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

Aluno: Arthur Lourenço Fritz (1-23-16114)
Disciplina: Estrutura de Dados
Professor: Flavio Motta

3 de dezembro de 2025

1 Objetivo do Trabalho

O objetivo principal deste trabalho foi comparar o desempenho de três estruturas de dados fundamentais (Vetor, Árvore Binária de Busca - ABB, e Árvore AVL) em operações de inserção e busca. A análise foi realizada através da medição de tempo, utilizando conjuntos de dados de diversos tamanhos (N) e ordens de inserção distintas.

Adicionalmente, foi analisada a eficiência de **quatro algoritmos de ordenação distintos** para vetores, com foco comparativo entre o Bubble Sort (simples, $\mathcal{O}(N^2)$) e o Merge Sort (avançado, $\mathcal{O}(N \log N)$), com o intuito de relacionar o comportamento prático com a complexidade teórica (Notação Big O) de cada operação.

2 Metodologia

2.1 Implementação e Estruturas

Todas as estruturas de dados (Vetor, ABB, AVL) e algoritmos de busca e ordenação foram implementados ”do zero” em Java.

- A Árvore AVL incluiu a lógica completa de rotações (simples e dupla) para garantir o autobalanceamento.
- Os algoritmos de ordenação implementados foram **Bubble Sort**, **Insertion Sort**, **Merge Sort** e **Quick Sort** (com otimizações como Mediana de Três e abordagem iterativa).

2.2 Geração dos Conjuntos de Dados

Os testes foram executados para três tamanhos de conjuntos de dados: $N = 100$, $N = 1.000$ e $N = 10.000$ elementos. Para cada tamanho, foram utilizadas três ordens de inserção: Ordenada (crescente), Inversamente Ordenada (decrescente) e Aleatória.

2.3 Medição e Coleta de Tempo

Para garantir a precisão e confiabilidade, cada teste (inserção, busca e ordenação) foi executado cinco vezes ($5\times$). O tempo registrado em milissegundos (ms) representa a média dessas 5 execuções.

Os testes de busca mediram o **tempo total necessário para executar um bloco de 7 operações de busca distintas** por execução: encontrar o primeiro, o último e o elemento do meio, três elementos aleatórios que existem e um elemento que não existe na estrutura, conforme a especificação do trabalho.

3 Resultados Obtidos

Os tempos médios de execução coletados são apresentados nas tabelas a seguir.

4 Análise dos Resultados

4.1 Relação entre Complexidade Teórica (Big O) e Inserção

A complexidade teórica da inserção em árvores binárias é $\mathcal{O}(h)$, onde h é a altura da árvore.

Tabela 1: Tempos Médios de Inserção (ms)

Tamanho (N)	**Ordem**	**Vetor**	**ABB**	**AVL**
100	Ordenada	$0,016$	$0,346$	$0,156$
100	Inversa	$0,008$	$0,879$	$0,042$
100	Aleatória	$0,008$	$0,016$	$0,025$
1.000	Ordenada	$0,082$	$1,539$	$0,336$
1.000	Inversa	$0,072$	$1,346$	$0,160$
1.000	Aleatória	$0,073$	$0,160$	$0,223$
10.000	Ordenada	$0,411$	$142,344$	$1,204$
10.000	Inversa	$0,012$	$133,813$	$1,123$
10.000	Aleatória	$0,045$	$1,041$	$1,256$

Tabela 2: Tempos Médios de Busca (ms) para $N = 10.000$ (Bloco de 7 operações)

Ordem de Inserção	**Vetor (Sequencial)**	**Vetor (Binária)**	**ABB**	**AVL**
Ordenada	$0,043$	$0,030$	$0,094$	$0,007$
Inversa	$0,011$	$0,001$	$0,108$	$0,021$
Aleatória	$0,010$	$0,002$	$0,007$	$0,006$

Tabela 3: Tempos Médios de Ordenação (ms) para $N = 10.000$ (Cenário de Pior Caso Prático)

Tamanho (N)	**Bubble Sort** $(\mathcal{O}(N^2))$	**Merge Sort** $(\mathcal{O}(N \log N))$
100	$0,217$	$0,022$
1.000	$0,917$	$0,035$
10.000	$36,492$	$0,744$

- **Degeneração da ABB ($\mathcal{O}(N)$):** O resultado mais significativo está na Tabela 1. Para $N = 10.000$ inseridos em ordem, a ABB levou $142,344$ ms. Este tempo, que é extremamente alto, comprova que a ABB colapsou para uma lista encadeada (árvore degenerada), onde a altura h se torna linear ($h \approx N$). Isto valida o pior caso teórico $\mathcal{O}(N)$ na prática.

- **Estabilidade da AVL ($\mathcal{O}(\log N)$):** Em contraste, a AVL manteve o tempo de inserção baixo e estável (aproximadamente $1,204$ ms), mesmo no cenário de dados ordenados. Isso demonstra que as rotações implementadas e o balanceamento automático garantem que a altura da árvore permaneça em $\mathcal{O}(\log N)$, assegurando eficiência para qualquer ordem de entrada.

-

4.2 Comparação de Busca: Logarítmica vs. Linear

A busca é onde a complexidade logarítmica mostra seu valor máximo para grandes volumes de dados.

- **Vetor (Busca Binária):** A Busca Binária, com complexidade $\mathcal{O}(\log N)$, demonstrou ser a operação de busca mais rápida (média de $0,002$ ms para $N = 10.000$ em dados aleatórios), beneficiando-se da localidade de memória do array.

- **Busca Sequencial:** Com complexidade $\mathcal{O}(N)$, o tempo de busca é significativamente maior ($0,043$ ms no pior caso), sendo inadequado para grandes coleções.

- **Busca em Árvores:** A AVL e a ABB (em cenários aleatórios, onde está bem balanceada) apresentaram tempos de busca extremamente rápidos ($\approx 0,007$ ms), confirmando sua eficiência $\mathcal{O}(\log N)$.

- **Análise Crítica da Busca na ABB Degenerada:** Embora a inserção na ABB tenha validado o pior caso $\mathcal{O}(N)$, notamos que o tempo de busca para este mesmo cenário ($N = 10.000$ ordenado) permaneceu baixo ($0,094$ ms). Teoricamente, a busca na lista encadeada resultante deveria ser $\mathcal{O}(N)$, apresentando tempos altos. Este fenômeno sugere que o tempo de processamento necessário para percorrer 10.000 nós é superado pelo *overhead* de inicialização e coleta da medição em Java, **mascarando o custo real da complexidade** $\mathcal{O}(N)$ em volume de dados tão pequeno. Isso reforça a importância de testar grandes volumes ($N \gg 10.000$) para operações que são intrinsecamente muito rápidas.

4.3 Eficiência de Algoritmos de Ordenação

A Tabela 3 ilustra claramente o impacto da complexidade de tempo no desempenho de ordenação.

- **Bubble Sort ($\mathcal{O}(N^2)$):** O algoritmo levou $36,492$ ms para ordenar 10.000 elementos no cenário de pior caso prático (aleatório). O crescimento quadrático de sua complexidade (N^2) é a razão direta de seu tempo de execução, tornando-o inviável para grandes conjuntos de dados.

- **Merge Sort ($\mathcal{O}(N \log N)$):** O algoritmo levou apenas $0,744$ ms para o mesmo volume de dados. Essa diferença (aproximadamente 49 vezes mais rápido que o Bubble Sort) confirma a superioridade e a escalabilidade dos algoritmos de complexidade $N \log N$.

5 Conclusão

O projeto de Análise de Desempenho cumpriu integralmente seu objetivo, fornecendo uma comprovação prática e quantitativa da teoria de complexidade de algoritmos (Notação Big O). As principais conclusões são:

- A **Árvore AVL** é a estrutura mais robusta e a mais indicada para uso geral, pois garante tempos rápidos de $\mathcal{O}(\log N)$ para inserção e busca em qualquer cenário de dados.
- A **Árvore Binária de Busca (ABB)** é perigosamente dependente da ordem de inserção, falhando catastroficamente (tempo $\mathcal{O}(N)$) quando recebe dados ordenados.
- A diferença entre complexidades $\mathcal{O}(N^2)$ e $\mathcal{O}(N \log N)$ em algoritmos de ordenação é massiva, confirmando que algoritmos avançados são essenciais para a escalabilidade de sistemas.

O aprendizado obtido reforça que a escolha da estrutura de dados e do algoritmo deve ser sempre guiada pela análise de sua complexidade teórica, priorizando o desempenho do pior caso para garantir a estabilidade e eficiência do sistema sob qualquer condição de entrada.