


Pensamento Computacional

The background is a collage of puzzle pieces in shades of blue and grey. Several puzzle pieces feature question marks. In the center, a puzzle piece shows a glowing lightbulb icon, symbolizing an idea or problem-solving. On the right side, there are several white circles of varying sizes, some overlapping the puzzle pieces.

2022/23

Aula 9

Objetivos

- Pesquisa e Ordenação
 - Pesquisa sequencial
 - Pesquisa binária
- Ordenação
 - Ordenação por seleção

Problema de Pesquisa

- Suponhamos que temos uma sequência de n valores (e.g., inteiros)
 $v[0], v[1], \dots, v[n-1]$
- Queremos saber se um valor κ ocorre na sequência, isto é:
encontrar j tal que $0 \leq j \leq n$ e $v[j] = \kappa$

Pesquisa sequencial

1. Para j de 0 até $n-1$:
 - Se $v[j] = \kappa$ então terminamos com resposta j ;
 - Caso contrário, continuamos a pesquisa.
2. Se chegarmos ao fim do ciclo, então κ não ocorre na sequência.

Este algoritmo encontra o menor índice tal que $v[j] = \kappa$ (se existir).

Pesquisa sequencial em Python

- Em Python, pesquisar um elemento x numa lista L (ou em qualquer outra sequência) é uma operação comum em vários problemas.
 - Por vezes, precisamos apenas de verificar **se** o elemento se encontra na sequência.

Em Python fazemos isto com: `x in L`

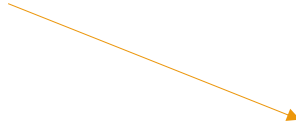
- Noutras situações, precisamos de saber onde está.

Em Python, podemos fazer isto com: `L.index(x)`.

Pesquisa sequencial em Python

- A **pesquisa sequencial** varre a sequência do início ao fim (ou do fim até ao início).

```
def seqSearch(lst, x):  
    """Return k such that x == lst[k], or None if no such k."""  
    for i in range(len(lst)):  
        if x == lst[i]:  
            return i  
    return None
```



É isto que o método `index` do operador `in` faz.

Pesquisa sequencial em Python

- Existem duas formas de terminar o ciclo:
 1. Se encontrarmos j tal que $lst[j] == \kappa$
 2. Ou se esgotarmos todos os índices válidos (e nesse caso, $j == n$ ou $len(lst)$)
- No pior caso, temos de percorrer todos os elementos da sequência, n , em que n é o comprimento da sequência.

- Podemos fazer melhor?

Pesquisa Binária

Queremos procurar x numa sequência de n valores inteiros

$v[0], v[1], v[2], \dots, v[n-1]$

- Pré-condição: os valores estão por ordem crescente:

Cada valor é menor ou igual ao seguinte: $v[0] \leq v[1] \leq v[2] \leq \dots \leq v[n-1]$

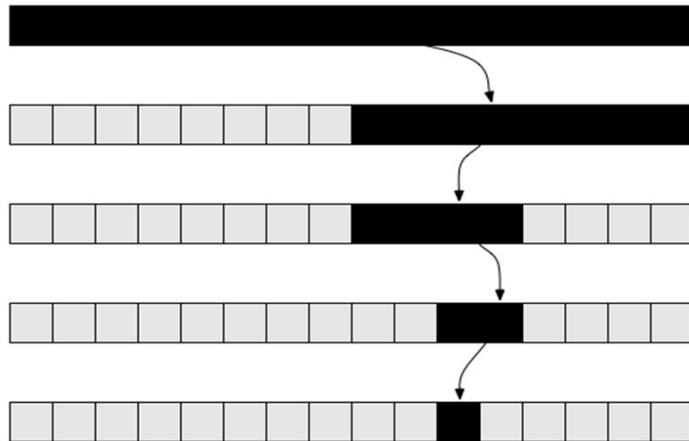
Pesquisa Binária: ideia

Consideramos o índice do meio $k = (n-1)//2$:

- Se $x = v[k]$ então encontramos o valor;
- Se $x < v[k]$ então continuamos a procurar na sub-sequência $v[0], \dots, v[k-1]$
- Se $x > v[k]$ então continuamos a procurar na sub-sequência $v[k+1], \dots, v[n-1]$

Repetimos até encontrar o valor ou a sequência ficar vazia.

Pesquisa Binária: ideia



- Se $N < 2^k \Rightarrow k$ comparações.

n	$\log_2 n$
8	3
16	4
32	5
64	6
128	7
256	8
512	9
1024	10

Pesquisa Binária: algoritmo

```
def binSearchExact(lst, x):  
    """Find k such that x == lst[k]. (Or None if no such k.)"""  
    first = 0          # first index that could be solution  
    last = len(lst)   # first index that cannot be solution  
    while first < last:  
        mid = (first+last)//2  
        if x < lst[mid]:  
            last = mid  
        elif x > lst[mid]:  
            first = mid+1  
        else:  
            return mid  
    return None
```

Pesquisa Binária: algoritmo

Alternativa

```
def binSearch(lst, x):  
    """Find k such that: lst[k-1] < x <= lst[k] (not quite!)."""  
    first = 0          # first index that can be result  
    last = len(lst)   # last index that can be result  
    while first < last:  
        mid = (first+last)//2  
        if x <= lst[mid]:    # (just 1 comparison inside loop!)  
            last = mid  
        else:  
            first = mid+1  
    return first
```

Funções bisect

- O módulo [bisect](#) inclui funções que fazem pesquisa binária em listas ordenadas.

```
import bisect

lst = [10, 20, 20, 30, 40, 50, 60, 70, 80, 90]

# Using bisect to search values
I40 = bisect.bisect_left(lst, 40)
print(lst[I40] == 40)

I65 = bisect.bisect_left(lst, 65)
print(lst[I65] == 65)

I05 = bisect.bisect_left(lst, 5)
I91 = bisect.bisect_left(lst, 91)

# Difference between _left and _right
L20 = bisect.bisect_left(lst, 20)
R20 = bisect.bisect_right(lst, 20)
```

Problema de Ordenação

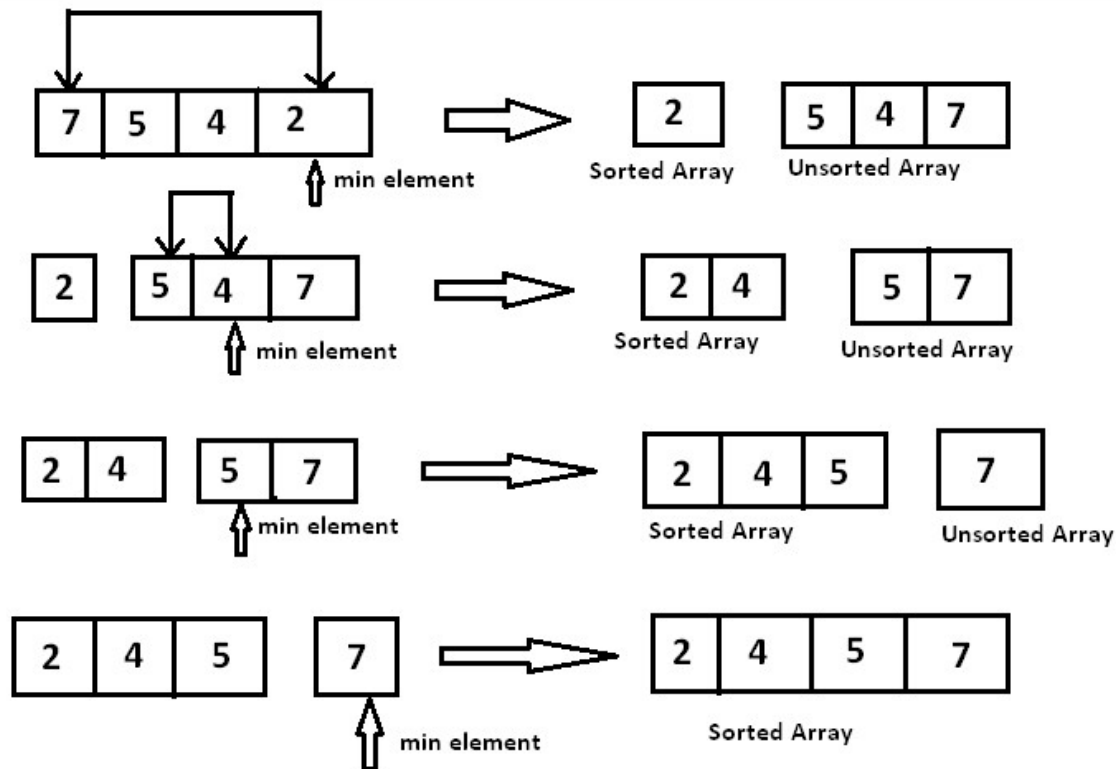
- Para utilizar a pesquisa binária, assumimos que a sequência está *ordenada*.
- Vamos agora estudar algoritmos para ordenar uma sequência arbitrária.

Objetivo: reorganizar os valores de forma a que fiquem por *ordem ascendente* ou *descendente*

Ordenação por seleção

1. Procuramos o menor valor da sequência e trocamos o seu lugar com o 1º elemento;
2. Procuramos o menor dos valores restantes e trocamos o seu lugar com o 2º elemento;
3. Continuamos este processo até colocarmos todos os elementos na posição correta.

Ordenação por seleção: exemplo



Ordenação por seleção: o algoritmo

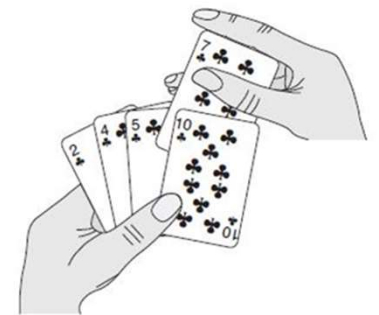
Repetimos para i de 0 até $n - 1$:

1. Inicializamos $i_{min} \leftarrow i$
2. Para j de $i + 1$ até $n - 1$:
 - se $v[j] < v[i_{min}]$ então $i_{min} \leftarrow j$
3. Se $i \neq i_{min}$ trocamos $v[i]$ com $v[i_{min}]$

Ordenação por inserção

1. Vamos considerar **segmentos** que começam no início da sequência;
2. O segmento com apenas um valor está trivialmente ordenado;
3. Em cada passo vamos **inserir um valor no segmento** mantendo a ordem.

Analogia: a forma como um jogador que mantém uma mão de cartas ordenada



Ordenação por inserção

6 5 3 1 8 7 2 4

(Autor: Swfung8 *via* Wikimedia Commons.)

Ordenação por seleção: o algoritmo

Basta inserir $v[1]$, depois $v[2]$, etc.

repetir para i de 1 até $n - 1$:

$x \leftarrow v[i]$

$j \leftarrow i - 1$

enquanto $j \geq 0 \wedge v[j] > x$:

$v[j + 1] \leftarrow v[j]$

$j \leftarrow j - 1$

$v[j + 1] \leftarrow x$

Problema de Ordenação

- Em Python, use a função `sorted` ou o método das listas `sort`
 - `L.sort()` # Modifica a lista L
 - `L2 = sorted(L)` # Cria L2. L não é alterada!
- `sorted` devolve uma lista mas aceita como *input*, qualquer coleção.
 - `sorted('banana')` #-> ['a', 'a', 'a', 'b', 'n', 'n']
 - `N = (9, 7, 2, 8, 5, 3)`
 - `print(sorted(N))` #-> [2, 3, 5, 7, 8, 9]
 - `S = {"maria", "carla", "anabela", "antonio", "nuno"}`
 - `print(sorted(S))`
 - #-> ['anabela', 'antonio', 'carla', 'maria', 'nuno']

Critérios de ordenação

- Estas funções podem ordenar segundo critérios diferentes:

```
L = ["Mario", "Carla", "anabela", "Maria", "nuno"]
```

```
print(sorted(L)) # lexicographic sort  
#-> ['Carla', 'Maria', 'Mario', 'anabela', 'nuno']
```

```
print(sorted(L, key=len)) # sort by length  
#-> ['nuno', 'Mario', 'Carla', 'Maria', 'anabela']
```

```
print(sorted(L, key=str.lower)) # case-insensitive  
#-> ['anabela', 'Carla', 'Maria', 'Mario', 'nuno']
```

Critérios de ordenação

- O argumento opcional `key` recebe uma função que permite determinar o critério de ordenação.
- A função `key` é aplicada a cada um dos elementos e os resultados são comparados para estabelecer a ordem.
- Para inverter a ordem, usar o argumento `reverse=True`.

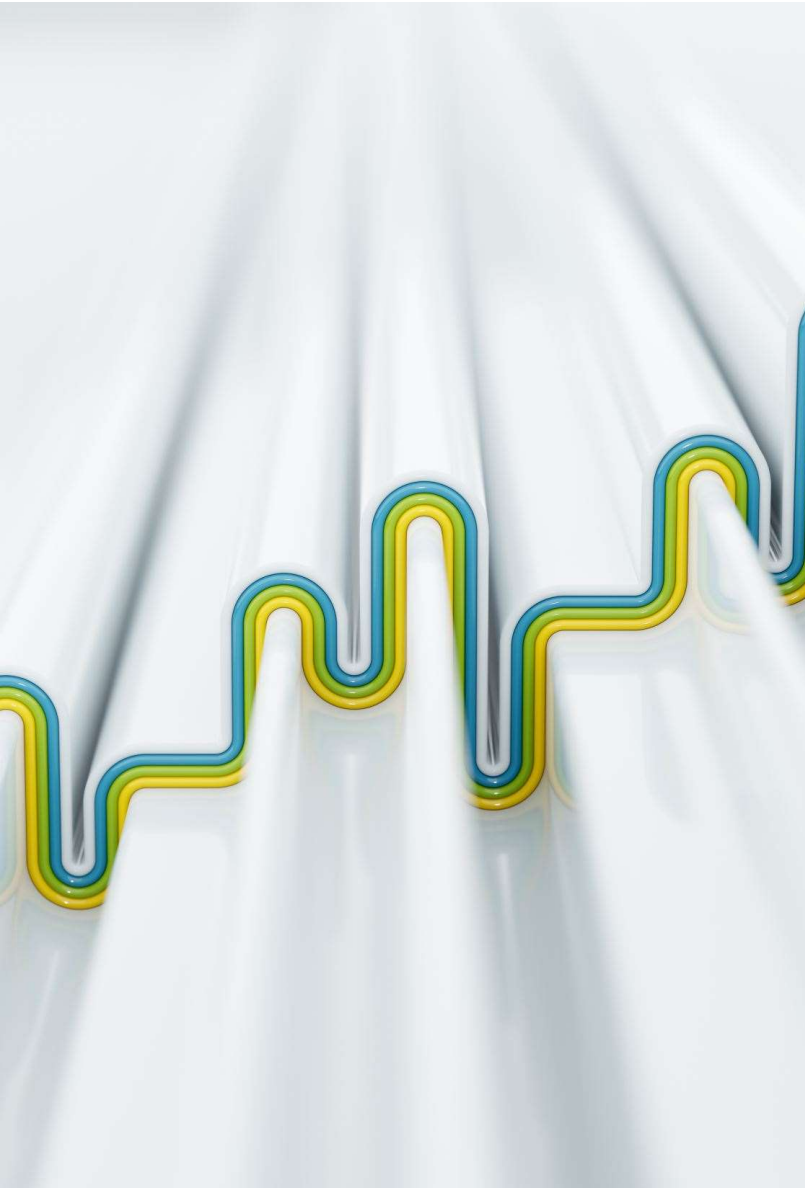
Exercício

Escreva uma função que calcule a mediana de uma lista de valores.

- A mediana é um valor que é maior que metade dos valores da lista e menor que a outra metade.
- Se a lista tiver um número ímpar de valores, a mediana é o valor a meio da lista ordenada. Se a lista tiver um número par de valores, a mediana é a média dos dois valores a meio da lista ordenada.

Objetivos

- Relembre:
 - Pensamento computacional e desenho de algoritmos.



Pensamento Computacional

2022/23

TPC: exercícios propostos aula 9
