RELATÓRIO TÉCNICO

Análise Comparativa de Desempenho de Estruturas de Dados em Java

Aluno: João Vitor Façanha Neves

RA: 1-23-14887

Disciplina: Estrutura de Dados

Professor: Flávio Motta

Data de Entrega: 05/12/2025

SUMÁRIO

==========

1. INTRODUÇÃO

Este trabalho apresenta uma análise experimental comparativa do desempenho de três estruturas de dados fundamentais em Ciência da Computação: vetores (arrays), árvores binárias de busca (ABB) e árvores AVL. O objetivo principal foi verificar na prática a correspondência entre a complexidade teórica, expressa em notação Big O, e o desempenho real dessas estruturas em operações básicas de inserção e busca.

A motivação para este estudo reside na importância de selecionar estruturas de dados apropriadas para diferentes cenários de aplicação. Embora a teoria forneça previsões sobre o comportamento assintótico, apenas testes empíricos podem confirmar essas previsões em ambientes reais e quantificar as diferenças de desempenho.

O trabalho seguiu rigorosamente uma metodologia experimental que incluiu a implementação completa das estruturas, geração sistemática de dados em diferentes ordens, e medição precisa dos tempos de execução para múltiplas operações.

2. METODOLOGIA

2.1 Implementação

Todas as estruturas foram implementadas do zero em linguagem Java, sem utilização de bibliotecas externas ou classes prontas da API Collections. As implementações incluíram:

• Vetor: Implementação com redimensionamento dinâmico, contendo métodos para inserção (O(1) amortizado), busca sequencial (O(n)) e busca binária (O(log n)).

• Árvore Binária de Busca (ABB): Implementação recursiva com inserção e busca seguindo as propriedades de BST.

• Árvore AVL: Implementação completa com balanceamento automático através das quatro rotações (simples à direita, simples à esquerda, dupla direita-esquerda, dupla esquerda-direita).

• Algoritmos de Ordenação: Bubble Sort ($O(n^2)$) e Quick Sort ($O(n \log n)$) implementados como classes separadas.

2.2 Geração de Dados

Foram utilizados três tamanhos de conjuntos de dados para verificar a escalabilidade:

• 100 elementos

• 1.000 elementos

• 10.000 elementos

Para cada tamanho, três ordens de inserção distintas foram testadas:

1. Ordenada crescente (1, 2, 3, ..., n)

2. Ordenada decrescente (n, n-1, n-2, ..., 1)

3. Aleatória (valores gerados pseudoaleatoriamente no intervalo [1, 10n])

2.3 Medição de Tempo

Para garantir confiabilidade estatística:

• Cada operação foi executada 5 vezes consecutivas

• O tempo médio foi calculado eliminando possíveis outliers

• Utilizou-se System.nanoTime() para precisão nanossegundo

• Implementou-se uma classe Cronometro para padronização

Todas as medições foram realizadas no mesmo computador, com sistema operacional Windows 10, processador Intel Core i5 de 8ª geração e 8GB de RAM, sem outros processos pesados em execução.

2.4 Operações Testadas

As seguintes operações foram submetidas a testes de desempenho:

1. Inserção: Tempo total para inserir todos os elementos do conjunto em cada estrutura.

2. Busca: Tempo para localizar sete elementos específicos em cada estrutura:

a) Primeiro elemento inserido

b) Último elemento inserido

c) Elemento na posição central

d) Três elementos selecionados aleatoriamente

e) Um elemento garantidamente inexistente

3. Ordenação: Aplicação dos algoritmos Bubble Sort e Quick Sort nos vetores previamente populados.

3. RESULTADOS

3.1 Tempos de Inserção

Os tempos médios de inserção para todas as combinações de tamanho e ordem encontram-se na Tabela 1.

TABELA 1 - Tempos médios de inserção (milissegundos)

Tamanho	Ordem	Vetor	Árvore Binária	Árvore AVL
100	Ordenada	0,169	0,639	0,558
	Inversa	0,006	0,102	0,044
	Aleatória	0,012	0,022	0,034
1.000	Ordenada	0,054	2,629	0,248
	Inversa	0,050	2,004	0,108
	Aleatória	0,035	0,090	0,140
10.000	Ordenada	0,337	137,362	0,959
	Inversa	0,170	210,727	0,839

| | Aleatória | 0,117 | 0,918 | 1,295 |

+----------+----------+--------+----------+----------+

3.2 Tempos de Ordenação

Os resultados da ordenação de vetores são apresentados na Tabela 2.

TABELA 2 - Tempos médios de ordenação (milissegundos)

+----------+----------+----------+----------+

| Tamanho | Ordem Inicial | Bubble Sort | Quick Sort |

+----------+----------+----------+----------+

| 100 | Ordenada | 0,121 | 0,115 |

| | Inversa | 0,243 | 0,030 |

| | Aleatória | 0,196 | 0,006 |

+----------+----------+----------+----------+

| 1.000 | Ordenada | 1,622 | 0,649 |

| | Inversa | 1,043 | 0,500 |

| | Aleatória | 0,713 | 0,045 |

+----------+----------+----------+----------+

| 10.000 | Ordenada | 23,674 | 50,963 |

| | Inversa | 73,823 | 49,179 |

| | Aleatória | 95,854 | 0,545 |

+----------+----------+----------+----------+

3.3 Análise de Busca

Os tempos de busca mostraram padrões consistentes com as expectativas teóricas:

• Vetor: A busca sequencial apresentou tempo proporcional à posição do elemento, confirmando O(n).

• Árvore Binária: Os tempos variaram significativamente conforme o balanceamento da árvore. Para dados ordenados, a busca aproximou-se de O(n), enquanto para dados aleatórios apresentou comportamento O(log n).

• Árvore AVL: Todos os tempos de busca mantiveram-se consistentes, independentemente do elemento procurado, confirmando a garantia de O(log n).

4. ANÁLISE DOS RESULTADOS

4.1 Comparação entre Estruturas

A Tabela 3 resume o comportamento observado para cada estrutura.

TABELA 3 - Análise comparativa do desempenho

Aspecto	Vetor	Árvore Binária	Árvore AVL
Inserção Ordenada	Constante (0,3ms)	Catastrófica (137ms)	Eficiente (1,0ms)
Inserção Aleatória	Constante (0,1ms)	Boa (0,9ms)	Boa (1,3ms)
Impacto da Ordem	Nenhum	Crítico (150× diferença)	Mínimo
Complexidade Busca	Linear O(n)	O(log n) a O(n)	Logarítmico O(log n)
Caso de Uso Ideal	Muitas inserções	Dados sempre aleatórios	Performance garantida

4.2 Análise Detalhada por Tamanho

4.2.1 Degradação da Árvore Binária

A degradação da ABB com dados ordenados mostrou-se exponencial:

• Para n=100: Diferença de 29× entre ordenado (0,639ms) e aleatório (0,022ms)

• Para n=1.000: Diferença de 29× entre ordenado (2,629ms) e aleatório (0,090ms)

• Para n=10.000: Diferença de 150× entre ordenado (137ms) e aleatório (0,918ms)

Interessantemente, os dados inversamente ordenados produziram degradação ainda pior que os ordenados (210ms vs 137ms para n=10.000).

4.2.2 Eficiência da Árvore AVL

A AVL demonstrou:

• Performance consistente em todas as ordens de inserção

• Overhead mínimo sobre a ABB para dados aleatórios (apenas ~0,3ms)

• Garantia prática de O(log n) independente da entrada

4.2.3 Comparação Bubble Sort vs Quick Sort

A Tabela 4 apresenta a análise comparativa dos algoritmos de ordenação.

TABELA 4 - Análise comparativa dos algoritmos de ordenação

Métrica	Bubble Sort	Quick Sort	Vantagem Quick Sort
n=10.000 (aleatório)	95,854ms	0,545ms	176× mais rápido
Crescimento n=1.000→10.000	134×	12×	Crescimento controlado
Complexidade melhor caso	O(n)	O(n log n)	Bubble teoricamente
Complexidade pior caso	$O(n^2)$	$O(n^2)$	equivalente
Estabilidade	Estável	Instável	Bubble mantém ordem

4.3 Relação Teoria-Prática

Os resultados experimentais confirmam plenamente as previsões teóricas:

1. Vetor: Confirma O(1) amortizado para inserção

2. ABB com dados ordenados: Confirma O(n) - degeneração para lista

3. AVL: Confirma O(log n) - comportamento logarítmico

4. Bubble Sort: Confirma $O(n^2)$ - crescimento quadrático evidente

5. Quick Sort: Confirma O(n log n) - crescimento quase linear

O caso interessante foi o pior caso do Quick Sort com dados ordenados (50ms), que se aproximou do comportamento quadrático, enquanto para dados aleatórios manteve excelente performance (0,5ms).

5. CONCLUSÕES

5.1 Principais Descobertas

1. Degradação Catastrófica da ABB: A árvore binária demonstrou degradação de 229× entre o pior caso (dados inversos) e o caso médio (dados aleatórios) para n=10.000.

2. Eficácia do Balanceamento AVL: O overhead das rotações AVL (apenas ~0,3ms) provou ser excelente investimento, garantindo performance consistente independente da ordem de inserção.

3. Superioridade do Quick Sort: Para dados aleatórios, Quick Sort foi 176× mais rápido que Bubble Sort em n=10.000, confirmando a importância de algoritmos O(n log n) para grandes conjuntos.

4. Vetor como Especialista em Inserção: A estrutura mais simples apresentou o melhor desempenho em inserção, confirmando O(1) amortizado.

5.2 Aprendizados Relevantes

- A notação Big O não é apenas teórica - tem implicações práticas mensuráveis
- A escolha da estrutura de dados deve considerar a distribuição real dos dados
- O custo do balanceamento (AVL) é justificado na maioria dos cenários reais
- Algoritmos $O(n^2)$ tornam-se rapidamente inviáveis com o crescimento de n

5.3 Recomendações Práticas

Cenário de Aplicação	Estrutura/Algoritmo Recomendado	Justificativa Técnica
Muitas inserções, poucas buscas	Vetor	O(1) para inserção, simplicidade
Dados que chegam ordenados	Árvore AVL	Evita degradação O(n) da ABB
Sistema com buscas frequentes	Árvore AVL	O(log n) garantido para todas as buscas
Ordenação geral de dados	Quick Sort	O(n log n) eficiente na maioria dos casos
Conjuntos pequenos (n < 100)	Bubble Sort	Simplicidade de implementação
Dados sensíveis à ordem relativa	Bubble Sort	Estabilidade do algoritmo

6. Disponibilidade

Disponível em repositório Git: https://github.com/Jotinha14/Trabalho-de-Estrutura-de-Dados

Estrutura do projeto:

Trabalho De Estrutura de Dados/

├── src/vetor/Vetor.java

├── src/vetor/BubbleSort.java

├── src/vetor/QuickSort.java

├── src/arvores/ArvoreBinaria.java

├── src/arvores/ArvoreAVL.java

├── src/arvores/No.java

├── src/utils/GeradorDados.java

└── src/testes/AnaliseDesempenho.java

Instruções de execução:

1. javac -d . src/vetor/*.java src/arvores/*.java src/utils/*.java src/testes/*.java

2. java testes.AnaliseDesempenho

Tabelas:

Inserção

Tamanho	Ordem	Vetor (ms)	ABB (ms)	AVL (ms)
100	ORDENADA	0,519	0,973	0,696
100	INVERSA	0,008	0,193	0,063
100	ALEATORIA	0,014	0,109	0,053
1000	ORDENADA	0,113	3,825	0,202
1000	INVERSA	0,063	5,322	0,187
1000	ALEATORIA	0,18	0,262	0,796
10000	ORDENADA	0,53	174,869	1,238
10000	INVERSA	0,144	272,658	1,119
10000	ALEATORIA	0,144	1,765	1,877

Ordenação

Tamanho	Ordem	Bubble (ms)	Quick (ms)
100	ORDENADA	0,187	0,203
100	INVERSA	0,564	0,067
100	ALEATORIA	1,055	0,008
1000	ORDENADA	3,507	1,64
1000	INVERSA	1,292	1,359
1000	ALEATORIA	1,072	0,069
10000	ORDENADA	32,066	65,048
10000	INVERSA	94,352	79,804
10000	ALEATORIA	118,596	0,864

Conclusões:

Vetor é extremamente rápido para inserção em listas pequenas, mas piora em dados desordenados e grandes.

ABB tem performance ruim para entrada ordenada/inversa (degenera).

AVL mantém ótimo desempenho estável porque é balanceada.

Bubble Sort explode em entradas grandes.

QuickSort é extremamente mais eficiente na maioria dos casos.