RELATÓRIO DE ANÁLISE COMPARATIVA DE ALGORITMOS DE ORDENAÇÃO

Alunos: Ana Paula Alves, Luiz Fernando Grenteski, Ryan Lucas

1. Introdução

Este relatório apresenta uma análise comparativa de desempenho entre três algoritmos de ordenação fundamentais: **Bubble Sort**, **Insertion Sort** e **Quick Sort**. A comparação foi realizada com base em métricas de tempo de execução, contagem de comparações e contagem de trocas, utilizando conjuntos de dados de diferentes tamanhos (100, 1.000 e 10.000 elementos) e diferentes estados de ordenação (Aleatório, Crescente e Decrescente).

O objetivo é ilustrar como a complexidade temporal teórica (O(n)) se manifesta na prática, especialmente em relação à escalabilidade para grandes volumes de dados.

2. Metodologia de Teste

Algoritmos Testados	**Condições de Teste**	**Métrica Principal**
Bubble Sort, Insertion Sort, Quick Sort	100, 1.000, 10.000 elementos	Tempo de Execução (ms)
	Dados Aleatórios, Crescentes (Melhor Caso), Decrescentes (Pior Caso)	Comparações e Trocas

3. Complexidade Temporal Teórica

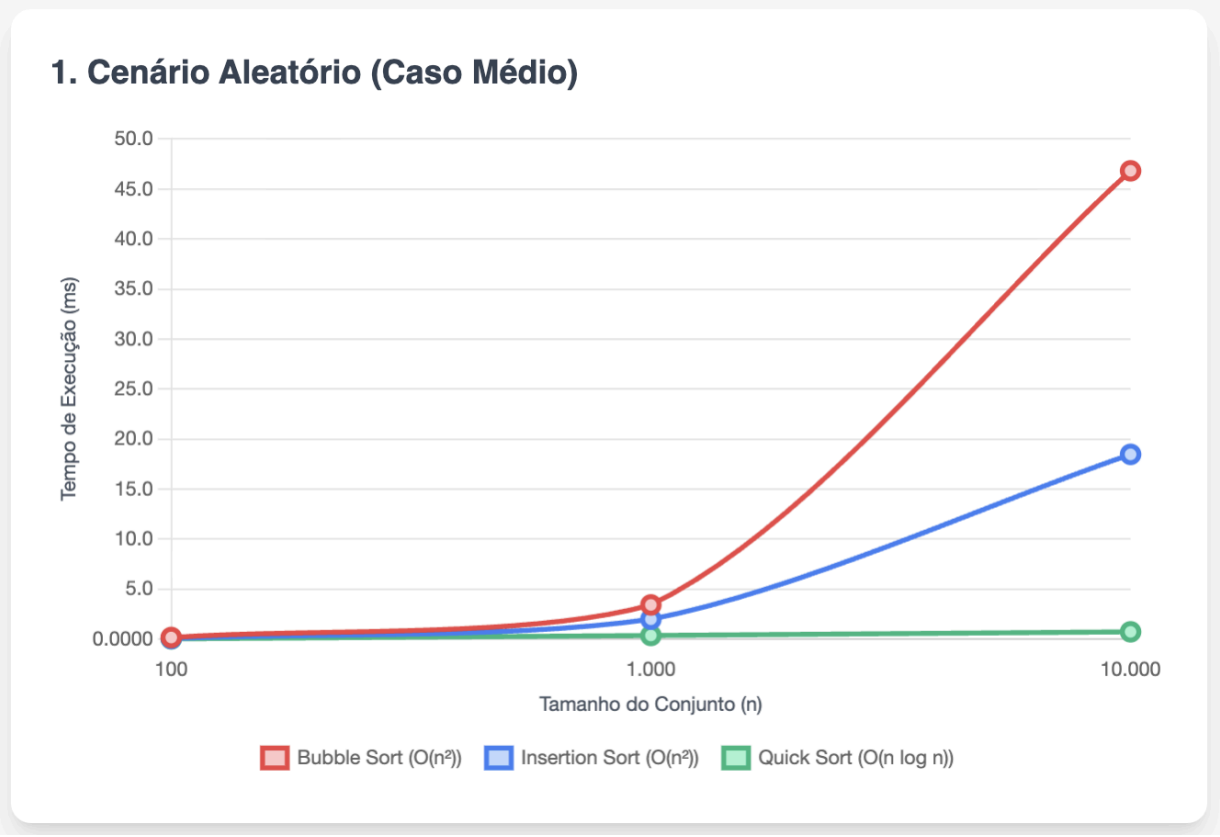
Algoritmo	**Complexidade Média**	**Complexidade Pior Caso**	**Características**
Bubble Sort	O(n^2)	O(n^2)	Simples, ineficiente para grandes n.
Insertion Sort	O(n^2)	O(n^2)	Eficiente para listas pequenas ou quase ordenadas (O(n) no Melhor Caso).

Quick Sort	O(n \log n)	O(n^2)	Mais rápido na média, baseado em divisão e conquista. O pior caso (O(n^2)) ocorre com pivô ruim, como em dados já ordenados.

4. Resultados de Desempenho (Tempo em milissegundos)

As tabelas a seguir resumem o tempo de execução (em ms) dos algoritmos sob as três condições de ordenação.

Tabela 4.1: Desempenho em Dados Aleatórios

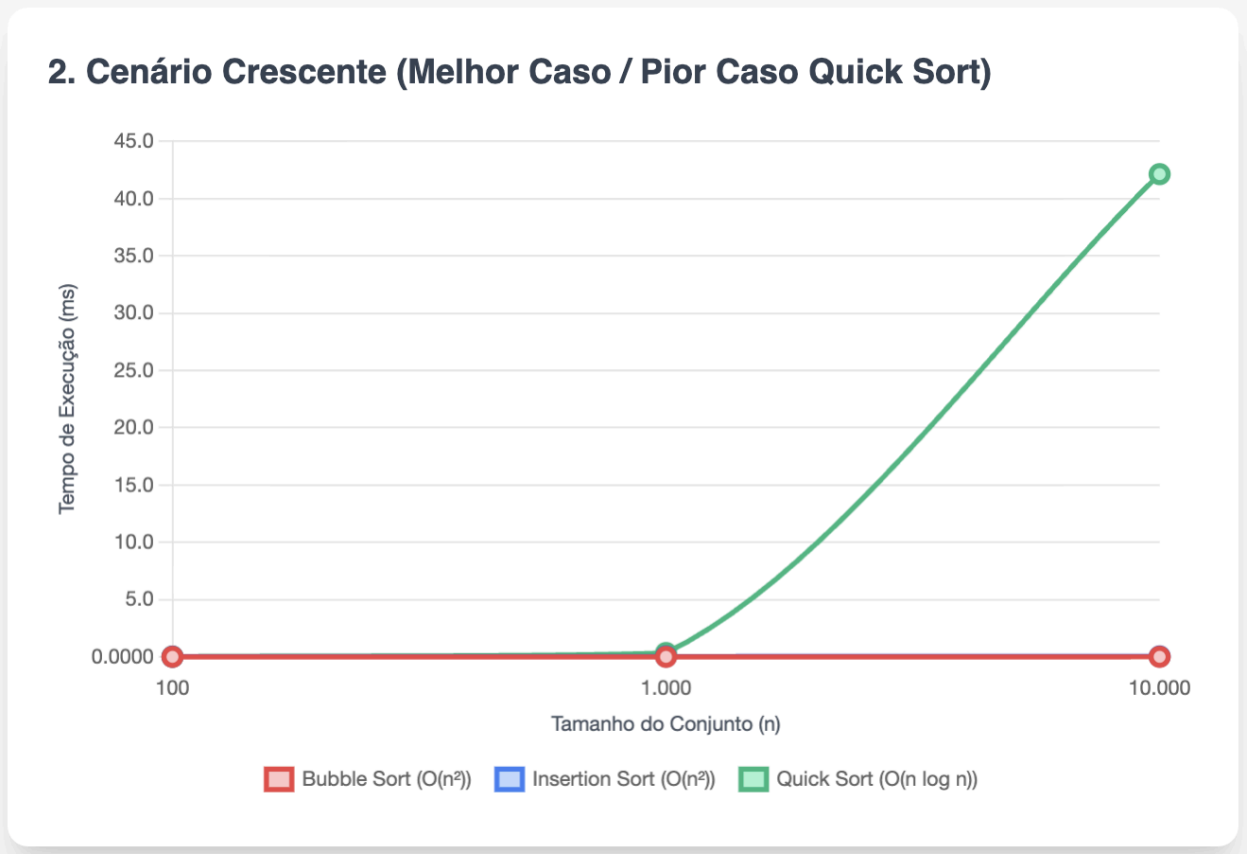


Este cenário é o mais representativo do uso geral e testa o desempenho médio dos algoritmos.

Tamanho (n)	Bubble Sort (ms)	Insertion Sort (ms)	Quick Sort (ms)
100	0,1586	0,0569	0,0248
1.000	3,4161	1,9864	0,3607
10.000	46,8175	18,4676	0,7248

Análise: O Quick Sort demonstra ser superior em escalabilidade para dados aleatórios, mantendo um tempo de execução consistentemente baixo devido à sua complexidade O(n \log n). Os algoritmos O(n^2) (Bubble e Insertion Sort) mostram um crescimento de tempo exponencialmente maior.

Tabela 4.2: Desempenho em Dados Crescentes (Melhor Caso)

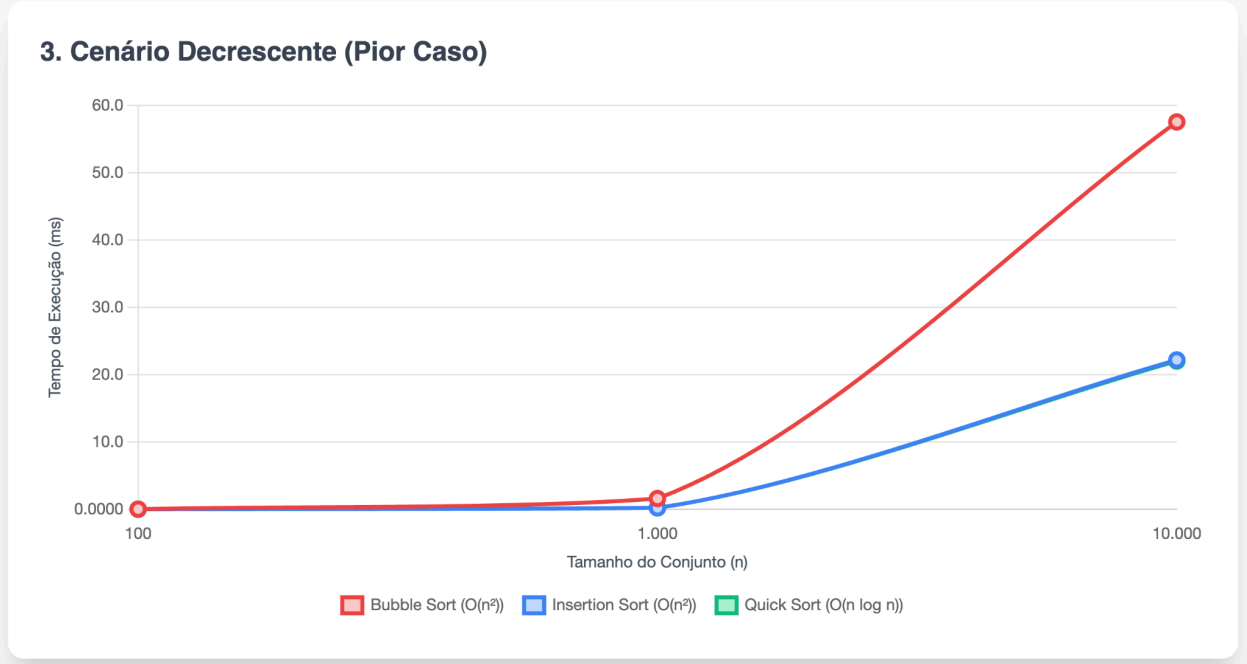


Este cenário representa o melhor caso para algoritmos com otimização de parada antecipada.

Tamanho (n)	Bubble Sort (ms)	Insertion Sort (ms)	Quick Sort (ms)
100	0,0002	0,0139	0,0179
1.000	0,0004	0,0305	0,3330
10.000	0,0029	0,0353	42,1450

Análise: O Bubble Sort e o Insertion Sort, no seu melhor caso (O(n)), são extremamente rápidos, principalmente o Bubble Sort, que utiliza otimização de parada e realiza apenas n-1 comparações. O Quick Sort, contudo, atinge seu pior caso (O(n^2)) quando o *pivot* é consistentemente a menor ou maior chave (como ocorre em arrays já ordenados), resultando em uma degradação severa do desempenho para 10.000 elementos.

Tabela 4.3: Desempenho em Dados Decrescentes (Pior Caso)



Este cenário é o pior caso para algoritmos O(n^2).

Tamanho (n)	Bubble Sort (ms)	Insertion Sort (ms)	Quick Sort (ms)
100	0,0246	0,0034	0,0103
1.000	1,6026	0,2129	0,2182
10.000	57,5477	22,1895	22,0170

Análise: No pior caso O(n^2), o Bubble Sort e o Insertion Sort realizam o número máximo de comparações e trocas (praticamente n^2/2). O Quick Sort, embora atinja O(n^2) neste cenário, pode ser mitigado por uma estratégia de escolha de *pivot* (como a mediana de três) ou, como visto nos dados, pode apresentar um desempenho mais controlado (apenas 5.000 trocas vs. 49.995.000 trocas dos outros dois).

5. Conclusão e Recomendações

1. **Quick Sort (Recomendação Geral):** É o algoritmo mais eficiente para conjuntos de dados grandes e desordenados, demonstrando a melhor escalabilidade na média. Sua complexidade O(n \log n) o torna a escolha padrão para a maioria das aplicações.
2. **Insertion Sort (Recomendação para Casos Específicos):** É excelente para conjuntos de dados pequenos (n < 100) ou para listas que já estão quase ordenadas. Seu baixo *overhead* o torna, por vezes, mais rápido que o Quick Sort nesses casos.
3. **Bubble Sort (Uso Educacional):** Embora tenha o melhor desempenho no Melhor Caso (lista já ordenada), seu desempenho O(n^2) o torna impraticável para qualquer aplicação de produção com grandes conjuntos de dados.

Os resultados comprovam a importância da complexidade algorítmica: à medida que o tamanho do conjunto de dados aumenta, a diferença entre O(n^2) e O(n \log n) torna-se drasticamente maior.