Laboratório de Estrutura de Dados

Primeira versão do projeto da disciplina

Comparação entre os algoritmos de ordenação elementar

Arthur Pereira da Silva; Klebson Amarante da Silva; Antônio Wenicios Ademar da Silva

1. Introdução

Este relatório corresponde ao relato dos resultados obtidos no projeto da disciplina de LEDA, em que foi visado estudar a implementação e desempenho dos algoritmos de ordenação em dados retirados do site *Kaggle*. O tema escolhido para o projeto, aqui relatado, foi o Lost Angeles Metro Bike Share, um sistema de compartilhamento de bicicletas na cidade de Los Angeles, Califórnia e área metropolitana. Neste tema, é apresentado, inicialmente, um dataset que contém informações gerais - id da viagem, duração, tempo inicial, tempo final, id da bike, tipo da bike, etc -, bem como um segundo dataset que contém as estações e suas informações - nome da estação, dia de ativação da estação, locação e status de atividade.

O projeto exige, na primeira parte, realizar transformações. De início, pede-se para substituir o id das estações pelo nome (campo station_name), que está contido no segundo dataset (station_csv), gerando um arquivo transformado (LAMetroTrips.csv). Com base no arquivo gerado da primeira transformação, pede-se para filtrar apenas as viagens que estão nas estações de Pasadena, gerando um segundo arquivo transformado (LAMetroTrips_F1.csv). Por fim, ainda com base na primeira transformação, exige-se filtrar apenas as viagens que possuem duração maior que a média geral, também gerando, ao fim, um arquivo transformado.

Na segunda parte, considerando o arquivo de dados da transformação 1, é proposto realizar ordenações: ordenar o arquivo pelo nome das estações, em ordem alfabética; ordenar pelo campo de duração da viagem, do menor para o maior; e ordenar pela data de início da viagem, da mais recente para a mais antiga.

Para cada algoritmo implementado - Insertion Sort, Selection Sort, Merge Sort, Quick Sort, Quick sort com mediana de 3, Counting Sort e Heap Sort - são gerados e analisados arquivos com o melhor, médio e pior caso.

Para fins de teste, todos os algoritmos foram testados com a mesma base de dados em condições iguais. As implementações foram feitas em JAVA, e foram utilizadas ferramentas, como o VisualVM e alguns recursos da IDE Intellij.

Neste relatório, utilizando o artifício descritivo, explora-se as características do método utilizado, as condições do ambiente de testes, bem como a exposição dos resultados e análises através de gráficos.

2. Descrição geral sobre o método utilizado

Objetivando-se tratar os dados, foram feitas modificações no dataset considerado para as ordenações, sendo que, no melhor caso, ocorre a modificação dos dados para serem dispostos de forma crescente; no pior caso, os dados são invertidos (ordem decrescente); e no médio caso eles são dispersos, ou seja, dispostos de forma randômica.

Ao executar o programa principal (main), primeiramente, são carregadas todas as transformações, e, logo em seguida, o usuário pode escolher qual tipo de ordenação deseja efetuar. Tanto para as transformações, quanto para as ordenações, são gerados os arquivos resultantes do processo, sendo que nas ordenações, para cada algoritmo, são gerados os arquivos do melhor, médio e pior caso.

Para executar os testes de complexidade, viabilizou-se o uso da ferramenta VisualVM, principalmente para a análise do uso de CPU e memória RAM, e recursos de interrupção da IDE Intellij, com o fito de analisar o tempo de execução das ordenações.

As análises seguem três padrões: gráficos com uso de CPU e memória RAM para cada tipo de ordenação; gráficos com o tempo para realizar as transformações e ordenações; e, por fim, três classes de gráficos, em que são analisados os desempenhos dos algoritmos frente ao tempo em cada ordenação.

Dependendo das configurações de uma máquina, pode-se obter diferentes interpretações para analisar a eficiência de um algoritmo. Por esse motivo, tendo como base a máquina descrita abaixo, as análises realizadas devem ser consideradas para casos de testes semelhantes.

Descrição geral do ambiente de testes:

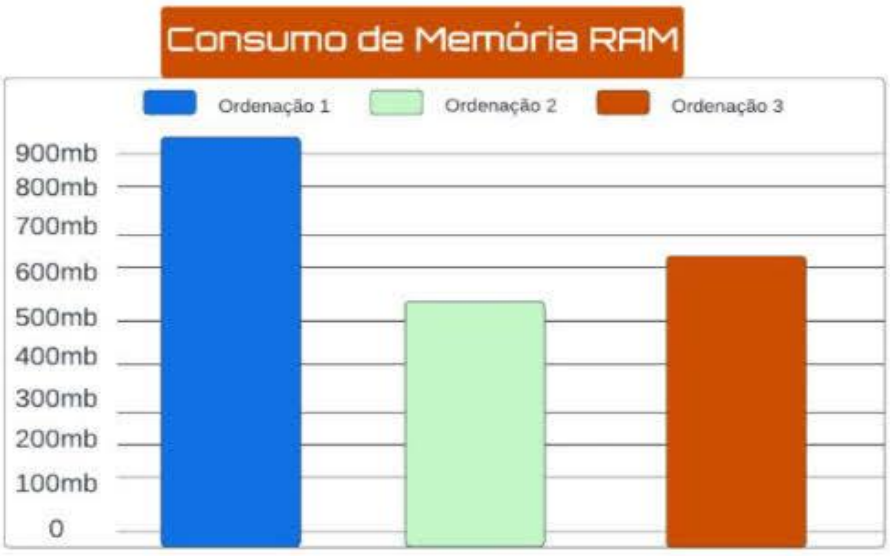
O código foi executado em diferentes máquinas, mas, para fins de análise, foram consideradas apenas uma. Segue, abaixo, as configurações:

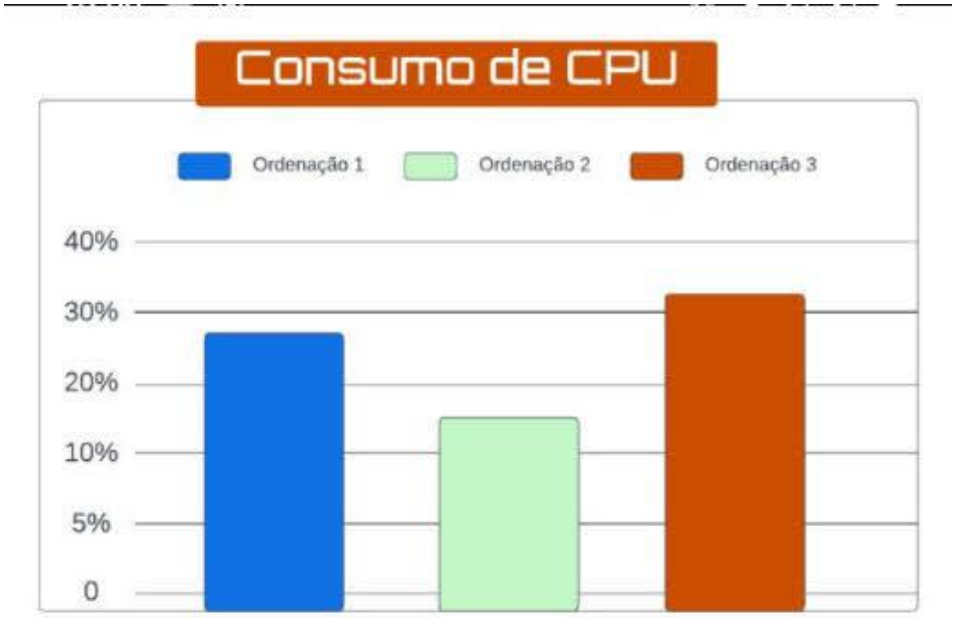
- Processador Intel(R) Core(™) i5-3470 CPU @ 3.20GHz
- Memória RAM instalada 16,0 GB (utilizável: 15,9 GB)
- Sistema Operacional: Windows 11 / 64 bits
- IDE Intellij

3. Resultados e Análise

Os resultados e análises estão dispostos em gráficos para cada padrão já descrito neste relatório. Seguem, abaixo, as análises desenvolvidas.

Análises para Consumo de CPU e memória RAM para cada ordenação, considerando a média dos melhores, médios e piores casos:





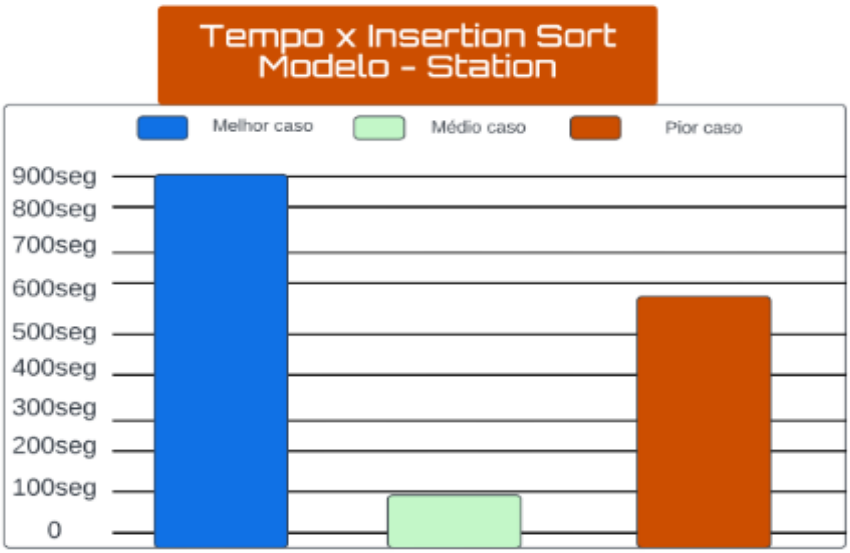
Nessa primeira análise, podemos concluir que a ordenação 1 (ordenar pelos nomes das estações) foi a que consumiu mais da memória RAM, por volta de 900 mb. Enquanto a ordenação 3 (ordenar pelo campo de duração da viagem), foi a que mais consumiu da CPU, por volta de 30%, mas a ordenação 1 também chegou a níveis de consumo próximos.

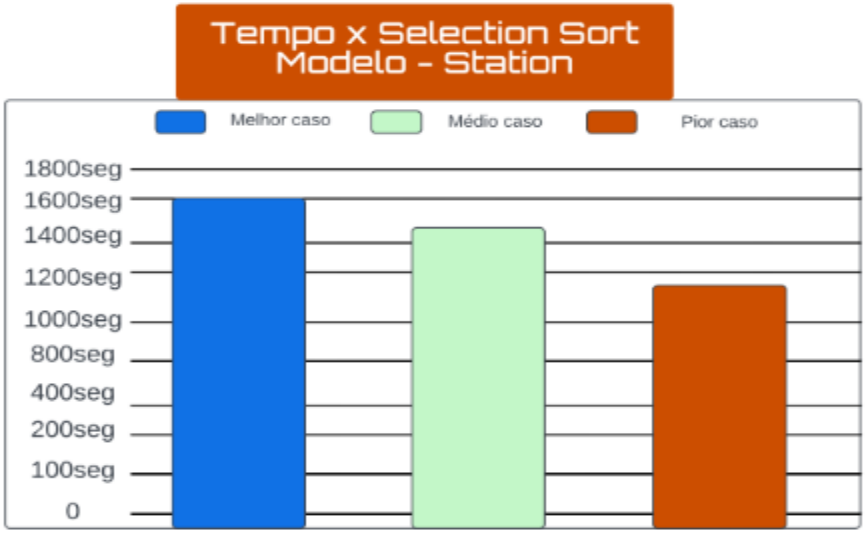
Análises para gasto do tempo das transformações e ordenações (no geral):

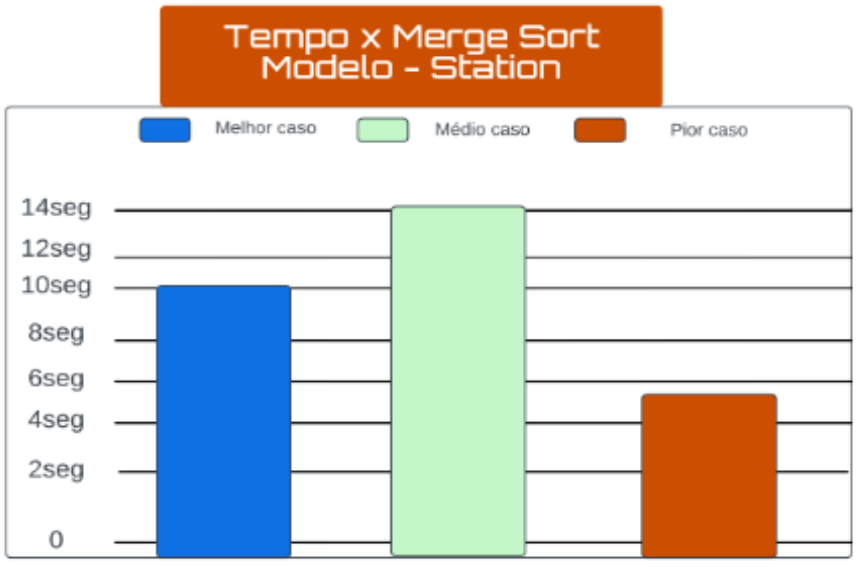
Pode-se perceber que a ordenação 1 foi a que mais demorou para ser concluída, enquanto a ordenação 2 e ordenação 3 (ordenar pela data de início da viagem) foram concluídas em tempos bastante semelhantes.

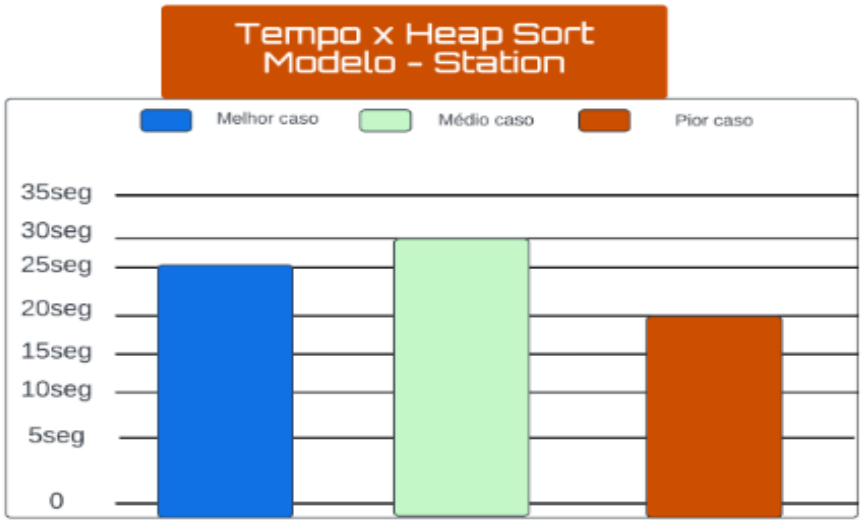
Já com relação às transformações a que mais levou tempo foi a transformação 1(substituir o campo id pelo nome das estações), cerca de 200 segundos. A transformação 3 foi a segunda a gastar mais tempo, cerca de 100 segundos, e a transformação 2 foi a que foi concluída em menor tempo, quase 200 segundos.

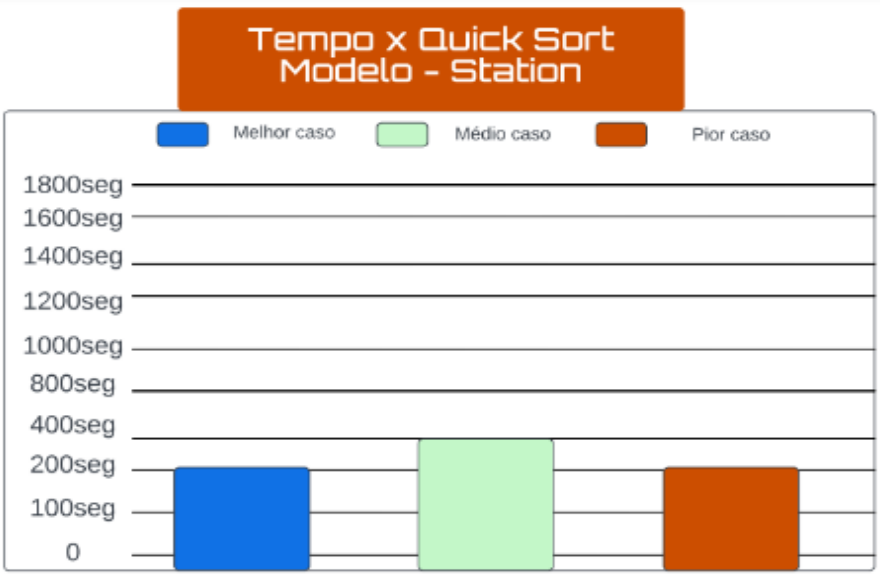
Análise de gasto de tempo para cada algoritmo implementado na ordenação 1 (ordenar pelos nomes das estações) - Insertion Sort, Selection Sort, Merge Sort, Heap Sort e Quick Sort.




Tempo x Selection Sort
Modelo - Station
Melhor caso
Médio caso
Pior caso
1800seg
1600seg
1400seg
1200seg
1000seg
800seg
400seg
200seg
100seg
0

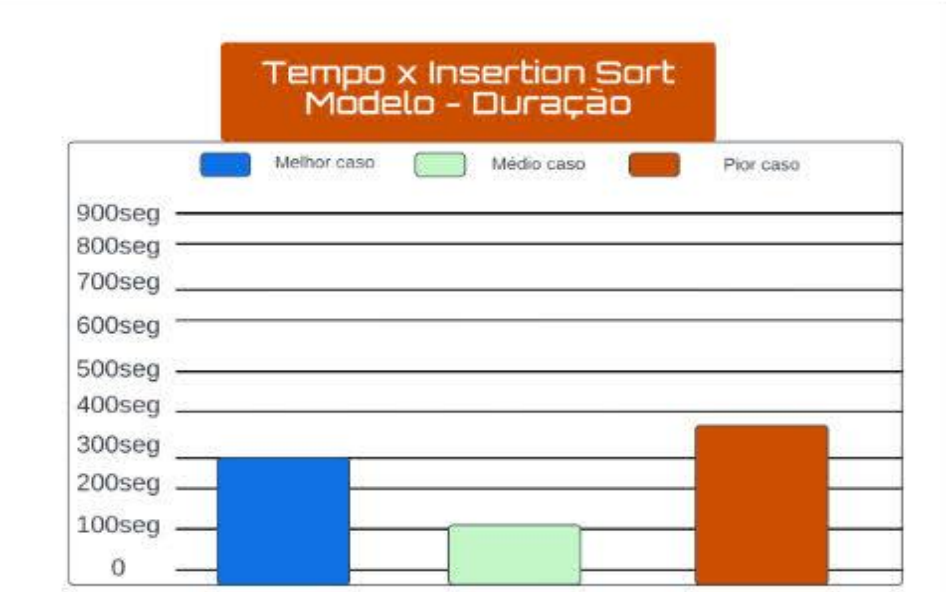

Tempo x Merge Sort
Modelo - Station
Melhor caso
Médio caso
Pior caso
14seg
12seg
10seg
8seg
6seg
4seg
2seg
0

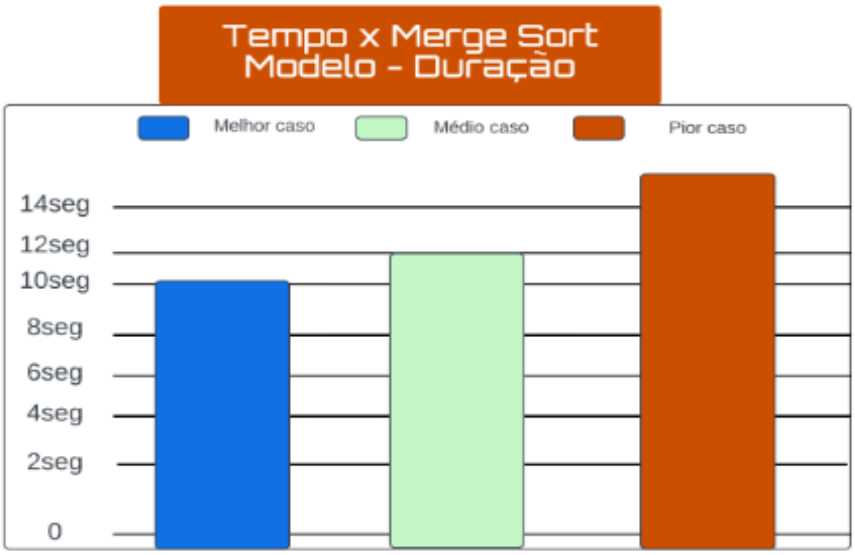


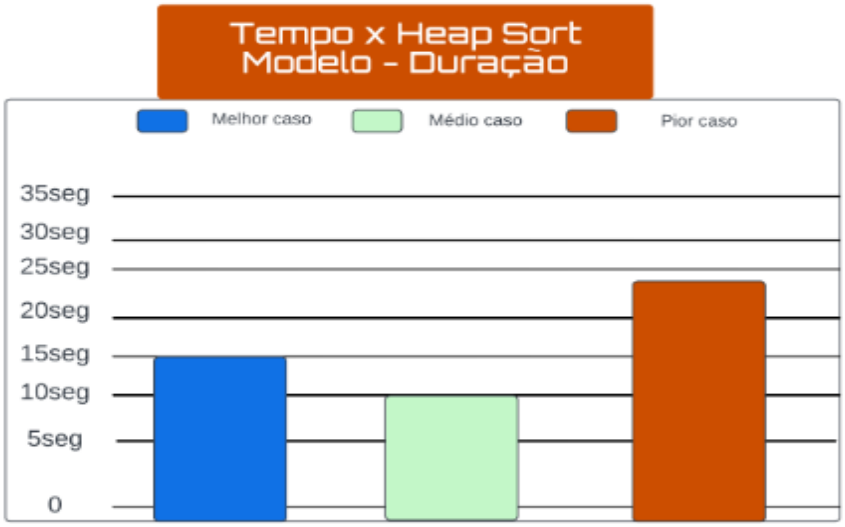


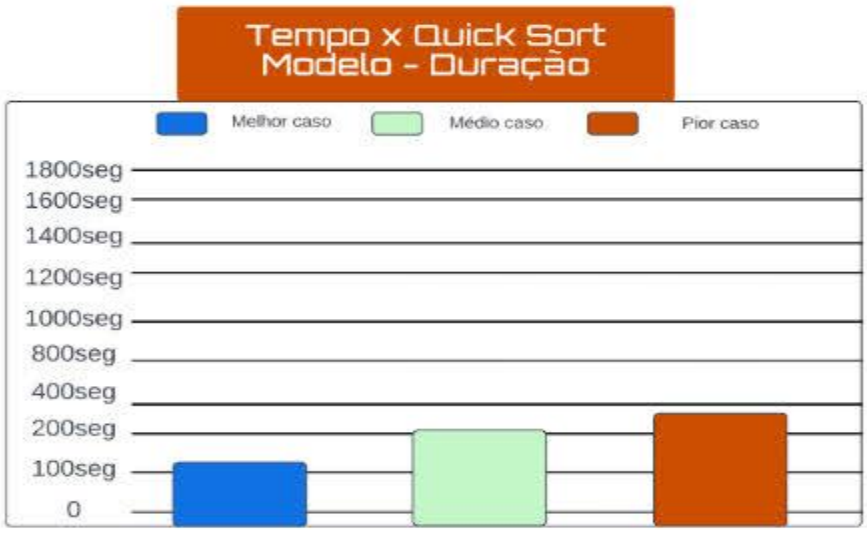
Para a ordenação 1, pode-se perceber que o algoritmo mais eficiente foi o Merge Sort, possuindo o melhor tempo em todos os casos, chegando a no máximo a 14 segundos para ordenar, o que acontece no médio caso. O menos eficiente foi o Selection Sort, chegando a gastar 1600 segundos para ordenar até mesmo no melhor caso.

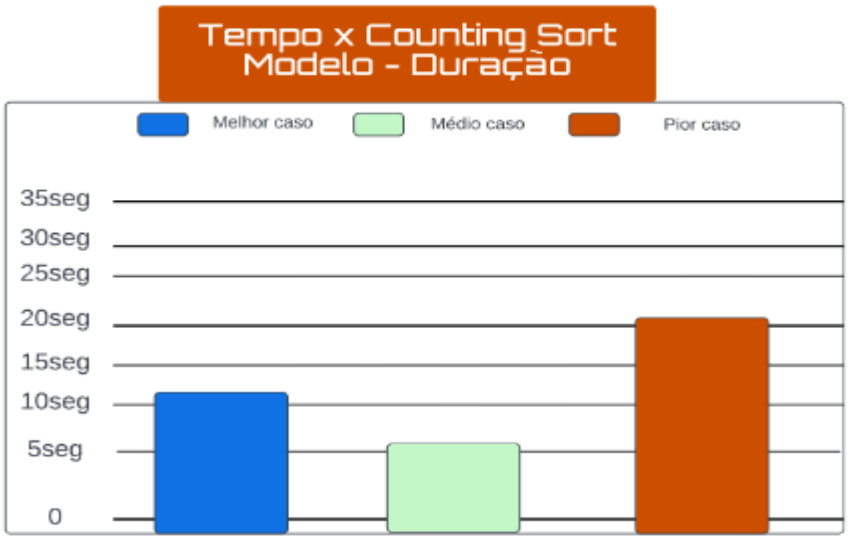
Análise de gasto de tempo para cada algoritmo implementado na ordenação 2 (ordenar pelo campo de duração da viagem) - Insertion Sort, Selection Sort, Merge Sort, Heap Sort e Quick Sort e Counting Sort.




Tempo x Merge Sort
Modelo - Duração
Melhor caso
Médio caso
Pior caso
14seg
12seg
10seg
8seg
6seg
4seg
2seg
0

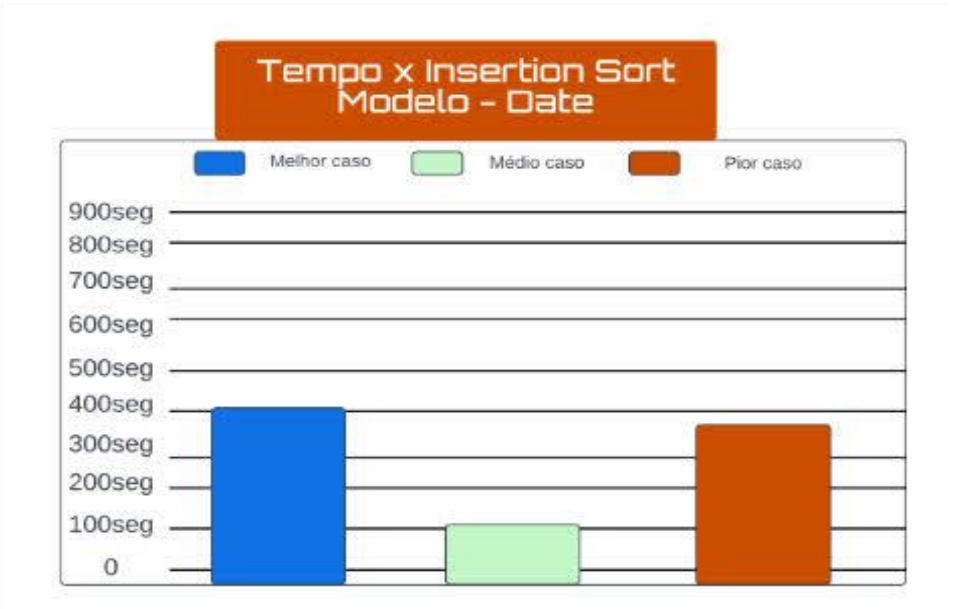

Tempo x Heap Sort
Modelo - Duração
Melhor caso
Médio caso
Pior caso
35seg
30seg
25seg
20seg
15seg
10seg
5seg
0

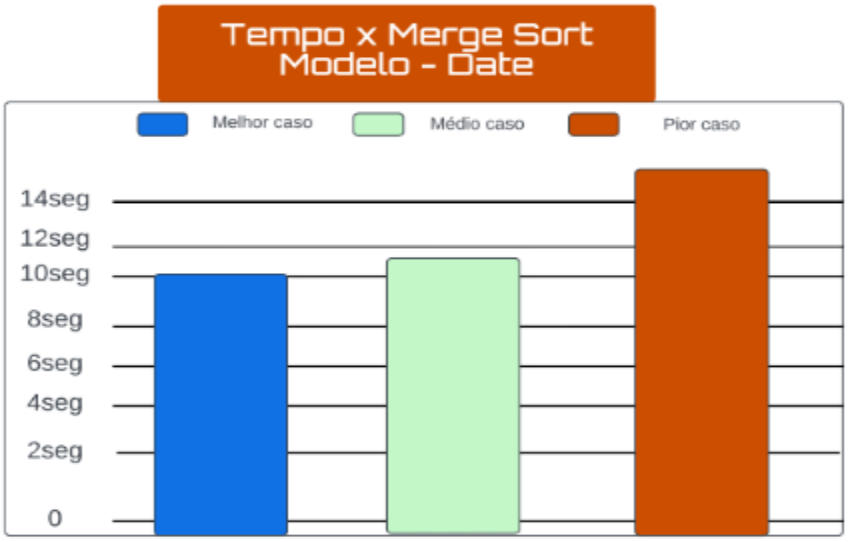


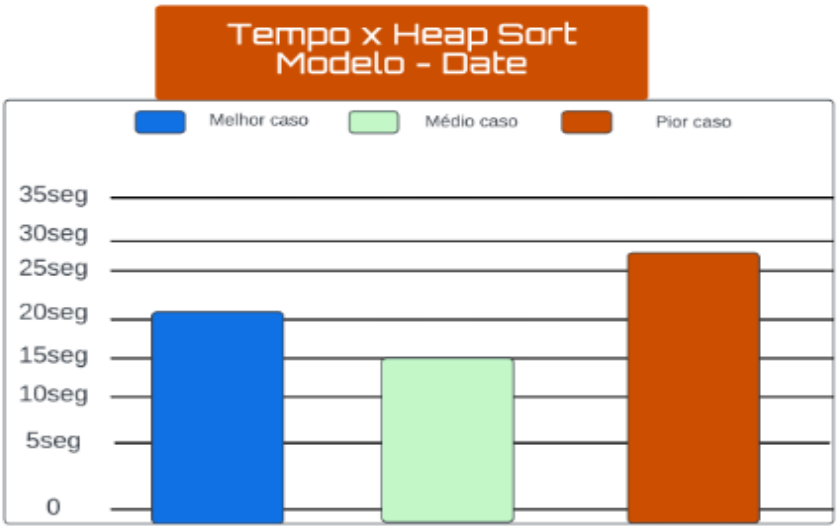


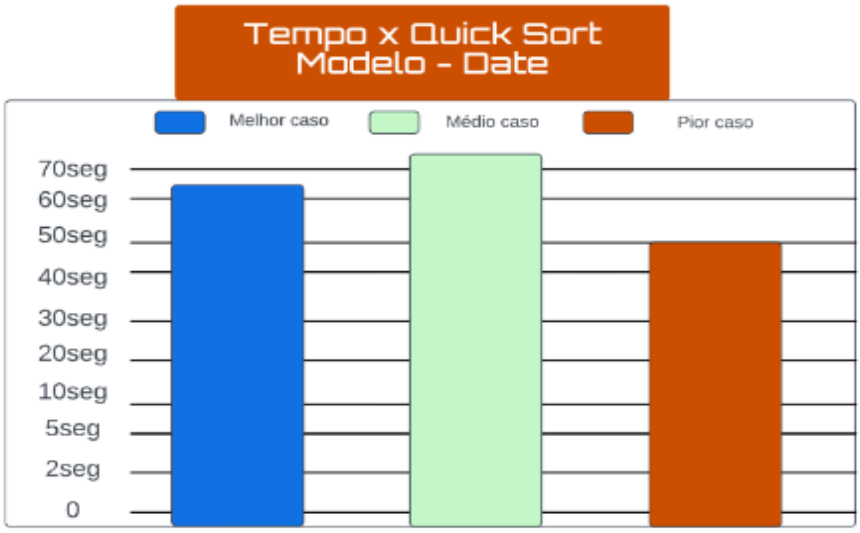
Para a ordenação 2, pode-se perceber que, no melhor caso, o Merge Sort foi o mais eficiente, com tempo próximo a 10 segundos, mas foi seguido de perto por Heap Sort, com aproximadamente 15 segundos, e Counting Sort, com aproximadamente 10 a 12 segundos. Já no médio caso, o algoritmo mais eficiente foi o Counting Sort, com 5 segundos. Além disso, no pior caso, quem se destacou foi, novamente, o Merge Sort, com um tempo entre 14 e 15 segundos. Enquanto isso, o menos eficiente em todos os casos foi o Selection Sort.

Análise de gasto de tempo para cada algoritmo implementado na ordenação 3 (ordenar pela data de início da viagem) - Insertion Sort, Selection Sort, Merge Sort, Heap Sort e Quick Sort.




Tempo x Merge Sort
Modelo - Date
Melhor caso
Médio caso
Pior caso
14seg
12seg
10seg
8seg
6seg
4seg
2seg
0


Tempo x Heap Sort
Modelo - Date
Melhor caso
Médio caso
Pior caso
35seg
30seg
25seg
20seg
15seg
10seg
5seg
0



Para a ordenação 3, pode-se perceber que o algoritmo Merge Sort foi o mais eficiente em todos os casos possíveis, chegando no máximo a 14 , 15 segundos para ordenar. O segundo mais eficiente foi o Heap Sort. Já o menos eficiente foi o Selection Sort, chegando a gastar 1500 segundos no melhor caso.

4. Conclusões

Para as ordenações 1 e 3, o algoritmo Merge Sort foi soberano como o mais eficiente, mas o Heap Sort também obteve bons resultados. Já para a ordenação 2, O Merge Sort foi o mais eficiente no melhor caso - seguido de perto pelo Heap Sort - e no pior caso. Enquanto que no médio caso quem se destacou foi o Counting Sort. (sem implementação na ordenação 1 e 3)

Enquanto isso, o posto de menos eficiente, em todas as ordenações, é dado ao Selection Sort, que obteve um gasto de tempo muito alto em todos os casos. O Insertion Sort e o Quick Sort não obtiveram resultados tão eficientes para serem considerados os melhores, nem resultados ruins para serem considerados os piores.