

Complexidade Assintótica



$O(n)$

**Em quanto
tempo um
programa de
vocês roda?**

**Programa: Dado
um array e um
número, diga se o
número esta no
array.**

```
for (int i = 0; i < N; i++) {  
    if (numero == array[i]) {  
        printf("Encontrei\n");  
        break;  
    }  
}
```

```
for (int i = 0; i < N; i++) {  
    if (numero == array[i]) {  
        printf("Encontrei\n");  
        break;  
    }  
}
```

Considerando que cada
loop demora 1s

array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

numero = 1;

numero = 2;

numero = 5;

numero = 10;

```
for (int i = 0; i < N; i++) {  
    if (numero == array[i]) {  
        printf("Encontrei\n");  
        break;  
    }  
}
```

Considerando que cada
loop demora 1s

array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

numero = 1; // 1 segundo

numero = 2;

numero = 5;

numero = 10;

```
for (int i = 0; i < N; i++) {  
    if (numero == array[i]) {  
        printf("Encontrei\n");  
        break;  
    }  
}
```

Considerando que cada
loop demora 1s

array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

numero = 1; // 1 segundo

numero = 2; // 2 segundos

numero = 5;

numero = 10;

```
for (int i = 0; i < N; i++) {  
    if (numero == array[i]) {  
        printf("Encontrei\n");  
        break;  
    }  
}
```

Considerando que cada
loop demora 1s

array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

numero = 1; // 1 segundo

numero = 2; // 2 segundos

numero = 5; // 5 segundos

numero = 10;


```
for (int i = 0; i < N; i++) {  
    if (numero == array[i]) {  
        printf("Encontrei\n");  
        break;  
    }  
}
```

Considerando que cada
loop demora 1s

array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

numero = 1; // 1 segundo

numero = 2; // 2 segundos

numero = 5; // 5 segundos

numero = 10; // 10 segundos

```
for (int i = 0; i < N; i++) {  
    if (numero == array[i]) {  
        printf("Encontrei\n");  
        break;  
    }  
}
```

Considerando que cada
loop demora 1s

array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

numero = 1; // 1 segundo

numero = 2; // 2 segundos

numero = 5; // 5 segundos

numero = 10; // 10 segundos

`array[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10}`

Melhor Caso

1

1 segundo

Caso Médio

5

5 segundos

Pior Caso

10

10 segundos

**Caso Genérico:
Array com N
elementos, e estou
procurando por M**

$\text{array}[N] = \{n_0, n_1, n_2, n_3, n_4, \dots, n_{N-1}\}$

Melhor Caso

Caso Médio

Pior Caso

$\text{array}[N] = \{n_0, n_1, n_2, n_3, n_4, \dots, n_{N-1}\}$

Melhor Caso

n_0

Caso Médio

Pior Caso

$\text{array}[N] = \{n_0, n_1, n_2, n_3, n_4, \dots, n_{N-1}\}$

Melhor Caso

n_0

Caso Médio

$n_{N/2}$

Pior Caso

$\text{array}[N] = \{n_0, n_1, n_2, n_3, n_4, \dots, n_{N-1}\}$

Melhor Caso

n_0

Caso Médio

$n_{N/2}$

Pior Caso

n_{N-1}

$\text{array}[N] = \{n_0, n_1, n_2, n_3, n_4, \dots, n_{N-1}\}$

Melhor Caso

n_0

Caso Médio

$n_{N/2}$

Pior Caso

n_{N-1}

Qual desses vai ser o caso do M?

$\text{array}[N] = \{n_0, n_1, n_2, n_3, n_4, \dots, n_{N-1}\}$

Melhor Caso

n_0

Caso Médio

$n_{N/2}$

Pior Caso

n_{N-1}

Qual desses vai
ser o caso do M?

Depende
¯_(ツ)_/¯

$\text{array}[N] = \{n_0, n_1, n_2, n_3, n_4, \dots, n_{N-1}\}$

Melhor Caso

n_0

Caso Médio

$n_{N/2}$

Pior Caso

n_{N-1}

Qual desses vai ser o caso do M?

Depende
¯_(ツ)_/¯

Sempre considere sendo o pior!

$\text{array}[N] = \{n_0, n_1, n_2, n_3, n_4, \dots, n_{N-1}\}$

Melhor Caso

n_0

Caso Médio

$n_{N/2}$

Pior Caso

n_{N-1}

Qual desses vai ser o caso do M?

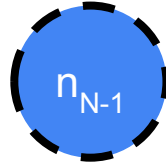
Depende
¯_(ツ)_/¯

Sempre considere sendo o pior!

N segundos

$\text{array}[N] = \{n_0, n_1, n_2, n_3, n_4, \dots, n_{N-1}\}$

Pior Caso

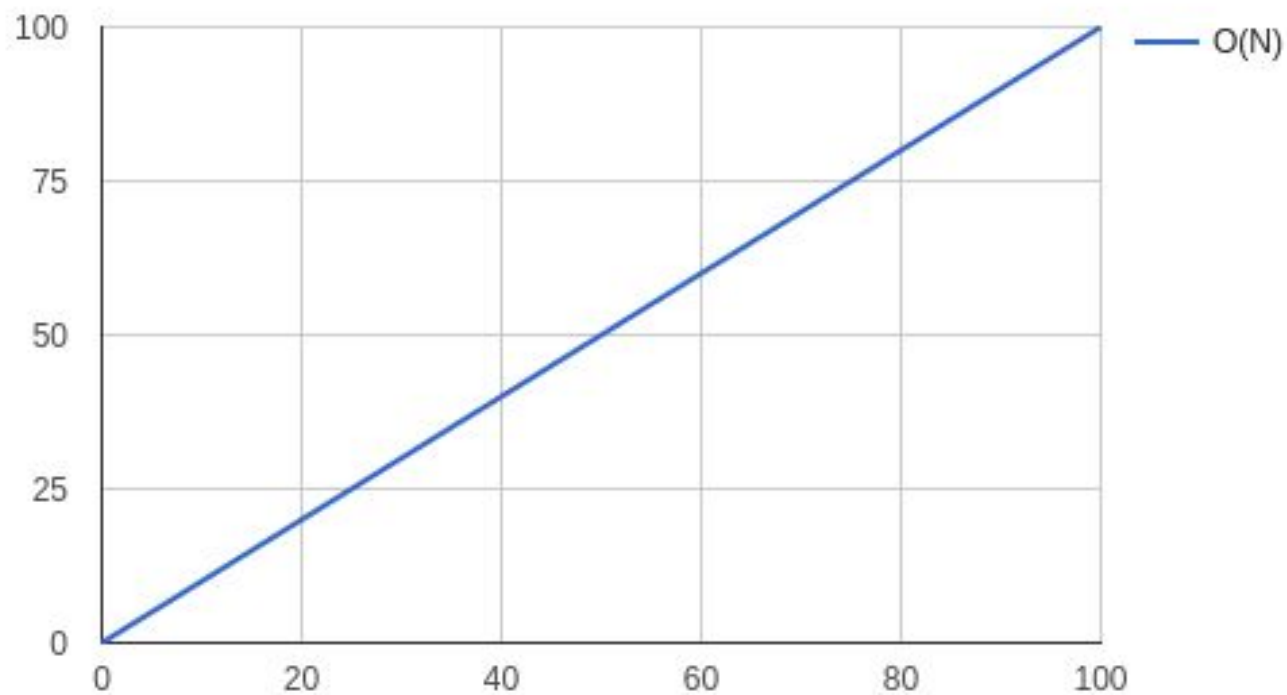


N segundos

Complexidade
Assintótica

$O(N)$

Complexidade Linear



$O(N)$

Complexidade Linear

Se meu algoritmo realizar operações com N elementos, ele tem complexidade no mínimo linear.

**Programa: Dado
duas matrizes, faça
a soma de ambas**


```
for (int i = 0; i < N; i++) {  
    for (int j = 0; j < M; j++) {  
        C[i][j] = A[i][j] + B[i][j];  
    }  
}
```

```
for (int i = 0; i < N; i++) {  
    for (int j = 0; j < M; j++) {  
        C[i][j] = A[i][j] + B[i][j];  
    }  
}
```

Considerando que cada
loop demora 1s

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 6 \\ 7 & 9 & 4 \\ 5 & 1 & 6 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 9 \\ 11 & 14 & 10 \\ 12 & 9 & 15 \end{bmatrix}$$

3 linhas

3 colunas

```
for (int i = 0; i < N; i++) {  
    for (int j = 0; j < M; j++) {  
        C[i][j] = A[i][j] + B[i][j];  
    }  
}
```

Considerando que cada
loop demora 1s

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 6 \\ 7 & 9 & 4 \\ 5 & 1 & 6 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 9 \\ 11 & 14 & 10 \\ 12 & 9 & 15 \end{bmatrix}$$

3 linhas

3 colunas

9 segundos

```

for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        C[i][j] = A[i][j] + B[i][j];
    }
}

```

Considerando que cada
loop demora 1s

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 6 \\ 7 & 9 & 4 \\ 5 & 1 & 6 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 9 \\ 11 & 14 & 10 \\ 12 & 9 & 15 \end{bmatrix}$$

3 linhas

3 colunas

9 segundos

2 linhas

4 colunas

3 linhas

7 colunas

```
for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        C[i][j] = A[i][j] + B[i][j];
    }
}
```

Considerando que cada
loop demora 1s

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 6 \\ 7 & 9 & 4 \\ 5 & 1 & 6 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 9 \\ 11 & 14 & 10 \\ 12 & 9 & 15 \end{bmatrix}$$

3 linhas

3 colunas

9 segundos

2 linhas

4 colunas

8 segundos

3 linhas

7 colunas

```

for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        C[i][j] = A[i][j] + B[i][j];
    }
}

```

Considerando que cada
loop demora 1s

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} + \begin{bmatrix} 1 & 2 & 6 \\ 7 & 9 & 4 \\ 5 & 1 & 6 \end{bmatrix} = \begin{bmatrix} 2 & 4 & 9 \\ 11 & 14 & 10 \\ 12 & 9 & 15 \end{bmatrix}$$

3 linhas

3 colunas

9 segundos

2 linhas

4 colunas

8 segundos

3 linhas

7 colunas

21 segundos

Caso Genérico:
Uma matriz
 $N \times M$

```

for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        C[i][j] = A[i][j] + B[i][j];
    }
}

```

Considerando que cada
loop demora 1s

$$\begin{bmatrix} a_{0,0} & \dots & a_{0,M-1} \\ \vdots & \ddots & \vdots \\ a_{N-1,0} & \dots & a_{N-1,M-1} \end{bmatrix} + \begin{bmatrix} b_{0,0} & \dots & b_{0,M-1} \\ \vdots & \ddots & \vdots \\ b_{N-1,0} & \dots & b_{N-1,M-1} \end{bmatrix} = \begin{bmatrix} c_{0,0} & \dots & c_{0,M-1} \\ \vdots & \ddots & \vdots \\ c_{N-1,0} & \dots & c_{N-1,M-1} \end{bmatrix}$$

N*M segundos


```

for (int i = 0; i < N; i++) {
    for (int j = 0; j < M; j++) {
        C[i][j] = A[i][j] + B[i][j];
    }
}

```

Considerando que cada
loop demora 1s

$$\begin{bmatrix} a_{0,0} & \dots & a_{0,M-1} \\ \vdots & \ddots & \vdots \\ a_{N-1,0} & \dots & a_{N-1,M-1} \end{bmatrix} + \begin{bmatrix} b_{0,0} & \dots & b_{0,M-1} \\ \vdots & \ddots & \vdots \\ b_{N-1,0} & \dots & b_{N-1,M-1} \end{bmatrix} = \begin{bmatrix} c_{0,0} & \dots & c_{0,M-1} \\ \vdots & \ddots & \vdots \\ c_{N-1,0} & \dots & c_{N-1,M-1} \end{bmatrix}$$

Pior Caso

$N \times N$ segundos

$>$

$N \times M$ segundos

$$\begin{bmatrix} c_{0,0} & \dots & c_{0,N-1} \\ \vdots & \ddots & \vdots \\ c_{N-1,0} & \dots & c_{N-1,N-1} \end{bmatrix} + \begin{bmatrix} c_{0,0} & \dots & c_{0,N-1} \\ \vdots & \ddots & \vdots \\ c_{N-1,0} & \dots & c_{N-1,N-1} \end{bmatrix} = \begin{bmatrix} c_{0,0} & \dots & c_{0,N-1} \\ \vdots & \ddots & \vdots \\ c_{N-1,0} & \dots & c_{N-1,N-1} \end{bmatrix}$$

Pior Caso

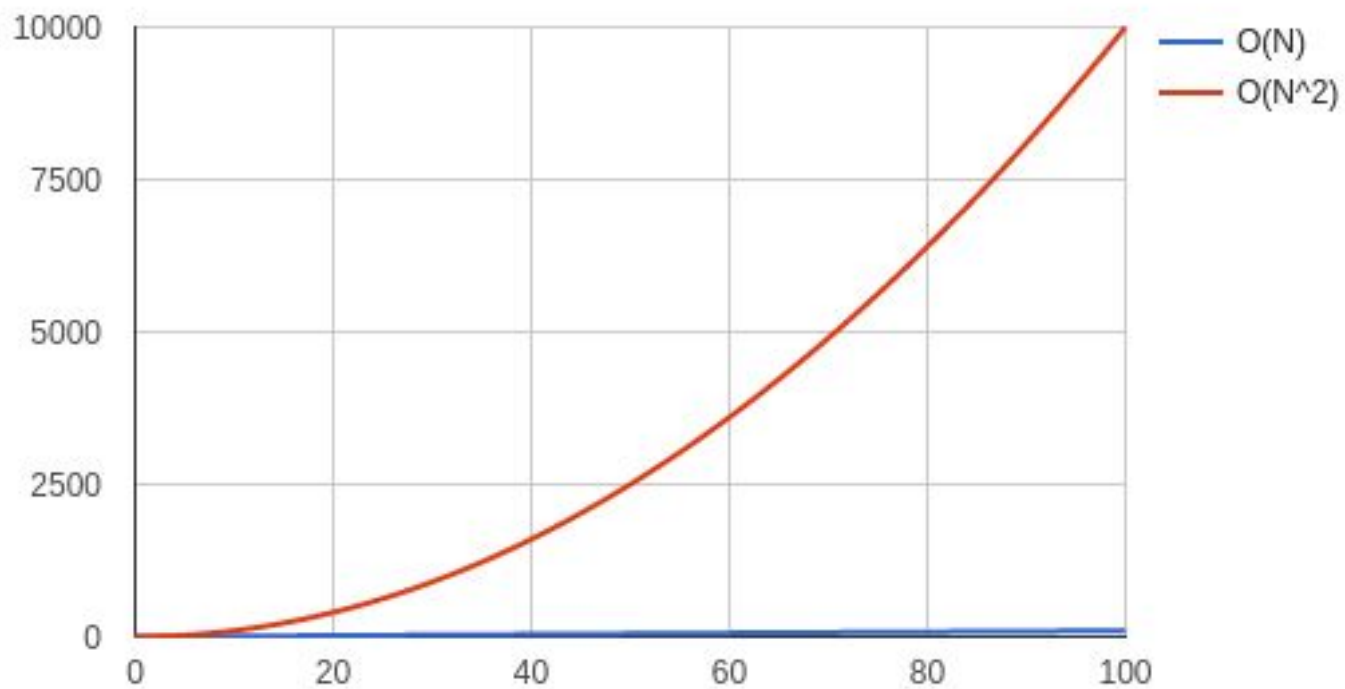
N*N

N*N segundos

Complexidade
Assintótica

$O(N^2)$

Complexidade Quadrática



$O(N^2)$

Complexidade Quadrática

Se meu algoritmo realizar operações com $N \times N$ elementos, ele tem complexidade no mínimo quadrática.

Para um código, qual a melhor complexidade?

A - $O(n)$

B - $O(n^2)$

**Para um código, qual a
melhor complexidade?**

A - $O(n)$



B - $O(n^2)$

**Programa: Some
dois números**

$$c = a + b;$$


```
c = a + b;
```

`c = a + b;`

1 operação

`c = a + b;`

1 operação

`a = 10, b = 20;`

`a = 2, b = 3;`

`a = 10000, b = 500;`

`c = a + b;`

1 operação

`a = 10, b = 20;`

1 operação

`a = 2, b = 3;`

`a = 10000, b = 500;`

`c = a + b;`

1 operação

`a = 10, b = 20;`

1 operação

`a = 2, b = 3;`

1 operação

`a = 10000, b = 500;`

`c = a + b;`

1 operação

`a = 10, b = 20;`

1 operação

`a = 2, b = 3;`

1 operação

`a = 10000, b = 500;`

1 operação

`c = a + b;`

1 operação

`a = 10, b = 20;`

1 operação

`a = 2, b = 3;`

1 operação

`a = 10000, b = 500;`

1 operação

Número de
operações
independe
dos valores
de entrada

c = a + b;

1 operação

```
for (int i = 0; i < N; i++) {  
    if (numero == array[i]) {  
        printf("Encontrei\n");  
        break;  
    }  
}
```


c = a + b;

```
for (int i = 0; i < N; i++) {  
    if (numero == array[i]) {  
        printf("Encontrei\n");  
        break;  
    }  
}
```

1 operação

N operações

$c = a + b;$

1 operação

```
for (int i = 0; i < N; i++) {  
    if (numero == array[i]) {  
        printf("Encontrei\n");  
        break;  
    }  
}
```

N operações

```
for (int i = 0; i < N; i++) {  
    for (int j = 0; j < N; j++) {  
        C[i][j] = A[i][j] + B[i][j];  
    }  
}
```

$c = a + b;$

1 operação

```
for (int i = 0; i < N; i++) {  
    if (numero == array[i]) {  
        printf("Encontrei\n");  
        break;  
    }  
}
```

N operações

```
for (int i = 0; i < N; i++) {  
    for (int j = 0; j < N; j++) {  
        C[i][j] = A[i][j] + B[i][j];  
    }  
}
```

N^2 operações

c = a + b;

```
for (int i = 0; i < N; i++) {  
    if (numero == array[i]) {  
        printf("Encontrei\n");  
        break;  
    }  
}
```

```
for (int i = 0; i < N; i++) {  
    for (int j = 0; j < N; j++) {  
        C[i][j] = A[i][j] + B[i][j];  
    }  
}
```

1 operação

N operações

$O(N)$

N^2 operações

$O(N^2)$

$c = a + b;$

```
for (int i = 0; i < N; i++) {  
    if (numero == array[i]) {  
        printf("Encontrei\n");  
        break;  
    }  
}
```

```
for (int i = 0; i < N; i++) {  
    for (int j = 0; j < N; j++) {  
        C[i][j] = A[i][j] + B[i][j];  
    }  
}
```

1 operação

$O(1)$

N operações

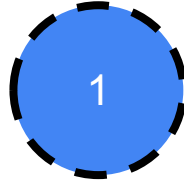
$O(N)$

N^2 operações

$O(N^2)$

$c = a + b;$

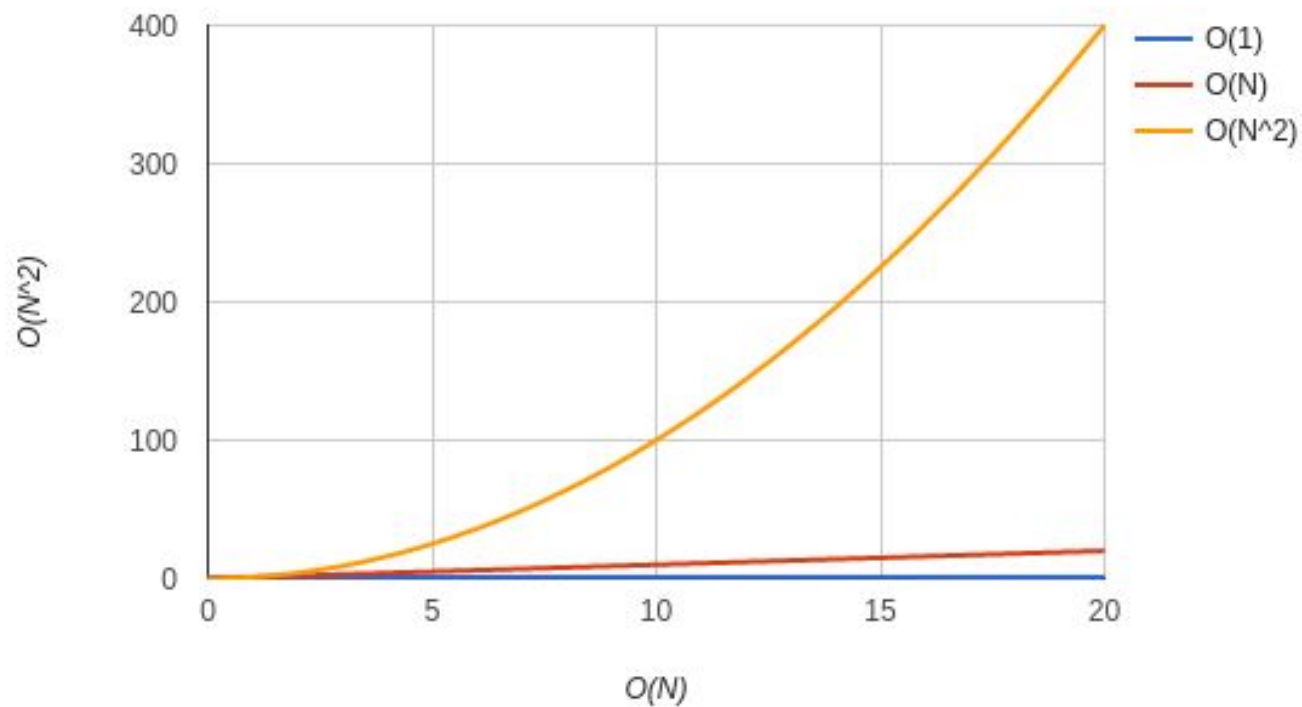
Pior Caso



Complexidade
Assintótica

$O(1)$

Complexidade Constante



$O(1)$

Complexidade Constante

Se meu algoritmo realizar o mesmo número de operações fixas sempre, ele tem complexidade no mínimo constante.

Qual a complexidade?

```
int funcao(int a, int b) {  
    int c = a + b;  
  
    c *= a * b;  
  
    c /= a % b;  
  
    return c;  
}
```

Qual a complexidade?

```
int funcao(int a, int b) {  
    int c = a + b;  
    c *= a * b;  
    c /= a % b;  
    return c;  
}
```

$O(1)$

Qual a complexidade?

```
int funcao(int a, int b) {  
    int c = a + b;  
    c *= a * b;  
    c /= a % b;  
    return c;  
}
```

$O(1)$

Os valores de entrada **NÃO**
afetam o número de
operações realizadas

Qual a complexidade?

```
int funcao(int a, int b, int array[]) {  
    for (int i = 0; i < b; i++) {  
        if (a == array[i]) {  
            return 1;  
        }  
    }  
  
    return 0;  
}
```

Qual a complexidade?

```
int funcao(int a, int b, int array[]) {  
    for (int i = 0; i < b; i++) {  
        if (a == array[i]) {  
            return 1;  
        }  
    }  
  
    return 0;  
}
```

$O(n)$

Qual a complexidade?

```
int funcao(int a, int b, int array[]) {  
    for (int i = 0; i < b; i++) {  
        if (a == array[i]) {  
            return 1;  
        }  
    }  
  
    return 0;  
}
```

$O(n)$

O tamanho b do array, influencia no número de operações executadas linearmente

Qual a complexidade?

```
int funcao(int b, int M[][10]) {  
    soma = 0;  
  
    for (int i = 0; i < b; i++) {  
        for (int j = 0; j < b; j++) {  
            soma += M[i][j];  
        }  
    }  
  
    return soma;  
}
```

Qual a complexidade?

```
int funcao(int b, int M[][10]) {  
    soma = 0;  
  
    for (int i = 0; i < b; i++) {  
        for (int j = 0; j < b; j++) {  
            soma += M[i][j];  
        }  
    }  
  
    return soma;  
}
```

$O(n^2)$

Qual a complexidade?

```
int funcao(int b, int M[][10]) {  
    soma = 0;  
  
    for (int i = 0; i < b; i++) {  
        for (int j = 0; j < b; j++) {  
            soma += M[i][j];  
        }  
    }  
  
    return soma;  
}
```

$O(n^2)$

O tamanho b da matriz, influencia no número de operações executadas quadraticamente

Qual a complexidade?

```
int funcao(int b, int M[][10]) {  
    soma = 0;  
  
    for (int i = 0; i < b; i++) {  
        soma += M[i][0];  
    }  
  
    return soma;  
}
```

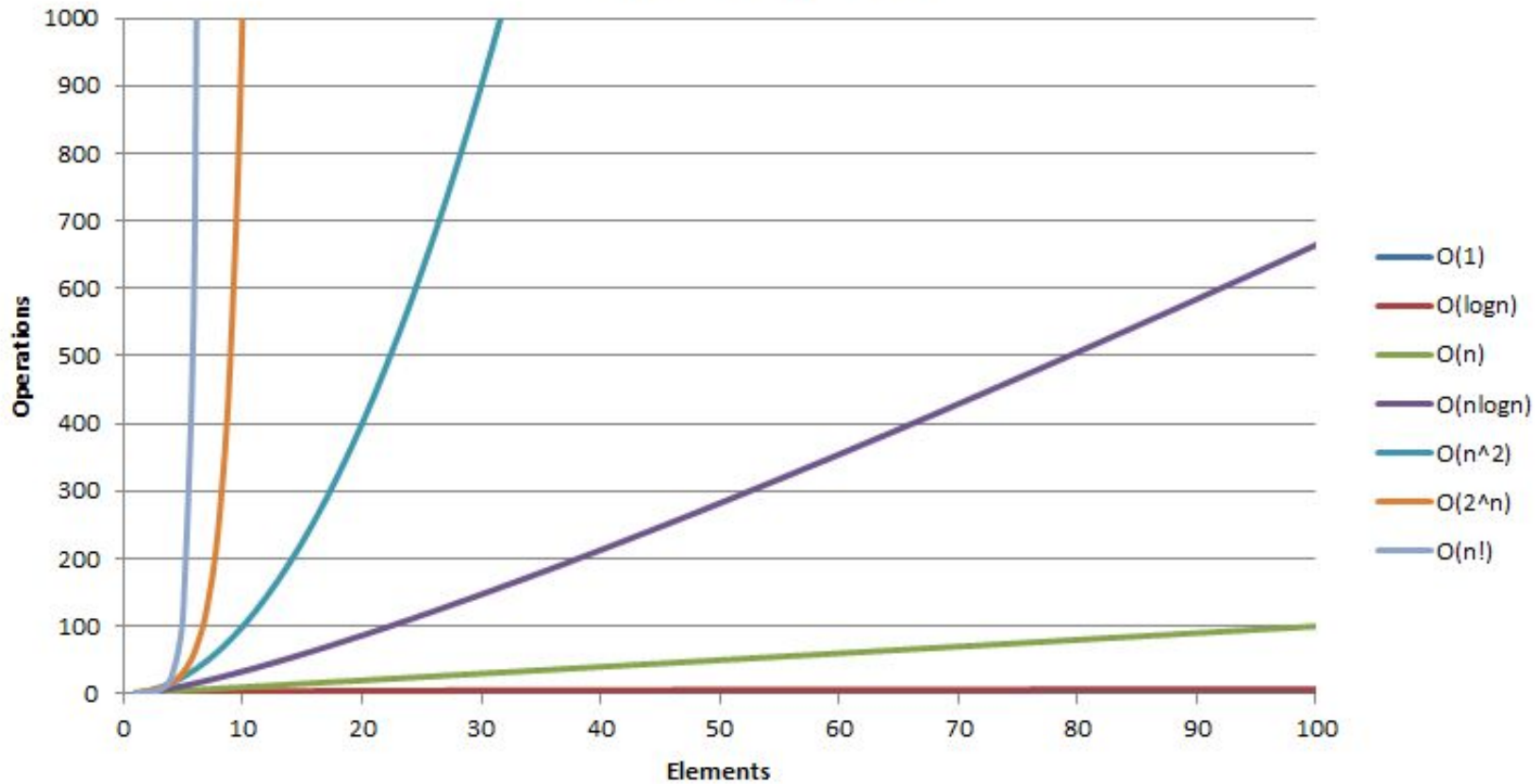
Qual a complexidade?

```
int funcao(int b, int M[][10]) {  
    soma = 0;  
  
    for (int i = 0; i < b; i++) {  
        soma += M[i][0];  
    }  
  
    return soma;  
}
```

$O(n)$

O tamanho b da matriz, influencia no número de operações executadas linearmente

Big-O Complexity



**Como procurar um
número em um
array ordenado,
sem ser $O(n)$**

Busca Binária

Objetivo: achar o número 18

Inicio

Fim

indices:

0

1

2

3

4

5

6

7

8

9

10

11

12

13

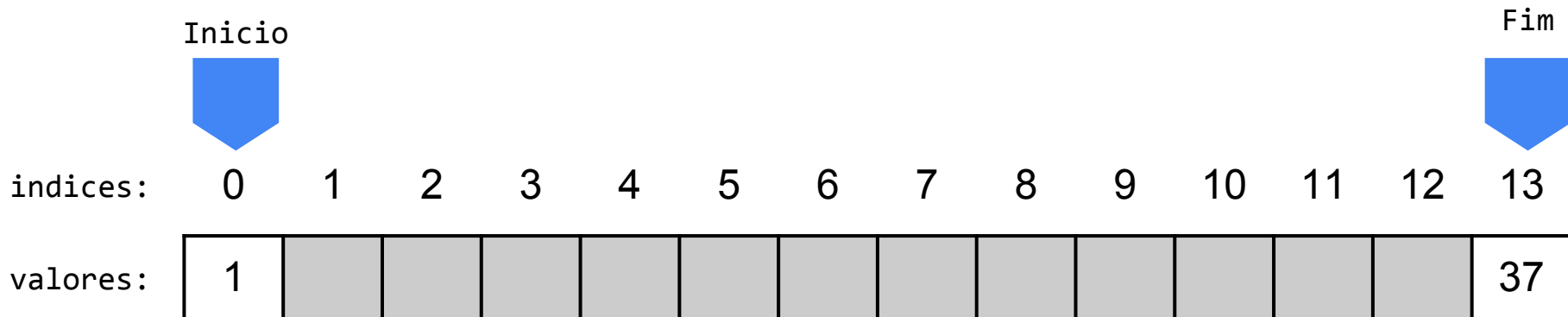
valores:

1

37

Busca Binária

Objetivo: achar o
número 18

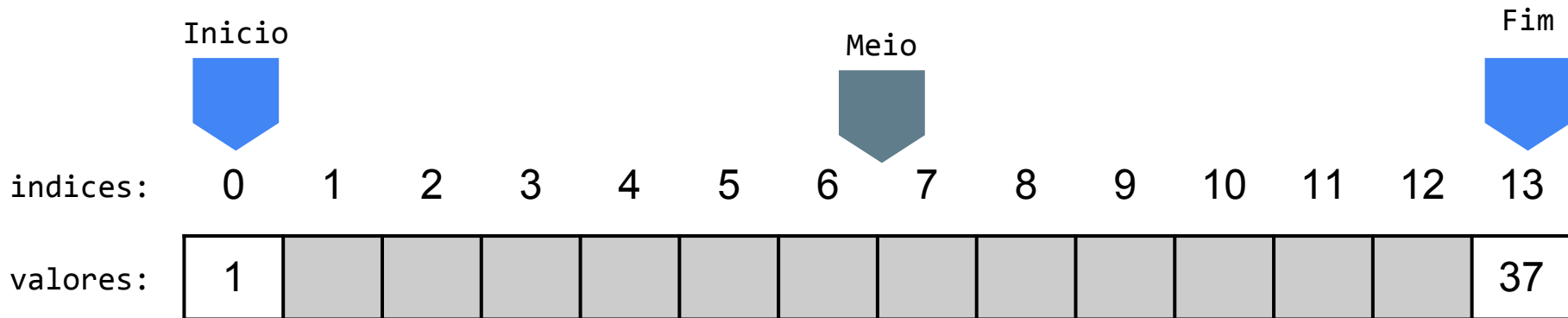


O array está **em ordem**
crescente

$A[0] < 18 < A[N - 1]$
18 está **entre os valores**
de início e fim.

Busca Binária

Objetivo: achar o
número 18

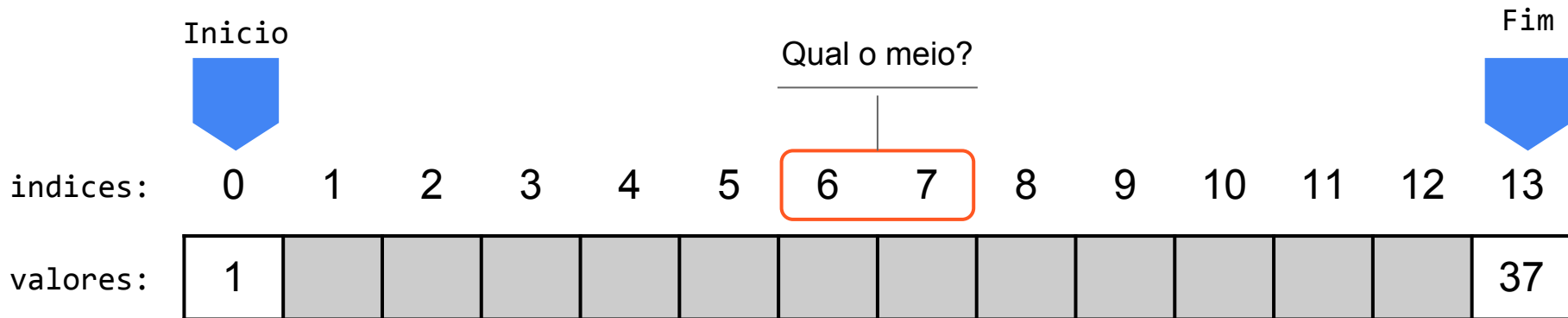


O array está **em ordem**
crescente

$A[0] < 18 < A[N - 1]$
18 está **entre os valores**
de início e fim.

Busca Binária

Objetivo: achar o
número 18

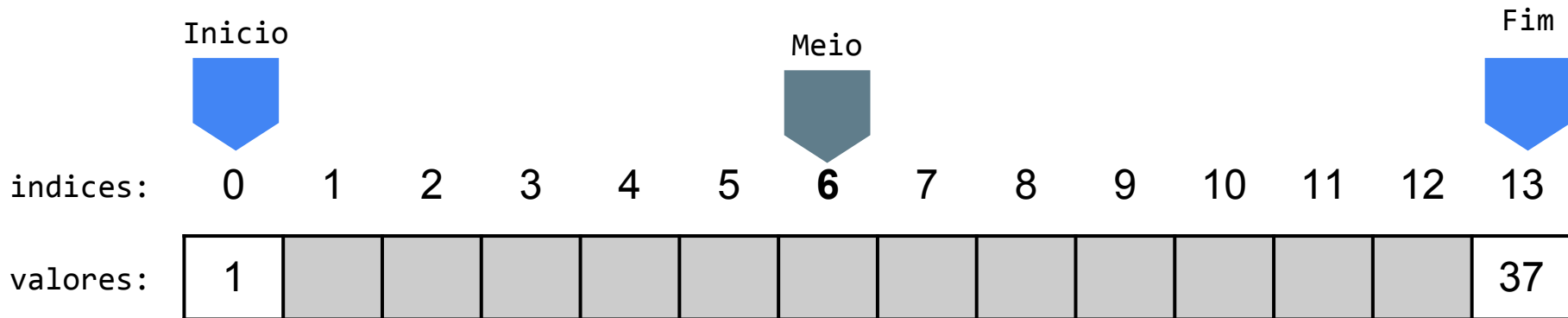


O array está **em ordem**
crescente

$A[0] < 18 < A[N - 1]$
18 está **entre os valores**
de início e fim.

Busca Binária

Objetivo: achar o
número 18






O array está **em ordem**
crescente

$A[0] < 18 < A[N - 1]$
18 está **entre os valores**
de início e fim.

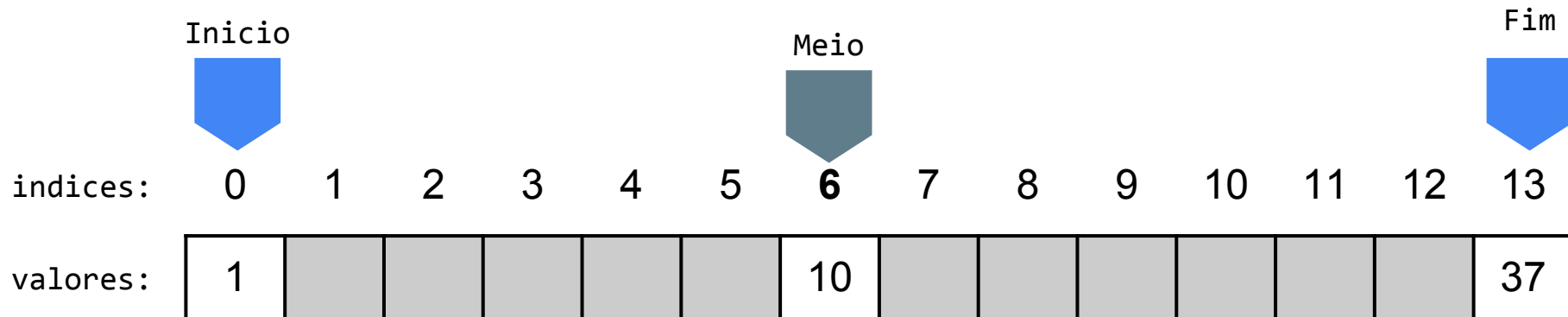
Busca Binária

Objetivo: achar o
número 18

	Início						Meio							Fim
														
índices:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
valores:	1						10							37

Busca Binária

Objetivo: achar o
número 18



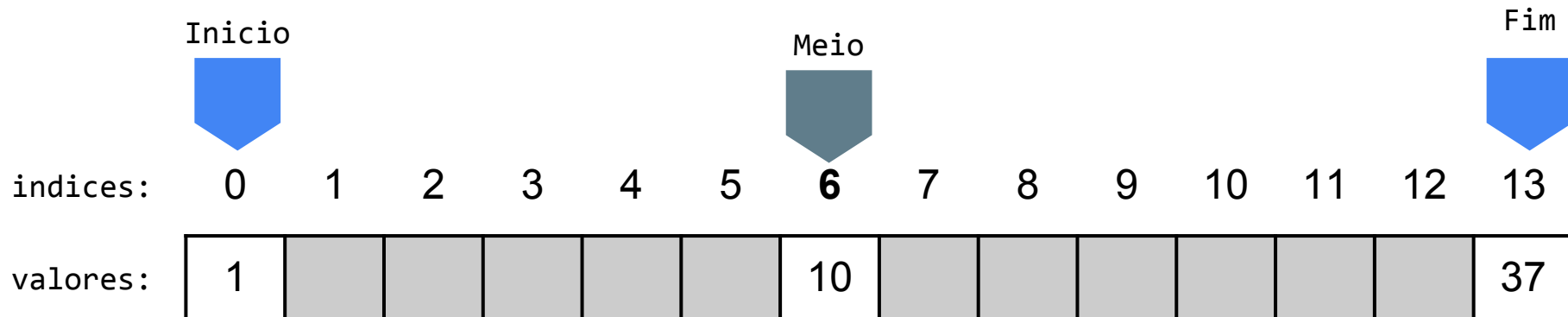
$18 > A[\text{meio}]$

ou

$18 < A[\text{meio}]$

Busca Binária

Objetivo: achar o
número 18



$18 > A[\text{meio}]$

ou



$18 < A[\text{meio}]$

$A[\text{meio}] < 18 < A[N - 1]$

18 está **entre os valores** de
meio e fim.

Busca Binária

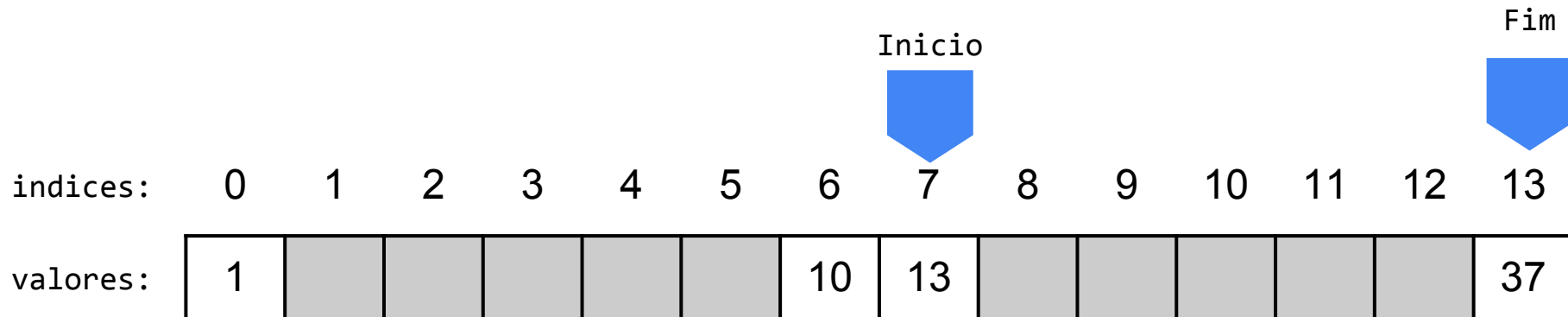
Objetivo: achar o
número 18

							Início							Fim
														
índices:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
valores:	1						10	13						37

Reiniciar o algoritmo a
partir do meio + 1.
 $\text{Início} = \text{meio} + 1$

Busca Binária

Objetivo: achar o
número 18






O array está **em ordem**
crescente

$A[\text{Início}] < 18 < A[\text{Fim}]$
18 está **entre os valores** de
início e fim.

Busca Binária

Objetivo: achar o
número 18

								Início			Meio			Fim
														
indices:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
valores:	1						10	13			19			37

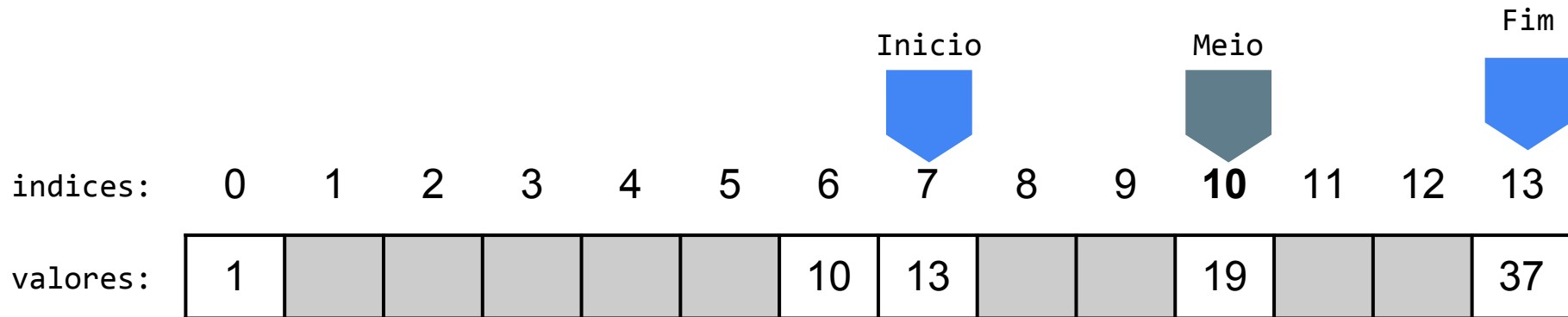
$18 > A[\text{meio}]$

ou

$18 < A[\text{meio}]$

Busca Binária

Objetivo: achar o
número 18



$18 > A[\text{meio}]$

ou



$18 < A[\text{meio}]$

$A[\text{Início}] < 18 < A[\text{meio}]$

18 está **entre os valores** de
início e meio.

Busca Binária



Objetivo: achar o
número 18

							Início		Fim					
														
índices:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
valores:	1						10	13		19	19			37

Reiniciar o algoritmo
até o meio - 1.
 $\text{Fim} = \text{meio} - 1$

Busca Binária

Objetivo: achar o
número 18




							Início		Fim					
														
índices:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
valores:	1						10	13		19	19			37

O array está **em ordem**
crescente

$A[\text{Início}] < 18 < A[\text{Fim}]$
18 está **entre os valores** de
início e fim.

Busca Binária

Objetivo: achar o
número 18

								Início	Meio	Fim				
														
índices:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
valores:	1						10	13	18	19	19			37




$18 > A[\text{meio}]$

ou

$18 < A[\text{meio}]$

Busca Binária

Objetivo: achar o
número 18

							Início	Meio	Fim					
														
índices:	0	1	2	3	4	5	6	7	8	9	10	11	12	13
valores:	1						10	13	18	19	19			37

$18 > A[\text{meio}]$

ou

$18 < A[\text{meio}]$

$18 == A[\text{meio}]$

**Objetivo
alcançado!**

Busca binária (pseudo código)

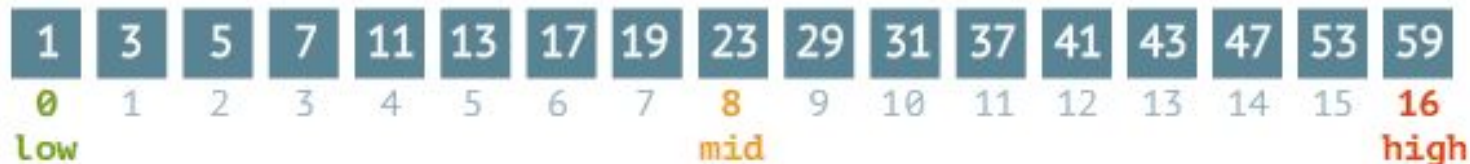
- 1) `inicio = 0, fim = tamanho - 1, objetivo = 18`
- 2) **SE** `inicio > fim`
 Para! O intervalo não é válido, então o objetivo não está dentro dele
- 3) `meio = (inicio + fim) / 2`
- 4) **SE** `A[meio] == objetivo`
 Cheguei ao objetivo. **Encerra** aqui!
- 5) **SE** `A[meio] < objetivo`
 `inicio = meio + 1. Volta` para 2
- 6) **SE** `A[meio] > objetivo`
 `fim = meio - 1. Volta` para 2

$O(\log n)$

Binary search

steps: 0

37



Sequential search

steps: 0

37



```
int inicio = 0, fim = N - 1;
int meio;
while (inicio <= fim) {
    meio = (inicio + fim)/2;

    if (array[meio] < numero) {
        inicio = meio + 1;
    }
    else if (array[meio] > numero) {
        fim = meio - 1;
    }
    else {
        // Encontrado
        break;
    }
}
// Não encontrado
```


Fim



$O(1)$