Laboratório de Estrutura de Dados

Versão final do projeto da disciplina

Comparação entre os algoritmos de ordenação elementar

Aluno:

Gustavo Silva Medeiros

1. Introdução

Após os estudos de análise de algoritmos de ordenação vistos em sala de aula na turma de Laboratório de Estrutura de Dados, foi apresentado aos alunos a criação de um projeto de disciplina que visa implementar sete algoritmos (CountingSort, MergeSort, QuickSort Mediana de Três, HeapSort, QuickSort, InsertionSort, SelectionSort). O objetivo do projeto é possibilitar aos alunos a visualização prática sobre o funcionamento de cada um desses algoritmos e analisar os seus desempenhos nas situações de melhores, médios e piores casos.

Para este estudo será utilizado o dataset sobre Covid19, disponível no site brasil.io, e serão aplicados os algoritmos de ordenação em dados do tipo Strings e Inteiros. Os resultados obtidos serão descritos neste relatório.

O código fonte do projeto está disponível no GitHub juntamente com as instruções sobre sua utilização. É importante ressaltar que o arquivo CSV utilizado para as execuções relatadas neste documento já se encontra na pasta “db/covid19.csv”, porém também é possível substituir este arquivo pela versão mais recente encontrada neste link, onde os filtros já estão aplicados, bastando clicar no botão “BAIXAR RESULTADO EM CSV”.

Na pasta “src” do repositório temos os códigos na linguagem Java, onde se encontram os pacotes:

- “app”, com o Main que deve ser executado;
- “data“, com as classes CSV e DadoCovid;
- “sort”, com os algoritmos de ordenação;
- “tests”, com os testes unitários que direcionaram o desenvolvimento do projeto;
- “util", com as classes “Array” e “Resultado”.

Ao final deste relatório, será possível verificar que alguns algoritmos se saíram melhores do que outros, dependendo da situação específica, mas no geral é possível concluir que algoritmos como o quickSort, tanto o normal como o mediana de três, o mergeSort e o HeapSort, obtiveram melhores resultados comparados ao selectionSort e insertionSort.

2. Descrição geral sobre o método utilizado

Devido a restrição definida pelo professor de não utilização das classes de estrutura de dados fornecidas pelos pacotes do Java, o primeiro passo foi criar uma classe chamada Array com todos os métodos que necessários para a manipulação dessas estruturas. Esta classe foi implementada utilizando o conceito de generics, para que fosse possível utilizá-la para qualquer tipo de variável. Durante a implementação dessa classe, foram criados testes unitários para seus métodos para garantir o seu perfeito funcionamento.

Em seguida, também de forma genérica e utilizando o conceito de desenvolvimento dirigido por testes, foram implementados todos os algoritmos de ordenação vistos na disciplina. Os algoritmos foram traduzidos para a linguagem Java a partir do livro "Introduction to Algorithms", 3ª edição, dos autores Cormen, Leiserson, Rivest e Stein.

O próximo passo foi baixar o CSV gerado pelo dataset da Covid 19, fornecido pelo site brasil.io e implementar as classes CSV e DadoCovid, onde a primeira lê as linhas do arquivo CSV e a segunda converte cada linha em um objeto do tipo DadoCovid. Então, foi gerado um Array<DadoCovid> com os objetos que representam todas as linhas do arquivo CSV.

Como os algoritmos de ordenação que foram implementados ordenavam apenas dados primitivos, como strings e inteiros, foi necessário adaptá-los para ordenar dados do tipo DadoCovid. Para garantir que esta adaptação funcionasse como o esperado, também foi necessário adaptar os testes iniciais para testarem ordenações de arrays do tipo DadoCovid.

Inicialmente, como seria necessário que analisar o desempenho dos algoritmos para os melhores, médios e piores casos, foi preciso tratar os dados do arquivo CSV original, uma vez que ele já vem pré-ordenado. Então, o nosso Main gera os seguintes arquivos na pasta "db/cases" a partir do arquivo "db/covid19.csv":

- covid19-cidade-asc.csv
- covid19-cidade-desc.csv
- covid19-confirmados-acumulados-asc.csv
- covid19-confirmados-acumulados-desc.csv
- covid19-obitos-acumulados-asc.csv

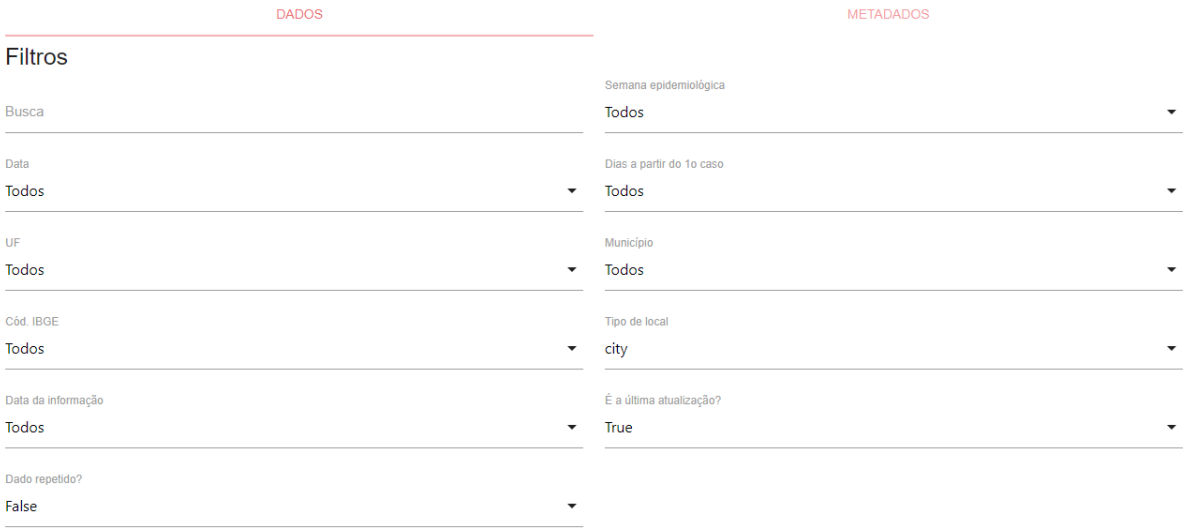
- covid19-obitos-acumulados-desc.csv
- covid19-shuffled.csv

O arquivo “covid19-shuffled.csv” contém os dados originais, porém embaralhados. Os arquivos com terminação “asc”, contêm os dados originais, porém ordenados de acordo com o parâmetro descrito em seu título de forma crescente. Os arquivos com terminação “desc”, contêm os dados originais, porém ordenados de acordo com o parâmetro descrito em seu título de forma decrescente.

Explicando de forma mais geral, o método utilizado foi simples. Uma tabela do tipo .CSV é passada para uma classe onde ela é lida e interpretada, e cada uma das suas linhas é passada para o construtor da classe DadoCovid, que transforma os dados da linha em atributos de classe que serão manipulados mais a frente. Em seguida, é gerado um vetor do tipo Array (classe de vetor que manipula os dados), onde cada elemento é um objeto instanciado da classe DadoCovid que contém os dados necessários para esta análise.

Com os dados tratados de forma que é possível utilizar no código, basta passar esse Array de DadoCovid e o parâmetro que será ordenado, seja ele: quantidade de óbitos acumulados, quantidades de casos registrados acumulados ou nome de cidades. Com isso cada um dos códigos de ordenação irá ordenar os dados e devolver os mesmos de forma ordenada, que por fim serão passados para a classe CSV onde o método “gravar” irá fazer o processo inverso e transformará os dados em um arquivo CSV. Um arquivo será gravado para cada parâmetro ordenado utilizando cada um dos algoritmos de ordenação. No total, serão gerados 20 arquivos na pasta “db/sorted”.

Devemos considerar para esta análise que foi simulado um “pior”, "médio" e “melhor” caso a partir dos arquivos gerados na pasta “db/cases”, onde o pior caso se trata dos dados estarem ordenados de forma decrescente, o melhor seria onde os dados já estão ordenados e por fim o caso médio são onde os dados se encontram de forma aleatória que é feito com um método chamado “shuflle”, que embaralha de forma aleatória os dados. Além disso, foram utilizados alguns filtros para baixar a planilha, que serão exibidos na imagem abaixo:



Ao fim do processamento de todos os dados, o programa gera um conjunto de tabelas para demonstrar o tempo de execução dos códigos de forma comparativa e suas comparações no melhor, pior e médio caso para Strings e para os numéricos.

Descrição geral do ambiente de testes

As configurações da máquina usada para criar o projeto e para submetê-lo aos teste foram:

- Processador Intel Core i5-9600K Coffee Lake Refresh, Cache 9MB, 4.6GHz Max Turbo
- Placa-Mãe Gigabyte Z390 M Gaming, Intel LGA 1151, mATX, DDR4
- Memória ram Crucial Ballistix Sport LT, 32GB, DDR4, 3000MHz;
- Placa de vídeo GALAX GeForce® GTX 1660 Ti (1-Click OC);
- 3TB (3x1TB) de SSD M.2 Adata XPG NVMe;
- Sistema operacional: Windows 10.

E para desenvolvimento o ambiente que usamos foi:

- Eclipse IDE for Java Developers (includes Incubating components);
- Version: 2020-12 (4.18.0);
- Build id: 20201210-1552;
- Projeto criado com usando o ambiente de execução JRE: JavaSE-1.8.

3. Resultados e Análise

A seguir serão demonstrados graficamente os resultados obtidos após a execução dos algoritmos.

Os gráficos contêm no eixo horizontal os algoritmos analisados e no eixo vertical os valores em milisegundos obtidos nos testes.

Observação: o countingSort não foi implementado para ordenação de Strings.

Gráfico 1

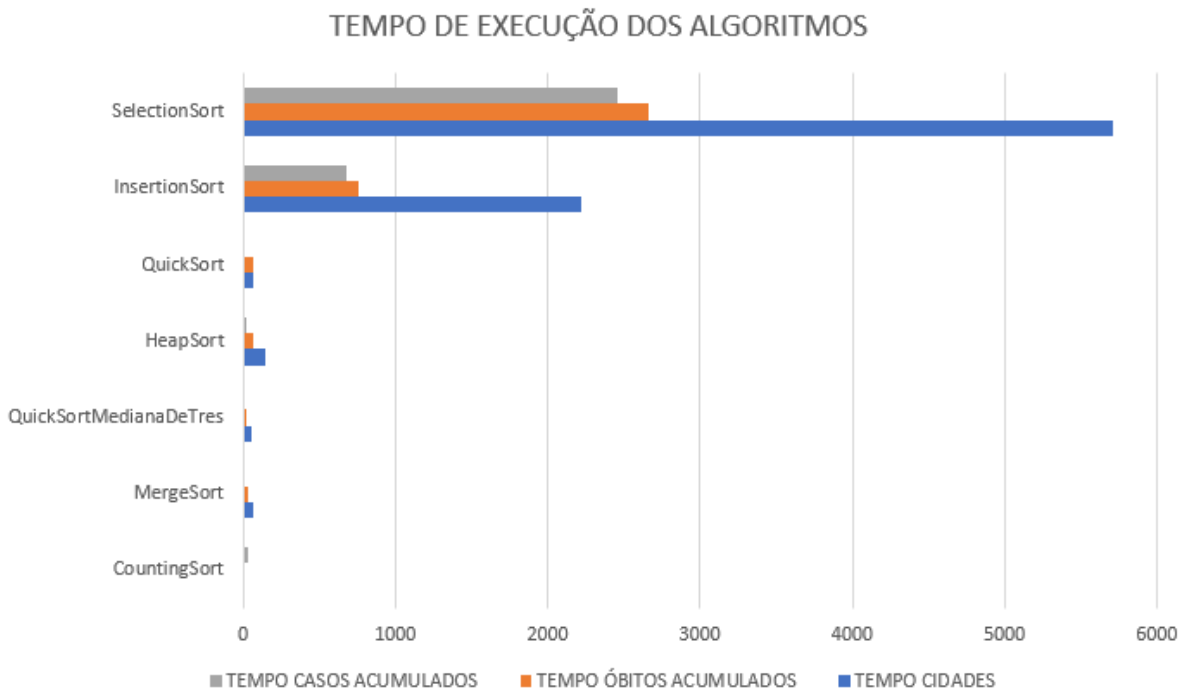


Gráfico 2

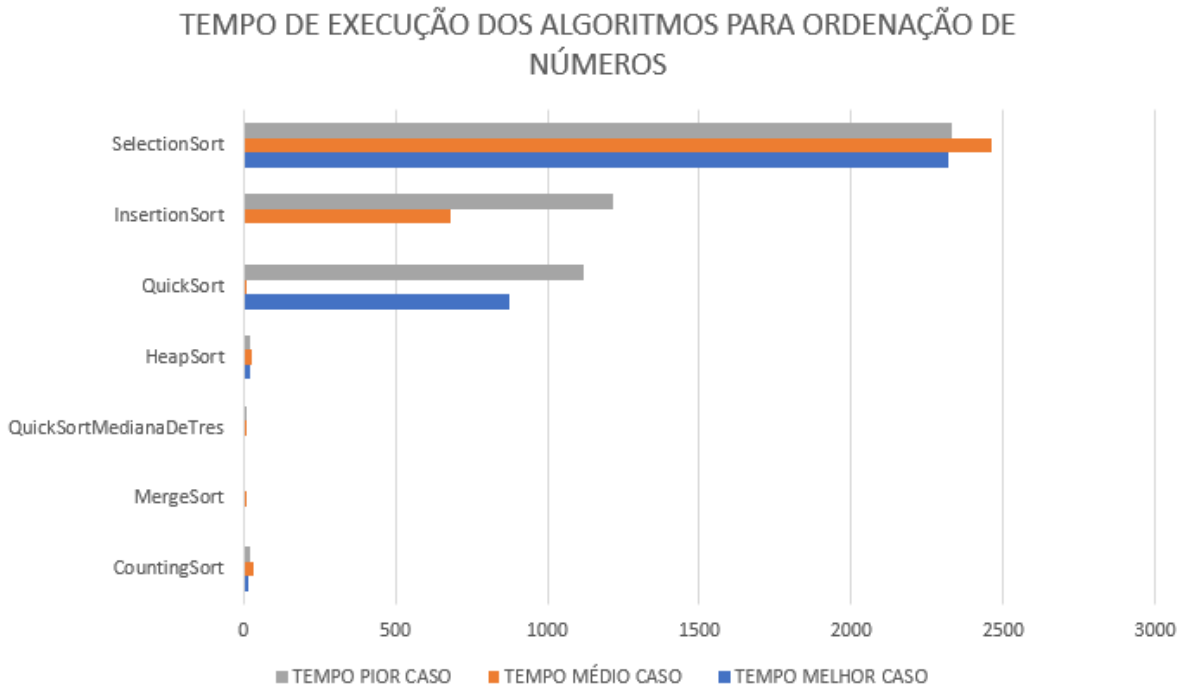
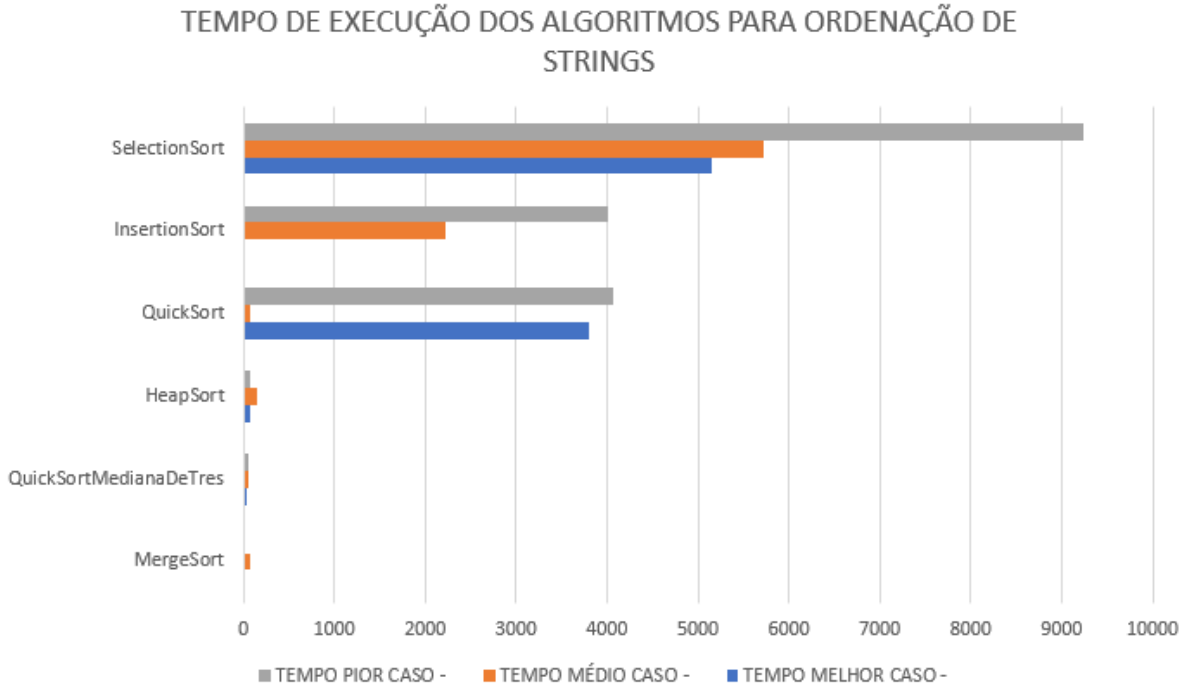


Gráfico 3



Os gráficos acima foram criados utilizando o Microsoft Excel, onde foram inseridos os dados das tabelas que o código produz. Após várias execuções, foi

percebido que os valores das tabelas estavam com os dados muito próximos aos dados das execuções anteriores. Então, os estudos comparativos foram realizados a partir da tabela abaixo:

TEMPO DE EXECUÇÃO DOS ALGORITMOS

ALGORITMOS	TEMPO CIDADES	TEMPO ÓBITOS ACUMULADOS	TEMPO CASOS ACUMULADOS
CountingSort	-	12ms	35ms
MergeSort	69ms	36ms	10ms
QuickSortMedianaDeTres	55ms	26ms	8ms
HeapSort	145ms	67ms	25ms
QuickSort	64ms	65ms	9ms
InsertionSort	2216ms	757ms	681ms
SelectionSort	5711ms	2658ms	2459ms

TEMPO DE EXECUÇÃO DOS ALGORITMOS PARA ORDENAÇÃO DE NÚMEROS

ALGORITMOS	TEMPO MELHOR CASO	TEMPO MÉDIO CASO	TEMPO PIOR CASO
CountingSort	18ms	35ms	20ms
MergeSort	6ms	10ms	6ms
QuickSortMedianaDeTres	6ms	8ms	8ms
HeapSort	23ms	25ms	22ms
QuickSort	876ms	9ms	1120ms
InsertionSort	1ms	681ms	1218ms
SelectionSort	2319ms	2459ms	2330ms

TEMPO DE EXECUÇÃO DOS ALGORITMOS PARA ORDENAÇÃO DE STRINGS

ALGORITMOS	TEMPO MELHOR CASO	TEMPO MÉDIO CASO	TEMPO PIOR CASO
CountingSort	-	-	-
MergeSort	23ms	69ms	23ms
QuickSortMedianaDeTres	44ms	55ms	59ms
HeapSort	78ms	145ms	79ms
QuickSort	3809ms	64ms	4062ms
InsertionSort	10ms	2216ms	4001ms
SelectionSort	5155ms	5711ms	9228ms

Após a análise dos gráficos fica notável que os algoritmos que mais se destacaram foram o mergesort, o quicksort utilizando mediana de três e o heapsort. O quicksort se saiu bem, mas em algumas situações foi notado um tempo muito longo de execução como por exemplo no gráfico de pior caso de ordenação de Strings. O countingsort apenas foi implementado para inteiros pela sua estrutura de ordenação, mas se comportou de maneira bem interessante e satisfatória nos testes. Por fim, como os piores algoritmos temos o insertionsort e o selectionsort que comparado aos outros possuem uma diferença gritante no seu tempo de execução, em casos a discrepância é tão grande que faz as barras de dados dos outros algoritmos mal aparecem no gráfico. Com isso é possível concluir a efetividade de alguns algoritmos perante outros em determinadas situações, para a ordenação massiva de dados como foi aplicada nessa situação, onde os dados a serem ordenados chegavam nas casas de 5 dígitos os algoritmos de divisão e conquista e de árvore mostraram-se incrivelmente superiores. Esse cenário onde alguns algoritmos se sobressaem perante outros pode ser alterado a

determinar do contexto em que estão mas ao final da execução desse algoritmo e deste projeto podemos ver que para arquivos grandes quais os algoritmos que se sobressaem.