UEPB
Universidade Estadual da Paraíba
Departamento de Computação – DC
Laboratório de estrutura de dados
Professor: Fabio Luiz Leite Junior

ARTHUR MARQUES ARAÚJO

ASCENDINO MARTINS DE AZEVEDO NETO

KEVIN SANTOS MARQUES

RELATÓRIO 2: BOLSA DE VALORES BOVESPA 1994-2020

CAMPINA GRANDE – PB

20 de novembro de 2024

ARTHUR MARQUES ARAÚJO

ASCENDINO MARTINS DE AZEVEDO NETO

KEVIN SANTOS MARQUES

RELATÓRIO 2: BOLSA DE VALORES BOVESPA 1994-2020

Relatório 2 apresentado no curso de Ciência da Computação da Universidade Estadual da Paraíba e na disciplina de Laboratório de Estrutura de dados referente ao período 2024.2

Professor: Fabio Luiz Leite Junior

CAMPINA GRANDE – PB

20 de novembro de 2024

SUMÁRIO

INTRODUÇÃO

O presente relatório tem como objetivo dar continuidade ao estudo iniciado na primeira etapa do projeto, em que foi realizada uma análise comparativa de diferentes algoritmos de ordenação aplicados na análise de dados de negociações realizadas na BOVESPA (Bolsa de Valores Brasileira), no período compreendido entre 1994 e 2020. Naquela fase, foram abordados os algoritmos Selection Sort, Insertion Sort, Quick Sort, Merge Sort, Counting Sort, Heapsort e Quick Sort com Mediana 3, todos implementados utilizando exclusivamente a estrutura de dados array.

Na presente fase do projeto, a abordagem foi ampliada incorporando as três melhores estruturas de dados da primeira parte do projeto, o Quick Sort, o Merge Sort e o Heap Sort, com o objetivo de otimizar a eficiência e a usabilidade dos algoritmos de ordenação. Esses algoritmos foram executados em diferentes cenários, analisando o desempenho em termos de tempo de execução e complexidade computacional nos casos médios, melhores e piores. Para cada uma das execuções, novos arquivos CSV foram gerados, apresentando os dados em diferentes formas de ordenação: os tickers foram ordenados alfabeticamente, os volumes negociados de forma crescente, e as variações de preço, de maneira decrescente.

Os resultados obtidos demonstram a relevância da escolha do algoritmo adequado para diferentes tipos de dados e critério de desempenho, contribuindo assim para a análise do comportamento dos algoritmos e para uma compreensão mais profunda das operações de mercado na BOVESPA ao longo do período investigado.

METODOLOGIA

Para realizar este estudo, todos os algoritmos foram implementados utilizando a linguagem Java (versão JDK-20) e as implementações foram executadas por meio do IntelliJ IDEA (versão 2023.2.5), visando manipular e modificar 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) entre os anos de 1994 à 2020.

As modificações realizadas no arquivo "b3_stocks_1994_2020.csv", incluíram:

- A criação de um arquivo "b3stocks_T1.csv", no qual o formato da data é DD/MM/AAAA (que antes estava no formato AAAA/MM/DD);
- A criação de um arquivo "b3stocks_F1.csv" para fazer uma filtragem de modo que fique apenas um registro por dia. Esse registro deve ser apenas aquele que possuir o maior volume negociado em bolsa.
- A realização de uma filtragem no arquivo "b3stocks_T1.csv", para que fique apenas os registros que possuírem volume negociado acima da média diária.

O código também realiza algumas ordenações no arquivo "b3stocks_T1.csv", incluindo a ordenação dos tickers por ordem alfabética, a ordenação do volume por ordem crescente e a ordenação das variações em ordem decrescente.

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

Ademais, é importante ressaltar as especificações da máquina utilizada:

Tabela 01: Características principais do computador utilizado.

Placa Mãe	BRX H110
Processador	Intel Core i7 8700
Memória RAM	16GB DDR4 3200MHz
Armazenamento	500GB de SSD + 500GB de HD
Placa de vídeo	rx 580 2048sp
Sistema Operacional	Windows 10

Fonte: Autoria Própria.

ESTRUTURA DO CÓDIGO

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

- **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.

ANÁLISE COMPARATIVA

Foram criadas tabelas comparativas do tempo de execução em milissegundos (ms) de cada algoritmo de ordenação, com base nos resultados de três ordenações distintas: 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.

Tabela 02: Ordenação de tickers em ordem alfabética.

	Caso Médio	Melhor Caso	Pior Caso
Quick Sort	53617 ms	250384 ms	236461 ms
Merge Sort	1111 ms	907 ms	801 ms
Heap Sort	2375 ms	1112 ms	1340 ms

Fonte: Autoria Própria.

Tabela 03: Ordenação dos volumes em ordem crescente.

	Caso Médio	Melhor Caso	Pior Caso
Quick Sort	945 ms	817911 ms	455850 ms
Merge Sort	876 ms	599 ms	558 ms
Heap Sort	1695 ms	760 ms	869 ms

Fonte: Autoria Própria.

Tabela 04: Ordenação de variações diárias em ordem decrescente.

	Caso Médio	Melhor Caso	Pior Caso
Quick Sort	3585 ms	27294 ms	29925 ms
Merge Sort	772 ms	643 ms	614 ms
Heap Sort	1559 ms	963 ms	1062 ms

Fonte: Autoria Própria.

Através da análise das três tabelas é possível verificar que os tempos do Merge Sort foram os melhores em ambos os casos, seguido do Heap Sort. Já os tempos do Quick Sort tiveram melhoras significativas em seus tempos em comparação com a etapa 1 do projeto, resultado das seguintes mudanças:

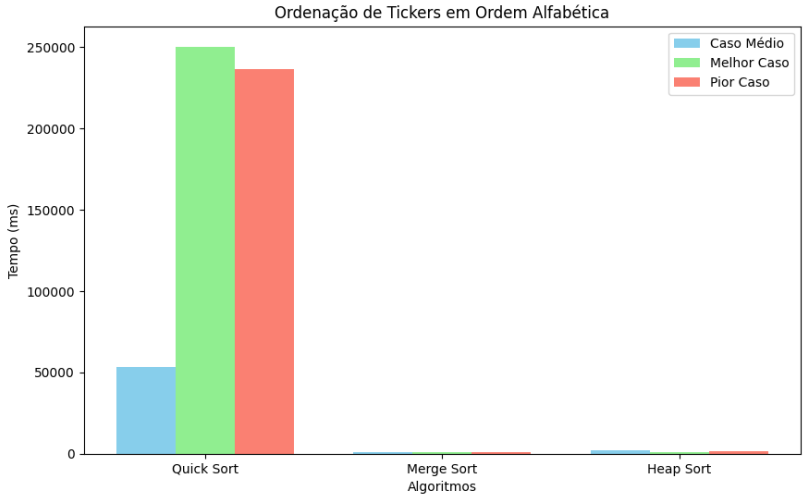
- **Try-catch para StackOverflowError:** Adicionado em cada método de ordenação recursivo.
- **Validação de índices:** Antes de chamar a recursão, verifica se os índices estão válidos.
- **Logger em caso de erro:** Usa System.err para exibir detalhes do erro e os intervalos problemáticos.

Nesse sentido, o try-catch foi inserido para capturar e tratar o StackOverflowError, que pode ocorrer caso a recursão no QuickSort exceda o limite de chamadas na pilha de execução. Com isso, a função do try executa o método de ordenação recursiva e a função do catch captura o erro caso ele ocorra e exibe uma mensagem de alerta no terminal. Dessa forma, evita-se que o programa seja encerrado abruptamente.

RESULTADOS E DISCUSSÕES

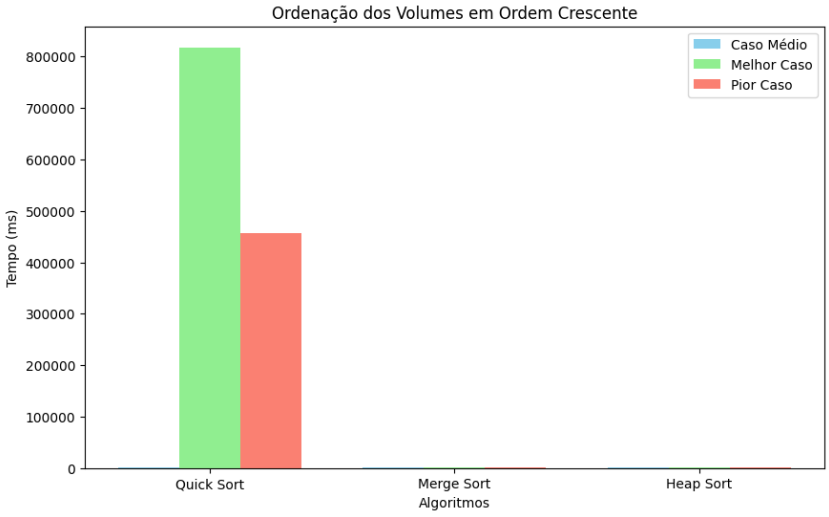
Em geral, os resultados demonstram que, em ambas as etapas do projeto, o Merge Sort e Heap Sort apresentaram os melhores tempos de execução. Nesse sentido, além da geração de tabelas, também foram plotados gráficos que visam comparar o tempo de execução dos algoritmos em cada uma das ordenações.

Figura 01: Gráfico em relação à ordenação dos tickers em ordem alfabética.



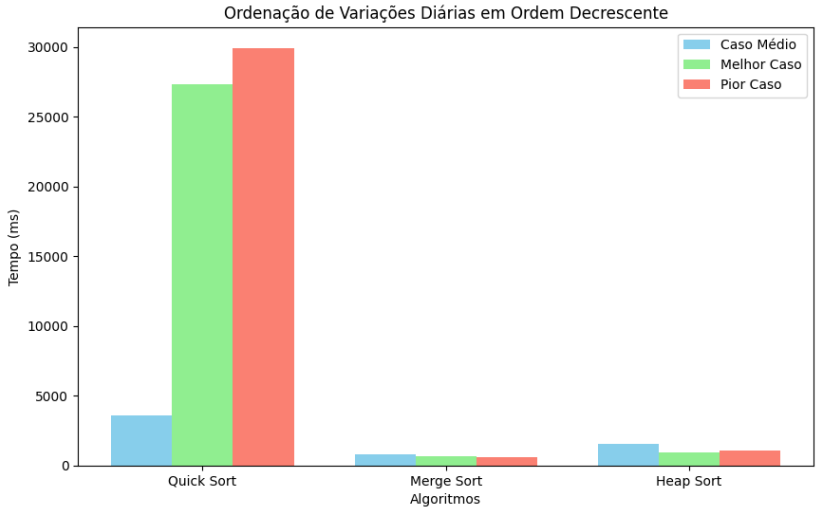
Fonte: Autoria Própria.

Figura 02: Gráfico em relação à ordenação dos volumes em ordem crescente.



Fonte: Autoria Própria.

Figura 03: Gráfico em relação à ordenação de variações diárias em ordem decrescente.



Fonte: Autoria Própria.

Ao examinar os gráficos fornecidos, fica claro que o Merge Sort e o Heap Sort se destacaram como os algoritmos mais eficazes em geral, pois possuem complexidade O(n log(n)) em todos os três casos, enquanto os outros algoritmos têm complexidade $O(n^2)$. 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.

Dessa forma, para obter melhor desempenho na ordenação de grandes conjuntos de dados, recomenda-se a utilização de algoritmos com complexidade O(n log(n)), que conseguem aliar rapidez e estabilidade, oferecendo uma solução mais robusta e eficiente para tarefas de ordenação complexas.

CONCLUSÃO

O presente relatório permitiu uma análise aprofundada da aplicação de algoritmos de ordenação sobre um grande volume de dados históricos da BOVESPA, evidenciando a importância da escolha adequada do algoritmo conforme as características dos dados.

Os experimentos realizados demonstraram que algoritmos com complexidade linear ou quase-linear, como o Merge Sort e o Heap Sort, apresentaram melhor desempenho na maioria dos casos, mesmo com melhorias no tempo do Quick Sort. Além disso, a transformação e filtragem dos dados originais, resultando em novas representações e estruturas de arquivo, mostraram-se essenciais para garantir uma análise precisa e otimizada, possibilitando a avaliação eficiente dos algoritmos. A implementação em Java, utilizando as versões mais recentes das ferramentas de desenvolvimento, assegurou um ambiente robusto e eficaz para a execução das tarefas propostas.

Por fim, esse estudo destaca a relevância de um entendimento profundo das características dos dados a serem processado, contribuindo para a compreensão dos comportamentos dos algoritmos em contextos reais de mercado.

REFERÊNCIAS

BRAGA, Henrique. *Algoritmos de Ordenação: Ordenação por Inserção. Disponível em: <https://henriquebraga92.medium.com/algoritmos-de-ordena%C3%A7%C3%A3o-iii-insertion-sort-bfade66c6bf1>* Acesso em: 20 de novembro de 2024.

DEVMEDIA. *Algoritmos de ordenação: análise e comparação. Disponível em: <https://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261>* Acesso em: 20 de novembro de 2024.

VIANA, Daniel. Conheça os principais algoritmos de ordenação. Disponível em: <https://www.treinaweb.com.br/blog/conheca-os-principais-algoritmos-de-ordenacao> Acesso em: 20 de novembro de 2024.