Comparação da Eficiência dos Algoritmos de Busca Sequencial Padrão, Busca por Saltos, Busca Binária, Busca em Listas Ligadas e Busca em Árvores Binárias

Marcos Alexandre dos Anjos[1]

[1]Programa de Pós Graduação em Ciência da Computação - PPGCOMP Universidade Estadual do Oeste do Paraná (UNIOESTE)
Caixa Postal 85.819-110 – Cascavel– PR – Brasil

dev.marcosanjos@gmail.com

Abstract. *This article empirically analyzes the efficiency of three search algorithms on static arrays with integer values: standard sequential search, jump search, binary search, linked list search, and binary tree search. The database was generated with arrays ranging from 100,000 to 1 million values, with increments of 100,000 positions and random numbers. The algorithms were subjected to the same search scenarios, including worst-case and random search, with three executions for each algorithm. We can conclude that binary tree search stands out as the most efficient and fastest algorithm for searching hierarchical data structures. This research can be useful in helping developers choose the most suitable search algorithm for their specific needs.*

Resumo. *Este artigo analisa empiricamente a eficiência de três algoritmos de busca em arranjos estáticos com valores inteiros: busca sequencial padrão, busca por saltos (Jump search), busca binária, busca em listas ligadas e busca em árvores binárias. A base de dados foi gerada com arranjos de valores variando de 100 mil até 1 milhão, com intervalos de 100 mil posições com números aleatórios. Os algoritmos foram submetidos aos mesmos cenários de busca, incluindo pior caso, caso busca aleatória, com três execuções de cada algoritmo. Podemos concluir que a busca em árvores binárias se destaca como o algoritmo mais eficiente e rápido para realizar buscas em estruturas de dados hierárquicas. Esta pesquisa pode servir para ajudar desenvolvedores a escolher o algoritmo de busca mais adequado para suas necessidades específicas.*

1. Introdução

O desenvolvimento e análise de algoritmos compreende nas dimensões de corretude e na complexidade. A análise de corretude pretende determinar se o algoritmo chega à solução desejada, enquanto a análise de complexidade determina qual será o custo desse algoritmo na busca pelo problema. Esses passos são importantes no desenvolvimento dos algoritmos, e por isso são estudados nas disciplinas de Estrutura de Dados e Análise de Algoritmos em cursos de computação.

Neste sentido, o presente trabalho propõe um estudo sobre os algoritmos de busca e uma análise de custo apresentado qual a eficiência, podendo de forma compará-los variando o tamanho do arranjo da amostra.

2. Referencial Teórico

O referencial teórico deste estudo abrange os conceitos e princípios fundamentais das técnicas de busca sequencial padrão, busca por saltos (jump search), busca binária, busca em listas ligadas e busca em árvores binárias bem como as suas respetivas complexidades computacionais e limitações.

2.1. Algoritmo de Busca Sequencial padrão

Trata-se do método de busca mais simples. Dado um conjunto de números, letras, endereços ou independentemente do tipo de variável existente num programa, basta percorrer e comparar um a partir do primeiro elemento da lista. Assim esse elemento é encontrado, o seu índice é retornado e a busca é finalizada. Caso não exista o elemento no conjunto, o programa retorna algum valor que não possa ser conjunto com índice indicado que não foi encontrado (Ziviani, 2004).

Analisando o algoritmo de busca sequencial, tempos que no seu melhor caso (a) elemento na primeira posição e o algoritmo possui complexidade O(1); (b) no caso médio o elemento na posição central tem complexidade O(n/2); e (c) o pior caso é na última posição apresenta complexidade igual a O(n) (Ziviani, 2004).

2.2. Algoritmo de Busca por saltos (*Jump search*)

O algoritmo de busca Jump Search, também conhecido como Busca por Salto, é um algoritmo de busca numa lista ordenada que utiliza saltos para percorrer a lista de forma mais eficiente que uma busca sequencial. O algoritmo é baseado numa ideia simples de saltar a uma posição específica na lista em vez de percorrê-la item por item.

Conforme o artigo "Jump Search Algorithm: A Review" de Ajay Kumar Pandey e Mukesh Kumar Vishwakarma, publicado na International Journal of Advanced Research in Computer Science and Software Engineering em 2013, o algoritmo de busca Jump Search possui uma complexidade de tempo de O(sqrt(n)), onde n é o tamanho da lista. Isso o torna mais eficiente do que a busca sequencial em grandes listas, enquanto ainda é fácil de implementar e requer uma quantidade razoável de memória (Pandey and Vishwakarma, 2013). O artigo também discute algumas variações e otimização do algoritmo de busca Jump Search, como o uso de interpolação para escolher o tamanho do salto e a verificação da frequência de saltos para melhorar ainda mais o desempenho.

2.3. Algoritmo de Busca Binária

O algoritmo de busca binária é um método eficiente de encontrar um elemento em um conjunto ordenado de dados. O algoritmo divide repetidamente o conjunto ao meio e verifica se o elemento buscado está na metade esquerda ou direita. Esse processo é repetido até que o elemento seja encontrado ou a metade atual, seja reduzida a zero.

Conforme o artigo "A Binary Search Implementation for Embedded Systems" (2019), a busca binária tem uma complexidade de tempo de O(log n), onde n é a composição no conjunto de dados (Ali, 2019). Isso significa que, para grandes conjuntos de dados, a busca binária é significativamente mais rápida do que a busca sequencial. Além disso, o artigo "Analysis of Binary Search Algorithm" (2017) apresenta uma análise matemática mais detalhada da eficiência do algoritmo de busca binária (Rahman

and Islam, 2017).

2.4. Algoritmo de Busca Listas Ligadas

A lista ligada é uma estrutura de dados amplamente utilizada em algoritmos de busca. Um algoritmo de busca em lista ligada é capaz de percorrer elementos sequencialmente até encontrar o valor desejado. Um algoritmo de busca em lista ligada tem complexidade de tempo O(n), onde n é o tamanho da lista" (Gonnet and Baeza-Yates, 1991). Isso significa que a eficiência deste algoritmo depende diretamente do tamanho da lista a ser percorrida. No entanto, é importante destacar que a lista ligada oferece vantagens como a inserção e exclusão eficientes de elementos. Portanto, embora a busca em lista ligada possa ser menos eficiente em termos de tempo, ela pode ser uma escolha viável para cenários em que a manipulação da estrutura de dados é frequente.

2.5. Algoritmos de Busca Árvores Binária

A árvore binária é uma estrutura de dados amplamente utilizada em algoritmos de busca. Um algoritmo de busca em árvores binárias é capaz de explorar eficientemente os elementos da estrutura, utilizando a propriedade de ordem entre os nós. "Os algoritmos de busca em árvores binárias podem ter complexidade de tempo O(h), onde h é a altura da árvore" (Cormen et al., n.d.). Isso significa que, em uma árvore balanceada, a busca pode ser realizada de forma eficiente. Além disso, a estrutura da árvore binária permite a implementação de operações de inserção e remoção de maneira eficaz. Dessa forma, os algoritmos de busca em árvores binárias são amplamente utilizados em diversas aplicações, como busca de elementos, ordenação e construção de índices.

3. Metodologia

Neste trabalho, o objetivo é comparar empiricamente a eficiência de três algoritmos de busca em arranjos estatísticos com valores inteiros. Os algoritmos escolhidos são a busca sequencial padrão, busca por saltos (jump search), busca binária, busca lista ligada e busca em árvores binárias.

Para isso, foi implementado os algoritmos em código em Python contendo vários scripts necessários para análise da proposta. Inicialmente, foi criado um script para gerar a base de dados, que consiste em arranjos de valores variando de 100 mil até 1 milhão, com intervalos de 100 mil posições com números aleatórios.

Segunda etapa do processo, foi realizado uma análise dos algoritmos de busca, e realizado a implementação somado aos scripts para controle de repetição. Onde cada algoritmo foi realizado o mesmo teste três vezes, para obter um tempo médio. Para ser possível realizar os cálculos de média, desvio padrão. Os três algoritmos foram submetidos aos mesmos cenários, que são eles: (a) pior caso, com três execuções de cada algoritmo; (b) caso aleatório com cem buscas para cada cenário.

É importante destacar que o custo de criação do arranjo e cálculo desvio padrão, criação de arquivos, tempo de ordenação foi descartado, foco principal está na análise do tempo de busca.

Para implementar os algoritmos de busca e de ordenação, foi escolhida a linguagem de programação Python na versão 3.11.3 e o ambiente de desenvolvimento usando a IDE

VSCode, o tempo de execução dos algoritmos foi avaliado em segundos, utilizando um notebook do modelo Avel Liv, com sistema operacional Windows 11 Pro Insider Preview, equipado com um processador i7 - 1050H, 16GB de memória DDR4 e com SSD com 250 GB.

4. Resultados

4.1. Algoritmo de Busca Sequencial

Os resultados obtidos pelo algoritmo de busca sequencial estão apresentados na Tabela 2 (Anexos). Podemos comparar os cenários A e B e observar que o custo de comparações desse algoritmo é extremamente alto. No caso A, o custo cresce proporcionalmente à quantidade de valores presentes na lista. Já no caso B, que envolve a busca por um valor aleatório, os resultados são dispersos, podendo apresentar resultados menos custosos em alguns casos.

Entretanto, ao se observar a média, o custo da busca é alto, já que muitas comparações precisam ser feitas. Na Figura 1, podemos ter uma visualização da comparação dos dois cenários, em azul representa cenário A e vermelho representa cenário B.

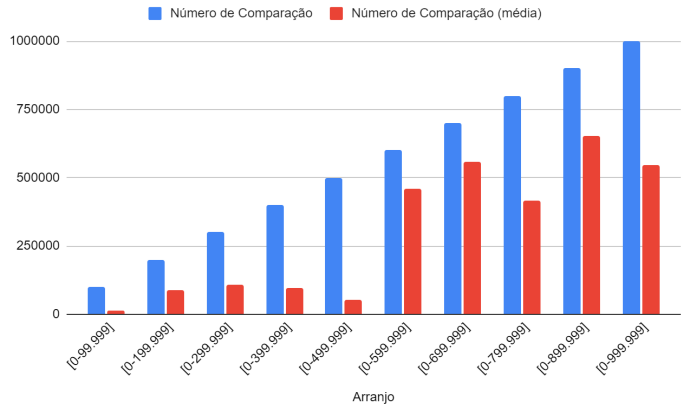

Figura 1. Gráfico relação, número de comparação case A vs número comparação case B.

4.2. Algoritmo de Busca por Saltos (jump search)

O algoritmo de busca por saltos, também conhecido como jump search, apresentou um desempenho superior relativamente ao algoritmo de busca sequencial. Analisando os dados da Tabela 3 (Anexos), os custos deste algoritmo, observando o número de comparações necessárias para realizar as buscas nos cenários A e B, os resultados foram surpreendentes. No caso A, foram necessárias entre 17 a 20 comparações para realizar a busca. Já no caso B, os resultados foram semelhantes, ficando entre 15 a 19 comparações. Isso nos permite entender que este algoritmo apresenta alta desempenho de busca.

Para uma melhor visualização a Figura 2, apresenta uma comparação dos cenários A e B com as suas respetivas cores azul e vermelho. Em que podemos observar

visualmente que a case A teve um número de comparações superiores ao case B. Mas mesmo assim os resultados são próximos.

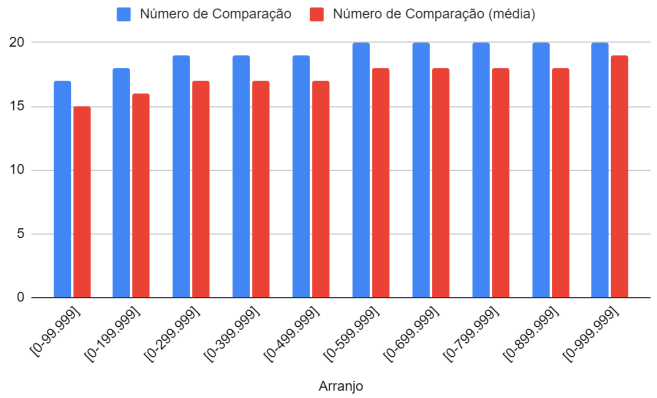

Figura 2. Gráfico relação, número de comparação case A vs número comparação case B.

Embora tenha uma alta desempenho, um ponto fraco deste algoritmo é sua complexidade para implementação. Para obter resultados de tempo mais palpáveis, foi adicionado um sleep de 0.001 segundo. No entanto, em comparação com o algoritmo de busca sequencial, o jump search se mostrou muito mais eficiente na busca de elementos numa lista ordenada.

4.3. Algoritmo de Busca Binária

Ao analisar os resultados da Tabela 4 (Anexos), é possível constatar que o algoritmo de busca binária apresentou um desempenho surpreendente em ambos os cenários, quando comparado ao algoritmo de busca por saltos. Observando a média de comparações necessárias para realizar a busca, podemos perceber que o algoritmo de busca binária obteve resultados muito eficientes. No entanto, é importante ressaltar que no cenário A, a média de comparações foi ligeiramente superior em comparação ao cenário B. Isso ocorre porque, no cenário B, os dados são gerados aleatoriamente, o que pode influenciar diretamente na relação entre o número de comparações e o tempo médio de execução.

A implementação do algoritmo de busca binária é relativamente simples, mas seu desempenho é bastante eficiente na busca por listas ordenadas. Devido à sua alta desempenho, foi adicionado um sleep de 0.001 segundos para apresentar uma medida de tempo mais fácil de entender e visualizar. Com essa medida, é possível observar que o algoritmo de busca binária apresenta resultados ótimos em termos de tempo de execução e número de comparações realizadas, principalmente quando comparado aos algoritmos de busca sequencial e busca por saltos. Na Figura 3, podemos ter uma visualização sobre o tempo case A e B com as suas respetivas cores azul e vermelho. Onde que temos número de compará-los são relativamente pequenas, quando comparamos o mesmo gráfico com busca sequencial.

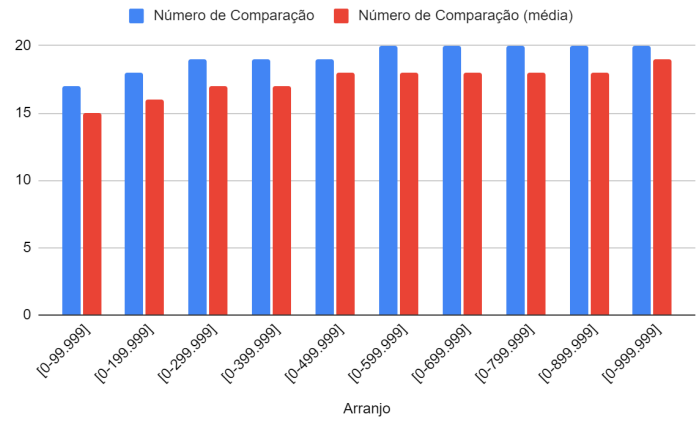

Figura 3. Gráfico relação, número de comparação case A vs número comparação case B.

5. Algoritmo de Busca em Lista Ligada

No cenário A, considerado o pior caso para busca em listas ligadas, Tabela 5 (Anexo), a busca se mostrou mais lenta e demorada em comparação com a busca sequencial. Ambos os métodos verificam toda a lista, resultando num tempo proporcional ao tamanho total da lista. No cenário B, foram gerados valores aleatórios para realizar 100 buscas num arranjo totalmente ordenado. Nesse caso, a busca em listas obteve resultados melhores em alguns casos. Isso ocorreu porque o valor procurado foi encontrado em situações mais favoráveis. Geralmente, as buscas em listas tendem a ser mais lentas e demoradas, mas em casos específicos podem superar a busca sequencial, dependendo dos valores gerados aleatoriamente. A Figura 4 apresenta os resultados de custo de tempo de execução e comparações para ambos os cenários.

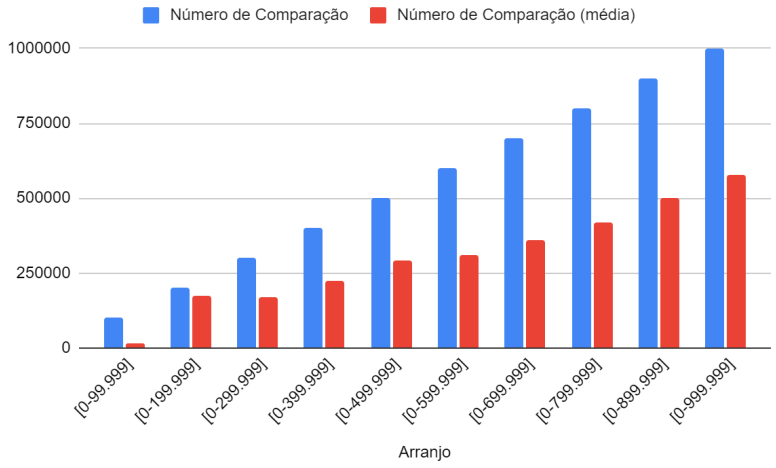

Figura 4. Gráfico relação, número de comparação case A vs número comparação case B.

A busca em listas ligadas é geralmente mais lenta que a busca sequencial. No

cenário A, a busca em listas teve um desempenho pior em termos de tempo. No cenário B, a busca em listas teve resultados variados, podendo ser melhor em alguns casos. O tempo de execução e o número de comparações aumentam com o tamanho do arranjo. A busca em listas ligadas pode ser mais lenta em arranjos maiores, mas em casos aleatórios pode ter um desempenho melhor.

6. Algoritmo em Busca em Árvore Binária

Observando os resultados da Tabela 6 (Anexos), podemos fazer algumas observações e conclusões: para o Case A, em que o arranjo possui valores sequenciais de 0 a 99.999, podemos notar que o tempo de execução médio é próximo de 1 segundo, com um desvio padrão bastante baixo. O número de comparações se mantém em torno de 15 para todos os arranjos analisados. Isso indica que a busca em árvores binárias é eficiente e consistente nesse cenário, independentemente do tamanho do arranjo. No Case B, em que foram gerados valores aleatórios para cada busca num arranjo desordenado, podemos observar resultados semelhantes. O tempo de execução médio é próximo de 1 segundo, também com um desvio padrão baixo. O número de comparações varia um pouco mais do que no Case A, mas ainda assim, a média fica em torno de 15 a 25 comparações.

Comparando os números de comparações nos dois cases como mostra Figura 5, podemos observar que, apesar de uma ligeira variação no Case B devido à desordem dos valores, a busca em árvores binárias continua apresentando um desempenho consistente. Em ambos os casos, os números de comparações são relativamente baixos, indicando uma eficiência na busca dos valores desejados.

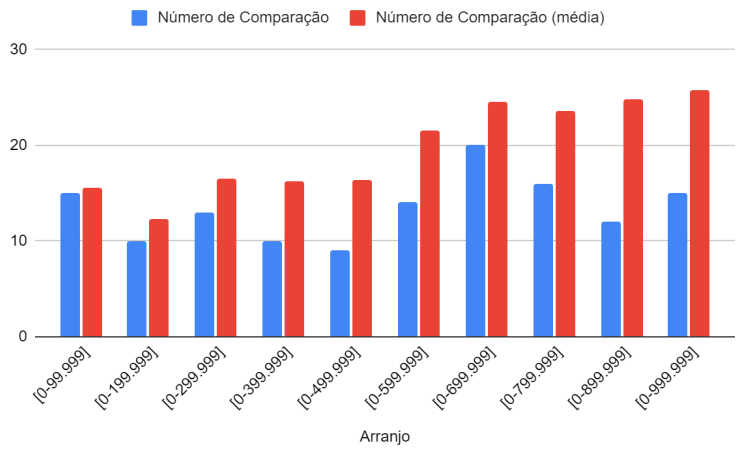

Figura 5. Gráfico relação, número de comparação case A vs número comparação case B.

Esses resultados mostram que a busca em árvores binárias é eficiente tanto em arranjos ordenados quanto em arranjos desordenados. A estrutura da árvore permite uma busca rápida e consistente, independentemente da distribuição dos valores. Comparando os resultados dos dois casos, podemos afirmar que a busca em árvores binárias obteve

desempenho muito melhor do que as outras opções de busca mencionadas anteriormente. Tanto em termos de tempo de execução quanto de comparação, a busca em árvores binárias se destaca como uma opção eficiente e versátil.

Em suma, os resultados da tabela reforçam a conclusão de que a busca em árvores binárias é uma excelente escolha para realizar buscas em arranjos, independentemente da sua ordenação, garantindo um desempenho rápido e consistente.

7. Resultados da Ordenação dos Arranjos

Conforme mencionado anteriormente, uma das observações feitas no artigo foi a necessidade de ordenar os valores para algumas buscas. Para isso, foi utilizado o método de ordenação sort, conhecido por sua eficiência na ordenação de arrays num curto período.

Em Python, o método sort() da classe list utiliza o algoritmo Timsort para realizar a ordenação dos elementos. O Timsort é um algoritmo de ordenação híbrido que combina as técnicas do merge sort e do insertion sort. Ele foi projetado para ter um desempenho eficiente em diferentes cenários, adaptando-se bem a arrays parcialmente ordenados e desordenados. O Timsort é o algoritmo de ordenação padrão utilizado em Python devido à sua eficiência e capacidade de lidar com uma variedade de situações.

Os resultados dessa ordenação podem ser vistos na Tabela 7.

Arranjo	Tempo de Execução Médio	Consumo de memória (MB)
[0-99.999]	0,309970	18,35
[0-199.999]	0,416984	23,46
[0-299.999]	0,067024	26,19
[0-399.999]	0,0812	31,30
[0-499.999]	0,104645	35,55
[0-599.999]	0,142831	38,54
[0-699.999]	0,287991	41,67
[0-799.999]	0,34563	46,70
[0-899.999]	0,301936	49,26
[0-999.999]	0,238719	54,35

Tabela 7: Ordenação dos arranjos utilizando o método sort da linguagem do python e apresentando o consumo memória ram.

8. Conclusão

Após a realização dos experimentos, podemos concluir que o algoritmo de busca sequencial é o menos eficiente em termos de desempenho. O custo de comparações é proporcional à quantidade de valores presentes na lista, o que o torna inviável para grandes conjuntos de dados.

Já o algoritmo de busca por saltos apresentou um desempenho superior relativamente ao sequencial, realizando uma quantidade significativamente menor de comparações. Isso o torna uma opção mais viável para grandes conjuntos de dados. Porém, a sua implementação é um pouco mais complexa do que a busca binária.

Falando na busca binária, este algoritmo apresentou um desempenho surpreendente em ambos os cenários. A quantidade média de comparações realizadas foi significativamente menor do que a da busca sequencial, e em alguns casos até mesmo menor do que a da busca por saltos. Além disso, a sua implementação é relativamente simples, o que o torna uma opção bastante viável em diversas situações.

Na busca em listas ligadas pode ser adequada para casos em que a estrutura de dados é uma lista e não é necessário realizar buscas frequentes ou em listas de tamanho muito grande. Para casos em que a eficiência e o desempenho são cruciais, especialmente em buscas frequentes ou em listas de tamanho considerável, a busca em árvores binárias é uma opção mais eficiente e preferível. A estrutura organizada das árvores binárias permite uma busca mais rápida e com menor número de comparações, garantindo um melhor desempenho geral.

A busca em árvores binárias é um algoritmo eficiente, rápido e versátil para realizar buscas em estruturas de dados hierárquicos. A sua complexidade de tempo logarítmica e a organização dos valores na árvore garantem um desempenho superior em comparação a outras opções de busca. Portanto, a busca em árvores binárias é uma escolha sólida para aplicativos que exigem eficiência e desempenho em operações de busca.

Podemos concluir que a busca em árvores binárias se destaca como o algoritmo mais eficiente e rápido para realizar buscas em estruturas de dados hierárquicos. Comparada à busca sequencial e à busca por saltos, a busca binária apresentou um desempenho significativamente superior, realizando menos comparações e garantindo um tempo de execução mais eficiente. A sua simplicidade de implementação e a capacidade de lidar com conjuntos de dados grandes a tornam uma escolha sólida em termos de eficiência e desempenho na realização de operações de busca.

Referências

Boulic, R. and Renault, O. (1991) "3D Hierarchies for Animation", In: New Trends in Animation and Visualization, Edited by Nadia Magnenat-Thalmann and Daniel Thalmann, John Wiley & Sons ltd., England.

Ali, k, 2019. A Binary Search Implementation for Embedded Systems. nternational Journal of Computer Science and Network Security 19, 52–55.

Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C., 2011. Mestrado Acadêmico em Ciência da Computação Disciplina Projeto e Análise de Algoritmos (01.803. 1) Carga Horária Créditos Obrigatória 60 horas/aula 4 SIM.

Gonnet, G.H., Baeza-Yates, R., 1991. Handbook of algorithms and data structures: in Pascal and C. Addison-Wesley Longman Publishing Co., Inc.

Pandey, A.K., Vishwakarma, M.K., 2013. Jump Search Algorithm: A Review. International Journal of Advanced Research in Computer Science and Software Engineering 777–780.

Rahman, M.F., Islam, M.A., 2017. Analysis of Binary Search Algorithm. International Journal of Scientific and Research Publications 231–234.

Ziviani, N., 2004. Projeto de algoritmos: com implementações em Pascal e C. Thomson Luton.

Dyer, S., Martin, J. and Zulauf, J. (1995) "Motion Capture White Paper", http://reality.sgi.com/employees/jam_sb/mocap/MoCapWP_v2.0.html, December.

Holton, M. and Alexander, S. (1995) "Soft Cellular Modeling: A Technique for the Simulation of Non-rigid Materials", Computer Graphics: Developments in Virtual Environments, R. A. Earnshaw and J. A. Vince, England, Academic Press Ltd., p. 449-460.

Knuth, D. E. (1984), The TeXbook, Addison Wesley, 15th edition.

Smith, A. and Jones, B. (1999). On the complexity of computing. In *Advances in Computer Science*, pages 555–566. Publishing Press.

Anexos

Tabela 2. Resultados da busca por Sequencial Aplicado nos Cenários A e B

	Case A			Case B		
Arranjo	**Tempo de Execução Médio**	**Desvio Padrão**	**Número de Comparação**	**Tempo de Execução Médio**	**Desvio Padrão**	**Número de Comparação (média)**
[0-99.999]	0,00469	0,000581	100000	0,00278	0,00008	13122
[0-199.999]	0,01065	0,001161	200000	0,00561	0,00021	89166
[0-299.999]	0,01541	0,001655	300000	0,00771	0,00062	107146
[0-399.999]	0,02119	0,001330	400000	0,01140	0,00092	94739
[0-499.999]	0,02819	0,001405	500000	0,01392	0,00142	53554
[0-599.999]	0,03066	0,001522	600000	0,01636	0,00067	460920
[0-699.999]	0,03633	0,002517	700000	0,02034	0,00159	557652
[0-799.999]	0,04236	0,002469	800000	0,02297	0,00160	416054
[0-899.999]	0,04816	0,003019	900000	0,02374	0,00127	652826
[0-999.999]	0,04901	0,001505	1000000	0,02760	0,00092	546230

Tabela 3. Resultados da Busca por Saltos Aplicado nos Cenários A e B

	Case A			Case B		
Arranjo	**Tempo de Execução Médio**	**Desvio Padrão**	**Número de Comparação**	**Tempo de Execução Médio**	**Desvio Padrão**	**Número de Comparação (média)**
[0-99.999]	0,36978	0,01242	17	0,50088	0,01584	15
[0-199.999]	0,67416	0,15786	18	0,70114	0,06839	16
[0-299.999]	0,78894	0,02179	19	0,77301	0,10126	17
[0-399.999]	0,90170	0,00720	19	1,03045	0,03086	17
[0-499.999]	1,00172	0,03159	19	1,11747	0,07748	17
[0-599.999]	1,12439	0,02475	20	1,11025	0,03356	18
[0-699.999]	1,21971	0,02933	20	1,24855	0,04266	18

[0-799.999]	1,23653	0,00446	20	1,28364	0,05018	18
[0-899.999]	1,16488	0,08673	20	1,35538	0,02959	18
[0-999.999]	1,33264	0,03699	20	1,39686	0,02817	19

Tabela 4. Resultados da Busca por Binária Aplicado nos Cenários A e B

	Case A			**Case B**		
Arranjo	**Tempo de Execução Médio**	**Desvio Padrão**	**Número de Comparação**	**Tempo de Execução Médio**	**Desvio Padrão**	**Número de Comparação (média)**
[0-99.999]	0,00192	0,00046	17	0,02147	0,00041	15
[0-199.999]	0,00174	0,00034	18	0,02420	0,00101	16
[0-299.999]	0,00150	0,00012	19	0,02425	0,00138	17
[0-399.999]	0,00236	0,00011	19	0,02441	0,00022	17
[0-499.999]	0,00157	0,00047	19	0,02422	0,00077	18
[0-599.999]	0,00210	0,00015	20	0,02423	0,00046	18
[0-699.999]	0,00197	0,00040	20	0,02573	0,00037	18
[0-799.999]	0,00191	0,00021	20	0,02493	0,00023	18
[0-899.999]	0,00166	0,00060	20	0,02708	0,00130	18
[0-999.999]	0,00133	0,00026	20	0,02590	0,00018	19

Tabela 5. Resultados da Busca em Listas Ligadas nos Cenários A e B

	Case A			Case B		
Arranjo	Tempo de Execução Médio	Desvio Padrão	Número de Comparação	Tempo de Execução Médio	Desvio Padrão	Número de Comparação (média)
[0-99.999]	0,021120	0,001095	100000	0,0176	0,01682	16080,01
[0-199.999]	0,057517	0,015260	200000	0,05389	0,06493	175380,85
[0-299.999]	0,050542	0,011500	300000	0,05962	0,10628	168745,08
[0-399.999]	0,068864	0,000318	400000	0,02357	0,03068	223952,5
[0-499.999]	0,074687	0,286300	500000	0,10862	0,0569	290378,75
[0-599.999]	0,143364	0,532000	600000	0,08627	0,03356	309642,04
[0-699.999]	0,134268	0,076523	700000	0,06521	0,05782	361168,2
[0-799.999]	0,081556	0,213100	800000	0,0691	0,05068	420610,8
[0-899.999]	0,936658	0,105920	900000	0,16399	0,0258	499735,7
[0-999.999]	0,997528	0,462310	1000000	0,11065	0,02319	578237

Tabela 6 . Resultados da Busca em Árvore Binária nos Cenários A e B

	Case A			Case B		
Arranjo	Tempo de Execução Médio	Desvio Padrão	Número de Comparação	Tempo de Execução Médio	Desvio Padrão	Número de Comparação (média)
[0-99.999]	1,000611	0,001000	15	1,000518	0,00282	15,55
[0-199.999]	1,001026	0,000269	10	1,000611	0,000352	12,23
[0-299.999]	1,000107	0,000185	13	1,000508	0,000339	16,56
[0-399.999]	1,001947	0,002860	10	1,000584	0,000328	16,19
[0-499.999]	1,00591	0,000026	9	1,000555	0,00342	16,41
[0-599.999]	1,002182	0,005680	14	1,001269	0,000356	21,55
[0-699.999]	1,00362	0,002832	20	1,000692	0,000361	24,49
[0-799.999]	1,00754	0,000128	16	1,000613	0,000349	23,63
[0-899.999]	1,000897	0,008315	12	1,000625	0,000344	24,88
[0-999.999]	1,001323	0,003200	15	1,000578	0,000323	25,78