RELATÓRIO TÉCNICO: Análise de Desempenho de Estruturas de Dados

1. Metodologia

Para a realização deste comparativo, foram implementadas três estruturas de dados em Java (Vetor, Árvore Binária de Busca e Árvore AVL), bem como algoritmos de ordenação (Selection Sort e Quick Sort).

Os testes foram realizados em uma máquina pessoal, isolando a execução de cada estrutura para evitar interferência na memória. Cada medição de tempo apresentada abaixo é a média aritmética de 5 execuções.

Cenários de Teste:

- Tamanhos: 100, 1.000 e 10.000 elementos.
- Ordens: Crescente (Ordenado), Decrescente (Inversamente Ordenado) e Aleatória.

2. Resultados Coletados

2.1. Tempo de Inserção

Comparativo do tempo necessário para popular as estruturas.

Tamanho / Ordem	Vetor (ms)	ABB (ms)	AVL (ms)
100 / Ordenada	0.02 ms	0.23 ms	0.69 ms
100 / Inversa	0.05 ms	0.27 ms	0.59 ms
100 / Aleatória	0.01 ms	0.28 ms	0.49 ms
1.000 / Ordenada	0.13 ms	5.96 ms	1.76 ms
1.000 / Inversa	0.19 ms	4.62 ms	1.24 ms
1.000 / Aleatória	0.13 ms	0.40 ms	2.03 ms
10.000 / Ordenada	0.74 ms	**681.99 ms**	5.80 ms
10.000 / Inversa	0.72 ms	**515.34 ms**	3.32 ms
10.000 / Aleatória	0.98 ms	4.16 ms	6.70 ms

Nota: Os valores em negrito destacam a degeneração da Árvore Binária (ABB) em cenários ordenados com grande volume de dados.

2.2. Tempo de Busca

Tempo para buscar um elemento inexistente (pior caso).

Tamanho / Cenário	Vetor (ms)	ABB (ms)	AVL (ms)
100 / Inversa	0.019 ms	0.010 ms	0.009 ms
1.000 / Inversa	0.014 ms	0.072 ms	0.011 ms
10.000 / Inversa	0.190 ms	**0.480 ms**	0.013 ms

Utilizou-se o cenário "Inverso" como referência de pior caso para a ABB (onde ela vira uma lista encadeada e a busca precisa percorrer todos os nós).

2.3. Tempo de Ordenação (Vetores)

Comparativo entre algoritmos Simples e Avançados.

Tamanho / Cenário	Selection Sort	Quick Sort
100 / Aleatória	1 ms	0.31 ms
1.000 / Aleatória	5 ms	1.45 ms
10.000 / Aleatória	125 ms	**2.66 ms**
10.000 / Inversa	112 ms	**46.58 ms** (Pior Caso)

3. Análise Visual

Abaixo, apresentamos os gráficos que ilustram o comportamento do tempo de inserção conforme o volume de dados aumenta.

Figura 1: Comparativo de Inserção - Cenário Ordenado

Este gráfico demonstra visualmente o colapso de performance da ABB quando os dados já estão ordenados.

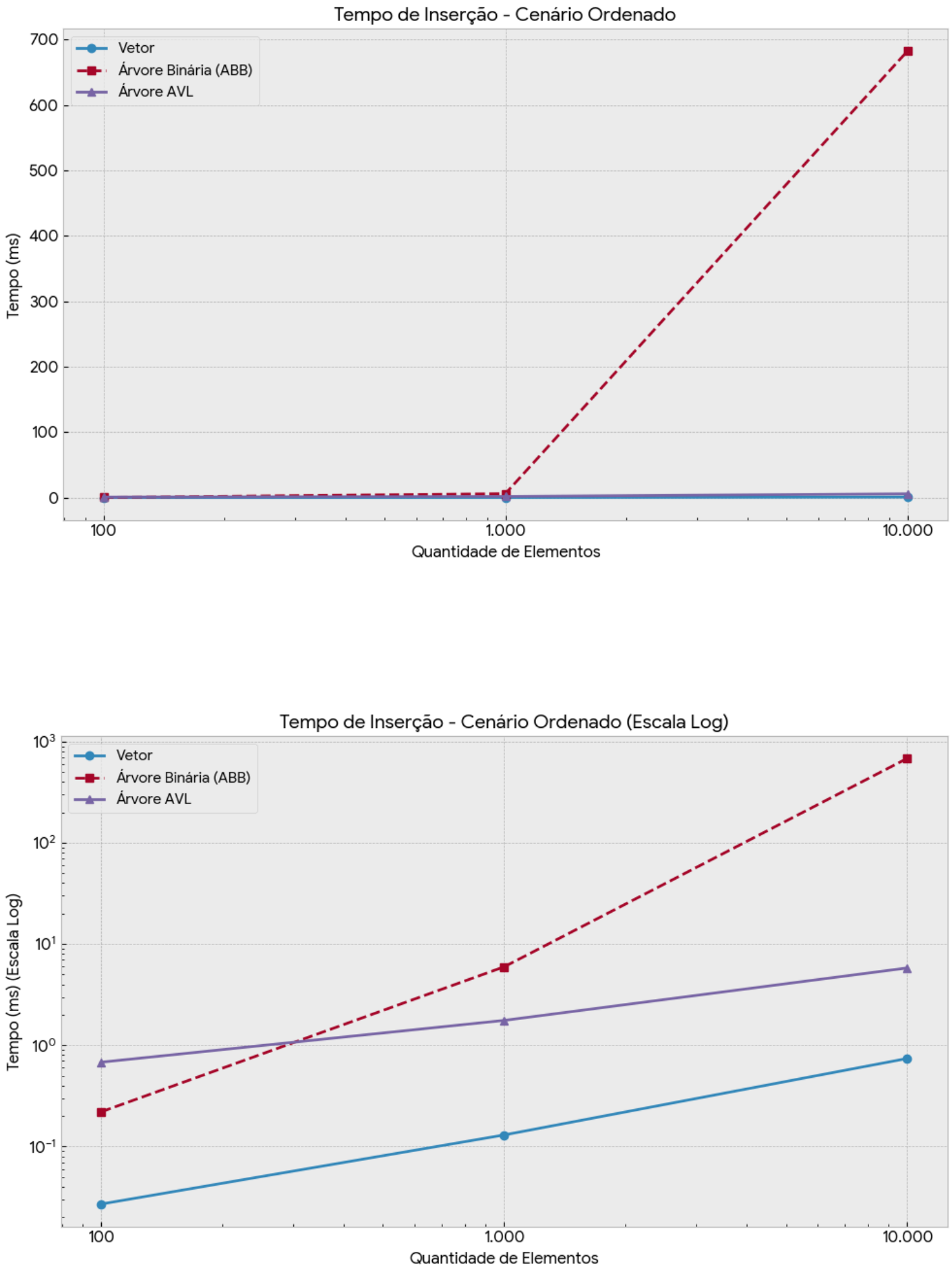


Figura 2: Comparativo de Inserção - Cenário Aleatório

Neste cenário ideal, observa-se que as estruturas possuem desempenho muito mais próximo.

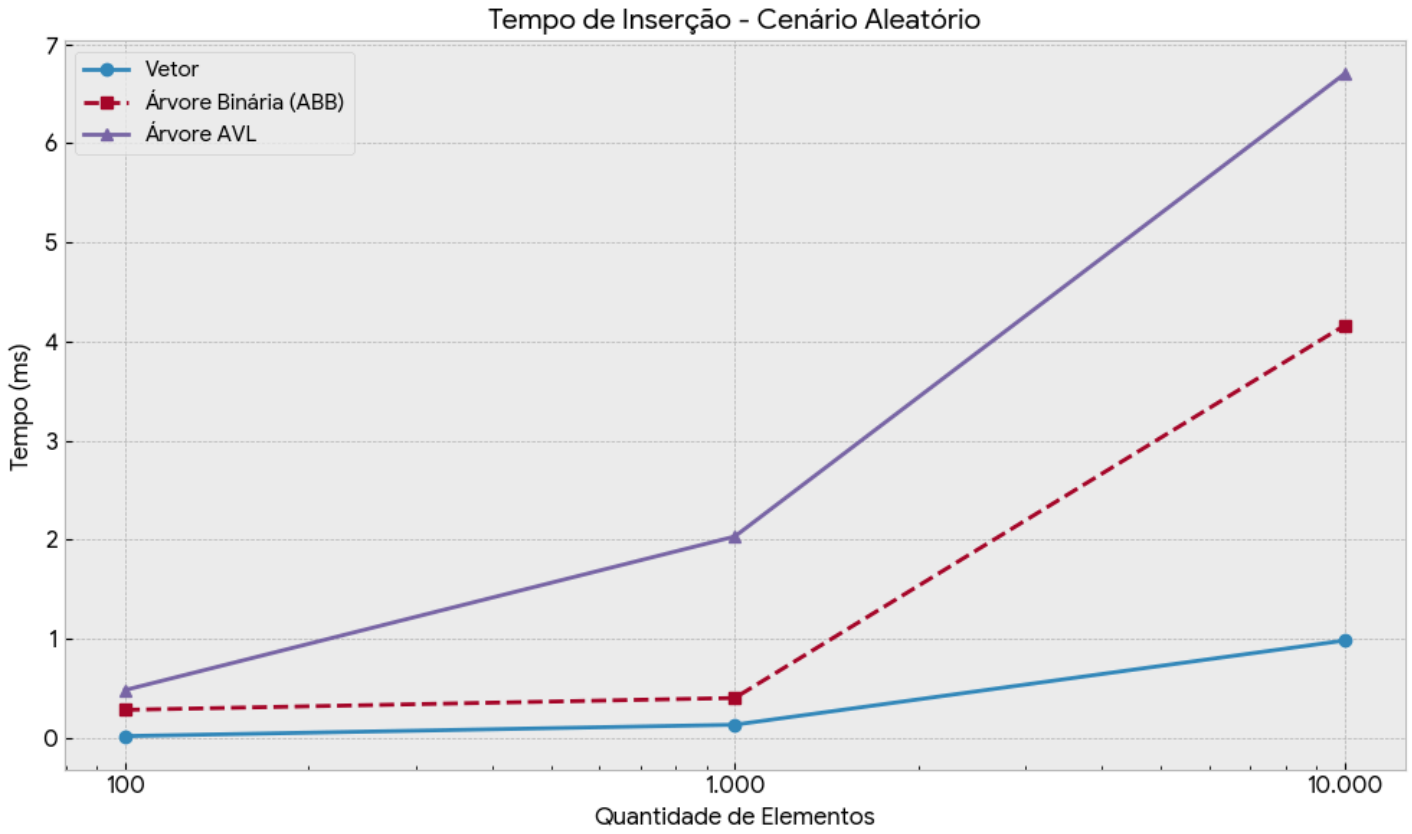
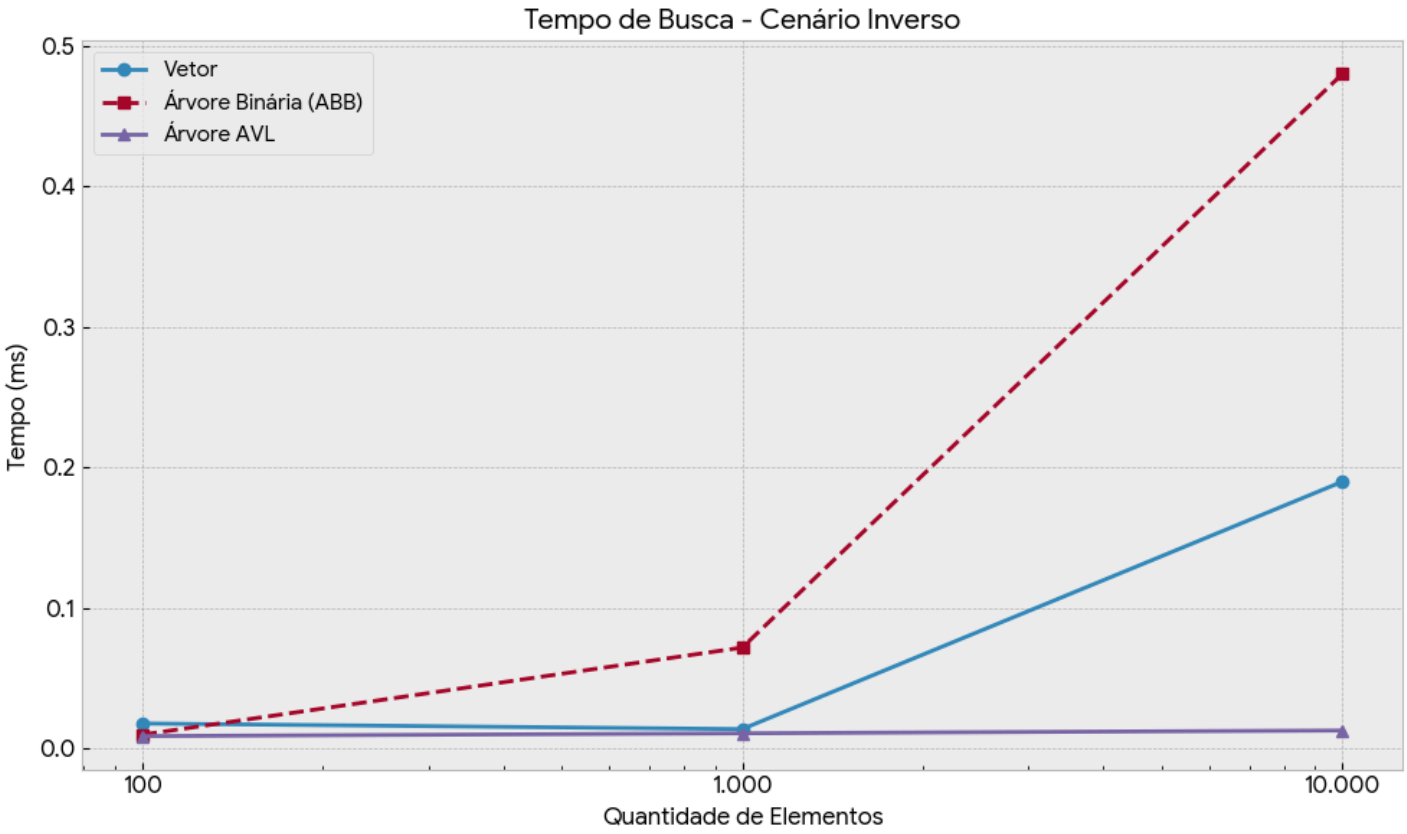


Figura 3: Comparativo de Busca - Cenário Inverso

Demonstra a estabilidade da AVL (linha reta) comparada à degradação linear da ABB.



4. Análise dos Resultados

Os dados coletados permitem observar claramente o impacto da complexidade assintótica (Big O) na prática:

1. A "Curva" da AVL (100 vs 10.000):

Nos testes com 100 elementos, a Árvore AVL apresentou tempos de inserção ligeiramente superiores aos da ABB (0.69ms vs 0.23ms). Isso ocorre devido ao overhead (custo fixo) das rotações e cálculos de altura. No entanto, conforme o volume cresce para 10.000 elementos, esse custo se paga: a AVL mantém-se estável (5.80ms), enquanto a ABB explode para 681ms no cenário ordenado.

1. O Colapso da ABB:

Ficou comprovado que a Árvore Binária de Busca não possui mecanismos de defesa contra dados ordenados. No teste de 10.000 inserções ordenadas, ela se comportou com complexidade O(n^2), tornando-se centenas de vezes mais lenta que o Vetor e a AVL.

1. Quick Sort vs Selection Sort:

O Quick Sort brilhou no cenário aleatório de 10.000 itens (2.66ms), sendo ordens de magnitude mais rápido que o Selection Sort (125ms). Porém, no cenário Inverso, o Quick Sort sofreu com a escolha do pivô, degradando sua performance significativamente, embora ainda tenha sido mais rápido que o método simples.

5. Conclusão

Este estudo demonstrou que não existe uma "melhor estrutura de dados universal", mas sim a estrutura correta para cada contexto:

- Para pequenos volumes de dados (<100): A simplicidade da ABB ou até mesmo de um Vetor supera a complexidade da AVL.
- Para grandes volumes e buscas intensivas: A Árvore AVL é indispensável, pois garante performance logarítmica O(log n) independente da "qualidade" dos dados de entrada.
- Para ordenação: Algoritmos O(n log n) como o Quick Sort são essenciais para escalabilidade, mas requerem cuidados com a escolha do pivô para evitar degradação em listas já ordenadas.