Universidade Estadual da Paraíba

Curso: Ciência da Computação

Componente Curricular: Laboratório de Estrutura de Dados

Alunos:

- **Arthur Salviano Ferreira**
- **Lucas Manoel Freire Monteiro Cabral**

Relatório do Projeto da Disciplina: Comparação de Algoritmos através da Bolsa de Valores BOVESPA 1994-2020

Sumário

1. Introdução

Este relatório corresponde ao projeto realizado na disciplina Laboratório de Estrutura de Dados. Esse projeto teve o intuito de ampliar o nosso conhecimento sobre os algoritmos de ordenação, bem como suas implementações em problemas do mundo real.

Para isso, foi escrito um código em Java que organiza e altera informações contidas em um arquivo .csv. Essas informações são referentes às ações negociadas na BOVESPA desde 1994 até 2020.

As transformações que o código executa no arquivo b3_stocks_1994_2020.csv são as seguintes:

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

O código também executa as seguintes ordenações no arquivo b3stocks_T1.csv:

- Ordena os tickers por ordem alfabética;
- Ordena o volume por ordem crescente;
- Ordena as variações em ordem decrescente.

Para testar a eficiência dos diversos algoritmos de ordenação, as tarefas acima foram executadas usando os seguintes algoritmos: Insertion Sort, Selection Sort, Quick Sort, Quick Sort com mediana de 3, Merge Sort, Heap Sort e Counting Sort. Cada um desses algoritmos foi executado três vezes, de modo a ordenar o melhor caso, o pior caso e o caso médio. Para cada uma dessas execuções, o código gera um novo arquivo .csv, que mostra as informações ordenadas.

Os tempos de execução de cada algoritmo foram capturados a fim de compararmos a eficiência de cada um (essa análise será feita na seção 3 deste documento).

2. Descrição geral sobre o 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;
- InsertionSort: ordena o arquivo utilizando insertion sort;
- SelectionSort: ordena o arquivo utilizando selection sort;
- QuickSort: ordena o arquivo utilizando quick sort;
- QuickSortMedianaDe3: ordena o arquivo utilizando quick sort com mediana de 3;
- MergeSort: ordena o arquivo utilizando merge sort;
- HeapSort: ordena o arquivo utilizando heap sort;
- CountingSort: ordena o arquivo utilizando counting sort;
- Main: executa o código.

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 10 64 bits.

3. Resultados e Análise

A seguir constam as tabelas que comparam o tempo de execução em milissegundos (ms) dos algoritmos de ordenação. Foram feitas três ordenações: ordenar os registros de acordo com seus tickets por ordem alfabética, ordenar os registros de acordo com os volumes em ordem crescente e ordenar os registros de acordo com suas variações diárias em ordem decrescente.

Os algoritmos foram utilizados nessas três ordenações, e cada um deles foi executado três vezes, considerando o melhor caso, o pior caso e o caso médio. Nota-se, porém, que o algoritmo counting sort não possui tempo de execução nas tabelas, pois ele não consegue ordenar letras (invalidando-o na primeira ordenação) nem valores flutuantes (invalidando-o na segunda e na terceira ordenações).

Tabela 1: Ordenação dos tickers em ordem alfabética

	Caso Médio	Melhor Caso	Pior Caso
Insertion Sort	**455 ms**	**202 ms**	**841 ms**
Selection Sort	**641 ms**	**626 ms**	**800 ms**
Quick Sort	**15 ms**	**59 ms**	**108 ms**
Quick Sort com mediana de 3	**17 ms**	**35 ms**	**49 ms**
Merge Sort	**6 ms**	**5 ms**	**3 ms**
Heap Sort	**7 ms**	**4 ms**	**6 ms**
Counting Sort	**-**	**-**	**-**

Tabela 2: Ordenação dos volumes em ordem crescente

	Caso Médio	Melhor Caso	Pior Caso
Insertion Sort	**201 ms**	**1 ms**	**404 ms**
Selection Sort	**240 ms**	**262 ms**	**328 ms**
Quick Sort	**6 ms**	**412 ms**	**419 ms**
Quick Sort com mediana de 3	**9 ms**	**169 ms**	**119 ms**
Merge Sort	**9 ms**	**8 ms**	**4 ms**
Heap Sort	**11 ms**	**3 ms**	**3 ms**
Counting Sort	**-**	**-**	**-**

Tabela 3: Ordenação das variações diárias em ordem decrescente

	Caso Médio	Melhor Caso	Pior Caso
Insertion Sort	**383 ms**	**0 ms**	**495 ms**
Selection Sort	**375 ms**	**311 ms**	**318 ms**
Quick Sort	**79 ms**	**74 ms**	**59 ms**
Quick Sort com mediana de 3	**29 ms**	**75 ms**	**68 ms**
Merge Sort	**11 ms**	**10 ms**	**5 ms**
Heap Sort	**11 ms**	**4 ms**	**4 ms**
Counting Sort	**-**	**-**	**-**

Abaixo constam os gráficos que comparam o tempo de execução dos algoritmos em cada uma das ordenações:

Gráfico 1: Ordenação dos tickers

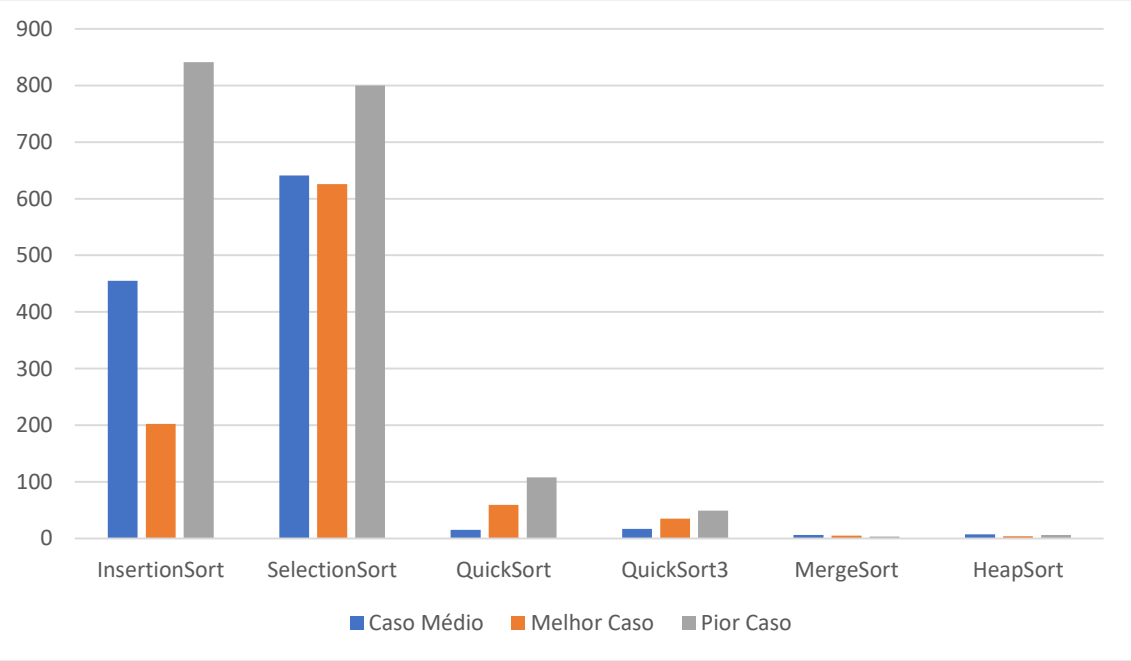


Gráfico 2: Ordenação dos volumes

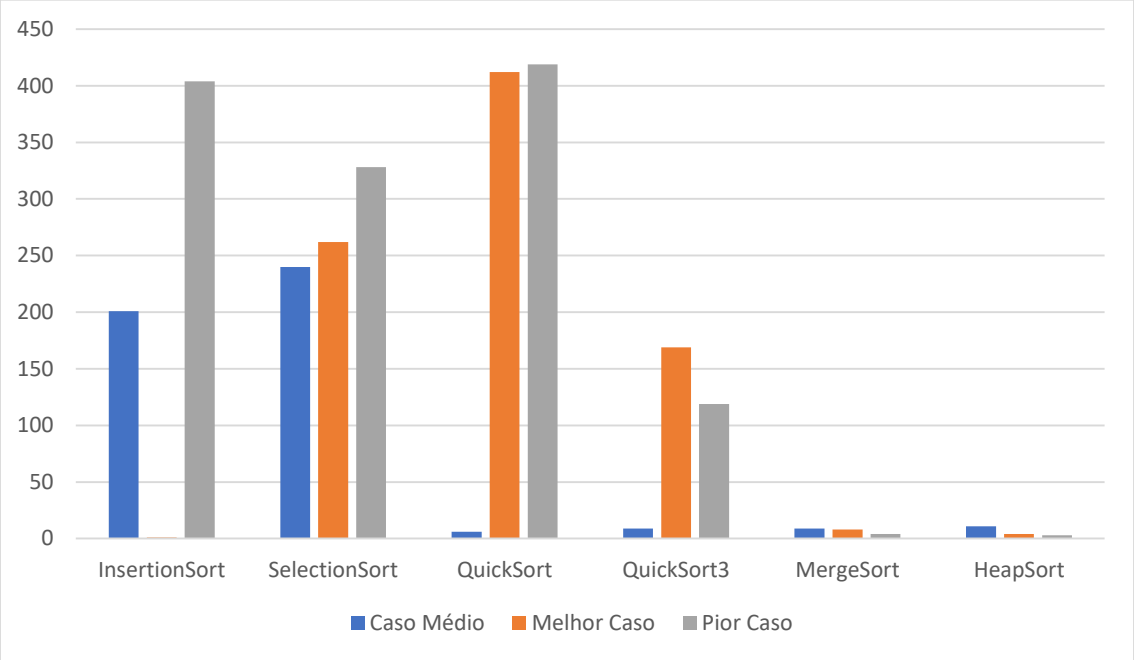
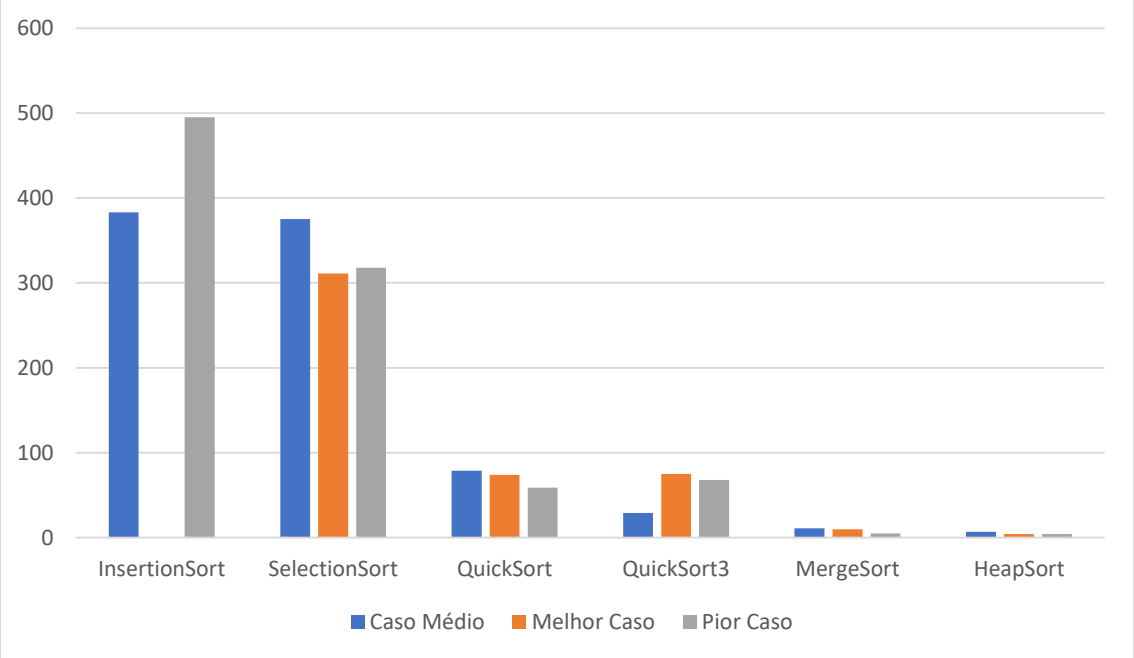


Gráfico 3: Ordenação das variações



Ao observar os gráficos acima, é notório que, no geral, o Heap Sort e o Merge Sort foram os algoritmos mais eficientes, pois são nos três casos eles possuem eficiência O(n*log(n)) (em geral, os outros algoritmos possuem eficiência $O(n^2)$). Embora o Quick Sort também tenha eficiência O(n*log(n)) no melhor caso e no caso médio, ele demonstrou ser menos eficiente que o Merge

Sort e o Heap Sort porque o array de entrada é muito grande (quase 2 milhões de registros), o que diminui a eficiência do Quick Sort.

Contudo, o Insertion Sort teve tempos de execução muito baixos no melhor caso. Isso porque a eficiência do Insertion Sort no melhor caso é O(n). Nos outros dois casos, o Insertion Sort mostrou ser o algoritmo menos eficiente, em conjunto com o Selection Sort.

Desse modo, foi possível observar que os algoritmos de ordenação realmente se comportam de acordo com a tabela de complexidade de tempo, como mostrada no livro “Algoritmos - Teoria e Prática”, de Thomas H. Cormen e Charles E. Leiserson, 2012.