

# Busca em Memória Principal

Estrutura de Dados e Algoritmos – Ciência da Computação



**INSTITUTO  
FEDERAL**  
Brasília

Prof. Daniel Saad Nogueira  
Nunes

IFB – Instituto Federal de Brasília,  
Campus Taguatinga



# Sumário

---

- 1 Introdução
- 2 Busca Binária
- 3 Análise



# Sumário

---

## 1 Introdução



# Introdução

---

- Dada uma sequência de entrada, como buscar um determinado elemento?
- Podemos utilizar um laço de repetição.



# Busca Sequencial

---

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



# Busca Sequencial

---

- O algoritmo de busca sequencial funciona.
- No pior caso ele tem que varrer o vetor inteiro.
- Tempo  $\Theta(n)$ .
- Temos como fazer melhor?



## Busca Sequencial Otimizada

---

- Se a entrada já estiver **ordenada**, podemos melhorar a busca sequencial.
- Assim que encontramos um elemento que é maior do que a chave, sabemos que ela não estará na sequência, pois a entrada está ordenada.



# Busca Sequencial

```
1  /** Pré-requisitos, v está ordenado **/
2  int busca_sequencial_otimizada(int *v, size_t n, int key) {
3      int i;
4      for (i = 0; i < n; i++) {
5          if (key < v[i]) {
6              break;
7          } else if (v[i] == key) {
8              return i;
9          }
10     }
11     return -1;
12 }
```





## Busca Sequencial Otimizada

---

- Conseguimos salvar algum tempo caso detectemos que a chave é menor do que algum elemento durante a varredura na busca sequencial otimizada.
- Contudo, no pior caso, ainda temos que varrer o vetor inteiro.
- Como tirar mais vantagem do fato da entrada estar ordenada?



## Busca Sequencial Otimizada

---

- Conseguimos salvar algum tempo caso detectemos que a chave é menor do que algum elemento durante a varredura na busca sequencial otimizada.
- Contudo, no pior caso, ainda temos que varrer o vetor inteiro.
- Como tirar mais vantagem do fato da entrada estar ordenada?
- **Busca Binária!**



# Sumário

---

## 2 Busca Binária



# Busca Binária

---

- Levando em consideração que o vetor está ordenado, podemos efetuar a busca binária.
- Ela funciona da seguinte forma. Suponha que a sequência  $V[0, n - 1]$ .
- Inicialmente calcula-se o ponto médio da sequência:  $m \leftarrow \lfloor \frac{n}{2} \rfloor$ .
- Se a chave corresponde à  $V[m]$ , então encontramos a chave no ponto médio.



## Busca Binária

---

- Se a chave não é igual ao elemento  $V[m]$ , temos duas opções.
  - ① A chave é menor do que  $V[m]$ .
  - ② A chave é maior do que  $V[m]$ .
- No primeiro caso, sabemos que se a chave se encontra em  $V$ , ela deve estar no intervalo  $V[0, m - 1]$ .
- No segundo, concluímos que se a chave está em  $V$ , ela se encontra em  $V[m + 1, n - 1]$ .
- Isso é permitido pois sabemos que o vetor está ordenado. Logo, sabemos que todos os elementos à esquerda de  $V[m]$  são menores ou iguais à  $V[m]$ . Simetricamente, todos os elementos à direita de  $V[m]$  são maiores ou iguais a  $V[m]$ .



# Busca Binária

---

- Caso a chave não corresponda ao elemento  $V[m]$ , continuamos a busca no subvetor à esquerda de  $V[m]$  ou à direita de  $V[m]$  utilizando a mesma estratégia!
- Descartamos metade dos elementos com uma única comparação!



# Busca Binária

---

## Exemplo

*chave* = 4

0	1	2	3	4	5	6	7	8
1	3	4	6	7	8	10	13	14







# Busca Binária

---

## Exemplo

$chave = 4$

0	1	2	3	4	5	6	7	8
1	3	4	6	7	8	10	13	14

$l$                        $m$                                        $r$

$chave > V[m]$



# Busca Binária

---

## Exemplo

*chave* = 4

0	1	2	3	4	5	6	7	8
1	3	4	6	7	8	10	13	14

↑  
*l*  
*m*

↑  
*r*  
*chave* == *V*[*m*]



# Busca Binária

---

## Exemplo

*chave* = 11

0	1	2	3	4	5	6	7	8
1	3	4	6	7	8	10	13	14





# Busca Binária

## Exemplo

*chave* = 11

0	1	2	3	4	5	6	7	8
1	3	4	6	7	8	10	13	14

$l$                        $m$                        $r$

*chave* >  $V[m]$



# Busca Binária

---

## Exemplo

*chave* = 11

0	1	2	3	4	5	6	7	8
1	3	4	6	7	8	10	13	14

*chave* <  $V[m]$

$l$                        $r$   
 $m$

The diagram shows a sorted array of 9 elements. The search key is 11. The current search range is from index  $l=7$  to  $r=8$ . The middle element  $V[m]$  is at index  $m=7$  with value 13. Since  $11 < 13$ , the search range will be updated to  $l=7$  and  $r=6$  in the next step.



# Busca Binária

---

## Exemplo

*chave* = 11

0	1	2	3	4	5	6	7	8
1	3	4	6	7	8	10	13	14

$r$                        $l$

Chave não encontrada!



# Busca Binária

```
1  int busca_binaria(int *v, size_t n, int key) {
2      int l = 0;
3      int r = n - 1;
4      while (l <= r) {
5          int mid = l + (r - l) / 2;
6          if (key == v[mid]) {
7              return mid; /**Retorna a posição da chave**/
8          } else if (chave < v[mid]) {
9              r = mid - 1;
10         } else {
11             l = mid + 1;
12         }
13     }
14     return -1; /**Chave não encontrada**/
15 }
```





## Busca Binária: Recursão

---

- Também é possível implementar a busca binária recursivamente.
- Casos base: subvetor vazio ou chave encontrada.



# Busca Binária: Recursão

---

```
1 int busca_binaria_rec(int *v, size_t n, int key) {  
2     return busca_binaria_rec_helper(v, 0, n - 1, key);  
3 }
```



## Busca Binária: Recursão

```
1 int busca_binaria_rec_helper(int *v, int l, int r, int key) {
2     if (l > r) /**Caso base, vetor vazio**/
3         return -1;
4     int mid = l + (r - l) / 2;
5     /**Caso base, a chave é igual ao elemento central**/
6     if (key == v[mid])
7         return mid;
8     if (key < v[mid]) /**Recursão na metade inferior**/
9         return busca_binaria_rec_helper(v, l, mid - 1, key);
10    else /**Recursão na metade superior**/
11        return busca_binaria_rec_helper(v, mid + 1, r, key)
12 }
```



# Sumário

---

## 3 Análise



## Análise

---

- No pior caso, a busca binária descarta metade dos elementos a cada comparação.
- $\Theta(\lg n)$  passos.

Número de Comparações		
$n$	Busca Sequencial	Busca Binária
$2^1$	$2^1 - 1$	2
$2^2$	$2^2 - 1$	3
$2^3$	$2^3 - 1$	4
$2^4$	$2^4 - 1$	5
$2^{10}$	$2^{10} - 1$	11
$2^{20}$	$2^{20} - 1$	21
$2^{30}$	$2^{30} - 1$	31