Busca

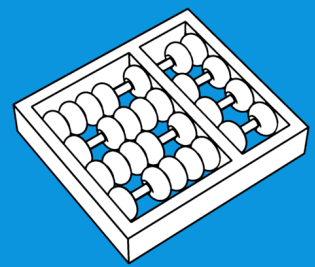
MC102 - Algoritmos e Programação de Computadores

06/25

22

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

UNICAMP




BINARY SEARCH
PROGRAMMER
Linear Search

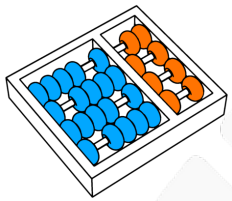
Maratona dos Bixes!!!

No dia 06/06 das 14 às 17h

Valendo nota extra na disciplina pelo seu desempenho ;-)

Inscrições: `https://docs.google.com/forms/d/e/1FAIpQLSfld1b2GEv17AAYCAqy09_mHARPZ9qzhCkJO3rxWy6qsKDcGw/viewform?usp=dialog`

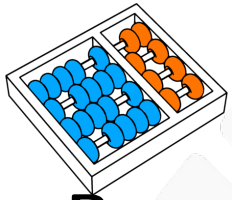
Dúvidas da aula anterior



Dúvidas selecionadas

- Assim como podemos dizer que um caractere é menor que o outro, como “a”<“e” por conta do encoding, podemos dizer o mesmo de uma combinação de caracteres, como “ae”<“ao”?
- Você mencionou que python usa dois sorts. Significa que ele analisa qual é melhor pra situação?
- Não entendi pq utilizar outro algoritmo se o selection é o com menor tempo.
- Qual a complexidade desses algoritmos mostrados em aula?
- Você decorou a tabela ascii?
- Nós vimos a comparação em tempo dos algoritmos de ordenação que o senhor mostrou, mas como seria o uso de memória de ambos?
- Porque os desenvolvedores do python (atualmente) não alteram funções menos eficientes como o sort deles para versões mais eficientes em novas versões da linguagem?
- Tem alguma regra geral pra gente saber qual algoritmo é o mais eficiente?
- Não daria pra melhorar o insert só inserindo na posição correta em vez de fazer várias trocas?
- Existem quantos tipos de algoritmos de organização?
- O tipo de ordenação utilizada tem algum impacto significativo na eficiência do código?
- Pode explicar o sort do tipo bubble novamente?
- Como escolher entre os algoritmos de ordenação apresentados para resolver um problema?

Busca



Busca

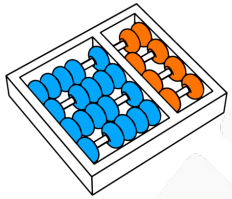
Busca: Dada uma lista **l** de números e um número **x**, encontrar, se existir, um índice **i** tal que **l[i] = x**.

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.

Algoritmo: BuscaSequencial(l, x)

```
1 n ← tamanho de l
2 para i = 0 até n − 1
3     se l[i] = x
4         devolva i
5 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**.

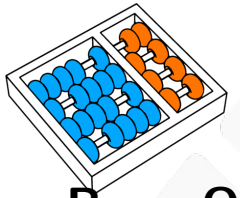
Algoritmo: BuscaSequencial(l, x)

```
1 n ← tamanho de l
2 para i = 0 até n − 1
3     se l[i] = x
4         devolva i
5 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 lista ordenada



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?

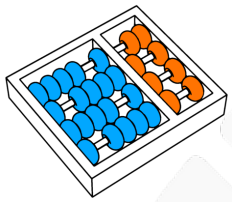
- Eu posso descobrir que l[m] = x... Ótimo!
- Eu posso descobrir que l[m] < x ou l[m] > x.
  - Se l[m] < x, então se x estiver na lista, está da posição m + 1 para a frente...
  - Se l[m] > x, então se x estiver na lista, está da posição m - 1 para trás...

Qual m escolher?

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

Posso repetir a mesma ideia para a parte não descartada.

Busca binária

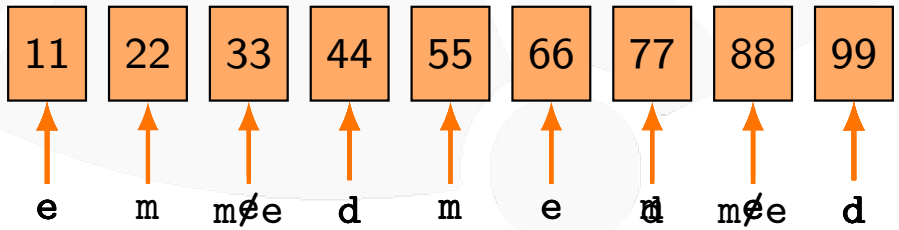


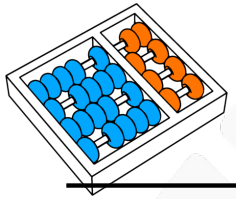
Busca Binária

Algoritmo: BuscaBinária(l, x)

```
1  n ← tamanho de l, e ← 0, d ← n − 1
2  enquanto e ≤ d
3      m ← (e + d)/2                          ▷ divisão inteira
4      se l[m] = x
5          devolva m
6      se l[m] < x
7          e ← m + 1                          ▷ está para a direita
8      se l[m] > x
9          d ← m − 1                          ▷ está para a esquerda
10 devolva −1
```

Buscando por 33: Buscando por 80:





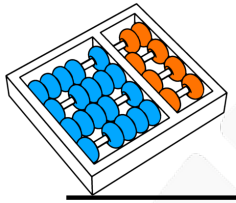
Busca Binária sempre termina?

Algoritmo: BuscaBinária(l, x)

```
1  n ← tamanho de l, e ← 0, d ← n − 1
2  enquanto e ≤ d
3      m ← (e + d)/2                    ▷ divisão inteira
4      se l[m] = x
5          devolva m
6      se l[m] < x
7          e ← m + 1                    ▷ está para a direita
8      se l[m] > x
9          d ← m − 1                    ▷ está para a esquerda
10 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:
  - **e** aumenta, ou
  - **d** diminui.



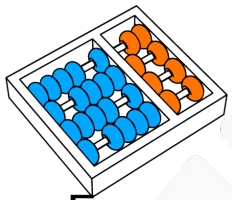
Busca Binária funciona?

Algoritmo: BuscaBinária(l, x)

```
1  n ← tamanho de l, e ← 0, d ← n − 1
2  enquanto e ≤ d
3      m ← (e + d)/2                    ▷ divisão inteira
4      se l[m] = x
5          devolva m
6      se l[m] < x
7          e ← m + 1                    ▷ está para a direita
8      se l[m] > x
9          d ← m − 1                    ▷ está para a esquerda
10 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:

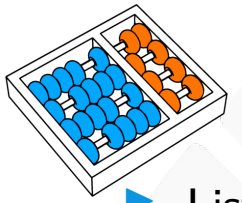
- Quando **x** não está na lista.

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:

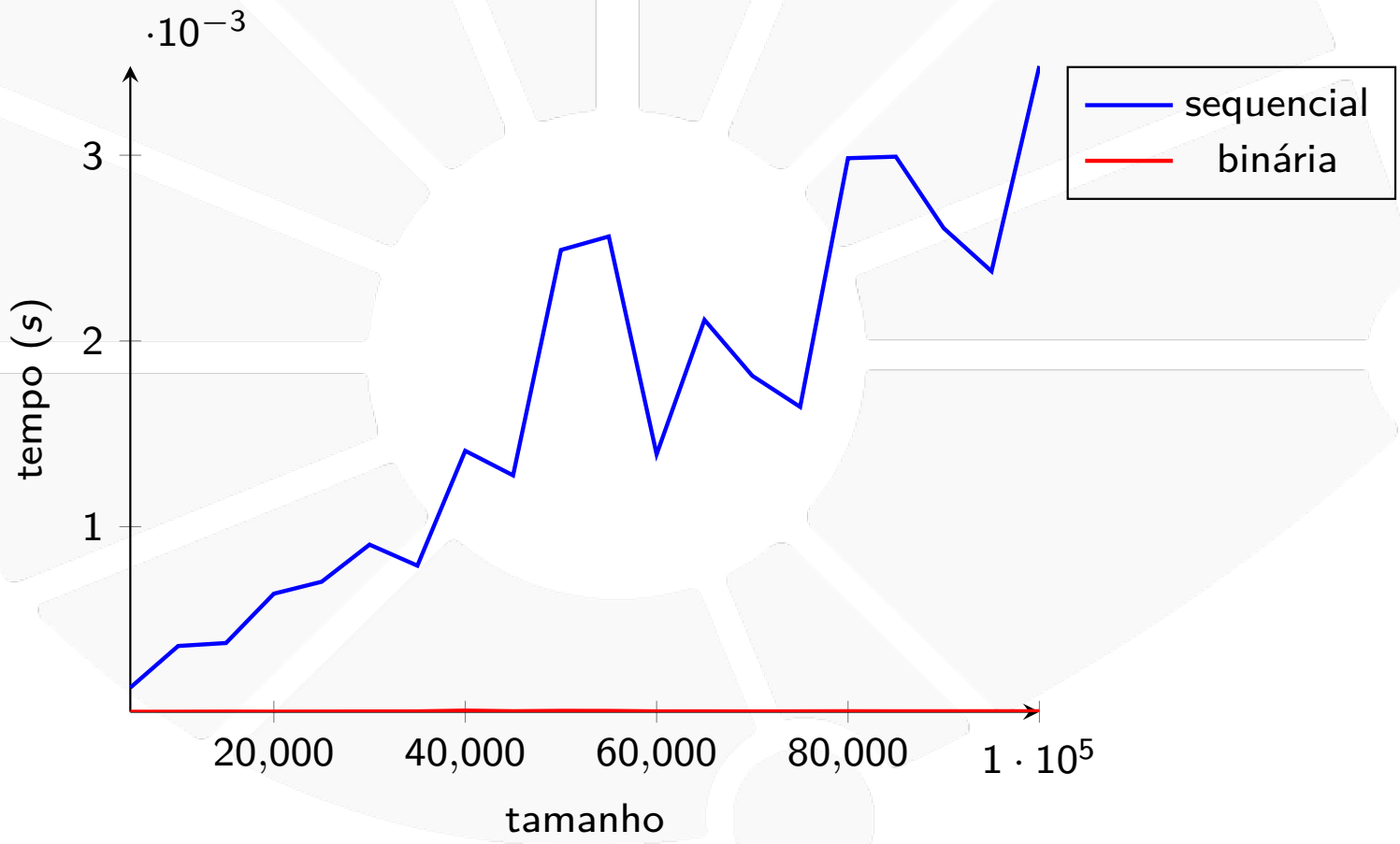
- $f(1) = 1$, pois o elemento do meio é o único elemento.
- $f(3) = 1 + f(1) = 2$, pois olhamos o elemento do meio e, se não encontramos, precisamos olhar uma lista de tamanho 1.
- $f(7) = 1 + f(3) = 3$, pois olhamos o elemento do meio e, se não encontramos, precisamos olhar uma lista de tamanho 3.
- 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.

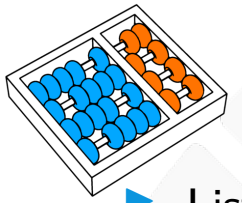
Experimentos



Experimento 1

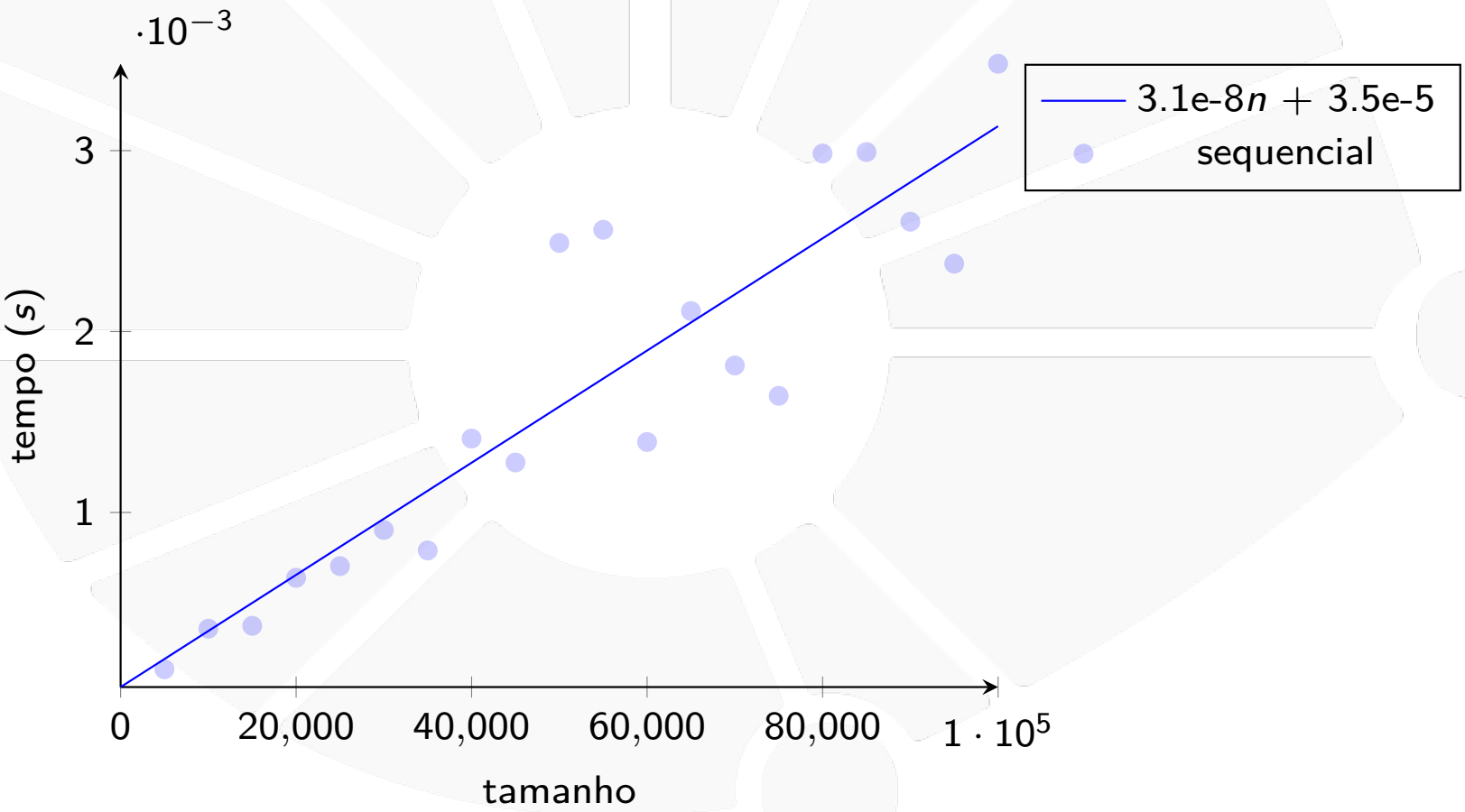
- Listas de tamanho 5000, 10000, . . . , 100000, com `l[i] = i`.
- Tiramos a média do tempo de 20 execuções.
- Escolhemos um `i` aleatório e procuramos por `l[i]` em `l`.

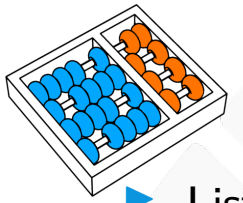




Experimento 1

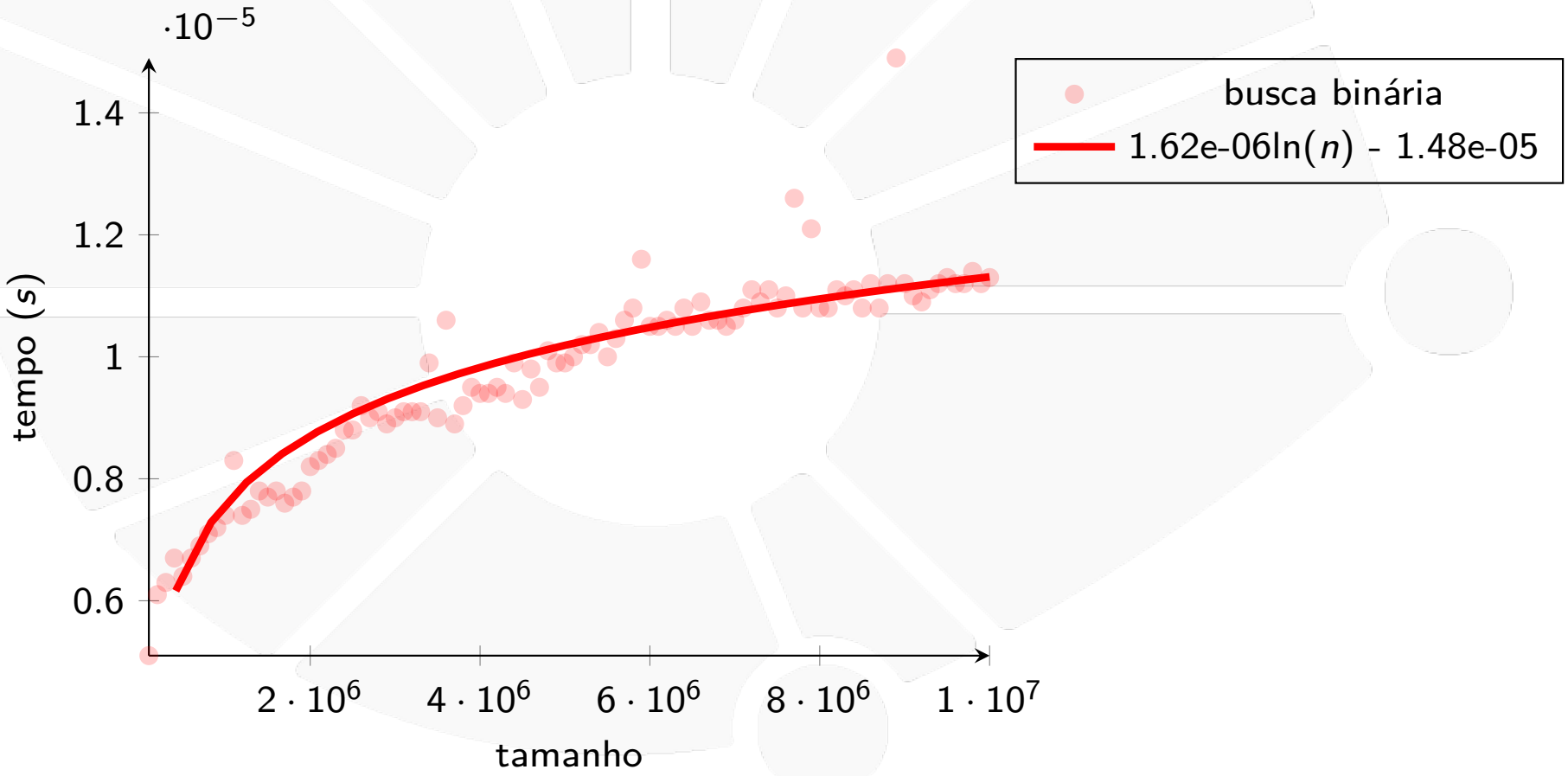
- Listas de tamanho 5000, 10000, . . . , 100000, com `l[i] = i`.
- Tiramos a média do tempo de 20 execuções.
- Escolhemos um `i` aleatório e procuramos por `l[i]` em `l`.

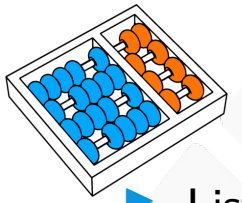




Experimento 1b — apenas busca binária

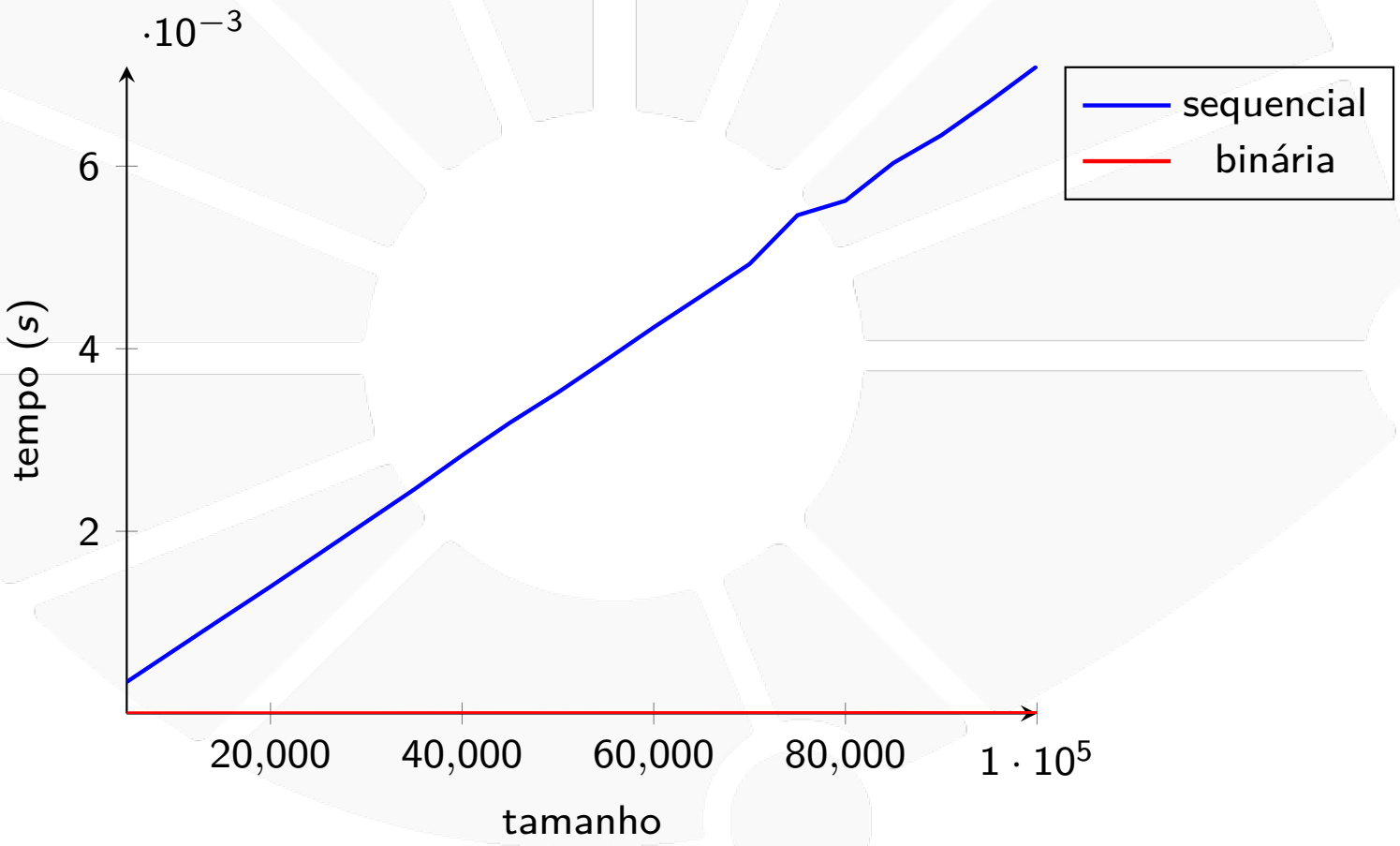
- Listas de tamanho 100000, 200000, . . . , 10000000, com `l[i] = i`.
- Tiramos a média do tempo de 100 execuções.
- Escolhemos um `i` aleatório e procuramos por `l[i]` em `l`.

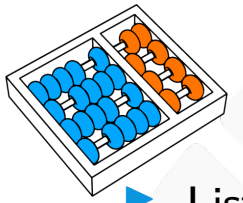




Experimento 2

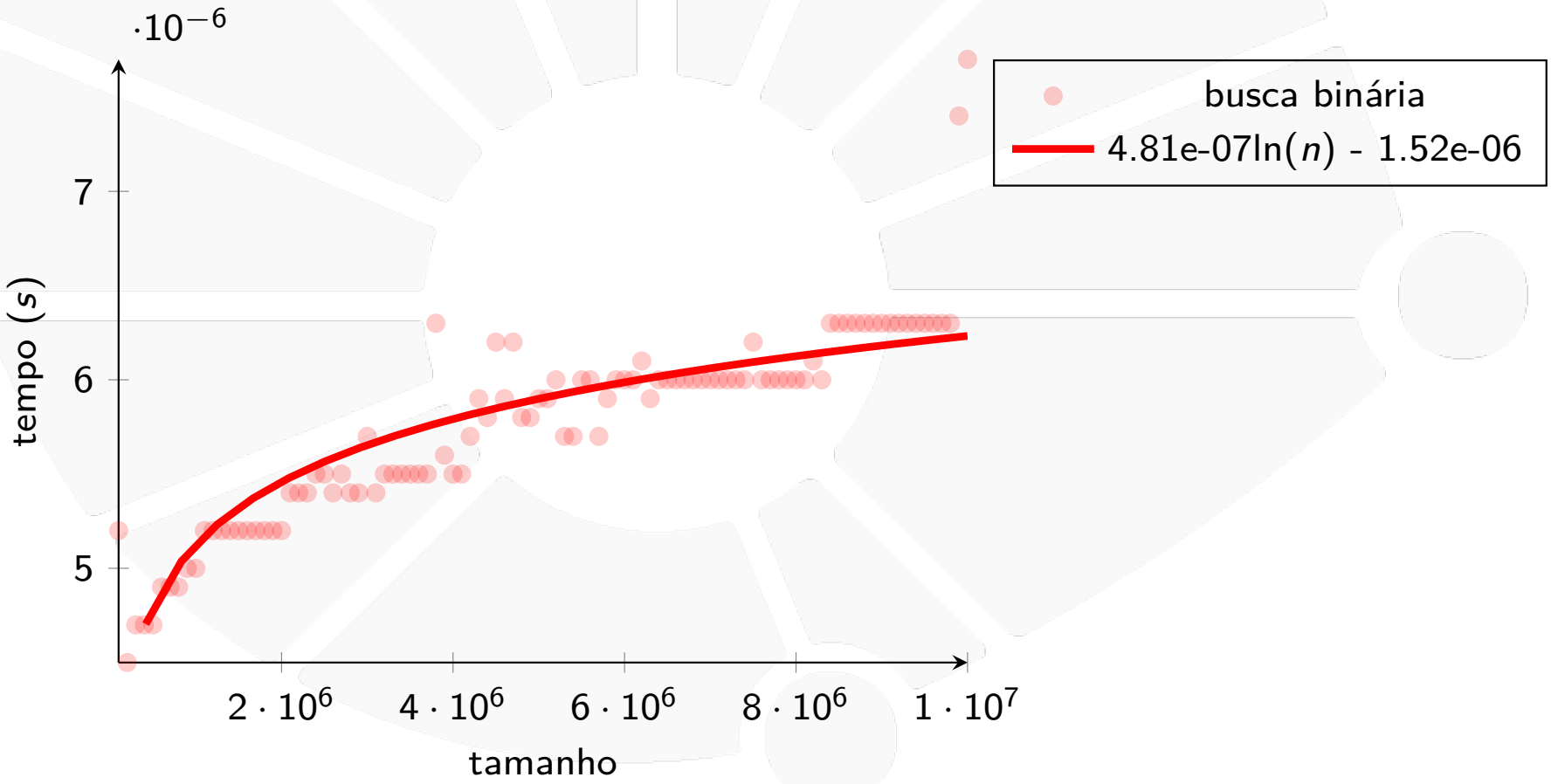
- Listas de tamanho 5000, 10000, . . . , 100000, com **l[i] = i**.
- Tiramos a média do tempo de 20 execuções.
- Buscamos por **len(l)** (que não está na lista).



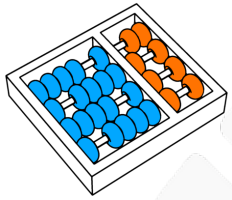


Experimento 2b — apenas busca binária

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

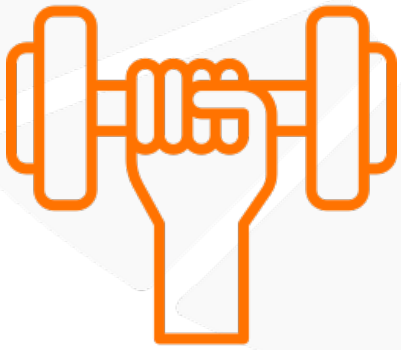


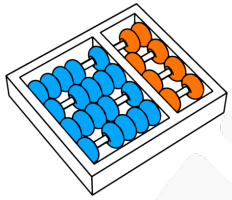
Exercícios



Sobre os algoritmos

Vamos fazer alguns exercícios?





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.

Busca

MC102 - Algoritmos e Programação de Computadores

06/25

22

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

UNICAMP