# Processamento da Informação
# – Teoria –

## Conjuntos e Busca de dados

Semana 10

Prof. Jesús P. Mena-Chalco

29/06/2013

# Conjuntos

Um conjunto é uma **coleção de objetos** de qualquer tipo (pessoas, plantas, animais, fenômenos).

Os elementos que constituem um conjunto são chamados de **elementos do conjunto.**

O conjunto **A** que possui os elementos **a**, **b**, e **c** é representado por:

$$A = \{a, b, c\}$$

# Representação de conjuntos

Duas formas de representar conjuntos:
- explícita
- implícita

**Exemplo:**
Conjunto dos números pares maiores do que 4 e menores do que 15:

# Representação de conjuntos

Duas formas de representar conjuntos:
- explícita
- implícita

**Exemplo:**
Conjunto dos números pares maiores do que 4 e menores do que 15:

$$A = \{6, 8, 10, 12, 14\}$$

# Representação de conjuntos

Duas formas de representar conjuntos:
- explícita
- implícita

**Exemplo:**
Conjunto dos números pares maiores do que 4 e menores do que 15:

$$A = \{6, 8, 10, 12, 14\}$$

$$A = \{x \mid 4<x<15 \text{ e } x \text{ é par}\}$$

# Equivalência

Dois conjuntos A e B são iguais quando A e B têm os mesmos elementos.

Equivale a dizer que $A \subset B$ e $B \subset A$

Em conjuntos a ordem dos elementos é totalmente irrelevante. A repetição, embora não constitua um erro, é totalmente desnecessária.

$A = \{2, 2, 1, 3\}$ e $B = \{2, 1, 3\}$ são iguais.

$A = \{5, 6, 7\}$ e $B = \{6, 7, 5\}$ são iguais.

# Exercício: Equivalência

Crie uma função que permita verificar se duas listas (conjuntos) são equivalentes.
Apenas use listas.

**Cabeçalho:** **def** conjuntos_iguais**(A, B):**

**Exemplo:**
[2,2,1,3] e [2,1,3] são iguais.
[5,6,7] e [6,7,5] são iguais.

[1,2] e [1,2,4] não são iguais.

# Exercício: Equivalência

```
def conjuntos_iguais(A, B):
    for a in A:
        if not a in B:
            return False
    for b in B:
        if not b in A:
            return False
    return True
```

# União de conjuntos

Dados dois conjuntos A e B, a união de A e B é o conjunto dos elementos que pertencem a A ou a B.

$$A \cup B = \{x | x \in A \text{ ou } x \in B\}$$

**Exemplo:**

A={1,1,2,3} e B={2,4,5} → AUB = {1,2,3,4,5}

A={1,2} e B={6,7} → AUB = {1,2,6,7}

# Exercício: União de conjuntos

Crie uma função que permita **unir** dois conjuntos A e B. A função deve retornar $A \cup B$ .

**Cabeçalho:** **def** uniao_conjuntos**(A, B):**

**Exemplo:**

A=[1,1,2,3] e B=[2,4,5] → AUB = [1,2,3,4,5]

A=[1,2] e B=[6,7] → AUB = [1,2,6,7]

# Exercício: União de conjuntos

```
def uniao_conjuntos(A, B):
    C = [ ]
    for a in A:
        if not a in C:
            C.append(a)
    for b in B:
        if not b in C:
            C.append(b)
    return C
```

# Interseção de conjuntos

A interseção $A \cap B$ de dois conjuntos A e B é o conjunto dos elementos que pertencem ao conjunto A e ao conjunto B.

$$A \cap B = \{x | x \in A \text{ e } x \in B\}$$

**Exemplo:**

A={1,1,2,3} e B={2,4,5} → A ∩ B = {2}

A={1,2} e B={6,7} → A ∩ B = { }

# Interseção de conjuntos

Crie uma função que calcule a interseção de dois conjuntos A e B dados como entrada.

**Cabeçalho:** **def** intersecao_conjuntos**(A, B):**

**Exemplo:**

A=[1,1,2,3] e B=[2,4,5] → A ∩B = [2]

A=[1,2] e B=[6,7] → A ∩B = [ ]

# Interseção de conjuntos

```
def intersecao_conjuntos(A, B):
    C = []
    for a in A:
        if a in B and not a in C:
            C.append(a)
    return C
```

# Busca sequencial

Podemos procurar ou buscar um elemento **x** em uma lista **L** inspecionando em sequencia as posições de **L** a partir da primeira posição.

```
def busca_sequencial(x, L):
    for i in range(0,len(L)):
        if x==L[i]:
            return i
    return -1
```

# Busca sequencial

Podemos procurar ou buscar um elemento **x** em uma lista **L** inspecionando em sequencia as posições de **L** a partir da primeira posição.

```
def busca_sequencial(x, L):
    for i in range(0,len(L)):
        if x==L[i]:
            return True
    return False
```

# Busca sequencial

Podemos procurar ou buscar um elemento **x** em uma lista **L** inspecionando em sequencia as posições de **L** a partir da primeira posição.

```
def busca_sequencial(x, L):
    for i in range(0,len(L)):
        if x==L[i]:
            return True
    return False
```

Crie uma versão recursiva

# Busca sequencial – recursivo

```
def busca_rec(x, L):
    if len(L)==0:
        return False
    else:
        if x==L[0]:
            return True
        else:
            return busca_rec(x,L[1:])
```

# Busca binária

Quando a lista L estiver na ordem crescente podemos identificar a existência de um elemento de forma mais rápida.

Procurar 55 na lista ordenada L

| 0 | 5 | 13 | 19 | 22 | 41 | 55 | 68 | 72 | 81 | 98 |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |

# Busca binária

Quando a lista L estiver na ordem crescente podemos identificar a existência de um elemento de forma mais rápida.

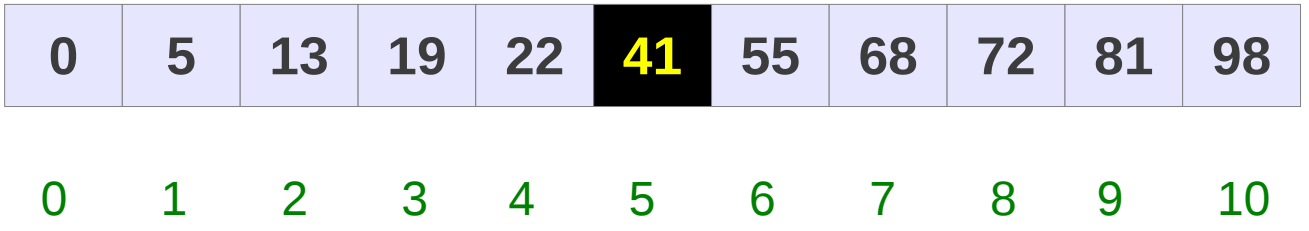
Procurar 55 na lista ordenada L

# Busca binária

Quando a lista L estiver na ordem crescente podemos identificar a existência de um elemento de forma mais rápida.

Procurar 55 na lista ordenada L

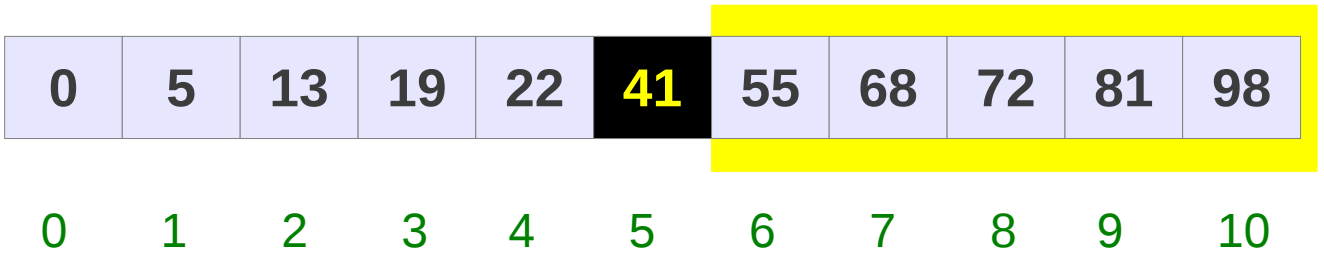
L[5]==55?



# Busca binária

Quando a lista L estiver na ordem crescente podemos identificar a existência de um elemento de forma mais rápida.

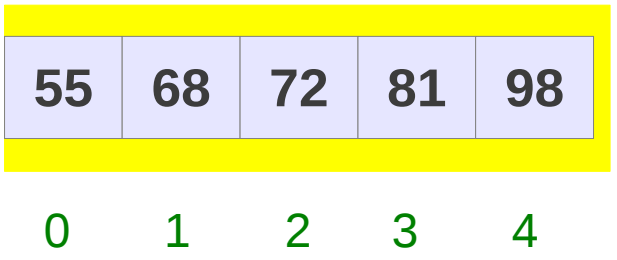
Procurar 55 na lista ordenada L



# Busca binária

Quando a lista L estiver na ordem crescente podemos identificar a existência de um elemento de forma mais rápida.

Procurar 55 na lista ordenada L



# Busca binária

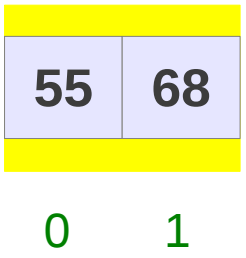
Quando a lista L estiver na ordem crescente podemos identificar a existência de um elemento de forma mais rápida.

Procurar 55 na lista ordenada L

# Busca binária

Quando a lista L estiver na ordem crescente podemos identificar a existência de um elemento de forma mais rápida.

Procurar 55 na lista ordenada L

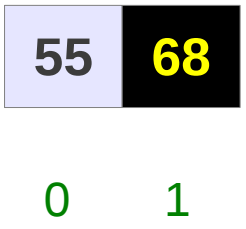


# Busca binária

Quando a lista L estiver na ordem crescente podemos identificar a existência de um elemento de forma mais rápida.

Procurar 55 na lista ordenada L

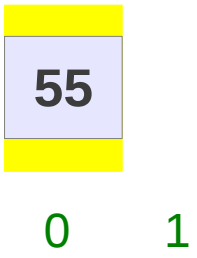
L[1]==55?



# Busca binária

Quando a lista L estiver na ordem crescente podemos identificar a existência de um elemento de forma mais rápida.

Procurar 55 na lista ordenada L



# Busca binária

Quando a lista L estiver na ordem crescente podemos identificar a existência de um elemento de forma mais rápida.

Procurar 55 na lista ordenada L

L[0]==55?

**55**

0

**return True**

# Busca binária

Quando a lista L estiver na ordem crescente podemos identificar a existência de um elemento de forma mais rápida.

Imagine que o elemento procurado seja 50

Procurar 55 na lista ordenada L

L[0]==50?

**55**

0

# Busca binária

Quando a lista L estiver na ordem crescente podemos identificar a existência de um elemento de forma mais rápida.

Imagine que o elemento procurado seja 50?

L[0]==50?

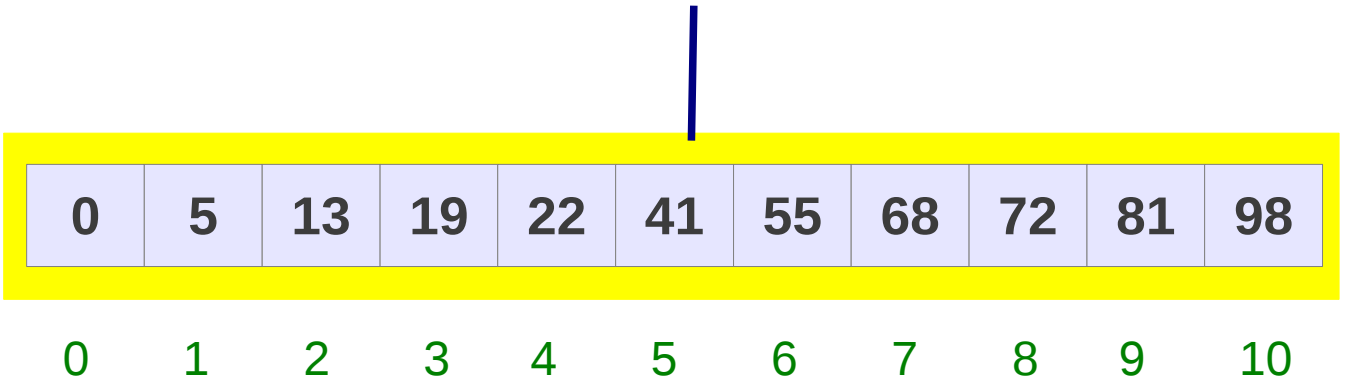
**55**

0

**return False**

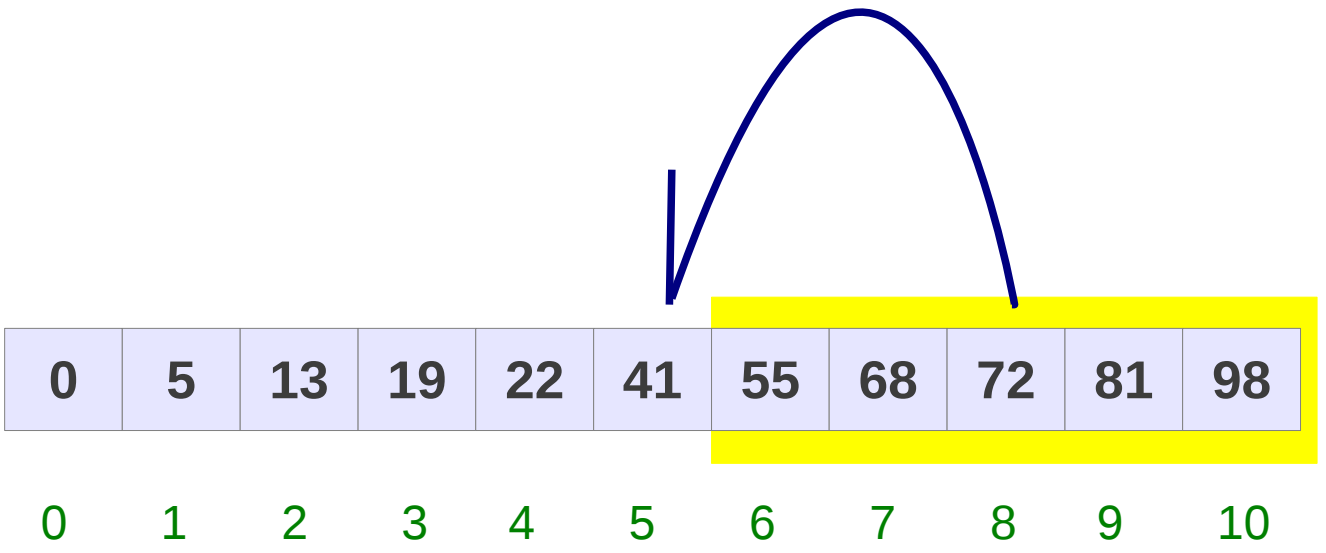
# Busca binária

Em resumo: Procurar 55 na lista ordenada L



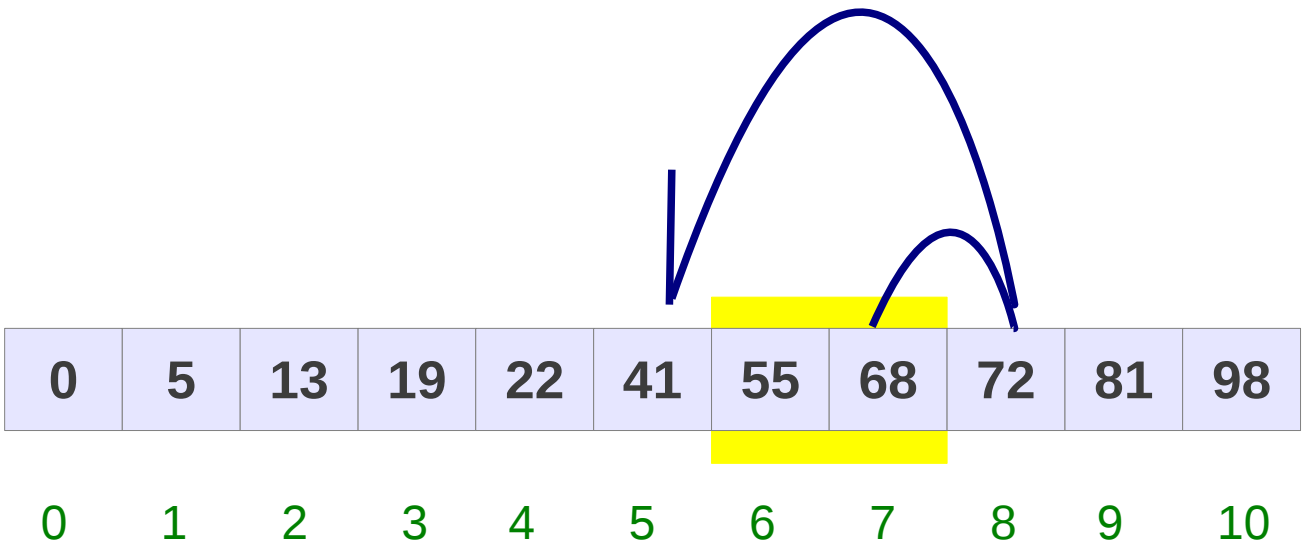
# Busca binária

Em resumo: Procurar 55 na lista ordenada L



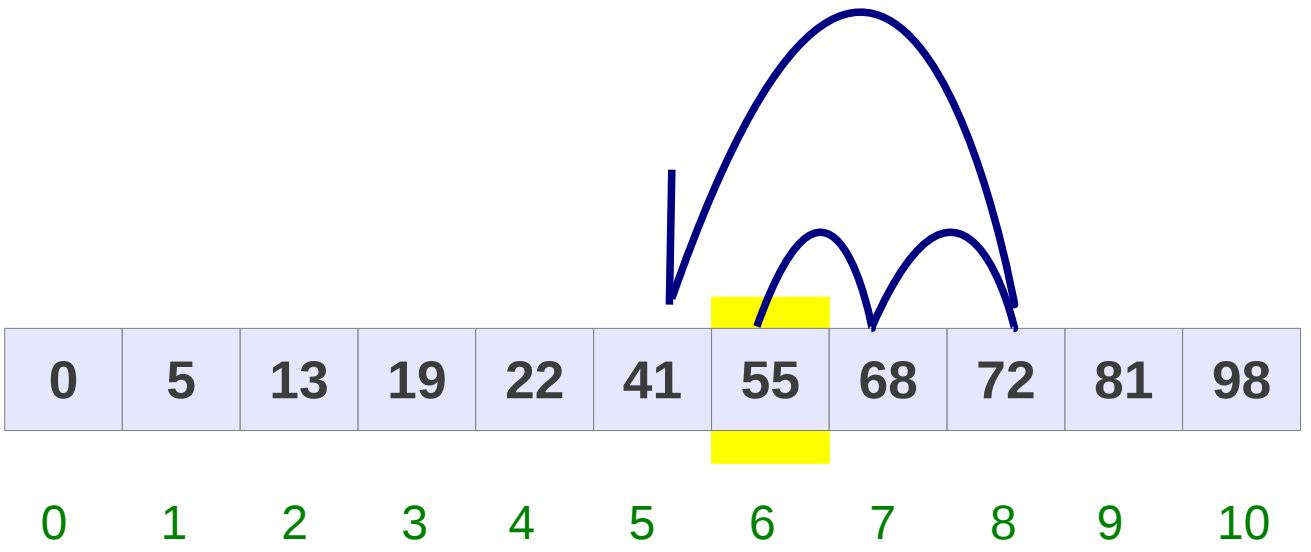
# Busca binária

Em resumo: Procurar 55 na lista ordenada L



# Busca binária

Em resumo: Procurar 55 na lista ordenada L



# Busca binária

Crie uma função que permita buscar um elemento **x** em uma lista **L** ordenada na forma crescente.

**Cabeçalho:** **def** busca_binaria**(x, L):**

# Busca binária

```
def busca_binaria(x, L):
    if len(L)==0:
        return False
    meio = len(L)/2
    if L[meio]==x:
        return True
    else:
        if x<L[meio]:
            return busca_binaria(x, L[:meio])
        else:
            return busca_binaria(x, L[meio+1:])
```

# Lista 07 (última)

**Questão única**.

**(a)** Dada uma lista com **n** números inteiros, determinar uma sub-sequência crescente de comprimento máximo. Exemplo**:** Na sequência 5,2,7,**1**,**4**,11,**6**,**9** a subsequência na cor vermelha é máxima e tem comprimento 4.

**Cabeçalho:** def subsequencia_maxima(L):
>>> subsequencia_maxima([9,5,6,3,9,6,4,7])
[5,6,9]

**(b)** Explique detalhadamente a abordagem considerada para uma lista de tamanho **n**.

**(c)** Apresente dois exemplo de execução passo a passo do algoritmo.

# Lista 07 (última)

A entrega da Lista 07 será através do Tidia-ae: Seção Atividades/lista-07. Até 07/07 (23h50) – Domingo.

Esta atividade pode ser realizada por grupos de 1, 2 ou 3 alunos.

OBS: **Todos** os membros do grupo devem obrigatoriamente enviar o relatório em formato PDF. No relatório deve de indicar-se o nome completo e RAs dos integrantes do grupo.