Trabalho: Trabalho_Estrutura_de_Dados_P2

Nome: João Pedro Mostaro

1. Metodologia
Este trabalho teve como objetivo comparar o desempenho prático de três estruturas de dados (Vetor, Árvore Binária de Busca - ABB, e Árvore AVL). As estruturas e algoritmos de ordenação (Insertion Sort e Merge Sort) foram implementados do zero, sem o uso de bibliotecas prontas do sistema.

A metodologia de testes consistiu nos seguintes passos:

Geração de Dados: Foram criados conjuntos de dados de 100, 1.000 e 10.000 elementos.

Ordens de Inserção: Para cada tamanho, as estruturas foram populadas usando três ordens: Ordenada (crescente), Inversamente Ordenada (decrescente) e Aleatória.

Medição de Tempo: O tempo de execução das operações de inserção, busca e ordenação foi medido.

Precisão: Para garantir resultados confiáveis, cada cenário de teste foi executado 5 vezes, e o tempo registrado foi a média dessas execuções.

Observação Importante: Durante a execução dos testes com 10.000 elementos, ocorreu um erro de java.lang.StackOverflowError. Este evento impediu a coleta de dados para o tamanho de 10.000 elementos e das tabelas de Busca e Ordenação. A análise deste erro está incluída na seção 3.

2. Resultados
Apresentamos os tempos médios coletados (em milissegundos) para as operações de inserção nos cenários de 100 e 1.000 elementos.

Tabela 1: Tempos Médios de Inserção (em milissegundos)

Tamanho/Ordem	Vetor	Árvore Binária (ABB)	Árvore AVL
100/Ordenada	0,014400	0,951140	21,724640
100/Inversa	0,003140	0,103100	0,090360
100/Aleatória	0,002940	0,025220	0,082400
1.000/Ordenada	0,027420	10,986100	1,229640
1.000/Inversa	0,042800	11,598720	1,401960
1.000/Aleatória	0,027960	0,361960	1,224180

(As tabelas de Busca e Ordenação, assim como os dados de 10.000 elementos, não puderam ser gerados devido à interrupção do programa.

No teste 1000/Ordenada, a recursão atingiu 1000 níveis.

No teste 10000/Ordenada (que foi onde o programa parou), a recursão tentou atingir 10.000 níveis de profundidade, o que é mais do que a memória de pilha

padrão do Java permite, causando o "estouro".)

3. Análise dos Resultados

Análise da Tabela 1 (Inserção)

Vetor: Apresentou o melhor desempenho em todos os cenários (média de ~0.02ms). Isso é esperado, pois a inserção no final de um vetor tem complexidade $O(1)$ (amortizado).

Árvore Binária de Busca (ABB):

Caso Médio (Aleatória): Foi muito rápida (0.36ms para 1000 elementos), demonstrando a eficiência da complexidade total $O(n \log n)$.

Pior Caso (Ordenada/Inversa): O desempenho foi drasticamente pior (11.0ms e 11.6ms para 1000 elementos). O tempo cresceu de forma quadrática ($O(n^2)$ total), pois cada nova inserção $O(n)$ era mais cara que a anterior.

Árvore AVL:

Estabilidade: A AVL mostrou seu ponto forte: consistência. Os tempos para 1000 elementos foram quase idênticos (1.2ms, 1.4ms, 1.2ms), independentemente da ordem.

Pior Caso vs. ABB: Em dados ordenados, a AVL (1.2ms) foi cerca de 10 vezes mais rápida que a ABB (11ms). Isso ocorre porque as rotações mantêm a árvore balanceada, garantindo a complexidade total de $O(n \log n)$7.

Anomalia (100/Ordenada): O tempo de 21,72ms para 100/Ordenada é visivelmente uma anomalia, provavelmente causada pelo "aquecimento" da Máquina Virtual Java (JIT Compiler) no primeiro teste complexo. Os dados de 1000 elementos são mais representativos.

4. Conclusão

Os resultados práticos validaram a análise de complexidade teórica das estruturas de dados.

A principal conclusão é a demonstração prática do risco da Árvore Binária de Busca (ABB): embora eficiente em média, seu pior caso (dados ordenados) não é apenas lento ($O(n^2)$ total), mas também pode causar uma falha de sistema (StackOverflowError) devido à recursão profunda.

A Árvore AVL provou ser uma solução robusta. O custo adicional das rotações (visível no caso aleatório, onde foi um pouco mais lenta que a ABB) é um pequeno preço a pagar pela garantia de que a árvore permanecerá balanceada, mantendo o desempenho rápido e estável ($O(n \log n)$) em todos os cenários, incluindo o pior caso.