

# **Trabalho I**

Caio Corrêa Chaves - 15444406

Vinicius Henrique P. Giroto - 11319656

Relatório de desempenho de algoritmos de busca e ordenação em C.

21 de setembro de 2025

# **Sumário**

1.	Introdução .....	3
1.1.	Especificações .....	3
1.1.1.	Clang .....	3
1.1.2.	Flags .....	3
1.1.3.	Sistema Operacional .....	3
1.1.4.	Processador .....	3
2.	Metodologia .....	4
3.	Resultados .....	5
3.1.	Busca sequencial .....	5
3.1.1.	Versão alternativa .....	5
3.2.	Busca binária .....	6
3.2.1.	Iterativa .....	6
3.2.2.	Recursiva .....	7
3.3.	Inversão .....	8
4.	Análise .....	9
4.1.	Algoritmos Lineares .....	9
4.2.	Algoritmos Logarítmicos .....	9
5.	Conclusão .....	11
6.	Participações .....	12
6.1.	Caio Corrêa Chaves .....	12
6.2.	Vinicius Henrique P. Giroto .....	12

# 1. Introdução

O projeto tem como objetivo estudar a performance de funções de busca e inversão em listas. As funções estudadas são a de busca sequencial, busca binária recursiva e busca binária iterativa, além de inversão.

A função de busca sequencial é uma busca linear que percorre a lista de elementos até encontrar o valor procurado ou chegar ao final da lista, enquanto a função de busca binária divide a lista em duas partes e busca o valor procurado na metade apropriada de maneira recursiva ou iterativa, por fim, a função de inversão é uma função que inverte a ordem dos elementos de uma lista.

O projeto foi escrito em C, e compilado utilizando o compilador Clang, e executado em uma máquina Linux. Os gráficos foram gerados utilizando Python e Matplotlib.

## 1.1. Especificações

### 1.1.1. Clang

```
> clang --version
clang version 20.1.8 (Fedora 20.1.8-4.fc42)
Target: x86_64-redhat-linux-gnu
Thread model: posix
InstalledDir: /usr/bin
Configuration file: /etc/clang/x86_64-redhat-linux-gnu-clang.cfg
```

### 1.1.2. Flags

```
CFLAGS := -std=c2x -Wall -Wextra
```

### 1.1.3. Sistema Operacional

```
> uname -a
Linux toolbox 6.16.7-200.fc42.x86_64 #1 SMP PREEMPT_DYNAMIC Thu Sep 11 17:46:54 UTC
2025 x86_64 GNU/Linux
```

### 1.1.4. Processador

```
AMD Ryzen™ 5 5600G with Radeon™ Graphics × 12 @ 3.90 GHz
```

## 2. Metodologia

Inicializou-se uma lista com  $n$  elementos ordenados de forma crescente, a ordenação é importante, pois é necessário para que a busca binária funcione corretamente. A lista foi então inicializada com os primeiros  $n$  números pares ordenados ( $0, 2, 4, 6, 8, \dots$ ). Mediú-se então o tempo de 100 buscas para cada valor, existente e inexistente no intervalo  $[0, 2n - 1]$ , juntamente aos valores `INT32_MIN` e `INT32_MAX` (garantidamente não existentes na lista), medindo-se o tempo de execução de cada busca e dividindo-o por 100, o número de repetições.

Optou-se pela análise separada para valores existentes e inexistentes, tendo em vista que a busca por valores inexistentes tendenciaria o resultado final. Essa abordagem permitiu a observação de diferenças de comportamento e tempos de execução para valores existentes e inexistentes.

### 3. Resultados

#### 3.1. Busca sequencial

Tem-se que este algoritmo é linear, ou seja, sua complexidade é  $O(n)$ , onde  $n$  é o tamanho da lista.

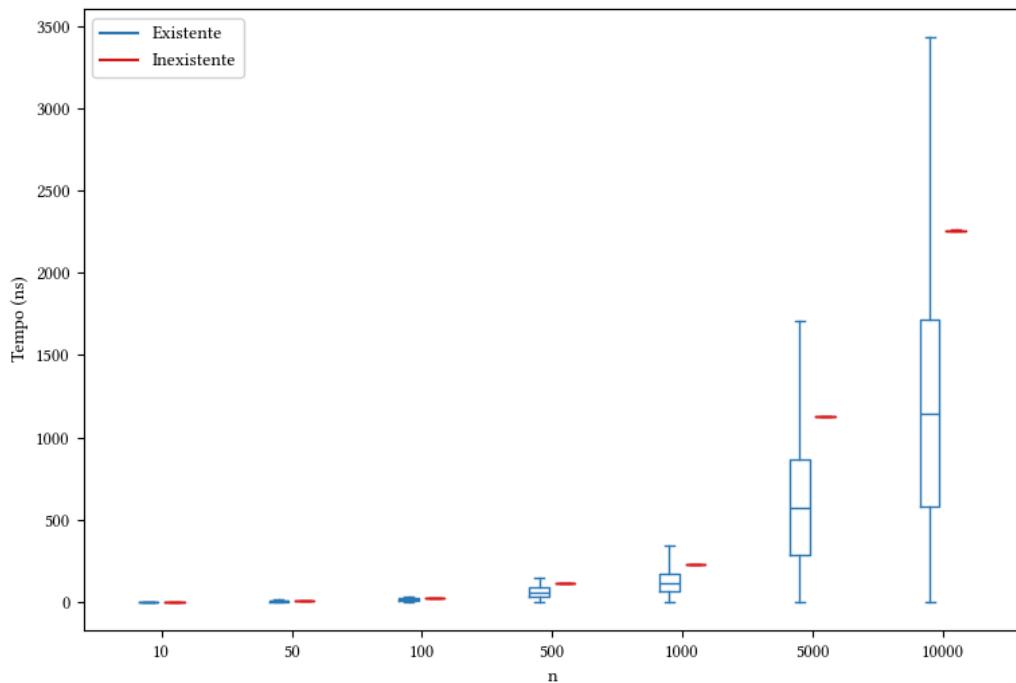


Figura 1: Comparaçāo da performance da busca sequencial.

Destaca-se que o tempo de execução para valores inexistentes é maior que para valores existentes (Figura 1), pois o algoritmo precisa percorrer toda a lista para concluir que o elemento não existe. A partir disso, cogitou-se que a posição teórica de um elemento – posição que o elemento assumiria se estivesse presente na lista – afeta a performance das funções de busca. Para testar essa hipótese, foi criada uma função de busca sequencial alternativa, qual assume a ordem crescente dos elementos e retorna assim que o elemento é encontrado ou quando ele é menor que o elemento atual. Isto faz com que a função não necessite percorrer o resto da lista.

##### 3.1.1. Versão alternativa

Assumindo que a lista está ordenada, a busca sequencial alternativa apresenta um desempenho melhor que a busca sequencial tradicional para valores inexistentes (Figura 2), tendo resultados semelhantes às buscas por valores existentes.

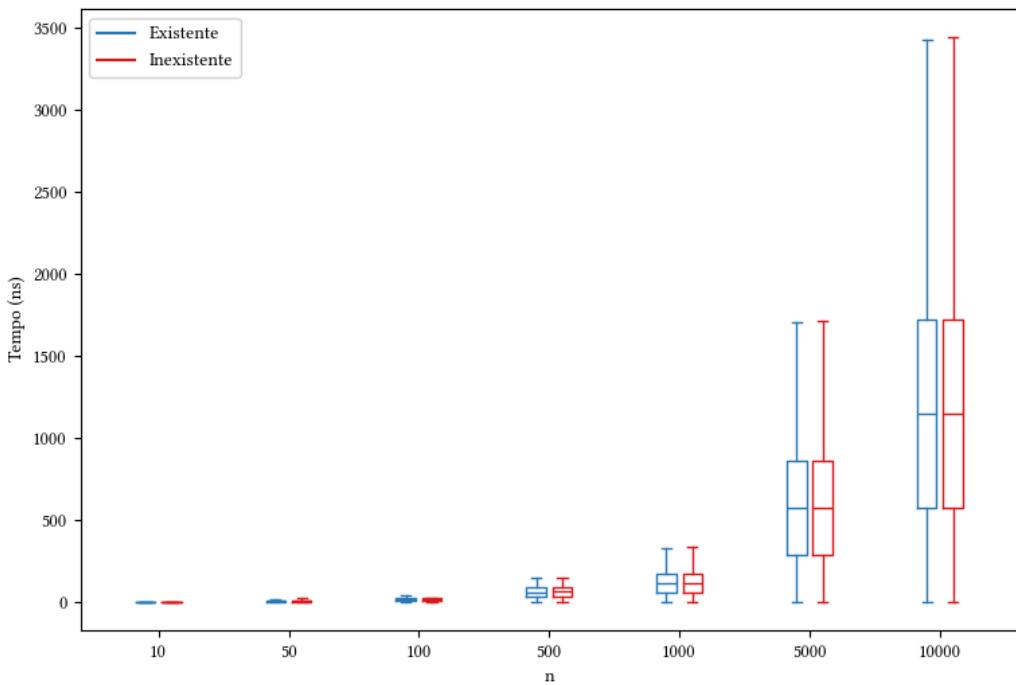


Figura 2: Comparação da performance da busca sequencial alternativa.

Este algoritmo pode ser útil para situações em que a lista está ordenada e as chances de não encontrar um elemento são consideráveis.

### 3.2. Busca binária

Diferindo-se do algoritmo de busca sequencial, este algoritmo é mais eficiente, tendo complexidade  $O(\log(n))$ , onde  $n$  é o tamanho da lista.

#### 3.2.1. Iterativa

A Figura 3 reflete o esperado do comportamento do algoritmo. Nota-se que a busca binária é mais eficiente que a busca sequencial e sequencial alternativa para valores existentes e inexistentes.

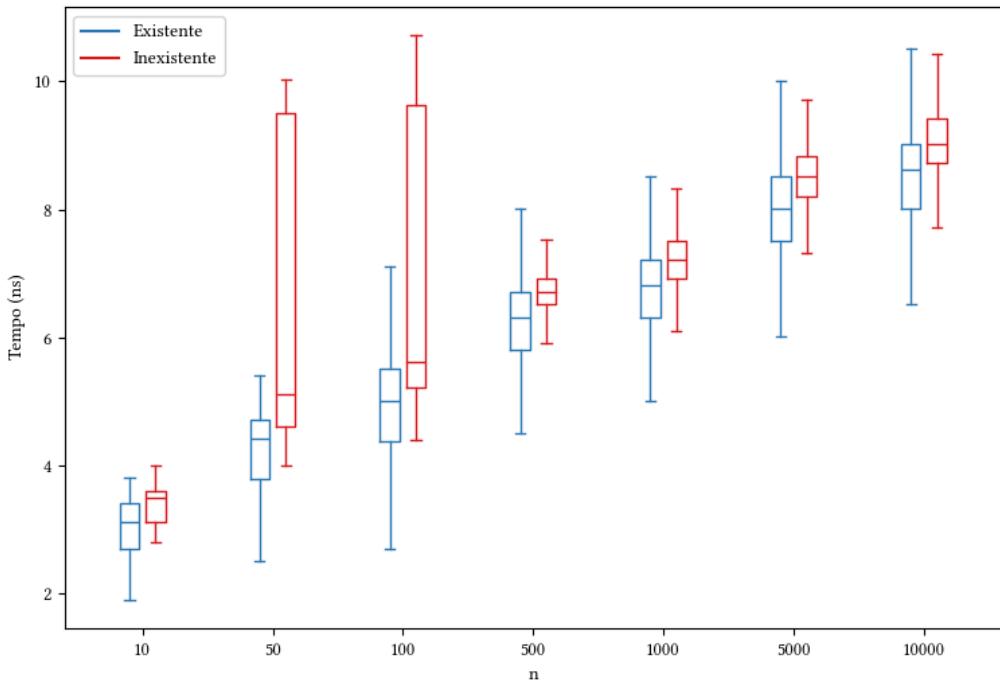


Figura 3: Comparação da performance da busca binária iterativa.

### 3.2.2. Recursiva

Nota-se que a busca recursiva performa semelhante a busca iterativa (Figura 4). Isso ocorre porque ambas têm a mesma complexidade de tempo  $O(\log(n))$ .

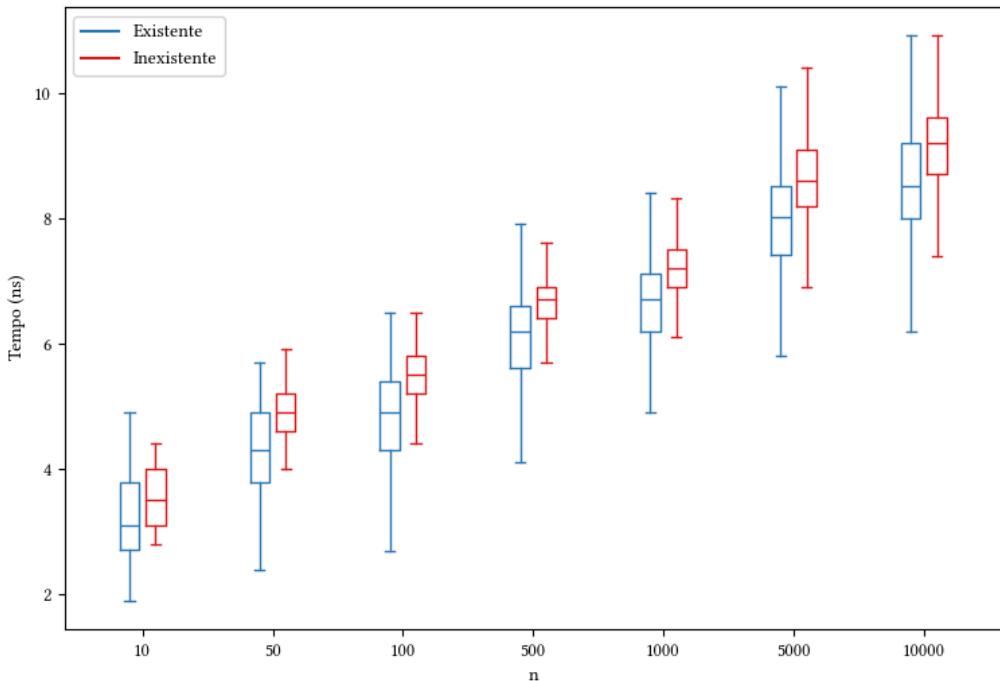


Figura 4: Comparação da performance da busca binária recursiva.

### 3.3. Inversão

O método de inversão é linear, ou seja, sua complexidade é  $O(n)$ , onde  $n$  é o tamanho da lista. O algoritmo foi implementado iterando sobre a lista e trocando os elementos de posição, começando pelo primeiro elemento e trocando-o com o último, depois pelo segundo elemento e trocando-o com o penúltimo, e assim por diante.

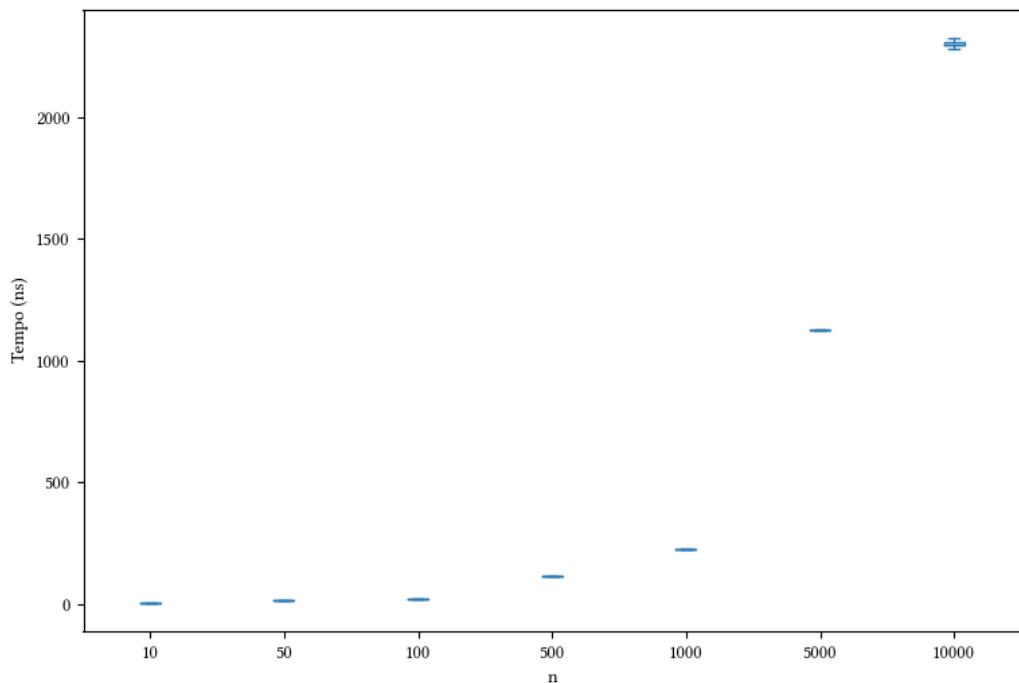


Figura 5: Inversão

Devido ao algoritmo não possuir outros parâmetros de entrada, sua performance depende apenas do tamanho da lista (diferente dos algoritmos de busca que requerem um valor de busca). Com isso, a variação do tempo de execução para um mesmo tamanho de lista é muito pequena (Figura 5). A baixa variância dos dados é responsável pelo achatamento do *boxplot*.

## 4. Análise

Tendo que os algoritmos aqui estudados possuem diferentes complexidades, analisou-se suas performances por categoria.

### 4.1. Algoritmos Lineares

Nota-se da Figura 6 que o algoritmo de busca sequencial alternativo para os casos de busca de valores inexistentes apresentou um desempenho médio muito bom comparado ao original. Nota-se também que o algoritmo de inversão apresenta o pior dos resultados, o que pode ser justificado por ser o único algoritmo que realiza escrita na memória, o que é uma operação muito mais lenta que a leitura.

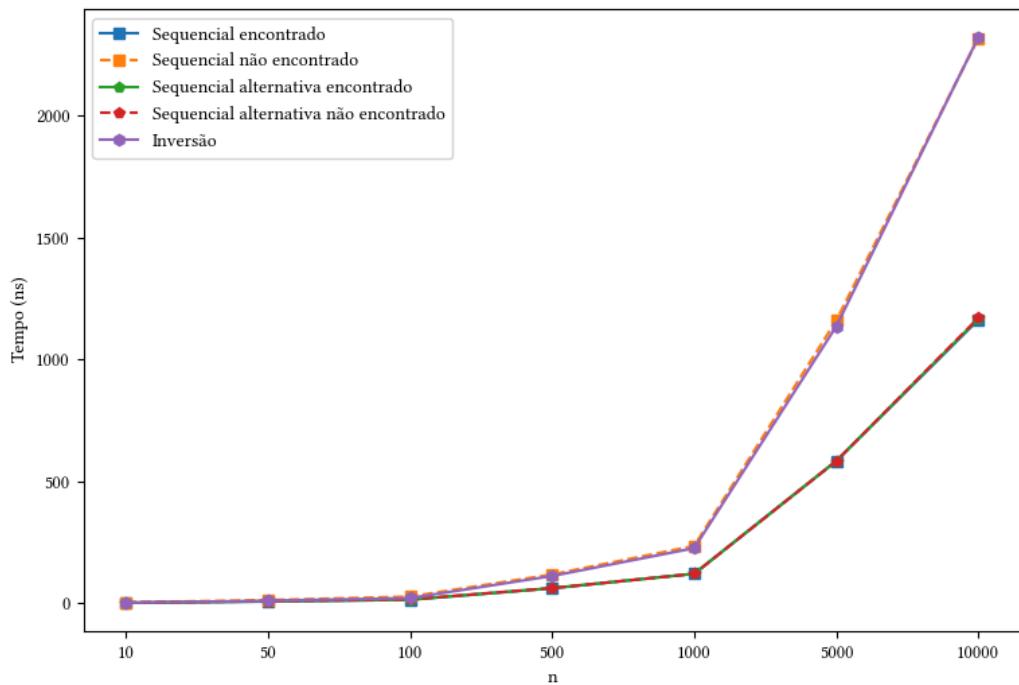


Figura 6: Comparação da performance dos algoritmos lineares.

### 4.2. Algoritmos Logarítmicos

Dentre os algoritmos de busca logarítmicos, a busca binária recursiva apresentou melhor desempenho em ambos os casos de valores inexistentes e existentes como pode ser visto na Figura 7.

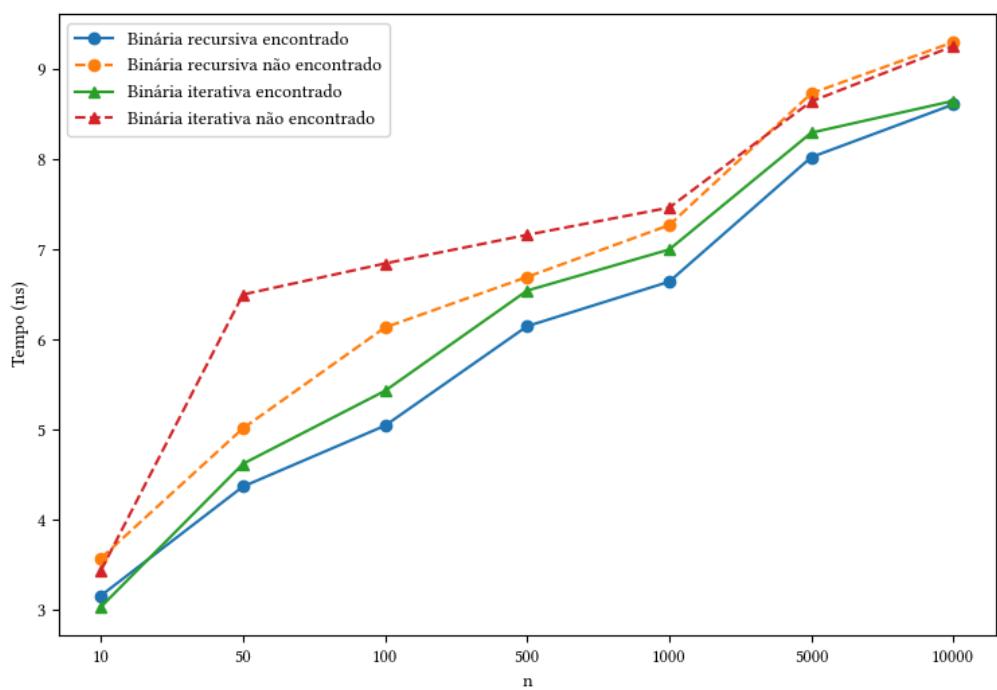


Figura 7: Comparaçāo da performance dos algoritmos logarítmicos.

## **5. Conclusão**

Dentre os algoritmos de busca estudados, a busca binária recursiva e iterativa apresentaram os melhores resultados, com destaque para o algoritmo recursivo. A busca sequencial alternativa possui a terceira melhor performance, deixando por fim a busca sequencial, a qual apresentou o pior desempenho.

Observou-se que a busca por valores inexistentes é majoritariamente pior que a busca por valores existentes para todos os algoritmos estudados, podendo ser melhorada para o caso da busca sequencial, como foi proposto e demonstrado no caso da busca sequencial alternativa.

O algoritmo de inversão apresenta comportamento linear, semelhante ao algoritmo de busca sequencial. Seu desempenho foi o pior dentre os algoritmos estudados.

## **6. Participações**

### **6.1. Caio Corrêa Chaves**

Responsável pela implementação dos algoritmos de busca e inversão.

### **6.2. Vinicius Henrique P. Giroto**

Realizou as análises e elaborou o relatório.