Processamento da Informação
– Teoria –

Algoritmos de ordenação

Semana 09
Prof. Jesús P. Mena-Chalco

19/06/2013

Algoritmo de ordenação

Um algoritmo de ordenação coloca os elementos de uma dada sequência em uma certa ordem (crescente ou decrescente) .

As ordens mais utilizadas são a numérica e lexicográfica.

Algoritmo de ordenação

Mais formalmente:

Um algoritmo de ordenação permite **permutar (ou seja rearranjar)** o elementos de uma lista L[0,...,n-1], de tal modo que eles fiquem em **ordem crescente**, ou seja, de tal forma que tenhamos:

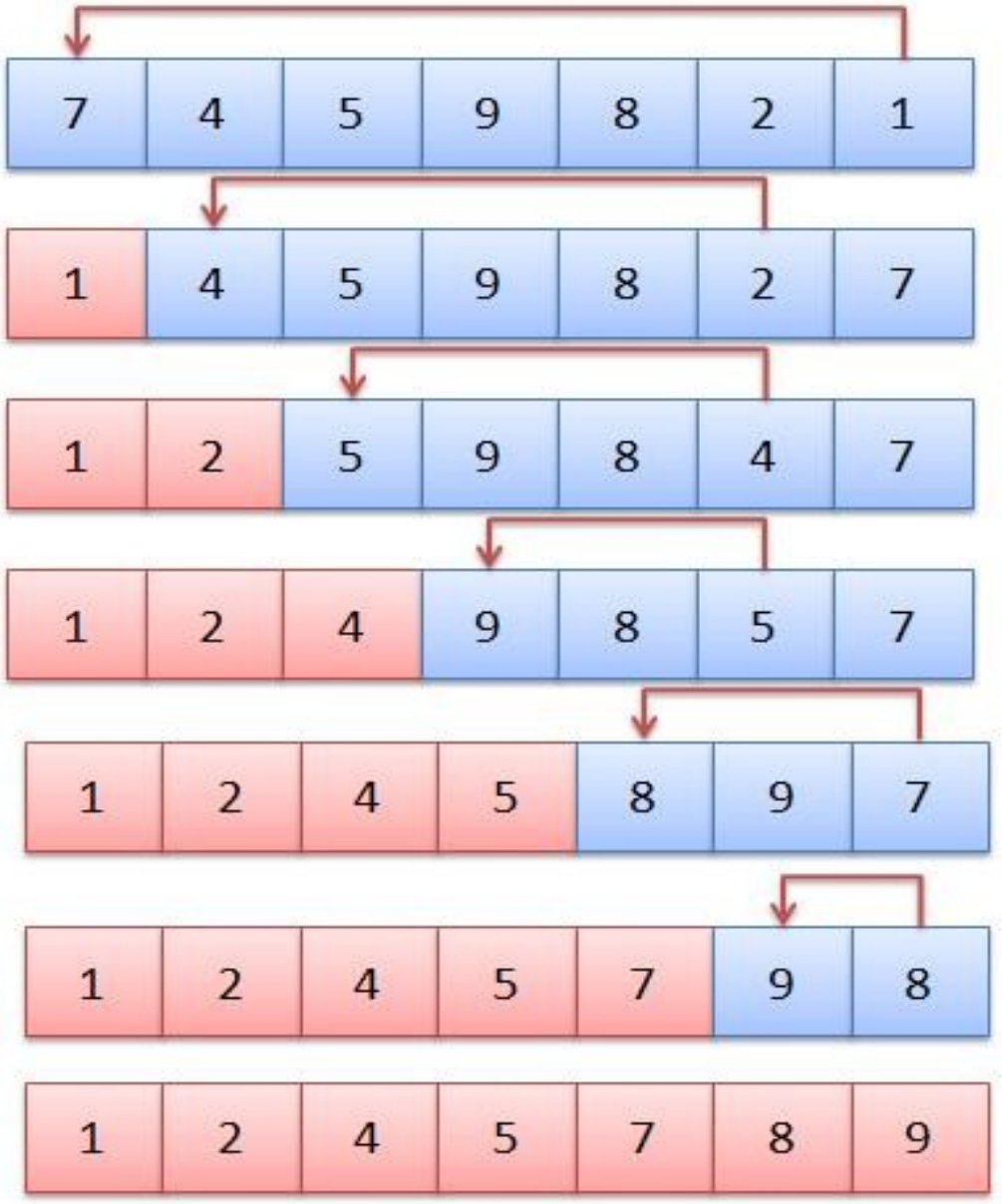
$$L[0] <= L[1] <= L[2] <= \ldots <= L[n-1]$$

Algoritmo de ordenação

Por que ordenar uma lista?

Os dados podem ser **acessados de forma mais eficiente** em uma lista ou sequência ordenada.

A. Ordenação por seleção (*Selection sort*)



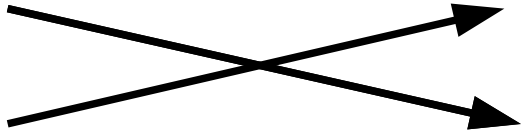
A. Ordenação por seleção (*Selection sort*)

```
def selection_sort(L):
    for i in range(0,len(L)-1):
        menor = i
        for j in range(i+1, len(L)):
            if L[menor]>L[j]:
                menor = j
        if menor!=i:
            temp = L[i]
            L[i] = L[menor]
            L[menor] = temp
    return L
```

A. Ordenação por seleção (*Selection sort*)

Troca de conteúdo de variáveis:

A = 'Joao'
B = 'Maria'



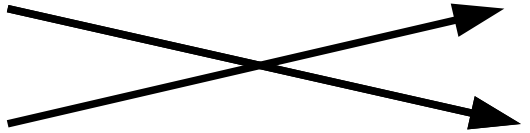
A = 'Maria'
B = 'Joao'

A = B
B = A

A. Ordenação por seleção (*Selection sort*)

Troca de conteúdo de variáveis:

A = 'Joao'
B = 'Maria'



A = 'Maria'
B = 'Joao'

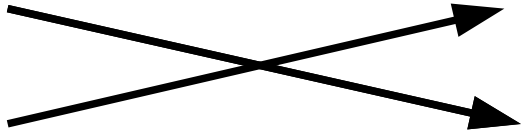
A = B
B = A

A = 'Maria'
B = 'Maria'

A. Ordenação por seleção (*Selection sort*)

Troca de conteúdo de variáveis:

A = 'Joao'
B = 'Maria'



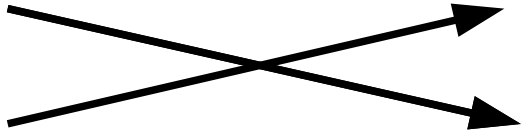
A = 'Maria'
B = 'Joao'

T = A
A = B
B = T

A. Ordenação por seleção (*Selection sort*)

Troca de conteúdo de variáveis:

A = 'Joao'
B = 'Maria'



A = 'Maria'
B = 'Joao'

T = A
A = B
B = T

T = 'Joao'
A = 'Maria'
B = 'Joao'

A. Ordenação por seleção (*Selection sort*)

```
def selection_sort(L):
    for i in range(0,len(L)-1):
        menor = i
        for j in range(i+1, len(L)):
            if L[menor]>L[j]:
                menor = j
        if menor!=i:
            temp = L[i]
            L[i] = L[menor]
            L[menor] = temp
    return L
```

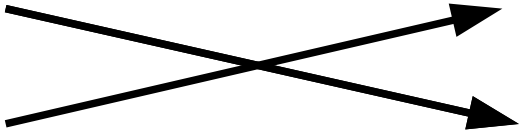
Primeira versão

Trocamos L[i] por L[menor]

A. Ordenação por seleção (*Selection sort*)

Troca de conteúdo de variáveis:

A = 'Joao'
B = 'Maria'



A = 'Maria'
B = 'Joao'

```
T = A
A = B
B = T
```

```
T = 'Joao'
A = 'Maria'
B = 'Joao'
```

```
A,B = B,A
```

Versão simplificada, usando tuplas em Python

A. Ordenação por seleção (*Selection sort*)

Segunda versão

```
def selection_sort(L):
    for i in range(0,len(L)-1):
        menor = i
        for j in range(i+1, len(L)):
            if L[menor]>L[j]:
                menor = j
            L[i], L[menor] = L[menor],L[i]
    return L
```

A. Ordenação por seleção (*Selection sort*)

Quando tempo o algoritmo consome para fazer o serviço?

```
def selection_sort(L):
    for i in range(0,len(L)-1):
        menor = i
        for j in range(i+1, len(L)):
            if L[menor]>L[j]:
                menor = j
        L[i], L[menor] = L[menor],L[i]
    return L
```

A. Ordenação por seleção (*Selection sort*)

O tempo é proporcional ao número de execuções da comparação L[menor]>L[j].

```
def selection_sort(L):
    for i in range(0,len(L)-1):
        menor = i
        for j in range(i+1, len(L)):
            if L[menor]>L[j]:
                menor = j
        L[i], L[menor] = L[menor],L[i]
    return L
```

A. Ordenação por seleção (*Selection sort*)

O tempo é proporcional ao número de execuções da comparação L[menor]>L[j].

```
def selection_sort(L):
    for i in range(0,len(L)-1):
        menor = i
        for j in range(i+1, len(L)):
            if L[menor]>L[j]:
                menor = j
        L[i], L[menor] = L[menor],L[i]
    return L
```

A. Ordenação por seleção (*Selection sort*)

O tempo é proporcional ao número de execuções da comparação L[menor]>L[j].

```
def selection_sort(L):
    for i in range(0,len(L)-1):
        menor = i
        for j in range(i+1, len(L)):
            if L[menor]>L[j]:
                menor = j
        L[i], L[menor] = L[menor],L[i]
    return L
```

9	8	7	6	5	4	3	2	1

1	8	7	6	5	4	3	2	9

1 iteração	n-1
2 iteração	n-2
3 iteração	n-3
...	
n-1iteração	1

A. Ordenação por seleção (*Selection sort*)

O tempo é proporcional ao número de execuções da comparação L[menor]>L[j].

1 iteração	n-1
2 iteração	n-2
3 iteração	n-3
…	
n-1iteração	1

Tempo = somatória de todas as comparações

A. Ordenação por seleção (*Selection sort*)

O tempo é proporcional ao número de execuções da comparação L[menor]>L[j].

1 iteração	n-1
2 iteração	n-2
3 iteração	n-3
…	
n-1iteração	1

Tempo = somatória de todas as comparações

Tempo = (n–1)(n)/2

Tempo = $n^2/2 – n/2$

A. Ordenação por seleção (*Selection sort*)

O tempo é proporcional ao número de execuções da comparação L[menor]>L[j].

1 iteração	n-1
2 iteração	n-2
3 iteração	n-3
…	
n-1iteração	1

Tempo = somatória de todas as comparações

Tempo = (n–1)(n)/2

Tempo = $n^2/2 – n/2$

Se a lista tiver n=1000 elementos, o número de comparações (tempo) será proporcional a: 499500.

A. Ordenação por seleção (*Selection sort*)

O tempo é proporcional ao número de execuções da comparação L[menor]>L[j].

1 iteração	n-1
2 iteração	n-2
3 iteração	n-3
…	
n-1iteração	1

Tempo = somatória de todas as comparações

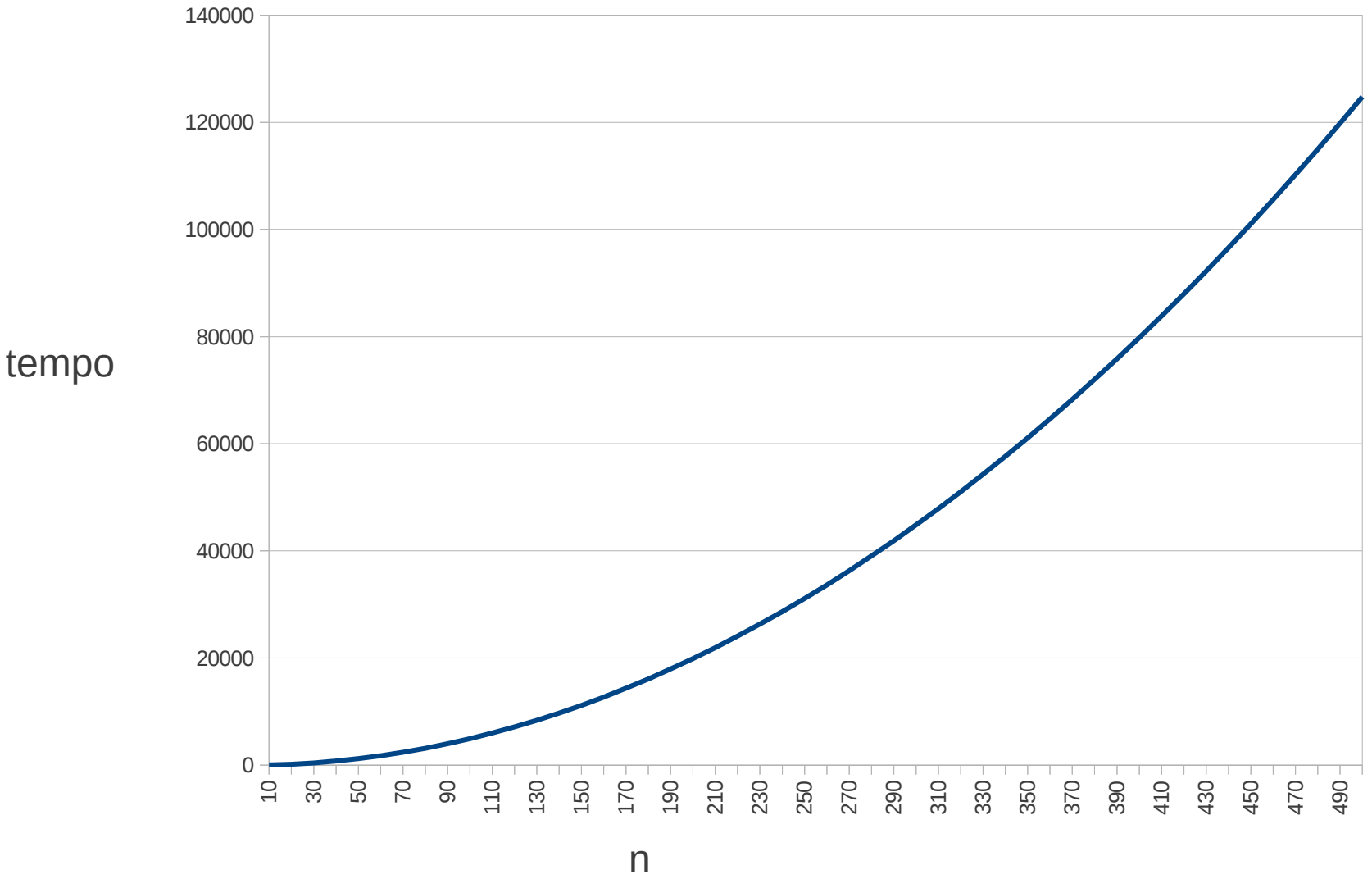
Tempo = (n–1)(n)/2

Tempo = $n^2/2 – n/2$

Se a lista tiver n=1000 elementos, o número de comparações (tempo) será proporcional a: 499500.

Se o computador fizer no máximo 10 comparações por segundo, o tempo gasto será de: 49950 segundos = 832 min = 13 horas

A. Ordenação por seleção (*Selection sort*)



A. Ordenação por seleção (*Selection sort*)

```
def selection_sort(L):
    for i in range(0,len(L)-1):
        menor = i
        for j in range(i+1, len(L)):
            if L[menor]>L[j]:
                menor = j
            L[i], L[menor] = L[menor],L[i]
    return L
```

```
>>> selection_sort([9,8,7,6,5,4,3,2,1])
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Lembrando recursividade...

```
def contagem_regressiva(n):
    if n==0:
        print "Fogo!"
    else:
        print n
        contagem_regressiva(n-1)
```

Lembrando recursividade...

```
def fact(n):
    if n==0:
        return 1
    else:
        return n*fatorial(n-1)
```

Lembrando recursividade...

```
def potencia(n,x):
    if x==1:
        return n
    else:
        return n*potencia(n,x-1)
```

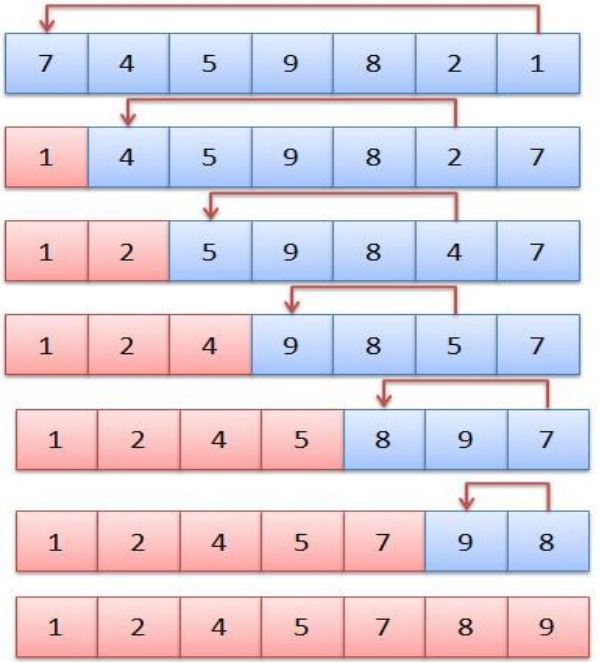
Lembrando recursividade...

```
def fib(n):
    if n==0:
        return 0
    elif n==1:
        return 1
    else:
        return fib(n-1)+fib(n-2)
```

A. Ordenação por seleção (*Selection sort*)

Crie uma versão recursiva do algoritmo Selection sort

```
def selection_sort(L):
    for i in range(0,len(L)-1):
        menor = i
        for j in range(i+1, len(L)):
            if L[menor]>L[j]:
                menor = j
        L[i], L[menor] = L[menor],L[i]
    return L
```



A. Ordenação por seleção (*Selection sort*)

```
def selection_sort_rec(L, indice):
    print L
    if indice >=len(L)-1:
        return L
    menor = indice
    for j in range(indice+1, len(L)):
        if L[menor]>L[j]:
            menor = j
    L[indice], L[menor] = L[menor],L[indice]
    return selection_sort_rec(L, indice+1)
```

A. Ordenação por seleção (*Selection sort*)

```
>>> a = selection_sort_rec([9,8,7,6,5,4,3,2,1],0)
[9, 8, 7, 6, 5, 4, 3, 2, 1]
```

Indice = 0

[1, 8, 7, 6, 5, 4, 3, 2, 9] Indice = 1

[1, 2, 7, 6, 5, 4, 3, 8, 9] Indice = 2

[1, 2, 3, 6, 5, 4, 7, 8, 9] Indice = 3

[1, 2, 3, 4, 5, 6, 7, 8, 9] Indice = 4

[1, 2, 3, 4, 5, 6, 7, 8, 9] Indice = 5

[1, 2, 3, 4, 5, 6, 7, 8, 9] Indice = 6

[1, 2, 3, 4, 5, 6, 7, 8, 9] Indice = 7

[1, 2, 3, 4, 5, 6, 7, 8, 9] Indice = 8

Indice > len(L)-1
8>9-1