Estudo Comparativo do Desempenho da Busca Linear: Uma Análise de Implementações em C, Java e Python

Lucas Gabriel Eloi Viera[1], Manoel Lucas Lopes Reis[2]

Departamento de Computação – Campus São Luís-Monte Castelo Av. Getúlio Vargas, nº 04, Monte Castelo – CEP 65030-005 – São Luís – MA – Brazil

[2]Department of Computing – Federal Institute of Education, Science and Technology of Maranhão

[3]Curso de Sistemas de Informação
Instituto Federal de Educação, Ciência e Tecnologia do Maranhão(IFMA) – São Luís, MA – Brazil

eloi.lucas@acad.ifma.edu.br, manoel.reis@acad.ifma.edu.br.

Abstract. *This paper presents a theoretical and experimental analysis of the Linear Search algorithm. The main objective is to empirically verify its theoretical time complexity. The algorithm was implemented in C, Java, and Python, followed by a comparative performance analysis. Experiments were conducted by measuring the execution time on random vectors of varying sizes, simulating an average-case scenario. The results compare the empirical performance of the three implementations against the expected theoretical behavior and discuss the performance differences observed between the programming languages.*

Resumo. *Este artigo apresenta uma análise teórica e experimental do algoritmo de Busca Linear. O objetivo principal é verificar empiricamente sua complexidade de tempo teórica. O algoritmo foi implementado em C, Java e Python, seguindo-se de uma análise comparativa de desempenho. Foram realizados experimentos medindo o tempo de execução em vetores aleatórios de tamanhos variados , simulando um cenário de caso médio. Os resultados comparam o desempenho empírico das três implementações com o comportamento teórico esperado e discutem as diferenças de performance observadas entre as linguagens de programação.*

1. Introdução:

Algoritmos de computador representam conjuntos de etapas descritas com a precisão necessária para que uma máquina possa executá-las. O propósito primordial de um algoritmo é não apenas produzir uma solução correta para um problema de computação, mas também utilizar os recursos computacionais de forma eficiente durante esse processo. Para avaliar essa eficiência, o foco da análise de algoritmos recai sobre a taxa de crescimento do tempo de execução em função do tamanho da entrada (n). Essa análise assintótica permite ignorar coeficientes constantes e termos de ordem inferior, destacando a ordem de crescimento do tempo de execução, frequentemente utilizando notações como O , Ω e Θ.

Dentro das operações fundamentais em ciência da computação, a busca é uma tarefa primordial que visa determinar se um valor específico está presente em um arranjo. A Busca Linear surge como a técnica mais elementar para esta verificação, baseando-se na varredura sequencial do arranjo, elemento por elemento

O presente trabalho visa analisar o funcionamento e a complexidade da Busca Linear, realizando a verificação empírica da complexidade teórica. O estudo se propõe a implementar e comparar o desempenho deste algoritmo em diferentes ambientes de execução, buscando medir, comparar e interpretar seu comportamento em diversos cenários. O objetivo central é comparar os resultados empíricos de tempo de execução com a análise teórica que prediz um comportamento de crescimento linear.

2. Análise Teórica do Funcionamento e Complexidade

A análise da eficiência dos algoritmos é central para a Ciência da Computação, focando na taxa de crescimento do tempo de execução em função do tamanho da entrada (n). Essa avaliação é feita por meio da notação assintótica (O, Ω, Θ), que permite caracterizar o tempo de execução de um algoritmo, ignorando coeficientes constantes e termos de ordem inferior, pois o foco está na ordem de crescimento quando n se torna muito grande.

O processo de análise assume que cada operação elementar (como atribuição, comparação, ou operação aritmética) leva um tempo constante fixo. A Busca Linear, ou LINEAR-SEARCH, é um algoritmo fundamental cuja tarefa é determinar se um valor particular (x) está presente em um arranjo (A).

2.1 Busca Linear Padrão (LINEAR-SEARCH)

A Busca Linear Padrão caracteriza-se por varrer todo o arranjo. Essa variação não interrompe a execução ao encontrar o elemento procurado, retornando o índice da última ocorrência de x, se presente.

Tabela 1. Implementação em pseudocódigo de uma Busca Linear

Linha	Pseudocódigo	Custo	Execução
1	Ajustamos resposta para NOT-FOUND	c1	1
2	Para i = 1 até n.	c2	n + 1
2a	Se A[i] = x, então ajuste resposta para o valor de i	c2a	n
3	Retorne o valor de resposta como saída.	c3	1

O pseudocódigo demonstra que o laço principal é executado um número fixo de vezes, n, independentemente dos dados de entrada (onde n é o número de elementos no arranjo A).
A expressão do tempo de execução, T(n), é calculada pela soma dos custos elementares (c_i) multiplicados pelo número de vezes que a instrução é executada.

$$T(n) = c_1 \cdot 1 + c_2 \cdot (n+1) + c_{2a} \cdot n + c_3 \cdot 1$$

$$T(n) = (c_2 + c_{2a}) \cdot n (c_1 + c_2 + c_3)$$

O resultado é uma função linear da forma $\boldsymbol{c \cdot n + d}$, onde $c = (c_2 + c_{2a})$ e $d = (c_1 + c_2 + c_3)$ são constantes. Como o número de iterações do laço é invariável, o tempo de execução não depende da distribuição dos dados. A complexidade assintótica é, portanto, a mesma para todos os casos:

Tabela 2. Comparação das complexidades de tempos da Busca Linear

Caso	Complexidade de tempo
Pior caso	Θ(n)
Melhor caso	Θ(n)
Caso médio	Θ(n)

Em termos de limites assintóticos, T(n) é limitado tanto superiormente (O) quanto inferiormente (Ω) por uma função linear.

2.2 Busca Linear Melhorada (BETTER-LINEAR-SEARCH)

A Busca Linear Melhorada otimiza o algoritmo padrão ao retornar imediatamente o índice assim que o elemento procurado (x) é encontrado. Essa alteração torna o número de iterações do laço dependente da posição de x na entrada.

Tabela 3. Implementação em pseudocódigo de uma Busca Linear Melhorada

Linha	Pseudocódigo	Custo	Execução
1	Para i = 1 até n.	c1	n + 1
1a	Se A[i] = x, então retorne o valor de i como saída.	c1a	n
2	Retorne NOT-FOUND como saída.	c2	1

2.2.1 Análise do Pior Caso

O pior caso ocorre quando o algoritmo executa o tempo máximo possível. Isso acontece se o valor x não estiver presente no arranjo ou se for o último elemento (A[n]), forçando o laço a completar n iterações

$$T(n) = c_1 \cdot (n+1) + c_{1a} \cdot n + c_2 \cdot 1$$

$$T(n) = c_1 \cdot n + c_1 + c_{1a} \cdot n + c_2$$

$$T(n) = (c_1 + c_{1a}) \cdot n + (c_1 + c_2)$$

A expressão do tempo de execução, focando no termo dominante, resulta em uma função linear de n: $T(n) = (c_1 + c_{1a}) \cdot n + (c_1 + c_2)$. O pior caso é classificado como Θ(n).

2.2.2 Análise do Melhor Caso

O melhor caso ocorre quando o algoritmo executa o tempo mínimo possível. Isso acontece se x for encontrado na primeira posição (A), exigindo apenas uma iteração do laço. Neste cenário, o tempo de execução é dado pela soma dos custos constantes das operações iniciais e da primeira iteração:

$$T(n) = c_1 \cdot 1 + c_{1a} \cdot 1$$

$$T(n) = c_1 + c_{1a}$$

Visto que o tempo é uma quantidade constante e não depende de n, o melhor caso é classificado como Θ(1). O tempo de execução é limitado superiormente por O(n) em todos os casos, embora possa ser melhor.

Tabela 4. Comparação da complexidade de tempo entre três situações

Caso	Complexidade de tempo
Pior caso	Θ(n)
Melhor caso	Θ(1)
Caso médio	Θ(n) (Proporcional a n/2, que é Θ(n)

2.3 Busca Linear com Sentinela (SENTINEL-LINEAR-SEARCH)

Esta variação visa reduzir o custo por iteração do laço. O algoritmo insere o valor procurado (x) na última posição do arranjo (A[n]), agindo como um sentinela. Isso permite eliminar a verificação da condição de parada do arranjo ($i \leq n$) dentro do laço principal, pois a presença do sentinela garante que o laço sempre terminará.

Tabela 5. Implementação em pseudocódigo de uma Busca Linear com Sentinela

Linha	Pseudocódigo	Custo	Execução
1	Salve A[n] em último e então ponha x em A[n].	c1	1
2	Iguale i a 1.	c2	1
3	Enquanto A[i] = x, faça o seguinte:	c3	n
3a	Incremente i.	c3a	n - 1
4	Restaure A[n] de último.	c4	1
5	Se i < n ou A[n] = x, retorne o valor de i como saída.	c5	1
6	Caso contrário, retorne NOT-FOUND como saída.	c6	1

2.3.1 Análise do Pior Caso

O pior caso (quando x não está no arranjo original) ainda requer n iterações até que o sentinela em A[n] seja encontrado. O tempo de execução, T(n), é uma função linear de n:

$$T(n) = c_1 \cdot 1 + c_2 \cdot 1 + c_3 \cdot n + c_{3a} \cdot (n-1) + c_4 \cdot 1 + c_5 \cdot 1 + c_6 \cdot 1$$

$$T(n) = (c_3 + c_{3a}) \cdot n \cdot (c_1 + c_2 - c_{3a} + c_4 + c_5 + c_6)$$

$$T(n) = (c_3 + c_{3a}) \cdot n$$

Embora esta versão possua um fator constante menor multiplicando n (tornando-a potencialmente mais rápida na prática), a ordem de crescimento assintótica do pior caso permanece $\Theta(n)$.

2.3.2 Análise do Melhor Caso

O melhor caso ocorre quando x é encontrado imediatamente em A. O laço executa apenas uma vez. O tempo de execução é constante:

$$T(n) = c_1 \cdot 1 + c_2 \cdot 1 + c_3 \cdot 1 + c_{3a} \cdot 0 + c_4 \cdot 1 + c_5 \cdot 1 + c_6 \cdot 1$$

$$T(n) = (c_1 + c_2 + c_3 + c_4 + c_5).$$

Assim, o melhor caso é classificado como $\Theta(1)$.

2.4 Busca Linear Recursiva (RECURSIVE-LINEAR-SEARCH)

A Busca Linear pode ser implementada utilizando recursão, definindo um subproblema de busca em um subvetor progressivamente menor. A solução do problema original é construída a partir da solução de instâncias menores do mesmo problema. O caso-base ocorre quando o subvetor está vazio (n = 0), e cada chamada recursiva deve tender a esse caso-base.

2.4.1 Equação de Recorrência

No pior caso (onde o elemento não é encontrado ou é o último, forçando n chamadas), a complexidade de tempo T(n) é descrita pela equação de recorrência:

$$T(n) = T(n\text{-}1) + c$$

Onde T(n−1) é o tempo gasto na chamada recursiva para um problema de tamanho n−1, e c representa o custo constante das operações realizadas em cada nível recursivo (comparação do elemento atual e chamada da próxima instância).

2.4.2 Resolução da Recorrência

O Método da Substituição (ou expansão iterativa) revela o custo acumulado ao longo dos níveis de recursão, onde k é o número de expansões:

$$T(n) = T(n\text{-}k) + k \cdot c$$

Tabela 5. Análise da implementação recursiva a partir do método de substituição

Expansão	Expressão matemática
k = 1	T(n) = T(n-1) + c
k = 2	T(n) = (T(n-1-1) + c) + c = T(n-2) + 2c
k = 3	T(n) = ((T(n-1-1-1) + c) +c)) = T(n-3) + 3c
k	T(n) = T(n - k) + k * c

O processo termina quando k = n, atingindo o caso base $T(0) = c_1$. Substituindo k = n na expressão generalizada, tem-se:

$$T(n) = T(0) + n \cdot c$$

$$T(n) = c_1 + n * c$$

A solução é uma função linear de n. O termo dominante é n · c, e o termo constante de ordem inferior é c_1 . Portanto, o tempo de execução no pior caso e no caso médio é Θ(n). O melhor caso (encontrar o elemento na primeira chamada) é, teoricamente, Θ(1), embora o overhead da recursão possa torná-lo ligeiramente menos eficiente na prática do que a versão iterativa BETTER-LINEAR-SEARCH.

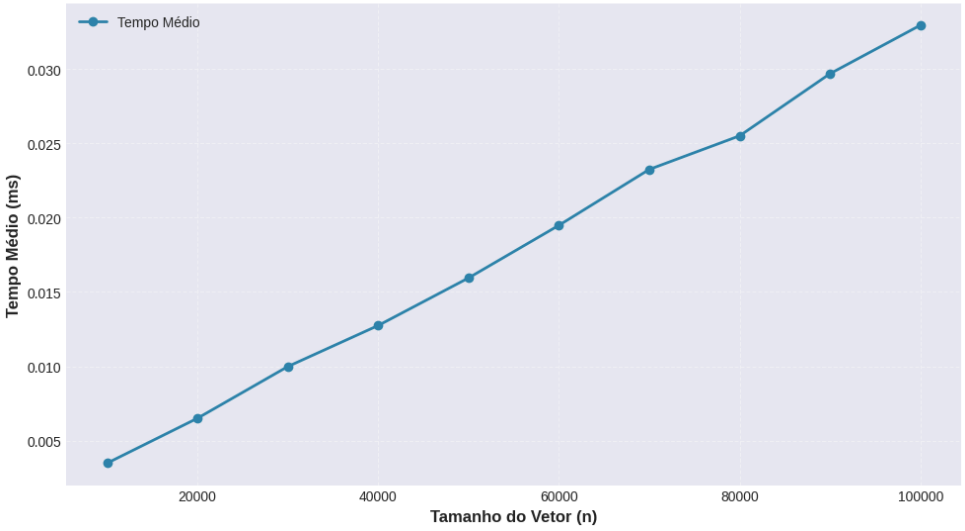
3. Resultados e Discussão

Os experimentos foram realizados em um computador com sistema operacional macOS, equipado com um processador Apple Silicon M1 e 8GB de memória RAM. Para garantir consistência e confiabilidade nas medições, cada algoritmo foi executado 50 vezes para cada tamanho de vetor, variando de 10.000 a 100.000 elementos, com incremento de

10.000. Em cada repetição, foi utilizada uma *seed* diferente no gerador de números aleatórios, assegurando que os 50 vetores analisados em cada faixa de tamanho fossem distintos entre si.

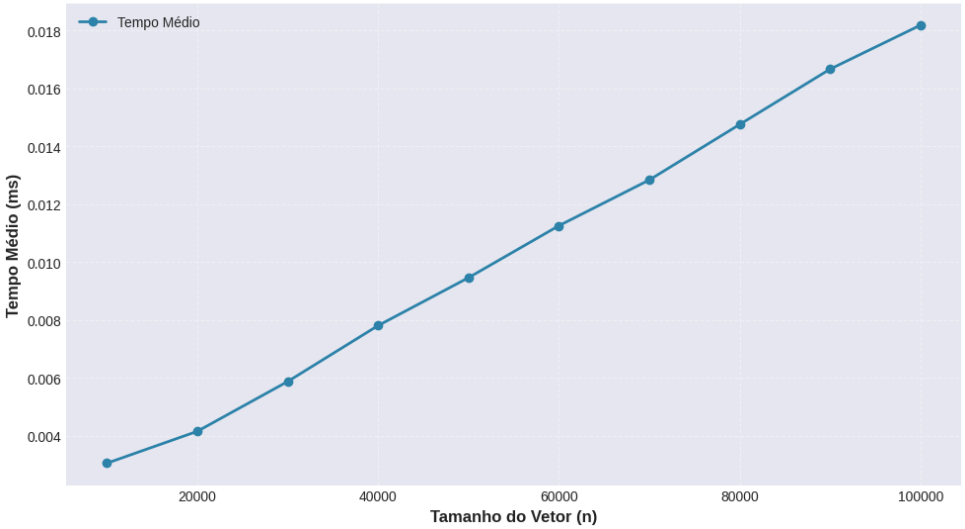
As implementações foram executadas utilizando as seguintes versões de ferramentas: Apple Clang 17.0.0 para o código em C, JDK 25 para o programa em Java e Python 3.13.3 para a versão em Python.

Figura 1. Desempenho do Algoritmo de Busca Linear em C



Fonte: O autor

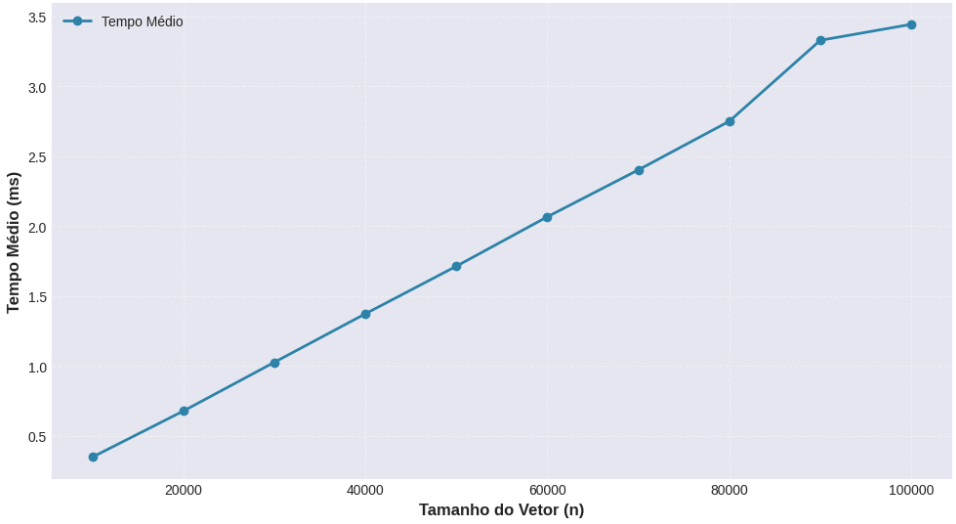
Figura 2. Desempenho do Algoritmo de Busca Linear em Java



Fonte: O autor

Inicialmente, foram analisados individualmente os resultados referentes ao tempo de execução de cada linguagem. Nos três casos, observou-se um crescimento aproximadamente linear do tempo de processamento à medida que o tamanho da entrada aumentava, conforme observado nas figuras de 1 a 3, o que era esperado considerando a natureza do algoritmo de busca linear.

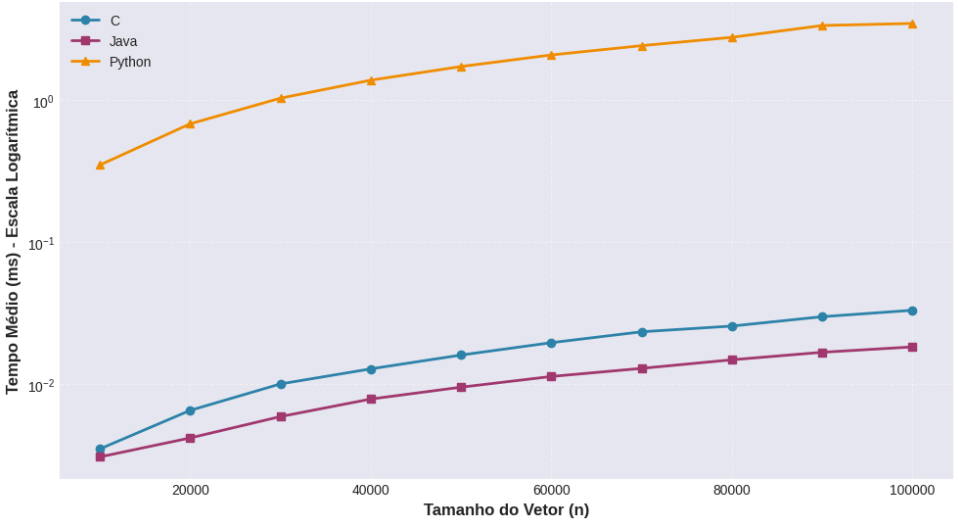
Figura 3. Desempenho do Algoritmo de Busca Linear em Python



Fonte: O autor

Ao comparar os resultados empíricos com a complexidade teórica prevista, confirmou-se o comportamento O(n) do algoritmo de busca linear. Em todas as linguagens, o tempo de execução cresceu proporcionalmente ao aumento do tamanho do vetor pesquisado, validando empiricamente a análise assintótica apresentada no pseudocódigo.

Figura 4. Comparativo de Desempenho - Busca Linear (C vs Java vs Python)



Fonte: O autor

Por fim, ao reunir os resultados das três linguagens em um único gráfico (Figura 4), foi possível realizar uma comparação direta entre os tempos de execução. Observou-se que as implementações em C e Java apresentaram desempenho significativamente melhor quando comparadas à versão em Python. Embora existam diferenças internas entre C e Java, ambas se beneficiam do fato de serem linguagens compiladas, o que reduz o custo adicional de processamento durante a execução.

Já o Python, por ser uma linguagem interpretada e de alto nível, executa com maior custo computacional. Ele realiza verificações em tempo de execução, possui tipagem

dinâmica e mantém uma série de abstrações adicionais que tornam o processamento mais lento quando comparado às outras duas linguagens. Esses fatores explicam o maior tempo de execução observado nos experimentos, mesmo mantendo o comportamento linear esperado.

4. Considerações finais

O presente trabalho cumpriu seu objetivo principal ao realizar uma análise teórica e experimental do algoritmo de Busca Linear e verificar empiricamente sua complexidade de tempo.

A análise assintótica do funcionamento da Busca Linear (Padrão, Melhorada, com Sentinela e Recursiva) confirmou que, para o pior caso e o caso médio, o tempo de execução é caracterizado como Θ(n), indicando um crescimento linear em relação ao tamanho da entrada n. O tempo de execução da Busca Linear é uma função linear da forma $c \cdot n + d$. Notavelmente, o melhor caso para a Busca Linear Melhorada (BETTER-LINEAR-SEARCH) é Θ(1), ocorrendo quando o elemento procurado é encontrado imediatamente na primeira posição.

Os resultados experimentais, obtidos em implementações em C, Java e Python, validaram a previsão teórica. Foi observado em todas as três linguagens um crescimento aproximadamente linear do tempo de processamento à medida que o tamanho do vetor aumentava. Esta correlação empírica com o comportamento de crescimento linear (O(n)) demonstra a validade da análise teórica de complexidade de tempo para o algoritmo estudado.

A análise comparativa entre as linguagens revelou diferenças de eficiência prática. A implementação em Python demonstrou ser a mais lenta quando comparada a C e Java, apesar de todas as linguagens apresentarem o comportamento assintótico linear esperado. A realização de experimentos sob condições controladas e a subsequente interpretação dos resultados em relação à teoria permitiram cumprir a proposta de integrar a teoria e a prática na avaliação do desempenho algorítmico.

Em suma, o estudo confirmou que, embora existam variações de implementação que influenciam o tempo de execução (como as constantes ocultas na notação assintótica, que diferem entre linguagens), a Busca Linear permanece assintoticamente limitada por Θ(n) no caso geral, um resultado fundamental da Ciência da Computação.

Referências Bibliográficas

CORMEN, Thomas H. (2014), Desmistificando algoritmos. Tradução: Arlete Simille Marques. 1. ed. Rio de Janeiro: Elsevier

CORMEN, Thomas H.; LEISERSON, Charles E.; RIVEST, Ronald L.; STEIN, Clifford. (2022), Introduction to Algorithms. Fourth Edition. Cambridge, Massachusetts; London, England: The MIT Press