

# Busca sequencial e binária

Prof. Paulo Henrique Pisani

abril/2022

# Tópicos

- Busca linear/sequencial
- Busca binária

# Busca

- **Problema de busca:** dados um vetor  $v$  e um valor  $x$ , verificar se o elemento  $x$  está em  $v$ . Se estiver, retornar o índice  $i$  da posição de  $x$  em  $v$ . Caso contrário, retorne  $-1$ .

**Busca linear/sequencial**

# Busca linear (ou sequencial)

- Nessa estratégia de busca, verifica-se todos os elementos de um vetor para verificar se o elemento  $x$  está em  $v$ .

$x = 8$

$v =$ 

6	9	40	3	8	16
---	---	----	---	---	----



$6 == x?$

# Busca linear (ou sequencial)

- Nessa estratégia de busca, verifica-se todos os elementos de um vetor para verificar se o elemento  $x$  está em  $v$ .

$x = 8$

$v =$ 

6	9	40	3	8	16
---	---	----	---	---	----



$9 == x?$

# Busca linear (ou sequencial)

- Nessa estratégia de busca, verifica-se todos os elementos de um vetor para verificar se o elemento  $x$  está em  $v$ .

$x = 8$

$v =$ 

6	9	40	3	8	16
---	---	----	---	---	----



$40 == x?$

# Busca linear (ou sequencial)

- Nessa estratégia de busca, verifica-se todos os elementos de um vetor para verificar se o elemento  $x$  está em  $v$ .

$x = 8$

$v =$ 

6	9	40	3	8	16
---	---	----	---	---	----



$3 == x?$



# Busca linear (ou sequencial)

- Nessa estratégia de busca, verifica-se todos os elementos de um vetor para verificar se o elemento  $x$  está em  $v$ .

$x = 8$

$v =$ 

6	9	40	3	8	16
---	---	----	---	---	----



$8 == x?$  Sim  
Retorne 4 (índice do  
elemento no vetor  $v$ )

# Busca linear (ou sequencial)

- Algoritmo de busca linear:

```
int busca_linear(int *v, int n, int x) {  
    int i;  
    for (i = 0; i < n; i++) | Percorre todo o vetor.  
        if (v[i] == x) | Compara cada elemento com x.  
            return i;  
    return -1;  
}
```

**Quantas comparações de elementos do vetor realizamos nesse algoritmo?**


# Busca linear (ou sequencial)

- Algoritmo de busca (versão sem *return* no for):

```
int busca_linear2(int *v, int n, int x) {  
    int i, indice_encontrado = -1;  
    for (i = 0; i < n && indice_encontrado == -1; i++)  
        if (v[i] == x) {  
            indice_encontrado = i;  
        }  
    return indice_encontrado;  
}
```

Percorre todo o vetor.

Compara cada elemento com x.



# Análise da busca linear

- Número de comparações no pior caso ( $x$  é o último elemento ou não está presente no vetor):

$n$

- Número de comparações no melhor caso ( $x$  é o primeiro elemento):

1

# Busca binária

# Busca binária

- Algoritmo de busca mais eficiente, mas requer que o vetor esteja ordenado;
- A busca é realizada dividindo o vetor, até finalizar a busca.

# Busca binária

$x = 32$

	0	1	2	3	4	5	6	7	8	9
$v =$	3	6	9	10	18	25	28	32	38	40



esq



$$meio = \lfloor (esq + dir) / 2 \rfloor = 4$$

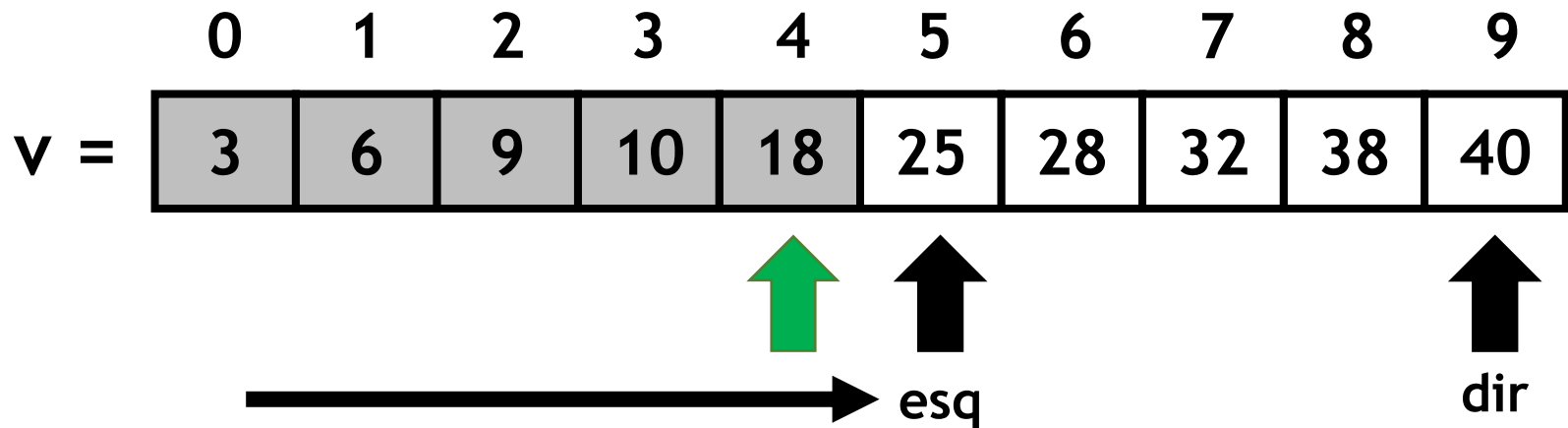
$v[meio] == 32?$  Não.



dir

# Busca binária

$x = 32$





# Busca binária

$x = 32$

	0	1	2	3	4	5	6	7	8	9
$v =$	3	6	9	10	18	25	28	32	38	40

↑ esq                      ↑                      ↑ dir

$$meio = \lfloor (esq + dir) / 2 \rfloor = 7$$

$v[meio] == 32?$  Sim.

Retorna o índice 7

# Busca binária (outro exemplo)

$x = 8$

	0	1	2	3	4	5	6	7	8	9
v =	3	6	9	10	18	25	28	32	38	40



esq



$$meio = \lfloor (esq + dir) / 2 \rfloor = 4$$

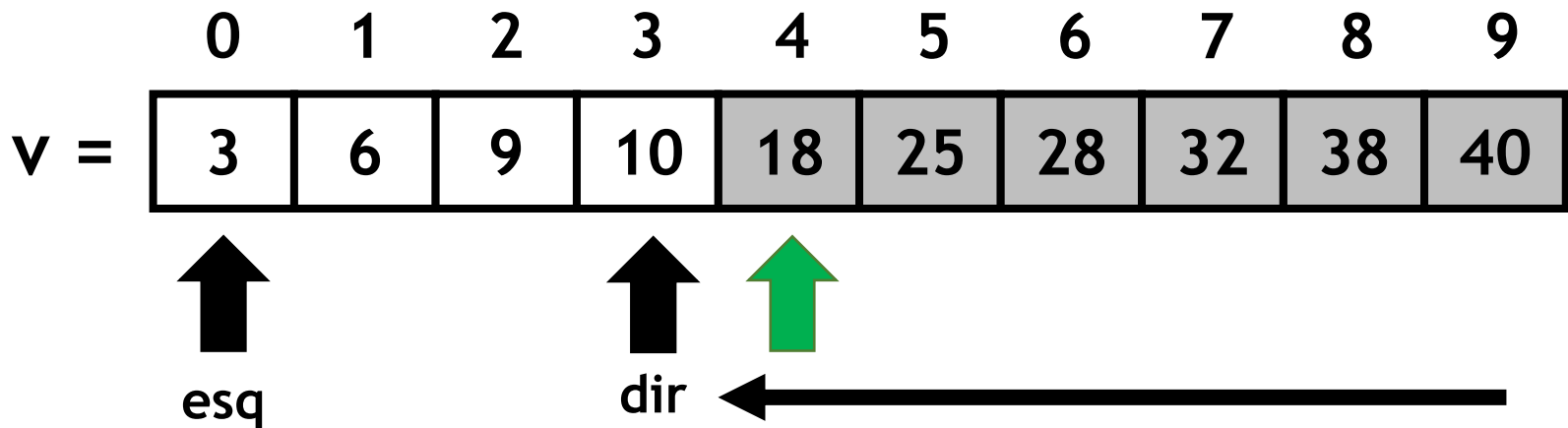
$v[meio] == 8$ ? Não.



dir

# Busca binária

$x = 8$



# Busca binária

**$x = 8$**

$v =$

0	1	2	3	4	5	6	7	8	9
3	6	9	10	18	25	28	32	38	40



esq



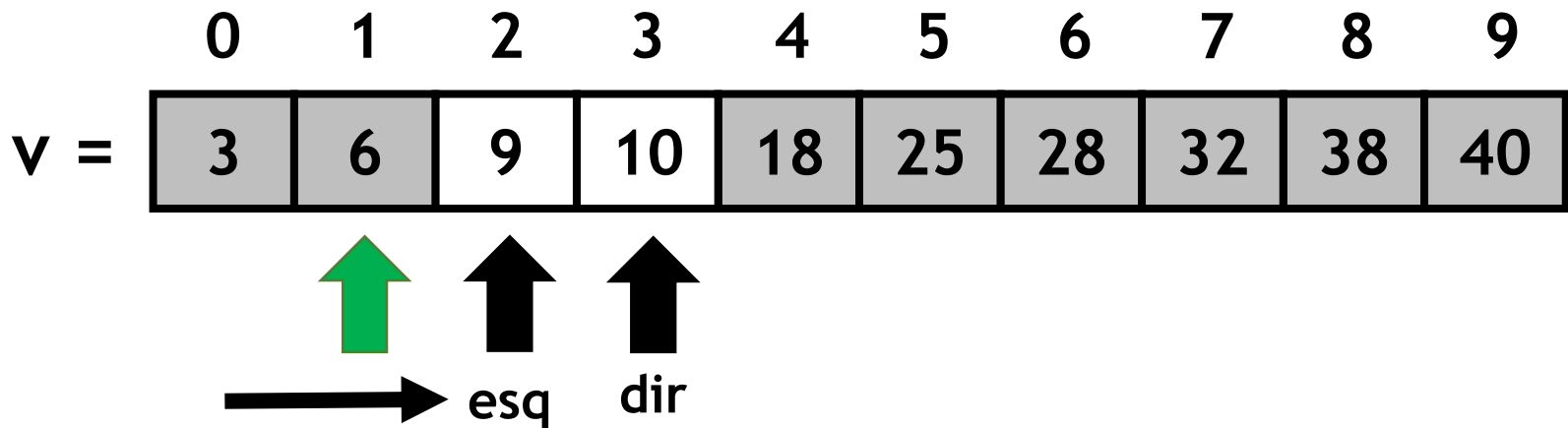
dir

$$meio = \left\lfloor \frac{(esq + dir)}{2} \right\rfloor = 1$$

**$v[meio] == 8?$  Não.**

# Busca binária

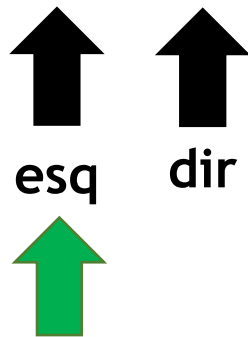
$x = 8$



# Busca binária

$x = 8$

	0	1	2	3	4	5	6	7	8	9
v =	3	6	9	10	18	25	28	32	38	40

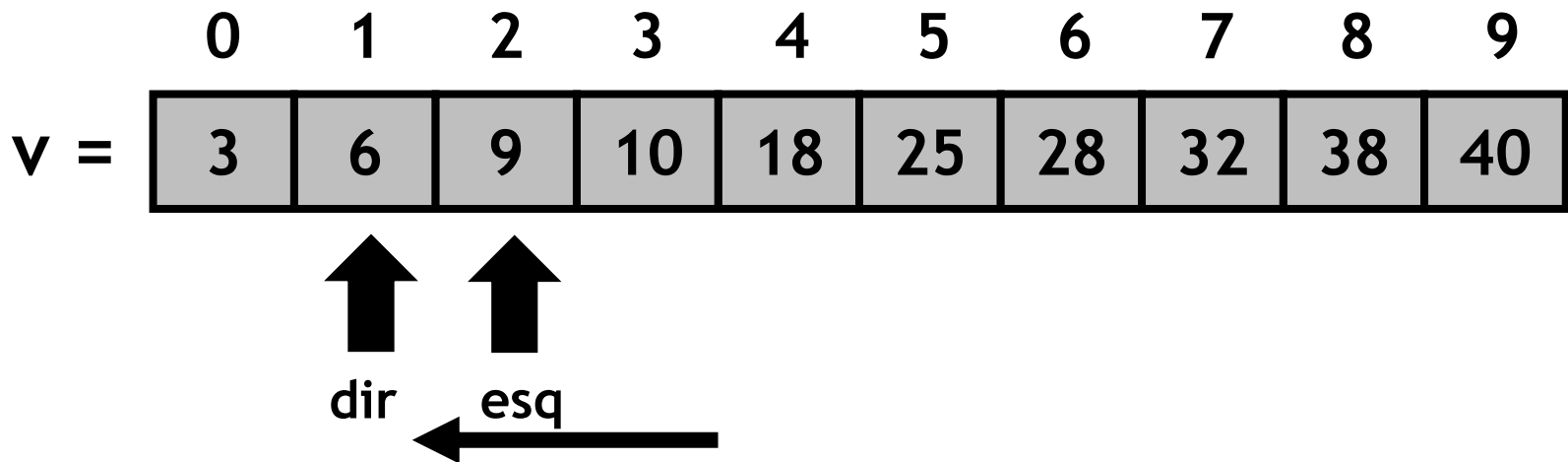


$$meio = \left\lfloor \frac{(esq + dir)}{2} \right\rfloor = 2$$

$v[meio] == 8$ ? Não.

# Busca binária

$x = 8$



$dir < esq \rightarrow$  Fim da busca. Elemento não foi encontrado! Retorne -1.

# Busca binária

- Algoritmo de busca binária:

```
int busca_binaria(int *v, int n, int x) {  
    int esq = 0, dir = n-1;  
    while (esq <= dir) {  
        int meio = (esq + dir) / 2;  
        if (v[meio] == x)  
            return meio;  
        else if (v[meio] < x)  
            esq = meio + 1;  
        else  
            dir = meio - 1;  
    }  
    return -1;  
}
```



# Complexidade da busca binária

- A cada iteração, o comprimento do vetor é reduzido pela metade:

- $n$       *Iteração 1*
- $\frac{n}{2}$       *Iteração 2*
- $\frac{n/2}{2}$       *Iteração 3*
- ...
- $\frac{n}{2^{k-1}}$       *Iteração k*

# Complexidade da busca binária

- A cada iteração, o comprimento do vetor é reduzido pela metade:

- $n$       *Iteração 1*

- $\frac{n}{2}$       *Iteração 2*

- $\frac{n/2}{2}$       *Iteração 3*

- ...

- $\frac{n}{2^{k-1}}$       *Iteração k*



Após k iterações, sobrará apenas um elemento

$$1 = \frac{n}{2^{k-1}}$$

# Complexidade da busca binária

- A cada iteração, o comprimento do vetor é reduzido pela metade:

- $n$       *Iteração 1*

- $\frac{n}{2}$       *Iteração 2*

- $\frac{n/2}{2}$       *Iteração 3*

- ...

- $\frac{n}{2^{k-1}}$       *Iteração k*



Após k iterações, sobrará apenas um elemento

$$k = \log_2(n) + 1$$



$$k - 1 = \log_2(n)$$



$$1 = \frac{n}{2^{k-1}}$$

# Complexidade da busca binária

- A cada iteração, o comprimento do vetor é reduzido pela metade:

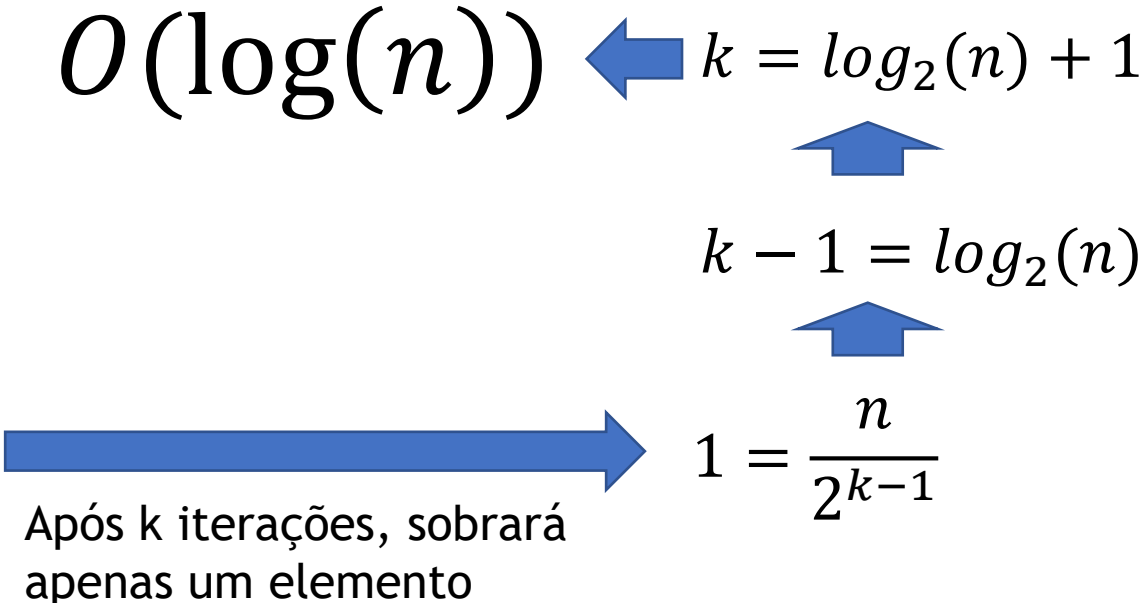
- $n$       *Iteração 1*
- $\frac{n}{2}$       *Iteração 2*
- $\frac{n/2}{2}$       *Iteração 3*
- ...
- $\frac{n}{2^{k-1}}$       *Iteração k*

$O(\log(n))$  ←  $k = \log_2(n) + 1$

↑  $k - 1 = \log_2(n)$

↑  $1 = \frac{n}{2^{k-1}}$

Após k iterações, sobrará apenas um elemento



# Referências

- Slides do Prof. Monael Pinheiro Riberio:
  - <https://sites.google.com/site/aed2018q1/>
- Slides do Prof. Jesús P. Mena-Chalco:
  - <http://professor.ufabc.edu.br/~jesus.mena/courses/mcta028-3q-2017/>
- Visualising Data Structures and algorithms through animation:
  - <https://visualgo.net/en>

# Referências

- Nivio Ziviani. Projeto de Algoritmos: com implementações em Pascal e C. Cengage Learning, 2015.
- Jayme L. Szwarcfiter, Lilian Markenzon. Estruturas de Dados e Seus Algoritmos. 3ª edição. LTC, 2012.
- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, Clifford Stein. Algoritmos: Teoria e Prática. Elsevier, 2012.