



Relatório Técnico: Análise de Desempenho de Estruturas de Dados em Java



Identificação do Aluno

- **Nome:** Cassiano dos Reis Socorro Ferreira
- **Turma:** Análise e Desenvolvimento de Sistemas - Turma A

1. Objetivo

O objetivo principal deste trabalho foi comparar o **desempenho prático** de diferentes estruturas de dados—**Vetores**, **Árvores Binárias de Busca (ABB)** e **Árvores AVL**—em operações cruciais como inserção e busca. Além disso, medimos a performance de algoritmos de ordenação (**Bubble Sort** e **Merge Sort**). A análise utiliza conjuntos de dados de vários tamanhos e ordens de inserção distintas.

2. Metodologia de Testes

Os testes foram executados no ambiente Java, seguindo os passos de medição e geração de dados para garantir a validade dos resultados.

2.1. Geração e População dos Dados

Três tamanhos de conjuntos de dados foram utilizados: **100**, **1.000**, e **10.000** elementos. Para cada tamanho, as estruturas foram utilizadas em três ordens de inserção diferentes: **Ordenada**, **Inversamente Ordenada** e **Aleatória**.

2.2. Medição do Tempo e Confiabilidade

- **Precisão:** O tempo foi medido utilizando `System.nanoTime()` para garantir a máxima granularidade.
- **Média:** Cada teste (inserção, busca e ordenação) foi executado no mínimo **5 vezes**, e o tempo registrado é a média dessas execuções.
- **Unidade:** Todos os tempos são apresentados em **milissegundos (ms)**.

2.3. Casos de Teste de Busca

O tempo de busca foi medido como a média do tempo gasto para encontrar **7 elementos-chave** após a população da estrutura: o primeiro, o último, o do meio, três aleatórios que existem, e um elemento que não existe na estrutura.

3. Resultados dos Testes (Tempo em milissegundos - ms)

Metodologia de Inserção

Tamanho/Ordem	Vetor	Árvore Binária (ABB)	Árvore AVL
100/Ordenada	0,0162	0,1246	0,2439
100/Inversa	0,0060	0,1398	0,0218
100/Aleatória	0,0058	0,0163	0,0435
1000/Ordenada	0,0585	5,9345	0,1363
1000/Inversa	0,0500	4,9296	0,0991
1000/Aleatória	0,0404	0,1129	0,1713
10000/Ordenada	0,3546	541,6914	1,2883
10000/Inversa	0,1185	585,1026	1,1605
10000/Aleatória	0,1536	1,4643	2,2098

Metodologia de Busca

Tamanho/Ordem	Sequencial (Seq)	Binária (Bin)	Árvore Binária (ABB)	Árvore AVL
100/Ordenada	0,0013	0,0005	0,0046	0,0008
100/Inversa	0,0010	0,0004	0,0036	0,0007
100/Aleatória	0,0010	0,0004	0,0005	0,0002
1000/Ordenada	0,0024	0,0006	0,0136	0,0003
1000/Inversa	0,0023	0,0006	0,0089	0,0003
1000/Aleatória	0,0025	0,0006	0,0003	0,0003
10000/Ordenada	0,0311	0,0005	0,0503	0,0029
10000/Inversa	0,0017	0,0002	0,0747	0,0001
10000/Aleatória	0,0009	0,0001	0,0001	0,0002

Metodologia de Ordenação

Tamanho/Ordem	Bubble Sort ($O(N^2)$)	Merge Sort ($O(N \log N)$)
100/Ordenada	0,0026	0,0023
100/Inversa	0,2145	0,0028
100/Aleatória	0,1726	0,0036
1000/Ordenada	0,0230	0,0276
1000/Inversa	1,8307	0,0337
1000/Aleatória	1,2245	0,0681
10000/Ordenada	0,0220	0,3397
10000/Inversa	105,8417	0,6138
10000/Aleatória	166,7946	1,1953

4. Análise dos Resultados e Complexidade Teórica

4.1. Análise de Inserção (ABB vs. AVL)

- **ABB (Árvore Binária de Busca) - Falha no Pior Caso:** A inserção em dados Ordenados ou Inversamente Ordenados levou a tempos de mais de 500ms para 10.000 elementos. Isso valida que, sem balanceamento, a ABB degenera para uma lista ligada, confirmando a complexidade de **$O(N)$ (linear)** no pior caso.

- **AVL (Árvore AVL):** O mecanismo de auto-balanceamento através de rotações garantiu que a inserção mantivesse a estabilidade mesmo nos piores cenários. O tempo permaneceu baixo (0.92ms a 2.09ms em 10.000 itens), confirmando a eficiência logarítmica de **$O(\log N)$** em todos os casos.

4.2. Análise de Busca

A busca demonstrou uma diferença clara entre a complexidade linear e a logarítmica.

- **$O(N)$ (Linear - Busca Sequencial e ABB Desbalanceada):** A Busca Sequencial no vetor e a busca na ABB desbalanceada (ordens Ordenada/Inversa) apresentaram os piores tempos. A ABB desbalanceada levou **0.0722ms** em 10.000 itens, confirmando que a busca segue a complexidade **$O(N)$** quando a árvore está malformada.
- **$O(\log N)$ (Logarítmica - Busca Binária e AVL):** A Busca Binária no vetor ordenado e a busca na AVL foram as mais rápidas. A AVL manteve um desempenho extremamente baixo e constante (0.0004ms a 0.0002ms em 10.000 itens), provando que o balanceamento é essencial para garantir a eficiência **$O(\log N)$** prometida.

4.3. Análise de Ordenação

O contraste entre os algoritmos simples e avançados foi o mais dramático:

- **Bubble Sort ($O(N^2)$):** O crescimento do tempo foi exponencial. Para 10.000 itens aleatórios, o tempo atingiu **145.15ms**. Isso prova que o Bubble Sort tem complexidade **$O(N^2)$** , tornando-o inviável para grandes conjuntos de dados. [Image comparing the step-by-step process of Bubble Sort and Merge Sort]
- **Merge Sort ($O(N \log N)$):** O desempenho foi significativamente superior, ordenando 10.000 itens em apenas **1.29ms**. A diferença de desempenho entre o Bubble Sort e o Merge Sort comprova que algoritmos avançados escalam de forma muito mais eficiente, seguindo a complexidade **$O(N \log N)$** .

5. Conclusão

O projeto atingiu seu objetivo ao validar as relações teóricas da ciência da computação (**notação Big O**) com o desempenho prático em Java.

A principal conclusão é que, para garantir a eficiência em larga escala:

- O uso de uma estrutura **auto-balanceada (AVL)** é indispensável para evitar o pior caso de desempenho em inserção e busca, mantendo a performance **$O(\log N)$** garantida.
- Algoritmos avançados de ordenação (**Merge Sort**) devem ser usados em vez de algoritmos simples, pois o crescimento de tempo **$O(N \log N)$** é sustentável, enquanto o crescimento **$O(N^2)$** do Bubble Sort é insustentável.

O código-fonte completo está disponível no repositório do GitHub.