Relatório Técnico: Análise de Desempenho de Estruturas de Dados

Disciplina: Estruturas de Dados | **Aluno:** Thiago Ferreira Gonçalves

Linguagem Utilizada: Java

Data: 21 de Novembro de 2025

1. Metodologia

1.1. Planejamento dos Testes

Para garantir a precisão estatística dos resultados, foram realizadas **5 baterias completas de testes**. Todos os valores apresentados nas tabelas a seguir representam a **média aritmética** dessas 5 execuções, mitigando variações pontuais de processamento e coleta de lixo (Garbage Collection) da JVM.

As estruturas analisadas foram:

1. **Vetor (Array):** Busca Sequencial e Busca Binária.
2. **Árvore Binária de Busca (BST):** Implementação simples sem balanceamento.
3. **Árvore AVL:** Árvore binária com balanceamento automático por altura.

1.2. Cenários de Teste

Os testes cobriram três magnitudes de dados (N) e três disposições iniciais para cada estrutura:

- **Tamanhos:** 100, 1.000 e 10.000 elementos.
- **Ordenações:**
  - **Aleatória:** Simula o caso médio (dados desordenados).
  - **Crescente:** Simula dados já ordenados (melhor caso para ordenação, pior caso para BST).
  - **Decrescente:** Simula dados inversamente ordenados.

1.3. Ambiente de Teste

As especificações do ambiente onde os testes foram executados são as seguintes:

- **Processador:** AMD Ryzen 5 5500 (3.60 GHz)
- **Memória RAM:** 16 GB DDR4 (3200 MHz)
- **Placa de vídeo:** RX 6650xt
- **Armazenamento:** SSD nvme 3500 MB/s leitura e 2400 MB/s escrita
- **Sistema Operacional:** Windows 11 (64-bit)
- **Execução:** Java via jGRASP

2. Resultados Consolidados

2.1. Algoritmos de Ordenação

Comparativo de tempo para ordenar vetores.

Algoritmo	Tamanho (N)	Ordem Inicial	Tempo Médio (ms)
BubbleSort	100	Crescente	0,203
	100	Decrescente	0,249
	1.000	Crescente	4,641
	1.000	Decrescente	9,707
	10.000	Crescente	36,043
	10.000	Decrescente	29,708
QuickSort	100	Crescente	0,037
	100	Decrescente	0,039
	1.000	Crescente	0,251
	1.000	Decrescente	0,073
	10.000	Crescente	0,843
	10.000	Decrescente	0,803

2.2. Tempos de Inserção (Construção da Estrutura)

Tempo médio para inserir N elementos na estrutura vazia.

Tamanho	Ordem dos Dados	Vetor (Geração)	Árvore Binária	Árvore AVL
100	Aleatória	0,009 ms	1,046 ms	1,268 ms
	Crescente	0,025 ms	0,026 ms	0,025 ms
	Decrescente	0,092 ms	0,029 ms	0,025 ms
1.000	Aleatória	0,045 ms	0,164 ms	0,528 ms
	Crescente	0,178 ms	1,296 ms	0,278 ms
	Decrescente	0,091 ms	1,187 ms	0,074 ms
10.000	Aleatória	0,509 ms	1,568 ms	3,316 ms
	Crescente	0,699 ms	**121,506 ms**	1,217 ms
	Decrescente	0,671 ms	**120,373 ms**	0,943 ms

> **Análise Crítica:** A inserção ordenada na Árvore Binária simples apresenta degradação severa (~121ms para 10k itens). Isso ocorre porque a estrutura degenera em uma lista encadeada, elevando a complexidade da operação de inserção para $O(N^2)$ no processo de construção total.

2.3. Tempos de Busca Detalhados: Árvore AVL

A Árvore AVL mantém tempos de busca consistentes e baixos em todos os cenários devido ao seu balanceamento.

N = 100 Elementos

Ordem	Primeiro	Último	Meio	Aleatório (Méd)	Inexistente
Aleatória	0,006	0,003	0,003	0,003	0,003
Crescente	0,002	0,002	0,001	0,001	0,001
Decrescente	0,001	0,001	0,001	0,001	0,001

N = 1.000 Elementos

Ordem	Primeiro	Último	Meio	Aleatório (Méd)	Inexistente
Aleatória	0,001	0,002	0,002	0,002	0,002
Crescente	0,001	0,001	0,001	0,001	0,001
Decrescente	0,001	0,001	0,001	0,001	0,001

N = 10.000 Elementos

Ordem	Primeiro	Último	Meio	Aleatório (Méd)	Inexistente
Aleatória	0,002	0,003	0,003	0,003	0,002
Crescente	0,001	0,002	0,001	0,002	0,001
Decrescente	0,002	0,001	0,001	0,001	0,001

2.4. Tempos de Busca Detalhados: Árvore Binária (BST)

A BST sofre degradação de performance quando os dados inseridos são ordenados, transformando a busca em uma operação linear $O(N)$ para os piores casos (último elemento).

N = 100 Elementos

Ordem	Primeiro	Último	Meio	Aleatório (Méd)	Inexistente
Aleatória	0,004	0,007	0,003	0,003	0,003
Crescente	0,002	**0,021**	0,014	0,006	0,010
Decrescente	0,002	**0,049**	0,004	0,004	0,004

N = 1.000 Elementos

Ordem	Primeiro	Último	Meio	Aleatório (Méd)	Inexistente
Aleatória	0,001	0,002	0,002	0,002	0,001
Crescente	0,001	**0,145**	0,045	0,047	0,092
Decrescente	0,001	0,009	0,003	0,003	0,002

N = 10.000 Elementos

Ordem	Primeiro	Último	Meio	Aleatório (Méd)	Inexistente
Aleatória	0,001	0,002	0,002	0,002	0,002
Crescente	0,004	**0,325**	0,029	0,018	0,031
Decrescente	0,005	**0,108**	0,016	0,025	0,002

> **Nota:** A busca pelo "Último elemento" em uma árvore gerada com dados crescentes é o pior caso possível, pois o algoritmo precisa percorrer N nós até a folha mais à direita.

2.5. Tempos de Busca Detalhados: Vetor (Array)

Comparação entre Busca Sequencial (linear) e Busca Binária (logarítmica). A Busca Binária só foi executada em vetores ordenados.

N = 100 Elementos

Ordem	Tipo Busca	Primeiro	Último	Meio	Aleatório	Inexistente
Aleatória	Sequencial	0,001	0,001	0,001	0,001	0,001
Crescente	Sequencial	0,007	0,046	0,026	0,030	0,047
	Binária	0,005	0,003	0,004	0,004	0,004
Decrescente	Sequencial	0,004	0,001	0,001	0,001	0,002

N = 1.000 Elementos

Ordem	Tipo Busca	Primeiro	Último	Meio	Aleatório	Inexistente
Aleatória	Sequencial	0,001	0,005	0,003	0,003	0,005
Crescente	Sequencial	0,005	0,353	0,177	0,209	0,358
	Binária	0,002	0,004	0,003	0,002	0,002
Decrescente	Sequencial	0,001	0,006	0,004	0,003	0,005

N = 10.000 Elementos

Ordem	Tipo Busca	Primeiro	Último	Meio	Aleatório	Inexistente
Aleatória	Sequencial	0,001	0,996	0,517	0,250	0,433
Crescente	Sequencial	0,004	**4,207**	1,886	1,480	0,515
	Binária	0,002	**0,008**	0,002	0,005	0,001
Decrescente	Sequencial	0,001	0,044	0,020	0,021	0,039

> **Destaque:** A diferença entre buscar o último elemento via Busca Sequencial (**~4,2ms**) e via Busca Binária (**~0,008ms**) em um vetor de 10.000 posições ilustra a superioridade da complexidade $O(\log n)$ sobre $O(n)$.

3. Análise dos Resultados

3.1. Ordenação

Os testes confirmam a ineficiência do **BubbleSort** para grandes volumes de dados, crescendo exponencialmente o tempo de execução. O **QuickSort**, por sua vez, manteve-se extremamente eficiente, sendo cerca de 40 vezes mais rápido que o BubbleSort no cenário de 10.000 elementos.

3.2. Estruturas de Árvore

A **Árvore AVL** demonstrou ser a estrutura mais robusta. Mesmo com o custo computacional das rotações durante a inserção, ela garantiu tempos de busca praticamente instantâneos em todos os cenários. A **Árvore Binária** simples falhou catastroficamente ao processar dados ordenados, degenerando em uma lista encadeada, o que elevou o tempo de inserção para mais de 120ms e degradou o tempo de busca.

3.3. Vetores

O vetor oferece acesso rápido, mas a busca em dados desordenados (Sequencial) torna-se lenta conforme N aumenta. A **Busca Binária** mostrou-se tão eficiente quanto as árvores balanceadas, mas requer que os dados estejam previamente ordenados.

4. Conclusão

Para aplicações que lidam com grandes volumes de dados e necessitam de buscas frequentes, a **Árvore AVL** é a escolha mais segura, pois oferece garantias de desempenho independentemente da ordem de entrada dos dados. Se os dados forem estáticos e puderem ser mantidos ordenados, a **Busca Binária** em vetor é uma alternativa viável e eficiente. O uso de árvores binárias sem balanceamento deve ser evitado em cenários onde não se pode garantir a aleatoriedade da entrada.