Laboratório de Estrutura de Dados

Primeira versão do projeto da disciplina

Comparação entre os algoritmos de ordenação elementar

Dirceu Araújo Macêdo (202080412), José Renan Alves Pereira (202080170) e Wallyson Silva Ferreira (202080323)

1. Introdução

Este relatório corresponde ao relato dos resultados obtidos no projeto da disciplina de LEDA que tem o foco nos dados do sistema de compartilhamentos de bicicletas de Metrô de Los Angeles, California e área metropolitana – Lost Angeles Metro Bike Share. O conjunto de dados utilizados representam as informações de 2016 à 2021, com informações gerais como duração da viagem, tipo de bicicleta, estação inicial, inicio da locação, fim da locação e entre outras informações.

O projeto demanda que transformações e ordenações sejam realizadas através de soluções usando JAVA. As soluções desenvolvidas estão segmentadas em dois grupos, transformações e ordenações. Em cada solução e necessário a geração de um arquivo no formato CSV, com o resultado esperado de cada grupo.

Nas soluções de transformação as mudanças realizadas no dataset são voltadas a referência do id das estações pelo nome da estação, filtros por localidade e viagens com duração maior que a média geral.

Em relação as soluções de ordenação, envolvem a aplicação de ordenação envolvendo o melhor, médio e pior caso. Nas seguintes situações:

• Ordenação pelo Nome das Estações em ordem alfabética;
• Ordenação pela Duração da Viagem, da menor viagem para a maior; e
• Ordenação pela data de início da viagem, do mais recente para o mais antigo.

Os algoritmos de ordenação utilizados, foram os seguintes:

• *Insertion Sort;*
• *Selection Sort;*
• *Merge Sort;*
• *Quick Sort;*
• *Quick Sort com Mediana de 3;*
• *Counting Sort; e*

• *Heap Sort.*

Este trabalho está dividido em 3 a seções, a primeira secção se refere a introdução, em que é fornecida uma visão geral do trabalho, segunda seção é a descrição geral, nela será abordado os testes realizados e como foi feita a implementação da ferramenta. Por fim temos a terceira e última secção, que se refere aos resultados e análise, consistindo na evidenciação da comparação dos testes realizados.

Os resultados obtidos nesse projeto permitiram inferir que os melhor algoritmo de ordenação em todos os tipos de ordenações é o Merge Sort, e o pior algoritmo em todas as situações é o *Selection Sort*. Entretanto o *Counting Sort* apresentou uma execução mais rápida que o *Merge Sort.*

2. Descrição geral sobre o método utilizado

Em virtude do tamanho do arquivo e a otimização da entrega, utilizamos uma amostra de 13.000 linhas para a execução dos testes de Ordenação, nos testes de transformação não teve necessidade de ajuste no conjunto de dados.

Em relação a análise de transformações, comparamos as transformações de acordo com a utilização de CPU, memória RAM, Tamanho do Arquivo Gerado e Tempo de Execução.

No que diz respeito a análise dos algoritmos de ordenação, as análises foram segmentadas em duas modalidades.

A primeira modalidade se refere a utilização de recursos da máquina na execução de cada algoritmo em cada uma das modalidades de ordenação – nome da estação, duração e data inicial - verificando o tempo de execução em segundos, de acordo com o modelo do arquivo: médio caso, melhor caso e pior caso.

Por fim, na segunda modalidade, realizamos uma comparação de todos os algoritmos de ordenação em relação ao tipo de caso, verificando qual deles possui o melhor tempo de execução.

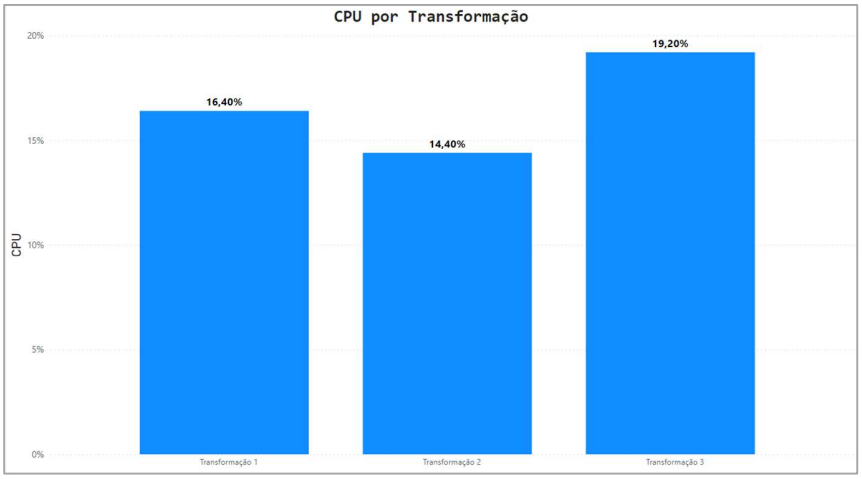
2.1. Descrição geral do ambiente de testes

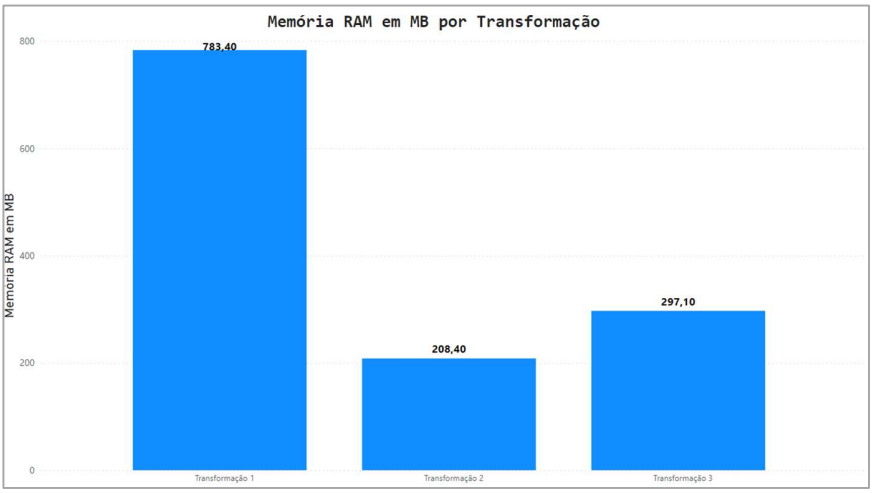
Os testes foram realizados em uma máquina virtual com as seguintes configurações:

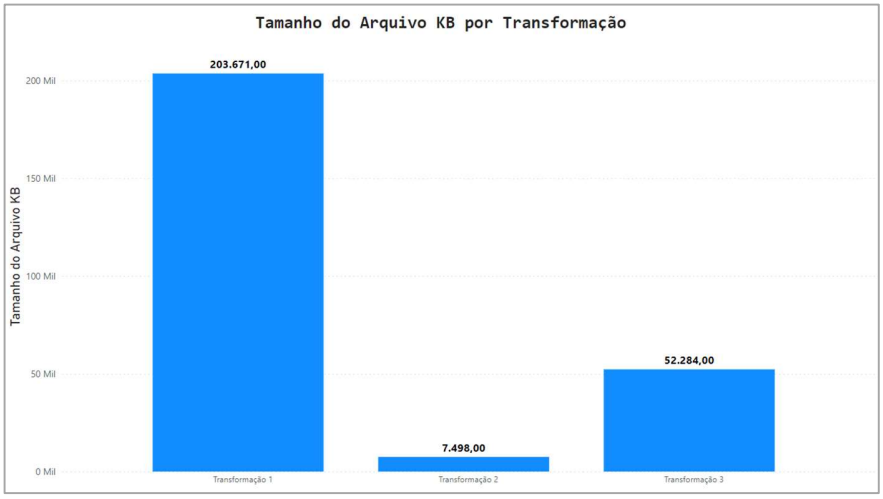
- **Processador**: AMD Ryzen 7 3800X (2 núcleos físicos e 8 threads);
- **Placa de Vídeo**: Radeon RX 590 series 8GB GDDR5;
- **Memória RAM**: 8 GB DDR4 2400mhz;
- **Sistema Operacional:** Windows 10;

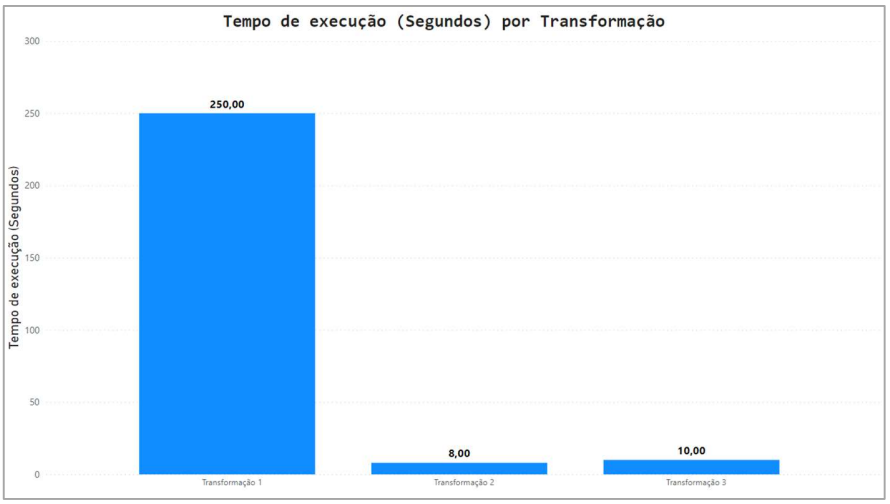
3. Resultados e Análise

Os resultados obtidos com as transformações podem ser visualizados de acordo com as imagens abaixo:




Memória RAM em MB por Transformação
Memoria RAM em MB
800
600
400
200
0
783,40
208,40
297,10
Transformação 1
Transformação 2
Transformação 3


Tamanho do Arquivo KB por Transformação
Tamanho do Arquivo KB
200 Mil
150 Mil
100 Mil
50 Mil
0 Mil
203.671,00
7.498,00
52.284,00
Transformação 1
Transformação 2
Transformação 3


Tempo de execução (Segundos) por Transformação
Tempo de execução (Segundos)
300
250
200
150
100
50
0
250,00
8,00
10,00
Transformação 1
Transformação 2
Transformação 3

A primeira transformação corresponde a uma substituição de informações da id_start_station e id_end_station do arquivo original por station_name do arquivo “stations.csv”. Apresentando um aumento do tamanho do arquivo original de 173.035 KB para 203.671 KB, representando um aumento de 30.636 KB (17,75%). Essa execução precisou realizar substituições em todo o arquivo, sendo a etapa que mais consumiu recursos.

A segunda etapa, por se tratar de somente um filtro, utilizamos um arquivo auxiliar para a execução dessa etapa, denominado “stations_passadena.csv”. Sendo a etapa mais rápida de transformação em todos os quesitos. A transformação 3 ficou em segundo lugar em todos os quesitos por necessitar percorrer o arquivo inteiro para identificar a média de 24,07 e realizar o filtro.

De modo geral todos os algoritmos de ordenação funcionaram corretamente utilizando uma amostra pequena do conjunto de dados 13.000 linhas, porém na execução completa do conjunto de dados, alguns algoritmos de ordenação não foram finalizados devido ao tamanho da data set.

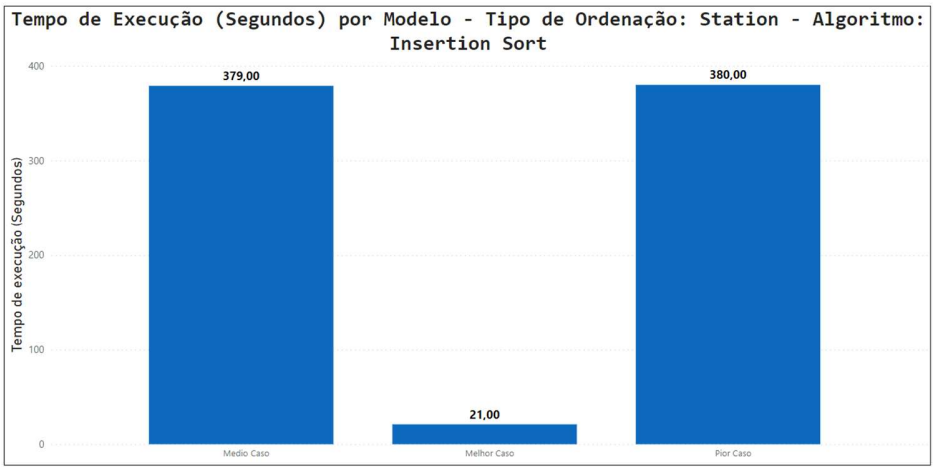
Ao realizar o teste utilizando algoritmos de ordenação do tipo *Selection Sort*, constata-se que devido a chamada recursiva, na ordenação - foram utilizados dois "for" para realizar a comparação. Dessa forma a ordenação precisou percorrer o arquivo de 1.250.835 linhas 1.250.835 vezes, cada percurso do arquivo levou em média 15 segundos para ser percorrido. Resultando na inviabilidade de tentar ordenar esse arquivo, pois levaria aproximadamente 5.212 horas para ordenar o arquivo no pior e melhor caso.

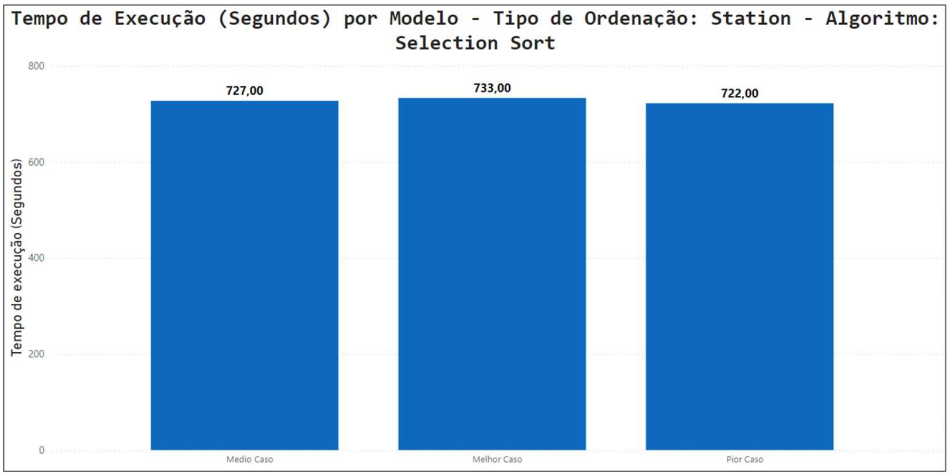
O *Insertion Sort* apresentou uma ordenação mais rápida em relação ao *Selection Sort*, porém devido ao tamanho do arquivo, a análise acabou ficando comprometida, pois esse algoritmo realiza aproximadamente 700 ordenações por minuto. Dessa forma para realizar a verificação desse tipo de ordenação seria necessário aproximadamente 30 hrs para a realização dessa ordenação. Esse algoritmo de ordenação é mais rápido pois utiliza somente um for pra realizar a comparação, entretanto necessita percorrer o arquivo de 1.250.835 linhas. Resultando nesse tempo estimado para realizar a ordenação.

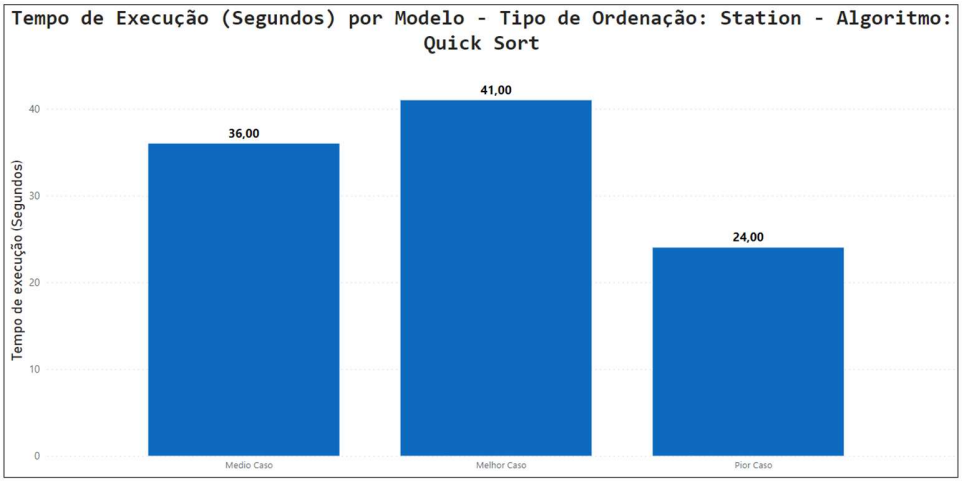
Ao realizar a ordenação pelo nome da estação inicial (string), não foi possível utilizar os algoritmos de *Counting Sort* e *Quick Sort* mediana de 3. No *Counting Sort* constatamos que só executa utilizando o formato char e int, e no *Quick Sort* de mediana de 3, não foi possível adaptar o algoritmo para o formato de string.

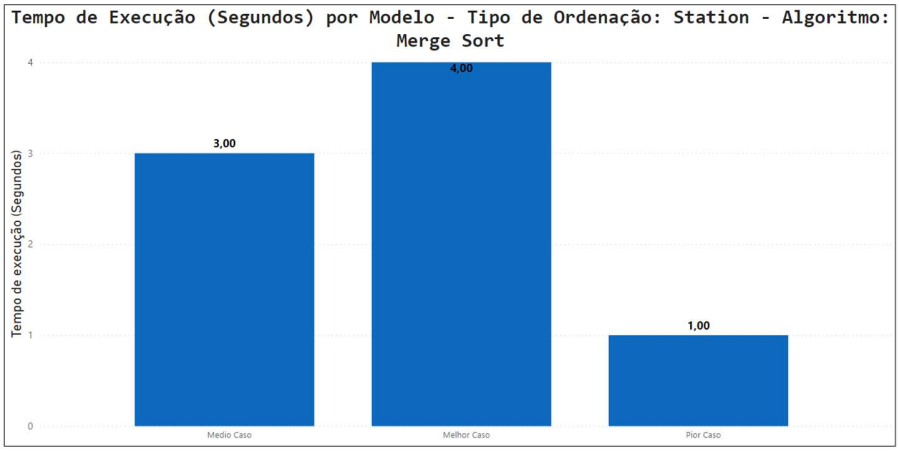
Ao ordenar pela data inicial, somente na ordenação por *Counting Sort* não foi possível realizar a ordenação, nos demais algoritmos as ordenações foram executadas sem complicações. Entretanto, observamos que o conjunto de dados na coluna de datas apresentou dois formatos, o formato pt-br "DD-MM-AA" (Dia – Mês – Ano) e o formato em-us "MM-DIA-AA" (Mês – Dia – Ano"), onde foi criado uma função para identificar o formato correto, porém no formato americano os meses menores que 10, apresentaram divergências, na função criada para corrigir esse erro no conjunto não funcionou corretamente nessa situação, pois não existia o valor 0.

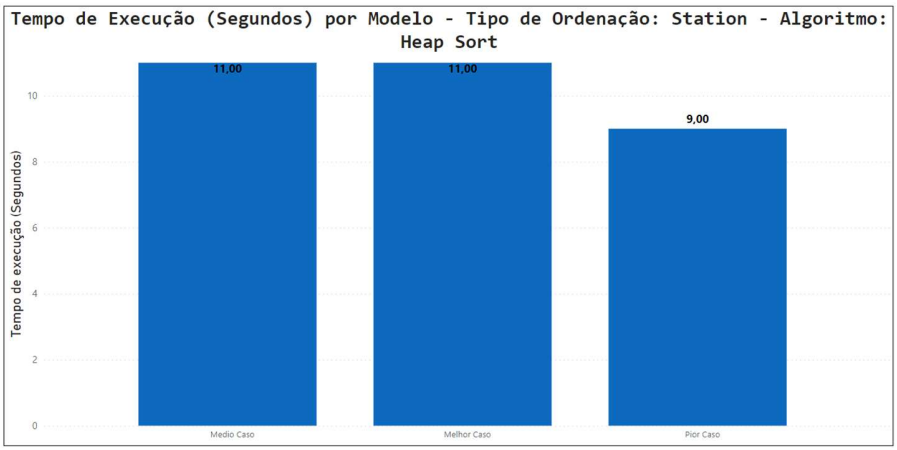
A ordenação por duração, foi a única que em que todos os algoritmos abordados foram executados com sucesso. Abaixo as análises no formato de gráfico abordadas:




Tempo de Execução (Segundos) por Modelo - Tipo de Ordenação: Station - Algoritmo: Selection Sort
Tempo de execução (Segundos)
800
600
400
200
0
727,00
733,00
722,00
Medio Caso
Melhor Caso
Pior Caso

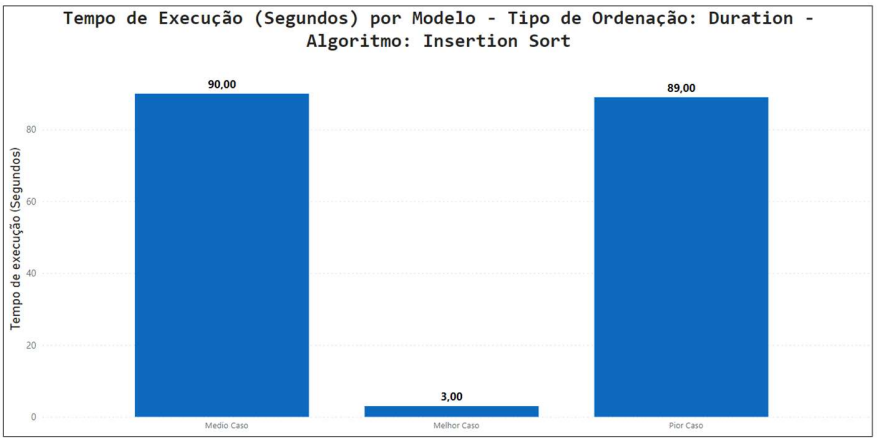

Tempo de Execução (Segundos) por Modelo - Tipo de Ordenação: Station - Algoritmo: Quick Sort
Tempo de execução (Segundos)
40
30
20
10
0
36,00
41,00
24,00
Medio Caso
Melhor Caso
Pior Caso

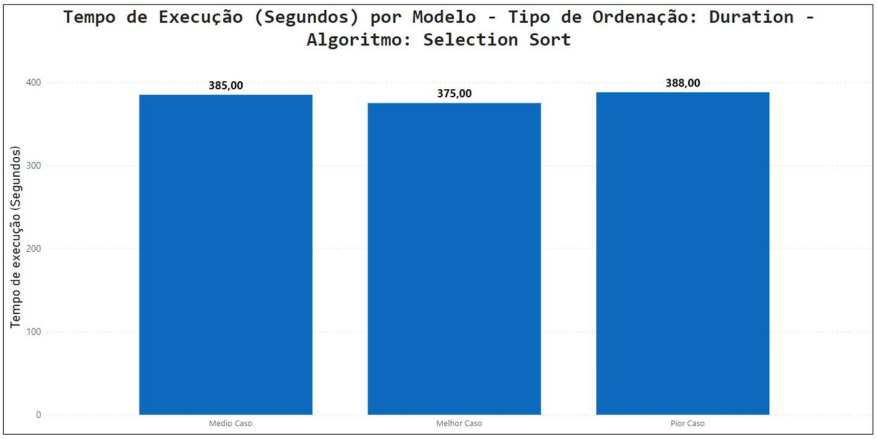

Tempo de Execução (Segundos) por Modelo - Tipo de Ordenação: Station - Algoritmo: Merge Sort
Tempo de execução (Segundos)
4
3
2
1
0
3,00
4,00
1,00
Medio Caso
Melhor Caso
Pior Caso

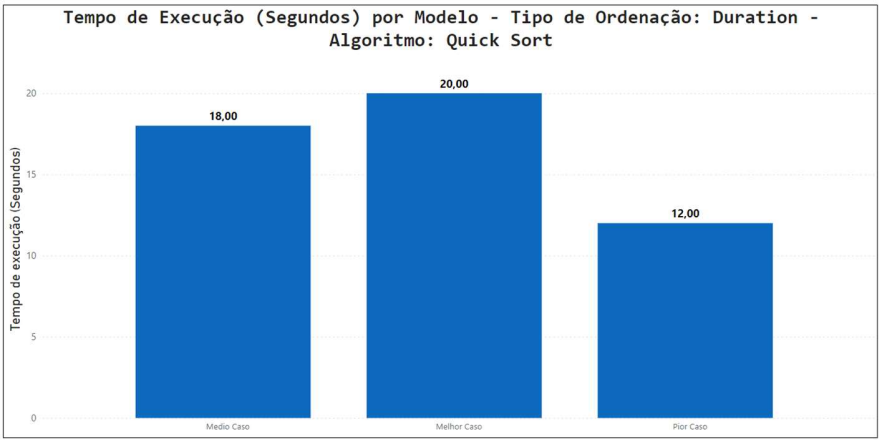


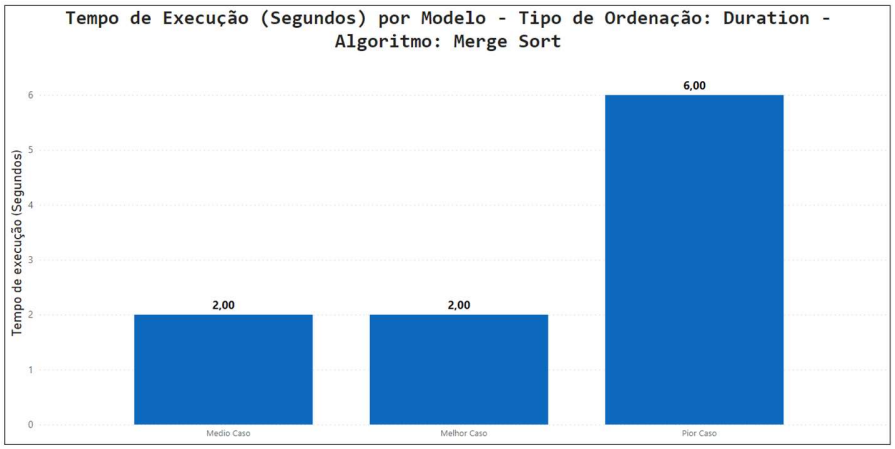
Na modalidade 1, considerando a Ordenação por *Station* (Nome), o algoritmo que executou mais rapidamente foi o *Merge Sort,* levando em média 3 segundos para realizar a ordenação nessa situação.

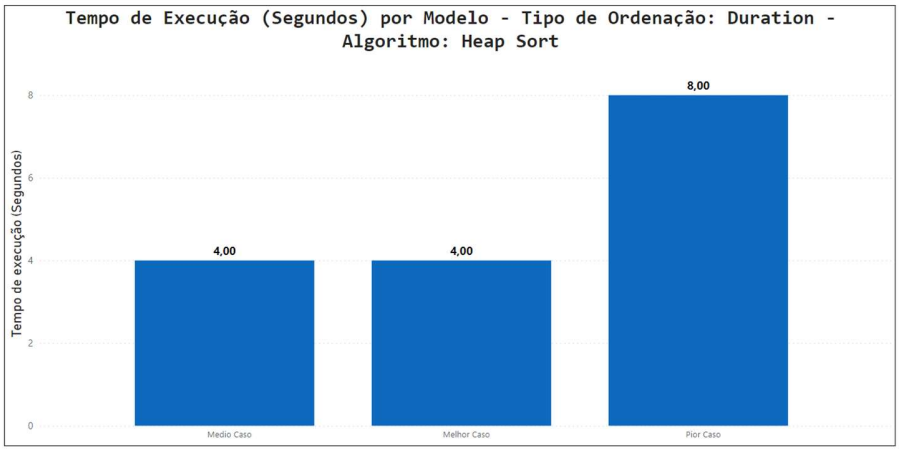
Ordenações por Duration:

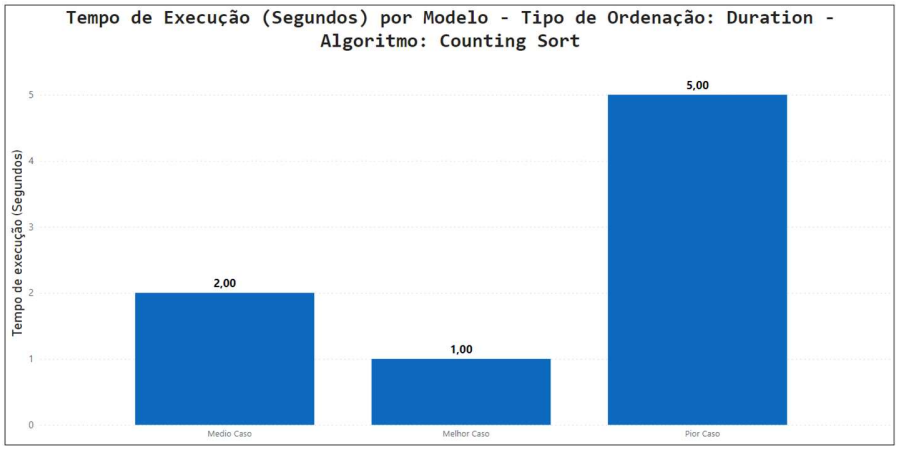



Tempo de Execução (Segundos) por Modelo - Tipo de Ordenação: Duration - Algoritmo: Selection Sort
Tempo de execução (Segundos)
400
300
200
100
0
385,00
375,00
388,00
Medio Caso
Melhor Caso
Pior Caso


Tempo de Execução (Segundos) por Modelo - Tipo de Ordenação: Duration - Algoritmo: Quick Sort
Tempo de execução (Segundos)
20
15
10
5
0
18,00
20,00
12,00
Medio Caso
Melhor Caso
Pior Caso

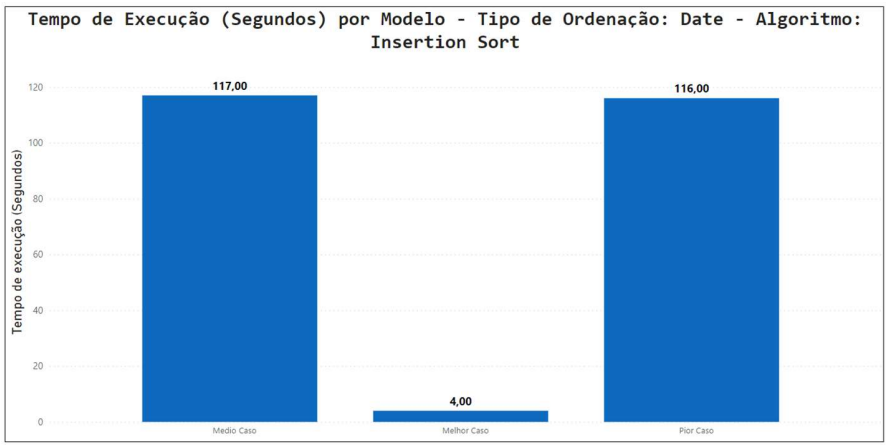

Tempo de Execução (Segundos) por Modelo - Tipo de Ordenação: Duration - Algoritmo: Merge Sort
Tempo de execução (Segundos)
6
5
4
3
2
1
0
2,00
2,00
6,00
Medio Caso
Melhor Caso
Pior Caso

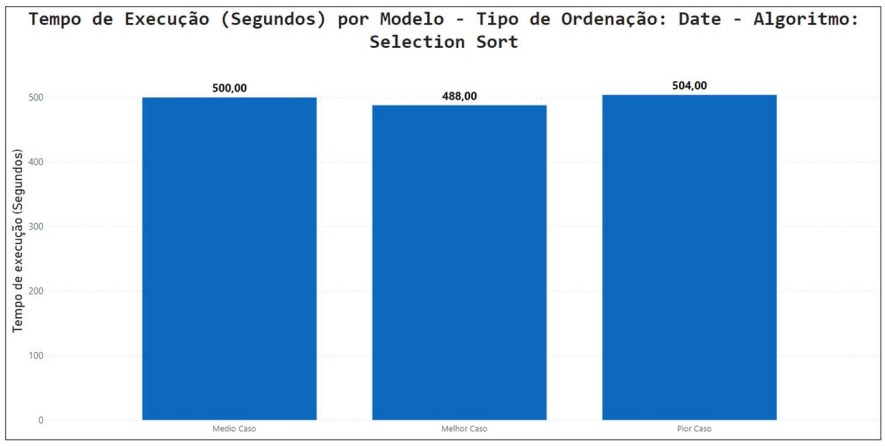


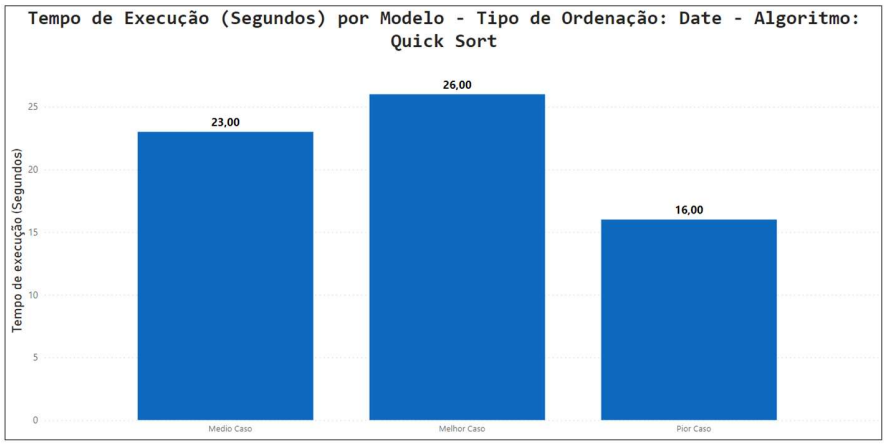


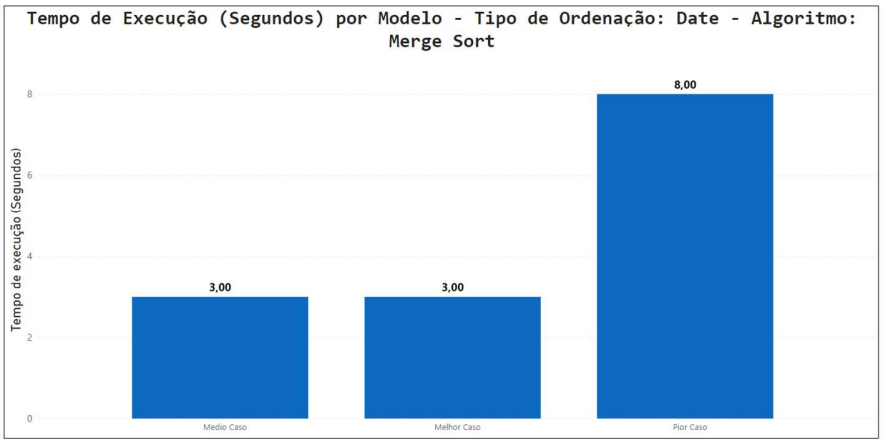
Ao realizar a ordenação pela coluna duração, observamos que os algoritmos *Merge Sort* e *Counting Sort*, apresentaram os melhores Resultados. O *Selection Sort* apresenta o pior resultados de ordenação.

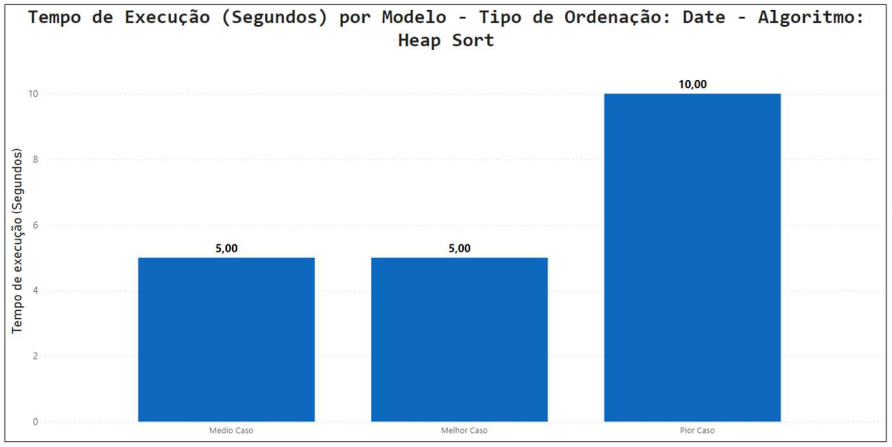
Ordenação Start_Date por algoritmos.

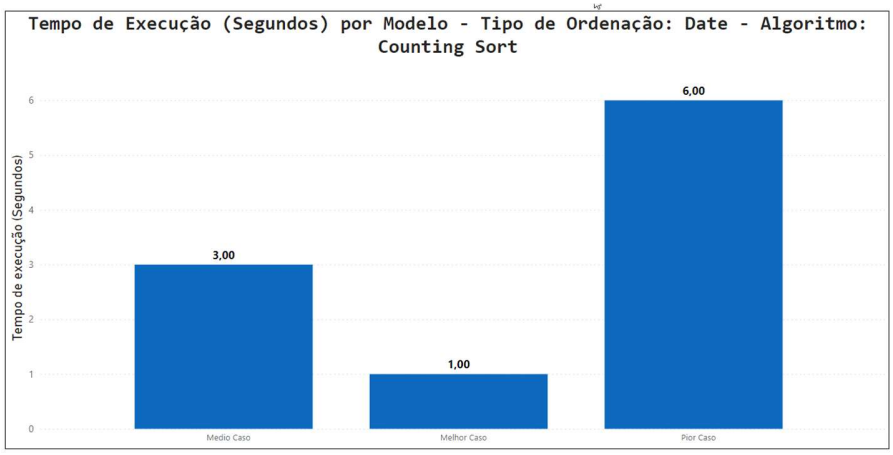

Tempo de Execução (Segundos) por Modelo - Tipo de Ordenação: Date - Algoritmo: Insertion Sort
Tempo de execução (Segundos)
120
100
80
60
40
20
0
117,00
4,00
116,00
Medio Caso
Melhor Caso
Pior Caso

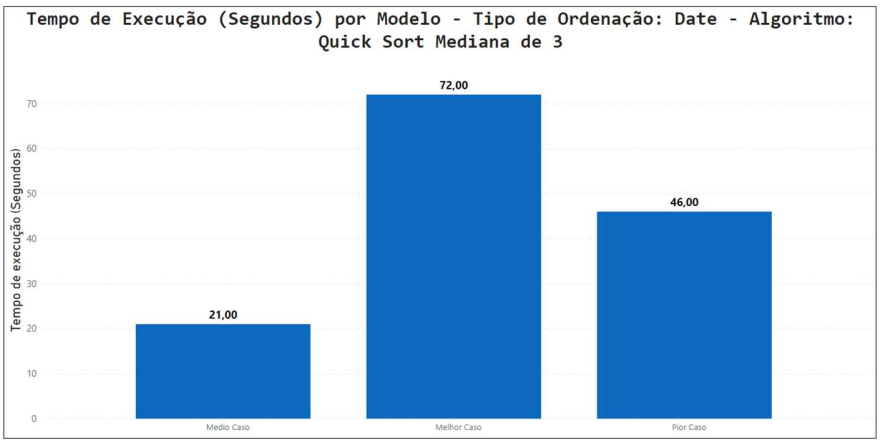

Tempo de Execução (Segundos) por Modelo - Tipo de Ordenação: Date - Algoritmo: Selection Sort
Tempo de execução (Segundos)
500
400
300
200
100
0
500,00
488,00
504,00
Medio Caso
Melhor Caso
Pior Caso


Tempo de Execução (Segundos) por Modelo - Tipo de Ordenação: Date - Algoritmo: Quick Sort
Tempo de execução (Segundos)
25
20
15
10
5
0
23,00
26,00
16,00
Medio Caso
Melhor Caso
Pior Caso

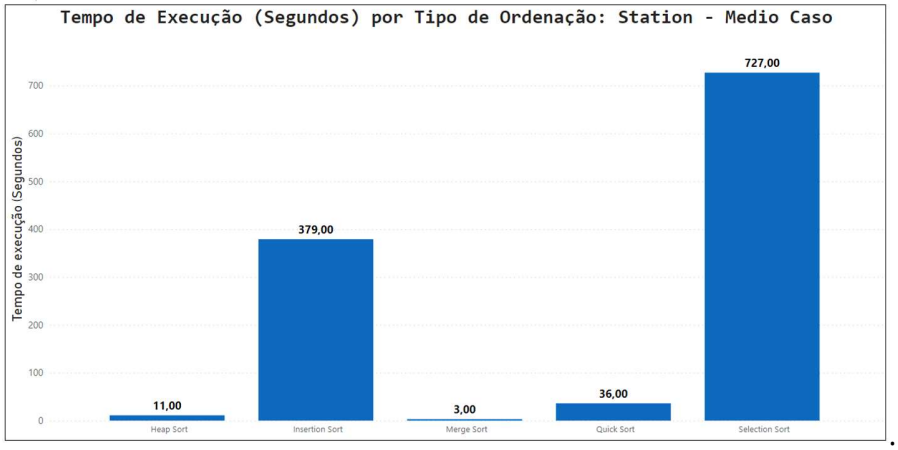

Tempo de Execução (Segundos) por Modelo - Tipo de Ordenação: Date - Algoritmo: Merge Sort
Tempo de execução (Segundos)
0
2
4
6
8
3,00
3,00
8,00
Medio Caso
Melhor Caso
Pior Caso

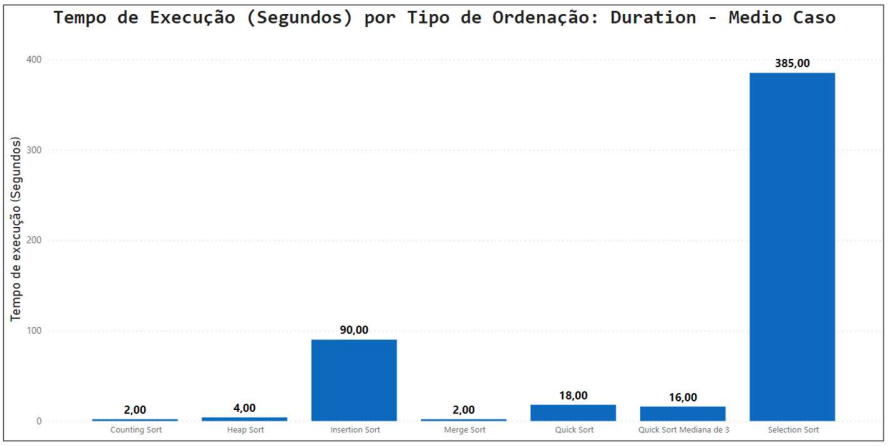

Tempo de Execução (Segundos) por Modelo - Tipo de Ordenação: Date - Algoritmo: Heap Sort
Tempo de execução (Segundos)
0
2
4
6
8
10
5,00
5,00
10,00
Medio Caso
Melhor Caso
Pior Caso

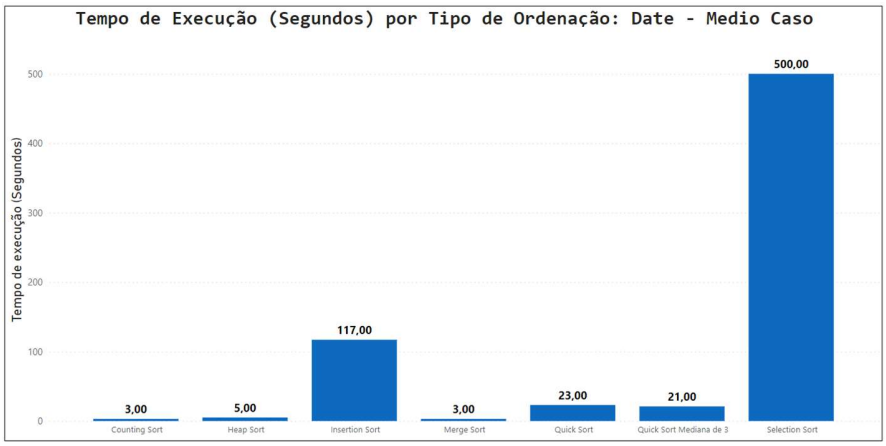

Tempo de Execução (Segundos) por Modelo - Tipo de Ordenação: Date - Algoritmo: Counting Sort
Tempo de execução (Segundos)
0
1
2
3
4
5
6
3,00
1,00
6,00
Medio Caso
Melhor Caso
Pior Caso



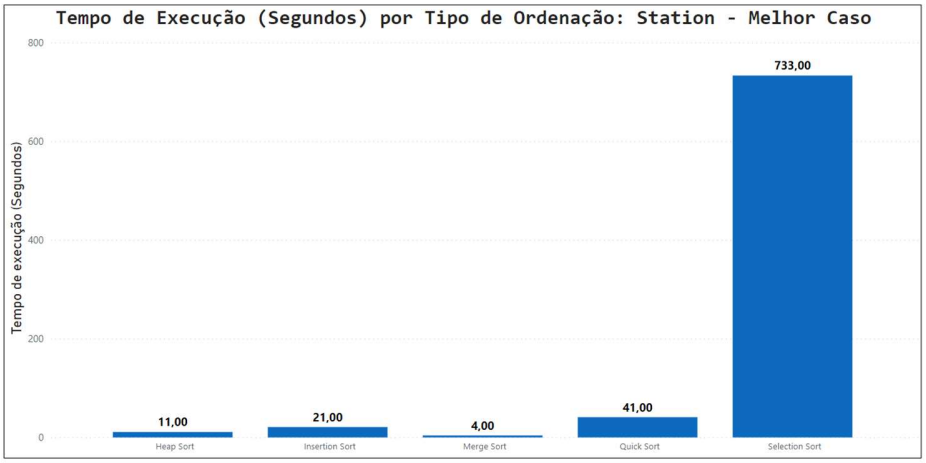
Comparações por tipo de ordenação e Médio Caso:

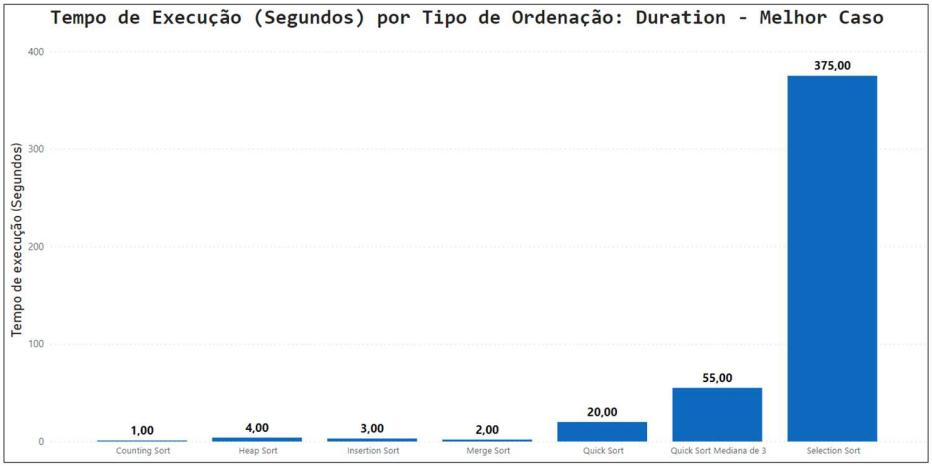


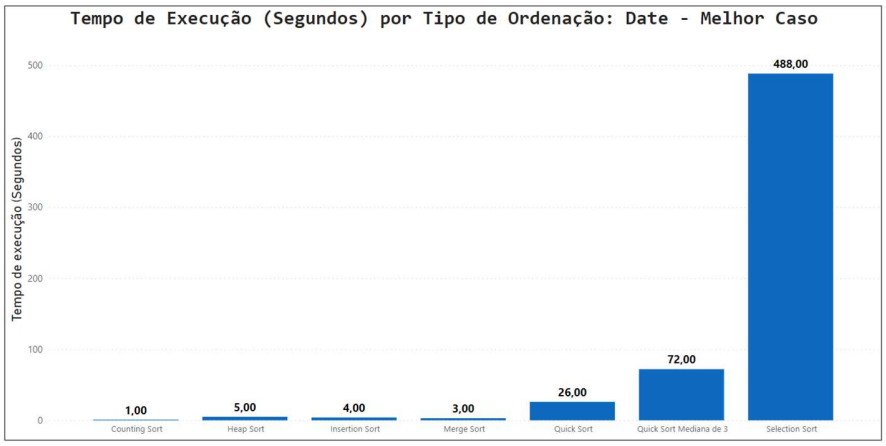




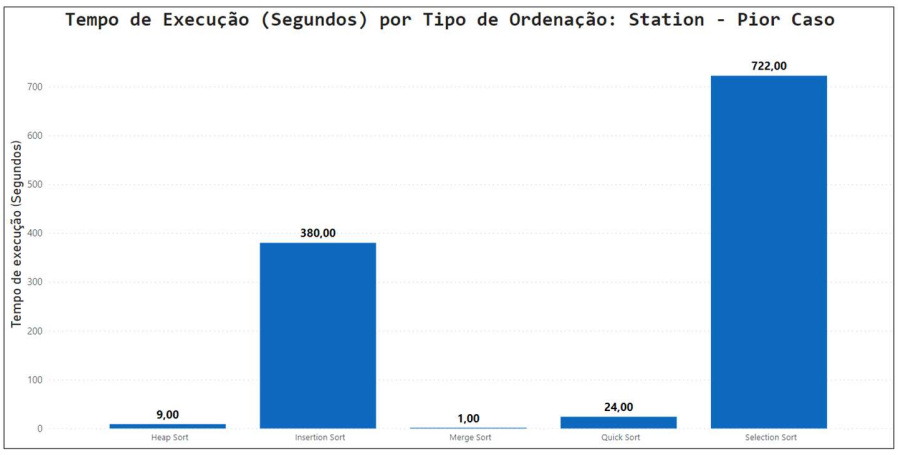
Comparações por tipo de ordenação e Melhor Caso:

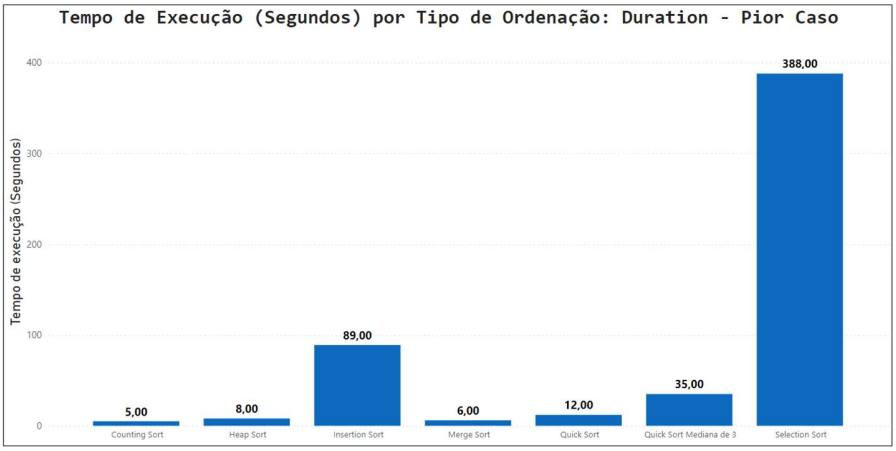


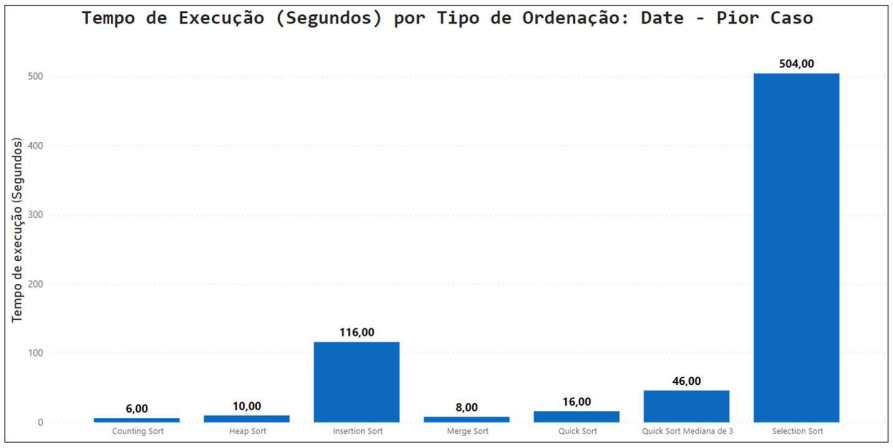




Comparações por tipo de ordenação e Pior Caso:







Com os resultados obtidos, constata-se que o algoritmo mais ineficiente na questão de tempo é o *Selection Sort* em todos os Tipos de Ordenação e em todos os casos de Ordenação. Em relação ao melhor algoritmo, o *Merge Sort* foi o que apresentou o melhor desempenho em relação ao tipo de ordenação e os tipos de caso, pois foi o único que conseguimos reproduzir em todos os cenários de ordenação. Ao verificarmos a ordenação por Duração (*Duration*) e data (*Date*) o *Counting Sort*, foi o que teve o melhor desempenho, porém não podemos afirmar que é o melhor pois não conseguimos executar esse algoritmo na ordenação por estação (*Station*).