Universidade Estadual da Paraíba – UEPB
Centro de Ciências e Tecnologia – CCT
Componente Curricular: Laboratório de Estrutura de Dados
Período: 2023.1
Professor: Fábio Leite
Turma: MANHÃ

José Jamilson Ferreira da Silva - 211080098

Noiana de Paula Noia - 222080469

Relatório – Segundo Projeto de Laboratório de Estrutura de Dados

CAMPINA GRANDE - PB

30 de Junho de 2022

Sumário

1. Introdução

Este relatório diz respeito ao segundo projeto da disciplina de *Laboratório de Estruturas de Dados*. O desenvolvimento desse projeto teve como principal intuito manipular e analisar as aplicações dos algoritmos de ordenação (QuickSort, MergeSort e HeapSort) em contextos do mundo real, para tanto, foi possível compreender melhor o funcionamento desses algoritmos, ampliando assim o nosso conhecimento a despeito de tais comportamentos.

O código foi escrito em Java, e manipula e modifica as informações contidas em um arquivo .csv (*b3_stocks_1994_2020.csv*). Esse arquivo corresponde às informações das negociações da BOVESPA (Bolsa de Valores brasileira) nos períodos entre 1994 à 2020.

As modificações que são realizadas no arquivo *b3_stocks_1994_2020.csv* são as seguintes:

A. Cria um arquivo b3stocks_T1.csv, no qual o formato da data é DD/MM/AAAA (antes estava no formato AAAA/MM/DD);
B. Cria um arquivo b3stocks_F1.csv, no qual fica apenas um registro por dia, sendo aquele com o maior volume;
C. Faz uma filtragem no arquivo b3stocks_T1.csv, deixando apenas os registros com volume acima da média diária.

O código também realiza algumas ordenações no arquivo *b3stocks_T1.csv* são as seguintes:

A. Ordena os tickers por ordem alfabética;
B. Ordena o volume por ordem crescente;
C. Ordena as variações em ordem decrescente.

Para analisar a eficiência dos diversos algoritmos de ordenação, as tarefas acima foram executadas usando os seguintes algoritmos: Quick Sort, Merge Sort e Heap Sort. Cada um desses algoritmos foram executados 3 vezes, de modo a se obter o caso médio, o melhor caso e o pior caso. Para cada uma dessas execuções, o código gera um novo arquivo .csv, que mostra as informações ordenadas

2. Método utilizado

O teste foi feito utilizando o arquivo b3_stocks_1994_2020.csv, que possui 1883204 linhas de registro.

Foram utilizadas ao total 13 classes para escrever o programa. A função de cada uma delas é a seguinte:

- **FiltrarRegistro:** cria o arquivo b3stocks_F1.csv;
- **TransformarData:** cria o arquivo b3stocks_T1.csv;
- **FiltrarMediaDiaria:** faz a filtragem no arquivo b3stocks_T1.csv;
- **Registro:** define o registro e seus campos (data, ticker, open, close, high, low, volume e linha);
- **Funcoes:** define as principais funções que serão utilizadas pelas demais classes herdeiras;
- **QuickSort:** ordena o arquivo utilizando quick sort;
- **MergeSort:** ordena o arquivo utilizando merge sort;
- **HeapSort:** ordena o arquivo utilizando heap sort;
- **Main:** executa o código. Para capturar o tempo de execução de cada algoritmo, foi utilizada a função do java currentTimeMillis.

Para capturar o tempo de execução de cada algoritmo, foi utilizada a função do java currentTimeMillis.

O teste foi realizado em um notebook com processador Intel Core i7-8565U, com sistema operacional Windows 11 64 bits.

3. Resultados

Com base nos resultados de três ordenações distintas, apresentamos a seguir tabelas comparativas do tempo de execução em milissegundos (ms) de algoritmos de

ordenação. As ordenações realizadas foram: a ordenação alfabética dos registros com base em seus tickets, a ordenação crescente dos registros com base em seus volumes, e a ordenação decrescente dos registros com base em suas variações diárias.

Os algoritmos foram testados em três cenários diferentes: o melhor caso, o pior caso e o caso médio. É importante ressaltar que o algoritmo counting sort não aparece nas tabelas, pois ele não é capaz de ordenar caracteres alfabéticos (o que o torna inválido para a primeira ordenação) nem valores flutuantes (o que o torna inválido para a segunda e terceira ordenações).

Tabela 1: Ordenação de tickers em ordem alfabética

	Caso Médio	Melhor Caso	Pior Caso
Quick Sort	**15 ms**	**40 ms**	**132 ms**
Merge Sort	**8 ms**	**5 ms**	**9 ms**
Heap Sort	**7 ms**	**4 ms**	**6 ms**

Tabela 2: Ordenação dos volumes em ordem crescente

	Caso Médio	Melhor Caso	Pior Caso
Quick Sort	**6 ms**	**200 ms**	**220 ms**
Merge Sort	**8 ms**	**7 ms**	**6 ms**
Heap Sort	**16 ms**	**4 ms**	**4 ms**

Tabela 3: Ordenação de variações diárias em ordem decrescente

	Caso Médio	Melhor Caso	Pior Caso
Quick Sort	**79 ms**	**74 ms**	**54 ms**
Merge Sort	**10 ms**	**9 ms**	**6 ms**
Heap Sort	**10 ms**	**4 ms**	**6 ms**

Logo a seguir os gráficos comparam o tempo de execução dos algoritmos em cada uma das ordenações:

Gráfico 1: Ordenação dos tickers

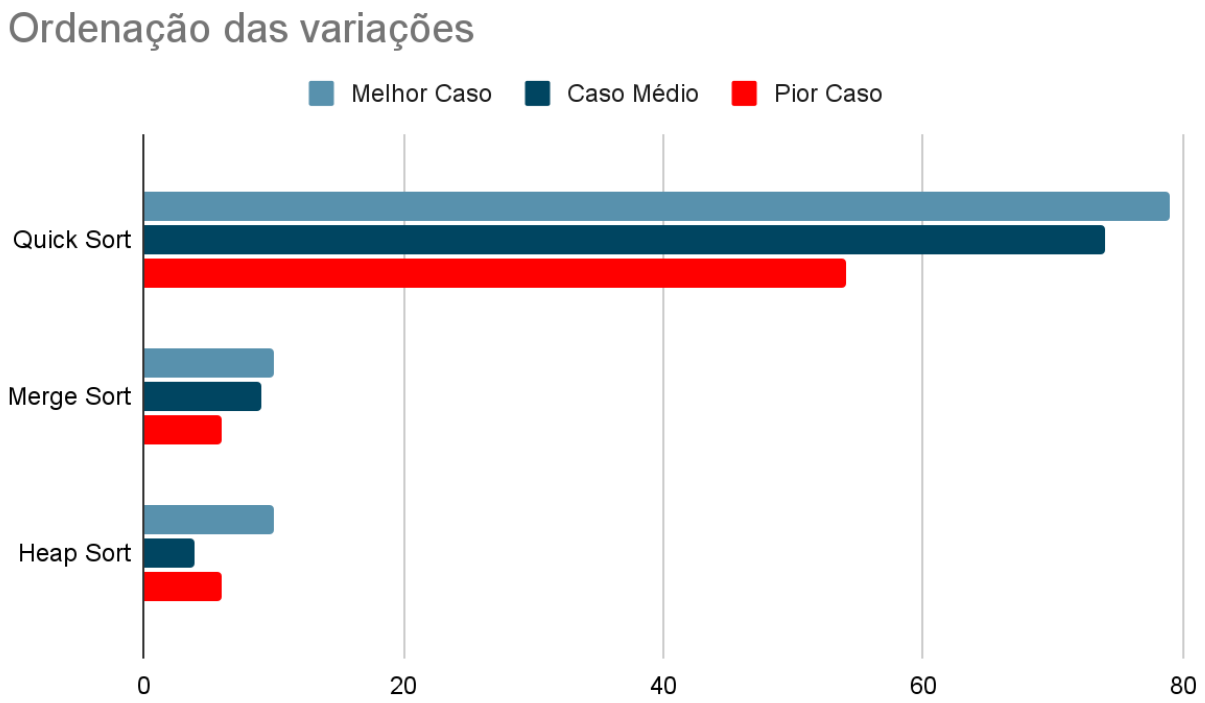


Gráfico 2: Ordenação dos volumes

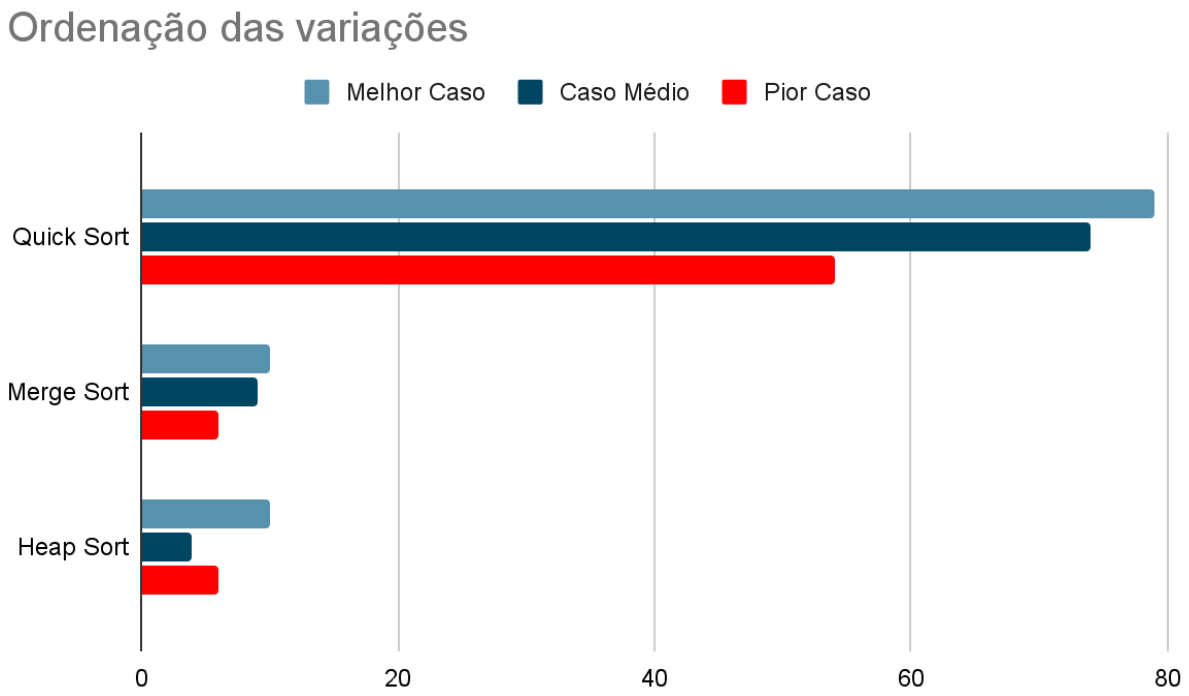
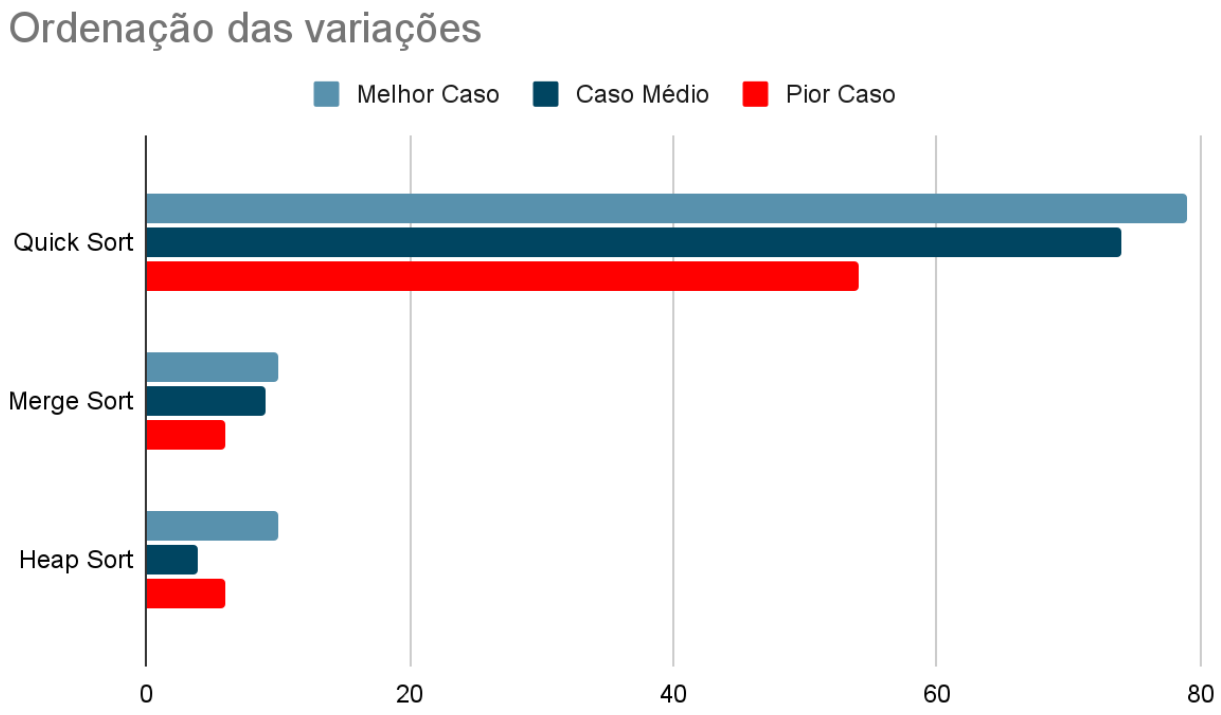


Gráfico 3: Ordenação das variações



Ao examinar os gráficos fornecidos, fica claro que o Heapsort e o Merge Sort se destacaram como os algoritmos mais eficazes em geral, pois possuem complexidade O(n*log(n)) em todos os três casos. Embora o Quicksort também tenha complexidade

O(n*log(n)) no melhor e caso médio, sua eficiência foi inferior à do Heapsort e do Merge Sort devido ao grande tamanho do array de entrada (quase 2 milhões de registros), o que diminuiu a eficiência do Quicksort.

Assim sendo, pode-se constatar que os algoritmos de ordenação se comportam de acordo com a tabela de complexidade de tempo, conforme descrito no livro "Algoritmos - Teoria e Prática" escrito por Thomas H. Cormen e Charles E. Leiserson, publicado em 2012.