Análise de Algoritmos de Busca

César Tallys, Emerson Souza
Instituto Federal de Brasília - Campus Taguatinga

Agosto 2024

1 Introdução

A análise de algoritmos é um pilar fundamental na ciência da computação, permitindo a avaliação da eficiência e do desempenho de algoritmos em diferentes contextos computacionais. Dentro desse vasto campo de pesquisa, a busca por elementos em conjuntos de dados emerge como um desafio crucial e recorrente. Para enfrentar esse desafio, foram desenvolvidas várias abordagens, dentre as quais se destacam três principais: a busca sequencial, a busca sequencial otimizada e a busca binária. Cada uma dessas estratégias apresenta características únicas, desde a simplicidade e flexibilidade até a eficiência e escalabilidade.

Neste relatório, exploraremos em profundidade esses métodos de busca, analisando não apenas suas definições e algoritmos, mas também realizando uma análise empírica detalhada de seu desempenho. Além disso, realizaremos uma análise assintótica dos algoritmos para avaliar teoricamente seu comportamento e compararemos os resultados teóricos com os empíricos para validar as previsões e confirmar os resultados. Nosso objetivo é compreender tanto os aspectos teóricos quanto os práticos dessas técnicas, oferecendo insights sobre sua viabilidade e eficiência em diferentes cenários computacionais. Ao avaliar o comportamento de cada método em diversos conjuntos de dados e contextos de aplicação, pretendemos fornecer orientações úteis para a seleção e utilização eficaz de algoritmos de busca em situações do mundo real.

2 Conceitos Preliminares

Busca Sequencial: Este método consiste em percorrer os elementos da coleção um a um do início ao fim até encontrar o elemento desejado. É uma abordagem simples e direta, adequada para coleções desorganizadas ou de pequeno e médio

porte. Temos, em média, para uma lista de tamanho n, uma busca sequencial realizará $\frac{n}{2}$ comparações em uma busca bem-sucedida e n comparações no pior caso. Portanto, sua maior desvantagem é o tempo de execução, que tende a crescer linearmente com o tamanho, tornando-o menos eficiente para grandes conjuntos de dados. Quando o conjunto de dados é relativamente pequeno, a busca sequencial pode ser uma escolha eficiente devido à sua simplicidade e baixo custo de implementação. Por exemplo, na busca de um contato em uma lista telefônica curta ou na verificação de presença de um item em um inventário limitado.

Busca Sequencial Otimizada: Esta variante da busca sequencial visa melhorar o desempenho da busca por meio de organização estratégica completa. Isto pode ser conseguido, por exemplo, mantendo uma coleção de números organizada em ordem crescente, fazendo com que a busca não necessariamente precise percorrer até o fim da lista. Além disso, estruturas de dados auxiliares, como índices ou árvores de busca, podem ser usadas para acelerar o processo de busca. Esta abordagem visa minimizar o número de comparações necessárias para encontrar o elemento desejado, resultando em melhor desempenho em comparação à busca sequencial tradicional. Temos, em média, se o elemento estiver presente, $\frac{n}{2}$ comparações, pois a busca pode encontrar o elemento entre a primeira posição e a última. E como pior caso, o número de buscas será n, pois o elemento que buscamos é o último da lista. No entanto, por necessitar de uma prévia ordenação dos dados, o algoritmo de busca sequencial otimizada pode ser menos eficiente que a busca sequencial simples para grandes conjuntos de dados, já que o tempo gasto na ordenação pode superar as vantagens obtidas pela busca otimizada.

Busca Binária: Conhecida como técnica de "dividir e conquistar" [1], a busca binária é particularmente eficaz quando usada em coleções ordenadas. O algoritmo funciona dividindo repetidamente o conjunto de dados ao meio e verificando se o elemento na posição central é maior, menor ou igual à chave que está sendo procurada. Se o elemento central for igual à chave, a busca termina com sucesso. Caso contrário, se a chave for menor que o elemento central, a busca continua recursivamente na metade inferior da coleção; se a chave for maior, a busca continua na metade superior. Este processo é repetido até que o elemento seja encontrado ou até que todas as possibilidades tenham sido descartadas, determinando que o elemento não existe na coleção. Este método de redução contínua do espaço de busca resulta em um tempo de execução proporcional ao $\log_2(n)$, tornando-o excepcionalmente eficiente para grandes conjuntos de dados.

Ao compreender os princípios destas técnicas de busca, temos a opção de escolher a abordagem mais adequada para uma determinada aplicação, levando em conta fatores como o tamanho e a ordem da coleção, além dos requisitos de eficiência e eficácia.

3 Análise Empírica

A análise empírica consiste em avaliar o desempenho de algoritmos através de experimentos práticos, observando como eles se comportam em cenários reais ao invés de apenas estudar suas propriedades teóricas. Esse tipo de análise envolve a implementação dos algoritmos e a execução de testes em diferentes conjuntos de dados, medindo o tempo de execução, o número de operações realizadas e outros indicadores de desempenho. Neste relatório, a análise empírica será realizada primeiramente para os três algoritmos de busca discutidos — busca sequencial, busca sequencial otimizada e busca binária — com o objetivo de obter uma compreensão inicial sobre como esses algoritmos se comportam em diferentes tamanhos de dados e distribuições. Em seguida, os resultados empíricos serão comparados com a análise assintótica, que fornecerá uma visão teórica detalhada sobre a eficiência e escalabilidade desses algoritmos, confirmando ou refutando as observações práticas.

Os testes foram realizados em uma máquina equipada com um processador AMD Ryzen 5 4600G e 16 GB de RAM a 3200 MHz. Os algoritmos foram construídos utilizando-se da linguagem Javascript [3]. Para a avaliação dos algoritmos de busca, foram utilizados arrays com tamanhos $N = 10^4$, $N = 10^5$, $N = 10^6$ e $N = 10^7$ preenchidos com números aleatórios entre 1 e 10^7. Em cada tamanho de array, foram medidas as performances dos algoritmos para a busca de Q chaves, onde Q assumiu os valores 10^2, 10^3, 10^4 e 10^5. Cada entrada precisou então ser ordenada apenas uma vez para cada tamanho de Q buscas. Isso resultou em quatro conjuntos distintos de medições de tempo (em segundos) de execução para cada um dos três algoritmos de busca. Os tempos registrados foram então dispostos em gráficos para facilitar a visualização e a comparação dos desempenhos dos algoritmos em relação às diferentes variáveis testadas. Esses gráficos serão apresentados a seguir para fornecer uma análise visual detalhada dos resultados obtidos:

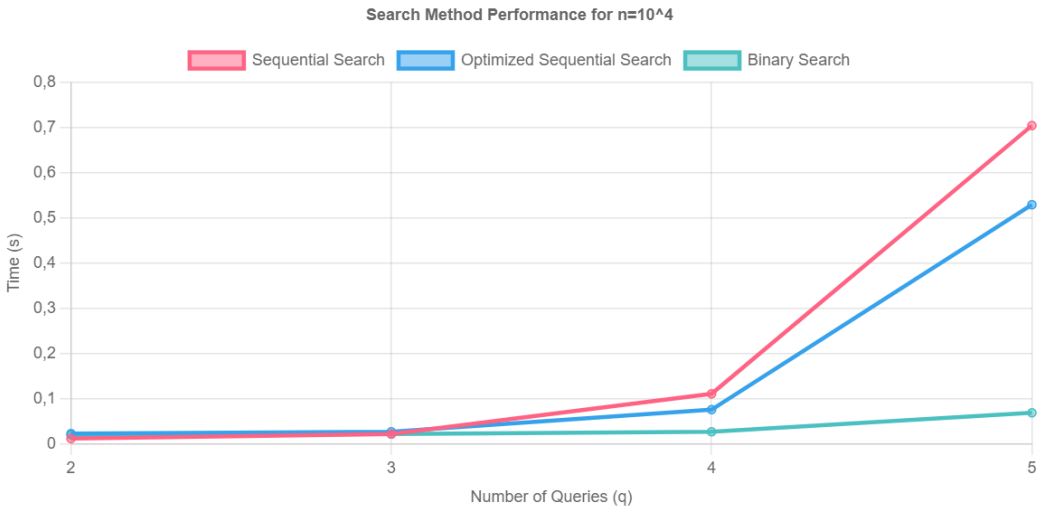

Figura 1: Performance para entradas de tamanho $N = 10^4$

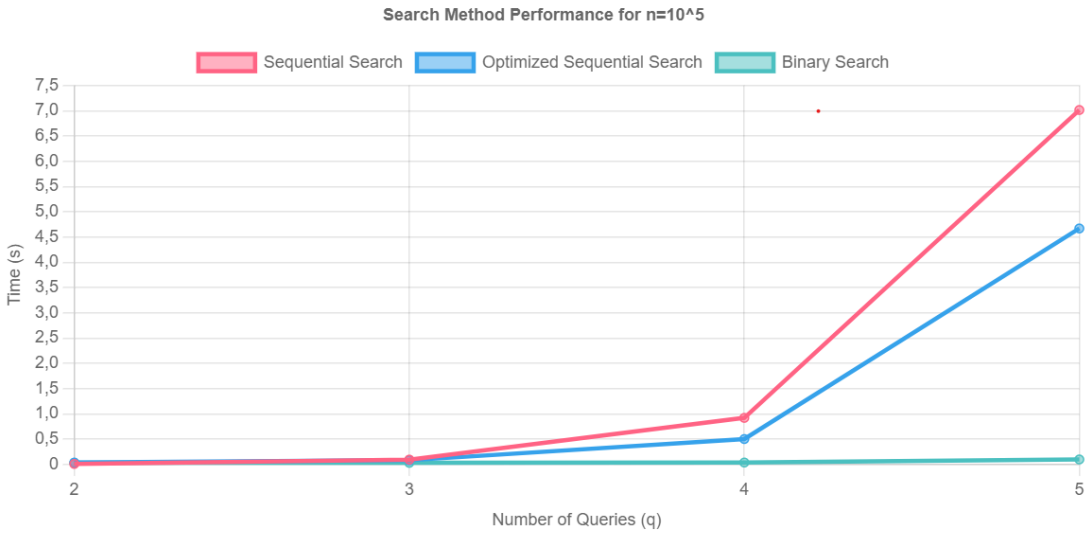

Figura 2: Performance para entradas de tamanho $N = 10^5$

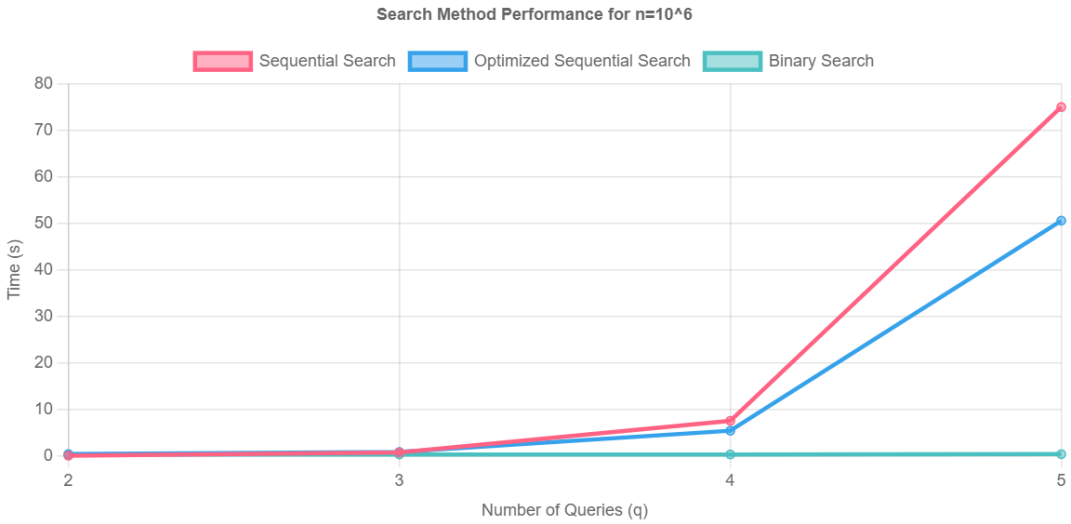

Figura 3: Performance para entradas de tamanho $N = 10^6$

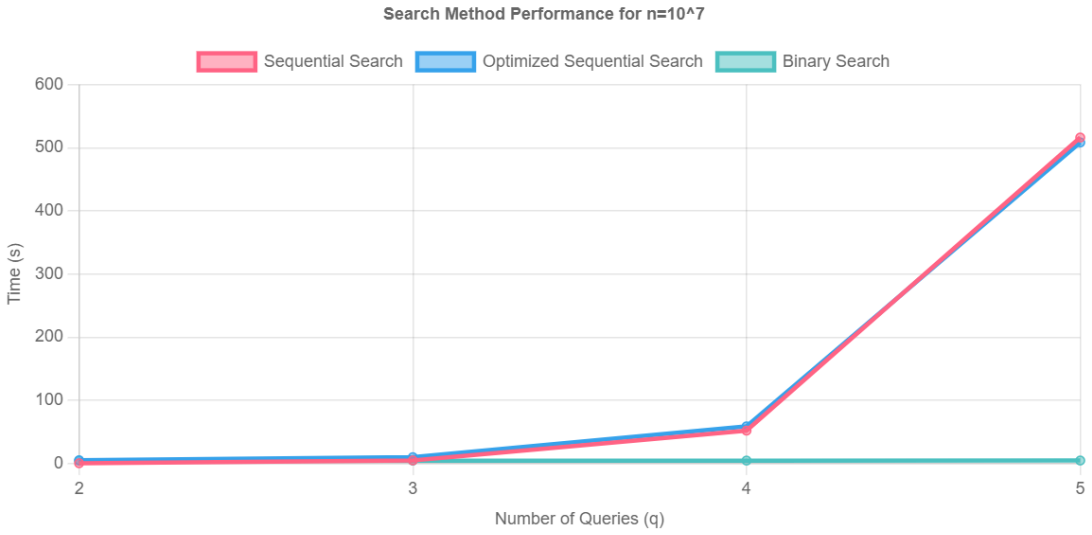

Figura 4: Performance para entradas de tamanho $N = 10^7$

Os tempos registrados foram dispostos em tabelas:

Busca Sequencial

N	Q	Time (s)
4	2	0.012
4	3	0.022
4	4	0.111
4	5	0.704
5	2	0.010
5	3	0.096
5	4	0.922
5	5	7.013
6	2	0.068
6	3	0.760
6	4	7.551
6	5	74.980
7	2	0.509
7	3	5.027
7	4	52.171
7	5	516.111

Busca Sequencial Otimizada

N	Q	Time (s)
4	2	0.023
4	3	0.027
4	4	0.076
4	5	0.529
5	2	0.037
5	3	0.083
5	4	0.503
5	5	4.669
6	2	0.396
6	3	0.863
6	4	5.424
6	5	50.567
7	2	5.197
7	3	10.077
7	4	58.878
7	5	508.348

Busca Binária

N	Q	Time-order (s)	Time (s)
4	2	0.006	0.006
4	3	0.006	0.013
4	4	0.006	0.016
4	5	0.006	0.043
5	2	0.017	0.018
5	3	0.017	0.018
5	4	0.017	0.022
5	5	0.017	0.057
6	2	0.263	0.263
6	3	0.263	0.265
6	4	0.263	0.270
6	5	0.263	0.321
7	2	4.848	4.848
7	3	4.848	4.849
7	4	4.848	4.857
7	5	4.848	4.935

A análise dos gráficos comparativos entre os algoritmos de busca revela insights importantes sobre o desempenho relativo de cada método em diferentes tamanhos de arrays e números de chaves a serem buscadas.

- **Busca Sequencial**: A busca sequencial apresenta um crescimento exponencial no tempo de execução à medida que o número de chaves a ser buscada aumenta. Para arrays maiores ($N = 6$ e $N = 7$), o tempo de execução sobe drasticamente com o aumento de Q, indicando que a busca sequencial é significativamente afetada pelo número de chaves a ser pesquisado. Esse comportamento é esperado, dado que a busca sequencial precisa percorrer toda a lista para cada chave, resultando em um aumento substancial do tempo conforme o tamanho do conjunto de dados e o número de chaves aumentam.

- **Busca Sequencial Otimizada**: A busca sequencial otimizada também mostra um aumento no tempo de execução com o aumento do número de chaves, mas de forma menos pronunciada em comparação com a busca sequencial simples. O desempenho da busca otimizada é melhor para arrays menores, mas a necessidade de ordenação dos dados ainda afeta seu desempenho em arrays maiores. Quando o tamanho do vetor é muito grande, o tempo de execução pode até superar o da busca sequencial para alguns casos, refletindo o impacto do tempo de ordenação adicional necessário.

- **Busca Binária**: A busca binária se destaca por sua eficiência, mostrando um tempo de execução muito menor em comparação com a busca sequencial e a busca sequencial otimizada, independentemente do tamanho do array ou do número de chaves. A busca binária mantém um tempo de execução relativamente constante mesmo com o aumento de Q, refletindo sua eficiência com uma complexidade de tempo de $O(\log n)$. Para arrays menores, o tempo de execução é muito baixo e, embora aumente para arrays maiores, o crescimento é quase insignificante se comparado aos outros algoritmos. O maior tempo de execução registrado foi de 4,935 segundos ($N = 10^7$ e $Q = 10^5$), com maior relevância do tempo de ordenação.

Em resumo, a busca binária se mostra a mais eficiente entre os três algoritmos testados, especialmente para conjuntos de dados maiores. A busca sequencial otimizada oferece vantagens em comparação com a busca sequencial simples, mas sua eficiência pode ser comprometida pelo custo da ordenação dos dados. A busca sequencial, por outro lado, demonstra uma escalabilidade pobre com o aumento do tamanho dos dados e do número de chaves, o que a torna menos adequada para grandes conjuntos de dados.

4 Análise Assintótica dos Algoritmos de Busca

4.1 Busca Sequencial

Descrição: A busca sequencial percorre toda a lista até encontrar o elemento desejado ou até chegar ao final da lista.

Análise Assintótica:

- **Melhor Caso:** O melhor caso ocorre quando o elemento buscado está na primeira posição do array. Nesse caso, a complexidade é $O(1)$.
- **Pior Caso:** O pior caso ocorre quando o elemento está na última posição ou não está presente na lista. Neste caso, a busca precisa percorrer todo o array, resultando em uma complexidade de $O(n)$, onde n é o número de elementos no array.
- **Caso Médio:** Em média, a busca sequencial faz $\frac{n}{2}$ comparações, o que ainda é $O(n)$.

Complexidade: A complexidade assintótica da busca sequencial é $O(n)$.

4.2 Busca Sequencial Otimizada

Descrição: A busca sequencial otimizada funciona apenas em coleções ordenadas e pode usar estruturas auxiliares para acelerar a busca. A análise da complexidade depende das operações de pré-processamento e da busca efetiva.

Análise Assintótica:

- **Pré-processamento (Ordenação):** Se os dados precisam ser ordenados antes da busca, a complexidade da ordenação é $O(n \log n)$, usando algoritmos de ordenação eficientes, como o Merge Sort ou o Quick Sort.
- **Busca Após Ordenação:** Depois da ordenação, a busca sequencial otimizada percorre a lista, o que é $O(n)$. Portanto, a busca otimizada após a ordenação tem uma complexidade total de $O(n \log n + n)$, que simplifica para $O(n \log n)$.

Complexidade: A complexidade assintótica da busca sequencial otimizada é $O(n \log n)$, considerando o custo da ordenação.

4.3 Busca Binária

Descrição: A busca binária funciona apenas em arrays ordenados e divide o array ao meio repetidamente até encontrar o elemento ou determinar que ele não está presente.

Análise Assintótica:

Para a busca binária, a complexidade pode ser analisada usando o método mestre para resolver recorrências do tipo $T(n) = T(n/2) + O(1)$.

Aplicação do Método Mestre:

A recorrência da busca binária é $T(n) = T(n/2) + O(1)$, onde:

- $a = 1$ (número de subproblemas),
- $b = 2$ (fator de divisão do problema),
- $f(n) = O(1)$ (custo de combinar as soluções dos subproblemas).

De acordo com o método mestre:

- Comparando com a forma geral $T(n) = aT(n/b) + f(n)$,
- Aqui, $a = 1$, $b = 2$, e $f(n) = O(1)$.
- Para o método mestre:
  - $f(n) = O(n^c)$, onde $c = 0$ (neste caso, $f(n)$ é constante).
  - $\log_b a = \log_2 1 = 0$.
- Como $f(n) = O(n^c)$ e $c < \log_b a$, pela primeira condição do método mestre, a complexidade é $T(n) = O(\log n)$.

Complexidade: A complexidade assintótica da busca binária é $O(\log n)$.

4.4 Resumo das Complexidades Assintóticas

- **Busca Sequencial:** $O(n)$
- **Busca Sequencial Otimizada:** $O(n \log n)$ (incluindo o custo de ordenação)
- **Busca Binária:** $O(\log n)$

Cada algoritmo possui diferentes perfis de desempenho baseados em suas propriedades e nas condições dos dados. A busca binária se mostra a mais eficiente entre os três algoritmos testados, especialmente para grandes conjuntos de dados. A

busca sequencial otimizada oferece vantagens em comparação com a busca sequencial simples, mas sua eficiência pode ser comprometida pelo custo da ordenação dos dados. A busca sequencial, por outro lado, demonstra uma escalabilidade pobre com o aumento do tamanho dos dados e do número de chaves, o que a torna menos adequada para grandes conjuntos de dados.

5 Discussão dos Resultados Empíricos e Experimentais

A análise teórica dos algoritmos de busca fornece uma visão valiosa sobre o comportamento assintótico esperado de cada método. No entanto, é crucial validar essas expectativas com experimentos empíricos para garantir que a teoria se alinha com a prática. Nesta seção, discutiremos como os resultados experimentais obtidos em nossos testes se comparam às análises teóricas.

5.1 Busca Sequencial

Análise Teórica: A busca sequencial possui uma complexidade assintótica de $O(n)$. Isso implica que o tempo de execução cresce linearmente com o aumento do tamanho do array e do número de chaves a serem buscadas.

Resultados Empíricos: Nossos experimentos confirmam a análise teórica, mostrando que o tempo de execução da busca sequencial aumenta linearmente com o tamanho do array. Para arrays maiores ($N = 6$ e $N = 7$), e quando o número de chaves (Q) aumenta, o tempo de execução sobe exponencialmente. Isso reflete o comportamento esperado, já que a busca sequencial precisa percorrer o array para cada chave, resultando em um crescimento significativo do tempo de execução.

Discussão: A análise teórica e os resultados empíricos estão alinhados, confirmando que a busca sequencial é eficiente para arrays pequenos e um número reduzido de chaves, mas se torna ineficiente para grandes conjuntos de dados devido ao aumento linear no tempo de execução.

5.2 Busca Sequencial Otimizada

Análise Teórica: A busca sequencial otimizada tem uma complexidade de $O(n \log n)$, considerando o custo da ordenação dos dados antes da busca. Esta abordagem visa reduzir o tempo de execução comparado à busca sequencial simples, mas a necessidade de ordenação impacta a eficiência geral.

Resultados Empíricos: Nossos testes mostram que a busca sequencial otimizada, após a ordenação, apresenta um desempenho melhor para arrays menores

comparado à busca sequencial simples. No entanto, o tempo de execução para arrays maiores ainda é substancialmente alto, especialmente quando o número de chaves é elevado. A necessidade de ordenar os dados antes da busca pode superar as melhorias obtidas pela otimização, especialmente em conjuntos de dados grandes.

Discussão: Os resultados empíricos corroboram a análise teórica. A busca sequencial otimizada mostra uma melhoria em relação à busca sequencial em termos de desempenho para arrays menores, mas o custo de ordenação pode ser um fator limitante em casos de dados grandes, resultando em um tempo de execução que pode até mesmo superar o da busca sequencial em alguns cenários.

5.3 Busca Binária

Análise Teórica: A busca binária possui uma complexidade assintótica de $O(\log n)$, o que indica que o tempo de execução cresce de forma logarítmica com o aumento do tamanho do array. Isso faz da busca binária uma abordagem muito eficiente para conjuntos de dados grandes, desde que os dados estejam ordenados.

Resultados Empíricos: Os experimentos confirmam a análise teórica, mostrando que a busca binária tem um tempo de execução significativamente menor em comparação com os outros algoritmos, mesmo para grandes arrays e um número elevado de chaves. O tempo de execução da busca binária aumenta muito mais lentamente com o aumento do tamanho do array e do número de chaves.

Discussão: Os resultados empíricos estão consistentes com a análise teórica. A busca binária se destaca como a abordagem mais eficiente entre os três algoritmos testados, confirmando que a teoria logarítmica se traduz em prática, especialmente para grandes conjuntos de dados.

6 Considerações finais

A análise teórica dos algoritmos de busca fornece uma base sólida para prever o desempenho dos algoritmos, mas os resultados empíricos são essenciais para validar essas previsões em condições práticas. Nossos experimentos confirmam a precisão das análises teóricas para todos os três algoritmos de busca. A busca binária é, de fato, a mais eficiente para grandes conjuntos de dados, conforme esperado. A busca sequencial otimizada melhora a eficiência sobre a busca sequencial simples, mas seu custo de ordenação pode limitar sua eficácia em dados grandes. A busca sequencial, embora simples e eficaz para arrays pequenos, demonstra uma escalabilidade deficiente para grandes conjuntos de dados, corroborando sua análise teórica. Esses resultados oferecem uma visão abrangente sobre a aplicabilidade de

cada algoritmo em diferentes cenários, guiando a escolha do método mais adequado com base nas características dos dados e nos requisitos de desempenho.

A busca sequencial é mais apropriada para pequenos conjuntos de dados onde a simplicidade é fundamental. Por exemplo, a busca sequencial pode ser utilizada em listas de contatos em dispositivos móveis, onde o número de entradas é relativamente pequeno, e a inserção e remoção frequentes de contatos tornam a ordenação ineficiente. A abordagem sequencial otimizada pode ser vantajosa em situações onde o conjunto de dados é moderadamente grande e a ordenação pode ser realizada de forma eficiente, por exemplo, a busca em um catálogo de produtos em um sistema de gestão de inventário de uma pequena empresa, onde as buscas são frequentes e os dados não mudam constantemente. Além disso, a busca sequencial otimizada é útil em cenários onde os dados são acessados frequentemente, mas podem ser atualizados periodicamente, como em sistemas de recomendação que utilizam listas de produtos ordenados por popularidade. Já a busca binária é ideal para grandes conjuntos de dados que são acessados frequentemente, como em sistemas de bancos de dados ou em mecanismos de pesquisa [2]. Um exemplo típico é a busca em uma tabela de índices de um grande banco de dados relacional, onde a busca binária permite localizar rapidamente registros específicos em conjuntos de dados vastos. Outro exemplo é o uso da busca binária em algoritmos de compressão de dados, onde grandes volumes de informações precisam ser processados de forma rápida e eficiente.

Referências

[1] Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2009). *Introduction to Algorithms* (3rd ed.). MIT Press.

[2] Khan Academy. (n.d.). *Binary Search*. Retrieved from `https://www.khanacademy.org/computing/computer-science/algorithms/binary-search/a/implementing-binary-search-of-an-array`

[3] Github *Repositório dos algoritmos*. `https://github.com/CesarTHD/Algoritmos-de-busca`