Relatório de Análise de Algoritmos de Ordenação

Análise Comparativa de Desempenho: Bubble Sort, Insertion Sort e Quick Sort

Metodologia Experimental:
• Execuções: 10 testes completos independentes
• Linguagem: Java
• Medição: System.nanoTime()
• Resultados: Médias aritméticas com desvio padrão
• Cenários testados: 27 combinações (3 algoritmos × 3 tipos × 3 tamanhos)

Este relatório apresenta uma análise rigorosa e baseada em dados reais do desempenho de três algoritmos clássicos de ordenação: **Bubble Sort**, **Insertion Sort** e **Quick Sort**.

Os algoritmos foram submetidos a 10 execuções completas em conjuntos de dados com características distintas (aleatório, ordenado crescente e ordenado decrescente) e tamanhos variados (100, 1.000 e 10.000 elementos).

Resultados de Tempo de Execução

Valores em milissegundos (ms) - Médias de 10 execuções

Conjuntos de 100 elementos

Tipo de Conjunto	Bubble Sort	Insertion Sort	Quick Sort
Aleatório	1.659 ± 0.210	0.300 ± 0.047	0.097 ± 0.009
Crescente	0.002 ± 0.001	0.060 ± 0.022	0.112 ± 0.030
Decrescente	0.048 ± 0.016	0.022 ± 0.017	0.022 ± 0.010

Conjuntos de 1.000 elementos

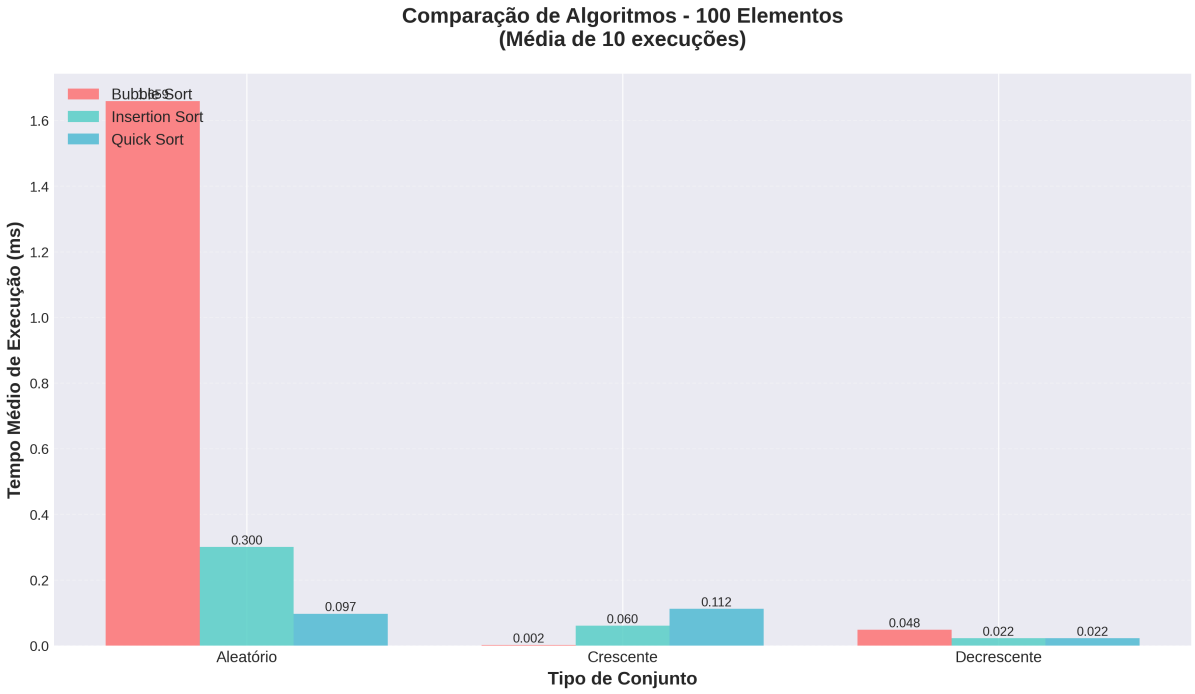
Tipo de Conjunto	Bubble Sort	Insertion Sort	Quick Sort
Aleatório	18.246 ± 1.769	9.528 ± 2.262	1.668 ± 0.884
Crescente	0.001 ± 0.000	0.117 ± 0.021	2.394 ± 0.610
Decrescente	10.496 ± 3.812	0.264 ± 0.160	0.652 ± 0.173

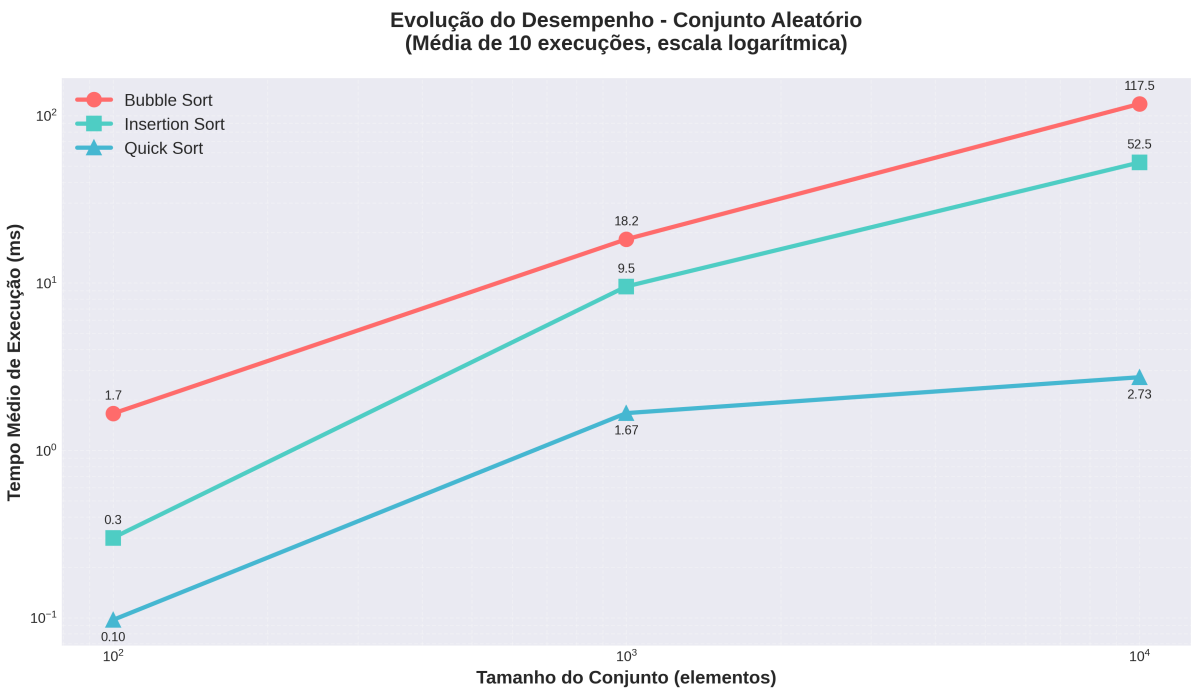
Conjuntos de 10.000 elementos

Tipo de Conjunto	Bubble Sort	Insertion Sort	Quick Sort
Aleatório	117.526 ± 9.802	52.482 ± 6.857	2.734 ± 0.649
Crescente	0.006 ± 0.001	0.661 ± 0.520	**60.340 ± 8.253**
Decrescente	78.065 ± 5.476	14.732 ± 1.408	38.304 ± 2.612

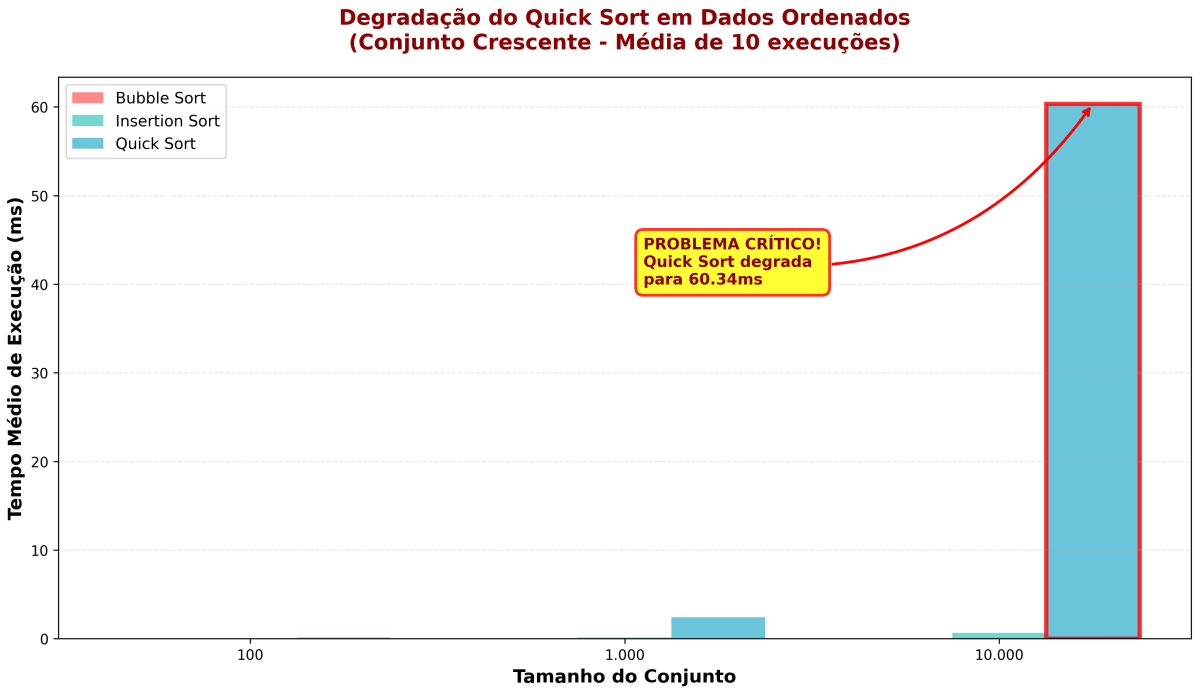
Nota: O valor destacado em vermelho indica degradação crítica de desempenho do Quick Sort em dados ordenados.

Visualização dos Resultados





Análise Crítica: Degradação do Quick Sort



Análise Detalhada dos Resultados

1. Bubble Sort

O Bubble Sort apresentou o pior desempenho geral entre os três algoritmos testados, confirmando sua inadequação para aplicações práticas com grandes volumes de dados.

Conjuntos Aleatórios: Demonstrou os maiores tempos de execução em todos os tamanhos testados, atingindo uma média de 117.53 ms (±9.80 ms) para 10.000 elementos. Este comportamento é esperado devido à complexidade $O(n^2)$ e à necessidade de múltiplas passagens pelo vetor.

Conjuntos Crescentes: Surpreendentemente, apresentou o melhor desempenho absoluto entre todos os algoritmos: apenas 0.006 ms (±0.001 ms) para 10.000 elementos. Esta eficiência extraordinária deve-se à otimização implementada que detecta quando não houve trocas, permitindo interrupção prematura.

Conjuntos Decrescentes: Representou o pior caso possível para o algoritmo, com 78.07 ms (±5.48 ms) para 10.000 elementos, onde cada elemento precisa ser movido através de todo o vetor.

2. Insertion Sort

O Insertion Sort demonstrou ser o algoritmo mais equilibrado, mantendo desempenho competitivo em múltiplos cenários e apresentando a menor variabilidade estatística.

Conjuntos Aleatórios: Alcançou desempenho intermediário, com 52.48 ms (±6.86 ms) para 10.000 elementos. Embora significativamente mais eficiente que o Bubble Sort, ainda sofre com a complexidade $O(n^2)$ em casos médios.

Conjuntos Crescentes: Apresentou excelente desempenho com apenas 0.661 ms (±0.520 ms) para 10.000 elementos, aproximando-se da complexidade $O(n)$ no melhor caso.

Conjuntos Decrescentes: Foi o algoritmo mais eficiente neste cenário, com 14.73 ms (±1.41 ms) para 10.000 elementos – 5.3x mais rápido que o Bubble Sort e 2.6x mais rápido que o Quick Sort.

3. Quick Sort

O Quick Sort apresentou resultados dicotômicos: excelência em cenários favoráveis e degradação crítica em casos específicos, revelando uma vulnerabilidade significativa na implementação testada.

Conjuntos Aleatórios: Demonstrou desempenho excepcional com apenas 2.73 ms (±0.65 ms) para 10.000 elementos, sendo 43x mais rápido que o Bubble Sort e 19x mais rápido que o Insertion Sort.

Conjuntos Crescentes - PROBLEMA CRÍTICO: Sofreu degradação catastrófica de desempenho, atingindo **60.34 ms (±8.25 ms)** para 10.000 elementos.

Este valor é:
• 22x pior que seu desempenho em dados aleatórios, e a sua causa raiz vem do fato que a implementação utiliza o último elemento como pivô. Em conjuntos ordenados, isto resulta em partições maximamente desbalanceadas, degradando a complexidade de $O(n \log n)$ para $O(n^2)$.