UNIRUY WYDEN - ALGORITMOS E COMPLEXIDADE

Sistema de Vendas com Ranking e Gerenciamento

Uma análise prática de performance de algoritmos de ordenação.

Docente: Prof. MSc Eng Heleno Cardoso

Alunos: Arthur Santos Oliveira, Antonio Victor Gonçalves Carvalho

Visão Geral do Projeto

Desenvolvemos um sistema que simula um **e-commerce** real para testar o comportamento de algoritmos com grandes volumes de dados.

- **Objetivo:** Ordenar rankings de vendas e comparar performance.
- **Frontend:** React.js + Tailwind CSS (Vercel).
- **Backend:** Java Spring Boot (Railway).
- **Banco de Dados:** Postgres.

Estruturas de Dados Utilizadas

Array / List

Utilizado para armazenamento sequencial de vendas, produtos e clientes. Permite ordenação eficiente e iteração simples para os algoritmos de sort.

HashMap (Cache)

Implementado para cache de dados. Garante busca de complexidade O(1) por quantidade, evitando requisições repetitivas ao banco.

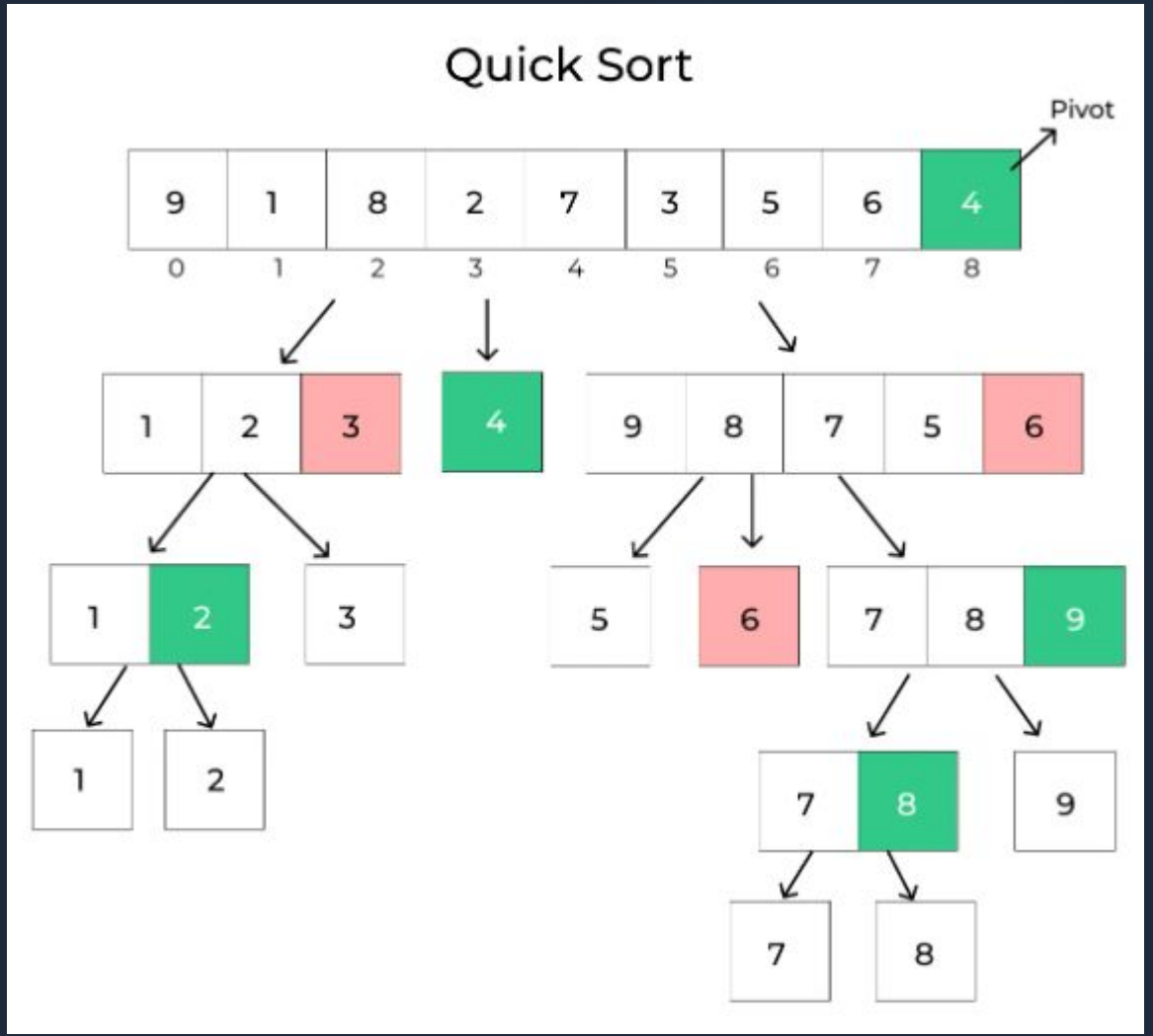
Ranking Ordenado

Estrutura específica (List) mantida para exibir o resultado final das vendas ordenadas pelo valor total após a execução dos algoritmos.

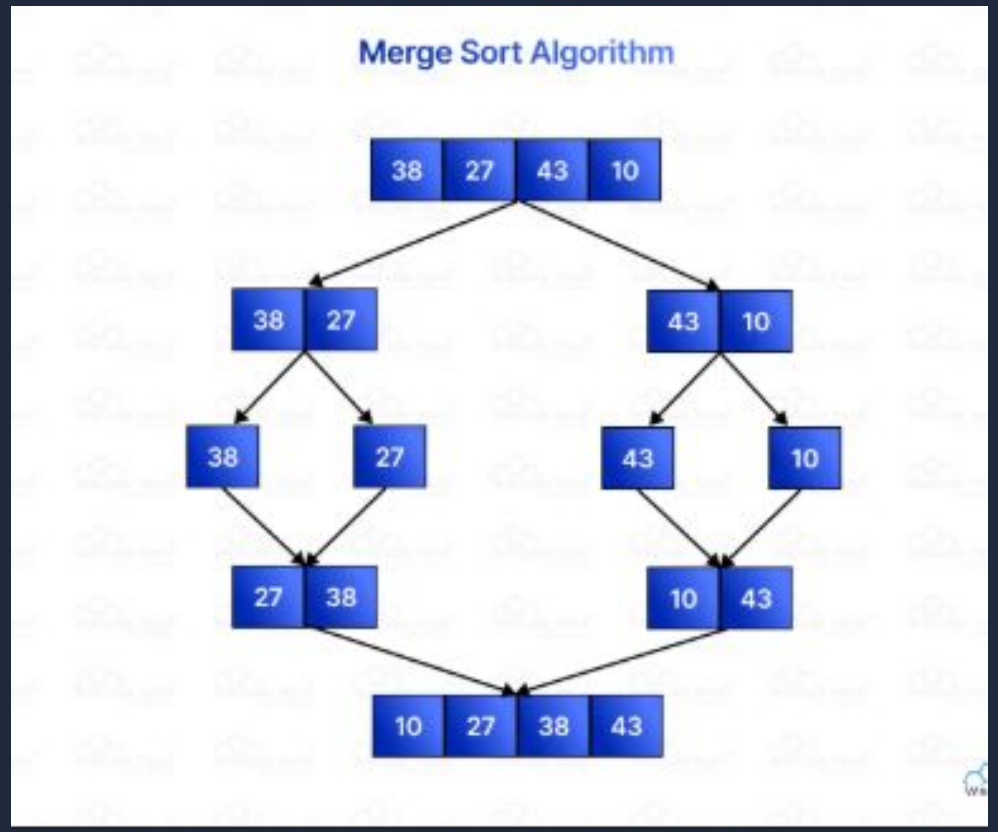
QuickSort

Algoritmo de **Divisão e Conquista**. Escolhe um elemento "pivô" e particiona o array em elementos menores e maiores que o pivô.

- **Melhor/Médio Caso:** $O(n \log n)$ - Distribuição equilibrada.
- **Pior Caso:** $O(n^2)$ - Array já ordenado (pivô no extremo).
- **Observação:** Foi o algoritmo mais eficiente nos testes práticos do projeto.



MergeSort



Consistência e Estabilidade

Também utiliza divisão e conquista, mas foca na fusão (merge) de sub-arrays ordenados. Garante performance previsível.

Complexidade: $O(n \log n)$

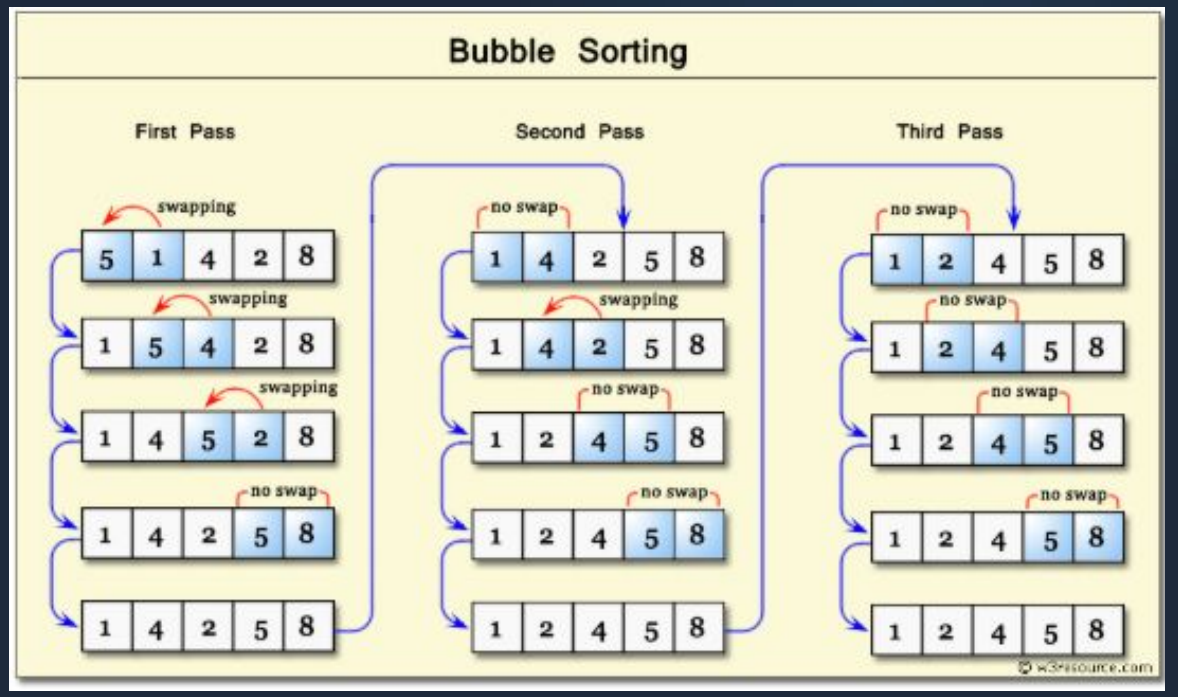
Constante em todos os casos (Melhor, Médio e Pior), porém consome mais memória auxiliar que o QuickSort.

BubbleSort

O Caso Base

Algoritmo simples de troca adjacente. Percorre a lista repetidamente, flutuando o maior valor para o topo.

- **Complexidade:** O (n 2) no caso médio e pior.
- **Desempenho:** Mostrou-se significativamente mais lento que o QuickSort e MergeSort conforme o volume de dados aumentou no ranking de vendas.



Busca e Otimização

Busca Linear

Utilizada para encontrar vendedores e produtos específicos.

Complexidade: O (n)

Percorre o array sequencialmente. Simples, mas pode ser lenta em grandes volumes se o item estiver no final.

Caching (HashMap)

Utilizado para armazenar listagens pré-processadas.

Complexidade: O (1)

Permite acesso instantâneo aos dados, eliminando a necessidade de reprocessar ou buscar linearmente informações frequentes.

Análise Comparativa de Complexidade

Algoritmo	Melhor Caso	Caso Médio	Pior Caso
QuickSort	$O(n\log n)$	$O(n\log n)$	$O(n^2)$
MergeSort	$O(n\log n)$	$O(n\log n)$	$O(n\log n)$
BubbleSort	$O(n)$	$O(n^2)$	$O(n^2)$
Busca Linear	$O(1)$	$O(n)$	$O(n)$

Recursividade e Recorrência

Equações de Recorrência

Tanto QuickSort quanto MergeSort utilizam recursão para dividir o problema.

Fórmula Geral:

$$T(n) = 2T(n/2) + O(n)$$

Isso significa que o tempo para ordenar n elementos é igual a duas vezes o tempo para ordenar metade deles, mais o tempo linear para dividir/combinar.

Isso resulta na complexidade:

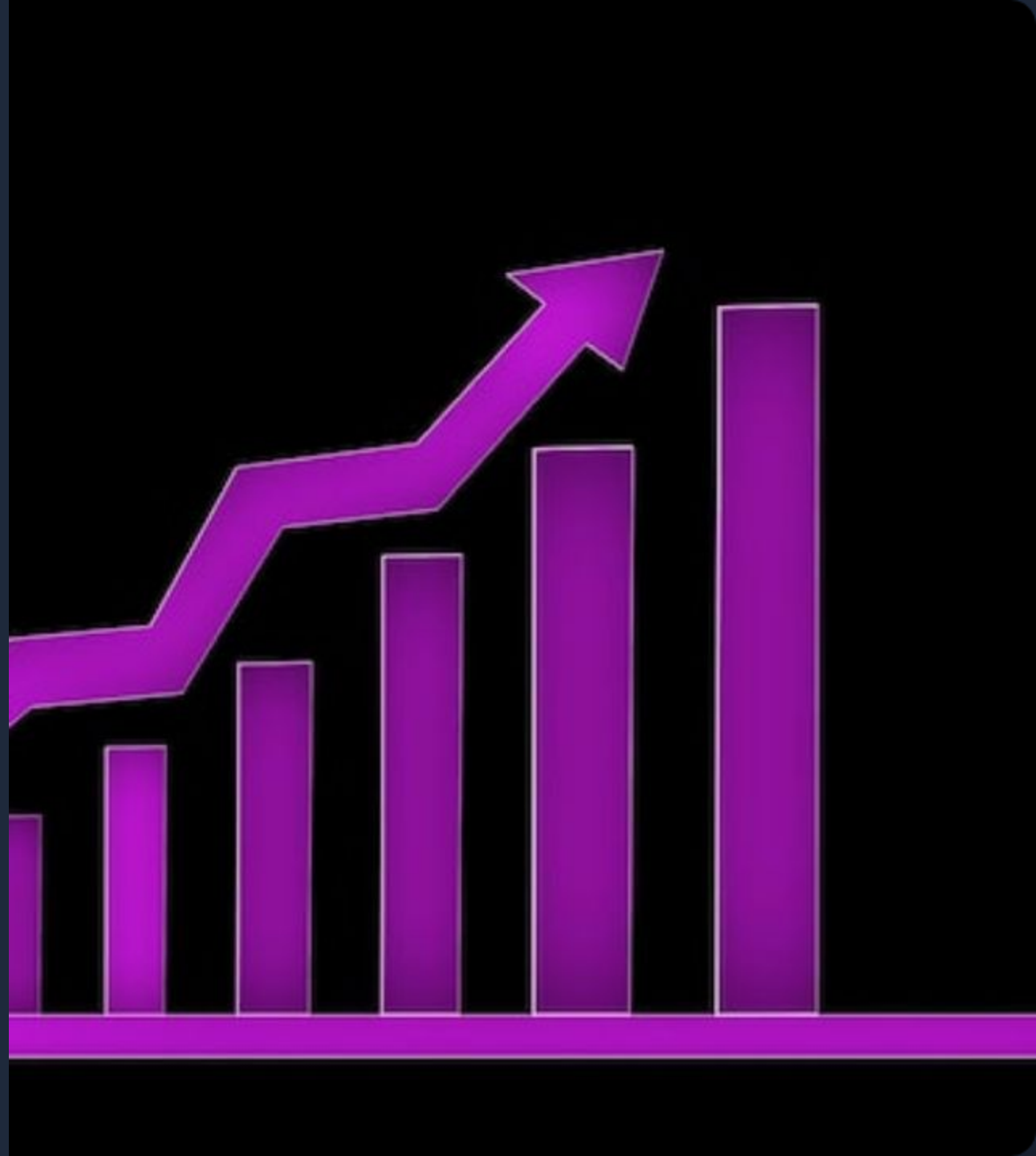
$$O(n \log n)$$

Conclusão dos Testes

A análise prática do sistema confirmou a teoria:

- **QuickSort:** Melhor desempenho geral na ordenação do ranking.
- **BubbleSort:** Inviável para grandes volumes de dados.
- **Busca Linear:** Eficiente para o escopo, mas com possibilidade de melhoria futura (Busca Binária).

Futuro: Implementação de Busca Binária para dados pré-ordenados.



Perguntas?

Obrigado!

Link do Projeto: sales-rank-algoritmos.vercel.app