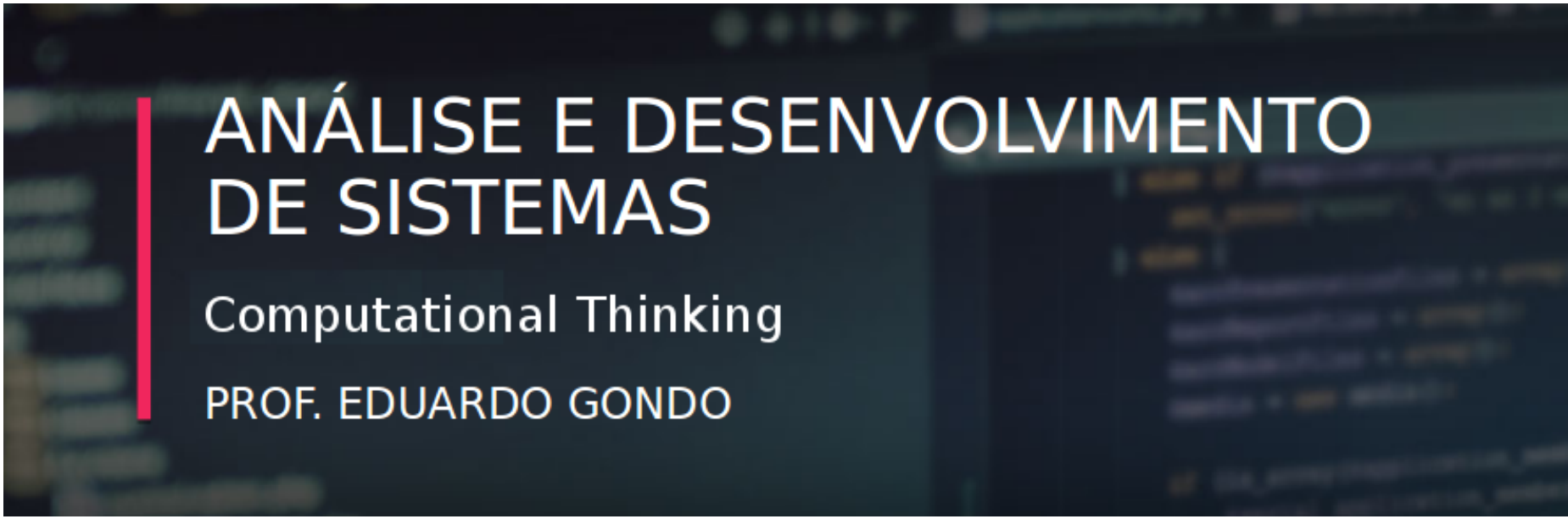


Programs must be written for people to read, and incidentally for machines to execute.

H. Abelson, G. Sussman

Ordenação

PROBLEMA: Encontrar uma permutação sobre um lista `v[0..n-1]` tal que $v[0] \leq v[1] \leq \ldots \leq v[n-1]$.

O interesse em encontrar algoritmos para ordenar elementos está na eficiência em buscas sobre o lista.

Por que o algoritmo de busca binária é muito mais eficiente do que o algoritmo de busca simples.

Ordenação — Algoritmo 1

- é fácil ver que podemos ordenar o lista gerando todas as permutações dos elementos de v e escolher uma permutação onde os elementos de v estão em ordem crescente
- o problema dessa solução é que ela demora muito quando o número de elementos de v é grande
- desenvolva uma função em Python que recebe um lista contendo uma lista e verifica se ela está ordenada: exercício 1 da lista de exercícios

Ordenação — Algoritmo 2

- considere o seguinte subproblema: dado um lista v encontre a posição do menor elemento do lista
- será que este problema nos ajuda a ordenar um lista?

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	12	84	5	10	17

Vamos procurar o índice do menor elemento da lista.

Ordenação — Algoritmo 2

- considere o seguinte subproblema: dado um lista v encontre a posição do menor elemento do lista
- será que este problema nos ajuda a ordenar um lista?

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	12	84	5	10	17

Vamos procurar o índice do menor elemento da lista.

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	12	84	5	10	17

A posição do menor índice está selecionado em vermelho, em qual posição devemos colocar este elemento se queremos ordenar o lista?

Ordenação — Simulação

Na primeira posição da lista!

Ordenação — Simulação

Na primeira posição da lista!

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	12	84	5	10	17

Vamos trocar de posição os elementos selecionados em azul.

Ordenação — Simulação

Na primeira posição da lista!

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	12	84	5	10	17

Vamos trocar de posição os elementos selecionados em azul.

0	1	2	3	4	5	6	7	8	9
5	15	8	19	30	12	84	10	10	17

agora, vamos procurar o índice do menor elemento da lista desconsiderando a primeira posição (em verde).

Ordenação — Simulação

Em vermelho temos o índice do menor elemento da lista desconsiderando a posição de índice 0

0	1	2	3	4	5	6	7	8	9
5	15	8	19	30	12	84	10	10	17

Ordenação — Simulação

Em vermelho temos o índice do menor elemento da lista desconsiderando a posição de índice 0

0	1	2	3	4	5	6	7	8	9
5	15	8	19	30	12	84	10	10	17

Vamos trocar de posição os elementos selecionados em azul.

0	1	2	3	4	5	6	7	8	9
5	15	8	19	30	12	84	10	10	17

Ordenação — Simulação

Em vermelho temos o índice do menor elemento da lista desconsiderando a posição de índice 0

0	1	2	3	4	5	6	7	8	9
5	15	8	19	30	12	84	10	10	17

Vamos trocar de posição os elementos selecionados em azul.

0	1	2	3	4	5	6	7	8	9
5	15	8	19	30	12	84	10	10	17

segue agora o lista com as duas primeiras posições em ordem

0	1	2	3	4	5	6	7	8	9
5	8	15	19	30	12	84	10	10	17

Ordenação — Simulação

Em vermelho temos o índice do menor elemento da lista desconsiderando as posições de índice 0 e 1

0	1	2	3	4	5	6	7	8	9
5	8	15	19	30	12	84	10	10	17

Ordenação — Simulação

Em vermelho temos o índice do menor elemento da lista desconsiderando as posições de índice 0 e 1

0	1	2	3	4	5	6	7	8	9
5	8	15	19	30	12	84	10	10	17

Vamos trocar de posição os elementos selecionados em azul.

0	1	2	3	4	5	6	7	8	9
5	8	15	19	30	12	84	10	10	17

Ordenação — Simulação

Em vermelho temos o índice do menor elemento da lista desconsiderando as posições de índice 0 e 1

0	1	2	3	4	5	6	7	8	9
5	8	15	19	30	12	84	10	10	17

Vamos trocar de posição os elementos selecionados em azul.

0	1	2	3	4	5	6	7	8	9
5	8	15	19	30	12	84	10	10	17

segue agora o lista com as três primeiras posições em ordem

0	1	2	3	4	5	6	7	8	9
5	8	10	19	30	12	84	15	10	17

Ordenacão — Selection Sort

- vamos voltar ao problema de encontrar a posição do menor elemento da lista
- dado um lista v e um natural i, encontre a posição de um menor elemento do lista a partir de i
- segue a assinatura da função `def menor(v, i)`
- chamando `j = menor(v, 0)` e trocando o elemento $v[0]$ com $v[j]$ posicionamos o elemento de menor valor na primeira posicao da lista v
- agora, basta repetir para `j = menor(v, 1)` e trocar o elemento $v[1]$ com $v[j]$ posicionamos o segundo elemento de menor valor na segunda posição da lista v
- repetimos o processo até para `j = menor(v, len(v)-1)` onde teremos nosso lista ordenado

Selection Sort — Separado em 2 funções

Segue a implementação da ordenação

```
1  def selectionSort(v):
2      for i in range(len(v)-1):
3          j = menor(v, i)
4          aux = v[i]
5          v[i] = v[j]
6          v[j] = aux
7          #ou
8          #v[i], v[j] = v[j], v[i]
```

a implementação da função `menor`:

```
1  def menor(vetor, i):
2      pos = i
3      i = i + 1
4      while i < len(vetor):
5          if vetor[i] < vetor[pos]:
6              pos = i
7          i = i + 1
8      return pos
```

Selection Sort — em uma única função

Segue a implementação da ordenação com os comandos de repetição encadeados:

```
 1  def selectionSort(v):
 2      for i in range(len(v)-1):
 3          pos = i
 4          j = i + 1
 5          while j < len(v):
 6              if v[j] < v[pos]:
 7                  pos = j
 8              j = j + 1
 9
10          aux = v[i]
11          v[i] = v[pos]
12          v[pos] = aux
```

Insertion Sort

Antes de mostrar o algoritmo de ordenação Insertion Sort, vamos analisar a situação: dados um lista de números inteiros onde, apenas a última posição, não está em ordem crescente. Veja um exemplo na figura abaixo:

0	1	2	3	4	5	6	7	8	9
10	15	18	19	30	32	44	55	67	27

Você consegue elaborar um algoritmo para colocar na ordem esse último elemento?

Insertion Sort

A ideia é abrir espaço no lista para colocar o 27 na posição correta.
Armazenando o 27 em uma variável movimentamos os elementos:

0	1	2	3	4	5	6	7	8	9
10	15	18	19	30	32	44	55		67

0	1	2	3	4	5	6	7	8	9
10	15	18	19	30	32	44		55	67

0	1	2	3	4	5	6	7	8	9
10	15	18	19	30	32		44	55	67

0	1	2	3	4	5	6	7	8	9
10	15	18	19	30		32	44	55	67

Insertion Sort

0	1	2	3	4	5	6	7	8	9
10	15	18	19		30	32	44	55	67

Neste momento, encontramos a posição que o 27 deve ser inserido:

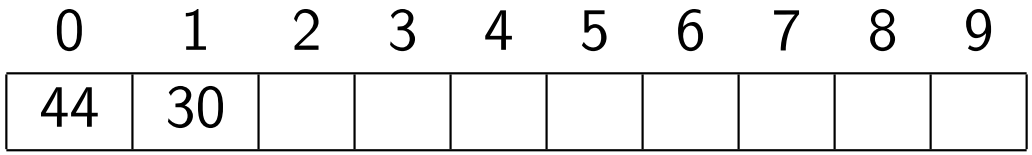
0	1	2	3	4	5	6	7	8	9
10	15	18	19	27	30	32	44	55	67

Segue o algoritmo que coloca o último elemento na posição correta:

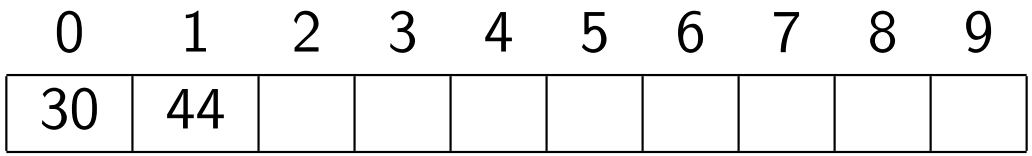
```
1  def organiza(lista):
2      i = len(lista) - 1
3      aux = lista[i]
4      while i > 0 and lista[i-1] > aux:
5          lista[i] = lista[i-1]
6          i = i - 1
7
8      lista[i] = aux
```

Insertion Sort

Considere a situação onde queremos ordenar apenas os dois primeiros elementos da lista:



Alterando a função `organiza(lista)` para `organiza(lista, pos)` onde pos indica a posição da lista que está fora de ordem. Chamando o método para a lista acima com pos=1, obtemos:



Insertion Sort

Suponha que no índice 2 temos o 18:

0	1	2	3	4	5	6	7	8	9
30	44	18							

Repetindo a chamada da função organiza para pos=2, obtemos:

0	1	2	3	4	5	6	7	8	9
18	30	44							

Continuando as chamadas até a última posição da lista, obtemos a lista ordenada!

Insertion Sort - Implementação

Veja abaixo, a implementação completa do insertion sort:

```
1  def insertionSort(lista):
2      for i in range(1, len(lista)):
3          j = i
4          aux = lista[j]
5          while j > 0 and lista[j-1] > aux:
6              lista[j] = lista[j-1]
7              j = j - 1
8
9          lista[j] = aux
```

Ordenação — BubbleSort

- imagine um vetor desenhado na vertical cujos elementos são bolhas em um tanque de água com densidade proporcional ao seu valor
- a tendência é que as bolhas mais "leves" (com valor menor) vão subir
- o algoritmo de ordenação bubble sort baseia-se nessa idéia
- pegamos o último elemento do vetor e comparamos com o anterior, se o elemento de baixo é menor que o elemento de cima trocamos ele de lugar
- repetimos esse processo até a primeira posição do vetor
- nessa situação na primeira posição temos o menor elemento do vetor na primeira posição

Ordenação — Simulação Bubble Sort

Vamos imaginar a situação de bolhas com densidades diferentes e movimentar os elementos do vetor:

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	12	84	5	10	**17**

Ordenação — Simulação Bubble Sort

Vamos imaginar a situação de bolhas com densidades diferentes e movimentar os elementos do vetor:

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	12	84	5	10	**17**

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	12	84	5	**10**	17

Simulação Bubble Sort

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	12	84	**5**	10	17

Simulação Bubble Sort

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	12	84	**5**	10	17

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	12	**5**	84	10	17

Ordenação — Simulação

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	**5**	12	84	10	17

Ordenação — Simulação

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	**5**	12	84	10	17

0	1	2	3	4	5	6	7	8	9
10	15	8	19	**5**	30	12	84	10	17

Ordenação — Simulação

0	1	2	3	4	5	6	7	8	9
10	15	8	19	30	**5**	12	84	10	17

0	1	2	3	4	5	6	7	8	9
10	15	8	19	**5**	30	12	84	10	17

0	1	2	3	4	5	6	7	8	9
10	15	8	**5**	19	30	12	84	10	17

Ordenação — Simulação

0	1	2	3	4	5	6	7	8	9
10	15	**5**	8	19	30	12	84	10	17

Ordenação — Simulação

0	1	2	3	4	5	6	7	8	9
10	15	**5**	8	19	30	12	84	10	17

0	1	2	3	4	5	6	7	8	9
10	**5**	15	8	19	30	12	84	10	17

Ordenação — Simulação

0	1	2	3	4	5	6	7	8	9
10	15	**5**	8	19	30	12	84	10	17

0	1	2	3	4	5	6	7	8	9
10	**5**	15	8	19	30	12	84	10	17

0	1	2	3	4	5	6	7	8	9
5	10	15	8	19	30	12	84	10	17

Ordenação — Simulação

Vamos repetir o processo já que a "bolha" mais leve está na 1ª posição do vetor:

0	1	2	3	4	5	6	7	8	9
5	10	15	8	19	30	12	84	10	**17**

Ordenação — Simulação

Vamos repetir o processo já que a "bolha" mais leve está na 1ª posição do vetor:

0	1	2	3	4	5	6	7	8	9
5	10	15	8	19	30	12	84	10	**17**

0	1	2	3	4	5	6	7	8	9
5	10	15	8	19	30	12	84	**10**	17

Ordenação — Simulação

Vamos repetir o processo já que a "bolha" mais leve está na 1ª posição do vetor:

0	1	2	3	4	5	6	7	8	9
5	10	15	8	19	30	12	84	10	**17**

0	1	2	3	4	5	6	7	8	9
5	10	15	8	19	30	12	84	**10**	17

0	1	2	3	4	5	6	7	8	9
5	10	15	8	19	30	12	**10**	84	17

Ordenação — Simulação

0	1	2	3	4	5	6	7	8	9
5	10	15	8	19	30	**10**	12	84	17

Ordenação — Simulação

0	1	2	3	4	5	6	7	8	9
5	10	15	8	19	30	**10**	12	84	17

0	1	2	3	4	5	6	7	8	9
5	10	15	8	19	**10**	30	12	84	17

Ordenação — Simulação

0	1	2	3	4	5	6	7	8	9
5	10	15	8	19	30	**10**	12	84	17

0	1	2	3	4	5	6	7	8	9
5	10	15	8	19	**10**	30	12	84	17

0	1	2	3	4	5	6	7	8	9
5	10	15	8	**10**	19	30	12	84	17

Ordenação — Simulação

0	1	2	3	4	5	6	7	8	9
5	10	15	**8**	10	19	30	12	84	17

Ordenação — Simulação

0	1	2	3	4	5	6	7	8	9
5	10	15	**8**	10	19	30	12	84	17

0	1	2	3	4	5	6	7	8	9
5	10	**8**	15	10	19	30	12	84	17

Ordenação — Simulação

0	1	2	3	4	5	6	7	8	9
5	10	15	**8**	10	19	30	12	84	17

0	1	2	3	4	5	6	7	8	9
5	10	**8**	15	10	19	30	12	84	17

0	1	2	3	4	5	6	7	8	9
5	**8**	10	15	10	19	30	12	84	17

BubbleSort - implementação

- implemente o algoritmo descrito acima no seguinte método:

```
1       def subir(vetor)
```

- agora modifique o método subir para que ele não suba o valor sempre até a posição 0, mas sim até a posição i passada como parâmetro:

```
1       def subir(vetor, i)
```

- após fazer o método subir, o algoritmo bubble sort fica:

```
1       def bubbleSort(vetor):
2           for i in range(len(vetor)):
3               subir(vetor, i)
```

Referência Bibliográfica

- Puga e Rissetti - Lógica de Programação e Estrutura de Dados
- Ascêncio e Campos - Fundamentos da Programação de Computadores
- Forbelone e Eberspacher - Lógica de programação: a construção de algoritmos e estruturas de dados
- Documentação do Python - https://docs.python.org/3.8/
- Python Programming For Beginners: Learn The Basics Of Python Programming (Python Crash Course, Programming for Dummies) (English Edition). Kindle
- Python: 3 Manuscripts in 1 book: - Python Programming For Beginners - Python Programming For Intermediates - Python Programming for Advanced (English Edition). Kindle

Copyleft