# Buscas



#### **Buscas em vetor**

- Considerar função em que:
  - Os parâmetros da função sejam:
    - Um vetor de dados denominado info.
    - Um valor de busca denominado valorBuscar.
  - O retorno da função seja:
    - i, se o elemento valorBuscar encontrar-se em info[i]
    - -1, se o elemento valorBuscar não constar em info.



#### **Busca linear**

 Percorre o vetor, partindo da primeira posição, até encontrar um elemento que armazena o valor de busca ou até atingir o final do vetor.

```
Algoritmo: buscaLinear(int[] info, int valorBuscar)

int n ← size(info);

para i ←0 até n-1 faça

se info[i] = valorBuscar então // encontrado

retornar i;

fim-se;

fim-para;

retornar -1; // não encontrado
```



#### Análise busca linear

#### Pior caso:

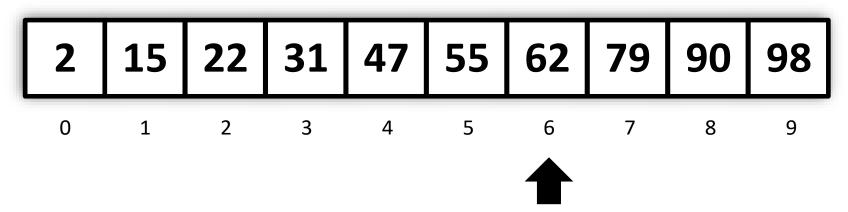
- O valor de busca não existe ou está localizado no final do vetor.
- Se a quantidade de elementos é n, a quantidade necessária de comparações é n também
  - Complexidade: O(n)

#### Caso médio:

- Necessária: n/2 comparações
- Desempenho computacional continua variando linearmente em relação ao problema.
  - Complexidade: Ω(n)



#### Busca linear em vetor ordenado



 Para buscar o valor 57, por exemplo, não é preciso percorrer toda a estrutura de dados A partir desta posição não precisa mais procurar

Isto é, quando o *valor de busca* for inferior ao valor do vetor, desiste de procurar



#### Busca linear em vetor ordenado

```
Algoritmo: buscaLinearVetorOrdenado(int[] info, int valorBuscar)
int n \leftarrow size(info);
para i ←0 até n-1 faça
  se info[i] = valorBuscar então
   retornar i;
  senão
   se valorBuscar < info[i] então
     break;
   fim-se;
 fim-se;
fim-para;
retornar -1;
```



#### Análise busca linear em vetor ordenado

- Caso o elemento procurado não pertença ao vetor, a busca linear com vetor ordenado apresenta um desempenho ligeiramente superior à busca linear
- O algoritmo continua sendo linear
  - Complexidade O(n)



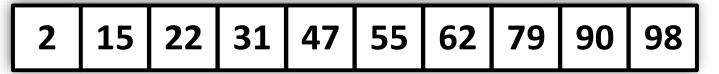
#### Busca binária

- Utilizada quando a estrutura de dados está ordenada
- Compara o valor de busca com o elemento do meio do vetor e:
  - Se o valor de busca for igual ao do vetor:
    - elemento localizado. Finaliza o algoritmo
  - Se o valor de busca for menor ao do vetor:
    - procura novamente na primeira metade do vetor
  - Se o valor de busca for maior ao do vetor:
    - procura novamente na segunda metade do vetor
- O procedimento é repetido, subdividindo o vetor até encontrar o elemento ou o sub-vetor atingir tamanho 0.



### Busca binária

• Procurar o número 55 no vetor:



meio 
$$\leftarrow [(inicio + fim)/2];$$

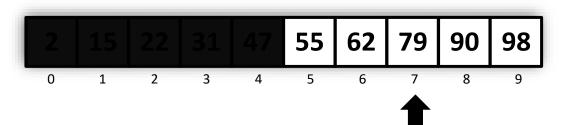
meio 
$$\leftarrow \lfloor (0+9)/2 \rfloor$$
;

meio  $\leftarrow 4$ ;

meio 
$$\leftarrow [(inicio + fim)/2];$$

meio 
$$\leftarrow [(5+9)/2];$$

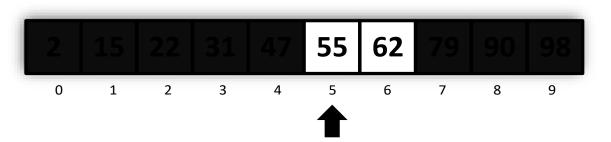
meio  $\leftarrow$  7;





## Busca binária

meio 
$$\leftarrow \lfloor (\text{inicio} + \text{fim})/2 \rfloor;$$
  
meio  $\leftarrow \lfloor (5+6)/2 \rfloor;$   
meio  $\leftarrow 5;$ 



Elemento encontrado



# Algoritmo iterativo para busca binária

```
Algoritmo: buscaBinaria(int[] info, int valorBuscar)
n \leftarrow size(info);
inicio \leftarrow 0;
fim \leftarrow n-1;
enquanto inicio ≤ fim faça
  meio \leftarrow |(inicio + fim)/2|;
  se valorBuscar < info[meio] então
    fim ← meio-1; // redefine posição final
  senão
    se valorBuscar > info[meio] então
      inicio ← meio+1; // redefine posição inicial
    senão
      retornar meio; // achou
    fim-se;
  fim-se;
fim-enquanto;
retornar -1;
```



## Análise do algoritmo de busca binária

- Pior caso:
  - O elemento n\u00e3o existe no vetor
  - A cada iteração:
    - duas comparações são realizadas
    - O escopo de busca é dividido pela metade

Repetição	Tamanho do problema
1	n
2	n/2
3	n/4
Log n	1

