

# Algoritmos de Busca

Busca Linear e Busca Binária

(Fonte: Material adaptado dos Slides do prof. Monael.)

# Problema da Busca











- Formalmente:
  - Suponha uma coleção  $\mathbf{V}$  de elementos de tamanho  $\mathbf{n}$ :
$$\mathbf{V} = \{v_0, v_1, v_2, \dots, v_{n-1}\}$$
  - E um elemento  $\mathbf{x}$  qualquer.
  - Averiguar se  $\mathbf{x} = \mathbf{v}_i$ , onde  $0 \leq i < \mathbf{n}$
- Informalmente:
  - Verificar se um elemento  $\mathbf{x}$  está no vetor  $\mathbf{V}$  de tamanho  $\mathbf{n}$ . Se sim, retorne o índice  $\mathbf{i}$ , tal que  $\mathbf{v}_i = \mathbf{x}$ , caso contrário retorne -1.


# Problema da Busca

- Entradas:
  - vetor  $V[0 \dots n-1]$
  - elemento  $x$
- Saída:
  - índice  $i$  tal que  $V[i] == x$
  - $-1$  se  $x$  não estiver em  $V$

# Busca Linear

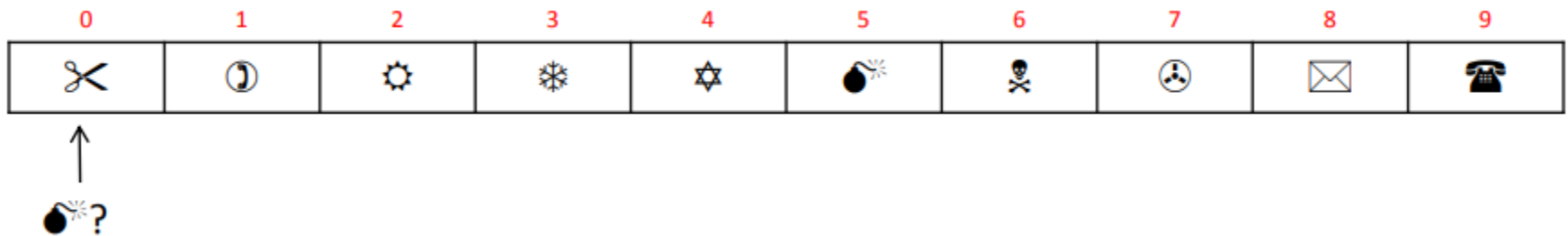
- Exemplo 1

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
|  |  |  |  |  |  |  |  |  |  |

Buscar na lista: 

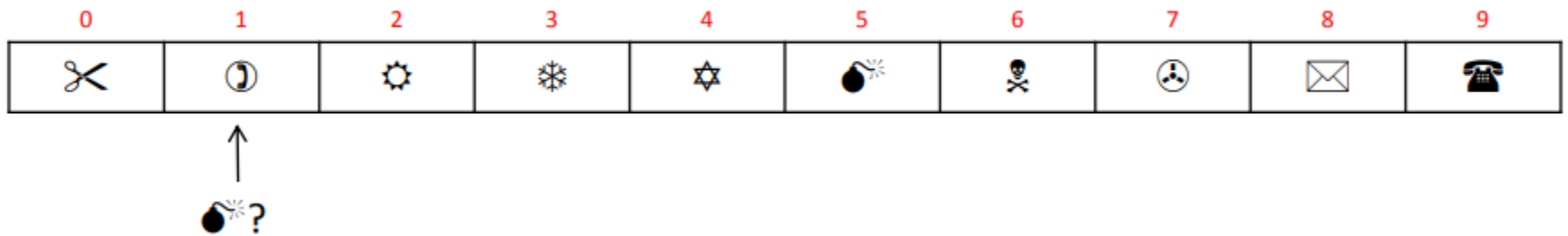
# Busca Linear

- Exemplo 1



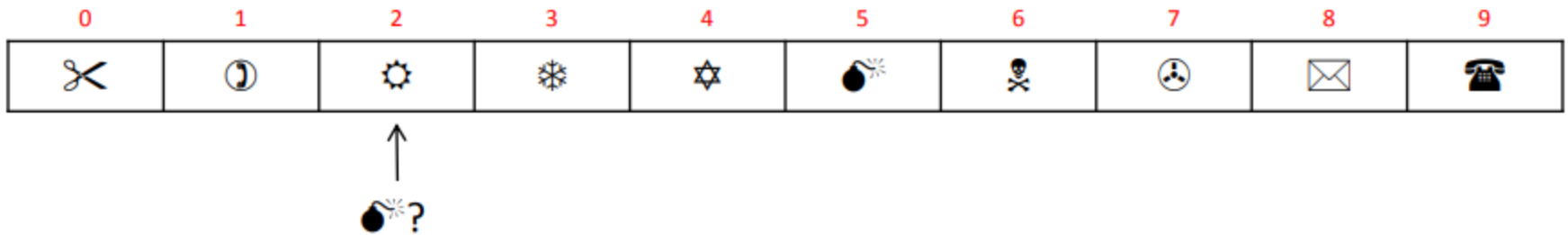
# Busca Linear

- Exemplo 1



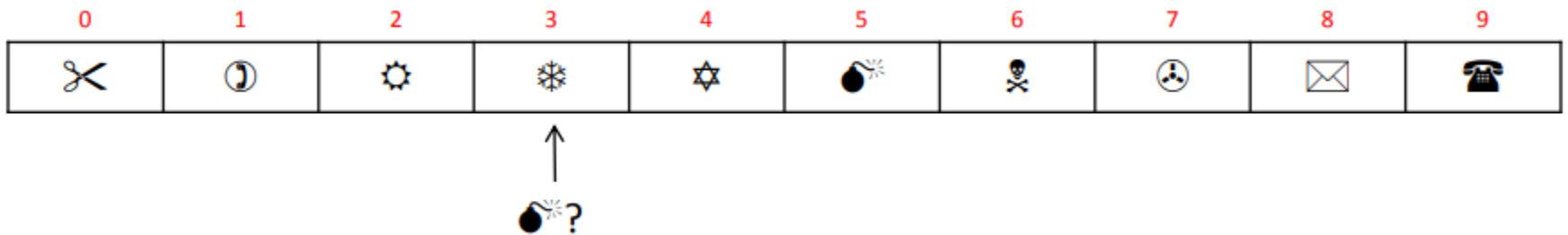
# Busca Linear

- Exemplo 1



# Busca Linear

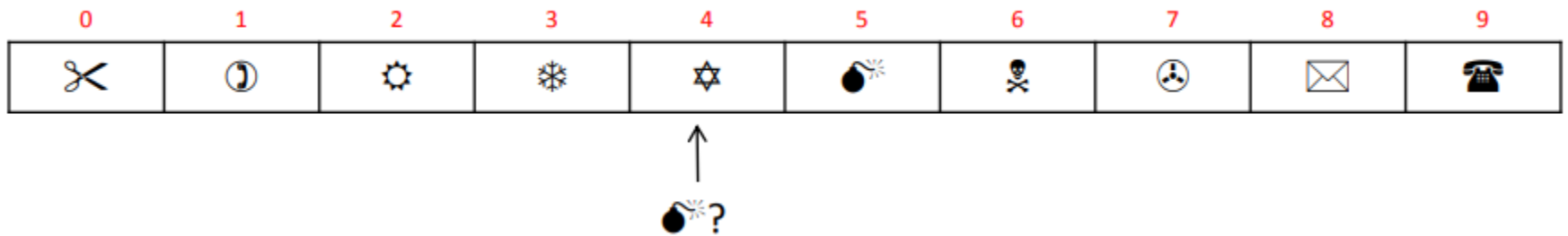
- Exemplo 1





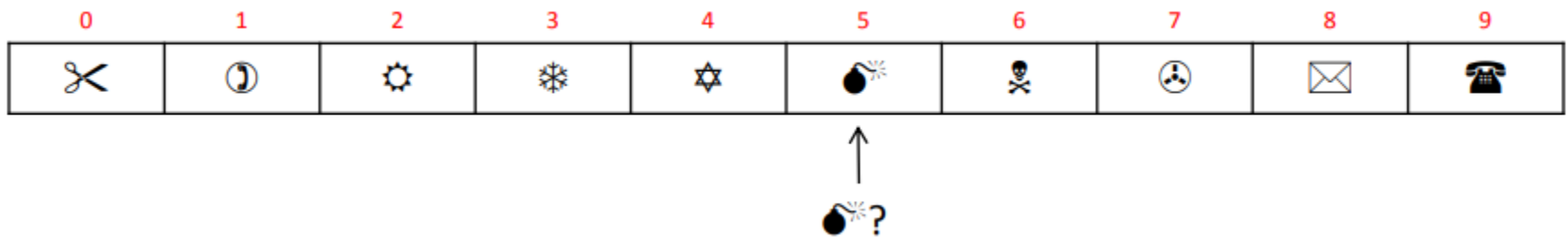
# Busca Linear

- Exemplo 1



# Busca Linear

- Exemplo 1













- Encontramos com **6** comparações!

(**Pergunta:** Quantas comparações no **PIOR** caso?)

# Busca Linear

- Exemplo 2

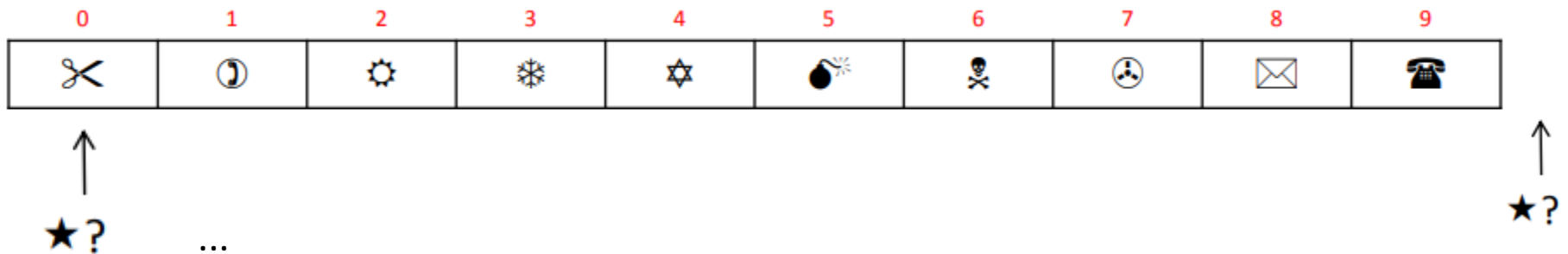
|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
|  |  |  |  |  |  |  |  |  |  |

Buscar na lista: ★

(**Pergunta:** Quantas comparações no **PIOR** caso?)

# Busca Linear

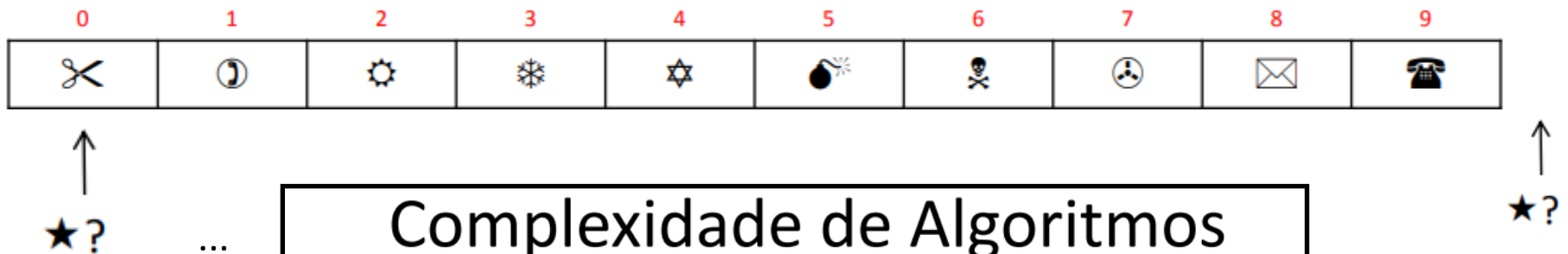
- Exemplo 2



(**Pergunta:** Quantas comparações no **PIOR** caso? Resp: **n = 10.**)

# Busca Linear

- Exemplo 2



## Complexidade de Algoritmos

- Foram necessárias 10 comparações para determinar que ★ não aparece na lista.
  - E se a lista tivesse 100 elementos?
  - E se a lista tivesse  $1 \times 10^6$ ?
  - E se a lista tivesse  $n$  elementos?
- Perceba que a quantidade de comparações aumenta conforme aumenta o tamanho da lista.

# Complexidade de Algoritmos

- Perceba que a quantidade de comparações aumenta conforme aumenta o tamanho da lista.

| Tamanho             | Comparações         |
|---------------------|---------------------|
| 10                  | 10                  |
| 100                 | 100                 |
| $1 \times 10^6$     | $1 \times 10^6$     |
| $1 \times 10^{100}$ | $1 \times 10^{100}$ |

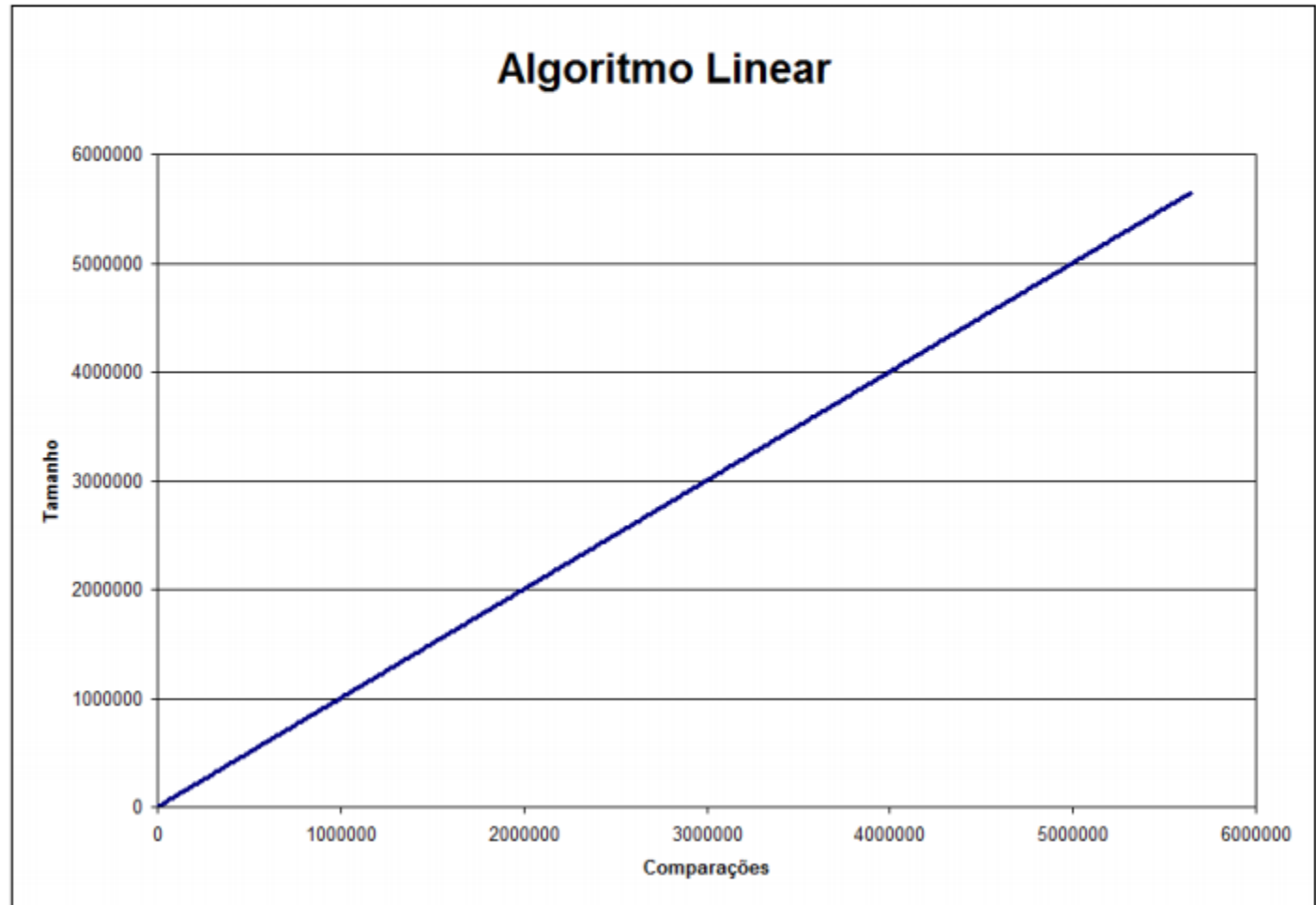
- O esforço do algoritmo (quantidade de comparações) aumenta conforme aumenta-se o tamanho da lista a uma **taxa linear**.

(**Pergunta:** Quantas comparações no **PIOR** caso?)

# Complexidade de Algoritmos

- Linear

$O(n)$



(**Pergunta:** Quantas comparações no **PIOR** caso?)

# Complexidade de Algoritmos

| Consumo de Tempo                   | Notação     |
|------------------------------------|-------------|
| constante                          | $O(1)$      |
| logarítmica<br>(Ex. Busca Binária) | $O(\log n)$ |
| linear<br>(Ex. Busca Linear)       | $O(n)$      |
| quadrática                         | $O(n^2)$    |
| cúbica                             | $O(n^3)$    |
| ...                                |             |



# Busca Linear Iterativa

- Diremos que o algoritmo abaixo consome tempo **linear**.
- Diremos também que o algoritmo é  $O(n)$ .

```
→ 1. int busca (int *v, int n, int x) {  
2.     int i;  
3.     for (i = 0; i < n; i++) {  
4.         if (v[i] == x) {  
5.             return i;  
6.         }  
7.     }  
8.     return -1;  
9. }
```

(**Pergunta:** Quantas comparações no **PIOR** caso?)

# Busca Linear Iterativa

- Diremos que o algoritmo abaixo consome tempo **linear**.
- Diremos também que o algoritmo é  $O(n)$ .

//“Rascunho”:

```
→ 1. int busca (int *v, int n, int x) {  
2.     int i; // 2. O(1)  
3.     for (i = 0; i < n; i++) { // 3. O(n)  
4.         if (v[i] == x) { // 4. O(n) * O(1)  
5.             return i; // 5. O(1)  
6.         }  
7.     }  
8.     return -1; // 8. O(1)  
9. }
```

(**Pergunta:** Quantas comparações no **PIOR** caso?)  
(Quantas vezes cada linha é **executada**?)

// Total:  $O(n) + O(n) + O(3)$

# Busca Linear Iterativa

- **Entradas:** vetor  $V[0..n-1]$ , tamanho de  $V$ , elemento  $x$
- **Saída:** índice  $i$  tal que  $V[i] == x$  ou  $-1$  se  $x$  não estiver em  $V$

//“Rascunho”:

```
→ 1. int busca (int *v, int n, int x) {  
2.     int i; // 2. O(1)  
3.     for (i = 0; i < n; i++) { // 3. O(n)  
4.         if (v[i] == x) { // 4. O(n) * O(1)  
5.             return i; // 5. O(1)  
6.         }  
7.     }  
8.     return -1; // 8. O(1)  
9. }
```

(**Pergunta:** Quantas comparações no **PIOR** caso?)  
(Quantas vezes cada linha é **executada**?)

// Total:  $O(n) + O(n) + O(3)$

# Busca Linear

- Seria possível fazer a busca linear recursiva?
  - Qual o caso base? (**Base da recursão**)
    - Quando se encontra o elemento buscado.
    - Quando não há mais onde buscar.  
(Acaba o espaço de busca)
  - Qual o passo recursivo?
    - Decrementa o espaço de busca.

( “Divisão e Conquista” ou “Dividir para conquistar” )

# Busca Linear Recursiva

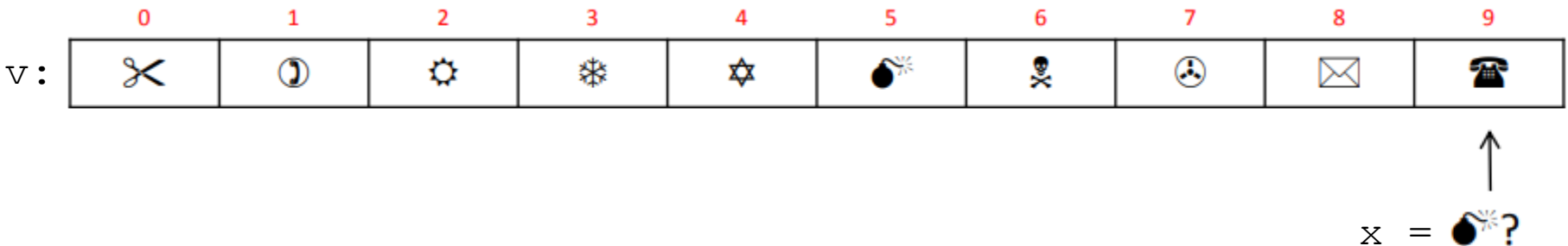
- Qual o caso base?
  - Quando se encontra o elemento buscado.
  - Quando não há mais onde buscar.  
(Acaba o espaço de busca)
- Qual o passo recursivo?
  - Decrementa o espaço de busca.

```
1. int busca (int *v, int n, int x) {  
→ 2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1;  
4.     }  
5.     else {  
→ 6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```

# Busca Linear Recursiva

- Exemplo 1

```
// n = 10  
int i = busca (v, 10, x);
```













```
→ 1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1;  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```

# Busca Linear Recursiva

- Exemplo 1

```
// n = 10  
int i = busca (v, 10-1, x);
```

v:

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
|  |  |  |  |  |  |  |  |  |  |

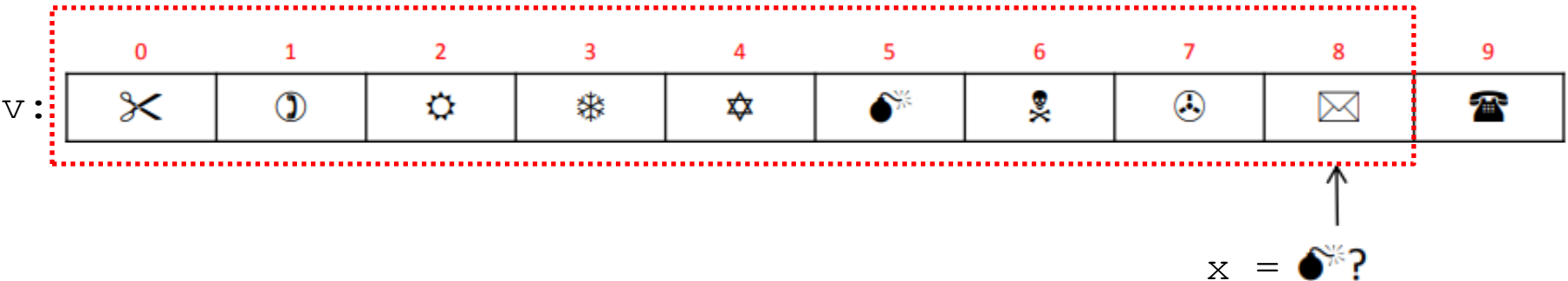


```
1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1;  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```

# Busca Linear Recursiva

- Exemplo 1

```
int i = busca (v, 9, x);
```



```
→ 1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1;  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```













# Busca Linear Recursiva

- Exemplo 1

```
int i = busca (v, 9-1, x);
```

v:

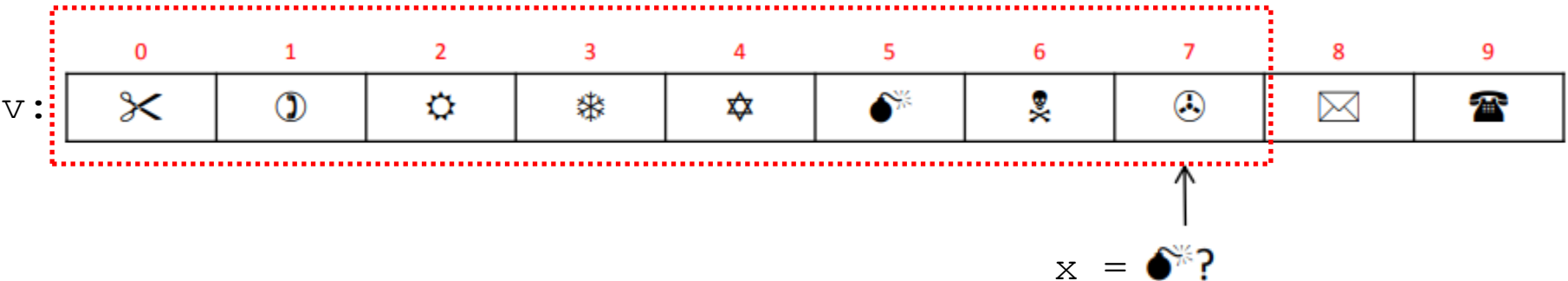
|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
|  |  |  |  |  |  |  |  |  |  |

```
1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1;  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```

# Busca Linear Recursiva

- Exemplo 1

```
int i = busca (v, 8, x);
```













```
→ 1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1;  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```

# Busca Linear Recursiva

- Exemplo 1

```
int i = busca (v, 8-1, x);
```

v:

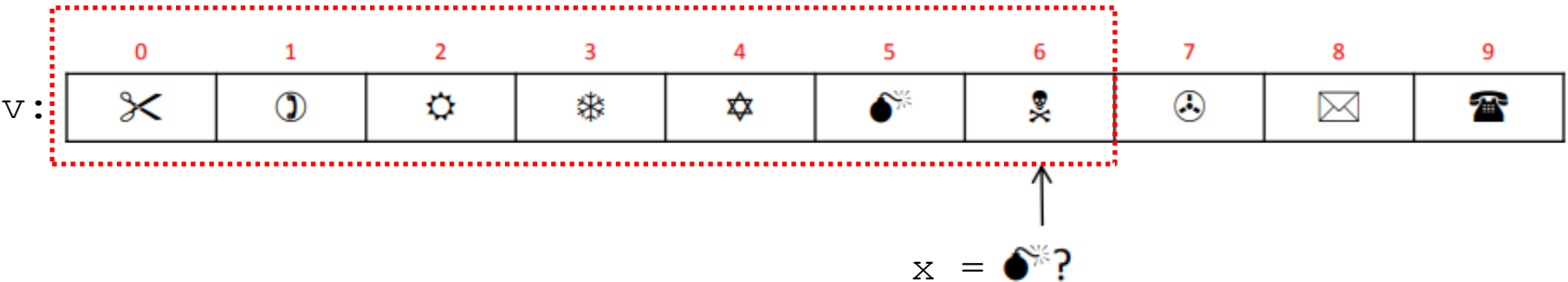
|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
|  |  |  |  |  |  |  |  |  |  |

```
1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1;  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```

# Busca Linear Recursiva

- Exemplo 1

```
int i = busca (v, 7, x);
```













```
→ 1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1;  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```

# Busca Linear Recursiva

- Exemplo 1

```
int i = busca (v, 7-1, x);
```

v:

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
|  |  |  |  |  |  |  |  |  |  |

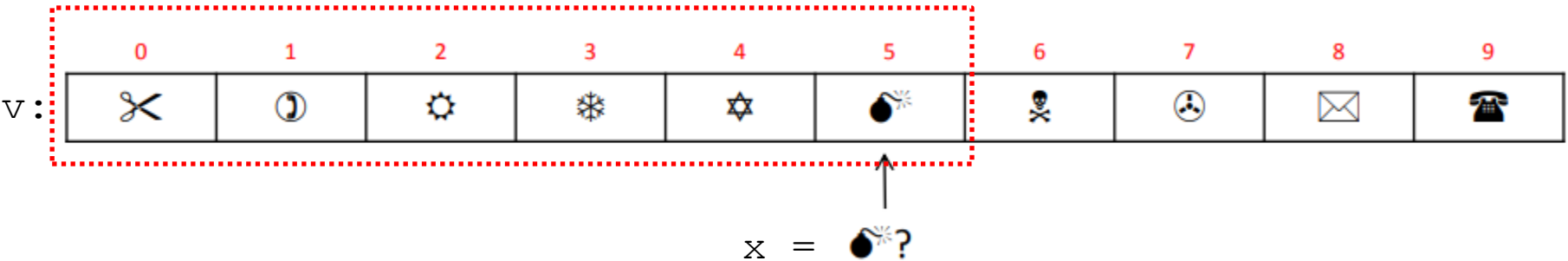


```
1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1;  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```

# Busca Linear Recursiva

- Exemplo 1

```
int i = busca (v, 6, x);
```

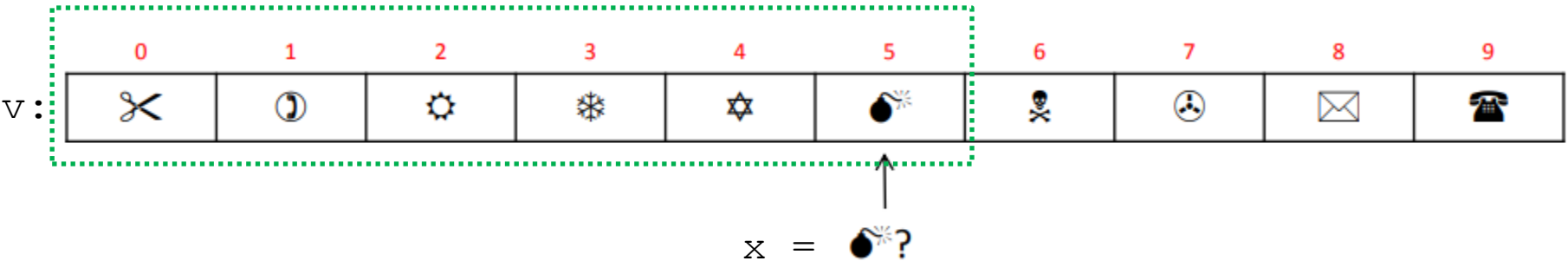


```
1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1;  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```

# Busca Linear Recursiva

- Exemplo 1

```
int i = busca (v, 6, x);
```



```
1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1; // Devolve índice 5  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```











(Pergunta: Quantas comparações?)

# Busca Linear Recursiva

- Exemplo 2

```
// n = 10  
int i = busca (v, 10, x);
```

v

|   |   |   |   |   |  |   |   |   |   |
|---|---|---|---|---|--|---|---|---|---|
| 0   | 1   | 2   | 3   | 4   | 5  | 6   | 7   | 8   | 9   |
|  |  |  |  |  |  |  |  |  |  |

x = ★?

```
1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1;  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```

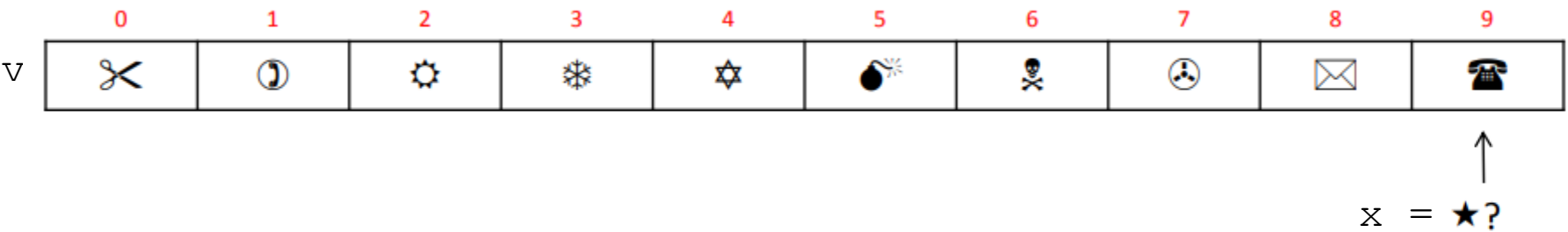
(Pergunta: Quantas comparações no **PIOR** caso?)



# Busca Linear Recursiva

- Exemplo 2

```
// n = 10  
int i = busca (v, 10, x);
```



```
→ 1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1;  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```











(Pergunta: Quantas comparações no **PIOR** caso?)

# Busca Linear Recursiva

- Exemplo 2

```
int i = busca (v, 10-1, x);
```

v

|   |   |   |   |   |  |   |   |   |   |
|---|---|---|---|---|--|---|---|---|---|
| 0   | 1   | 2   | 3   | 4   | 5  | 6   | 7   | 8   | 9   |
|  |  |  |  |  |  |  |  |  |  |

x = ★?

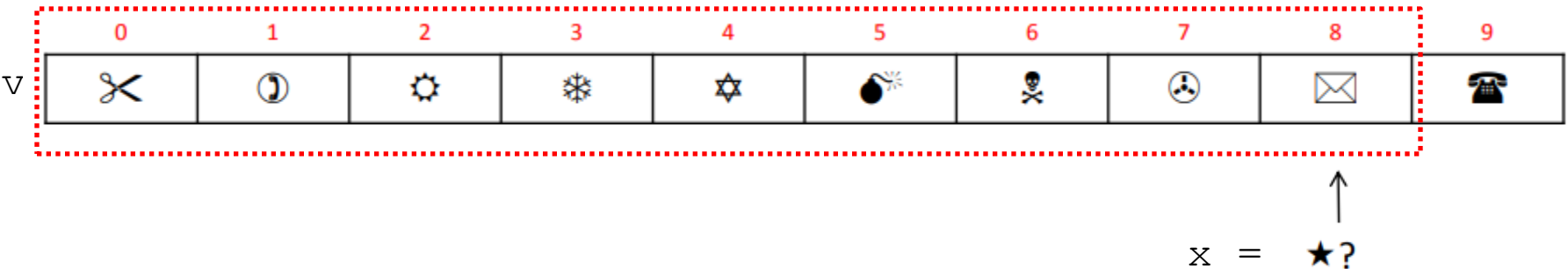
```
1. int busca (int *v, int n, int x) {
2.     if (n == 0 || v[n-1] == x) { // Base da recursão
3.         return n-1;
4.     }
5.     else {
6.         return busca (v, n-1, x); // Passo recursivo
7.     }
8. }
```

(Pergunta: Quantas comparações no **PIOR** caso?)

# Busca Linear Recursiva

- Exemplo 2

```
int i = busca (v, 9, x);
```



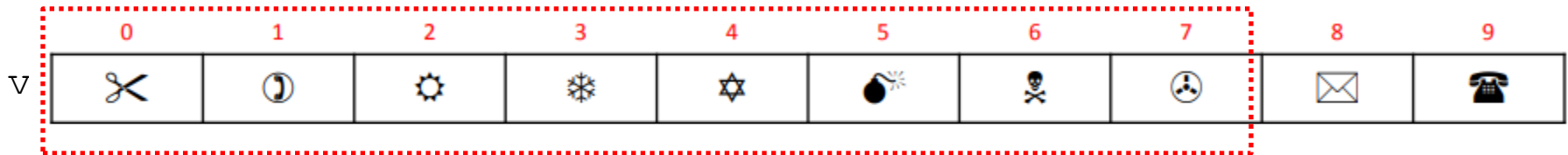
```
→ 1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1;  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```

(Pergunta: Quantas comparações no **PIOR** caso?)

# Busca Linear Recursiva

- Exemplo 2

```
int i = busca (v, 9-1, x);  
//          .... 0
```



x = ★?



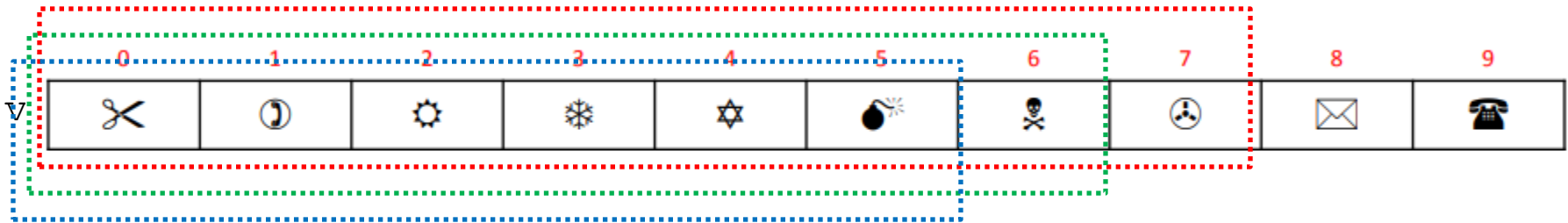
```
1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1;  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```

(Pergunta: Quantas comparações no **PIOR** caso?)

# Busca Linear Recursiva

- Exemplo 2

```
int i = busca (v, 9-1, x);  
//          .... 0
```



`x = ★?`



```
1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1;  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```


(Pergunta: Quantas comparações no **PIOR** caso?)

# Busca Linear Recursiva

- Exemplo 2

```
int i = busca (v, 0, x);
```

v:

|   |   |   |   |   |   |   |   |   |   |
|---|---|---|---|---|---|---|---|---|---|
| 0   | 1   | 2   | 3   | 4   | 5   | 6   | 7   | 8   | 9   |
|  |  |  |  |  |  |  |  |  |  |

x = ★?

```
→ 1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1; // Devolve índice -1  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```

(Pergunta: Quantas comparações no **PIOR** caso?)

# Busca Linear Recursiva

- Diremos que o algoritmo abaixo consome tempo **linear**.
- Diremos também que o algoritmo é  $O(n)$ .

```
1. int busca (int *v, int n, int x) {  
2.     if (n == 0 || v[n-1] == x) { // Base da recursão  
3.         return n-1;  
4.     }  
5.     else {  
6.         return busca (v, n-1, x); // Passo recursivo  
7.     }  
8. }
```

(**Pergunta:** Quantas comparações no **PIOR** caso?)

# Busca Linear Recursiva

- **Entradas:** vetor  $V[0..n-1]$ , tamanho de  $V$ , elemento  $x$
- **Saída:** índice  $i$  tal que  $V[i] == x$  ou  $-1$  se  $x$  não estiver em  $V$

```
1. int busca (int *v, int n, int x) { //“Rascunho”: (Recorrência)
2.     if (n == 0 || v[n-1] == x) { // 2. O(1)
3.         return n-1; // 3. O(1)
4.     } // 5. O(1)
5.     else { // 6. T(n-1)
6.         return busca (v, n-1, x); // Total: T(n) = T(n-1) + O(3)
7.     }
8. }
```

(Pergunta: Quantas comparações no **PIOR** caso?)



# Complexidade de Algoritmos

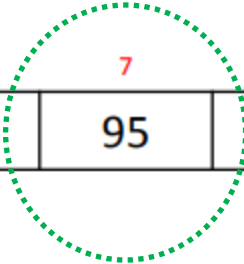
- Existe algum algoritmo mais eficiente para se realizar a busca?
  - Sim. Entretanto, a lista de elementos deve estar ordenada.

(**Pergunta:** Quantas comparações no **PIOR** caso?)

# Busca Binária

- Exemplo 1: buscar  $x = 95$

|    |    |    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|----|----|-----|
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9   |
| 43 | 65 | 69 | 76 | 85 | 89 | 93 | 95 | 99 | 107 |



(Vetor ordenado. "Dividir para conquistar".)

# Busca Binária

- Exemplo 1: buscar  $x = 95$

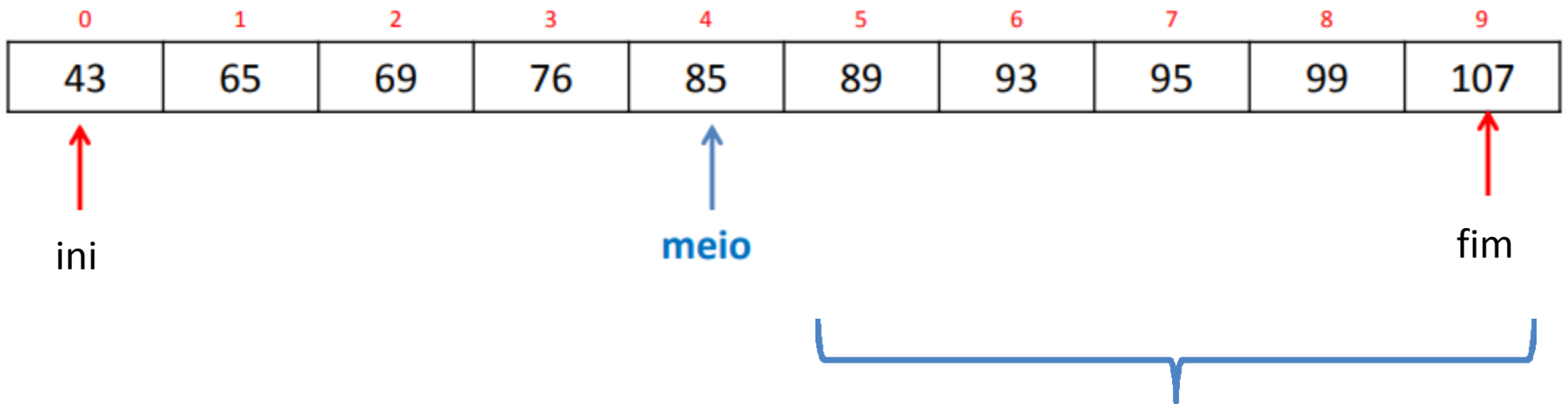
|     |    |    |    |      |    |    |    |    |     |
|-----|----|----|----|------|----|----|----|----|-----|
| 0   | 1  | 2  | 3  | 4    | 5  | 6  | 7  | 8  | 9   |
| 43  | 65 | 69 | 76 | 85   | 89 | 93 | 95 | 99 | 107 |
| ↑   |    |    |    | ↑    |    |    |    |    | ↑   |
| ini |    |    |    | meio |    |    |    |    | fim |

1.  $\text{meio} = \lfloor (\text{ini} + \text{fim}) / 2 \rfloor$
2. É o 95?

(Vetor ordenado. "Dividir para conquistar". Direita ou esquerda?)

# Busca Binária

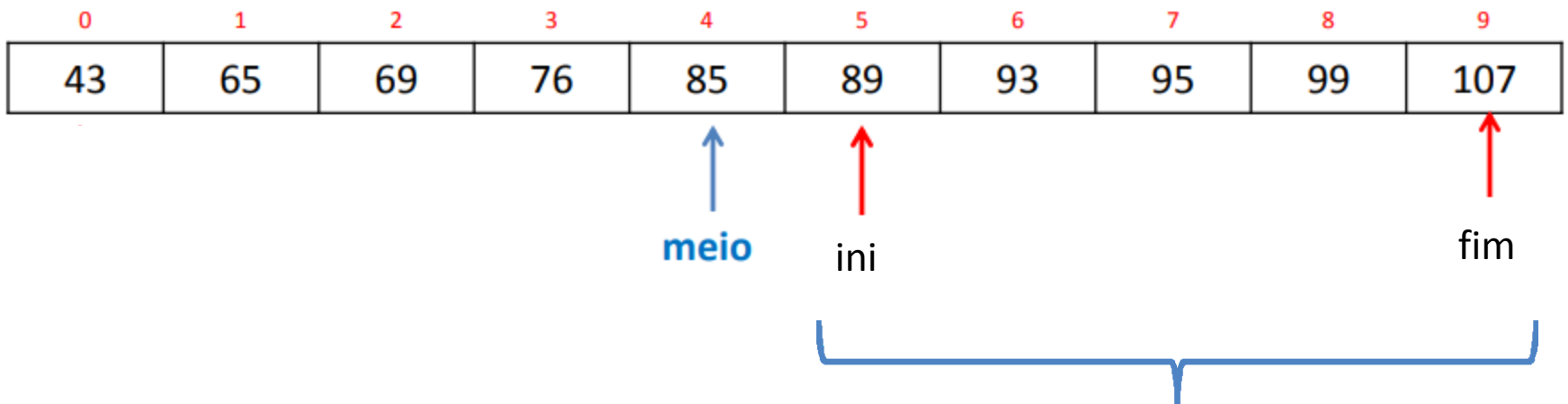
- Exemplo 1: buscar  $x = 95$



1. meio =  $\lfloor (ini+fim) / 2 \rfloor$
2. É o 95?
3. Elemento na posição meio é > 95?
  - 3.1. Não: Então: ini = meio+1

# Busca Binária

- Exemplo 1: buscar  $x = 95$

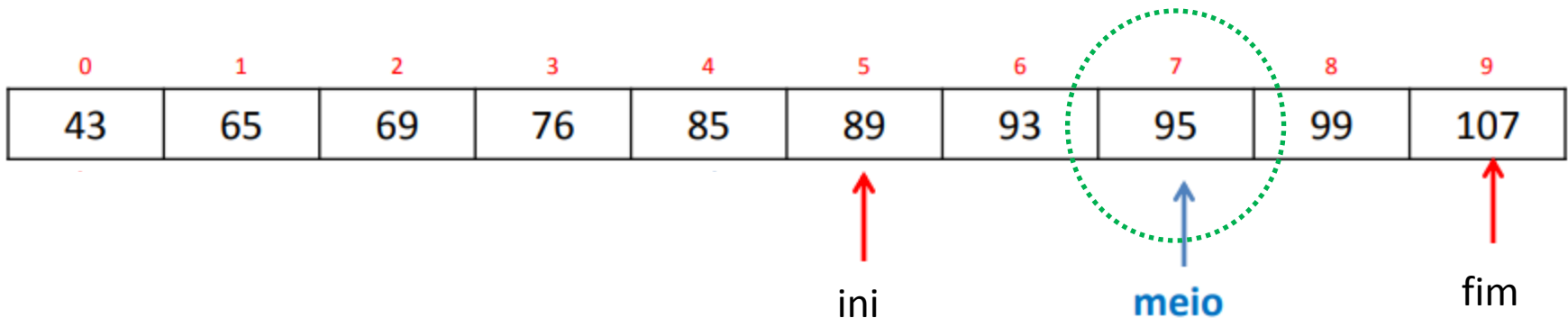


$$1. \text{ meio} = \lfloor (\text{ini} + \text{fim}) / 2 \rfloor$$

(Vetor ordenado. "Dividir para conquistar". Direita ou esquerda? R: **Direita**)

# Busca Binária

- Exemplo 1: buscar  $x = 95$



1.  $\text{meio} = \lfloor (\text{ini} + \text{fim}) / 2 \rfloor$
2. É o 95? Sim!

(**Pergunta:** Quantas comparações para encontrar  $x$ ? **Resp: 2** )

# Busca Binária

- Exemplo 2: buscar  $x = 74$

|    |    |    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|----|----|-----|
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9   |
| 43 | 65 | 69 | 76 | 85 | 89 | 93 | 95 | 99 | 107 |

(**Pergunta:** Quantas comparações no **pior caso**?)

# Busca Binária

- Exemplo 2: buscar  $x = 74$

|     |    |    |    |      |    |    |    |    |     |
|-----|----|----|----|------|----|----|----|----|-----|
| 0   | 1  | 2  | 3  | 4    | 5  | 6  | 7  | 8  | 9   |
| 43  | 65 | 69 | 76 | 85   | 89 | 93 | 95 | 99 | 107 |
| ↑   |    |    |    | ↑    |    |    |    |    | ↑   |
| ini |    |    |    | meio |    |    |    |    | fim |

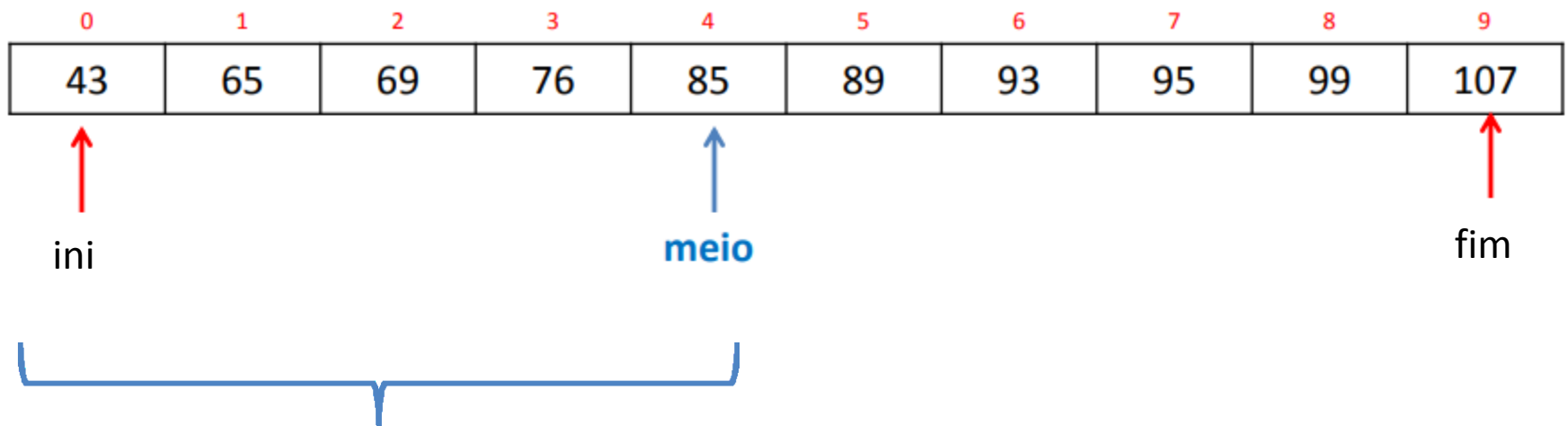
1.  $\text{meio} = \lfloor (\text{ini} + \text{fim}) / 2 \rfloor$
2. É o 74?

(Pergunta: Quantas comparações no **pior caso**?)



# Busca Binária

- Exemplo 2: buscar  $x = 74$



1.  $\text{meio} = \lfloor (\text{ini} + \text{fim}) / 2 \rfloor$

2. É o 74?

3. Elemento na posição meio é  $> 74$ ?

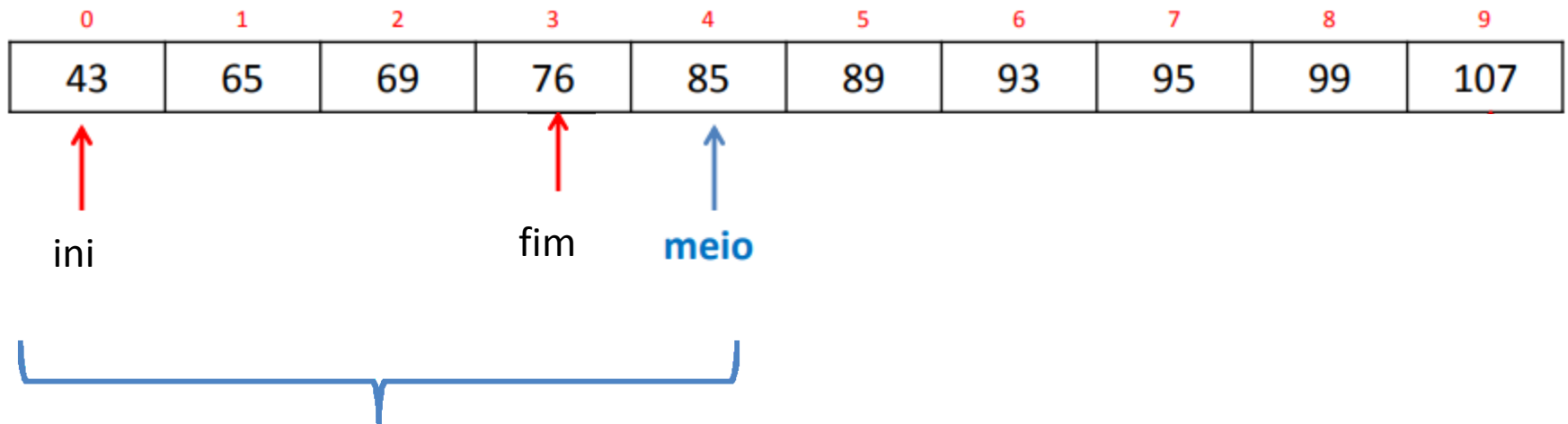
3.1. Não: Então:  $\text{ini} = \text{meio} + 1$

→ 3.2. Sim: Então  $\text{fim} = \text{meio} - 1$

(Pergunta: Quantas comparações no **pior caso**?)

# Busca Binária

- Exemplo 2: buscar  $x = 74$



1.  $meio = \lfloor (ini + fim) / 2 \rfloor$

2. É o 74?

3. Elemento na posição meio é  $> 74$ ?

3.1. Não: Então:  $ini = meio + 1$

→ 3.2. Sim: Então  $fim = meio - 1$

(Pergunta: Quantas comparações no **pior caso**?)

# Busca Binária

- Exemplo 2: buscar  $x = 74$

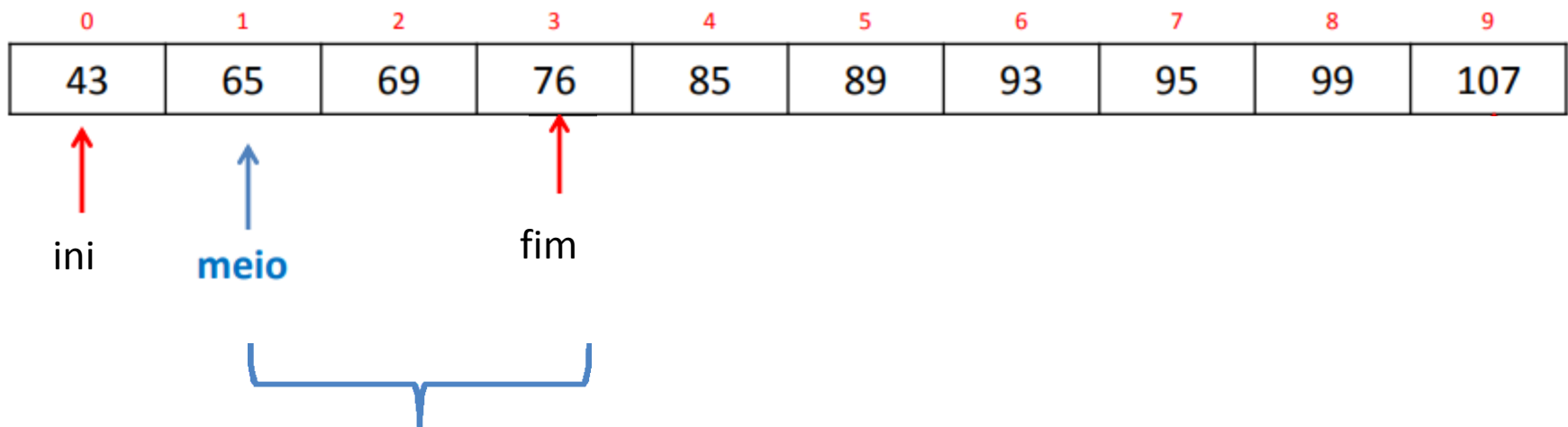
|     |      |    |     |    |    |    |    |    |     |
|-----|------|----|-----|----|----|----|----|----|-----|
| 0   | 1    | 2  | 3   | 4  | 5  | 6  | 7  | 8  | 9   |
| 43  | 65   | 69 | 76  | 85 | 89 | 93 | 95 | 99 | 107 |
| ↑   | ↑    |    | ↑   |    |    |    |    |    |     |
| ini | meio |    | fim |    |    |    |    |    |     |

1.  $\text{meio} = \lfloor (\text{ini} + \text{fim}) / 2 \rfloor$
2. É o 74?

(Pergunta: Quantas comparações no **pior caso**?)

# Busca Binária

- Exemplo 2: buscar  $x = 74$



1.  $\text{meio} = \lfloor (\text{ini} + \text{fim}) / 2 \rfloor$
2. É o 74?
3. Elemento na posição meio é  $> 74$ ?
  - 3.1. Não: Então:  $\text{ini} = \text{meio} + 1$
  - 3.2. Sim: Então  $\text{fim} = \text{meio} - 1$

(Pergunta: Quantas comparações no **pior caso**?)

# Busca Binária

- Exemplo 2: buscar  $x = 74$

|    |    |    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|----|----|-----|
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9   |
| 43 | 65 | 69 | 76 | 85 | 89 | 93 | 95 | 99 | 107 |

Diagram illustrating the binary search process on a sorted array. The array contains the values: 43, 65, 69, 76, 85, 89, 93, 95, 99, 107. The indices are 0 through 9. The current search range is from index 1 (ini) to index 3 (fim). The middle element (meio) is at index 2, which is 69. A bracket indicates the current search range from index 1 to index 3.

1.  $\text{meio} = \lfloor (\text{ini} + \text{fim}) / 2 \rfloor$

2. É o 74?

3. Elemento na posição meio é  $> 74$ ?

→ 3.1. Não: Então:  $\text{ini} = \text{meio} + 1$

3.2. Sim: Então  $\text{fim} = \text{meio} - 1$

(Pergunta: Quantas comparações no **pior caso**?)

# Busca Binária

- Exemplo 2: buscar  $x = 74$

|    |    |    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|----|----|-----|
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9   |
| 43 | 65 | 69 | 76 | 85 | 89 | 93 | 95 | 99 | 107 |

ini   meio   fim

1.  $\text{meio} = \lfloor (\text{ini} + \text{fim}) / 2 \rfloor$
2. É o 74?

(Pergunta: Quantas comparações no **pior caso**?)

# Busca Binária

- Exemplo 2: buscar  $x = 74$

|    |    |    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|----|----|-----|
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9   |
| 43 | 65 | 69 | 76 | 85 | 89 | 93 | 95 | 99 | 107 |

ini   meio   fim

1.  $\text{meio} = \lfloor (\text{ini} + \text{fim}) / 2 \rfloor$

2. É o 74?

3. Elemento na posição meio é  $> 74$ ?

→ 3.1. Não: Então:  $\text{ini} = \text{meio} + 1$

3.2. Sim: Então  $\text{fim} = \text{meio} - 1$

(Pergunta: Quantas comparações no **pior caso**?)

# Busca Binária

- Exemplo 2: buscar  $x = 74$

|    |    |    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|----|----|-----|
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9   |
| 43 | 65 | 69 | 76 | 85 | 89 | 93 | 95 | 99 | 107 |

Diagram illustrating the binary search process on a sorted array. The array contains elements: 43, 65, 69, 76, 85, 89, 93, 95, 99, 107. The search target is 74. The diagram shows the initial state with **meio** (middle) pointing to index 2 (value 69), **ini** (initial) pointing to index 3 (value 76), and **fim** (final) pointing to index 3 (value 76). A bracket indicates the range from index 2 to index 3.

1.  $\text{meio} = \lfloor (\text{ini} + \text{fim}) / 2 \rfloor$

2. É o 74?

3. Elemento na posição meio é  $> 74$ ?

→ 3.1. Não: Então:  $\text{ini} = \text{meio} + 1$

3.2. Sim: Então  $\text{fim} = \text{meio} - 1$

(Pergunta: Quantas comparações no **pior caso**?)



# Busca Binária

- Exemplo 2: buscar  $x = 74$

|    |    |    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|----|----|-----|
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9   |
| 43 | 65 | 69 | 76 | 85 | 89 | 93 | 95 | 99 | 107 |

↑   ↑   ↑  
ini meio fim

1.  $\text{meio} = \lfloor (\text{ini} + \text{fim}) / 2 \rfloor$
2. É o 74?

(Pergunta: Quantas comparações no **pior caso**?)

# Busca Binária

- Exemplo 2: buscar  $x = 74$

|    |    |    |    |    |    |    |    |    |     |
|----|----|----|----|----|----|----|----|----|-----|
| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9   |
| 43 | 65 | 69 | 76 | 85 | 89 | 93 | 95 | 99 | 107 |

ini meio fim

1.  $\text{meio} = \lfloor (\text{ini} + \text{fim}) / 2 \rfloor$

2. É o 74?

3. Elemento na posição meio é  $> 74$ ?

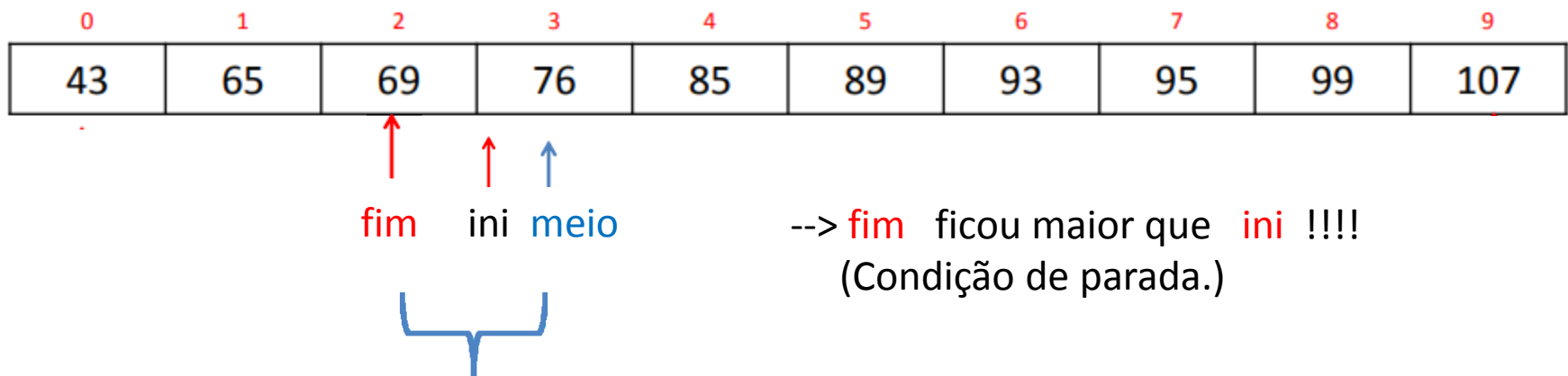
3.1. Não: Então:  $\text{ini} = \text{meio} + 1$

→ 3.2. Sim: Então  $\text{fim} = \text{meio} - 1$

(Pergunta: Quantas comparações no pior caso?)

# Busca Binária

- Exemplo 2: buscar  $x = 74$



1.  $\text{meio} = \lfloor (\text{ini} + \text{fim}) / 2 \rfloor$

2. É o 74?

3. Elemento na posição meio é  $> 74$ ?

3.1. Não: Então:  $\text{ini} = \text{meio} + 1$

→ 3.2. Sim: Então  $\text{fim} = \text{meio} - 1$

(**Pergunta:** Quantas comparações no **pior caso**? **Resp. 4** =  $\lceil \log_2 10 \rceil$  )

# Busca Binária

```
01. int buscaBin (int *v, int n, int x) {
02.     int ini = 0, fim = n-1, meio;
03.     while (ini <= fim) {
04.         meio = (ini + fim) / 2;
05.         if (v[meio] == x) {
06.             return meio;
07.         }
08.         else if (v[meio] > x) {
09.             fim = meio-1;
10.         }
11.         else {
12.             ini = meio + 1;
13.         }
14.     }
15.     return -1;
16. }
```

(**Pergunta:** Quantas comparações no **PIOR** caso? Resp.  $\lceil \log_2 n \rceil$  )

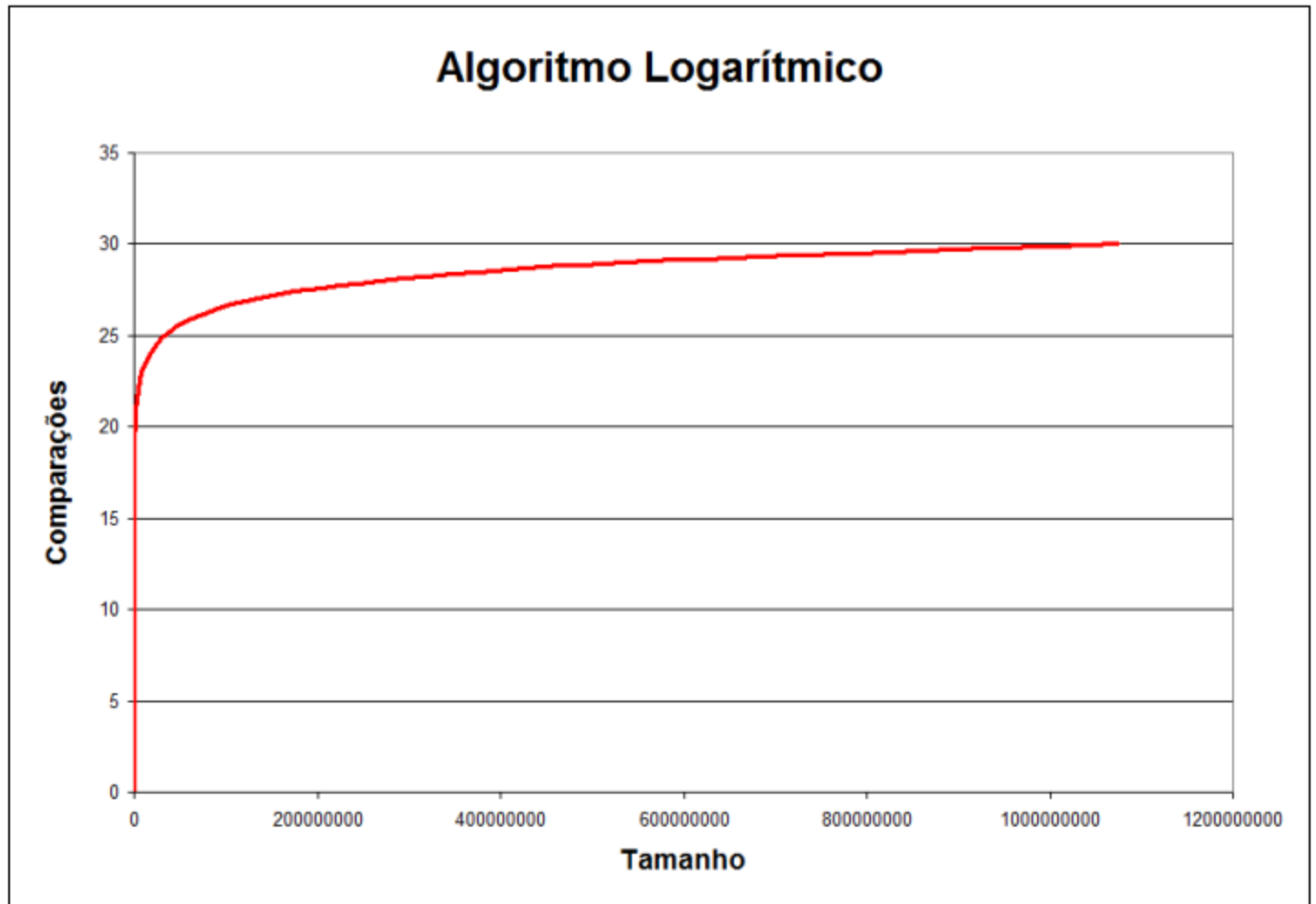
# Análise de Algoritmos

- Pior Caso: Busca Binária
  - No pior caso são necessárias  **$\log_2 n$**  comparações.
  - Portanto a Busca Binária é um algoritmo **logarítmico** ou  **$O(\log_2 n)$** .

# Análise de Algoritmos

- Pior Caso: Busca Binária

$O(\log_2 n)$



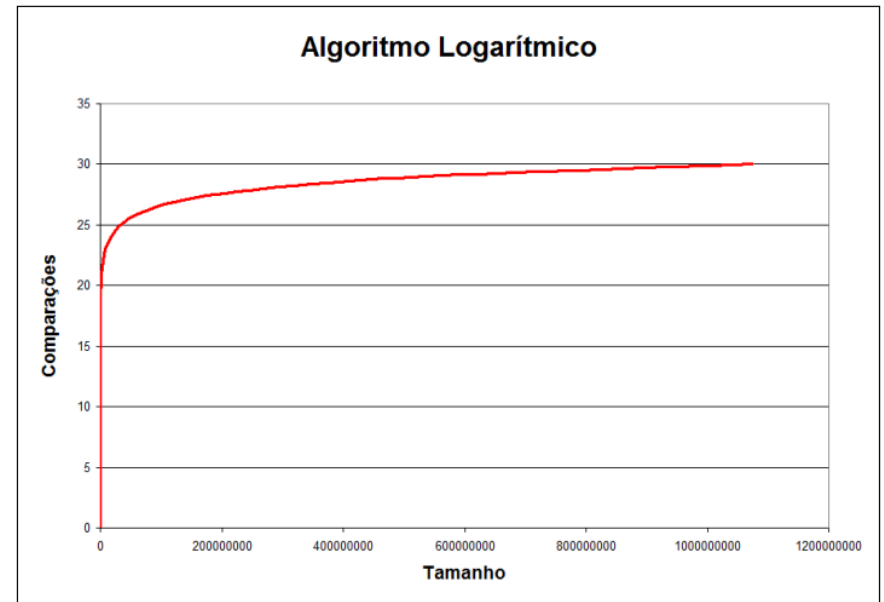
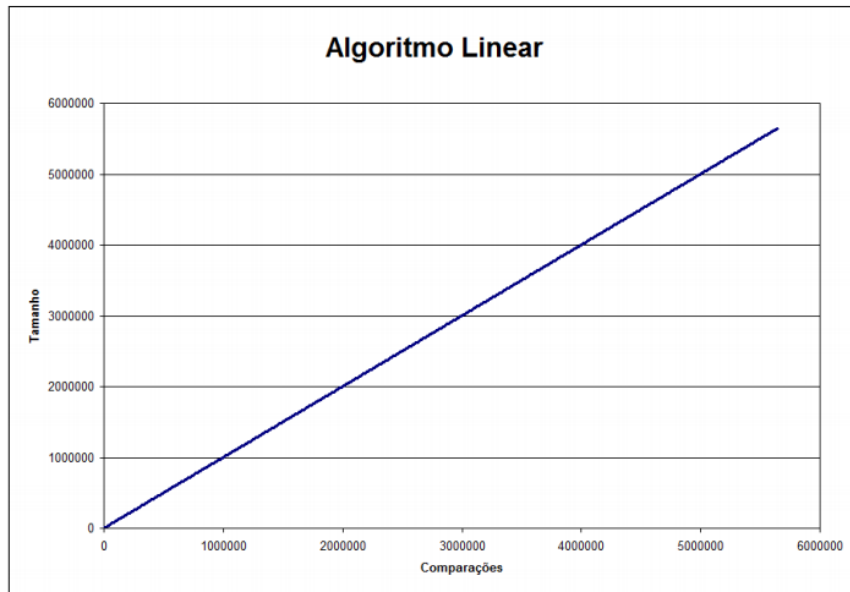
# Linear vs. Logarítmico

- Algoritmos **logarítmicos** são mais eficientes que algoritmos **lineares**. Por quê?

| n   | log(n)             |
|-----|--------------------|
| 2   | 1                  |
| 4   | 2                  |
| 8   | 3                  |
| 10  | 3.3219280948873626 |
| 16  | 4                  |
| 32  | 5                  |
| 64  | 6                  |
| 128 | 7                  |

# Velocidade de crescimento

- Qual curva cresce mais lentamente?





# Complexidade de Algoritmos

| Consumo de Tempo                   | Notação      |
|------------------------------------|--------------|
| constante                          | $O(1)$       |
| logarítmica<br>(Ex. Busca Binária) | $O(\log n)$  |
| linear<br>(Ex. Busca Linear)       | $O(n)$       |
| quadrática                         | $O(n^2)$     |
| cúbica                             | $O(n^3)$     |
| exponencial                        | Ex. $O(2^n)$ |

# Análise de Algoritmos

- Notação  $O(.)$  ("Ó-grande")
  - análise do comportamento assintótico
    - "para  $n$  grande"
  - consumo de tempo
  - consumo de espaço (memória)

# Análise de Algoritmos

- **Pergunta:** Qual seria uma "boa" medida para uma análise de **consumo de tempo** de um algoritmo?
  - Para cada linha do código, podemos somar:
    - 1) tempo de execução (ex. segundos)?
    - 2) quantidade de operações executadas?
    - 3) quantidade de vezes que cada linha é executada?

# Análise de Algoritmos

- **Pergunta:** Qual seria uma "boa" medida para uma análise de **consumo de tempo** de um algoritmo?
  - Para cada linha do código, podemos somar:
    - 1) tempo de execução (ex. segundos)?  
(depende da velocidade da máquina)
    - 2) quantidade de operações executadas?
    - 3) quantidade de vezes que cada linha é executada?  
("Simplificação" = Notação assintótica)

# *Complexidade de Algoritmos*

## Comparação entre Complexidades

---

- ★  $O(1)$  : constante – mais rápido, impossível
- ★  $O(\log \log n)$  : super-rápido
- ★  $O(\log n)$  : logarítmico – muito bom
- ★  $O(n)$  : linear – é o melhor que se pode esperar se algo não pode ser determinado sem examinar toda a entrada
- ★  $O(n \log n)$  : limite de muitos problemas práticos, ex.: ordenar uma coleção de números
- ★  $O(n^2)$  : quadrático
- ★  $O(n^k)$  : polinomial – ok para  $n$  pequeno
- ★  $O(k^n)$ ,  $O(n!)$ ,  $O(n^n)$  : exponencial – evite!