Algoritmos e Programação de Computadores

Ordenação

Ref.: material original (1o S., T. KLMN). por **Profa. Sandra Avila**, Instituto de Computação (IC/ Unicamp)

Agenda

- O Problema da Ordenação
- Selection Sort
- Bubble Sort
- Insertion Sort
- Exercício

Ordenação

- Vamos estudar alguns algoritmos para o seguinte problema:

Dado uma coleção de elementos com ***uma relação de ordem entre si****, devemos gerar uma saída com os elementos ordenados.*

Ordenação

- O problema de ordenação é um dos mais básicos em computação.
  - Provavelmente é um dos problemas com o maior número de aplicações diretas ou indiretas.
- Exemplos de aplicações diretas
  - Criação de rankings; definir preferências em atendimentos por prioridade; criação de listas.
- Exemplos de aplicações indiretas
  - Otimizar sistemas de busca; manutenção de estruturas de bancos de dados.

Selection Sort
(Ordenação por Seleção)

Selection Sort (Ordenação por Seleção)

- Seja `vet` uma lista contendo números.
  - `vet = [5,3,2,1,90,6]`.
- Devemos deixar `vet` em ordem crescente.

Selection Sort (Ordenação por Seleção)

- A ideia do algoritmo é a seguinte:
  - Ache o menor elemento a partir da posição 0. Troque então este elemento com o elemento da posição 0.
  - Ache o menor elemento a partir da posição 1. Troque então este elemento com o elemento da posição 1.
  - Ache o menor elemento a partir da posição 2. Troque então este elemento com o elemento da posição 2.
  - E assim sucessivamente...

Selection Sort (Ordenação por Seleção)

- Exemplo: `[5,3,2,1,90,6]`.

Selection Sort (Ordenação por Seleção)

- Exemplo: `[5,3,2,1,90,6]`.
  - Iteração 1. Acha menor: `[5,3,2,1,90,6]` Faz troca: `[1,3,2,5,90,6]`
  - Iteração 2. Acha menor: `[1,3,2,5,90,6]` Faz troca: `[1,2,3,5,90,6]`
  - Iteração 3. Acha menor: `[1,2,3,5,90,6]` Faz troca: `[1,2,3,5,90,6]`
  - Iteração 4. Acha menor: `[1,2,3,5,90,6]` Faz troca: `[1,2,3,5,90,6]`
  - Iteração 5. Acha menor: `[1,2,3,5,90,6]` Faz troca: `[1,2,3,5,6,90]`

Selection Sort (Ordenação por Seleção)

- Como achar o menor elemento a partir de uma posição inicial?
- Vamos achar o índice do menor elemento em uma lista, a partir de uma posição inicial:

```
menor = inicio
for j in range(inicio, fim):
    if vet[menor] > vet[j]:
        menor = j
```

Selection Sort (Ordenação por Seleção)

- Criamos então uma função que retorna o índice do elemento mínimo de um vetor, a partir de uma posição `inicio` passado por parâmetro:

```
def indiceMenor(vet, inicio):
    menor = inicio
    for j in range(inicio, len(vet)):
        if vet[menor] > vet[j]:
            menor = j
    return menor
```

Selection Sort (Ordenação por Seleção)

- Dado a função anterior para achar o índice do menor elemento, como implementar o algoritmo de ordenação?
  - Ache o menor elemento a partir da posição 0, e troque com o elemento da posição 0.
  - Ache o menor elemento a partir da posição 1, e troque com o elemento da posição 1.
  - Ache o menor elemento a partir da posição 2, e troque com o elemento da posição 2.
  - E assim sucessivamente...

Selection Sort (Ordenação por Seleção)

```
def selectionSort(vet):
    for i in range(len(vet)-1):
        #Acha o menor elemento a partir da posição i
        menor = indiceMenor(vet, i)
        #Troca com o elemento da posição i
        aux = vet[i]
        vet[i] = vet[menor]
        vet[menor] = aux
```

Selection Sort (Ordenação por Seleção)

```
lista = [14, 7, 8, 34, 56, 4, 0, 9, -8, 100]
selectionSort(lista)
lista
```

```
[-8, 0, 4, 7, 8, 9, 14, 34, 56, 100]
```

Selection Sort (Ordenação por Seleção)

- Passo a passo para `[14, 7, 8, 34, 56, 4, 0, 9, -8, 100]`:

```
[-8, 7, 8, 34, 56, 4, 0, 9, 14, 100]
[-8, 0, 8, 34, 56, 4, 7, 9, 14, 100]
[-8, 0, 4, 34, 56, 8, 7, 9, 14, 100]
[-8, 0, 4, 7, 56, 8, 34, 9, 14, 100]
[-8, 0, 4, 7, 8, 56, 34, 9, 14, 100]
[-8, 0, 4, 7, 8, 9, 34, 56, 14, 100]
[-8, 0, 4, 7, 8, 9, 14, 56, 34, 100]
[-8, 0, 4, 7, 8, 9, 14, 34, 56, 100]
[-8, 0, 4, 7, 8, 9, 14, 34, 56, 100]
```

Selection Sort (Ordenação por Seleção)

- O uso da função para achar o índice do menor elemento não é estritamente necessária.

```
def selectionSort(vet):
    for i in range(len(vet)-1):
        #Acha o menor elemento a partir da posição i
        menor = i
        for j in range(i, len(vet)):
            if vet[menor] > vet[j]:
                menor = j
        #Troca com o elemento da posição i
        aux = vet[i]
        vet[i] = vet[menor]
        vet[menor] = aux
```

Selection Sort (Ordenação por Seleção)

- É muito comum a operação de troca de valores entre duas posições de uma lista.
- Python possui uma sintaxe resumida para fazer estas trocas.

```
def selectionSort(vet):
    for i in range(len(vet)-1):
        #Acha o menor elemento a partir da posição i
        menor = i
        for j in range(i, len(vet)):
            if vet[menor] > vet[j]:
                menor = j
        #Troca com o elemento da posição i
        vet[i], vet[menor] = vet[menor], vet[i]
```

Bubble Sort
(Ordenação por Bolha)

Bubble Sort (Ordenação por Bolha)

- Seja `vet` uma lista contendo números.
  - `vet = [5,3,2,1,90,6]`.
- Seja `tam` o tamanho da lista.
- Devemos deixar `vet` em ordem crescente.

Bubble Sort (Ordenação por Bolha)

- A ideia do algoritmo é a seguinte:
  - Compare `vet[0]` com `vet[1]` e troque-os se `vet[0] > vet[1]`.
  - Compare `vet[1]` com `vet[2]` e troque-os se `vet[1] > vet[2]`.
  - Compare `vet[2]` com `vet[3]` e troque-os se `vet[2] > vet[3]`.
  - ...
  - Compare `vet[tam-2]` com `vet[tam-1]` e troque-os se `vet[tam-2] > vet[tam-1]`.
  - E assim sucessivamente ...

Bubble Sort (Ordenação por Bolha)

- Após uma iteração repetindo estes passos o que podemos garantir?
  - O maior elemento estará na posição correta!
- Após outra iteração de trocas, o segundo maior elemento estará na posição correta.
- E assim sucessivamente.
- Quantas iterações destas trocas precisamos para deixar a lista ordenada?

Bubble Sort (Ordenação por Bolha)

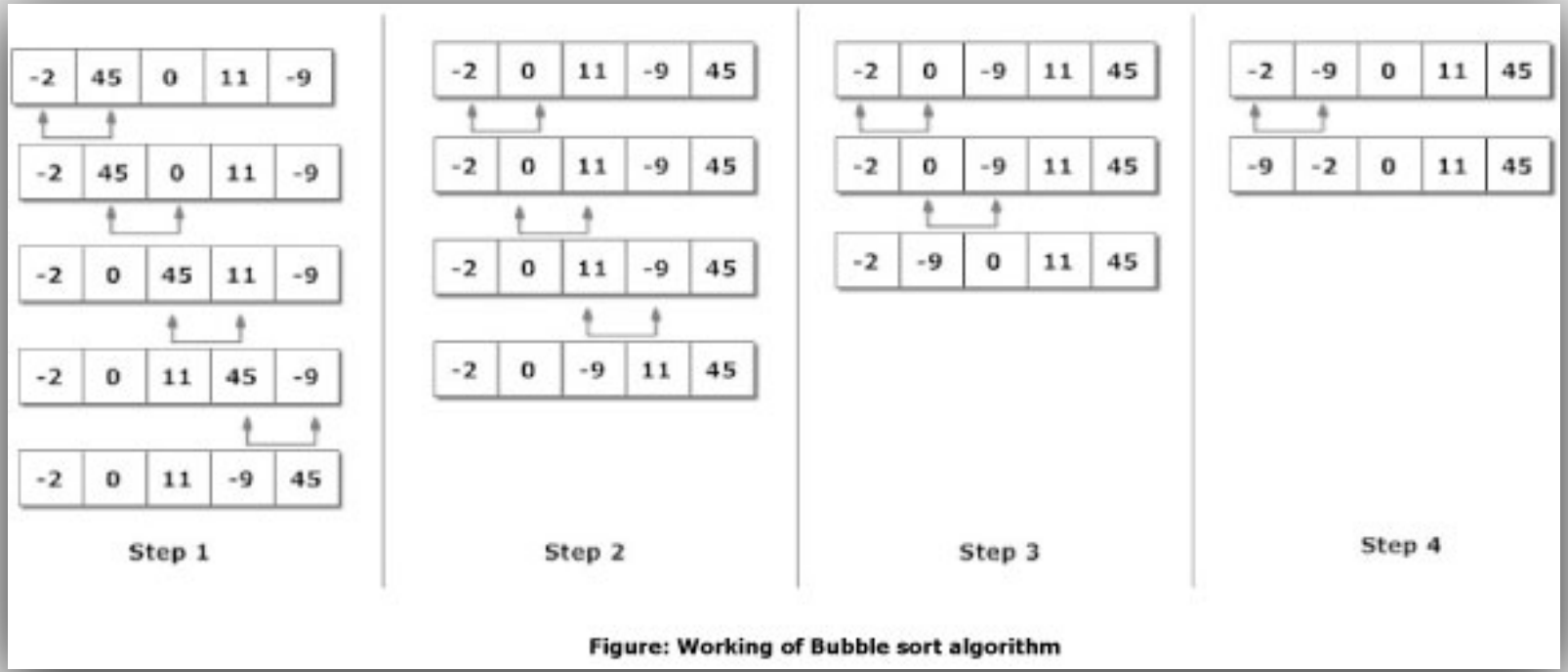


Figure: Working of Bubble sort algorithm

Bubble Sort (Ordenação por Bolha)

- Exemplo: `[5,3,2,1,90,6]`.
  - Iteração 1. `[5,3,2,1,90,6]` Faz troca: `[3,5,2,1,90,6]`
  - `[3,5,2,1,90,6]` Faz troca: `[3,2,5,1,90,6]`
  - `[3,2,5,1,90,6]` Faz troca: `[3,2,1,5,90,6]`
  - `[3,2,1,5,90,6]`
  - `[3,2,1,5,90,6]` Faz troca: `[3,2,1,5,6,90]`

- Isto termina a primeira iteração de trocas. Temos que repetir todo o processo mais 4 vezes!

Bubble Sort (Ordenação por Bolha)

- Exemplo: `[5,3,2,1,90,6]`.
  - Iteração 2. `[3,2,1,5,6,90]` Faz troca: `[2,3,1,5,6,90]`
    `[2,3,1,5,6,90]` Faz troca: `[2,1,3,5,6,90]`
    `[2,1,3,5,6,90]`
    `[2,1,3,5,6,90]`

Bubble Sort (Ordenação por Bolha)

- Exemplo: `[5,3,2,1,90,6]`.
  - Iteração 3. `[2,1,3,5,6,90]` Faz troca: `[1,2,3,5,6,90]`
    `[1,2,3,5,6,90]`
    `[1,2,3,5,6,90]`
  - Iteração 4. `[1,2,3,5,6,90]`
    `[1,2,3,5,6,90]`
  - Iteração 5. `[1,2,3,5,6,90]`

Bubble Sort (Ordenação por Bolha)

- O código abaixo realiza as trocas de uma iteração.
- São comparados e trocados, os elementos das posições: `0` e `1`; `1` e `2`; ...; `i-1` e `i`.
- Assumimos que de `(i+1)` até `(tam-1)`, a lista já tem os maiores elementos ordenados.

```
for j in range(i):
    if vet[j] > vet[j+1]:
        vet[j], vet[j+1] = vet[j+1], vet[j]
```

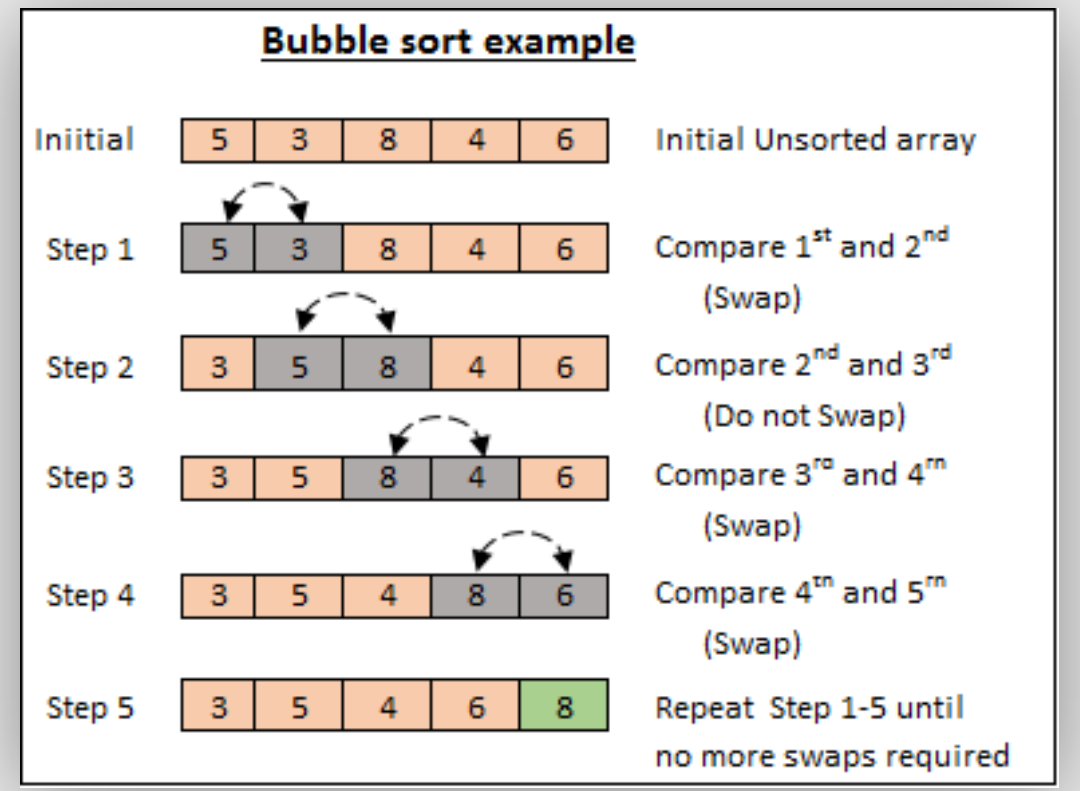
Bubble Sort (Ordenação por Bolha)

```
def bubbleSort(vet):
    #Índices i em ordem decrescente
    for i in range(len(vet)-1,0,-1):
        #Troca com o elemento da posição I
        for j in range(i):
            if vet[j] > vet[j+1]:
                vet[j], vet[j+1] = vet[j+1], vet[j]
```

Bubble Sort (Ordenação por Bolha)

- Note que as trocas na primeira iteração ocorrem até a última posição.
- Na segunda iteração ocorrem até a penúltima posição.
- E assim sucessivamente.
- Por que?

Bubble Sort (Ordenação por Bolha)



Insertion Sort
(Ordenação por Inserção)

Insertion Sort (Ordenação por Inserção)

- Seja `vet` uma lista contendo números.
  - `vet = [5,3,2,1,90,6]`.
- Devemos deixar `vet` em ordem crescente.

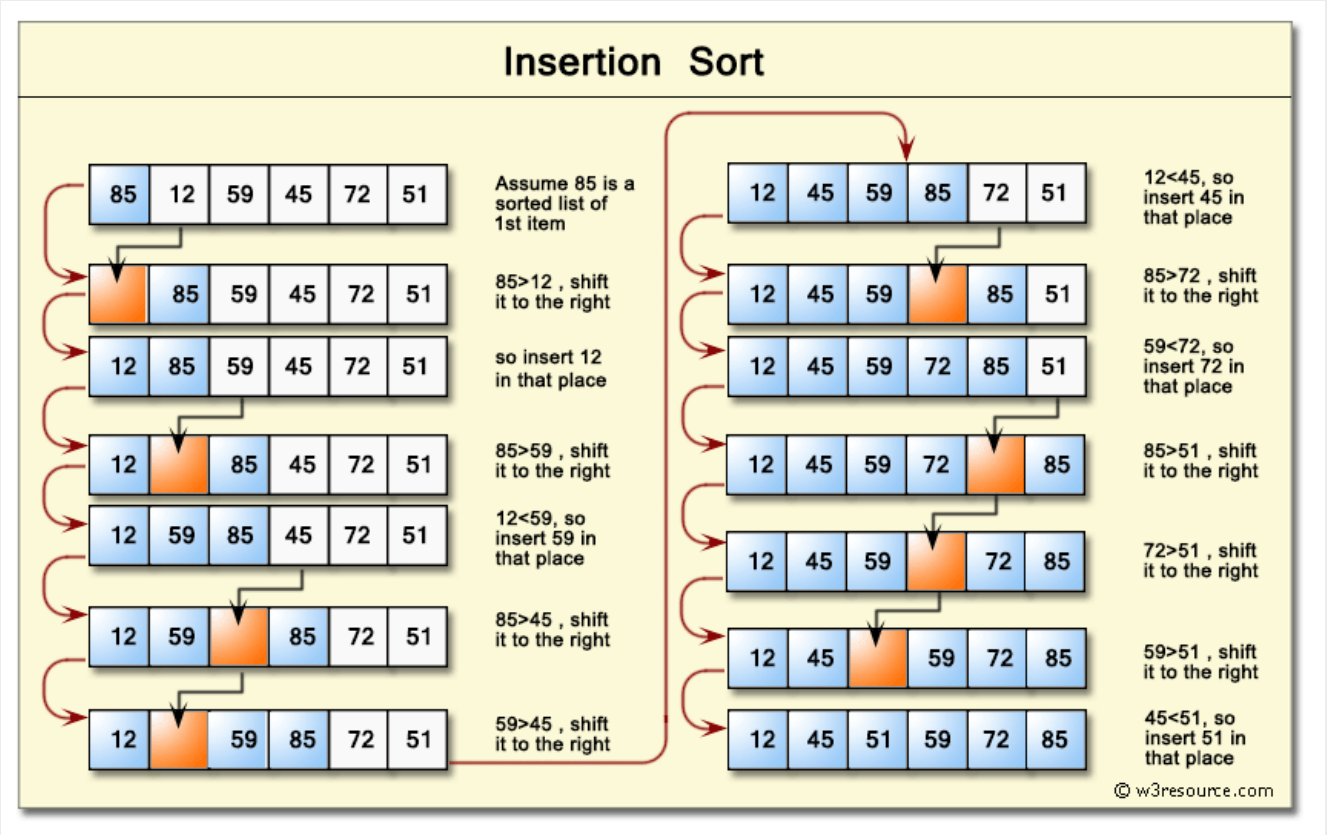
Insertion Sort (Ordenação por Inserção)

- A ideia do algoritmo é a seguinte:
  - A cada passo, uma porção de `0` até `i-1` da lista já está ordenada.
  - Devemos inserir o item da posição `i` na posição correta para deixar a lista ordenada até a posição `i`.
  - No passo seguinte consideramos que a lista está ordenada até `i`.

Insertion Sort (Ordenação por Inserção)

- Exemplo: `[5,3,2,1,90,6]`.
  - `[5,3,2,1,90,6]` : lista ordenada de `0-0`.
  - `[3,5,2,1,90,6]` : lista ordenada de `0-1`.
  - `[2,3,5,1,90,6]` : lista ordenada de `0-2`.
  - `[1,2,3,5,90,6]` : lista ordenada de `0-3`.
  - `[1,2,3,5,90,6]` : lista ordenada de `0-4`.
  - `[1,2,3,5,6,90]` : lista ordenada de `0-5`.

Insertion Sort (Ordenação por Inserção)



Insertion Sort (Ordenação por Inserção)

Funcionamento do algoritmo em passos
j=1 até j=n; n: tamanho da lista

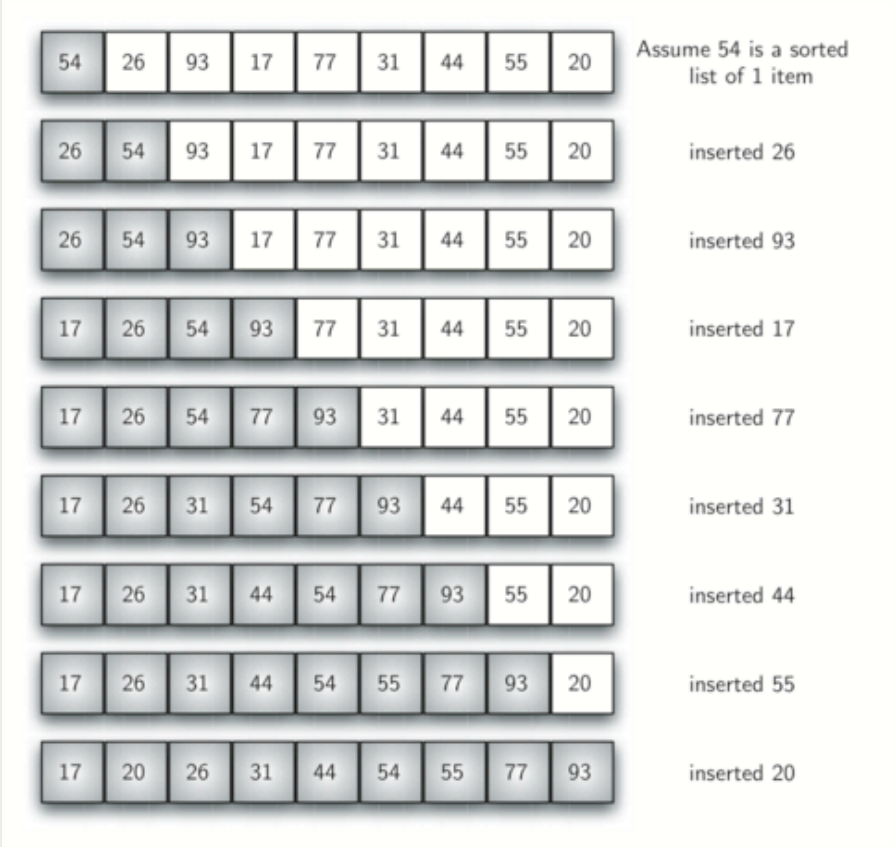
```
v = [54,26,93,17,77, ... , 20]
Inicio com j=1: elemento j-1, v[0], já
esta ordenado (v[0]=54)

j=2: elem. i=0 até i=1 comparados:
v[1]=26 fora da ordem? -> sim: inserção
v[1] em v[0](troca de posição)
v = [26,54,93,17,77, ... , 20]

j=3: elem. i=0 até i=2 comparados
[26,54,93, ... ]  (comp. até j-1)
v[2] fora da ordem?-> não: sem mudança;
v=[26,54,93,17,77, ... , 20]

j=4: elem. i=0 até i=3 comparados
[26,54,93,17, ... ] (comp. até j-1)
v[3]=17 fora da ordem? -> sim, inserção
v[3] na posição correta (v[0], ...)
v = [17,26,54,93,77, ... , 20]

... Continuar até j=n
```



Insertion Sort (Ordenação por Inserção)

- Presssuposição: a lista está ordenada de `0` até `i-1`.
- Caso o elemento **`i`** esteja fora de ordem, será inserido na posição correta.
  - Assim, todos os elementos à direita de `i` sao deslcocados em + 1 posição (*shift-to-right*)

```
aux = vet[i]  #inserir aux na posição correta
j = i-1 #analisar elementos das posições anteriores
while (j >=0 and vet[j] > aux): #enquanto v[j] > v[i] empurra
    vet[j+1] = vet[j] #vet[j] para frente
    j = j-1
vet[j+1] = aux
```

Insertion Sort (Ordenação por Inserção)

- Exemplo `[1,3,5,10,20,2,4]` com `i=5`.
  - `[1,3,5,10,20,2,4] : aux=2, j=4.`
  - `[1,3,5,10,20,2,4] : aux=2, j=3.`
  - `[1,3,5,10,20,2,4] : aux=2, j=2.`
  - `[1,3,5,10,20,2,4] : aux=2, j=1.`
  - `[1,3,5,10,20,2,4] : aux=2, j=0.`
- Aqui temos que `vet[j] < aux` logo fazemos `vet[j+1] = aux`.
- `[1,2,3,5,10,20,4] : aux=2, j=0.`

Insertion Sort (Ordenação por Inserção)

```
def insertionSort(vet):
    for i in range(1,len(vet)):
        aux = vet[i]
        j=i-1
        while (j >=0 and vet[j] > aux): #põe elementos vet[j] > vet[i]
            vet[j+1] = vet[j] #para frente
            j = j-1
        vet[j+1] = aux #põe v[i] na posição correta
```

Exercícios

Exercício

- Altere os algoritmos vistos nesta aula para que estes ordenem uma lista de inteiros em ordem decrescente ao invés de ordem crescente.

Referências & Exercícios

- Os slides dessa aula foram baseados no material de MC102 dos Profs. Sandra Avila e Eduardo Xavier (IC/Unicamp).

- https://panda.ime.usp.br/aulasPython/static/aulasPython/aula24.html
- Selection Sort 3D Animation: https://youtu.be/EdUWyka7kpI?t=57s
- Bubble Sort 3D Animation: https://youtu.be/NiyEqLZmngY?t=55s
- Livro:
  - Introduction to Algorithms (2nd. edition), por Thomas Cormen, C. Leiserson, R. Rivest, C. Stein, The MIT Press, McGraw-Hill Book Co., 2001 (1rst ed. 1990).