Relatório Técnico: Análise de Desempenho de Estruturas de Dados

Os testes foram planejados para comparar o desempenho de três estruturas de dados principais: Vetor (Array), Árvore AVL e Árvore B. Além disso, foram analisados algoritmos de ordenação (Bubble Sort e Merge Sort) e métodos de busca (Sequencial e Binária).

Link para ver os gráficos : https://camilyolivei.github.io/graficos/

1. Metodologia

Os testes foram projetados para comparar o desempenho das seguintes estruturas e algoritmos:

Estruturas de Dados:

- Vetor (Array)
- Árvore AVL
- Árvore Binária Simples

Algoritmos de Ordenação:

- Bubble Sort
- Merge Sort

Métodos de Busca:

- Sequencial
- Binária
- Busca em AVL
- Busca em Árvore Binária

Execução dos Testes

Os testes foram implementados na linguagem Java. A classe **Relatorios.java** orquestra a execução, onde cada operação é repetida 10 vezes (QUANTIDADE_EXECUCAO) para calcular o tempo médio, minimizando a influência de processos concorrentes do sistema operacional.

Medição de Tempo

A medição utiliza:

```
long ini = System.nanoTime();
// operação...
tempo += System.nanoTime() - ini;
```

Isso garante precisão em nanosegundos.

Geração dos Dados

Foram avaliados três tipos de entrada:

- Ordenada (crescente)
- Invertida (decrescente)
- Aleatória

Os tamanhos utilizados:

- 100
- 1.000
- 10.000 elementos

Para buscas, até 10.000 elementos

2. Resultados

2.1 Inserção

2.1.1 Vetor

Tamanho	Ordenado	Invertido	Aleatório
100	770 ns	920 ns	45.300 ns
1.000	6.330 ns	8.370 ns	73.840 ns
10.000	59.660 ns	66.520 ns	200.350 ns

2.1.2 Árvore AVL

Tamanho	Ordenado	Invertido	Aleatório
100	228.450 ns	7.910 ns	98.890 ns
1.000	179.580 ns	114.120 ns	406.370 ns
10.000	800.590 ns	1.426.410 ns	1.985.780 ns

2.1.3 Árvore Binária Simples

Tamanho	Ordenado	Invertido	Aleatório
100	236.230 ns	50.380 ns	53.460 ns
1.000	2.986.870 ns	7.197.470 ns	218.440 ns
10.000	307.540.810 ns	641.870.320 ns	2.161.080 ns

2.2 Busca

2.2.1 Busca em Vetor (Sequencial)

Tamanho	Primeiro	Último	Meio	Aleatórios (3)	Inexistente
100	730 ns	1.780 ns	980 ns	6.570 ns	1.320 ns
1.000	310 ns	14.390 ns	8.930 ns	41.250 ns	8.300 ns
10.000	70 ns	159.510 ns	88.850 ns	148.410 ns	80.870 ns

2.2.2 Busca em Vetor (Binária)

Tamanho	Primeiro	Último	Meio	Aleatórios (3)	Inexistente
100	590 ns	390 ns	350 ns	1.430 ns	380 ns
1.000	310 ns	300 ns	270 ns	1.310 ns	290 ns
10.000	340 ns	350 ns	340 ns	1.640 ns	410 ns

2.2.3 Busca em Árvore AVL

Tamanho	Primeiro	Último	Meio	Aleatórios (3)	Inexistente
100	490 ns	1.340 ns	280 ns	2.420 ns	320 ns
1.000	1.260 ns	450 ns	380 ns	1.390 ns	410 ns
10.000	1.280 ns	430 ns	200 ns	1.140 ns	150 ns

2.2.4 Busca em Árvore Binária Simples

Tamanho	Primeiro	Último	Meio	Aleatórios (3)	Inexistente
100	430 ns	13.420 ns	2.240 ns	6.460 ns	730 ns
1.000	340 ns	15.980 ns	6.500 ns	21.400 ns	120 ns
10.000	940 ns	44.850 ns	21.430 ns	107.640 ns	1.690 ns

2.3 Ordenação

2.3.1 Bubble Sort

Tamanho	Ordenado	Aleatório	Invertido
100	2.470 ns	26.340 ns	141.800 ns
1.000	19.480 ns	813.690 ns	843.560 ns
10.000	126.540 ns	89.665.450 ns	63.148.430 ns

2.3.2 Merge Sort

Tamanho	Ordenado	Aleatório	Invertido
100	40.780 ns	11.570 ns	6.040 ns
1.000	98.470 ns	171.620 ns	88.510 ns
10.000	1.026.940 ns	1.633.530 ns	1.427.710 ns

3. Análise dos Resultados

3.1 Inserção

Vetor

Os tempos foram extremamente baixos (de 770 ns a 200.350 ns), confirmando que a operação é O(1) devido ao uso de vetores prealocados.

Árvore AVL

Dados invertidos tiveram os melhores tempos para pequenas entradas.

Inserções aleatórias ficam mais caras conforme o tamanho cresce.

Inserções ordenadas foram mais lentas que invertidas, devido às rotações necessárias.

Árvore Binária Simples

Apresentou desempenho muito ruim em dados ordenados e invertidos, chegando a 641 milhões de ns para 10.000 elementos — comportamento típico de pior caso (O(n) profundidade).

Insersões aleatórias foram muito melhores, pois evitam o desbalanceamento extremo.

3.2 Busca

Vetor (Sequencial)

Ótima para o primeiro elemento (70–730 ns).

Extremamente lenta para o último elemento e elementos aleatórios (até 159.510 ns).

Confirma complexidade O(n).

Vetor (Binária)

Tempos consistentes e muito baixos (270–1.640 ns).

Excelente escalabilidade.

Complexidade O(log n) comprovada.

Árvore AVL

Tempos muito baixos e estáveis (200–2.420 ns).

O melhor desempenho geral dentre as estruturas de árvore.

Complexidade O(log n) claramente observada.

Árvore Binária Simples

Desempenho extremamente variado.

Busca pelo último elemento pode levar 44.850 ns em árvores degeneradas.

Apenas dados aleatórios produzem buscas razoáveis.

3.3 Ordenação

Bubble Sort

Para entradas aleatórias e invertidas, apresentou tempos enormes:

- 89 milhões de ns para 10.000 aleatórios
- 63 milhões de ns para 10.000 invertidos

Confirmada complexidade $O(n^2)$ — **totalmente inviável para dados grandes**.

Merge Sort

Tempos muito menores e estáveis:

- ~1 a 1,6 milhões de ns para 10.000 elementos

Performance consistente com O(n log n).

4. Conclusão

Principais conclusões:

4.1. Inserção

- **Vetor** continua sendo a estrutura mais rápida.
- **AVL** apresenta comportamento estável e eficiente, exceto em certos padrões ordenados.
- **Árvore Binária Simples** sofre enormemente com desbalanceamento.

4.2. Busca

- Busca binária e AVL apresentam os melhores tempos.
- Busca sequencial se torna inviável conforme o tamanho aumenta.
- Árvore Binária só é boa quando os dados são aleatórios.

5. Ordenação

Bubble Sort é impraticável para grandes entradas.

Merge Sort é altamente eficiente e consistente.

6. Escolha da Estrutura

Para muitas buscas:

AVL ou vetor ordenado com busca binária.

Para muitas inserções rápidas:

Vetor.

Evitar árvore binária simples

quando houver dados ordenados.

Para grandes volumes:

sempre usar algoritmos de ordenação O(n log n).