

**Instituto de  
Computação**

UNIVERSIDADE ESTADUAL DE CAMPINAS



**MC102 - Aula 17**

**Algoritmos de Busca**

Algoritmos e Programação de Computadores

Turmas

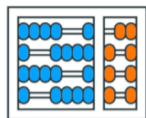
**OVXZ**

**Prof. Lise R. R. Navarrete**

[lrommel@ic.unicamp.br](mailto:lrommel@ic.unicamp.br)

Quinta-feira, 19 de maio de 2022

19:00h - 21:00h (CB06)



**Instituto de  
Computação**

UNIVERSIDADE ESTADUAL DE CAMPINAS



UNICAMP

**MC102 – Algoritmos e Programação de Computadores**

---

Turmas

**OVXZ**

<https://ic.unicamp.br/~mc102/>

Site da Coordenação de MC102

Aulas teóricas:

Terça-feira, 21:00h - 23:00h (CB06)

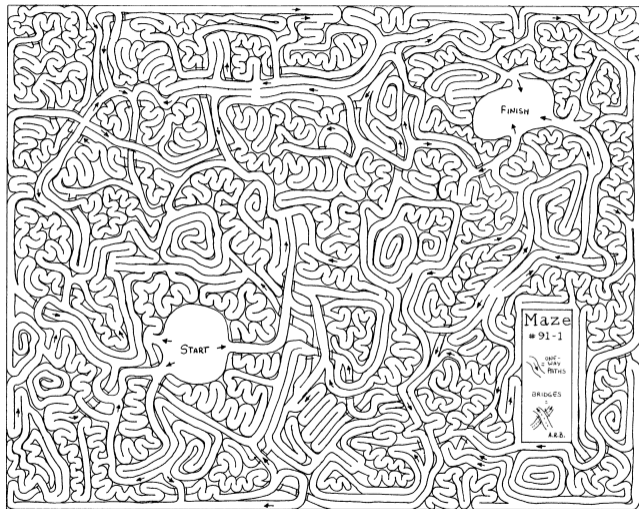
Quinta-feira, 19:00h - 21:00h (CB06)

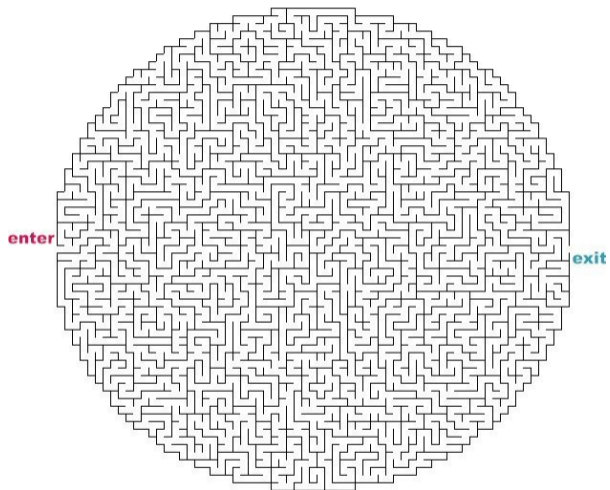
# Conteúdo

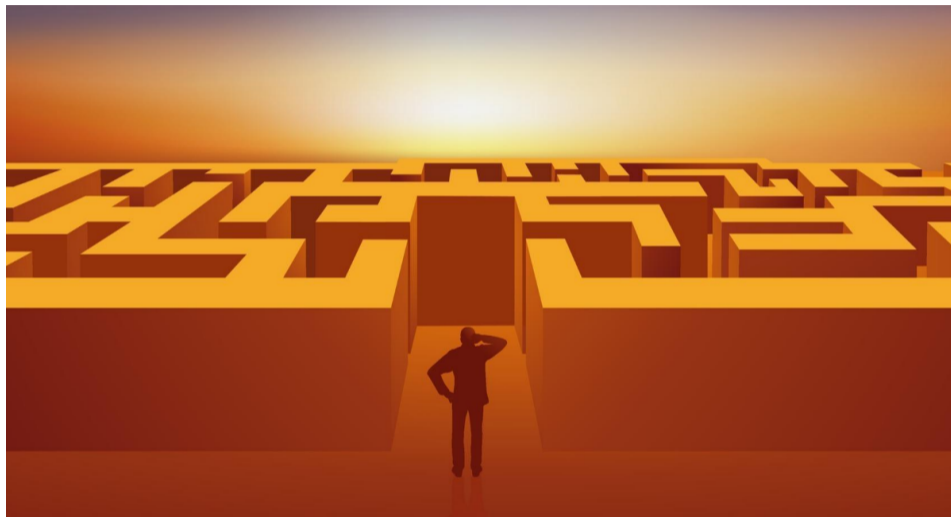
- O problema da Busca
- Busca Sequencial
- Eficiência da Busca Sequencial
- Busca Binaria
- Eficiência da Busca Binaria
- Exemplo comparativo
- Exercícios

# O problema da Busca

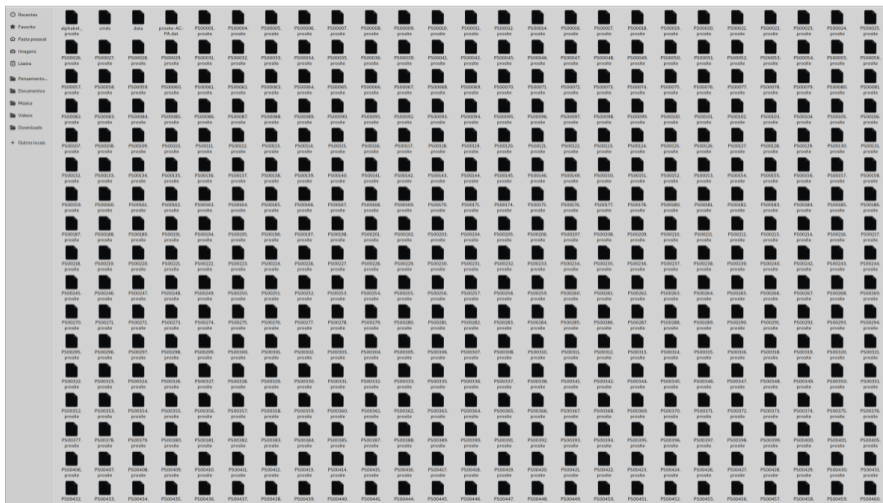


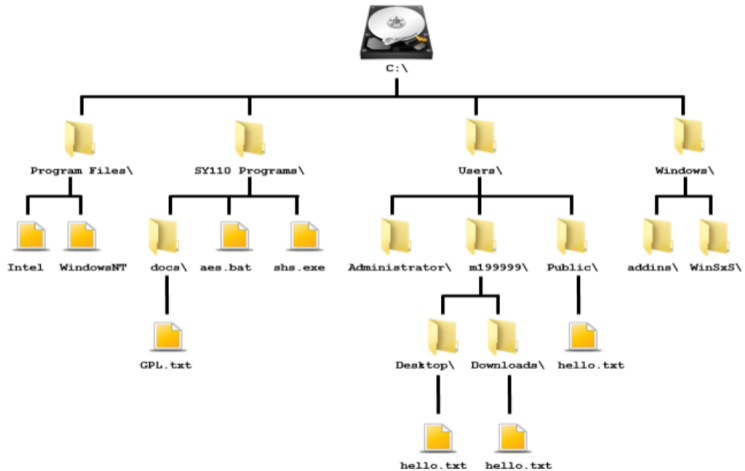












S	U	I	R	A	T	T	I	G	A	S	E	C	I	N	E	R	E	B	A	M	O	C	H	A
U	M	S	C	E	P	H	E	U	S	D	L	A	A	R	O	N	I	M	S	I	N	A	C	S
N	O	I	R	O	I	R	G	O	E	A	N	M	E	S	U	P	U	L	C	I	R	A	C	
I	N	X	S	V	S	A	S	R	O	I	R	C	E	T	U	S	G	S	I	R	L	I	E	U
H	O	Y	A	K	U	E	N	C	R	A	T	E	R	T	N	A	U	C	R	O	S	N	L	L
P	C	P	O	S	P	F	C	O	I	M	S	R	A	L	I	H	A	A	U	S	N	A	U	P
L	E	T	K	S	E	T	O	O	B	S	U	N	R	O	C	I	R	P	A	C	A	S	M	T
E	R	E	S	I	L	A	R	T	S	U	A	A	N	O	R	O	C	E	L	O	T	E	U	O
D	O	B	X	N	Y	L	O	A	A	L	M	U	V	R	I	A	S	G	I	P	X	C	T	R
R	S	A	A	M	R	O	N	U	L	D	U	E	S	E	C	B	E	A	A	I	E	S	U	O
A	E	E	I	R	A	A	R	B	I	L	A	N	T	P	E	R	S	E	U	S	I	C	J	
C	L	Q	E	B	R	S	B	U	E	A	U	T	E	L	R	A	P	U	S	M	G	P	S	A
O	U	P	E	O	N	O	S	U	N	G	Y	C	A	R	I	E	S	I	P	P	P	M		
L	C	U	O	J	E	R	I	D	A	N	U	S	S	R	O	N	I	M	O	E	L	E	A	
U	R	L	I	C	A	M	E	L	O	P	A	R	D	A	L	I	S	I	P	R	O	C	S	
M	E	E	S	H	M	W	A	N	T	L	I	A	A	Z	N	D	E	N	S	E	O	B	E	R
B	H	U	S	A	S	G	L	A	C	E	R	T	A	R	D	Y	H	A	I	E	G	N	N	U
A	Y	S	A	M	I	N	I	M	E	G	T	A	P	H	O	E	N	I	X	U	R	C	T	R
N	D	N	C	A	N	E	S	V	E	N	A	T	I	C	I	D	H	A	T	T	I	G	A	S
A	R	A	G	E	A	Q	U	A	R	I	U	S	H	U	R	S	R	A	B	I	V	Z	U	A
C	U	T	A	L	C	O	R	V	U	S	U	R	G	O	M	U	L	U	C	I	T	E	R	M
U	S	C	T	E	L	E	S	C	O	P	I	U	M	R	S	D	R	O	T	C	I	P	U	I
T	H	O	R	O	L	O	G	I	U	M	E	F	O	R	N	A	X	H	A	I	L	S	N	
T	R	I	A	N	G	U	L	U	M	O	D	A	R	O	D	I	S	U	C	U	I	H	P	O
A	G	I	R	U	A	Q	U	I	L	A	L	U	C	E	P	L	U	V	O	L	A	N	S	R

Ara • **Aries** • Auriga • Boötes • Caelum • Camelopardalis • Cancer • Canes Venatici • Canis Major • Canis Minor • Capricornus • Carina • Cassiopeia • Centaurus • Cepheus • Cetus • Chamaeleon • Circinus • Columba • Coma Berenices • Corona Australis • Corona Borealis • Corvus • Crater • Crux • Cygnus • Delphinus • Dorado • Draco • Equuleus • Eridanus • Fornax • Gemini • Grus • Hercules • Horologium • Hydra • Hydrus • Indus • Lacerta • Leo • Leo Minor • Lepus • Libra • Lupus • Lynx • Lyra • Mensa • Microscopium • Monoceros • Musca • Norma • Octans • Ophiuchus • Orion • **Pavo** • Pegasus • Perseus • Phoenix • Pictor • Pisces • Piscis Austrinus • Puppis • Pyxis • Reticulum • Sagitta • Sagittarius • Scorpius • Sculptor • Scutum • Serpens • Sextans • Taurus • Telescopium • Triangulum • **Triangulum Australe** • Tucana • Ursa Major • Ursa Minor • Vela • Virgo • Volans • Vulpecula









<https://ic.unicamp.br/~mc102/aulas/aula11.pdf>

- Vamos estudar alguns algoritmos para o seguinte problema:

### Definição do Problema

Dada uma chave de busca e uma coleção de elementos, onde cada elemento possui um identificador único, desejamos encontrar o elemento da coleção que possui o identificador igual ao da chave de busca ou verificar que não existe nenhum elemento na coleção com a chave fornecida.

- Nos nossos exemplos, a coleção de elementos será representada por uma lista de inteiros.
  - O identificador do elemento será o próprio valor de cada elemento.
- Apesar de usarmos inteiros, os algoritmos que estudaremos servem para buscar elementos em qualquer coleção de elementos que possuam chaves que possam ser comparadas.



- O problema da busca é um dos mais básicos na área de Computação e possui diversas aplicações.
  - Buscar um aluno dado o seu RA.
  - Buscar um cliente dado o seu CPF.
  - Buscar uma pessoa dado o seu RG.
- Estudaremos algoritmos simples para realizar a busca assumindo que os dados estão em uma lista.
- Existem estruturas de dados e algoritmos mais complexos utilizados para armazenar e buscar elementos. Estas abordagens não serão estudadas nesta disciplina.

<https://ic.unicamp.br/~mc102/aulas/aula11.pdf>

- Vamos criar uma função `busca(lista, chave)`:
  - A função deve receber uma `lista` de números inteiros e uma `chave` para busca.
  - A função deve retornar o índice da lista que contém a chave ou o valor `-1`, caso a chave não esteja na lista.

chave = 45

lista	20	5	15	24	67	45	1	76	21	11
	0	1	2	3	4	5	6	7	8	9

chave = 100

lista	20	5	15	24	67	45	1	76	21	11
	0	1	2	3	4	5	6	7	8	9

- No primeiro exemplo, a função deve retornar 5, enquanto no segundo exemplo, a função deve retornar  $-1$ .

# Busca Sequencial

- A busca sequencial é o algoritmo mais simples de busca:
  - Percorra a lista comparando a chave com os valores dos elementos em cada uma das posições.
  - Se a chave for igual a algum dos elementos, retorne a posição correspondente na lista.
  - Se a lista toda foi percorrida e a chave não for encontrada, retorne o valor  $-1$ .

```
1 def buscaSequencial(lista, chave):  
2     indice = 0  
3     for número in lista:  
4         if número == chave:  
5             return indice  
6         indice = indice + 1  
7     return -1
```

```
1 def buscaSequencial(lista, chave):  
2     n = len(lista)  
3     for índice in range(n):  
4         if lista[índice] == chave:  
5             return índice  
6  
7     return -1
```

- Podemos usar também a função `enumerate(lista)`, que retorna uma lista com tuplas da forma `(índice, elemento)`.

```
1 def buscaSequencial(lista, chave):  
2     for (índice, número) in enumerate(lista):  
3         if número == chave:  
4             return índice  
5  
6     return -1
```



```
1 def buscaSequencial(lista, chave):
2     ...
3
4     chave = 45
5     lista = [20, 5, 15, 24, 67, 45, 1, 76, 21, 11]
6
7     pos = buscaSequencial(lista, chave)
8
9     if pos != -1:
10        print("Posição da chave", chave, "na lista:", pos)
11    else:
12        print("A chave", chave, "não se encontra na lista")
13
14    # Posição da chave 45 na lista: 5
```

```
1 def buscaSequencial(lista, chave):
2     ...
3
4     chave = 100
5     lista = [20, 5, 15, 24, 67, 45, 1, 76, 21, 11]
6
7     pos = buscaSequencial(lista, chave)
8
9     if pos != -1:
10        print("Posição da chave", chave, "na lista:", pos)
11    else:
12        print("A chave", chave, "não se encontra na lista")
13
14    # A chave 100 não se encontra na lista
```

# Eficiência da Busca Sequencial

## Complexidade da Busca Sequencial

- A busca sequencial é a estratégia de busca mais simples que existe.
- Assume-se que os dados não estão ordenados.
- No pior caso, percorre-se a estrutura que armazena os dados na sua totalidade, desde sua primeira posição até sua última posição.
- Como o custo deste processo é linear respeito ao número de elementos do "container" é chamada também de "busca linear".

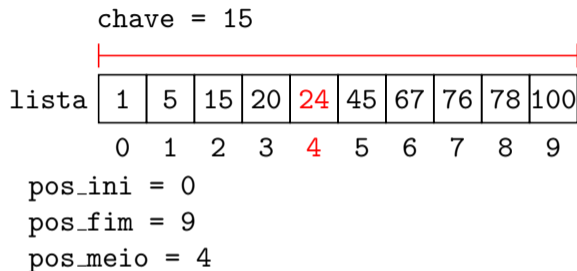
<https://ic.unicamp.br/~mc102/aulas/aula11.pdf>

- Na melhor das hipóteses, a chave de busca estará na posição 0. Portanto, teremos um único acesso em `lista[0]`.
- Na pior das hipóteses, a chave é o último elemento ou não pertence à lista e, portanto, acessamos todos os  $n$  elementos da lista.
- É possível mostrar que, se as chaves possuírem a mesma probabilidade de serem requisitadas, o número médio de acessos nas buscas cujas chaves encontram-se na lista será igual a:

$$\frac{n + 1}{2}$$

# Busca Binaria

- A busca binária é um algoritmo mais eficiente, entretanto, requer que a lista esteja ordenada pelos valores da chave de busca.
- A ideia do algoritmo é a seguinte (assuma que a lista está ordenada pelos valores da chave de busca):
  - Verifique se a chave de busca é igual ao valor da posição do meio da lista.
  - Caso seja igual, devolva esta posição.
  - Caso o valor desta posição seja maior que a chave, então repita o processo, mas considere uma lista reduzida, com os elementos do começo da lista até a posição anterior a do meio.
  - Caso o valor desta posição seja menor que chave, então repita o processo, mas considere uma lista reduzida, com os elementos da posição seguinte a do meio até o final da lista.



- Como `lista[pos_meio] > chave`, devemos continuar a busca na primeira metade da região e, para isso, atualizamos a variável `pos_fim`.



chave = 15

lista 

1	5	15	20	24	45	67	76	78	100
0	1	2	3	4	5	6	7	8	9

pos\_ini = 0  
pos\_fim = 3  
pos\_meio = 1

- Como `lista[pos_meio] < chave`, devemos continuar a busca na segunda metade da região e, para isso, atualizamos a variável `pos_ini`.

chave = 15

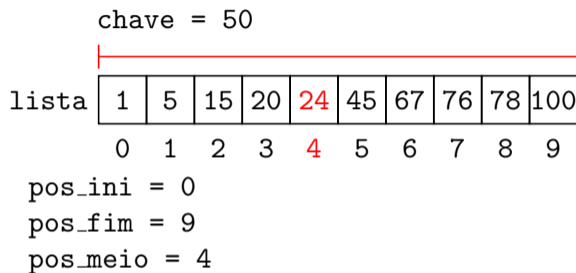
			┌──────────┐							
lista	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9

pos\_ini = 2

pos\_fim = 3

pos\_meio = 2

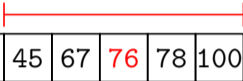
- Finalmente, encontramos a chave ( $lista[pos\_meio] = chave$ ) e, sendo assim, devolvemos a sua posição na lista ( $pos\_meio$ ).



- Como `lista[pos_meio] < chave`, devemos continuar a busca na segunda metade da região e, para isso, atualizamos a variável `pos_ini`.

chave = 50

lista	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9



pos\_ini = 5

pos\_fim = 9

pos\_meio = 7

- Como `lista[pos_meio] > chave`, devemos continuar a busca na primeira metade da região e, para isso, atualizamos a variável `pos_fim`.

chave = 50

lista	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9

pos\_ini = 5

pos\_fim = 6

pos\_meio = 5

- Como `lista[pos_meio] < chave`, devemos continuar a busca na segunda metade da região e, para isso, atualizamos a variável `pos_ini`.

chave = 50

lista	1	5	15	20	24	45	67	76	78	100
	0	1	2	3	4	5	6	7	8	9

pos\_ini = 6

pos\_fim = 6

pos\_meio = 6

- Como `lista[pos_meio] > chave`, devemos continuar a busca na primeira metade da região e, para isso, atualizamos a variável `pos_fim`.

chave = 50

lista

1	5	15	20	24	45	67	76	78	100
0	1	2	3	4	5	6	7	8	9

pos\_ini = 6

pos\_fim = 5

pos\_meio = 5

- Como  $\text{pos\_ini} > \text{pos\_fim}$ , determinamos que a chave não está na lista e retornamos o valor  $-1$ .

```
1 def buscaBinária(lista, chave):
2     pos_ini = 0
3     pos_fim = len(lista) - 1
4
5     while pos_ini <= pos_fim:
6         pos_meio = (pos_ini + pos_fim) // 2
7
8         if lista[pos_meio] == chave:
9             return pos_meio
10        if lista[pos_meio] > chave:
11            pos_fim = pos_meio - 1
12        if lista[pos_meio] < chave:
13            pos_ini = pos_meio + 1
14
15    return -1
```



```
1 def buscaBinária(lista, chave):
2     pos_ini = 0
3     pos_fim = len(lista) - 1
4
5     while pos_ini <= pos_fim:
6         pos_meio = (pos_ini + pos_fim) // 2
7
8         if lista[pos_meio] == chave:
9             return pos_meio
10        if lista[pos_meio] > chave:
11            pos_fim = pos_meio - 1
12        else:
13            pos_ini = pos_meio + 1
14
15    return -1
```

```
1 def buscaBinária(lista, chave):
2     ...
3
4     chave = 15
5     # Para usar a busca binária a lista deve estar ordenada
6     lista = [1, 5, 15, 20, 24, 45, 67, 76, 78, 100]
7
8     pos = buscaBinária(lista, chave)
9
10    if pos != -1:
11        print("Posição da chave", chave, "na lista:", pos)
12    else:
13        print("A chave", chave, "não se encontra na lista")
14
15    # Posição da chave 15 na lista: 2
```

```
1 def buscaBinária(lista, chave):
2     ...
3
4     chave = 50
5     # Para usar a busca binária a lista deve estar ordenada
6     lista = [1, 5, 15, 20, 24, 45, 67, 76, 78, 100]
7
8     pos = buscaBinária(lista, chave)
9
10    if pos != -1:
11        print("Posição da chave", chave, "na lista:", pos)
12    else:
13        print("A chave", chave, "não se encontra na lista")
14
15    # A chave 50 não se encontra na lista
```

# Eficiência da Busca Binária

<https://ic.unicamp.br/~mc102/aulas/aula11.pdf>

- Na melhor das hipóteses, a chave de busca estará na posição do meio da lista. Portanto, teremos um único acesso.
- Na pior das hipóteses, dividimos a lista até a que ela fique com um único elemento (último acesso realizado à lista).
- Note que, a cada acesso, o tamanho da lista é diminuído, pelo menos, pela metade.
- Quantas vezes um número pode ser dividido por dois antes dele se tornar igual a um?
- Esta é exatamente a definição de logaritmo na base 2.
- Ou seja, no pior caso o número de acesso é igual a  $\log_2 n$ .
- É possível mostrar que, se as chaves possuírem a mesma probabilidade de serem requisitadas, o número médio de acessos nas buscas cujas chaves encontram-se na lista será igual a:

$$(\log_2 n) - 1$$

# Exemplo comparativo

<https://ic.unicamp.br/~mc102/aulas/aula11.pdf>

- Para se ter uma ideia da diferença de eficiência dos dois algoritmos, considere uma lista com um milhão de itens ( $10^6$  itens).
- Com a busca sequencial, para buscar um elemento qualquer da lista necessitamos, em média, de:

$$(10^6 + 1)/2 \approx 500000 \text{ acessos.}$$

- Com a busca binária, para buscar um elemento qualquer da lista necessitamos, em média, de:

$$(\log_2 10^6) - 1 \approx 19 \text{ acessos.}$$

<https://ic.unicamp.br/~mc102/aulas/aula11.pdf>

- Uma ressalva importante deve ser feita: para utilizar a busca binária, a lista precisa estar ordenada.
- Se você tiver um cadastro onde vários itens são removidos e inseridos com frequência e a busca deve ser feita de forma intercalada com essas operações, então a busca binária pode não ser a melhor opção, já que você precisará manter a lista ordenada.
- Caso o número de buscas seja muito maior que as demais operações de atualização do cadastro, então a busca binária pode ser uma boa opção.



567

Linear Search

Binary Search

 Small Large

```
def linearSearch(listData, value)
```

```
    index = 0
```

```
    while (index < len(listData) and listData[index] < value):
```

```
        index++;
```

```
    if (index >= len(listData) or listData[index] != value):
```

```
        return -1
```

```
    return index
```

Searching For

567

Result

-1

Element Not found

index

24

21	40	53	103	134	148	154	161	180	235	239	272	312	321	351	442
----	----	----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

0

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

477	480	495	509	531	533	556	564	635	738	784	894	901	926	969	985
-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----	-----

16

17

18

19

20

21

22

23

24

25

26

27

28

29

30

31

# Exercícios

<https://ic.unicamp.br/~mc102/aulas/aula11.pdf>

1. Refaça as funções de busca sequencial e busca binária assumindo que a lista possui chaves que podem ocorrer múltiplas vezes na lista. Neste caso, você deve retornar uma lista com todas as posições onde a chave foi encontrada. Se a chave não for encontrada na lista, retornar uma lista vazia.

<https://ic.unicamp.br/~mc102/aulas/aula11.pdf>

2. Mostre como implementar uma variação da busca binária que retorne um inteiro  $k$  entre  $0$  e  $n$ , tal que, ou  $\text{lista}[k] = \text{chave}$ , ou a chave não se encontra na lista, mas poderia ser inserida entre as posições  $(k-1)$  e  $k$  de forma a manter a lista ordenada. Note que, se  $k = 0$ , então a chave deveria ser inserida antes da primeira posição da lista, assim como, se  $k = n$ , a chave deveria ser inserida após a última posição da lista.
3. Use a função desenvolvida acima para, dada uma lista ordenada de  $n$  números inteiros e distintos e dois outros inteiros  $X$  e  $Y$ , retornar o número de chaves da lista que são maiores ou iguais a  $X$  e menores ou iguais a  $Y$ .

# Perguntas ....

# Referências

- Zanoni Dias, MC102, Algoritmos e Programação de Computadores, IC/UNICAMP, 2021. <https://ic.unicamp.br/~mc102/>
  - Aula Introdutória [ [slides](#) ] [ [vídeo](#) ]
  - Primeira Aula de Laboratório [ [slides](#) ] [ [vídeo](#) ]
  - Python Básico: Tipos, Variáveis, Operadores, Entrada e Saída [ [slides](#) ] [ [vídeo](#) ]
  - Comandos Condicionais [ [slides](#) ] [ [vídeo](#) ]
  - Comandos de Repetição [ [slides](#) ] [ [vídeo](#) ]
  - Listas e Tuplas [ [slides](#) ] [ [vídeo](#) ]
  - Strings [ [slides](#) ] [ [vídeo](#) ]
  - Dicionários [ [slides](#) ] [ [vídeo](#) ]
  - Funções [ [slides](#) ] [ [vídeo](#) ]
  - Objetos Multidimensionais [ [slides](#) ] [ [vídeo](#) ]
  - Algoritmos de Ordenação [ [slides](#) ] [ [vídeo](#) ]
  - Algoritmos de Busca [ [slides](#) ] [ [vídeo](#) ]
  - Recursão [ [slides](#) ] [ [vídeo](#) ]
  - Algoritmos de Ordenação Recursivos [ [slides](#) ] [ [vídeo](#) ]
  - Arquivos [ [slides](#) ] [ [vídeo](#) ]
  - Expressões Regulares [ [slides](#) ] [ [vídeo](#) ]
  - Execução de Testes no Google Cloud Shell [ [slides](#) ] [ [vídeo](#) ]
  - Numpy [ [slides](#) ] [ [vídeo](#) ]
  - Pandas [ [slides](#) ] [ [vídeo](#) ]
- Panda - Cursos de Computação em Python (IME -USP) <https://panda.ime.usp.br/>
  - Como Pensar Como um Cientista da Computação <https://panda.ime.usp.br/pensepy/static/pensepy/>
  - Aulas de Introdução à Computação em Python <https://panda.ime.usp.br/aulasPython/static/aulasPython/>
- Fabio Kon, Introdução à Ciência da Computação com Python <http://bit.ly/FabioKon/>
- Socratica, Python Programming Tutorials <http://bit.ly/SocraticaPython/>
- Google - online editor for cloud-native applications (Python programming) <https://shell.cloud.google.com/>
- w3schools - Python Tutorial <https://www.w3schools.com/python/>
- Outros, citados nos Slides.