

# Busca e Ordenação

---

Fabio Lubacheski  
fabio.lubacheski@mackenzie.br

# Busca de um elemento

Considere o problema em se determinar se um elemento está presente em uma lista de elementos, ou seja:

**Dado um inteiro  $x$  e um vetor  $v[0 \dots n-1]$  de inteiros, o problema da busca consiste em encontrar  $x$  em  $v$ , ou seja, encontrar um índice  $k$  tal que  $v[k] == x$ .**

# Busca de um elemento

Poderíamos realizar a busca comparando o valor do elemento a ser encontrado com os elementos do vetor, um a um, da esquerda para a direita, sequencialmente (=linearmente).

Esse tipo de busca é denominada busca linear, e para implementar a busca linear é preciso tomar algumas decisões importante antes de começar, vamos considerar as configurações de entrada abaixo:

- O que fazer quando  $x$  está no vetor ?
- O que fazer se  $x$  não está no vetor?
- O que fazer se tivermos elementos repetidos, ou seja, aparece duas vezes no vetor ?

# Busca Linear

A função implementada abaixo devolve a posição em que o elemento  $x$  se encontra no vetor, caso ele esteja lá, ou um valor inválido ( $-1$ ) que represente o insucesso da busca caso o elemento não esteja no vetor.

```
def busca( v, x ):
    k=0
    while k < len(v):
        if v[k] == x:
            return k
        k+=1

    return -1;
```

# Exercícios

Considere o método que faz a Busca Linear o vetor, ele funciona ? Critique o método:

```
def buscaCriticar(v, x):  
    m = 0  
    while v[m] < x and m < len(v):  
        m+=1;  
  
    if v[m] == x:  
        return m;  
    else:  
        return -1;
```

# Busca linear

Existe alguma outra forma de fazer a busca de um elemento dentro do vetor ?

Será que é melhor ? Como comparar as duas formas ?

Vejam o vídeo abaixo para se inspirarem da computação desplugada.

<https://www.youtube.com/watch?v=iDVH3oCTc2c>

# Busca Binária

Quando o vetor está **ordenado** podemos utilizar uma técnica de programação chamada de divisão-e-conquista para realizarmos uma busca binária:

- Dividimos o vetor ao meio
- Se o elemento procurado for o elemento do meio, pare.
- Senão, se o elemento procurado for menor que o elemento do meio, repetimos o processo para os elementos à esquerda do elemento do meio; caso contrário, repetimos o processo nos elementos à direita do elemento do meio.

# Busca Binária

A função recebe um número  $x$  e um vetor em ordem crescente  $v[0 \dots n-1]$  com  $n$  elementos. A função devolve um índice  $m$  tal que  $v[m] == x$ , ou seja achou  $x$  em  $v[]$ , ou devolve  $-1$  se tal  $m$  não existe.

Vocês conseguem fazer o método?



# Busca Binária

A função recebe um número  $x$  e um vetor em ordem crescente  $v[0..n-1]$  com  $n$  elementos. A função devolve um índice  $m$  tal que  $v[m] == x$  ou devolve  $-1$  se tal  $m$  não existe.

```
def BuscaBinaria( v, x ):
    i = 0 #inicio do vetor
    f = len(v)-1 #fim do vetor
    while i <= f:
        m = (i + f)//2 #divisao inteira
        if v[m] == x:
            return m

        if v[m] < x:
            i = m + 1
        else:
            f = m - 1
    return -1
```

# Curiosidade

A Busca Binária implementa a técnica de programação conhecida como “divisão e conquista”, que pode ser usado para resolver eficientemente muitos problemas computacionais.



## Novo problema ...

Como obter o vetor ordenado em ordem crescente para realizarmos a Busca Binária ?

Ordenar um vetor significa:

*Permutar* (ou seja, *rearranjar*) os elementos de um vetor  $v[0..n-1]$  de tal modo que eles fiquem em ordem crescente, ou seja, de tal forma que tenhamos  $v[0] \leq v[1] \leq \dots \leq v[n-1]$ .

# Algoritmos ordenação - links

- Simulação de algoritmos de ordenação:  
<http://nicholasandre.com.br/sorting/>
- Comparação de algoritmos de ordenação  
<https://www.cs.usfca.edu/~galles/visualization/ComparisonSort.html>
- Dança húngara com método bolha:  
<https://www.youtube.com/watch?v=lyZQPjUT5B4>
- Dança romena com método inserção:  
<https://www.youtube.com/watch?v=ROaIU379I3U>
- Dança alemã com MergeSort:  
[https://www.youtube.com/watch?v=XaqR3G\\_NVoo](https://www.youtube.com/watch?v=XaqR3G_NVoo)

# Ordenação pelo algoritmo Bolha

O algoritmo Bolha, em cada iteração, “borbulha” o maior elemento para fim do vetor, percorrendo o vetor da esquerda para a direita, comparando pares de elementos consecutivos, trocando de lugar os que estão fora de ordem, ou seja,  $v[i] > v[i+1]$ .

Implementação do algoritmo Bolha: `bolha.py`.

Tente identificar no algoritmo uma configuração de entrada em que acontece a maior quantidade de trocas em função do tamanho do vetor.

# Ordenação por Inserção

Em termos gerais, o algoritmo percorre um vetor de elementos da esquerda para a direita e à medida que avança vai deixando os elementos mais à esquerda ordenados. O algoritmo de inserção funciona da mesma maneira com que muitas pessoas ordenam cartas em um jogo de baralho.

Implementação do algoritmo de inserção: `insercao.py`

# Exercícios

- 1) Dado um vetor com números pares e ímpares, escreva uma função para colocar todos os números pares à frente no vetor e os ímpares ao final. Você não pode usar outro vetor como área auxiliar. Resolva esse exercício usando somente uma estrutura de repetição e sem um vetor um auxiliar.
- 2) Escreva uma função que selecione o primeiro elemento de um vetor, e rearranje o vetor de forma que todos os elementos menor ou igual ao primeiro elemento fiquem a sua esquerda e os maiores a sua direita. Se o vetor informado for  
 $\{5, 6, 2, 7, 9, 1, 8, 3, 7\}$  após ser rearranjado teríamos  
 $\{1, 3, 2, 5, 9, 7, 8, 6, 7\}$ . Note que o vetor ainda não está ordenado.  
Resolva esse exercício usando somente uma estrutura de repetição e sem um vetor um auxiliar.

# Exercícios

- 3) Escreva uma função que receba dois vetores ( $A[]$  e  $B[]$ ) já ordenados em ordem crescente, a sua função aloca um vetor  $C[]$  exatamente com soma dos tamanhos de  $A$  e  $B$ , e intercala os elementos de  $A[]$  e  $B[]$  em  $C[]$ , de forma que o vetor  $C[]$  fique ordenado em ordem crescente. Escreva função o mais eficiente possível. Não é para juntar os vetores e ordenar o vetor  $C[]$  usando o método Bolha nem o Inserção. Exemplo:

$A[] = \{ 1, 3, 6, 7 \}$  e  $B[] = \{ 2, 4, 5 \}$ , o novo vetor  $C$  construído a partir de  $A[]$  e  $B[]$  é  $C[] = \{ 1, 2, 3, 4, 5, 6, 7 \}$

Resolva esse exercício sem usar laços aninhados e também não vale concatenar os vetores e depois mandar ordenar.



# Exercícios

- 4) Escreva um algoritmo que calcula a soma dos prefixos de um vetor. A soma de prefixos de um vetor  $V$  em  $S$  pode ser definida por:

$$S[0] = V[0]$$

$$S[i] = V[i] + V[i - 1] + V[i - 2] + \dots + V[0]$$

- 5) Pesquise como funciona o método de ordenação por seleção, ele é melhor que o bolha ou o inserção ? Implemente o método em Python

- 6) Escreva um função que receba um vetor já ordenado com sua última posição disponível e um valor a ser inserido no vetor, após a inserção o vetor deve continuar ordenado. Considere o exemplo abaixo, o vetor tem 8 elementos:

0	1	2	3	4	5	6	7
3	5	9	11	15	17	19	

após inserir o **valor = 6** o vetor ficaria com a seguinte configuração.

0	1	2	3	4	5	6	7
3	5	6	9	11	15	17	19

# Exercícios

- 7) Dado um vetor de  $n$  números inteiros, faça uma função para determinar o comprimento do maior segmento crescente.

Exemplos: na sequência { 5, 10, 3, 2, 4, 7, 9, 8, 5 } o comprimento do segmento crescente máximo é 4 {2, 4, 7, 9}.

Na sequência { 10, 8, 7, 5, 2 } o comprimento de um segmento crescente máximo é 1.

- 8) Dada uma sequência de números inteiros com  $n$  elementos, determinar quantas subsequências de números iguais consecutivos compõem essa sequência. Exemplos:

A sequência 5, 2, 2, 3, 4, 4, 4, 4, 4, 1, 1, é composta por 5 subsequências: {5}, {2,2}, {3}, {4, 4, 4, 4, 4}, {1,1};

A sequência: 3, 3, -1, -1, -1, 12, 12, 12, 3, 3, é composta por 4 subsequências: {3,3}, {-1, -1, -1}, {12, 12, 12}, {3,3};

Escreva uma função que determina a quantidade de subsequências em sequência dada.

Fim