Laboratório de Estrutura de Dados

Primeira versão do projeto da disciplina

Comparação entre os algoritmos de ordenação elementar

Wilton Lima Silva Saturnino, **João Victor Barbosa**

1 Introdução

Este trabalho foi desenvolvido para aplicação dos conceitos teóricos de programação estudados na disciplina Estrutura de Dados em paralelo com Laboratório de Estrutura de Dados. Espera-se com os resultados aplicar os algoritmos de ordenação e comparalos conforme os diferentes casos de complexidade estudados na disciplina.

Com base no que foi dito, os dados devem ser baixados na plataforma indicada pelo professor e a partir dos filtros disponibilizados deve-se baixarr os arquivos de dados para o sistema de compartilhamento de biciletas na cidade de Los Angeles, California e região metropolitana da empresa Los Angeles Bike Share.

O tratamento dos dados está compreendiada entre 2016 e 2021 podem ser realizados a partir da aplicação dos algoritmos para ordenar os dados desenvolvidos em linfuagem JAVA e aplicando sobre arquivos CSV e para melhor visualização os resultados de tempo e desempenho podem ser plotados para verificar qual o método aplicado é mais eficiente para os diferentes casos de complexidade (melhor caso, pior caso, caso médio).

As tranformações sugeridas são realizadas a partir da identificação do id da estação substituindo pelo nome da estação contida no arquivo CSV, filtrar apenas as viagens que estão nas estações de maneira que seja considerado como prioridade as viagens que possuem duração maior que a média geral.

As ordenação são feias com base no resultado obtido pelas transformações e tratados em três casos. Ordenar o arquivo completo pelo nomes das estações em ordem alfabética, ordenar pelo campo de duração da viagem (campo duration) do menor para o maior, ordenar pela data de início da viagem mais recente para o mais antigo.

Para cada caso, melhor, médio e pior foram aplicados os seguintes algoritmos:

Insertion Sort; Selection Sort; Merge Sort; Quick Sort; Quick Sort com Mediana de 3; Counting Sort; e Heap Sort.

2 Descrição geral sobre o método utilizado

Neste trabalho foi necessário a utilização de um computador pessoal Windows e a teoria ministrada na disciplina a partir de reuniões em aulas presenciais. Para tratar os dados nos foi realizado testes de complexidade, os dados foram modificados sendo quê no melhor caso modificas-se a ordem de forma crescente, no pior caso os dados são ajustados de forma invertida, e no caso médio eles são ajustados de forma randômica.

Executando o programa os dados contidos na pasta de dados csv são utilizados com upload no programa a partir do caminho aonde se encontra a pasta no computador e em seguida esses dados são filtrados pelo programa aplicando as ordenações e realizado os testes de complexidade para o melhor, médio e pior caso além da verificação do tempo esforço da máquina nos diferentes algoritmos de ordenação. Os dados gerados como resultados são separados e disponibilizado em uma pasta de dados filtrados em csv ao qual pode-se utilizar para entender o comparativo de tempo de trabalho dos diferentes métodos sendo possível utilizar softwares como o visualvm para visualizá-los.

A depender da CPU, Memória RAM, tamanho de arquivo e tempo de execução, pode-se se fazer defierêntes interpratações sobe o custo benefício da ulilização dos métodos e transforções, por isso foi feita uma análise que se refere aos recursos de máquina disponível para o processamento de cada algoritmo em cada caso de complexidade.

2.1 Descrição geral do ambiente de testes

(processamento, memória, sistema operacional)

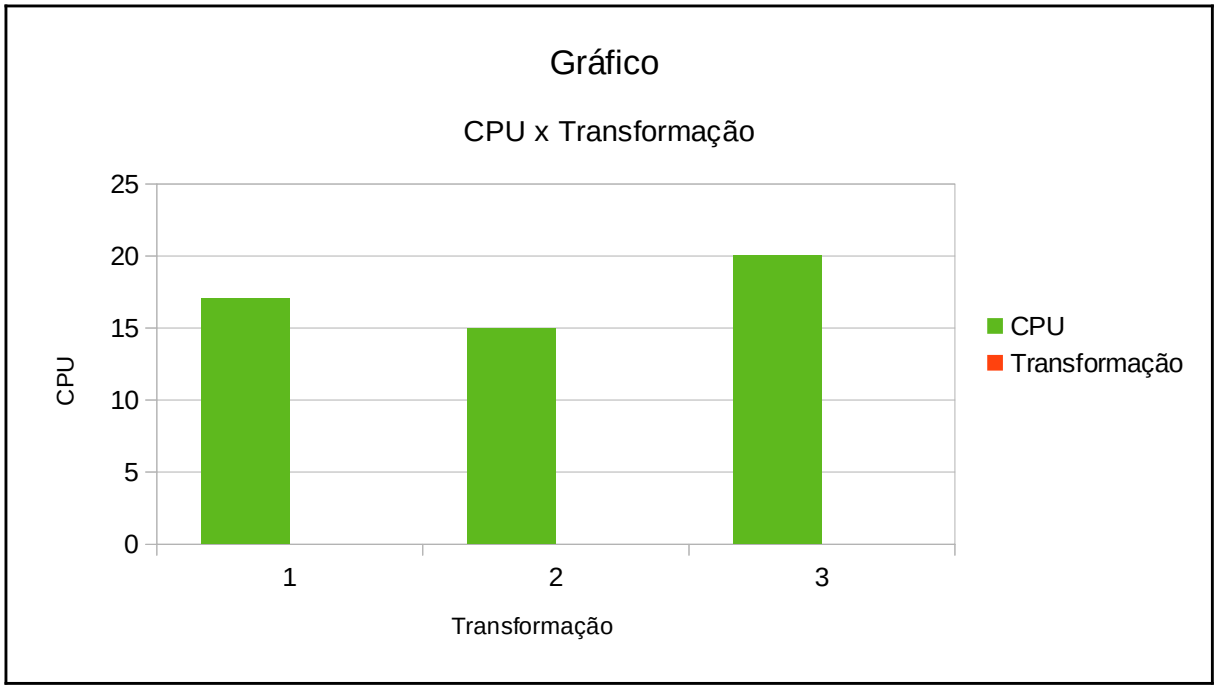
- Computador da empresa Dell;

- Memória 16gb Ddr4 P/ Notebook Dell Inspiron 14 5000 ;
- Processador Intel Core i5 10210U;
- Eclipse software;
- Sistema operacional Microsoft Windows;
- Microsoft Excel;

3 Resultados e Análise

Resultados dos testes usando tabelas e gráficos

Os resultados obtidos podem ser observados por meio da visulalização do comportamento da máquina e desempenho como no gráfico abaixo:



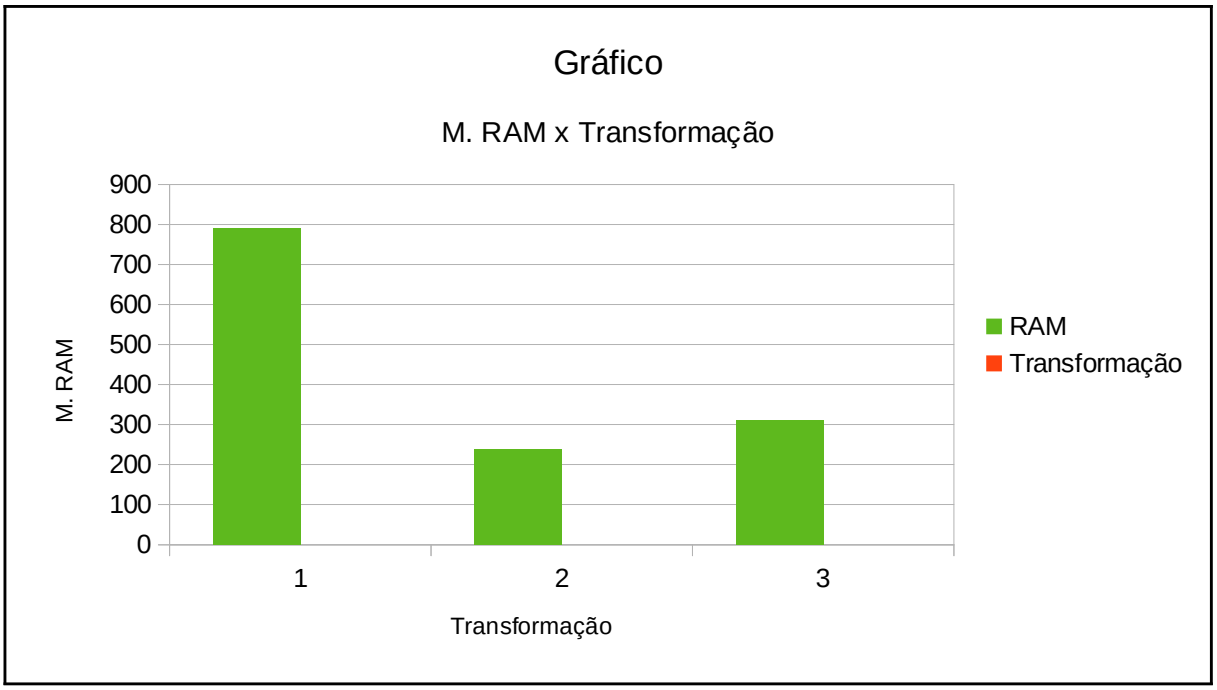
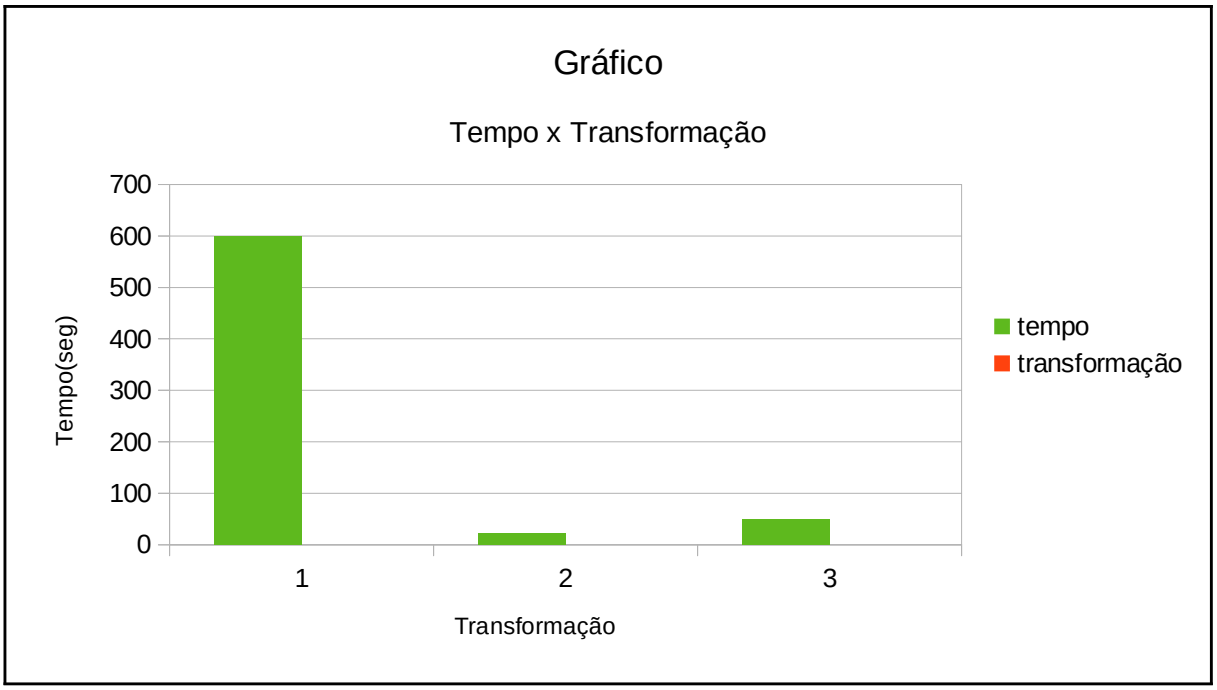


Gráfico
M. RAM x Transformação
900
800
700
600
500
400
300
200
100
0
M. RAM
1
2
3
Transformação
RAM
Transformação



No gráfico 01 é apresentado a primera transformação substituindo os dados dos arquios id_star_station e id_end_station como solicido pelos nomes gerando o arquivo station_name do “station.csv”. Com esse procedimento o arquivo da um salto de tamanho originaltente de 173.035 kb para **235,567**KB, um acrescimo de aproximadamente 62,532KB**.**

O proximo passo será utilizar um arquivo chamado “station_passadena” filtrar apenas as viagens que estão nas estações de Pasadena. Da mesma forma o aqruivo é processado de forma mais agil do processo pro se tratar de um filtro de forma que a terceira transformação se porta de forma intermédiaria no tempo pela necessidade de percorrer todo o arquivo para a realização da filtragem.

Partindo para os métodos de ordenações ultilizando o algoritmo Selection Sort, é perceptivél sua inviabilidade de maneira que pelo modelo de comparação utilizado baseado em se passar sempre o menor valor do vetor para a primeira posição, ajustando o menor valor pra segunda posição e assim sucessivamente, o processo se mostrou trabalhoso e pela quantidade de dados a serem processados seriam necessários processsar **XXX (milhões)** vezes o código gastando para isso xx seg, o que gastariam **xx horas** para serem ordenados seus casos de complexidade.

Comparado ao Selection Sort o inserction sort mostrou estrategicamente mais inteligente a ser utilizado neste projeto porque é um dos altoritmos mais eficientes em modelo de ordem quadrática, porém ele tem um comportamento melhor perante pequenas entradas, o que acabou sendo inviável nesse problema, necessitando de mais de 700 ordenações por min, extrapolando 40 hr para resolução de todas as ordenações.

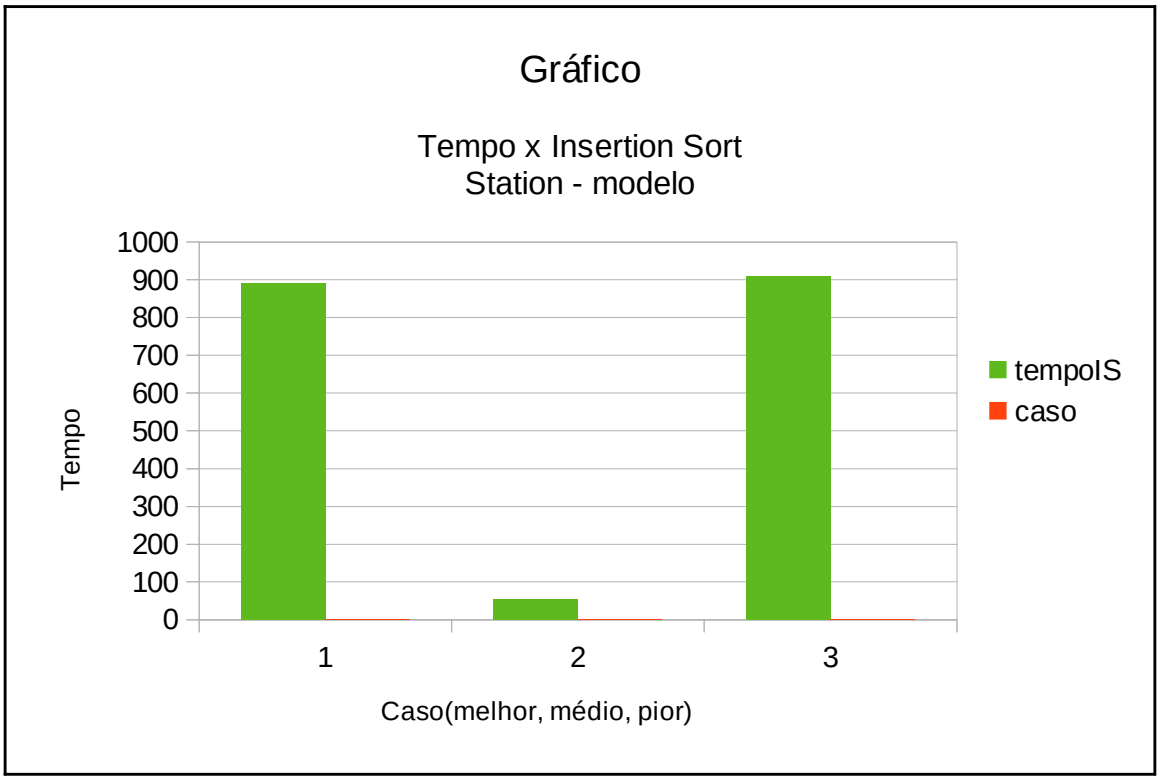
Sendo assim seria necessário entender o tradeoff porque apesar de necessitar de menos código para seu funionamento é necessário percorrer cada linha e realizando a ordenação por inserção começando pelo indice um e cada nova posíçaõ deve ser ordenada procurando a posição correta no subarray ordenado à esquerda daquela posição.

Alguns algoritmos testados não foram efetivos ou acessiveis a esse problema de maneira que foi inviável adaptar a ordenação ao tipo string. O Quick Sort, Count Sort e Mediana de 3

Na figura abaixo o gráfico da ordenações abordadas anteriormente:

Como resultado visivelmente o algoritmo selection sort se mostrou mais inviável pois leva um tempo elevado em todos os tipos de complexidade de

ordenação, por ser composto de dois laços um interno e outro externo, a checagem pra esse tipo de problema exige comparação a cada iteração um elemento com todos os outros exigindo mais esforço da máquina.



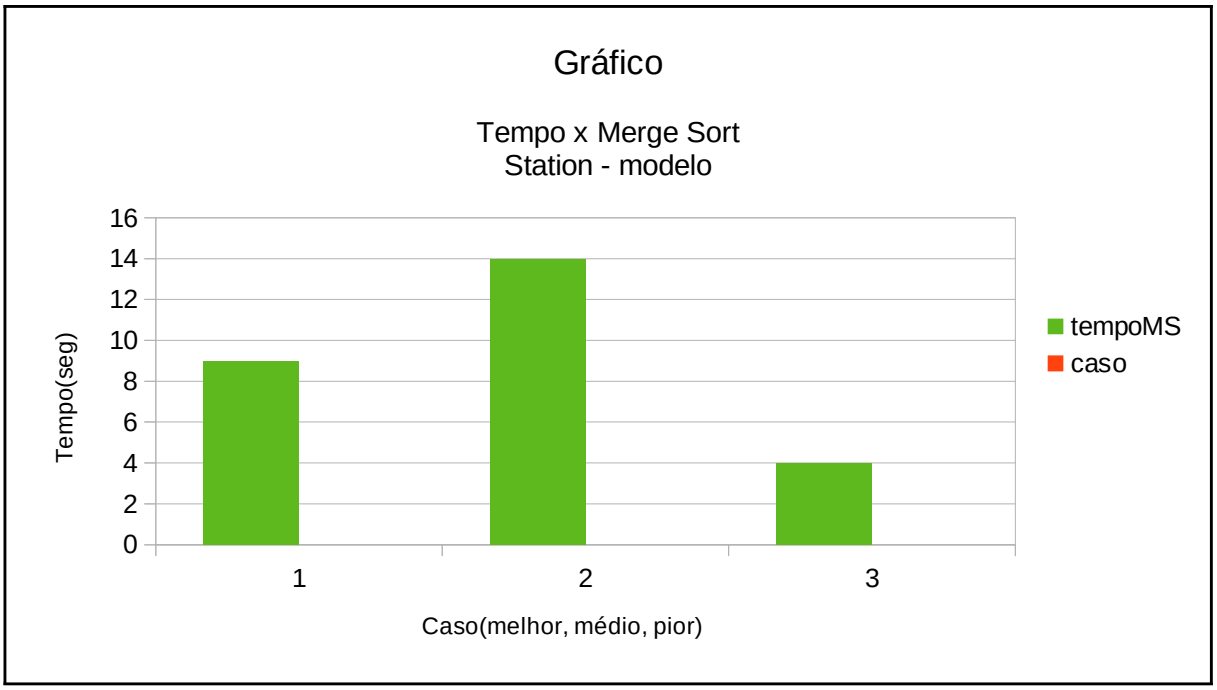


Gráfico
Tempo x Merge Sort
Station - modelo
16
14
12
10
8
6
4
2
0
Tempo(seg)
1
2
3
Caso(melhor, médio, pior)
tempoMS
caso

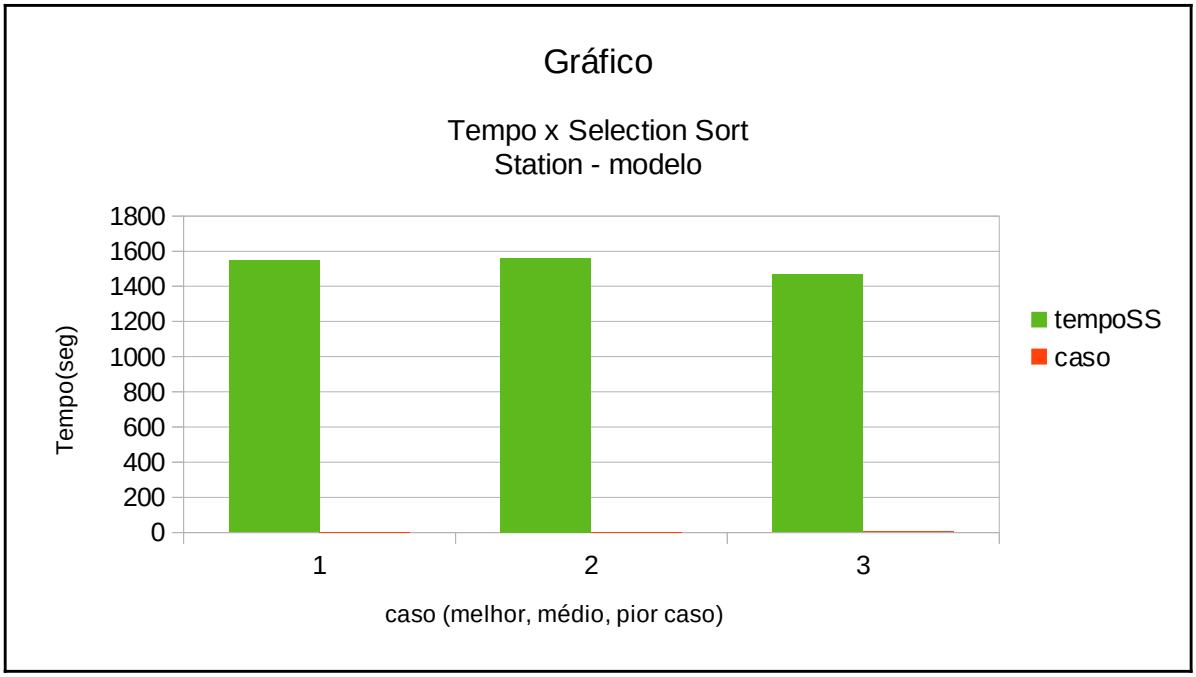
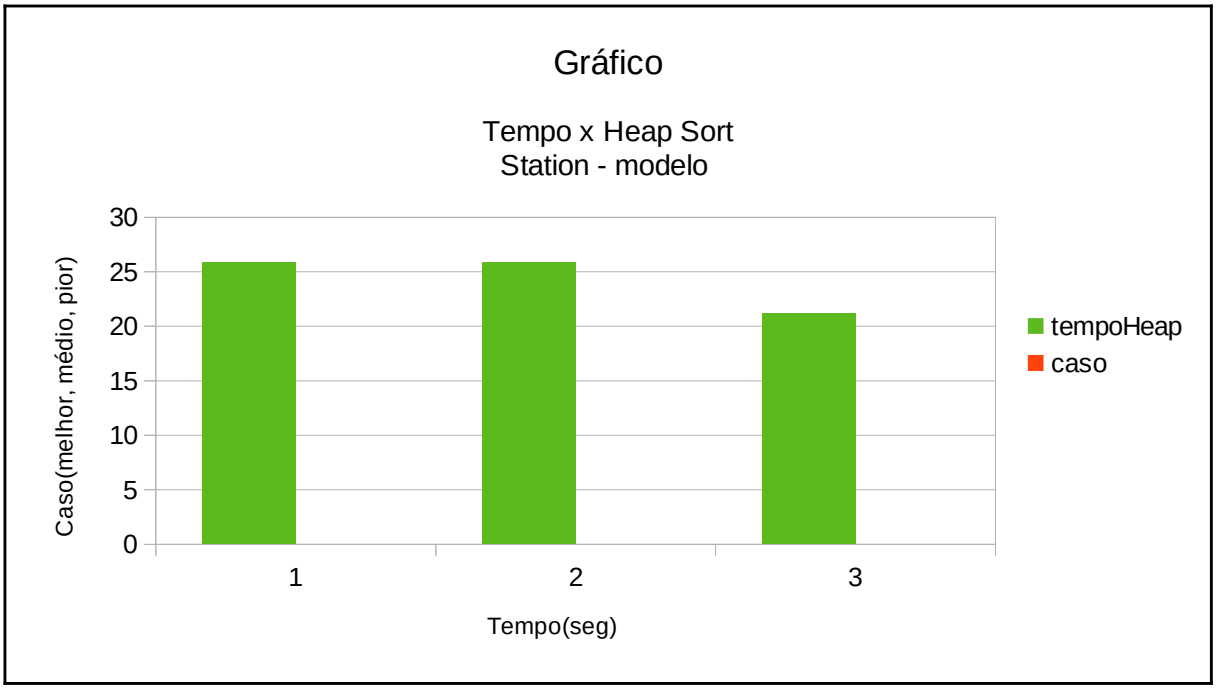


Gráfico
Tempo x Selection Sort
Station - modelo
1800
1600
1400
1200
1000
800
600
400
200
0
Tempo(seg)
1
2
3
caso (melhor, médio, pior caso)
tempoSS
caso



A Ordenação por Station (Nome), o algoritmo que executou mais rapidamente foi o Merge Sort, excutando no menor tempo de ordenação em aproximadamente 3 seg.

Oredanções por Duração:

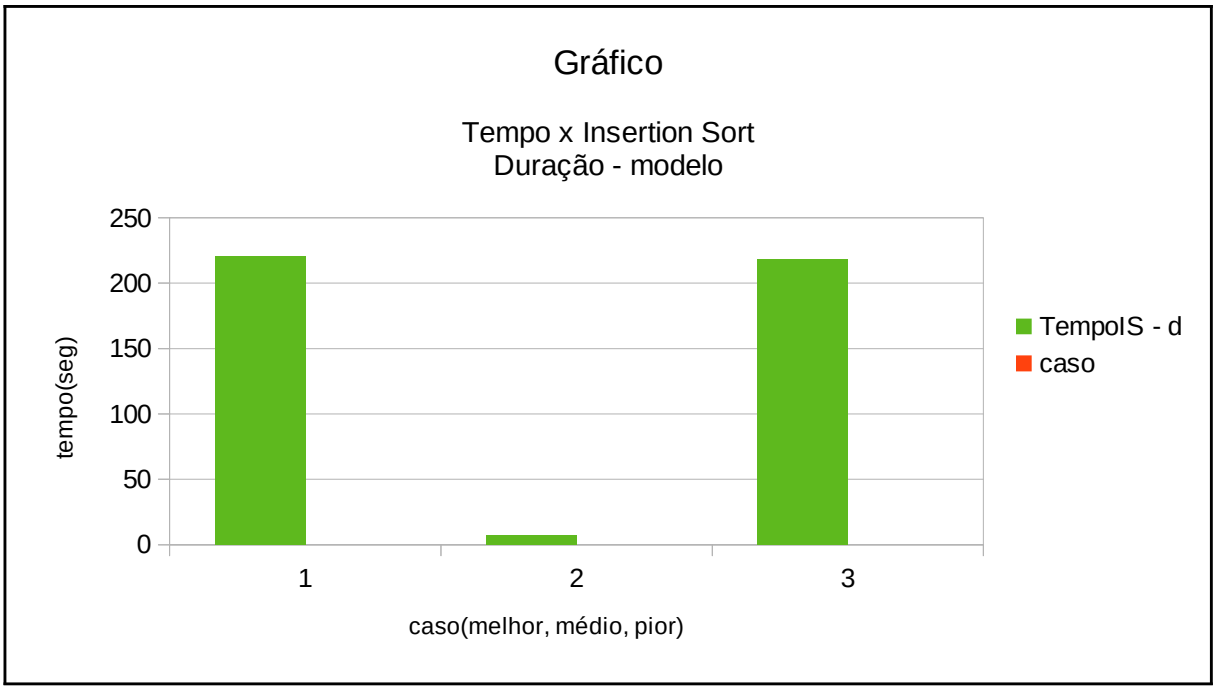


Gráfico
Tempo x Insertion Sort
Duração - modelo
250
200
150
100
50
0
tempo(seg)
1
2
3
caso(melhor, médio, pior)
TempoIS - d
caso

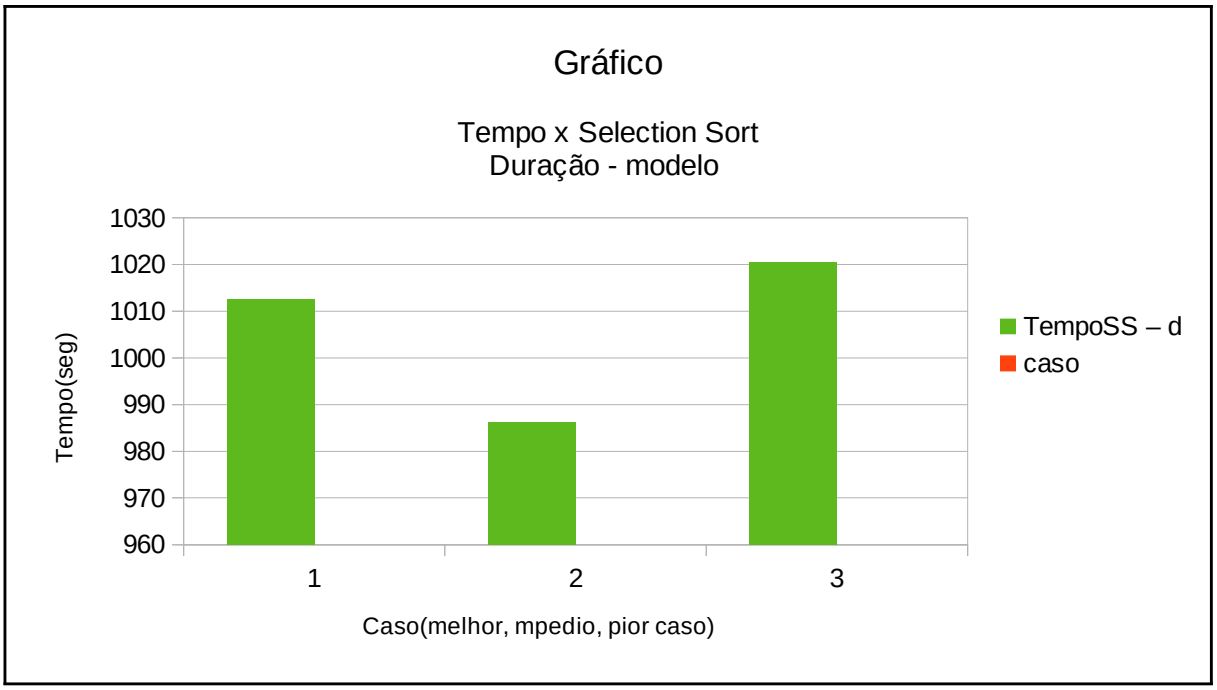


Gráfico
Tempo x Selection Sort
Duração - modelo
1030
1020
1010
1000
990
980
970
960
Tempo(seg)
1
2
3
Caso(melhor, mpedio, pior caso)
TempoSS – d
caso

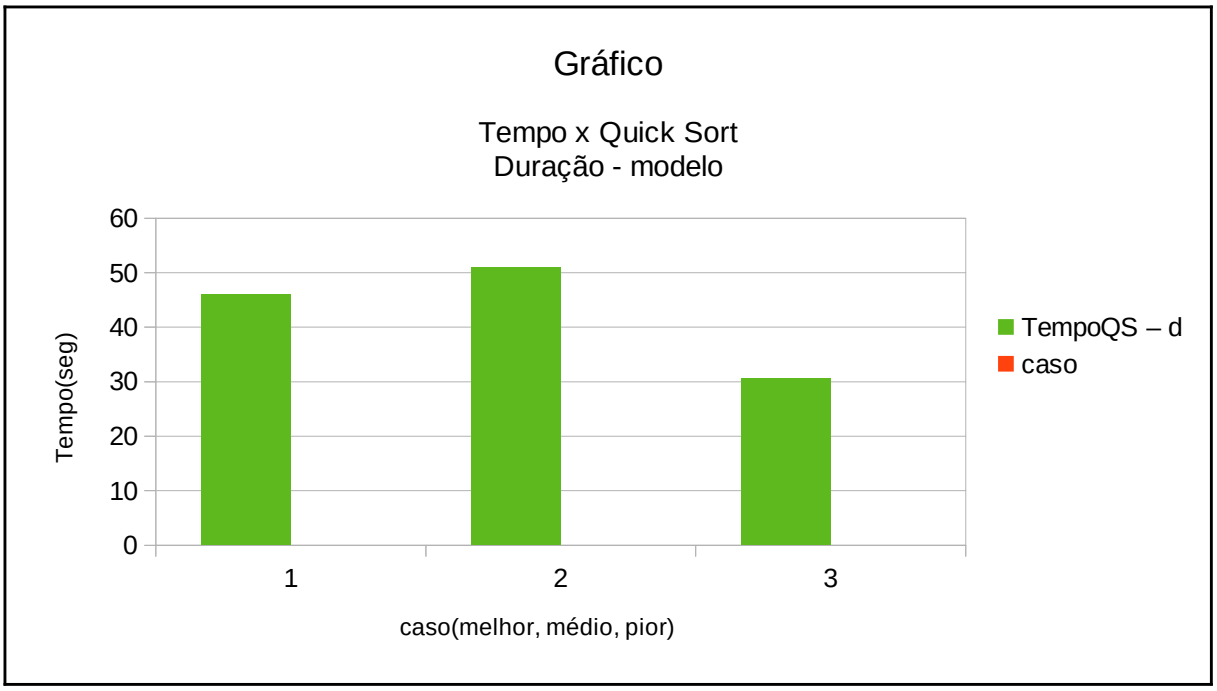


Gráfico
Tempo x Quick Sort
Duração - modelo
60
50
40
30
20
10
0
Tempo(seg)
1
2
3
caso(melhor, médio, pior)
TempoQS – d
caso

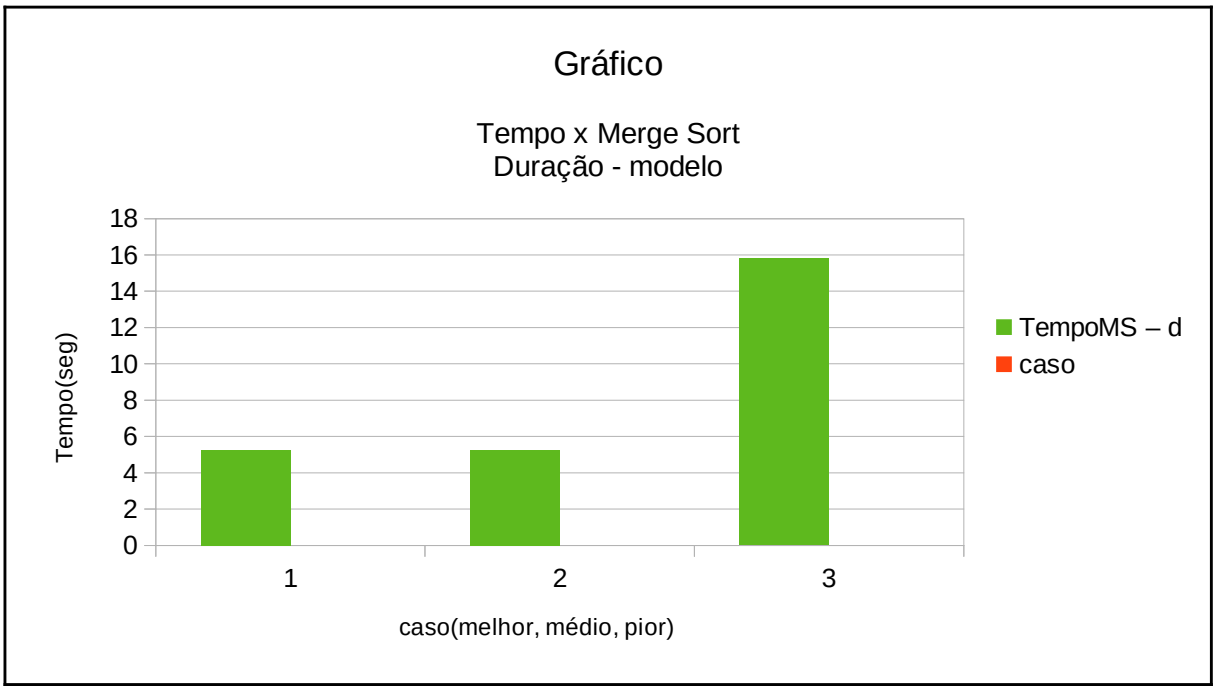


Gráfico
Tempo x Merge Sort
Duração - modelo
18
16
14
12
10
8
6
4
2
0
Tempo(seg)
1
2
3
caso(melhor, médio, pior)
TempoMS – d
caso

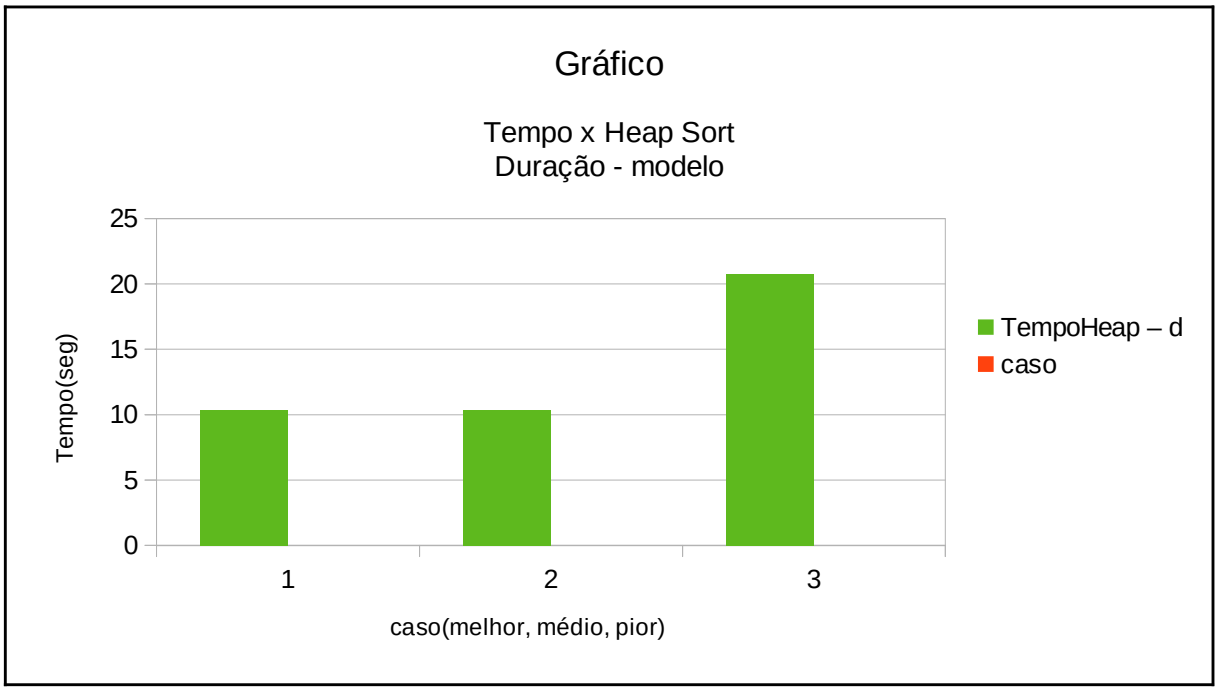


Gráfico
Tempo x Heap Sort
Duração - modelo
Tempo(seg)
25
20
15
10
5
0
1
2
3
caso(melhor, médio, pior)
TempoHeap – d
caso

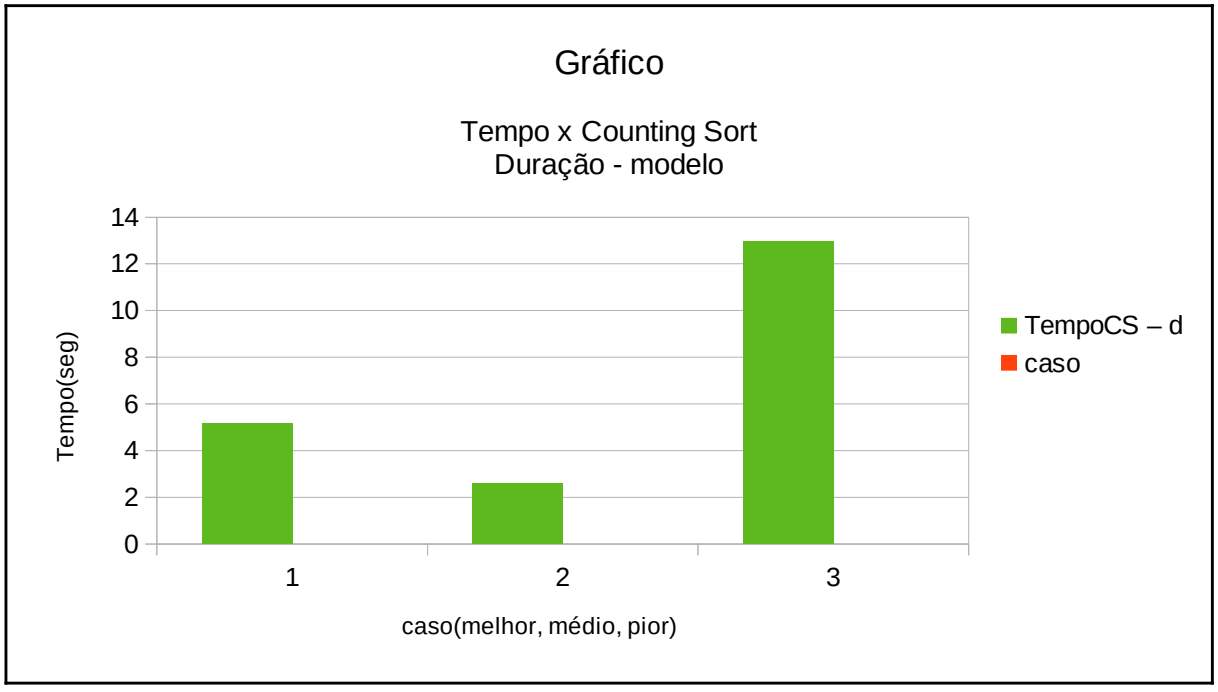
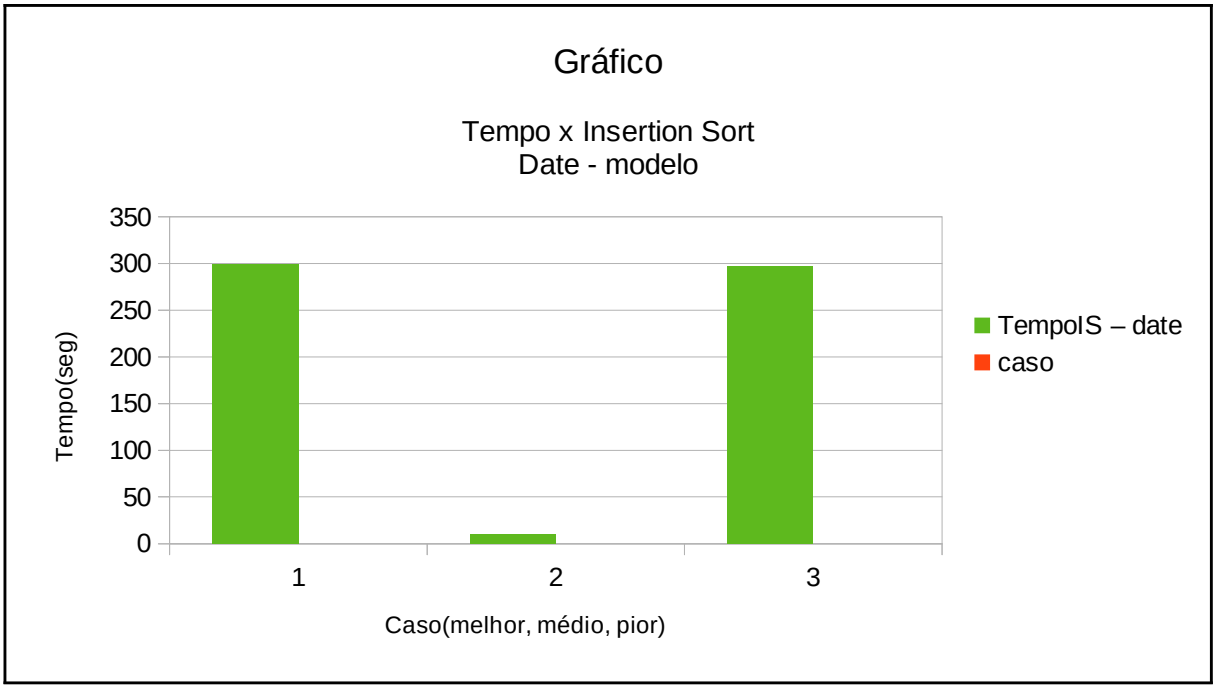


Gráfico
Tempo x Counting Sort
Duração - modelo
Tempo(seg)
14
12
10
8
6
4
2
0
1
2
3
caso(melhor, médio, pior)
TempoCS – d
caso

Ordenação Start_Date por algoritmos:



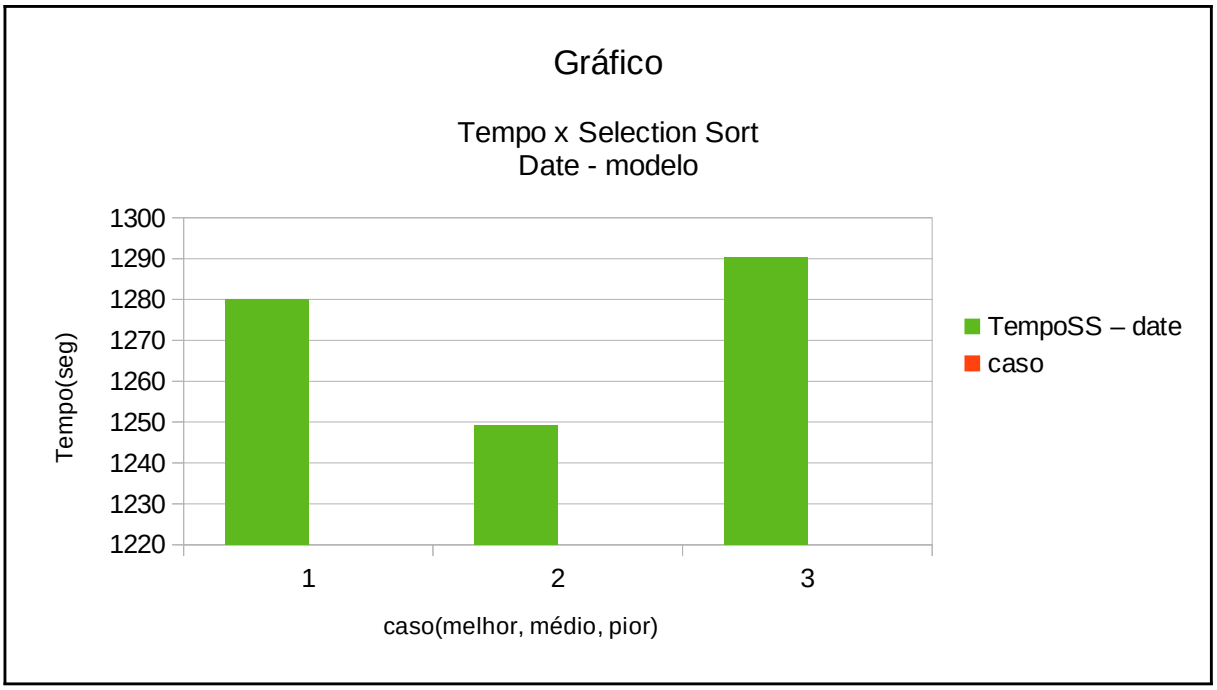


Gráfico
Tempo x Selection Sort
Date - modelo
1300
1290
1280
1270
1260
1250
1240
1230
1220
Tempo(seg)
1
2
3
caso(melhor, médio, pior)
TempoSS – date
caso

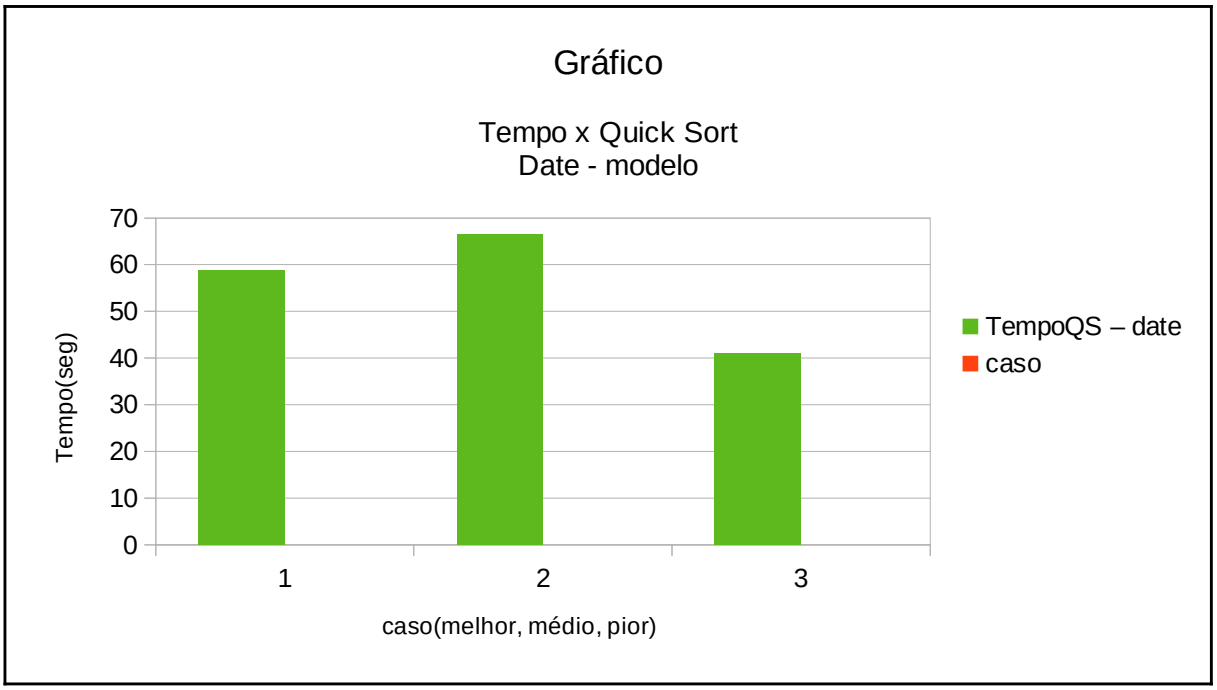


Gráfico
Tempo x Quick Sort
Date - modelo
70
60
50
40
30
20
10
0
Tempo(seg)
1
2
3
caso(melhor, médio, pior)
TempoQS – date
caso

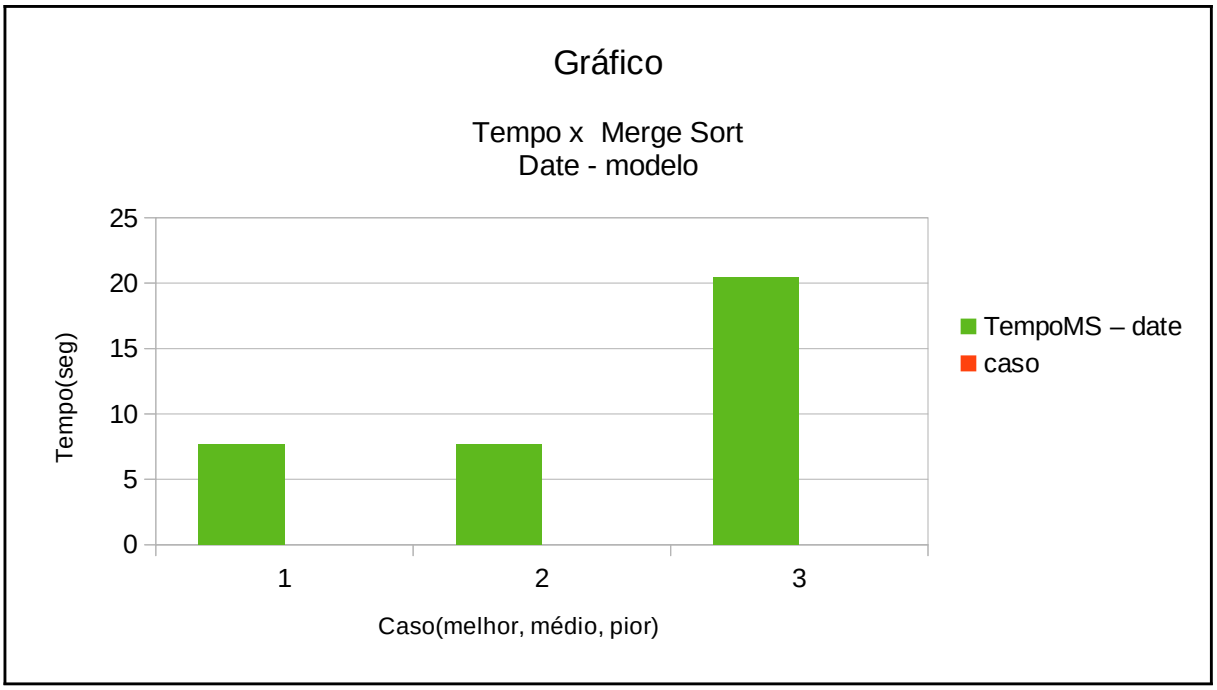


Gráfico
Tempo x Merge Sort
Date - modelo
25
20
15
10
5
0
Tempo(seg)
1
2
3
Caso(melhor, médio, pior)
TempoMS – date
caso

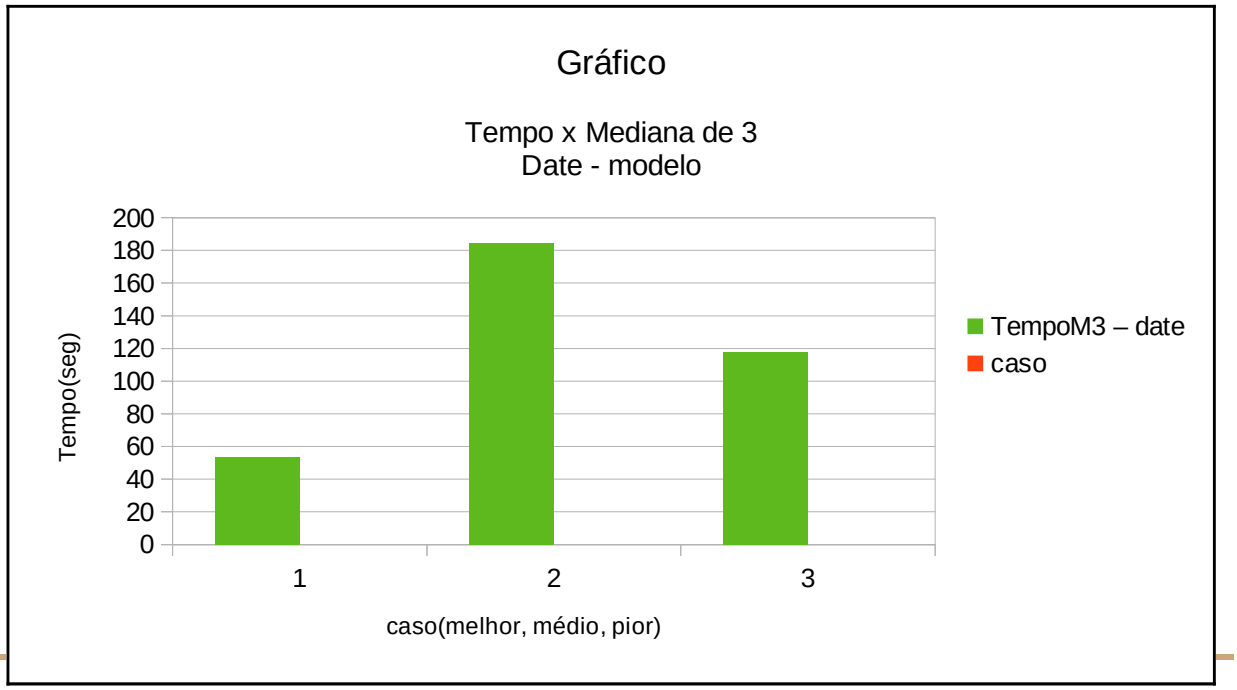
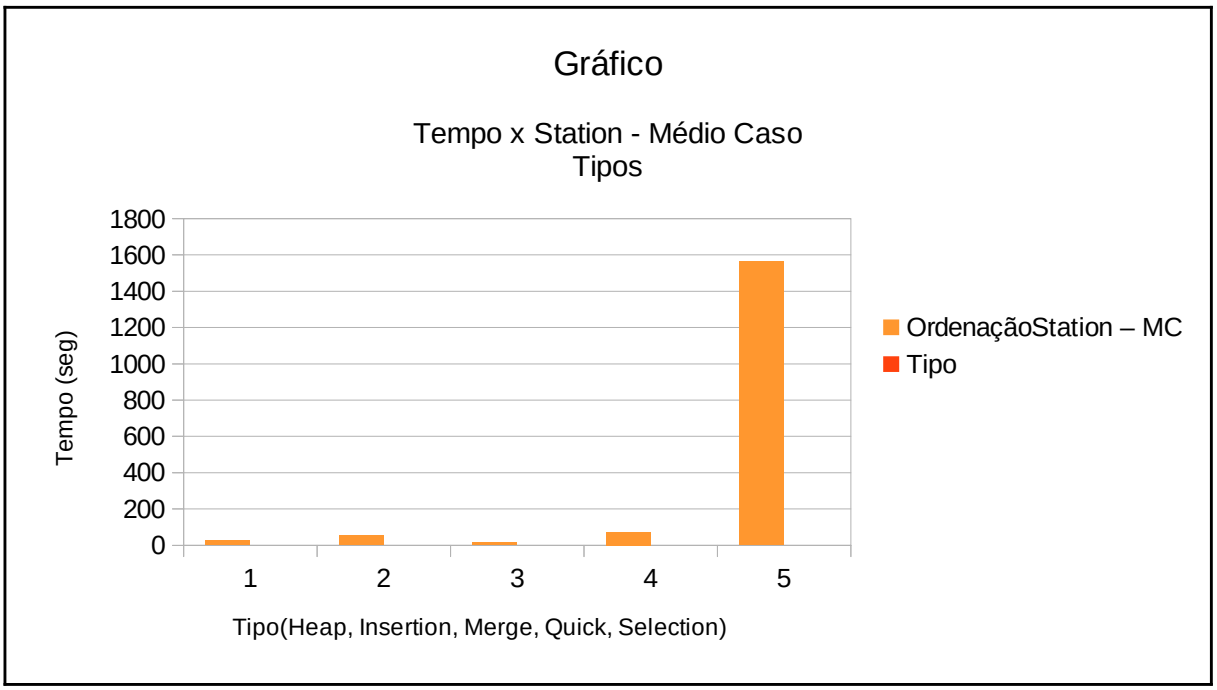


Gráfico
Tempo x Mediana de 3
Date - modelo
200
180
160
140
120
100
80
60
40
20
0
Tempo(seg)
1
2
3
caso(melhor, médio, pior)
TempoM3 – date
caso

Comparar ordenação x caso médio:



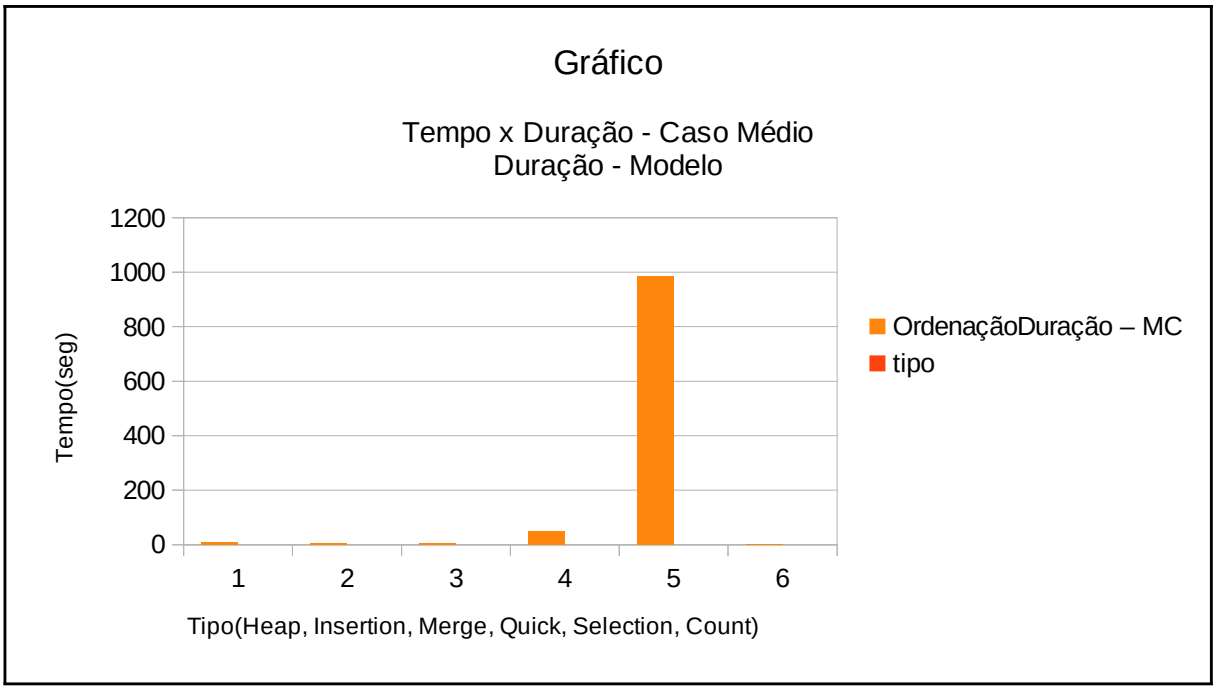


Gráfico
Tempo x Duração - Caso Médio
Duração - Modelo
Tempo(seg)
1200
1000
800
600
400
200
0
1
2
3
4
5
6
Tipo(Heap, Insertion, Merge, Quick, Selection, Count)
OrdenaçãoDuração – MC
tipo

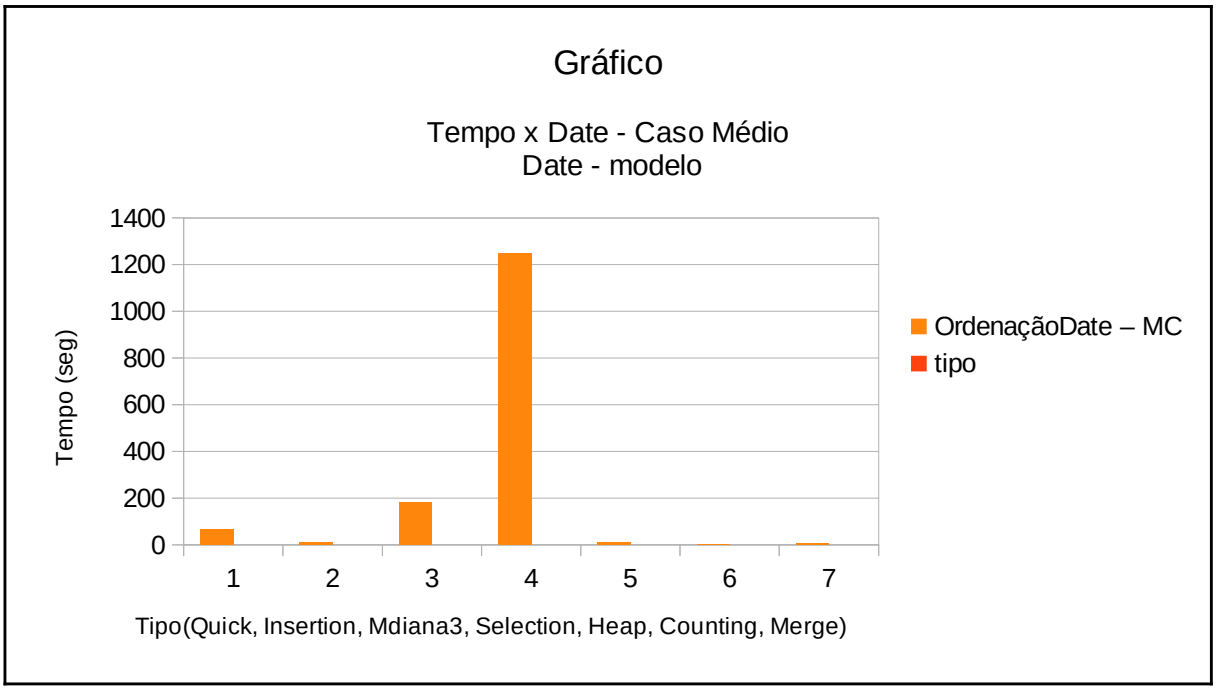


Gráfico
Tempo x Date - Caso Médio
Date - modelo
Tempo (seg)
1400
1200
1000
800
600
400
200
0
1
2
3
4
5
6
7
Tipo(Quick, Insertion, Mdiana3, Selection, Heap, Counting, Merge)
OrdenaçãoDate – MC
tipo

O algoritmo mais eficiente foi o Merge Sort porque foi o que apresentou melhor performance em cada caso (melhor, médio e pior caso) e que conseguiu se ajustar adequadamente aos diferentes modelos de ordenação sugeridos ou seja independente da disposição dos dados a ordenação é eficiente de maneria que ele se comporta como . $n * \log n$ nos 3 casos.