MC102 — Algoritmos, Busca e Ordenação

Rafael C. S. Schouery
rafael@ic.unicamp.br

Universidade Estadual de Campinas

Atualizado em: 2023-06-12 17:38

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 l de números, encontrar o índice do menor valor que aparece em l

- Instância: $l = [7, 1, 3, -2, 9, -2]$
- 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

$$\begin{aligned} a_{11}x_1 + a_{12}x_2 &= b_1 \\ a_{21}x_1 + a_{22}x_2 &= b_2 \end{aligned}$$

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

$$\begin{aligned} 5x_1 + 20x_2 &= 5 \\ x_1 + 2x_2 &= 3 \end{aligned}$$

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

Sistema de Equações Lineares: Dados $A \in \mathbb{R}^{n\times m}$ e $b \in \mathbb{R}^n$, encontrar $x \in \mathbb{R}^m$ 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 lab era um problema computacional.

- Que podiam ser quebrados em subproblemas

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

Mínimo

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

Ideia do Algoritmo:

- Chutar que 0 é o índice do mínimo
- Percorrer l do início para o fim
  - Se eu encontrar um valor menor no índice atual
  - Eu atualizo qual é o meu chute

```
Minimo(l)
    Seja n o tamanho de l
    min = 0
    Para i = 1 até n - 1
        Se l[min] > l[i]
            min = i
    Devolva min
```

Mínimo

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

```
Minimo(l)
    Seja n o tamanho de l
    min = 0
    Para i = 1 até n - 1
        Se l[min] > l[i]
            min = i
    Devolva min
```

É um algoritmo para Mínimo?

- Seus passos são simples o suficiente
- Ele claramente sempre termina
- Ele dá a resposta correta?
  - Precisamos de ferramental matemático para provar isso...
  - Mas vamos ver a ideia

Mínimo

```
Minimo(l)
    Seja n o tamanho de l
    min = 0
    Para i = 1 até n - 1
        Se l[min] > l[i]
            min = i
    Devolva min
```

Suponha que, no começo de uma iteração, `min` é o índice do menor valor de `l[0], ..., l[i - 1]`

- É verdade que, ao final da iteração `i`, `min` é o índice do menor valor de `l[0], ..., l[i - 1], l[i]`?
  - Se `l[min] <= l[i]`, `min` já é o índice do menor valor
  - Caso contrário, `min` passa a ser o índice do menor valor

Note que:

- No começo da iteração `1`, `min` é o índice do menor valor de `l[0], ..., l[i - 1]`
- No final da última iteração, `min` é o índice do menor valor de `l[0], ..., l[n - 1]`

Ordenação

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

3	7	1	6	5	2	4	0	8	9

$$\Downarrow$$

0	1	2	3	4	5	6	7	8	9

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 `l[1]` e `l[2]`

```
1 Se l[0] > l[1]
2     Troque l[0] com l[1]
3 Se l[0] > l[2]
4     Troque l[0] com l[2]
5 Se l[1] > l[2]
6     Troque l[1] com l[2]
```

O algoritmo resolve o problema pois:

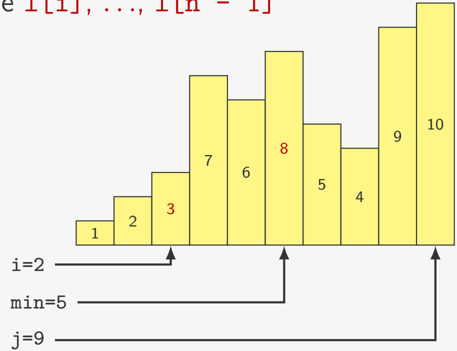
- Após a linha 2, `l[0]` tem o valor mínimo entre `l[0]` e `l[1]`
- Após a linha 4, `l[0]` tem o valor mínimo entre `l[0]`, `l[1]` e `l[2]`
- Após a linha 6, `l` está ordenada

Ordenação por Seleção

Ideia:

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

```
SelectionSort(l)
  Seja n o tamanho de l
  Para i = 0 até n - 2
    min = i
    Para j = i + 1 até n - 1
      Se l[min] > l[j]
        min = j
    Troque l[min] com l[i]
```



Exercício

Implemente o SelectionSort em Python

SelectionSort funciona?

```
1 SelectionSort(l)
2   Seja n o tamanho de l
3   Para i = 0 até n - 2
4     min = i
5     Para j = i + 1 até n - 1
6       Se l[min] > l[j]
7         min = j
8     Troque l[min] com l[i]
```

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?

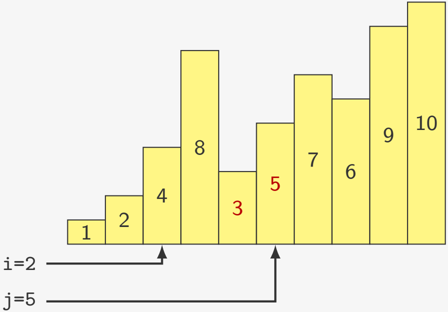
- Sim, pois pegamos o menor valor de `l[i], l[i + 1], ..., l[n - 1]` e trocamos com `l[i]`

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

BubbleSort

- 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

```
BubbleSort(l)
  Seja n o tamanho de l
  Para i = 0 até n - 2
    Para j = n - 1 até i + 1
      Se l[j - 1] > l[j]
        Troque l[j - 1] com l[j]
```



Otimização

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

```
BubbleSort(l)
  Seja n o tamanho de l
  Para i = 0 até n - 2
    trocou = Falso
    Para j = n - 1 até i + 1
      Se l[j - 1] > l[j]
        Troque l[j - 1] com l[j]
        trocou = Verdadeiro
    Se trocou é Falso
        Pare
```

Exercício

Implemente o BubbleSort em Python

BubbleSort funciona?

```
1 BubbleSort(l)
2   Seja n o tamanho de l
3   Para i = 0 até n - 2
4     Para j = n - 1 até i + 1
5       Se l[j - 1] > l[j]
6         Troque l[j - 1] com l[j]
```

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?

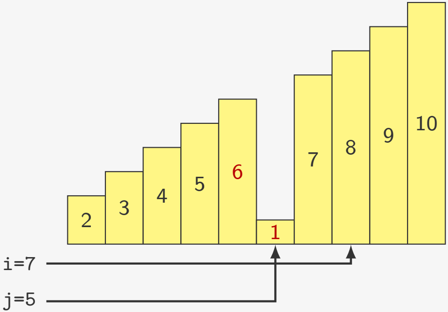
- Sim, pois o menor valor de `l[i], ..., l[n - 1]` será deslocado até `l[i]` através de trocas

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

Ordenação por Inserção

- Se já temos `l[0]`, `l[1]`, ..., `l[i-1]` ordenados
- Inserimos `l[i]` na posição correta
  - fazemos algo similar ao BubbleSort
- Ficamos com `l[0]`, `l[1]`, ..., `l[i]` ordenados

```
InsertionSort(l)
  Seja n o tamanho de l
  Para i = 1 até n - 1
    Para j = i até 1
      Se l[j - 1] > l[j]
        Troque l[j - 1] com l[j]
```



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

```
InsertionSort(l)
  Seja n o tamanho de l
  Para i = 1 até n - 1
    aux = l[i]
    j = i
    Enquanto j > 0 e l[j - 1] > aux
        l[j] = l[j - 1]
        j = j - 1
    l[j] = aux
```

Exercício

Implemente o InsertionSort em Python

InsertionSort funciona?

```
1 InsertionSort(l)
2   Seja n o tamanho de l
3   Para i = 1 até n - 1
4     aux = l[i]
5     j = i
6     Enquanto j > 0 e l[j - 1] > aux
7         l[j] = l[j - 1]
8         j = j - 1
9     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

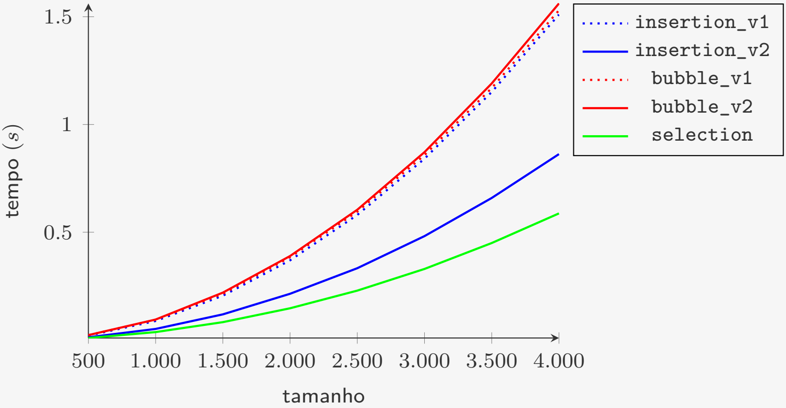
No final da iteração `i`, `l[0], ..., l[i]` estão ordenados?

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

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

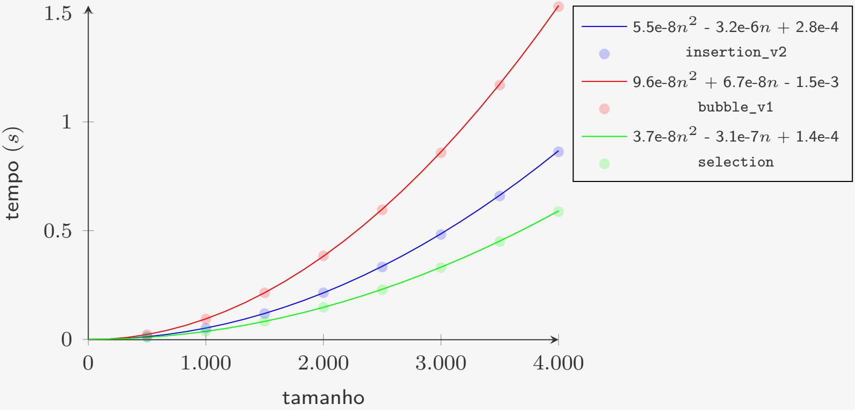
Experimento

- 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



Busca

Busca: Dada uma lista `l` de números e um número `x`, encontrar, se existir, um índice `i` tal que `l[i] = x`.

Ideia do algoritmo:

- Percorrer a lista do início para o fim, procurando `x`
- Se encontrarmos, devolvemos o índice onde `x` está
- Se não encontrarmos, então `x` não está na lista

```
BuscaSequencial(l, x)
    Seja n o tamanho de l
    Para i = 0 até n - 1
        Se l[i] = x
            Devolva i
    Devolva -1
```

Busca

Busca: Dada uma lista `l` de números e um número `x`, encontrar, se existir, um índice `i` tal que `l[i] = x`.

```
1 BuscaSequencial(l, x)
2     Seja n o tamanho de l
3     Para i = 0 até n - 1
4         Se l[i] = x
5             Devolva i
6     Devolva -1
```

`BuscaSequencial` é um algoritmo para Busca?

- Seus passos são simples o suficiente
- Ele claramente sempre termina
- Ele dá a resposta correta?
  - Se a 1ª ocorrência de `x` em `l` é a posição `k`, ele devolve `k`
  - Se `x` não está em `l`, a linha 4 sempre falha e devolvemos `-1`

Busca em uma lista ordenada

BUSCAORDENADA: Dada uma lista `l` ordenada de números e um número `x`, encontrar, se existir, um índice `i` tal que `l[i] = x`.

Já temos `BuscaSequencial`, mas ele não se aproveita do fato da lista estar ordenada...

Ideia: Digamos que, ao invés de olhar a primeira posição da lista, eu olhe uma posição `m`. O que pode acontecer?

- Eu posso descobrir que `l[m] = x`... Ótimo!
- Eu posso descobrir que `l[m] < x` ou `l[m] > x`
  - Se `l[m] < x`, então se `x` estiver na lista, está da posição `m + 1` para a frente...
  - Se `l[m] > x`, então se `x` estiver na lista, está da posição `m - 1` para trás...

Qual `m` escolher?

- `n / 2` parece bom porque "descarto" metade da lista

Posso repetir a mesma ideia para a parte não descartada

Busca Binária

```
BuscaBinaria(l, x)
  Seja n o tamanho de l
  e = 0
  d = n - 1
  Enquanto e <= d
    m = (e + d) / 2        # divisão inteira
    Se l[m] == x
        Devolva m
    Se l[m] < x            # está para a direita
        e = m + 1
    Se l[m] > x            # está para a esquerda
        d = m - 1
  Devolva -1
```

Buscando por 80:

11	22	33	44	55	66	77	88	99

Exercício

Implemente a Busca Binária em Python

Busca Binária sempre termina?

```
BuscaBinaria(l, x)
  Seja n o tamanho de l
  e = 0
  d = n - 1
  Enquanto e <= d
    m = (e + d) / 2        # divisão inteira
    Se l[m] == x
        Devolva m
    Se l[m] < x            # está para a direita
        e = m + 1
    Se l[m] > x            # está para a esquerda
        d = m - 1
  Devolva -1
```

No começo `e = 0` e `d = n - 1` e a cada passo:

- `m` é um número maior ou igual a `e`
- `m` é um número menor ou igual a `d`
- Portanto, exatamente uma das opções ocorre:
  - ou `e` aumenta
  - ou `d` diminui

Busca Binária funciona?

```
BuscaBinaria(l, x)
  Seja n o tamanho de l
  e = 0
  d = n - 1
  Enquanto e <= d
    m = (e + d) / 2        # divisão inteira
    Se l[m] == x
        Devolva m
    Se l[m] < x            # está para a direita
        e = m + 1
    Se l[m] > x            # está para a esquerda
        d = m - 1
  Devolva -1
```

Suponha que `x` esteja em `l[e], ..., l[d]` no começo da iteração, então:

- Se `l[m] == x`, o algoritmo dá a resposta correta
- Se `l[m] > x`, então `x` está em `l[e], ..., l[m - 1]`
- Se `l[m] < x`, então `x` está em `l[m + 1], ..., l[d]`

Quantas posições da lista olhamos?

Em uma busca sequencial, podemos ter que olhar todas as `n` posições da lista

- Quando `x` não está na lista

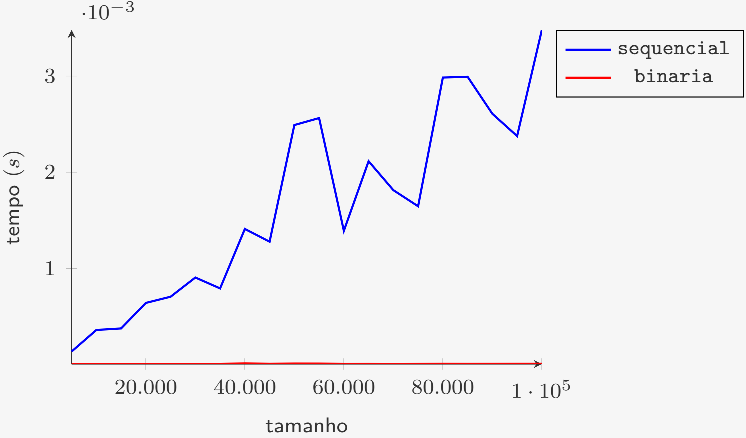
Mas e na busca binária?

Chame de $f(n)$ o maior número de posições que olhamos entre todas as instâncias onde a lista tem n elementos

- $f(1) = 1$, pois o elemento do meio é o único elemento
- $f(3) = 1 + f(1) = 2$, pois olhamos o elemento do meio e, se não encontramos, precisamos olhar uma lista de tamanho 1
- $f(7) = 1 + f(3) = 3$, pois olhamos o elemento do meio e, se não encontramos, precisamos olhar uma lista de tamanho 3
- De forma geral, $f(2^k - 1) = 1 + f(2^{k-1} - 1) = k$
- Ou seja, para $n = 2^k - 1$, olhamos k posições
- Isto é, $\log_2(n + 1)$ posições

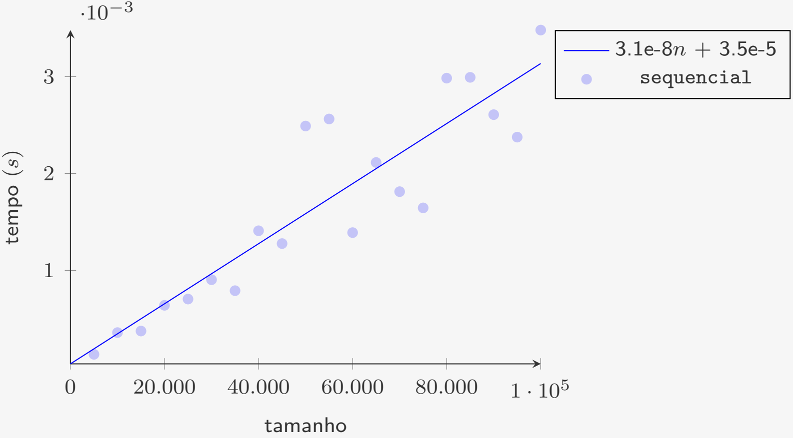
Experimento 1

- Listas de tamanho 5000, 10000, ..., 100000
- `l[i] = i`
- Tiramos a média do tempo de 20 execuções
- Escolhemos um `i` aleatório e procuramos por `l[i]` em `l`



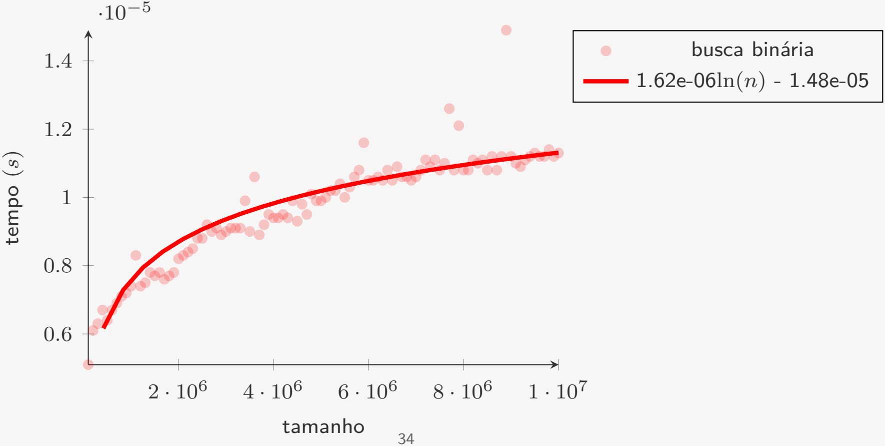
Experimento 1

- Listas de tamanho 5000, 10000, ..., 100000
- `l[i] = i`
- Tiramos a média do tempo de 20 execuções
- Escolhemos um `i` aleatório e procuramos por `l[i]` em `l`



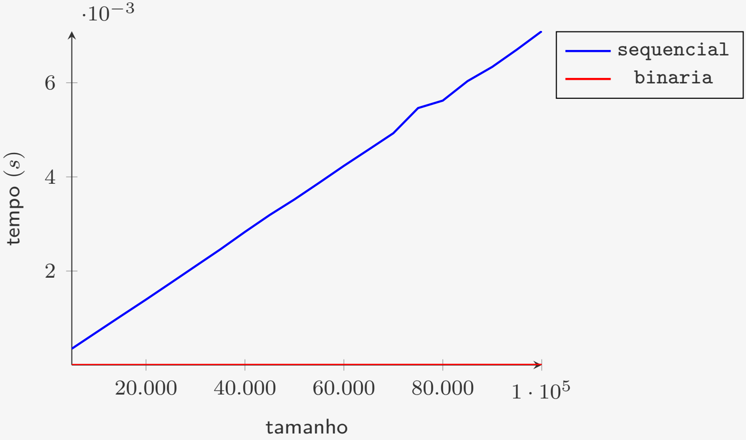
Experimento 1b — apenas busca binária

- Listas de tamanho 100000, 200000, ..., 10000000
- `l[i] = i`
- Tiramos a média do tempo de 100 execuções
- Escolhemos um `i` aleatório e procuramos por `l[i]` em `l`



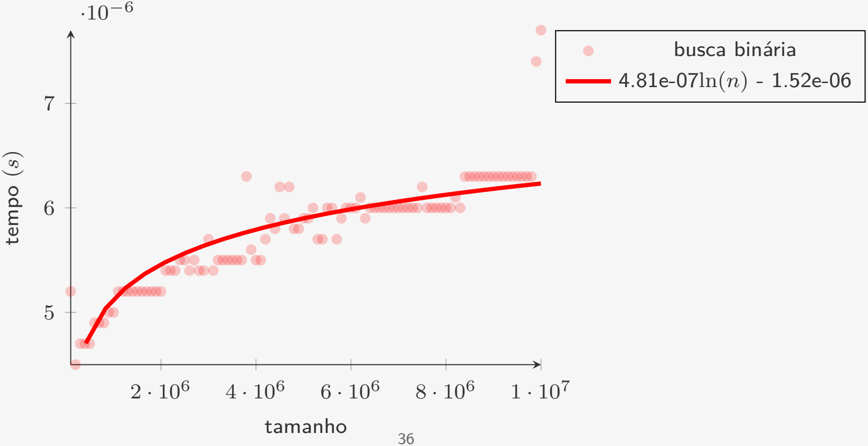
Experimento 2

- Listas de tamanho 5000, 10000, ..., 100000
- `l[i] = i`
- Tiramos a média do tempo de 20 execuções
- Buscamos por `len(l)` (que não está na lista)



Experimento 2b — apenas busca binária

- Listas de tamanho 100000, 200000, ..., 10000000
- `l[i] = i`
- Tiramos a média do tempo de 100 execuções
- Buscamos por `len(l)` (que não está na lista)



Exercícios

1. Faça um algoritmo que dado uma lista ordenada e um elemento indica onde inserir esse elemento na lista para mantê-la ordenada...
   - Se o elemento já estiver na lista, você deve inseri-lo de forma a ser o primeiro do bloco repetido.
2. Use o algoritmo anterior para implementar uma busca em uma lista ordenada
3. Repita o primeiro exercício mas de forma que o elemento é o último do bloco repetido.
4. Faça um algoritmo que dado uma lista ordenada e um elemento, encontra o intervalo que contém todos os elementos iguais a ele na lista.