

# Estrutura de Dados 2

## Aula 4

### Busca binária



# Tópicos da aula

---

- **O problema da busca:**
  - Busca linear;
  - Busca binária.
- **Comparando soluções.**

# Buscando elementos

- **Problema da busca:**

- Instância:

- $(x, V, a, b)$ , onde  $x$  é um valor de busca e  $V[a .. b]$  é um vetor indexado por  $[a .. b]$ .

- Resposta:

- $m \in [a .. b]$  tal que  $V[m] = x$  ou  $-1$  se  $m \notin [a .. b]$ .

# Buscando elementos

- **Problema da busca:**

- Instância:

- $(x, V, a, b)$ , onde  $x$  é um valor de busca e  $V[a .. b]$  é um vetor indexado por  $[a .. b]$ .

- Resposta:

- $m \in [a .. b]$  tal que  $V[m] = x$  ou  $-1$  se  $m \notin [a .. b]$ .

- **Resumindo:**

Dada uma chave de busca  **$x$**  e uma arranjo de elementos  **$V$** , onde cada elemento possui um identificador único, desejamos encontrar o índice do elemento no arranjo que possui o identificador igual ao da chave de busca ou verificar que não existe nenhum elemento no arranjo com o identificador igual ao da chave de busca.

**Busca linear**

# Busca linear

---

- **Abordagem:**
  - Percorra o arranjo comparando a chave de busca com o valor de cada elemento.

# Busca linear

- **Abordagem:**

- Percorra o arranjo comparando a chave de busca com o valor de cada elemento.

- **Implementação iterativa:**

```
int buscaLinear(int v[], int a, int b, int x){  
    for (int i = a; i <= b; i++){  
        if (v[i] == x){  
            return i;  
        }  
    }  
    return -1;  
}
```

**Exemplo de entrada:**

```
int v[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
int x = 7, a = 0, b = 9;
```



**Saída esperada:**

índice 6 (v[6] = 7)

# Busca linear

- **Exemplo ( $x = 15$ ):**

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Número de comparações: 0


m: --



# Busca linear

- Exemplo ( $x = 15$ ):

$x == 6$




i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Número de comparações: 1

m: --

# Busca linear

- Exemplo ( $x = 15$ ):

$x == 10$   


i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Número de comparações: 2

m: --

# Busca linear

- Exemplo ( $x = 15$ ):

$x == 14$   
↓


i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Número de comparações: 3

m: --

# Busca linear

- Exemplo ( $x = 15$ ):

$x == 15$   


i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Número de comparações: 4

m: --

# Busca linear

- Exemplo ( $x = 15$ ):

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Número de comparações: **4**

m: **3**

# Busca linear

- Exemplo ( $x = 15$ ):

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Número de comparações: 4  
m: 3

**E se a chave de busca não estiver no vetor?**

# Busca linear

- **Exemplo ( $x = 16$ ):**

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90


Número de comparações: 0

m: --

# Busca linear

- Exemplo ( $x = 16$ ):

$x == 6$



i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90


Número de comparações: 1

m: --



# Busca linear

- Exemplo ( $x = 16$ ):

$x == 10$   



i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Número de comparações: 2

m: --

# Busca linear

- Exemplo ( $x = 16$ ):

$x == 14$   



i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Número de comparações: 3

m: --

# Busca linear

- Exemplo ( $x = 16$ ):

$x == 15$   


i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Número de comparações: 4

m: --

# Busca linear

- Exemplo ( $x = 16$ ):

$x == 23$   
↓


i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Número de comparações: 5

m: --

# Busca linear

- Exemplo ( $x = 16$ ):

$x == 48$   


i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90


Número de comparações: 6

m: --

# Busca linear

- Exemplo ( $x = 16$ ):

$x == 64$




i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Número de comparações: 7

m: --

# Busca linear

- Exemplo ( $x = 16$ ):

$x == 90$   


i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Número de comparações: 8

m: --

# Busca linear

- **Exemplo (x = 16):**

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Número de comparações: **8**

m: **-1**



# Busca linear

---


- **Custo computacional:**

- No pior caso teremos que percorrer todo o arranjo para encontrar o elemento desejado (sem ter a garantia de encontrá-lo).
  - Dizemos que o número de comparações no pior caso para a busca sequencial é  $O(n)$ .

# Busca linear

- **Custo computacional:**

- No pior caso teremos que percorrer todo o arranjo para encontrar o elemento desejado (sem ter a garantia de encontrá-lo).
  - Dizemos que o número de comparações no pior caso para a busca sequencial é  $O(n)$ .



*Big-O*: notação assintótica para indicar a pior cenário de execução do algoritmo com base no valor  $n$  (tamanho do problema).

# **Busca binária**

# Busca binária

- **Abordagem:**

- 1) Verifique se a chave de busca é igual ao valor da posição do meio do arranjo.
- 2) Se igual, devolva o índice da posição.
- 3) Se o valor da posição for maior que a chave de busca, repita o processo para os elementos do começo do arranjo até a posição anterior ao do meio.
- 4) Se o valor da posição for menor que a chave de busca, repita o processo para os elementos da posição seguinte a do meio até o final do arranjo.

# Busca binária

- **Abordagem:**

- 1) Verifique se a chave de busca é igual ao valor da posição do meio do arranjo.
- 2) Se igual, devolva o índice da posição.
- 3) Se o valor da posição for maior que a chave de busca, repita o processo para os elementos do começo do arranjo até a posição anterior ao do meio.
- 4) Se o valor da posição for menor que a chave de busca, repita o processo para os elementos da posição seguinte a do meio até o final do arranjo.

Condição de utilização: o arranjo deve estar ordenado.

# Busca binária

- Implementação iterativa:

```
int buscaBinaria(int v[], int a, int b, int x) {  
    while (a <= b) {  
        int m = a + (b - a) / 2;  
  
        if (v[m] == x) {  
            return m;  
        }  
        else if (v[m] < x) {  
            a = m + 1;  
        }  
        else {  
            b = m - 1;  
        }  
    }  
  
    return -1;  
}
```

**Exemplo de entrada:**

int v[] = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};  
int x = 7, a = 0, b = 9;



**Saída esperada:**

índice 6 (v[6] = 7)

# Busca binária

- Exemplo ( $x = 65$ ):

	a				b			
i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

**n (tamanho da instância) = 8**

Número de comparações: 0

m: --

# Busca binária

- Exemplo ( $x = 65$ ):

	a				b			
i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

$x == 65$

	a				b			
i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

$< 65$                        $> 65$

$n$  (tamanho da instância) = 8

Número de comparações: 0

m: --

Elemento do meio =  $\lfloor (a + b)/2 \rfloor = 3$

Número de comparações: 1

m: --



# Busca binária

- Exemplo ( $x = 65$ ):

	a				b			
i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

$x == 65$

	a				b			
i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

$< 65$        $> 65$

	a				b			
i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

**n (tamanho da instância) = 8**

Número de comparações: 0

m: --

Elemento do meio =  $\lfloor (a + b)/2 \rfloor = 3$

Número de comparações: 1

m: --

**n (tamanho da instância) = 4**

Número de comparações: 1

m: --

# Busca binária

- Exemplo ( $x = 65$ ):

$x == 65$

a                      b

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

$< 65$                        $> 65$

Elemento do meio =  $\lfloor (a + b)/2 \rfloor = 5$   
Número de comparações: **2**  
m: --

# Busca binária

- Exemplo ( $x = 65$ ):

$x == 65$

a                      b

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

< 65

> 65

a                      b

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Elemento do meio =  $\lfloor (a + b)/2 \rfloor = 5$

Número de comparações: 2

m: --

**n (tamanho da instância) = 2**

Número de comparações: 2

m: --

# Busca binária

- Exemplo ( $x = 65$ ):

$x == 65$

a                      b

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

< 65                      > 65

Elemento do meio =  $\lfloor (a + b)/2 \rfloor = 5$   
Número de comparações: 2  
m: --

a                      b

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

**n (tamanho da instância) = 2**  
Número de comparações: 2  
m: --

$x == 65$

b

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

< 65                      > 65

Elemento do meio =  $\lfloor (a + b)/2 \rfloor = 6$   
Número de comparações: 3  
m: --

# Busca binária

- Exemplo ( $x = 65$ ):

								a == b
i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

**n (tamanho da instância) = 1**

Número de comparações: 3

m: --

# Busca binária


- Exemplo ( $x = 65$ ):

$a == b$

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

$n$  (tamanho da instância) = 1  
Número de comparações: 3  
m: --

$x == 65$



i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Elemento do meio =  $\lfloor (a + b)/2 \rfloor = 7$   
Número de comparações: 4  
m: --

caso base

# Busca binária


- Exemplo ( $x = 65$ ):

$a == b$

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

**n (tamanho da instância) = 1**  
Número de comparações: 3  
m: --

$x == 65$



i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

Elemento do meio =  $\lfloor (a + b)/2 \rfloor = 7$   
Número de comparações: 4  
m: --

caso base

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

**n (tamanho da instância) = 0**  
Número de comparações: 4  
m: -1

# Busca binária

- **Custo computacional:**

- Métrica de avaliação: número de comparações realizadas.

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

n (tamanho da instância) = 8  
Número de comparações: 4

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

n (tamanho da instância) = 4  
Número de comparações: 3

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

n (tamanho da instância) = 2  
Número de comparações: 2

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

n (tamanho da instância) = 1  
Número de comparações: 1

i	0	1	2	3	4	5	6	7
V[i]	6	10	14	15	23	48	64	90

n (tamanho da instância) = 0  
Número de comparações: 0



# Busca binária

- Custo computacional:**

- Métrica de avaliação: número de comparações realizadas.

i	0	n	$\log_2 n + 1$	7
V[i]	6	1	1	90
i	0	2	2	7
V[i]	6	4	3	90
i	0	8	4	7
V[i]	6	16	5	90
i	0	...	...	7
V[i]	6	512	10	90
i	0	1024	11	7
V[i]	6	...	...	90

n (tamanho da instância) = 8  
Número de comparações: 4

n (tamanho da instância) = 4  
Número de comparações: 3

n (tamanho da instância) = 2  
Número de comparações: 2

n (tamanho da instância) = 1  
Número de comparações: 1

n (tamanho da instância) = 0  
Número de comparações: 0

# Busca binária

- **Custo computacional:**
  - Métrica de avaliação: número de comparações realizadas.

i	0	n	$\log_2 n + 1$	7	n (tamanho da instância) = 8 Número de comparações: 4
V[i]	6	1	1	90	
i	0	2	2	7	n (tamanho da instância) = 4 Número de comparações: 3
V[i]	6	4	3	90	
i	0	8	4	7	n (tamanho da instância) = 2 Número de comparações: 2
V[i]	6	16	5	90	
i	0	...	...	7	n (tamanho da instância) = 1 Número de comparações: 1
V[i]	6	512	10	90	
i	0	1024	11	7	n (tamanho da instância) = 0 Número de comparações: 0
V[i]	6	...	...	90	

$O(\log_2 n)$  ←

# Busca binária

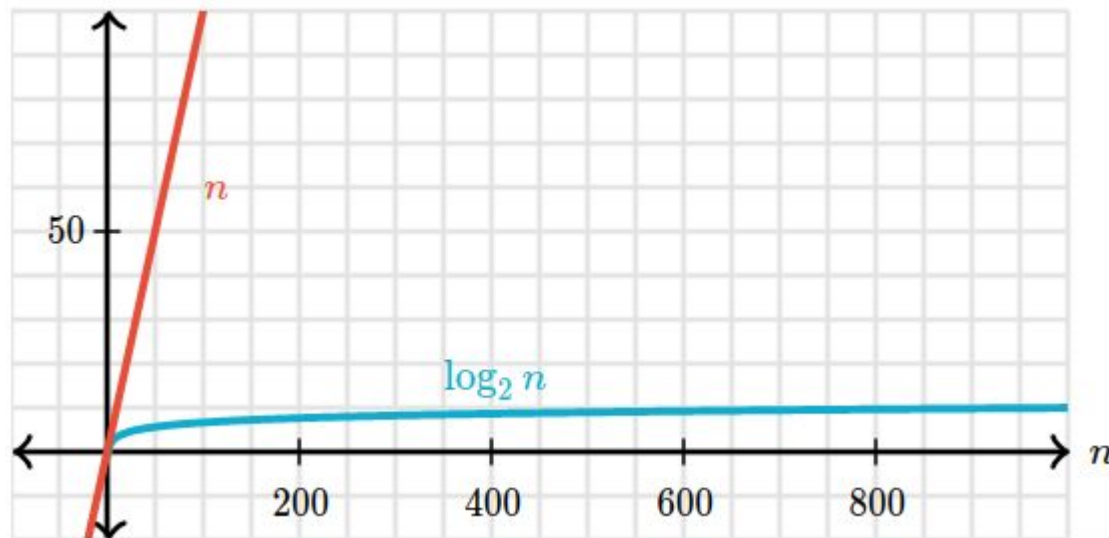
- **Custo computacional:**

- Para se ter uma ideia da diferença de eficiência dos dois algoritmos, considere um arranjo com um milhão de itens ( $10^6$  itens).
- Com a busca linear, considerando o pior caso teríamos:
  - $O(n) = 10^6$  comparações.
- Com a busca binária teríamos:
  - $O(\log_2 n) = \log_2 10^6 = 19,9316$  comparações

# Busca binária

- **Custo computacional:**

- Outra forma de analisar a diferença entre o desempenho dos dois algoritmos de busca é a partir da comparação das suas funções de crescimento.



# Hora de praticar!

---

## **Exercício 1:**

Implemente a versão iterativa da busca linear e da busca binária. Use como entrada um vetor de 1000 posições inicializado randomicamente com valores inteiros entre 0 e 5000 (sem repetição de valores). Compare o tempo de execução da busca linear e da busca binária.

*Importante:* o vetor deve estar ordenado para execução da busca binária.

## **Exercício 2:**

Implemente a versão recursiva da busca linear e da busca binária.

# Hora de praticar!

## Código para geração de números randômicos:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(){
    int i, cont = 0;
    srand(time(NULL));

    for(i = 0; i < 10; i++){
        .....
        printf("%d\n", 10 + rand() % 191);
    }

    return 0;
}
```

Altera o valor da semente  
randômica a cada execução  
do programa

# Hora de praticar!

## Código para geração de números randômicos:

```
#include <stdio.h>
#include <stdlib.h>
#include <time.h>

int main(){
    int i, cont = 0;
    srand(time(NULL));

    for(i = 0; i < 10; i++){
        printf("%d\n", 10 + rand() % 191);
    }

    return 0;
}
```

Gerador de números  
aleatórios, de 10 a 190

# Hora de praticar!

## Código para medir tempo de execução em c:

```
#include <time.h>

int main() {

    clock_t ticks[2];
    ticks[0] = clock();

    //Código da sua aplicação.

    ticks[1] = clock();
    double tempo = (ticks[1] - ticks[0]) * 1000.0 / CLOCKS_PER_SEC;

    printf("Tempo gasto: %g ms.", tempo);

    return 0;
}
```



**Dúvidas?**