

RELATÓRIO – ANÁLISE DE DESEMPENHO DE ESTRUTURAS DE DADOS EM JAVA

Aluno: Gabriel Sales Silva

Disciplina: Estrutura de Dados

Curso: (Análise e desenvolvimento de sistemas)

Instituição: (Faminas)

Ano: 2025

1. Introdução

O presente trabalho tem como objetivo analisar e comparar o desempenho de diferentes estruturas de dados utilizando a linguagem de programação Java. Foram avaliadas as estruturas **Vetor (Array Dinâmico)**, **Árvore Binária de Busca (BST)** e **Árvore AVL**, além dos algoritmos de ordenação **Insertion Sort** e **Merge Sort**.

A análise foi feita medindo o tempo de execução das operações de **inserção**, **busca** e **ordenação**, considerando diferentes tamanhos de entrada e diferentes ordens de organização dos dados. Dessa forma, foi possível observar na prática o impacto da escolha da estrutura de dados no desempenho dos programas.

2. Conjuntos de Dados Avaliados

Os testes foram realizados utilizando três tamanhos diferentes de conjuntos de dados:

- 100 elementos
- 1.000 elementos
- 10.000 elementos

Para cada tamanho, os dados foram organizados em três tipos de ordem:

- Ordem crescente (ordenada)
- Ordem decrescente (invertida)
- Ordem aleatória

Cada cenário foi executado **5 vezes**, sendo utilizado o **tempo médio** para a análise final.

3. Estruturas e Algoritmos Implementados

3.1 Vetor (Array Dinâmico)

O vetor foi implementado de forma dinâmica, permitindo o aumento automático da capacidade à medida que novos elementos são inseridos. Foram avaliadas as seguintes operações:

- Inserção
- Busca sequencial
- Busca binária (após ordenação)
- Insertion Sort
- Merge Sort

3.2 Árvore Binária de Busca (BST)

A BST foi implementada sem balanceamento automático. Seu desempenho depende diretamente da ordem de inserção dos elementos. Foram avaliadas:

- Inserção
- Busca

3.3 Árvore AVL

A árvore AVL é uma árvore binária de busca **auto-balanceada**, realizando rotações sempre que necessário para manter sua altura mínima. Foram avaliadas:

- Inserção
- Busca

4. Metodologia dos Testes

Para cada estrutura, os dados foram inseridos conforme o conjunto e a ordem definidos. Em seguida, foram realizadas as seguintes buscas:

- Primeiro elemento
- Último elemento
- Elemento do meio
- Três elementos aleatórios existentes
- Um elemento inexistente

O tempo final de busca corresponde à **média das seis buscas realizadas**.
Os tempos foram medidos utilizando o método `System.nanoTime()` e convertidos para milissegundos.

5. Análise Teórica do Desempenho

5.1 Vetor

- Inserção: $O(1)$ amortizado
- Busca sequencial: $O(n)$
- Busca binária: $O(\log n)$ (necessita vetor ordenado)
- Insertion Sort: $O(n^2)$ no pior caso
- Merge Sort: $O(n \log n)$

5.2 Árvore Binária de Busca (BST)

- Melhor caso: $O(\log n)$
- Pior caso: $O(n)$
- Pode se tornar ineficiente em entradas ordenadas ou invertidas.

5.3 Árvore AVL

- Inserção: $O(\log n)$
 - Busca: $O(\log n)$
 - Mantém desempenho estável independentemente da ordem de inserção.
-

6. Comparação Geral entre as Estruturas

- O **vetor** apresenta inserção muito rápida, porém busca sequencial lenta para grandes volumes de dados.
 - A **busca binária no vetor** é extremamente eficiente após a ordenação.
 - O **Insertion Sort** se torna inviável para grandes volumes de dados.
 - O **Merge Sort** mantém ótimo desempenho para qualquer tamanho.
 - A **BST** tem desempenho instável.
 - A **AVL** apresenta o melhor desempenho geral em buscas.
-

7. Conclusão

A partir dos experimentos realizados, foi possível concluir que a escolha correta da estrutura de dados é fundamental para o desempenho das aplicações.

Estruturas simples, como vetores e árvores não balanceadas, são adequadas para pequenos volumes de dados, mas se tornam ineficientes à medida que o tamanho da entrada cresce.

Já a **Árvore AVL** e o **Merge Sort** mostraram-se as melhores opções para grandes conjuntos de dados, por garantirem desempenho estável e eficiente.

O projeto ajudou a consolidar na prática os conceitos estudados em sala, reforçando a importância da análise de complexidade de algoritmos.