

# Estrutura de Dados

## 2º semestre de 2025

### Pesquisa Binária

Tema #05

Professor Marcelo Eustáquio



# Pesquisa binária

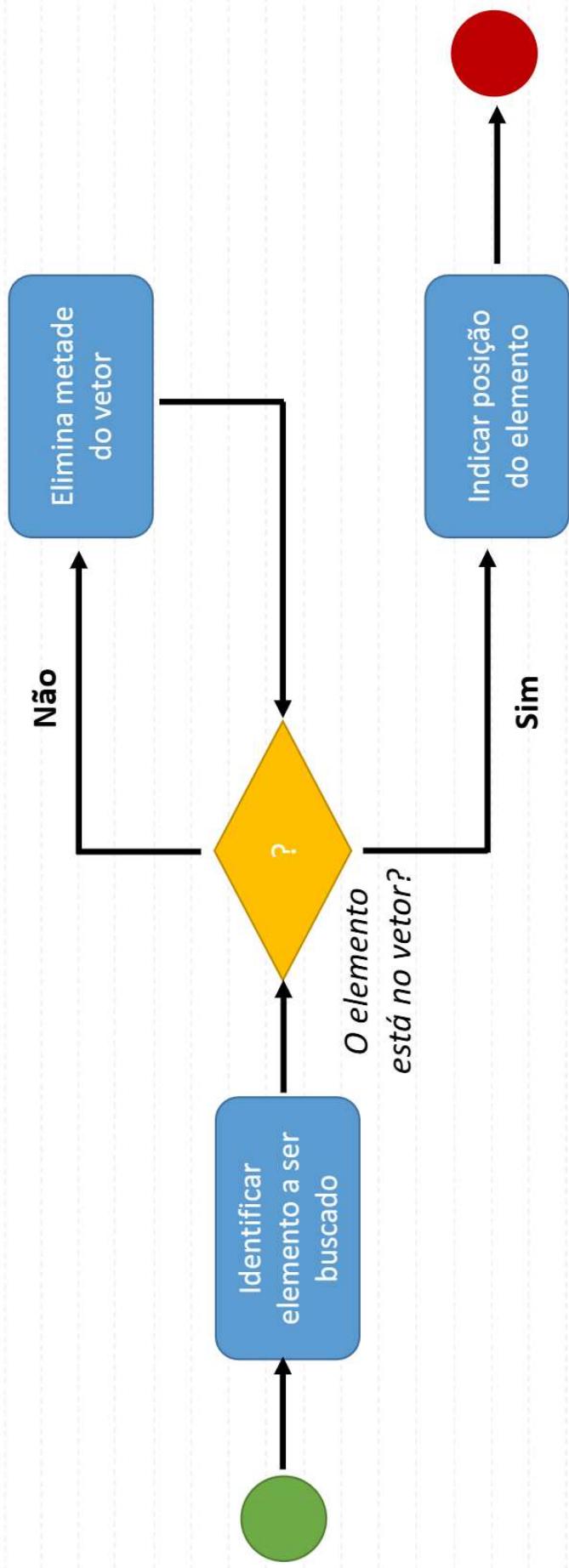
A **pesquisa binária** é um algoritmo eficiente para encontrar um elemento específico em um vetor que funciona dividindo repetidamente ao meio, eliminando metade dos elementos do conjunto a cada passo, a cada iteração. Apresenta o seguinte funcionamento:

- 1º Compare o elemento a ser pesquisado com o elemento central do vetor.
- 2º Se o elemento do meio for igual ao procurado, a busca termina com sucesso.
- 3º Se o elemento do meio for maior, repita a busca na metade esquerda.
- 3º Se o elemento do meio for menor, repita a busca na metade direita.

Pesquisa binária é um algoritmo de pesquisa mais rápido do que a busca linear, especialmente se usado para grandes conjuntos de dados.

# Pesquisa binária

A **pesquisa binária** é um algoritmo eficiente para encontrar um elemento específico em um vetor que funciona dividindo repetidamente ao meio, eliminando metade dos elementos do conjunto a cada passo, a cada iteração.





Para que o algoritmo possa ser aplicado, é  
condição necessária que o  
**vetor esteja ordenado.**

# Pesquisa binária



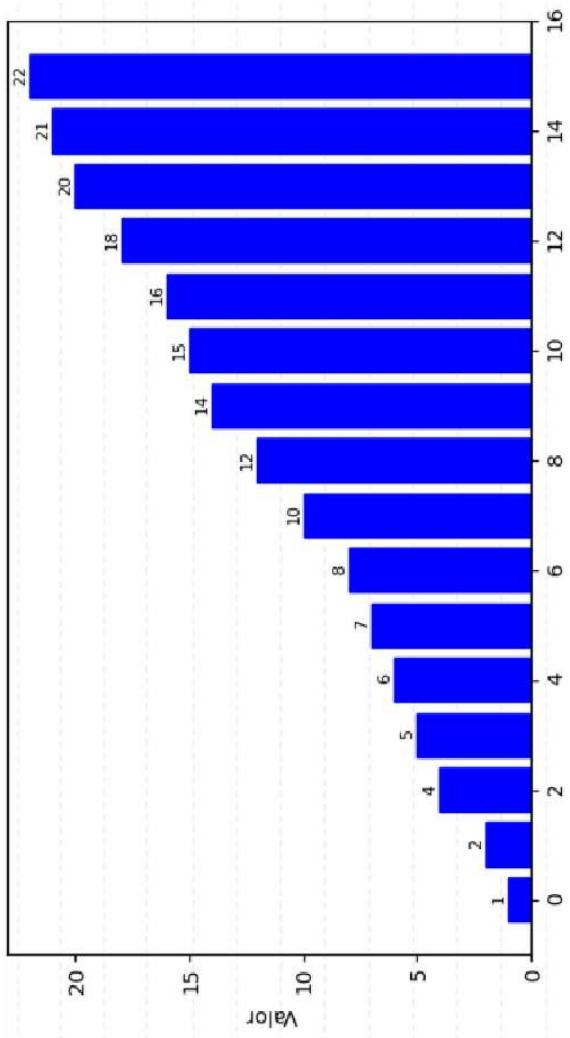
Para poder ser aplicado, o algoritmo de pesquisa binária exige que os elementos do array:

- A) sejam números;
- B) estejam ordenados;
- C) estejam representados em base múltipla de 2;
- D) ocupem somente as posições pares;
- E) não sejam repetidos.

**Gabarito:** Alternativa B. O algoritmo de pesquisa binária funciona dividindo repetidamente o array ao meio, comparando o elemento central com o valor buscado. Para que isso seja possível, é obrigatório que o array esteja ordenado.

# Pesquisa binária

Considere o conjunto (já ordenado) cujos elementos são **1, 2, 4, 5, 6, 7, 8, 10, 12, 14, 15, 16, 18, 20, 21, 22**. A seguir está uma representação gráfica dos elementos desse conjunto.



Mostre, nesse conjunto, como se dá o processo de pesquisa pelas chaves **12, 21** e **35**.

# Pesquisa binária

Pesquisando pelos elementos **12** no conjunto **1, 2, 4, 5, 6, 7, 8, 10, 12, 14, 15, 16, 18, 20, 21, 22**. No desenvolvimento do algoritmo, indique os elementos e as posições dos elementos que foram visitados.



Determine o custo computacional (em número de **ifs** executados) para se determinar a posição do elemento **12** no conjunto de dados indicados.

# Pesquisa binária

Pesquisando pelos elementos **21** no conjunto **1, 2, 4, 5, 6, 7, 8, 10, 12, 14, 15, 16, 18, 20, 21, 22**. No desenvolvimento do algoritmo, indique os elementos e as posições dos elementos que foram visitados.



Determine o custo computacional (em número de **ifs** executados) para se determinar a posição do elemento **21** no conjunto de dados indicados.

# Pesquisa binária

Pesquisando pelos elementos **35** no conjunto **1, 2, 4, 5, 6, 7, 8, 10, 12, 14, 15, 16, 18, 20, 21, 22**. No desenvolvimento do algoritmo, indique os elementos e as posições dos elementos que foram visitados.



Determine o **custo computacional** (em número de **ifs** executados) para se descobrir que o elemento **35** não está no conjunto de dados propostos.

# Pesquisa binária



O método de busca mais rápido, em qualquer tipo de arquivo, denomina-se pesquisa binária.

**Gabarito:** Item ERRADO.  
*A pesquisa binária apresenta excelente desempenho em conjuntos ordenados, mas não pode ser aplicada em arquivos desordenados sem prévia ordenação.*

# Pesquisa binária

A cada iteração, o algoritmo compara o elemento central do vetor com a chave de comparação e, caso sejam diferentes, considera apenas uma das partições para efeitos de pesquisa, na próxima iteração. Em C, esta é uma possível implementação:

```
int main() {  
  
    int Numeros[] = {1, 2, 4, 5, 6, 7, 8, 10, 12, 14, 16, 18, 20, 21, 22};  
    int Tamanho = sizeof(numeros)/sizeof(int);  
    int Elemento = 10;  
    int Posicao = PesquisaBinaria(Numeros, Tamanho, Elemento);  
  
    if(Posicao != ERRO) printf("O elemento %d está na posição %d.\n", Elemento, Posicao);  
    else printf("O elemento %d não está no vetor.\n", Elemento);  
    return 0;  
}
```

# Pesquisa binária

A cada iteração, o algoritmo compara o elemento central do vetor com a chave de comparação e, caso sejam diferentes, considera apenas uma das partições para efeitos de pesquisa, na próxima iteração (...)

```
int PesquisaBinaria(int *Vetor, int N, int Chave){  
  
    int Esquerda = 0;  
    int Direita = N - 1;  
  
    while(Esquerda <= Direita) {  
        int Meio = (Esquerda + Direita) / 2;  
        if (Vetor[Meio] == Chave) return Meio;  
        else if (Vetor[Meio] < Chave) Esquerda = Meio + 1;  
        else Direita = Meio - 1;  
    }  
  
    return ErrO;  
}
```

# Pesquisa binária

A cada iteração, o algoritmo compara o elemento central do vetor com a chave de comparação e, caso sejam diferentes, considera apenas uma das partições para efeitos de pesquisa, na próxima iteração. A seguir, temos a versão recursiva do algoritmo:

```
int PBRecursivo (int *Vetor, int Esquerda, int Direita, int Chave){  
  
    if (Esquerda > Direita) return ERRO;  
    int Meio = (Esquerda + Direita) / 2;  
  
    if (Vetor[Meio] == Chave) return Meio;  
    else if (Vetor[Meio] < Chave) return PBRecursivo (Vetor, Meio + 1, Direita, Chave);  
    else return PBRecursivo (Vetor, Esquerda, Meio - 1, Chave);  
}
```

# Pesquisa binária

Em relação à complexidade de tempo, são características da busca binária:

- |   |               |   |
|---|---------------|---|
| 1 | Eficiência    | <b>O(log n)</b> é muito mais rápido do que a busca linear, cuja complexidade é $O(n)$ . |
| 2 | Simplicidade  | A lógica é clara e direta.  |
| 3 | Pré-requisito | O vetor deve estar ordenado antes de aplicar a pesquisa binária.                        |
| 4 | Recursividade | A versão recursiva pode consumir mais memória devido à pilha de chamadas.               |

**Conclusão:**

A eficiência torna a busca binária a abordagem preferencial em muitas aplicações práticas.

# Pesquisa binária

A pesquisa binária é eficiente para encontrar um elemento em um vetor ordenado. Sua complexidade de tempo é  $O(\log n)$ , onde  $n$  é o número de elementos no vetor.

## Demonstração:

*1<sup>a</sup> iteração:* determinar termo central e pesquisar elemento procurado em  $n/2$  elementos.

*2<sup>a</sup> iteração:* buscar termo central da partição e pesquisar chave nos  $n/4$  elementos da partição.

*3<sup>a</sup> iteração:* repetir processo acima com  $n/8$  elementos da partição.

Ou seja, após  $k$  iterações, o tamanho do subconjunto a ser pesquisado é  $n/2^k$  e considerando que o algoritmo termina (para) quando esse é unitário, tem-se que  $n/2^k \leq 1$ . Logo:

$$\frac{n}{2^k} \leq 1 \Rightarrow n \leq 2^k \Rightarrow \log_2 n \leq \log_2 2^k \Rightarrow \log_2 n \leq k \cdot \log_2 2 \Rightarrow \log_2 n \leq k \Rightarrow k = \lceil \log_2 n \rceil$$

# Pesquisa binária

A pesquisa binária é eficiente para encontrar um elemento em um vetor ordenado. Sua complexidade de tempo é  $O(\log n)$ , onde  $n$  é o número de elementos no vetor.

**Exemplo:**

Comparação entre o número da iteração e a quantidade de elementos da partição.

Iteração	0	1	2	3	4
Tamanho	16	8	4	2	1

Ou seja, após  $k = 4$  iterações, o intervalo de busca é reduzindo a 1 elemento, gerando o final do processo de pesquisa. Observa que  $\log_2 16 = 4$ .

# Estrutura de Dados

## 2º semestre de 2025

### Pesquisa Binária

Questões propostas

Professor Marcelo Eustáquio



**Q01**

Julgue a afirmação a seguir, classificando-a como correta (C) ou errada (E):

Na pesquisa binária, realiza-se a varredura de uma estrutura de dados desde o seu início até o final dessa estrutura, ou até que uma informação desejada seja encontrada.

**Gabarito:** Item ERRADO. Isso descreve busca linear (varredura sequencial). A pesquisa binária exige vetor ordenado e compara com o elemento central, descartando metade do espaço a cada passo (não percorre do início ao fim).

**Q02**

Julgue a afirmação a seguir, classificando-a como correta (C) ou errada (E):

A busca binária é mais eficiente do que a busca sequencial, uma vez que naquela o vetor que contém o valor a ser pesquisado está sempre ordenado pela chave de busca.

**Gabarito:** Item CORRETO. A busca binária, aplicada sobre um vetor ordenado e de acesso aleatório, tem custo  $O(\log n)$ , enquanto a busca sequencial custa  $O(n)$ . Observação: ela só é mais eficiente se os dados já estiverem ordenados (ou houver muitas buscas); ordenar apenas para uma única busca pode anular a vantagem, e em listas ligadas a binária não se aplica.

**Q03**

Julgue a afirmação a seguir, classificando-a como correta (C) ou errada (E):

A pesquisa binária, em relação à pesquisa sequencial, possui a vantagem de executar menos comparações, em média, para encontrar a chave procurada em uma lista ordenada.

**Gabarito:** Item CORRETO. Em uma lista ordenada e com acesso aleatório, a busca binária faz, em média, cerca de  $\log_2 n$  comparações (máximo  $\lceil \log_2 n \rceil$ ), enquanto a sequencial faz  $(n+1)/2$  em média (e n no pior caso). Ex.: para  $n = 1024$ , a binária faz no máximo 10 comparações; e a sequencial cerca 512.5.

**Q04**

Pesquisar um valor que corresponda a um valor-chave em um vetor com 128 elementos, usando o algoritmo de pesquisa binária, requer no máximo:

- A) oito comparações.
- B) quatro comparações.
- C) cinco comparações.
- D) seis comparações.
- E) sete comparações.

**Gabarito:** Alternativa A. Com 7 comparações, as partições reduzem a tamanho unitário. A seguir, deve-se acrescentar 1 comparação para verificar se o elemento procurado está nessa partição unitária.

**Q05**

Considere o problema de pesquisar por um número em um array ordenado contendo dez números. Se for utilizado o método da pesquisa binária, qual é o menor número de comparações que permite concluir que um número não está presente no array?

- A) 4
- B) 5
- C) 2
- D) 3
- E) 6

**Gabarito:** Alternativa A.

**Q06**

Para poder ser aplicado, o algoritmo de pesquisa binária exige que os elementos do array:

- A) sejam números;
- B) estejam ordenados;
- C) estejam representados em base múltipla de 2;
- D) ocupem somente as posições pares;
- E) não sejam repetidos.

**Gabarito:** Alternativa B.

**Q07**

Julgue a afirmação a seguir, classificando-a em C (correta) ou E (errada):

O método de busca mais rápido, em qualquer tipo de arquivo, denomina-se pesquisa binária.

**Gabarito:** Item ERRADO. A pesquisa binária não é universalmente a mais rápida: exige dados ordenados e acesso aleatório (vetor/memória). Em muitas situações há métodos superiores ou mais adequados, como tabelas de hash (busca média  $O(1)$ ), árvores B/B+ para dados em disco (otimizam I/Os), ou índices específicos. Em listas ligadas ou dados não ordenados, a binária nem se aplica; resta, por exemplo, a busca linear.

**Q08**

Basicamente, existem dois métodos de pesquisa em um vetor de números, a Busca Linear e a Busca Binária. A Busca Binária é mais eficiente do que a Busca Linear, mas ela só funciona se o vetor estiver ordenado. Assinale a alternativa que indique a ordem de complexidade do pior caso da Busca Binária em um vetor de n números ordenados.

- A)  $O(n)$
- B)  $O(n \log n)$
- C)  $O(\log n)$
- D)  $O(1)$
- E)  $O(n^2)$

**Gabarito:** Alternativa C.

**Q09**

O número de comparações, para o pior caso, de uma pesquisa binária em uma estrutura sequencial ordenada com N elementos é:

- A)  $(\log_2 N) + 1$
- B)  $\log_2 N$
- C)  $\log_{10} N$
- D) N
- E)  $N \log N$

**Gabarito:** Alternativa A.

**Q10**

Busca ou pesquisa binária é um algoritmo de busca em vetores ordenados. Sobre o algoritmo de busca binária é correto afirmar:

- I - No pior caso tem complexidade  $O(\log n)$ .
- II - No melhor caso tem complexidade  $O(\log n)$ .
- III - No caso médio tem complexidade  $O(1)$ .
- IV - No melhor caso tem complexidade  $O(n)$ .

Está(ão) correta(s)

- A) Apenas I.
- B) Apenas II e III.
- C) Apenas III e IV.
- D) Apenas II e IV.
- E) I, II, III e IV.

**Q11**

Com o objetivo de armazenar e recuperar os resultados obtidos pelos alunos de determinado curso de treinamento, foi desenvolvido um sistema em que foram processados os seguinte dados: nome, número de matrícula, nota final e total de abstenções. Nesse aplicativo, a chave primária para a localização dos dados de um aluno consiste em sua matrícula.

A partir dessa situação hipotética, julgue os itens a seguir, relativos à organização de arquivos e aos métodos de acesso a banco de dados.

Caso o arquivo seja ordenado pelo número de matrícula, para a localização da nota de um aluno a partir do nome desse aluno, a pesquisa binária será a mais eficiente.

**Q12**

Considere as listas a seguir, cujos elementos são números inteiros:

- I. 1, 5, 2, 4, 3;
- II. 1, 2, 3, 4, 5;
- III. 5, 4, 3, 2, 1.

Seja  $x=3$  a chave a ser pesquisada. Um algoritmo de pesquisa deverá responder SIM se a chave pertencer à lista e NÃO, caso contrário. O algoritmo de pesquisa conhecido como busca binária:

- A) pode ser aplicado às listas I, II e III;
- B) pode ser aplicado apenas às listas I e II;
- C) pode ser aplicado apenas à lista I;
- D) pode ser aplicado apenas às listas II e III;
- E) não se aplica às listas I, II e III.

**Q1**

Considere que no setor de atendimento da Universidade há uma lista ordenada com o nome de 1000 estudantes. Utilizando o método de pesquisa binária para localizar o nome de um destes estudantes, serão necessárias, no máximo,

- A) 1.000 comparações.
- B) 10 comparações.
- C) 500 comparações.
- D) 200 comparações.
- E) 5 comparações.

**Q15**

Considere o seguinte vetor:

[45, 58, 86, 104, 134, 250, 315, 367, 408, 410, 502, 510, 600, 785, 846, 901]

Utilizando-se uma pesquisa binária, o número máximo de buscas para encontrar a chave 600 será:

- A) 5 acessos.
- B) 10 acessos.
- C) 4 acessos.
- D) 13 acessos.
- E) 3 acessos.

**Q16**

Considere uma lista ordenada com o nome de 1000 funcionários de uma empresa. Utilizando o método de pesquisa binária para localizar o nome de um destes cidadãos, serão necessárias, no máximo,

- A) 1.000 comparações.
- B) 10 comparações.
- C) 500 comparações.
- D) 200 comparações.
- E) 5 comparações.

**Q17**

Considerando que se deseje efetuar uma pesquisa de um valor sobre a chave primária de uma tabela de um banco de dados com uma chave primária com um tipo de campo que receba um valor inteiro e que se possa fazer essa pesquisa utilizando-se a busca sequencial ou a busca binária, assinale a opção correta.

- A) O método de busca binária requer, no máximo,  $\ln(n)$  comparações para determinar o elemento pesquisado, em que  $n$  é o número de registros.
- B) O método de busca binária será sempre mais rápido que o método de busca sequencial, independentemente de a tabela estar ordenada com base no elemento pesquisado.
- C) O método de busca sequencial requererá, no máximo,  $n^2$  comparações para determinar o elemento pesquisado, em que  $n$  será o número de registros.
- D) O método de busca binária sempre efetuará menos comparações que o método de pesquisa sequencial.
- E) O método de busca sequencial efetuará menos comparações para encontrar o elemento pesquisado quando a tabela estiver ordenada em comparação à situação quando a tabela estiver desordenada.

**Q18**

Dispõe-se de uma tabela contendo os dados de 5.000 inscritos em um processo seletivo. A tabela está rigorosamente classificada em ordem alfabética crescente do nome completo do candidato e também já se verificou que não há homônimos inscritos no processo. Deseja-se localizar um candidato na tabela a partir de seu nome completo usando a técnica de Pesquisa Binária (Binary Search). Qual é o número máximo de incursões à tabela para localizar o candidato procurado (ou descobrir que ele não existe)?

- A) 12.
- B) 13.
- C) 500.
- D) 2.500
- E) 5.000

**Q19**

Modifique a pesquisa binária para encontrar a última ocorrência de um elemento em um vetor ordenado que pode conter elementos repetidos.

```
#include <stdio.h>
#define ERRO -1

int ultimaOcorrencia(int *v, int n, int x) {

    int l = 0, r = n - 1;
    int resultado = ERRO;

    while (l <= r) {
        int m = (l + r) / 2;
        if (v[m] == x) {
            resultado = m; // guarda posição atual
            l = m + 1; // continua à direita
        }
        else if (v[m] < x) l = m + 1;
        else r = m - 1;
    }
    return resultado;
}
```

```
// Exemplo de utilização: determinando a (última) posição do elemento 2 no vetor.

int main(void) {

    int v[] = {1, 2, 2, 4, 5, 5, 6};
    int n = sizeof(v)/sizeof(v[0]);
    int x = 2;

    int idx = ultimaOcorrencia(v, n, x);
    if (idx != ERRO) printf("Última ocorrência de %d está no índice %d\n", x, idx);
    else printf("Elemento %d não encontrado.\n", x);

    return 0;
}
```

**Q21**

Dado um vetor ordenado que pode conter elementos repetidos, escreva uma função que use a pesquisa binária para contar quantas vezes um elemento específico aparece no vetor.

```
#include <stdio.h>

#define ERRO -1

int primeiraOcorrencia(int *v, int n, int x);
int ultimaOcorrencia(int *v, int n, int x);
int contarOcorrencias(int *v, int n, int x);

// Exemplo de utilização do algoritmo

int main(void) {

    int v[] = {1, 2, 2, 4, 5, 5, 6};
    int n = sizeof(v) / sizeof(v[0]);

    int qtd = contarOcorrencias(v, n, 2);
    printf("Elemento %d aparece %d vezes\n", 2, qtd);

}
```

```
// Encontrando a primeira ocorrência do elemento x no vetor
```

```
int primeiraOcorrencia(int *v, int n, int x) {  
  
    int l = 0, r = n - 1, resultado = ERRO;  
    while (l <= r) {  
        int m = (l + r) / 2;  
        if (v[m] == x) {  
            resultado = m;  
            r = m - 1; // continua procurando à esquerda  
        } else if (v[m] < x) {  
            l = m + 1;  
        } else {  
            r = m - 1;  
        }  
    }  
    return resultado;  
}
```

```
// Encontrando a última ocorrência de x no vetor

int ultimaOcorrencia(int *v, int n, int x) {

    int l = 0, r = n - 1, resultado = ERRO;
    while (l <= r) {
        int m = (l + r) / 2;
        if (v[m] == x) {
            resultado = m;
            l = m + 1; // continua procurando à direita
        } else if (v[m] < x) {
            l = m + 1;
        } else {
            r = m - 1;
        }
    }
    return resultado;
}
```

```
// Conta quantas vezes x aparece no vetor ordenado

int contarOcorrencias(int *v, int n, int x) {

    int primeira = primeiraOcorrencia(v, n, x);
    if (primeira == ERRO) return 0; // não encontrado
    int ultima = ultimaOcorrencia(v, n, x);
    return (ultima - primeira + 1);

}
```

**Q11**

Dado um vetor ordenado, como **{-5, -2, 0, 3, 10, 15}**, e um número alvo, como **4**, encontre o elemento no vetor que é mais próximo do alvo (que no caso é **3**).

**Outros exemplos:**

- 1)** Para o elemento alvo **4**, os candidatos seriam **3** (distância **1**) e **10** (distância **6**). A resposta seria **3**.
- 2)** Se o alvo fosse **14**, os candidatos seriam **10** (distância **4**) e **15** (distância **1**). A resposta seria **15**.
- 3)** Se o alvo fosse **100** (maior do que todos), a resposta seria **15** (último elemento do conjunto).
- 4)** Se o alvo fosse **-20** (menor do que todos), a resposta seria **-5** (primeiro elemento do conjunto).

```
#include <stdio.h>
#include <stdlib.h> // abs

#define ERRO -1

// Função recursiva para encontrar índice do elemento mais próximo

int PesquisaMaisProximaRec(int *vetor, int Esquerda, int Direita, int Chave) {

    if (Esquerda > Direita) {
        // Caso base: não encontrou, retornar índice mais próximo entre limites
        if (Direita < 0) return Esquerda; // alvo menor que todos
        if (Esquerda >= 0) return Direita + 1;
        return ERRO;
    }

    int Meio = (Esquerda + Direita) / 2;

    if (Chave == vetor[Meio]) return Meio;
    else if (Chave < vetor[Meio]) return PesquisaMaisProximaRec(vetor, Esquerda, Meio - 1, Chave);
    else return PesquisaMaisProximaRec(vetor, Meio + 1, Direita, Chave);
}
```

```
int Meio = (Esquerda + Direita) / 2;

if (Vetor[Meio] == Chave) return Meio; // encontrou exatamente
else if (Vetor[Meio] < Chave) {
    int idx = PesquisaMaisProximaRec(Vetor, Meio + 1, Direita, Chave);

    // Ajusta comparando distâncias

    if (idx == ERRO) return Meio;
    int dist1 = abs(Vetor[idx] - Chave);
    int dist2 = abs(Vetor[Meio] - Chave);
    return (dist2 <= dist1) ? Meio : idx;
}

}
```

```
else {
    int idx = PesquisaMaisProximaRec(Vetor, Esquerda, Meio - 1, Chave);

    // Ajusta comparando distâncias

    if (idx == ERRO) return Meio;
    int dist1 = abs(Vetor[idx] - Chave);
    int dist2 = abs(Vetor[Meio] - Chave);
    return (dist2 <= dist1) ? Meio : idx;
}
```

// Exemplo de utilização do algoritmo

```
int main(void) {  
  
    int v[] = {1, 4, 6, 8, 12, 15};  
    int n = sizeof(v)/sizeof(v[0]);  
    int alvo = 2;  
    int idx = PesquisaMaisProximaRec(v, 0, n - 1, alvo);  
  
    printf("Mais próximo de %d: v[%d] = %d\n", alvo, idx, v[idx]);  
    return 0;  
}
```