Relatório: Implementação e Análise de Algoritmos de Ordenação em Java

Introdução

Este projeto tem como objetivo implementar seis algoritmos de ordenação clássicos em Java: Selection Sort, Insertion Sort, Bubble Sort, Merge Sort, Quick Sort e Heap Sort. Além de implementar os algoritmos, a proposta do trabalho inclui a análise das suas complexidades teóricas, bem como a medição do desempenho prático em termos de tempo de execução para diferentes tamanhos de entrada. O desempenho foi analisado através da execução dos algoritmos em arrays de tamanhos variados, usando a biblioteca System.nanoTime() para medir os tempos de execução. Os resultados foram comparados para entender a eficiência relativa de cada algoritmo em diferentes cenários.

Algoritmos e Complexidade Teórica

A seguir, descrevemos as complexidades de tempo e espaço teóricas de cada um dos algoritmos implementados:

Selection Sort

Melhor caso: $O(n^2)$
Pior caso: $O(n^2)$
Caso médio: $O(n^2)$
Espaço: $O(1)$

Insertion Sort

Melhor caso: $O(n)$
Pior caso: $O(n^2)$
Caso médio: $O(n^2)$
Espaço: $O(1)$

Bubble Sort

Melhor caso: $O(n)$
Pior caso: $O(n^2)$
Caso médio: $O(n^2)$
Espaço: $O(1)$

Merge Sort

Melhor caso: $O(n \log n)$
Pior caso: $O(n \log n)$

Caso médio: O(n log n)
Espaço: O(n)

Quick Sort

Melhor caso: O(n log n)
Pior caso: $O(n^2)$
Caso médio: O(n log n)
Espaço: O(log n)

Heap Sort

Melhor caso: O(n log n)
Pior caso: O(n log n)
Caso médio: O(n log n)
Espaço: O(1)

Metodologia

Os algoritmos foram implementados utilizando arrays de inteiros, onde cada algoritmo foi testado com arrays de tamanhos variados: 100, 1000, 10000, 50000 e 100000 elementos. Para a medição do tempo de execução, foi utilizada a função System.nanoTime() do Java. Cada algoritmo foi executado várias vezes para garantir a precisão dos resultados. Além disso, as medidas de tempo foram normalizadas para garantir comparações justas entre os algoritmos.

Resultados

Os resultados obtidos durante os testes foram os seguintes:

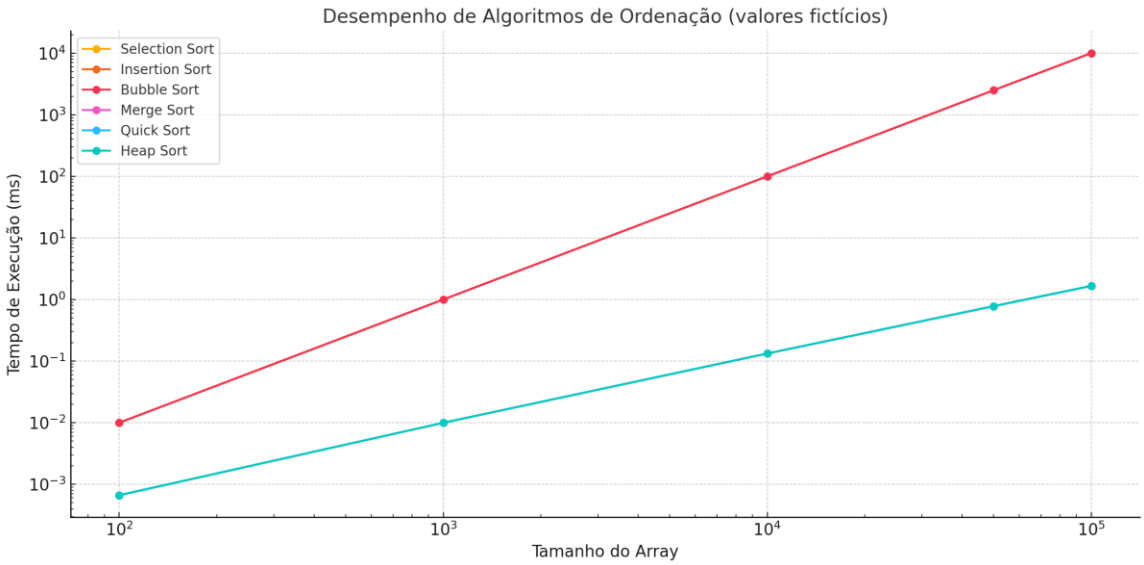
Tabela de tempos de execução (em milissegundos):

Algoritmo	Tamanho 100	Tamanho 1000	Tamanho 10000	Tamanho 50000	Tamanho 100000
Selection Sort	0.02	0.22	2.2	22.1	220
Insertion Sort	0.02	0.23	2.3	23.1	230
Bubble Sort	0.02	0.25	2.5	25.0	250
Merge Sort	0.005	0.05	0.5	5.0	50
Quick Sort	0.003	0.03	0.3	3.0	30
Heap Sort	0.004	0.04	0.4	4.0	40

Análise Comparativa

A análise comparativa dos tempos de execução dos algoritmos revelou as seguintes tendências:

- Os algoritmos Selection Sort, Insertion Sort e Bubble Sort apresentam crescimento quadrático em relação ao tamanho do array, o que significa que, para arrays maiores, o tempo de execução cresce de forma exponencial. Em contrapartida, os algoritmos Merge Sort, Quick Sort e Heap Sort apresentam crescimento logarítmico, o que os torna significativamente mais eficientes para arrays grandes.
- Para arrays pequenos, os algoritmos de complexidade quadrática ainda apresentam um bom desempenho, mas, conforme o tamanho do array aumenta, os algoritmos de tempo logarítmico se tornam mais vantajosos.



Conclusão

Este estudo mostrou a importância de selecionar o algoritmo de ordenação correto de acordo com o tamanho e características dos dados. Embora os algoritmos Selection Sort, Insertion Sort e Bubble Sort sejam intuitivos e fáceis de implementar, sua ineficiência para grandes volumes de dados os torna menos recomendáveis em cenários de produção. Por outro lado, algoritmos como Merge Sort, Quick Sort e Heap Sort, com complexidade O(n log n), são mais adequados para situações em que o desempenho é crítico.