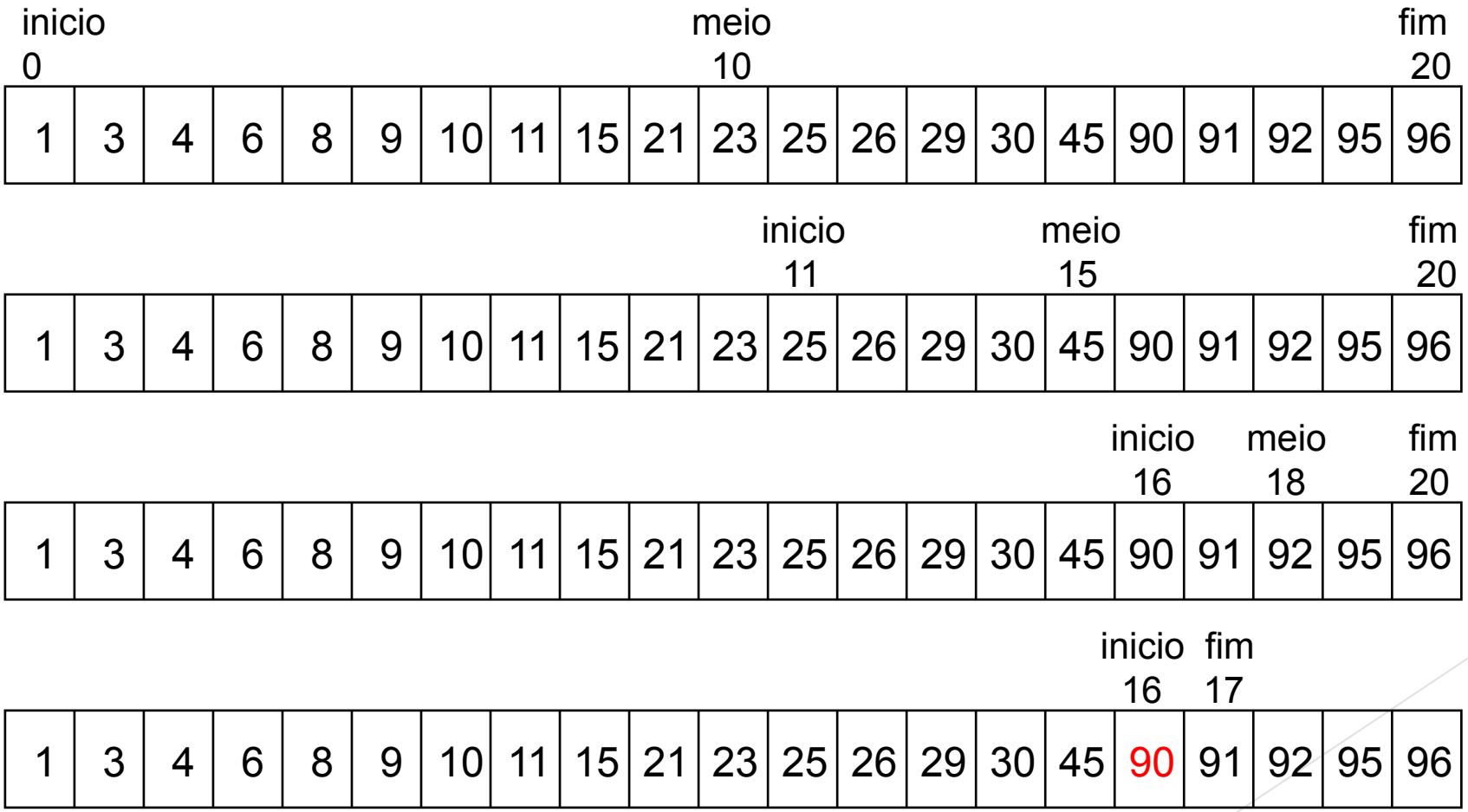
Pesquisa Binária

- A pesquisa binária utiliza a técnica de "**dividir e conquistar**":
  - primeiro, testamos se o elemento procurado é **menor** que o elemento do meio do vetor. Se for o caso, então passamos a buscar apenas na **primeira metade do vetor;**
  - Senão, testamos se o elemento procurado é **maior** que o elemento do meio do vetor. Se for o caso, então passamos a buscar apenas na **segunda metade do vetor;**
  - caso contrário o valor procurado é igual ao elemento que está no **meio** do vetor.
  - Esse procedimento é repetido até que o elemento seja encontrado ou não haja mais elementos a testar.

Note que, a cada teste, descartamos uma das metades do (sub)vetor pesquisado.

Pesquisa Binária - Exemplo

- Buscar o número 90 em um vetor (ordenado) contendo 21 números.



Pesquisa Binária

Lê um conjunto ordenado de 15 números e faz uma pesquisa binária.

```
Algoritmo PesquisaBinaria
Variáveis
    conjunto[15], inicio, fim, meio, valor, achou : inteiro
Início
    /* leitura do vetor foi omitida */
    Escreva ("Informe o valor a ser procurado:")
    Leia (valor)
    inicio ← 0
    fim ← 14
    achou ← 0
    Enquanto (inicio ≤ fim) e (achou = 0) faça
        meio ← int((inicio + fim) / 2)
        Se valor < conjunto[meio] então
            fim ← meio - 1
        Senão Se valor > conjunto[meio] então
            inicio = meio + 1
        Senão
            achou ← 1
        Fim-Se
    Fim-Enquanto
    Se achou = 1 então
        Escreva (valor, " foi encontrado na posição ", meio + 1, ".")
    Senão
        Escreva (valor, " não foi encontrado.")
    Fim-Se
Fim
```

Pesquisa Binária

Lê um conjunto ordenado de 15 números e faz uma pesquisa binária.

```
Algoritmo PesquisaBinaria
Variáveis
    conjunto[15], inicio, fim, meio, valor, achou : inteiro
Início
    /* leitura do vetor foi omitida */
    Escreva ("Informe o valor a ser procurado:")
    Leia (valor)
    inicio ← 0
    fim ← 14
    achou ← 0
    Enquanto (inicio ≤ fim) e (achou = 0) faça
        meio ← int((inicio + fim) / 2)
        Se valor < conjunto[meio] então
            fim ← meio - 1
        Senão Se valor > conjunto[meio] então
            inicio = meio + 1
        Senão
            achou ← 1
        Fim-Se
    Fim-Enquanto
    Se achou = 1 então
        Escreva (valor, " foi encontrado na posição ", meio + 1, ".")
    Senão
        Escreva (valor, " não foi encontrado.")
    Fim-Se
Fim
```

IMPORTANTE!
O vetor de entrada DEVE estar ORDENADO !!!

Pesquisa Binária

Programa em C do exemplo anterior

```c
#include <stdio.h>

int main () {
    int conjunto[15], inicio, fim, meio, valor, achou;
    /* leitura do vetor foi omitida */
    printf("Informe o valor a ser procurado:\n");
    scanf("%d", &valor);

    inicio = 0;
    fim = 14;
    achou = 0;
    while ((inicio <= fim) && (!achou)) {
        meio = (inicio + fim) / 2;
        if (valor < conjunto[meio])
            fim = meio - 1;
        else {
            if (valor > conjunto[meio])
                inicio = meio + 1;
            else
                achou = 1;
        }
    }
    if (achou)
        printf("%d encontrado na posicao %d.\n", valor, meio + 1);
    else
        printf("%d nao encontrado.\n", valor);

    return 0;
}
```

Pesquisa Binária

Valor de Pesquisa: 25														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28
								16	18	20	22	24	26	28
												24	26	28
												24		

Valor de Pesquisa: 8														
0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
0	2	4	6	8	10	12	14	16	18	20	22	24	26	28
0	2	4	6	8	10	12								
				8	10	12								
				8										

Note que em ambos os casos a pesquisa sequencial levaria a muito mais comparações, no primeiro caso 13 comparações até se chegar no 24, e no segundo 5 até se chegar no 8.

Exercícios

- Fazer os 3 programas a seguir em linguagem C e submeter nos links correspondentes no Moodle:
  - **BubbleSort** - Implementar um programa em C com 3 funções chamadas na main: uma para ler um vetor de 10 inteiros, outra para ordenar este vetor utilizando o Método BubbleSort e uma última função para mostrar o vetor ordenado na tela.
  - **Pesquisa Sequencial** - Implementar um programa em C com 2 funções chamadas na main: uma para ler um vetor de 10 inteiros, outra para fazer a pesquisa sequêncial de um valor neste vetor de inteiros. O valor a ser procurado deve ser lido na main e repassado à função juntamente com o vetor. A função deve retornar o índice onde foi encontrado o elemento ou -1 caso a busca não tenha tido sucesso. As mensagens correspondentes devem ser mostradas na main após testar o retorno da função.
  - **Pesquisa Binária** - Implementar um programa em C com 2 funções chamadas na main: uma para ler um vetor de 10 inteiros, outra para fazer a pesquisa binária de um valor neste vetor de inteiros. O valor a ser procurado deve ser lido na main e repassado à função juntamente com o vetor. A função deve retornar o índice onde foi encontrado o elemento ou -1 caso a busca não tenha tido sucesso. As mensagens correspondentes devem ser mostradas na main após testar o retorno da função. **ATENÇÃO: NESTE PROGRAMA O VETOR DE ENTRADA DEVE SER DIGITADO ORDENADO!!!!**