Ordenação

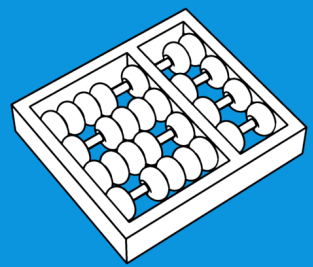
MC102 - Algoritmos e Programação de Computadores

05/24

21

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

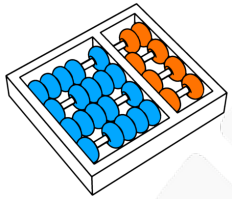
UNICAMP



“A boa ordem é a base de todas as coisas.”

Edmund Burke.

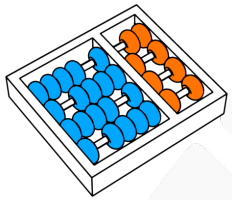
Dúvidas da aula anterior



Dúvidas selecionadas

- Não entendi direito como, na prática, seria escrever algo no meio de um arquivo com append. Você pode dar um exemplo?
- Tem como eu executar um código que esta dentro de um arquivo quando dou open?
- Se todo o arquivo que eu leio vem no formato de string, como que eu conseguiria ler uma imagem? Ela vem num formato tipo pbm?
- Não entendi exatamente a utilidade de w+, a+ e r+.
- Um programa em Python pode modificar o arquivo de onde próprio está rodando?
- Não entendi direito como funciona o encoding e como eu saberia qual usar em cada caso.
- Não entendi a diferença do arquivo.write() e arquivo.writelines().
- Diferentes partes de um software podem ter diferentes tipos de codificação?
- Existe alguma diferença de abrir o arquivo com o endereço local do "global"?
- Não entendi o conceito de "encoding".
- Quando os slides serao disponibilizados na pagina da disciplina?
- Não entendi como funciona o with as em python e se ele pode ser usado pra outras coisas além de arquivos.
- Quando devo usar os.path.join ao invés de concatenar strings para formar caminhos de arquivos?

Lembrando definições



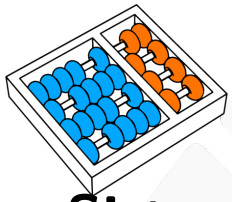
Problema Computacional

Um **problema computacional** é uma **função** que relaciona cada possível **entrada** com um conjunto de **saídas**:

- A entrada é o que chamamos de **instância**.
- A saída é o que chamamos de **solução**.

Mínimo: Dada uma lista I de números, encontrar o índice do menor valor que aparece em I:

- Instância: $I = [7, 1, 3, -2, 9, -2]$.
- Soluções: 3 e 5.



Problema Computacional

Sistema de Equações Lineares 2×2: Dados números a_{11}, a_{12}, a_{21}, a_{22}, b_1 e b_2, encontrar x_1 e x_2 tal que:

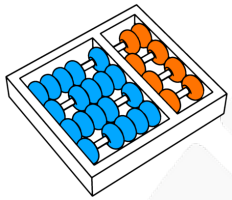
$$a_{11}x_1 + a_{12}x_2 = b_1$$
$$a_{21}x_1 + a_{22}x_2 = b_2$$

Exemplo: Se a instância é $a_{11} = 5$, $a_{12} = 20$, $a_{21} = 1$, $a_{22} = 2$, $b_1 = 5$ e $b_2 = 3$ então temos o seguinte sistema:

$$5x_1 + 20x_2 = 5$$
$$x_1 + 2x_2 = 3$$

e a solução (única) é $x_1 = 5$ e $x_2 = -1$.

Sistema de Equações Lineares: Dados $\mathbf{A} \in \mathbb{R}^{n \times m}$ e $\mathbf{b} \in \mathbb{R}^n$, encontrar $\mathbf{x} \in \mathbb{R}^m$ tal que $A \cdot x = b$.



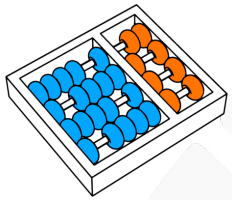
Problema Computacional

Caixeiro Viajante: Dados um número n de cidades e, para cada par (i, j) de cidades com $1 \leq i, j \leq n$, um número d_{ij} indicando a distância entre as cidades i e j, encontrar uma rota de distância mínima que percorra todas as cidades.

Temos vários outros problemas:

- Cálculos de expressões em geral.
- Detectar primalidade.
- Buscar um texto em uma string.
- etc.

No final das contas, cada exercício nos lab era um problema computacional.



Algoritmos

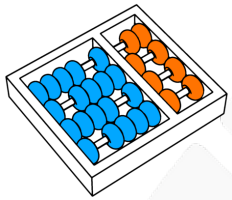
Um **ALGORITMO** é:

- Uma sequência de passos suficientemente simples:
  - Simples: o computador é capaz de executá-los.
  - Ou até mesmo uma pessoa com papel (e muita paciência).
  - De fato, existe uma definição matemática para isso...
- Que termina para qualquer entrada.

Um algoritmo A resolve um problema computacional P se:

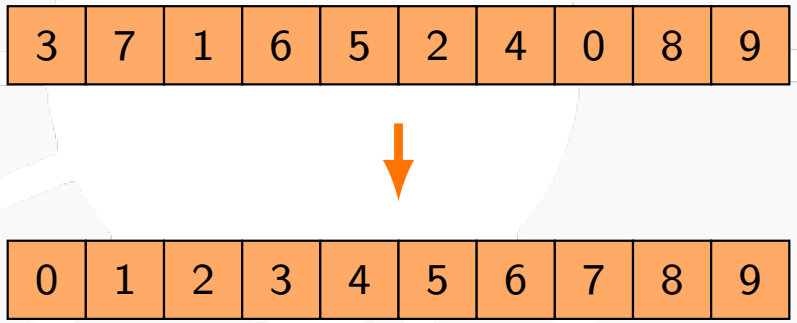
- Para qualquer instância de P.
- A devolve uma solução para esta instância.
- Isto é, A sempre dá uma resposta correta para P.

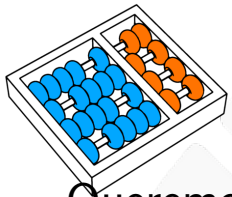
Ordenação



Ordenação

Ordenação: Dada uma lista l de n elementos, rearranjar os elementos de l de forma que $l[0] \leq l[1] \leq \cdots \leq l[n-1]$.





Aquecimento — Ordenação de três elementos

Queremos ordenar uma lista de três elementos.

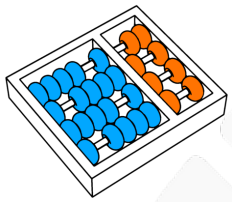
- Há um algoritmo que verifica as $3! = 6$ possibilidade e ordena:
  - Mas isso claramente não escala para n elementos. . .
- Ideia de um algoritmo menor (e útil):
  - Vamos colocar o menor elemento na primeira posição.
  - Em seguida ordenamos `I[1]` e `I[2]`.

```
1 se I[0] > I[1]
2   |  Troque I[0] com I[1]
3 se I[0] > I[2]
4   |  Troque I[0] com I[2]
5 se I[1] > I[2]
6   |  Troque I[1] com I[2]
```

O algoritmo resolve o problema pois:

- Após a linha 2, `I[0]` tem o valor mínimo entre `I[0]` e `I[1]`.
- Após a linha 4, `I[0]` tem o valor mínimo entre `I[0]`, `I[1]` e `I[2]`.
- Após a linha 6, `I` está ordenada.

SelectionSort



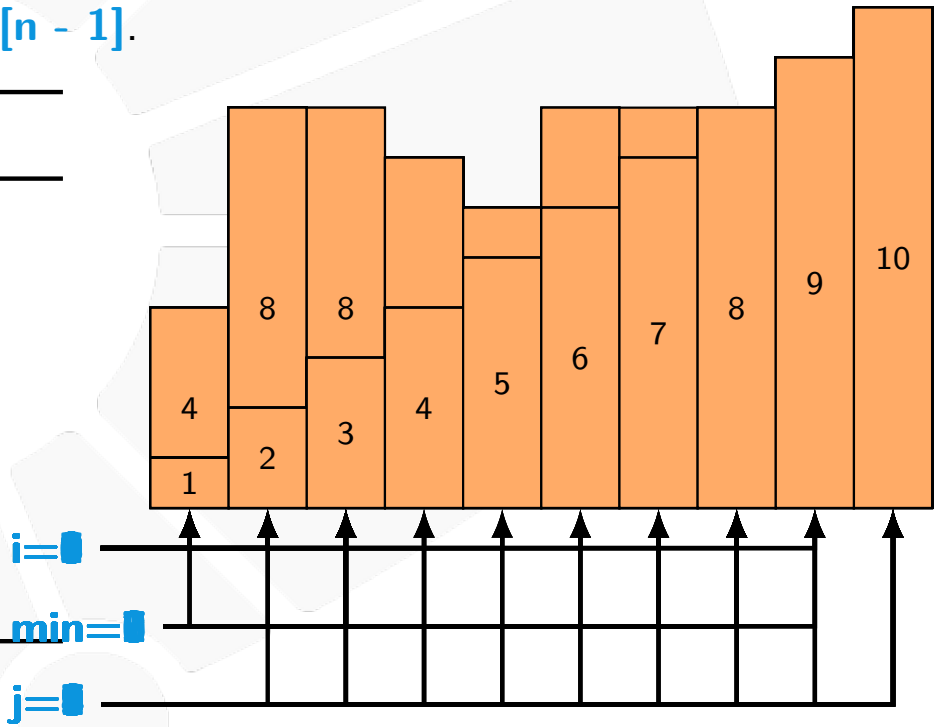
Ordenação por Seleção

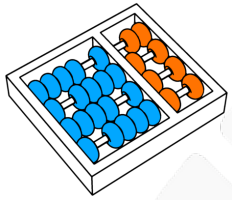
Ideia:

- Trocar I[0] com o mínimo de I[0], . . ., I[n - 1].
- Trocar I[1] com o mínimo de I[1], . . ., I[n - 1].
- . . .
- Trocar I[i] com o mínimo de I[i], . . ., I[n - 1].

Algoritmo: SelectionSort(I)

```
1 n ← tamanho de I
2 para i = 0 até n − 2
3     min ← i
4     para j = i + 1 até n − 1
5         se I[j] < I[min]
6             min ← j
7     Troque I[i] com I[min]
```





SelectionSort funciona?

Algoritmo: SelectionSort(l)

```
1 n ← tamanho de l
2 para i = 0 até n − 2
3     min ← i
4     para j = i + 1 até n − 1
5         se l[j] < l[min]
6             min ← j
7     Troque l[i] com l[min]
```

Os passos de SelectionSort são simples e ele sempre termina.
Suponha que no início da iteração (linha 3) **l[0], ..., l[i - 1]** estão ordenados e são os menores elementos de **l**:

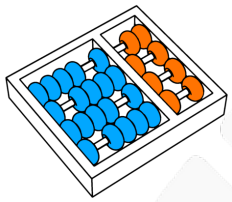
- Isso é verdade no início da primeira iteração.

No final da iteração, **l[0], ..., l[i]** estão ordenados e são os menores elementos da lista?

- Sim, pois pegamos o menor valor de **l[i], l[i + 1], ..., l[n - 1]** e trocamos com **l[i]**.

Ou seja, no final da última iteração, a lista está ordenada.

BubbleSort

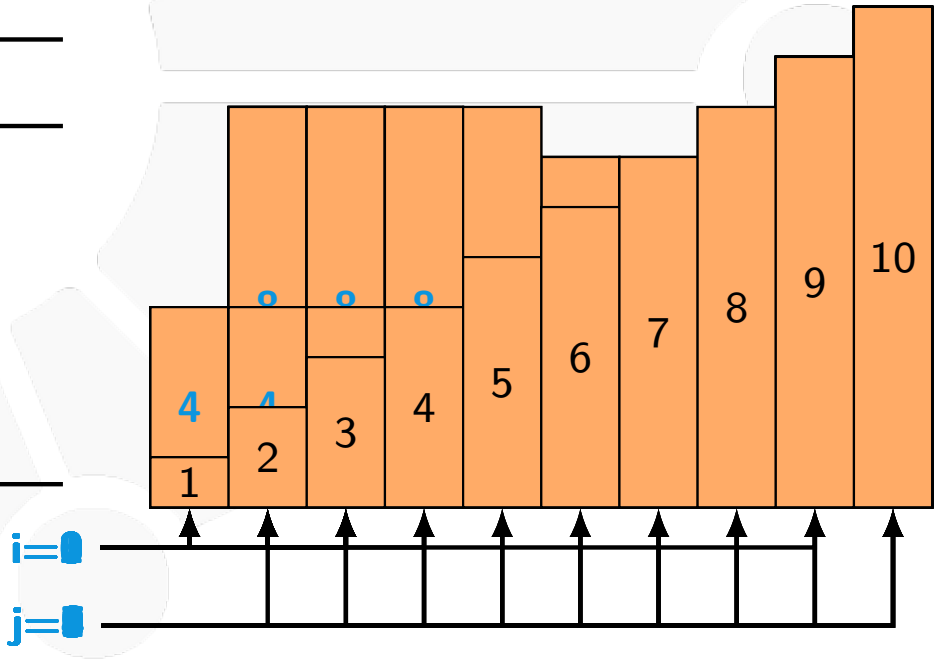


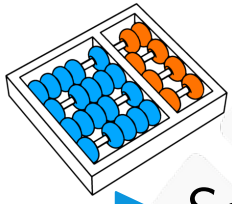
BubbleSort

- Se l não está ordenada, existe j com l[j - 1] > l[j].
- Então, do fim para o começo, trocamos pares invertidos.
- Porém, apenas uma passada pode ser insuficiente. . .
- Após a primeira passada, o menor elemento está em l[0].
- Após a segunda, o segundo menor elemento está em l[1].
- E assim por diante. . . realizando n - 1 passadas.

Algoritmo: BubbleSort(l)

```
1 n ← tamanho de l
2 para i = 0 até n − 2
3     para j = n − 1 até i + 1
4         se l[j] < l[j − 1]
5             Troque l[j] com l[j − 1]
```



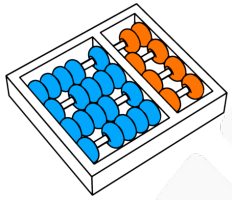


Otimização

- Se fizermos uma passada e não houver trocas, a lista já está ordenada:
  - Não havia posição `j` com `l[j - 1] > l[j]`.

Algoritmo: BubbleSort(l)

```
1 n ← tamanho de l
2 para i = 0 até n − 2
3     trocou ← Falso
4     para j = n − 1 até i + 1
5         se l[j] < l[j − 1]
6             Troque l[j] com l[j − 1]
7             trocou ← Verdadeiro
8     se trocou e Falso
9         pare
```



BubbleSort funciona?

Algoritmo: BubbleSort(l)

```
1 n ← tamanho de l
2 para i = 0 até n − 2
3     para j = n − 1 até i + 1
4         se l[j] < l[j − 1]
5             Troque l[j] com l[j − 1]
```

Claramente os passos de BubbleSort são simples e ele termina.
Suponha que no início da iteração (linha 3) **l[0], ..., l[i - 1]** estão ordenados e são os menores elementos de **l**:

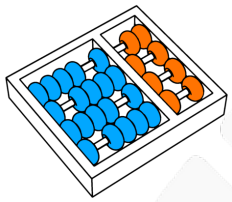
- Isso é verdade no início da primeira iteração

No final da iteração **i**, **l[0], ..., l[i]** estão ordenados e são os menores elementos da lista?

- Sim, pois o menor valor de **l[i], ..., l[n - 1]** será deslocado até **l[i]** através de trocas.

Ou seja, no final da última iteração, a lista está ordenada.

InsertionSort

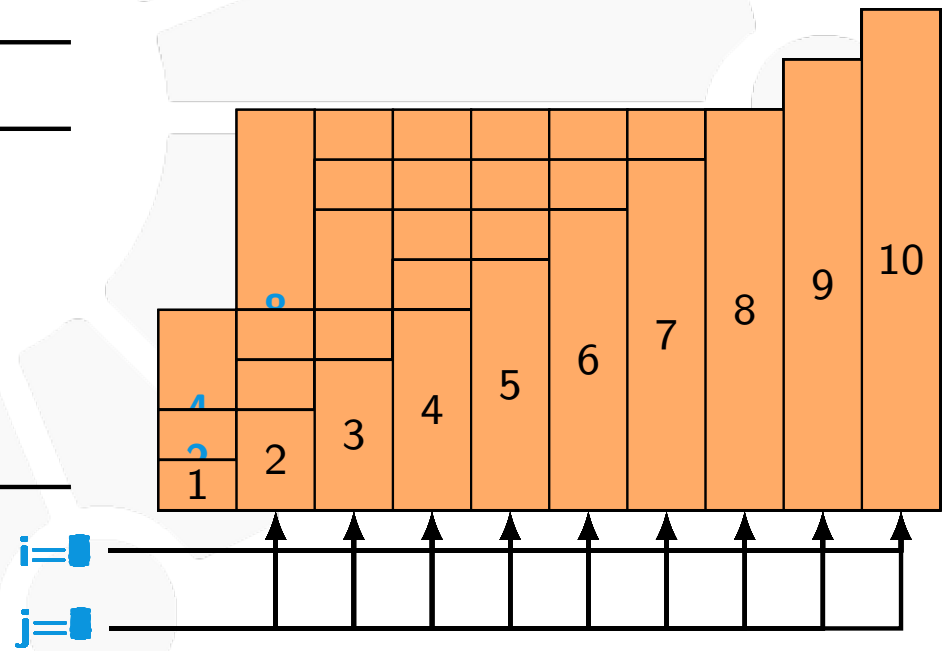


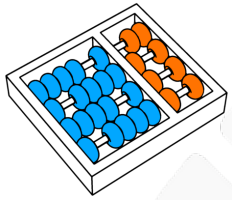
Ordenação por Inserção

- Se já temos **I[0], I[1], . . ., I[i-1]** ordenados.
- Inserimos **I[i]** na posição correta:
  - Fazemos algo similar ao BubbleSort.
- Ficamos com **I[0], I[1], . . ., I[i]** ordenados.

Algoritmo: InsertionSort(I)

```
1 n ← tamanho de I
2 para i = 1 até n − 1
3     para j = i até 1
4         se I[j] < I[j − 1]
5             Troque I[j] com I[j − 1]
```



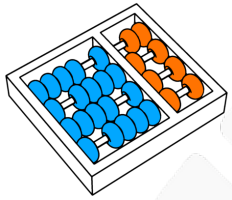


Otimização

- Não é necessário trocar sempre até o começo da lista:
  - Podemos parar quando o elemento está na posição correta.
- Não é necessário fazer trocas:
  - Podemos ir deslocando os elementos para a direita.
  - Abrindo o espaço para o elemento a ser inserido.

Algoritmo: InsertionSort(l)

```
1 n ← tamanho de l
2 para i = 1 até n − 1
3     aux ← l[i]
4     j ← i
5     enquanto j > 0 e l[j − 1] > aux
6         l[j] ← l[j − 1]
7         j ← j − 1
8     l[j] ← aux
```



InsertionSort funciona?

Algoritmo: InsertionSort(l)

```
1 n ← tamanho de l
2 para i = 1 até n − 1
3     aux ← l[i]
4     j ← i
5     enquanto j > 0 e l[j − 1] > aux
6         l[j] ← l[j − 1]
7         j ← j − 1
8     l[j] ← aux
```

Claramente os passos são simples e ele termina
Suponha que no início da iteração da linha 3 **l[0], ..., l[i - 1]** estão ordenados:

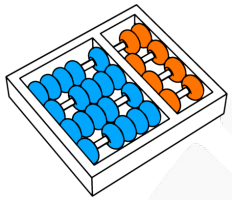
- Isso é verdade no início da primeira iteração.
  - **l[0]** sozinho está ordenado.

No final da iteração **i**, **l[0], ..., l[i]** estão ordenados?

- Sim, pois **l[i]** é inserido de forma a manter a ordenação.

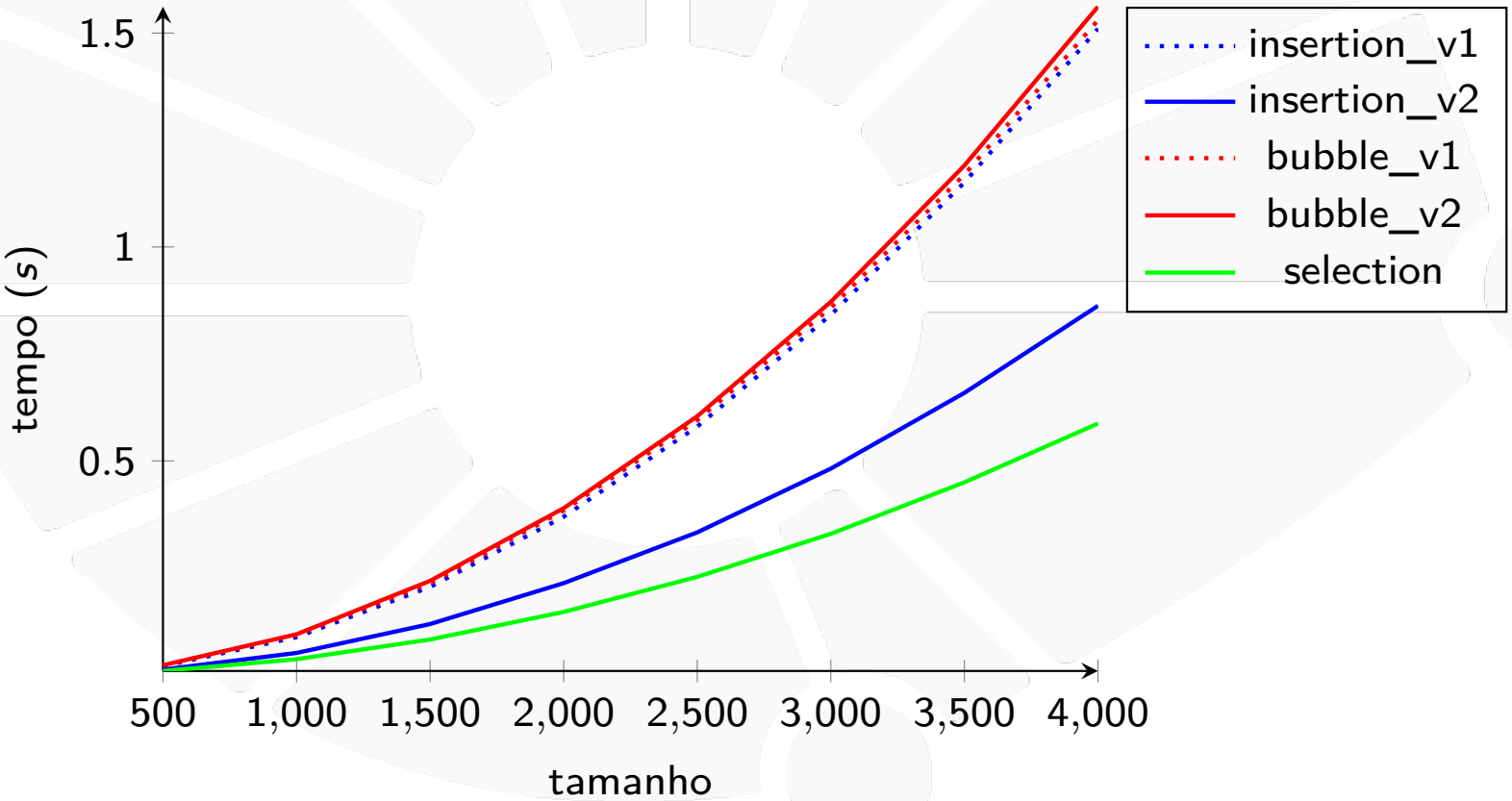
Ou seja, no final da última iteração, a lista está ordenada.

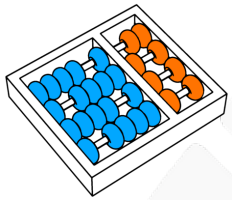
Experimentos



Experimento

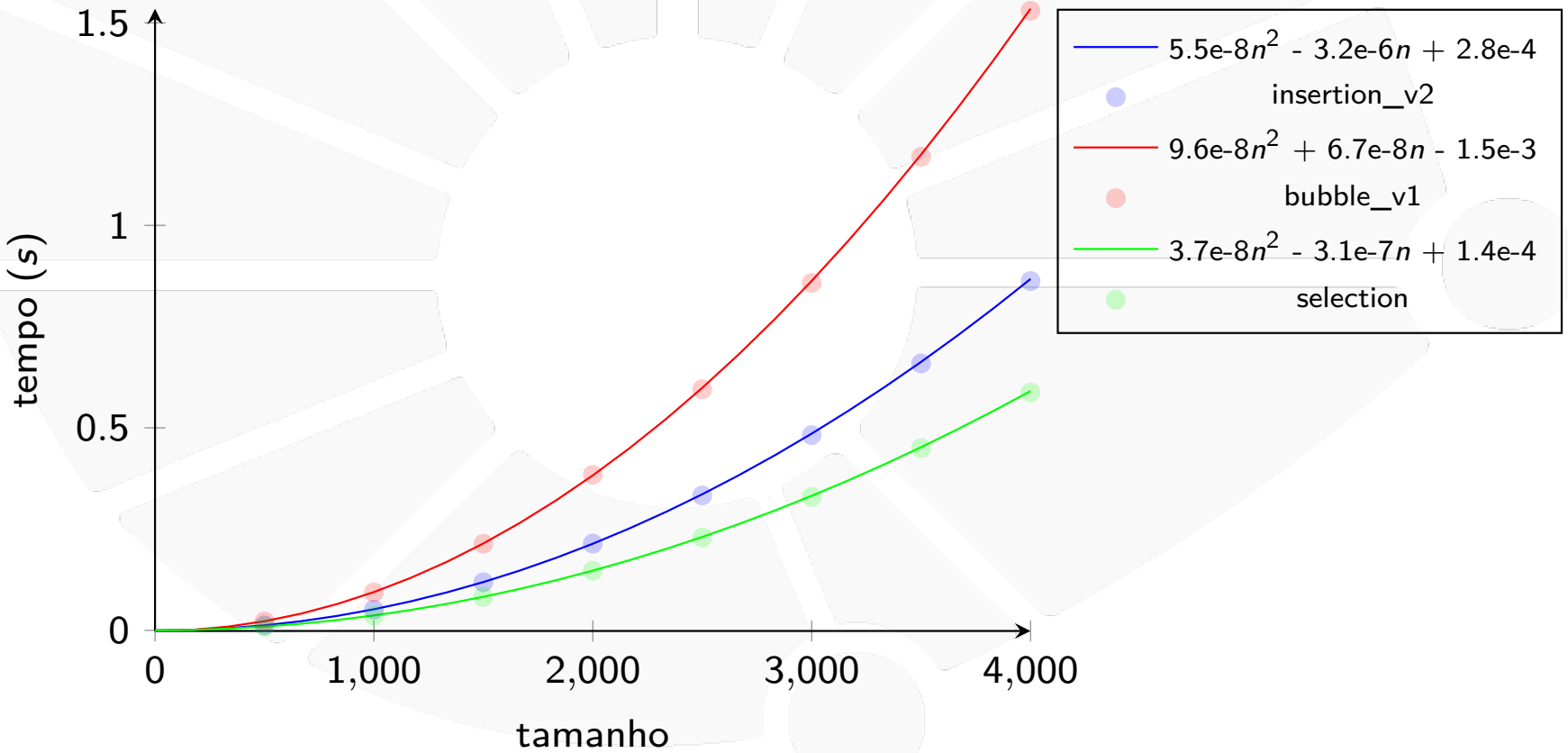
- Listas de tamanho 500, 1000, . . . , 4000.
- Cada elemento escolhido aleatoriamente entre 0 e 1.
- Tiramos a média do tempo de 10 execuções.



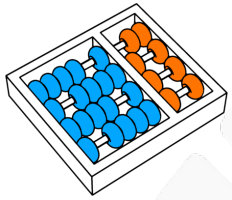


Experimento

- Listas de tamanho 500, 1000, . . . , 4000.
- Cada elemento escolhido aleatoriamente entre 0 e 1.
- Tiramos a média do tempo de 10 execuções.

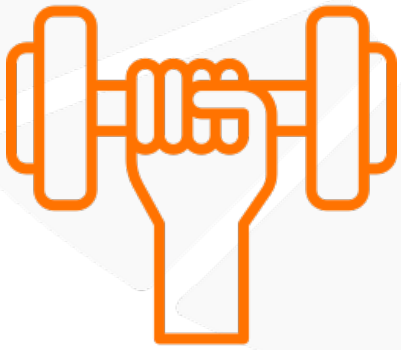


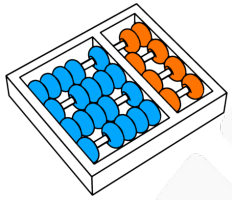
Exercícios



Sobre os algoritmos

Vamos fazer alguns exercícios?





Exercícios

1. Implemente o SelectionSort em Python.
2. Implemente as duas versões (otimizada e não otimizada) do BubbleSort em Python.
3. Implemente as duas versões (otimizada e não otimizada) do InsertionSort em Python.
4. Realize experimentos computacioneis com as diferentes implementações.

Ordenação

MC102 - Algoritmos e Programação de Computadores

05/24

21

Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP