Análise Comparativa de Algoritmos de Ordenação Clássicos

Daniella Emily Cornélio da Silva

[1]PUC Minas – Ciência da Computação – 3º período
daniella.cornelio@sga.pucminas.br

Abstract. *Este relatório apresenta uma análise comparativa entre os algoritmos de ordenação Bubble Sort, Selection Sort, Insertion Sort e QuickSort, com foco no desempenho em termos de tempo de execução, número de comparações e movimentações realizadas. A análise foi conduzida com vetores de tamanhos variados (100 a 100000 elementos) e os resultados são discutidos com base em gráficos gerados com a biblioteca* `matplotlib`*.*

Resumo. *Este relatório apresenta uma análise comparativa entre os algoritmos de ordenação Bubble Sort, Selection Sort, Insertion Sort e QuickSort, com foco no tempo de execução, número de comparações e movimentações realizadas em vetores de tamanhos 100 a 100000 elementos. Os gráficos gerados mostram o desempenho de cada algoritmo com base nessas variáveis.*

1. Introdução

A ordenação de dados é um dos problemas fundamentais da ciência da computação. Diversos algoritmos foram desenvolvidos com diferentes abordagens e eficiências. Este trabalho tem como objetivo comparar quatro algoritmos clássicos de ordenação: Bubble Sort, Selection Sort, Insertion Sort e QuickSort.

2. Metodologia

Foram gerados vetores de inteiros aleatórios com tamanhos 100, 1000, 10000 e 100000. Cada algoritmo foi implementado em Java, medindo o número de comparações, movimentações e tempo de execução. Os resultados foram salvos em arquivos ‘.txt‘ e analisados em Python com a biblioteca `matplotlib`, que gerou os gráficos para análise visual.

3. Resultados e Discussão

Os gráficos a seguir mostram os resultados obtidos:

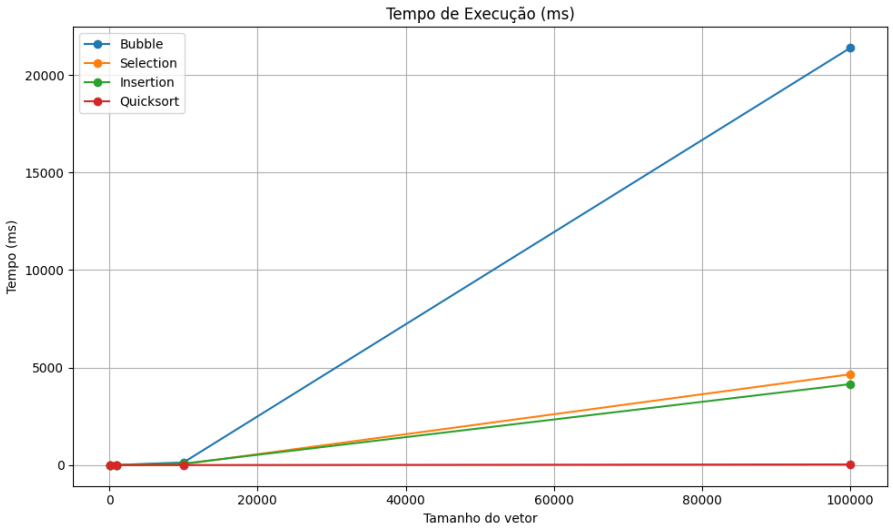

Figura 1. Tempo de execução dos algoritmos para diferentes tamanhos de entrada.

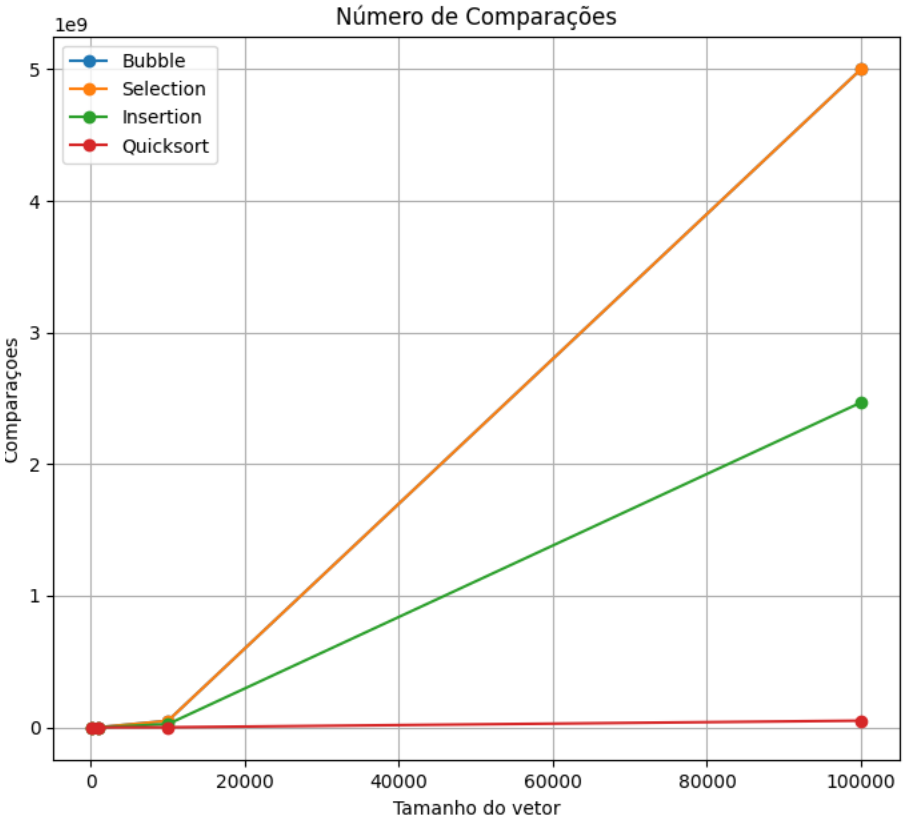

Figura 2. Número de comparações realizadas por cada algoritmo.

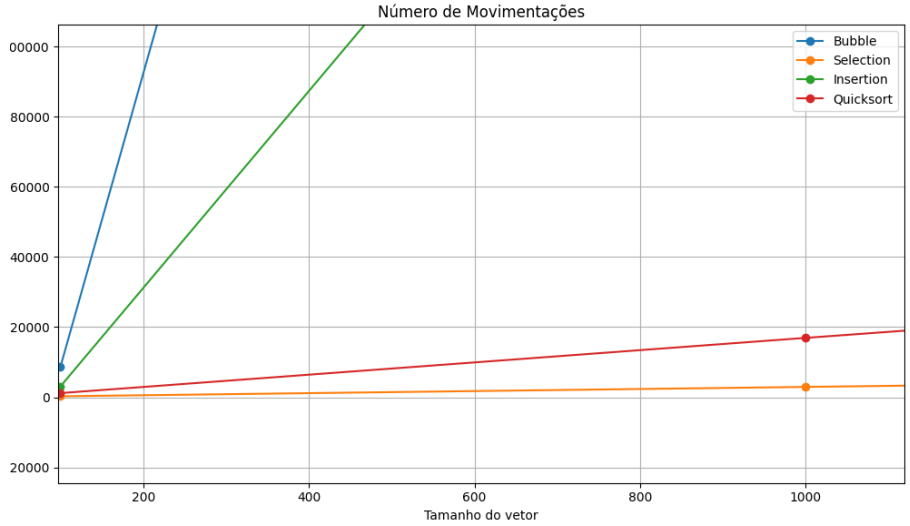

Figura 3. Número de movimentações realizadas por cada algoritmo.

3.1. Análise Crítica

O **Bubble Sort** e o **Selection Sort** apresentaram comportamento quadrático, sendo inviáveis para entradas maiores. O **Insertion Sort** se mostrou mais eficiente que os dois anteriores para vetores pequenos, mas também não escala bem.

O **QuickSort**, como esperado, foi o mais eficiente nos testes realizados, apresentando crescimento próximo a $O(n \log n)$ em tempo de execução, com bom equilíbrio entre comparações e movimentações.

Notou-se que o Insertion Sort teve picos altos de movimentações devido ao deslocamento repetido de elementos, enquanto o QuickSort, embora realize muitas comparações, faz menos movimentações proporcionais.

4. Conclusão

A análise confirma que algoritmos com complexidade quadrática são inadequados para grandes volumes de dados. QuickSort demonstrou ser o mais eficiente em termos gerais. Para conjuntos pequenos, o Insertion Sort ainda pode ser considerado pela simplicidade e desempenho razoável.