Laboratório de Estrutura de Dados

Primeira versão do projeto da disciplina

Comparação entre os algoritmos de ordenação elementar

Otávio Ferreira Clementino, 2024

1. Introdução

Este relatório corresponde ao relato dos resultados obtidos no projeto da disciplina de LEDA que tem por objetivo fazer uma análise comparativa entre os algoritmos de ordenação elementares.

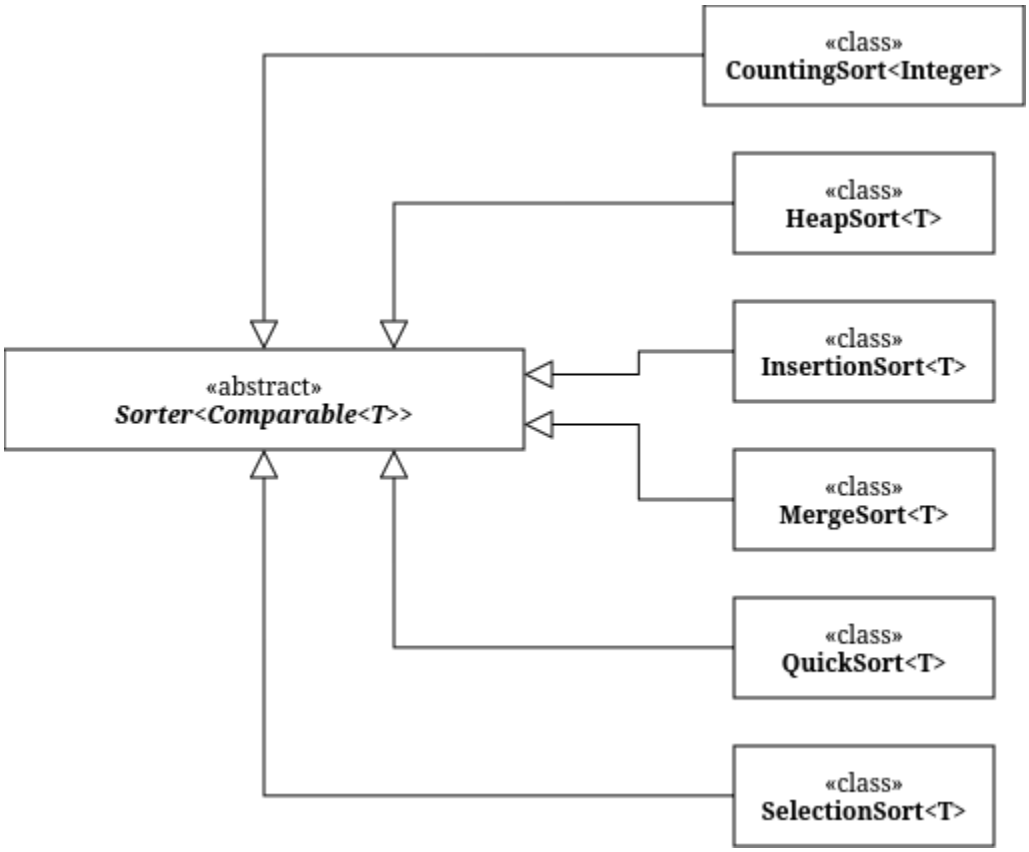
Para esse fim, foi implementado um projeto de exploração de dados no qual através da aplicação prática dos algoritmos de ordenação sobre um conjunto de dados real foi possível verificar a performance desses algoritmos.

(resumo dos principais resultados)

(apresentação de todas as seções do documento)

2. Descrição geral sobre o método utilizado

A ferramenta foi implementada em Java versão 1.8, usando a plataforma *maven* para automação de *build*. Foi usada uma superclasse abstrata como base para a implementação dos algoritmos de ordenação. Foi necessário configurar o tamanho máximo do *heap* da JVM para 2048M através das configurações da IDE.



Implementação dos algoritmos de ordenação

As classes derivadas da classe abstrata *Sorter* processam *array*s de tipos genéricos comparáveis, e retornam um array de inteiros que indicam a posição de cada elemento.

A ordenação dos dados ocorre através do processamento de colunas, as quais são extraídas do arquivo em disco e ordenadas usando os métodos de ordenação. O *array* de inteiros resultante é usado para reorganizar as linhas do arquivo original, o qual é salvo novamente para o disco.

Descrição geral do ambiente de testes

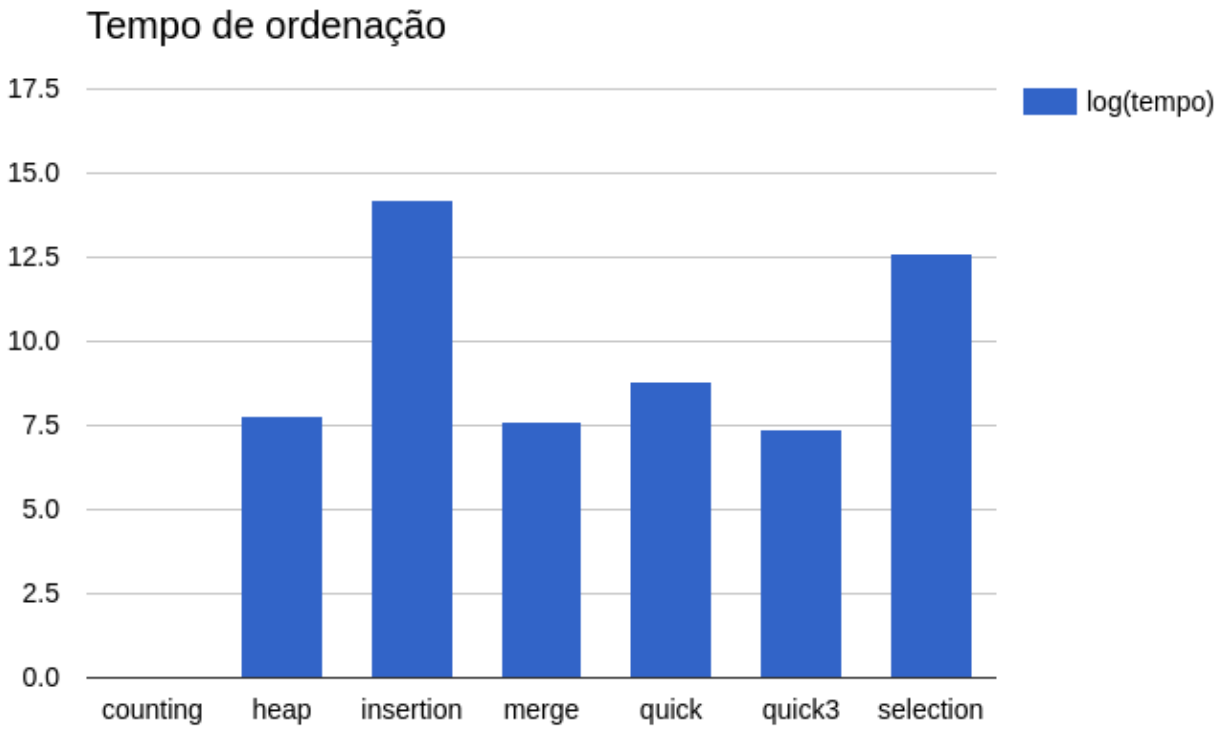
Os sistemas utilizados foram um computador do tipo *notebook* com o processador AMD A12-9720P, de quatro núcleos e arquitetura *x86-64*, 8 GB (*gigabytes*) de memória e 120 GB de armazenamento em disco. O sistema operacional utilizado foi o *linux*, versão 6.11.0, distribuição CachyOS.

3. Resultados e Análise

Elaborar os resultados dos testes usando tabelas e gráficos

Ordenação pela data do acidente (coluna *date*) em ordem decrescente. Tempo de execução medido em milissegundos.

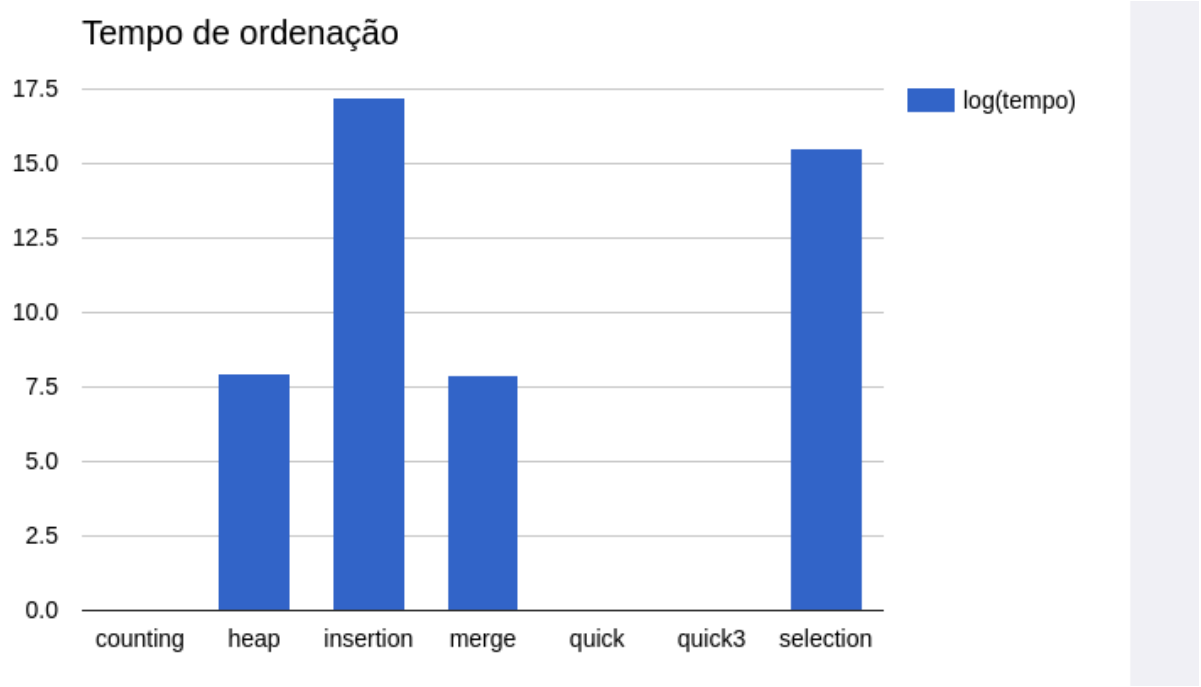
Counting	Heap	Insertion	Merge	Quick	Quick-Mediana de 3	Selection
Não aplicável	2396 ms	1508713 ms	2047 ms	6568 ms	1614 ms	2999972 ms



O gráfico reforça a disparidade em eficiência dos métodos recursivos sobre os métodos tradicionais. Dentre os métodos recursivos, o *quick-sort* foi o mais demorado, justificado pela variação de complexidade entre caso médio *O(log n)* e pior caso *O(n^2)*.

Ordenação pela coluna *time* em ordem crescente. Tempo de execução medido em milissegundos.

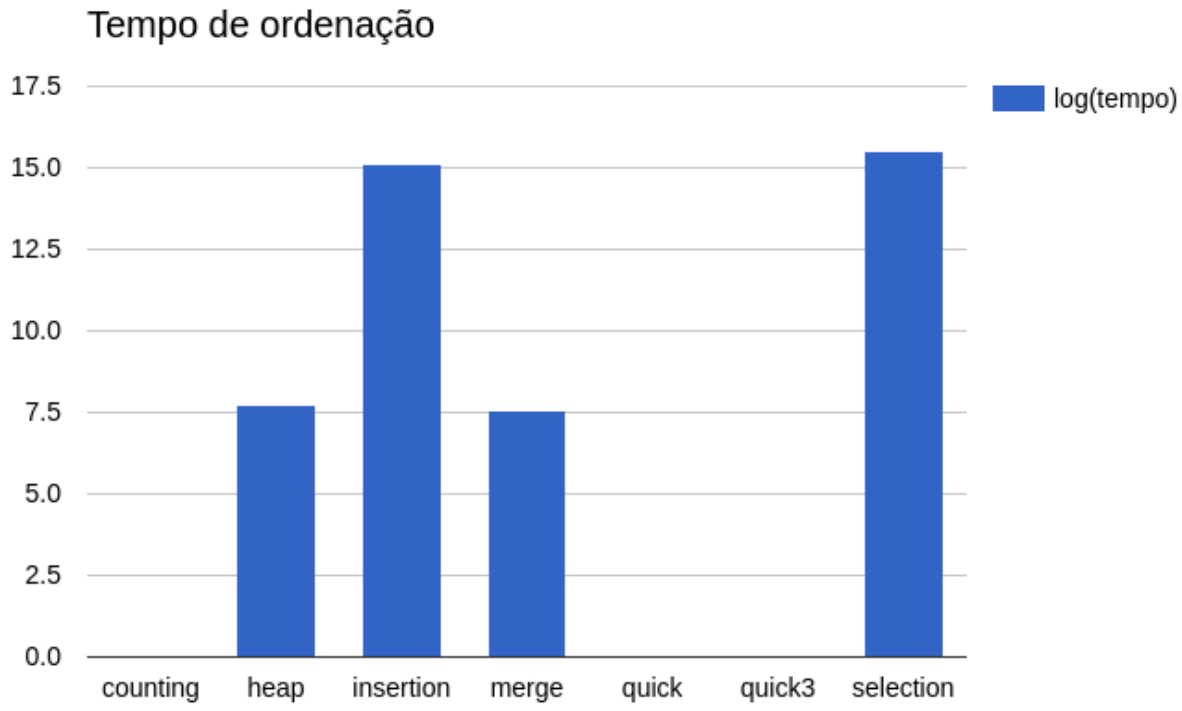
Counting	Heap	Insertion	Merge	Quick	Quick-Mediana de 3	Selection
Não Aplicável	2805 ms	30422157 ms	2687 ms	**Estouro da *Stack***	**Estouro da *Stack***	5409667 ms



Mais uma vez os algoritmos *heap-sort* e *merge-sort* são rápidos. A variação de uso de espaço no *quick-sort* aparente pelo *StackOverflowError* durante a execução, provavelmente devido à partição desbalanceada causada pelo algoritmo *partition()*.

Ordenação pelo tipo de comunicação (coluna *communication_kind*) em ordem alfabética crescente. Tempo de execução medido em milissegundos.

Counting	Heap	Insertion	Merge	Quick	Quick-Mediana de 3	Selection
Não Aplicável	2330 ms	3784142 ms	1895 ms	**Estouro da *Stack***	**Estouro da *Stack***	5430091 ms



Os resultados demonstram a eficiência dos algoritmos recursivos sobre os iterativos, sobretudo do algoritmo *merge-sort*. O algoritmo *heap-sort* foi o segundo melhor em termo de rapidez de execução, seguido pelo *quick-sort*, o pior dentre os algoritmos recursivos devido à variação de complexidade de uso de espaço e tempo que o algoritmo apresenta. O *insertion-sort* foi o melhor algoritmo iterativo, e o *selection-sort* o pior devido ao excesso de comparações necessárias para se obter o menor elemento em cada iteração, o que causa o caso médio de execução de se aproximar do pior caso mais frequentemente.