Busca

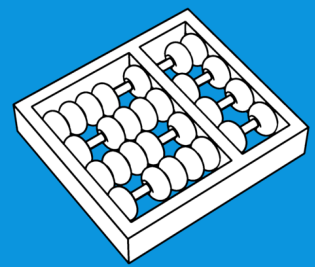
MC102 - Algoritmos e Programação de Computadores

05/24

20

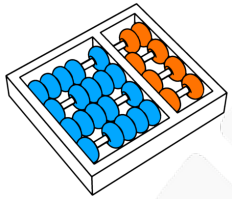
Santiago Valdés Ravelo
https://ic.unicamp.br/~santiago/
ravelo@unicamp.br

UNICAMP




BINARY SEARCH
PROGRAMMER
Linear Search

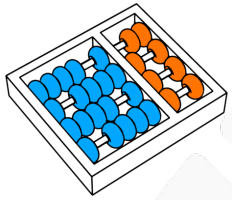
Dúvidas da aula anterior



Dúvidas selecionadas

- Um algoritmo é mais eficiente do que o outro. Mas existem situações em que é melhor usar um do que o outro?
- O desempenho de um algoritmo - como os vistos na última aula - são analisados apenas pelo tempo de execução? Como funciona?
- Para uma pequena quantidade de elementos, há muita diferença de eficiência entre os métodos de sort?
- A ordenação selection sempre vai ser a mais eficiente?
- Qual é o melhor algoritmo de ordenação atualmente?
- Existe algum método de ordenação que não necessite percorrer a lista várias vezes pra ir testando os valores?
- Existe algum documento ou site onde posso estudar e ver mais sobre aqueles e outros algoritmos?
- Não entendi porque mesmo após de ser otimizado, o bubble-sort não teve um bom desempenho comparado aos outros.
- Para que tantos algoritmos de ordenação, se um deles é o melhor?
- Foi dito em aula que quando usamos 'sort' o compilador escolhe um tipo de algoritmo para ordenar a lista e a escolha depende do tamanho da lista. Mas me pergunto, como pode existir algoritmos melhores do que outros dependendo do tamanho da lista? Inclusive, três tigres tristes para três pratos de trigo?
- Como os algoritmos mostrados se comportam com listas não necessariamente aleatórias (concatenação de listas organizadas, poucos elementos fora do lugar, elementos pouco distantes da posição correta)?
- O método sort que a gente usa no python é o selection-sort?
- Sobre o bubblesort, por que ele acaba sendo o método com o tempo de execução mais longo?

Busca



Busca

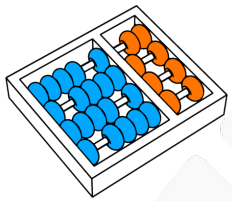
Busca: Dada uma lista **l** de números e um número **x**, encontrar, se existir, um índice **i** tal que **l[i] = x**.

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**.

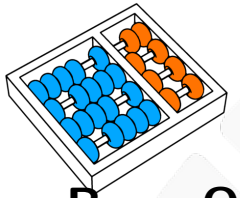
Algoritmo: BuscaSequencial(l, x)

```
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?

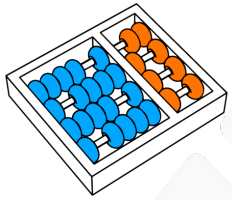
- Eu posso descobrir que l[m] = x... Ótimo!
- Eu posso descobrir que l[m] < x ou l[m] > x.
  - Se l[m] < x, então se x estiver na lista, está da posição m + 1 para a frente...
  - Se l[m] > x, então se x estiver na lista, está da posição m - 1 para trás...

Qual m escolher?

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

Posso repetir a mesma ideia para a parte não descartada.

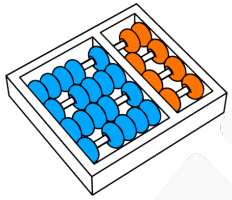
Busca binária



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
```



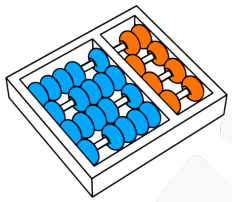
Busca Binária

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:

11	22	33	44	55	66	77	88	99

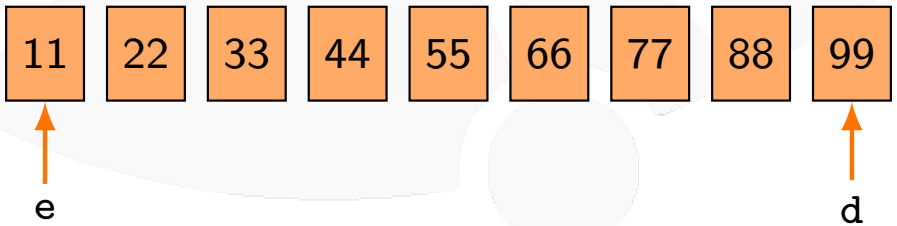


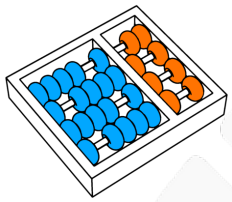
Busca Binária

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:



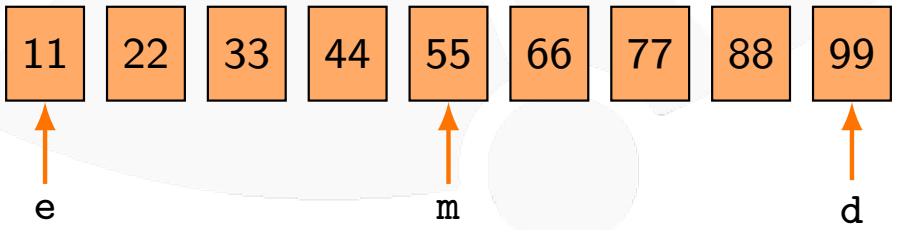


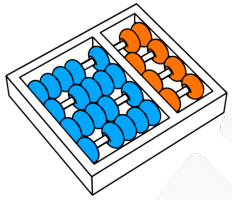
Busca Binária

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:



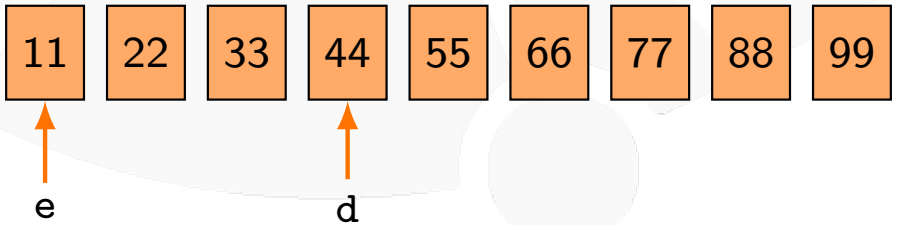


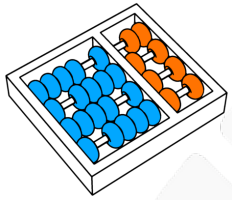
Busca Binária

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:



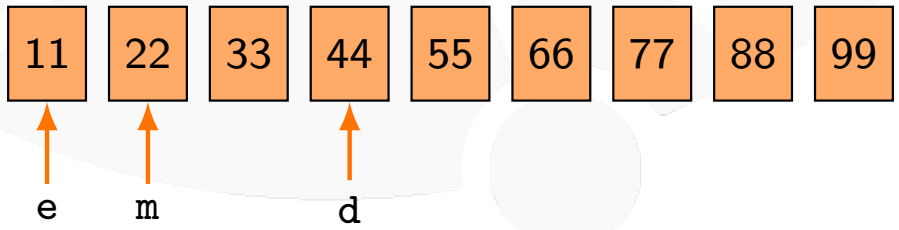


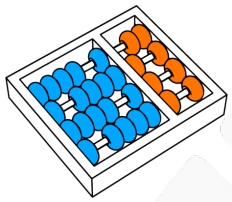
Busca Binária

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:



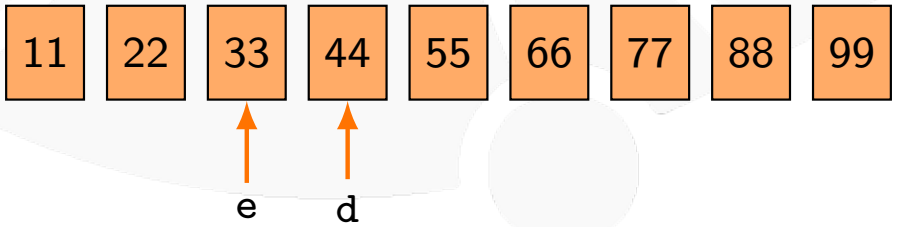


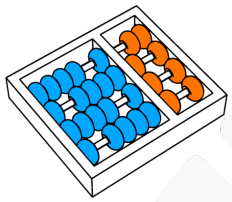
Busca Binária

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:



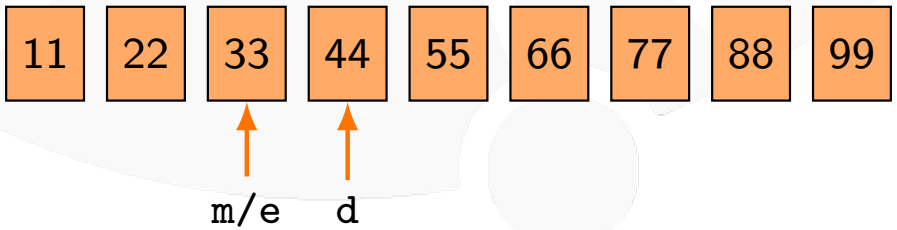


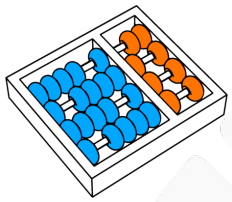
Busca Binária

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:



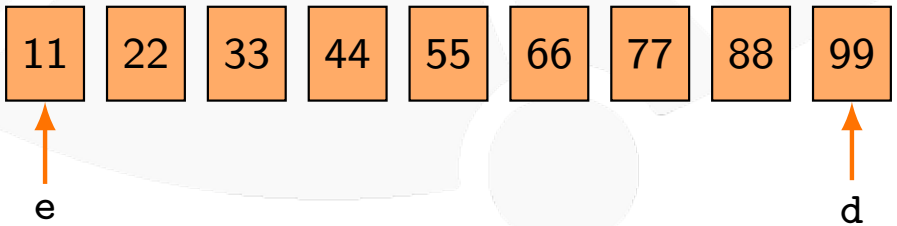


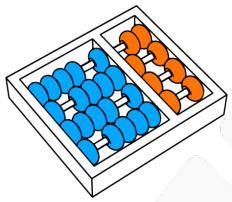
Busca Binária

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 80:



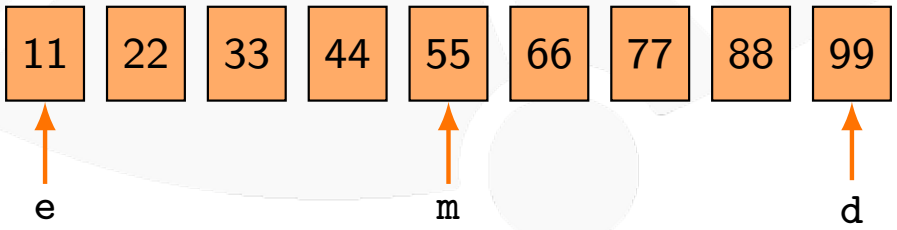


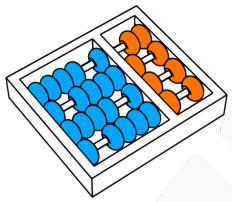
Busca Binária

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 80:



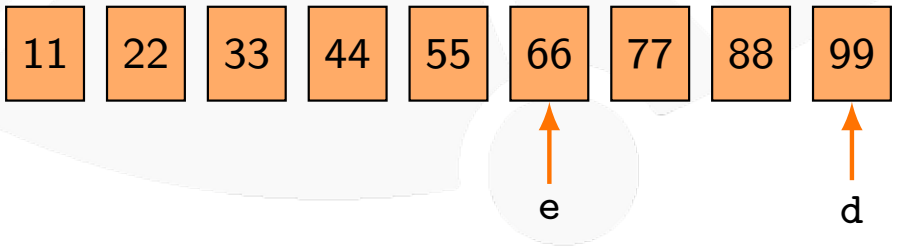


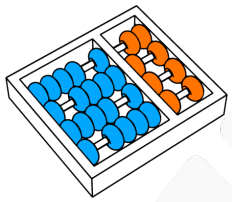
Busca Binária

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 80:



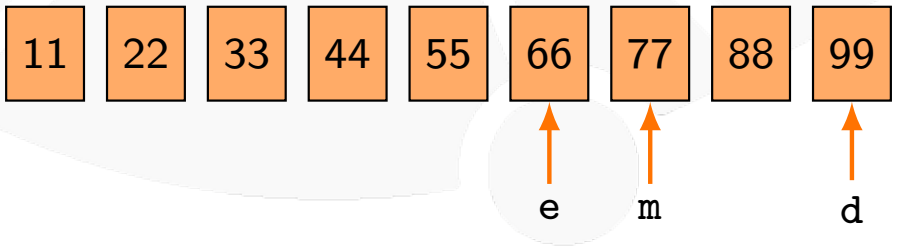


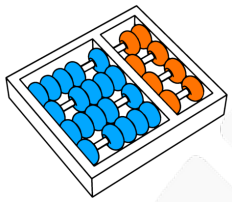
Busca Binária

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 80:



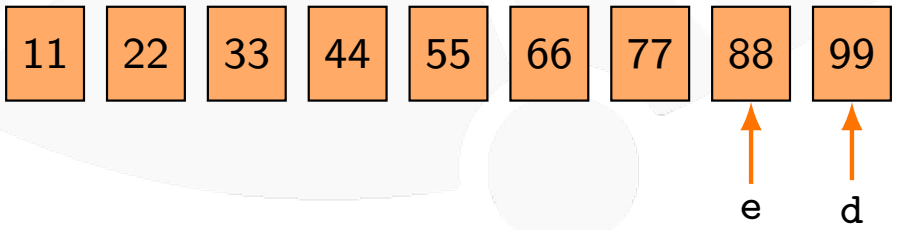


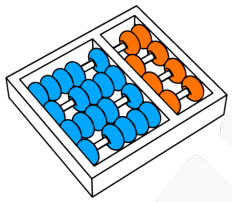
Busca Binária

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 80:



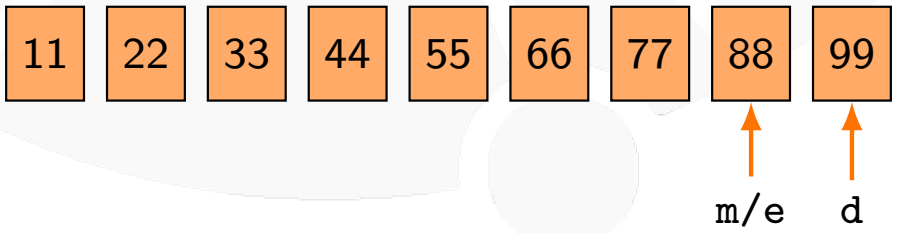


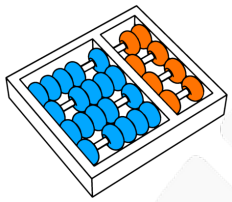
Busca Binária

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 80:



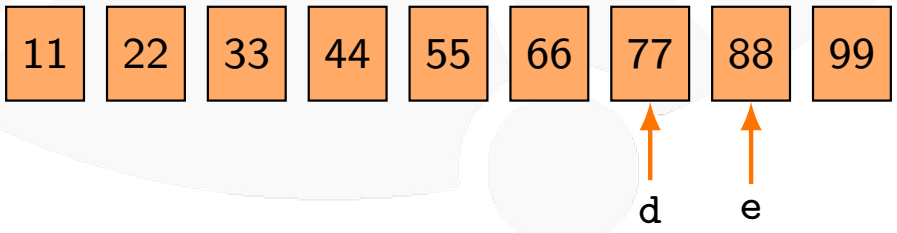


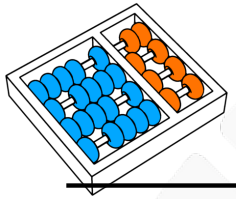
Busca Binária

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 80:

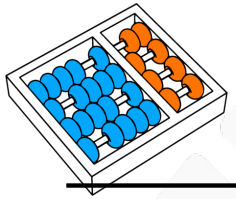




Busca Binária sempre termina?

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
```

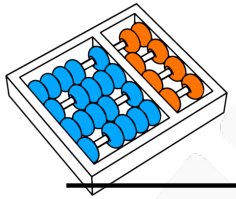


Busca Binária sempre termina?

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:



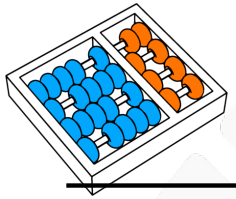
Busca Binária sempre termina?

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**.



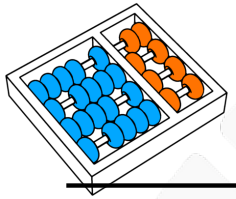
Busca Binária sempre termina?

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**.



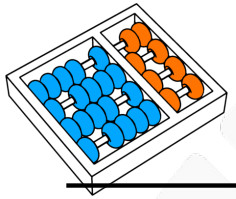
Busca Binária sempre termina?

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:



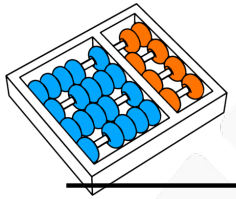
Busca Binária sempre termina?

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



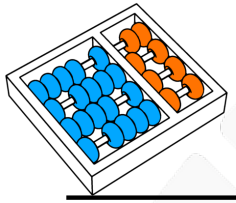
Busca Binária sempre termina?

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:

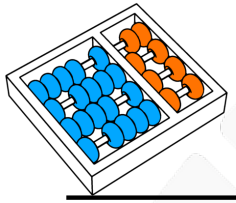
- **m** é um número maior ou igual a **e**.
- **m** é um número menor ou igual a **d**.
- Portanto, exatamente uma das opções ocorre:
  - **e** aumenta, ou
  - **d** diminui.



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
```

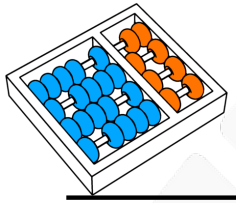


Busca Binária funciona?

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:



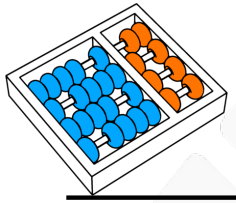
Busca Binária funciona?

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.



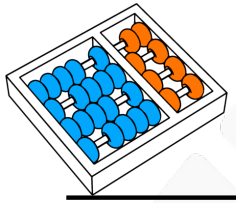
Busca Binária funciona?

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]**.



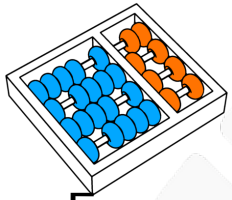
Busca Binária funciona?

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:

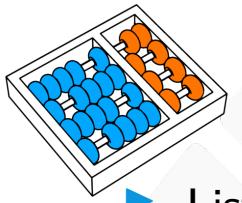
- Quando **x** não está na lista.

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:

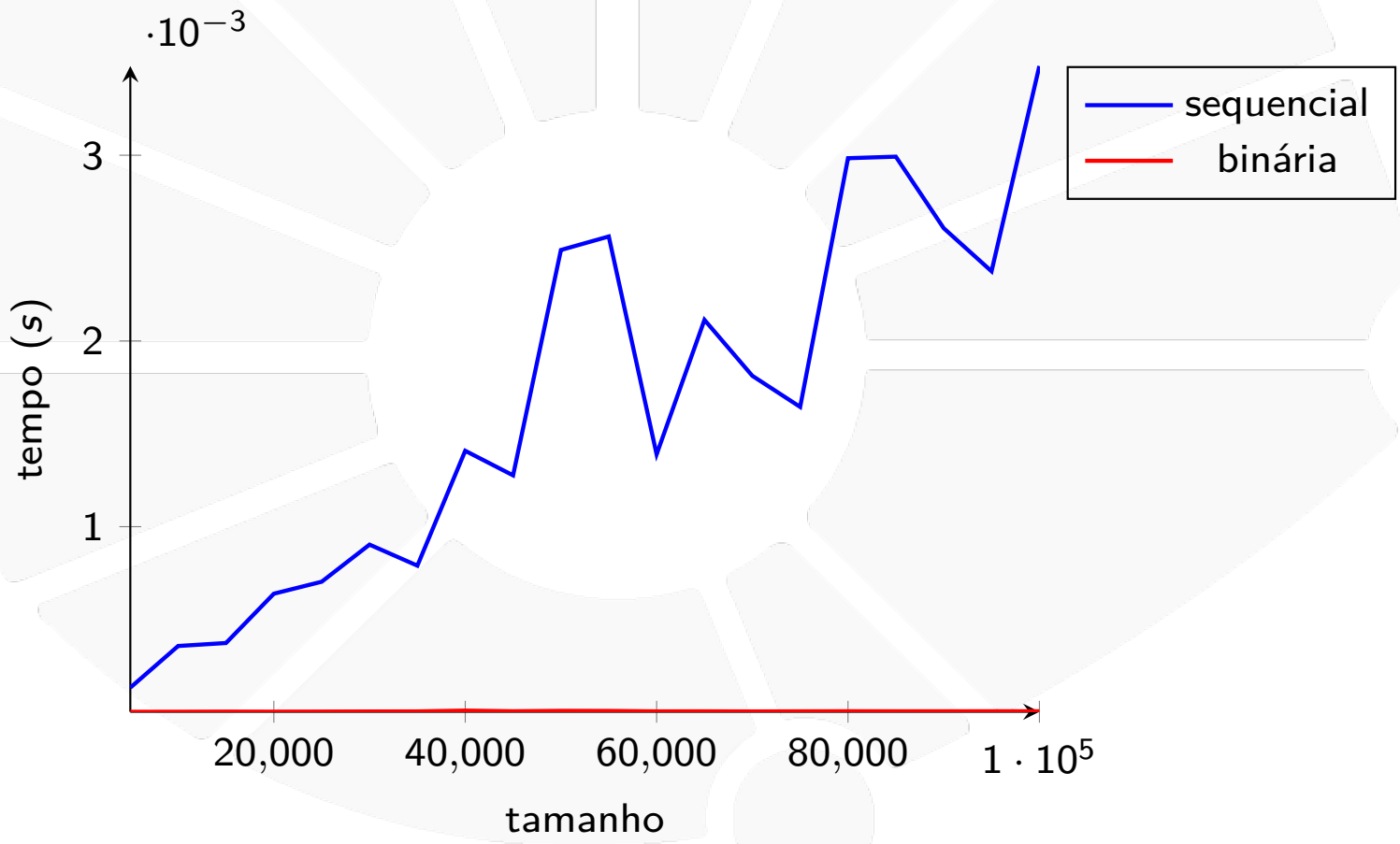
- $f(1) = 1$, pois o elemento do meio é o único elemento.
- $f(3) = 1 + f(1) = 2$, pois olhamos o elemento do meio e, se não encontramos, precisamos olhar uma lista de tamanho 1.
- $f(7) = 1 + f(3) = 3$, pois olhamos o elemento do meio e, se não encontramos, precisamos olhar uma lista de tamanho 3.
-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

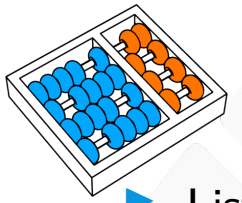
Experimentos



Experimento 1

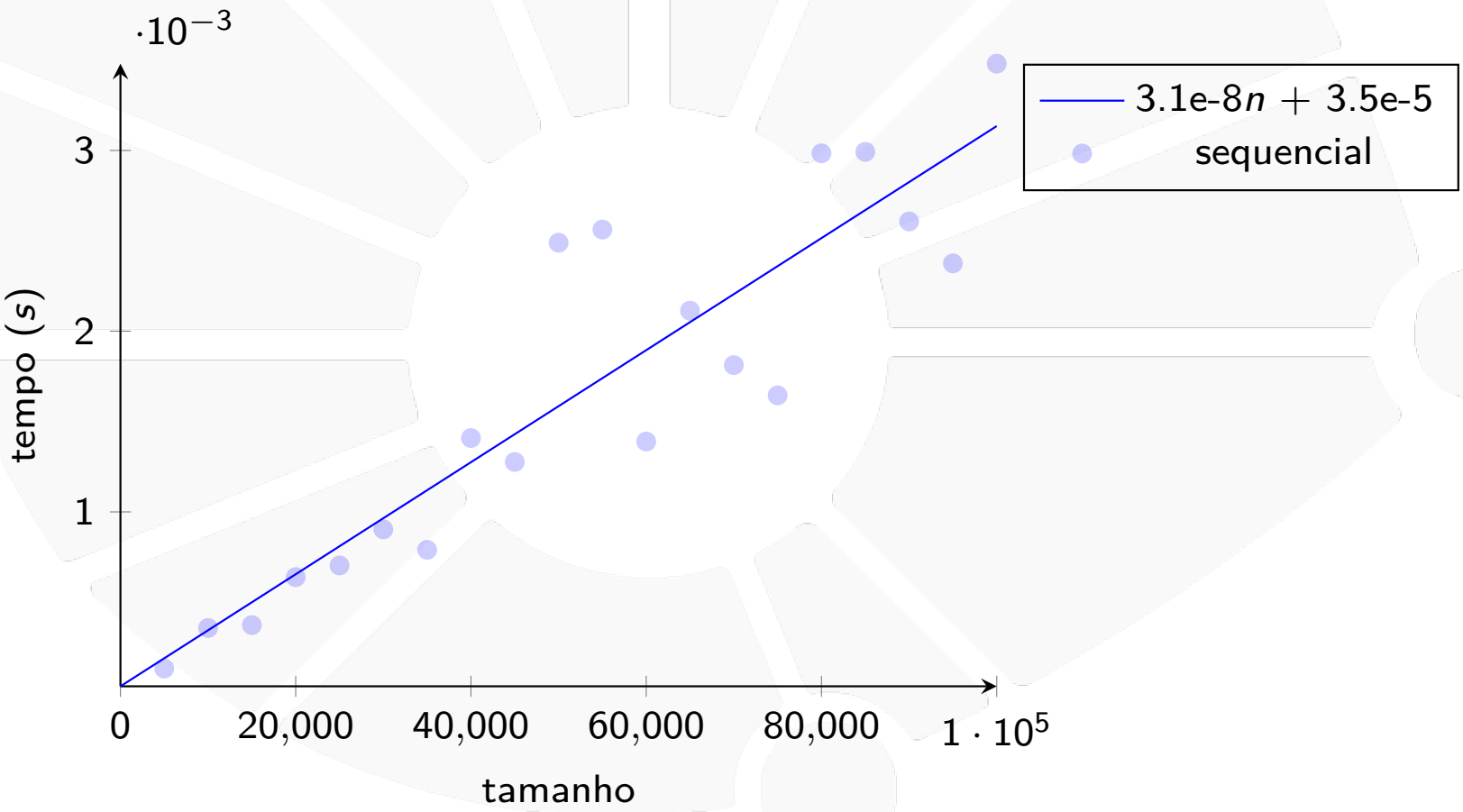
- Listas de tamanho 5000, 10000, . . . , 100000, com `l[i] = i`.
- Tiramos a média do tempo de 20 execuções.
- Escolhemos um `i` aleatório e procuramos por `l[i]` em `l`.

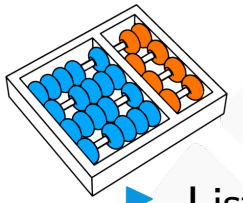




Experimento 1

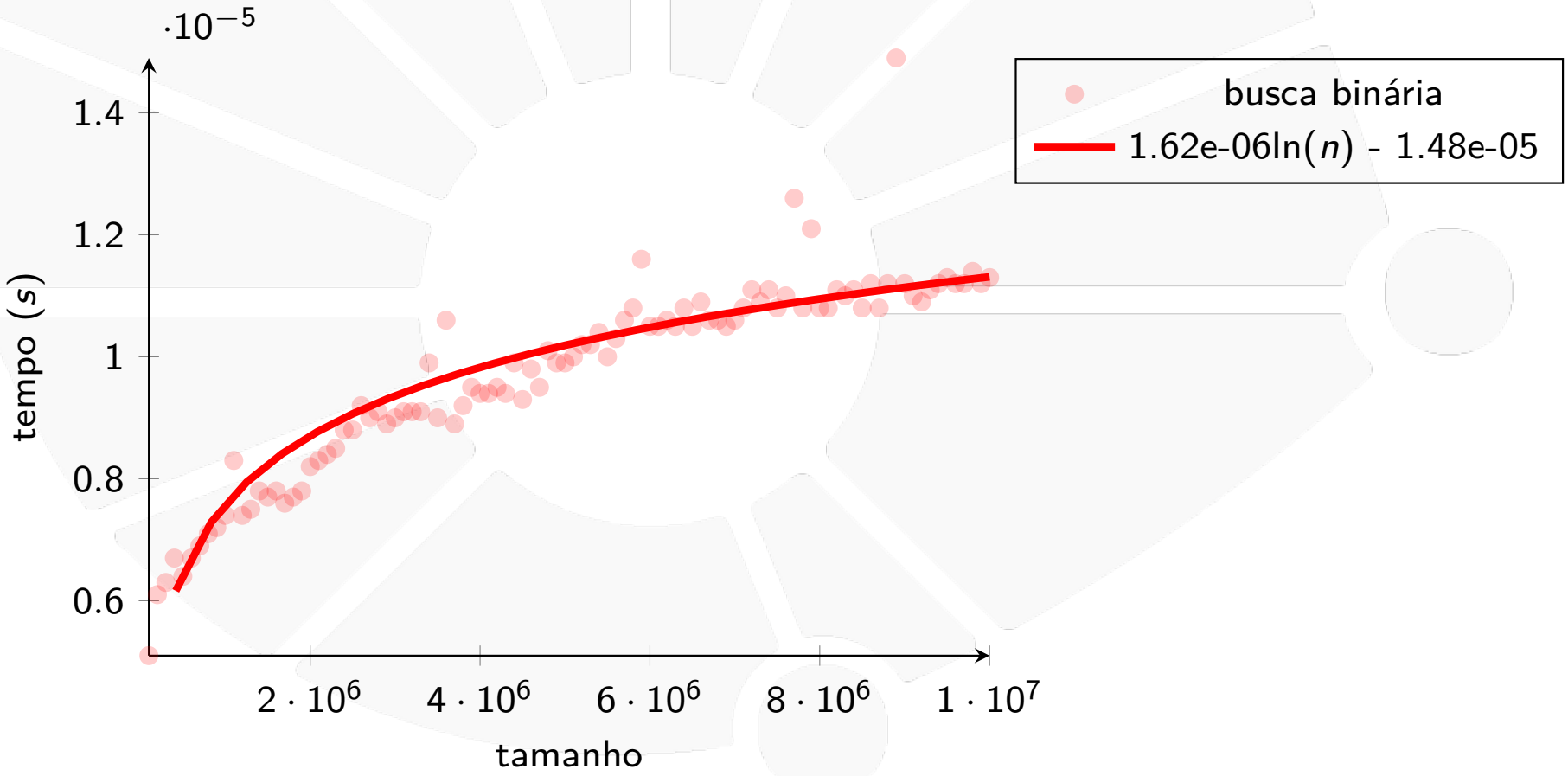
- Listas de tamanho 5000, 10000, . . . , 100000, com `l[i] = i`.
- Tiramos a média do tempo de 20 execuções.
- Escolhemos um `i` aleatório e procuramos por `l[i]` em `l`.

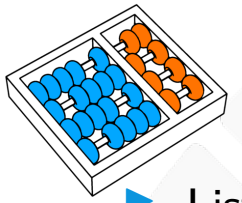




Experimento 1b — apenas busca binária

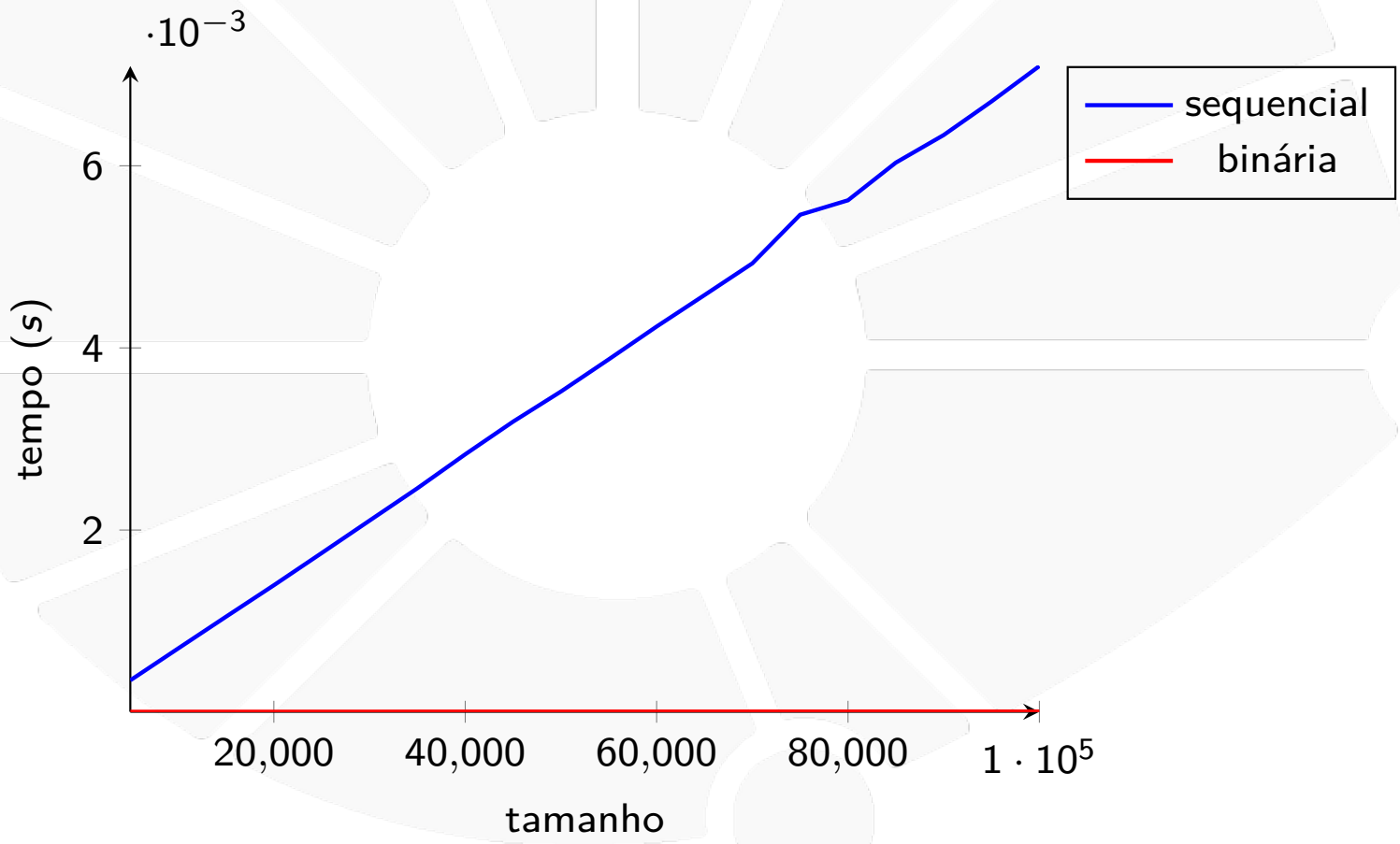
- Listas de tamanho 100000, 200000, . . . , 10000000, com `l[i] = i`.
- Tiramos a média do tempo de 100 execuções.
- Escolhemos um `i` aleatório e procuramos por `l[i]` em `l`.

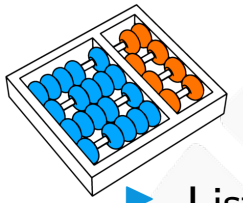




Experimento 2

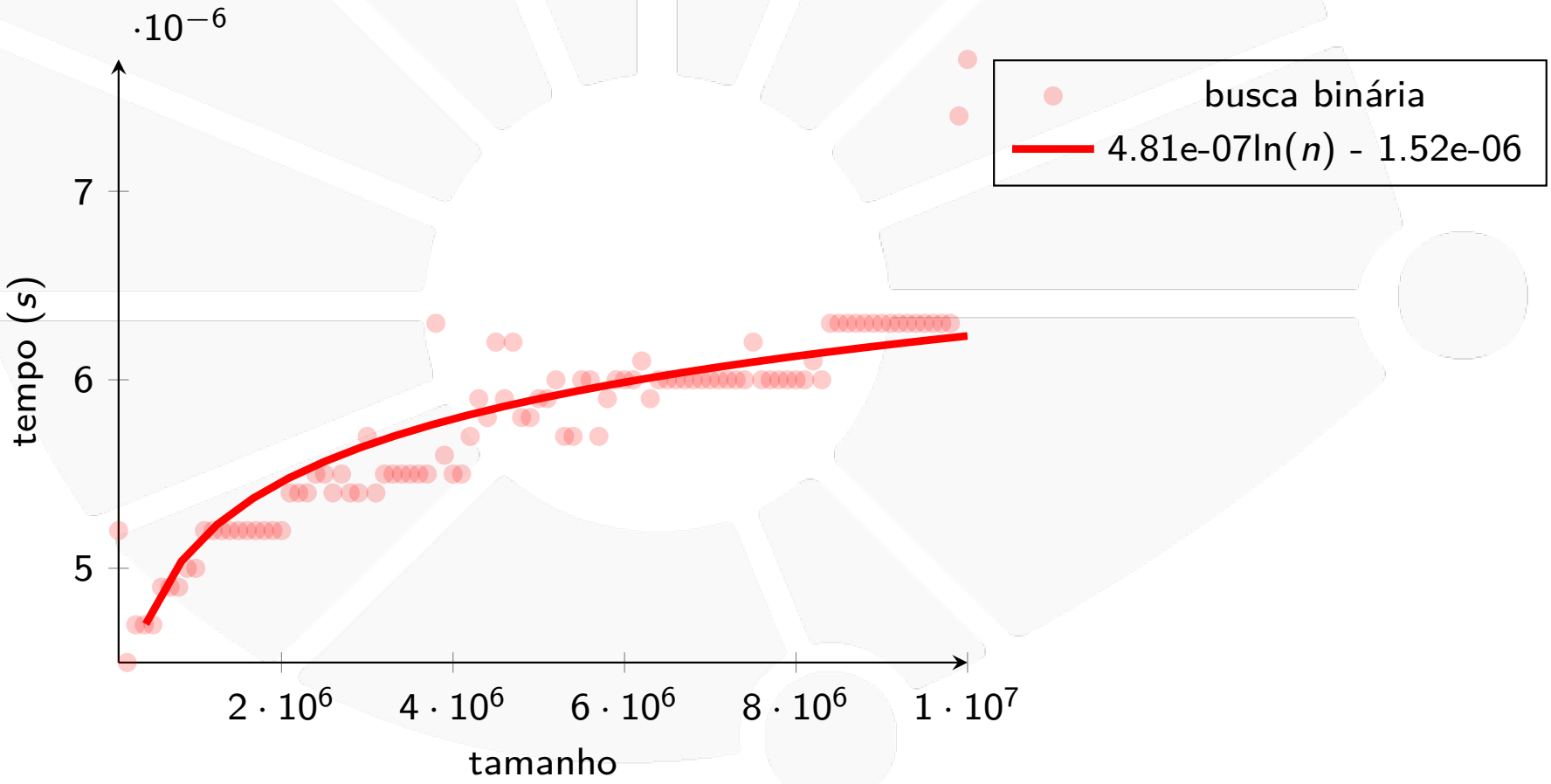
- Listas de tamanho 5000, 10000, . . . , 100000, com **l[i] = i**.
- Tiramos a média do tempo de 20 execuções.
- Buscamos por **len(l)** (que não está na lista).



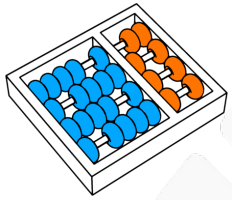


Experimento 2b — apenas busca binária

- 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).

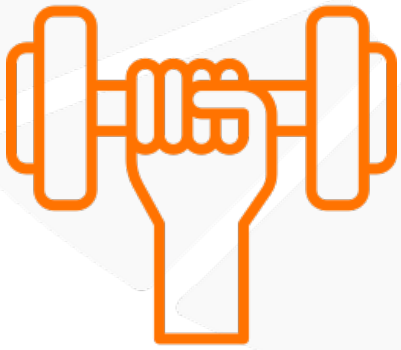


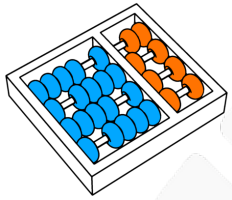
Exercícios



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

05/24

20

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

UNICAMP