Laboratório de Estrutura de Dados

Primeira versão do projeto da disciplina

Comparação entre os algoritmos de ordenação elementar

Irlla Martins Barbosa da Silva

Isley Martins Barbosa da Silva

1. Introdução

Este relatório corresponde ao relato dos resultados obtidos no projeto das disciplinas de LEDA e EDA que tem o objetivo de programarmos o código e analisar o tempo do resultado.

O nosso projeto foi o Passwords e o dataset apresentado compreende uma listagem de mais de 650 mil senhas, as quais fizemos a classificação, a filtragem e a ordenação.

Para a classificação, foram consideradas entradas dos dados pelo arquivo intitulado passwords.csv.

As transformações de data e password classifier.csv consideram apenas as entradas do arquivo intitulado password classifier.csv.

As classificações foram feitas pela classe chamada "ClassificaSenha.java". A primeira linha automaticamente é escrita para o próximo arquivo, e logo em seguida, a senha é capturada por um método e avaliada. Assim, logo depois, o resultado é escrito no arquivo "password classifier.csv".

O password_format foi criado logo em seguida para formatar as datas que antes eram apresentados separado por "-" para "/".

Para agrupar os dados de cada linha, foi utilizado a estratégia de armazenar em um objeto, e logo depois, em uma lista de objetos.

Foram feitas ordenações de melhor, médio e pior caso com as seguintes ordenações: Selection Sort, Insertion Sort, Merge Sort, QuickSort, QuickSort com Mediana de 3 (intitulado de QuickSortMedian), HeapSort e Counting Sort(apenas possível para Length).

Os pontos mais relevantes foram a dificuldade dos algoritmos Quick Sort, Quick Sort Mediana de 3 e Merge Sort ordenarem data-mês por existirem mais dados iguais dificultando a agrupamento e causando "engarrafamento".

Na segunda seção vamos comentar o método escolhido para o teste com a descrição geral do ambiente e na terceira seção vai ser mostrados os resultados obtidos

2. Descrição geral sobre o método utilizado

Os testes foram feitos a partir de uma amostra de dados de 3000 linhas. Com base nesses dados, podemos perceber alguns resultados. Para contar o tempo dos métodos de ordenação, foi usado a biblioteca do Java TimeMillis() considerando apenas o tempo entre o começo e o fim de cada ordenação. Para melhor visualização, utilizamos o planilhas da google para gerar gráficos e tabelas com os resultados

Descrição geral do ambiente de testes

O teste foi feito em um computador Desktop com o processador AMD C-60 APU with Radeon(tm) HD Graphics 1.00 GHz, 4GB de RAM, com Sistema operacional de 32 bits, processador baseado em x64.

3. Resultados e Análise

Analisando os dados em relação a ordenação de Length, pode-se observar que entre os três pedidos de ordenação essa foi a mais rápida de ser executada pelos algoritmos por envolver variáveis com pouca ou quase nada de conversão.

Por causa,das ordenações de QuickSort e Quick mediana de 3 “engarrafar” em teste de valores muitos altos, analisaremos dados com 30 entradas e com 3000 entradas em relação ao tempo gasto.

Como mencionado anteriormente, os números não deram muito trabalho para ordenar e poderemos mostrar isso abaixo:

Tabela 1: Ordenação por Length(30 entradas)

Ordenação	Melhor	Medio caso	Pior caso
SelectionSort	0	0	0
InsertionSort	0	0	0
QuickSort	0	0	0
QuickSort Mediana 3	0	0	0
MergeSort	0	0	0
HeapSort	0	0	0
Counting	0	0	0

Não tendo muitos dados, foi feita uma segunda remessa de teste excluindo o quickSort e quickSort Mediana com 3000 entradas.

Tabela 2: Ordenação pelo Length (3000 entradas)

Ordenação	Melhor	Medio caso	Pior caso
SelectionSort	78	141	172
InsertionSort	0	78	140
QuickSort	-	-	-
QuickSort Mediana 3	-	-	-
MergeSort	0	8	2
HeapSort	3	16	0
Counting	0	16	0

Analisando o tempo de cada ordenação, o InsertionSort, MergerSort e CountingSort demoraram muito pouco comparado com o selection que lidera como o pior das ordenações para o caso mesmo na melhor condição de entrada.

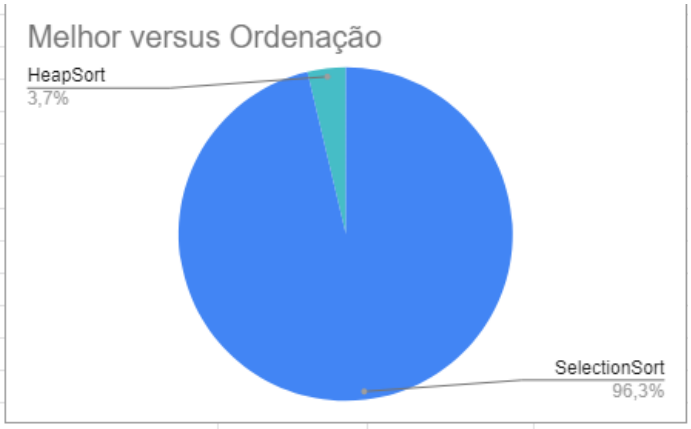


Figura 1: No melhor casos, HeapSort e SelectionSort foram os piores com 3000 entradas

Analisando a ordenação de caso médio; o MergerSort teve melhor desempenho. O Counting e HeapSort tiveram coincidentemente o mesmo desempenho. Novamente o SelectSort ficou como o último nessa análise.

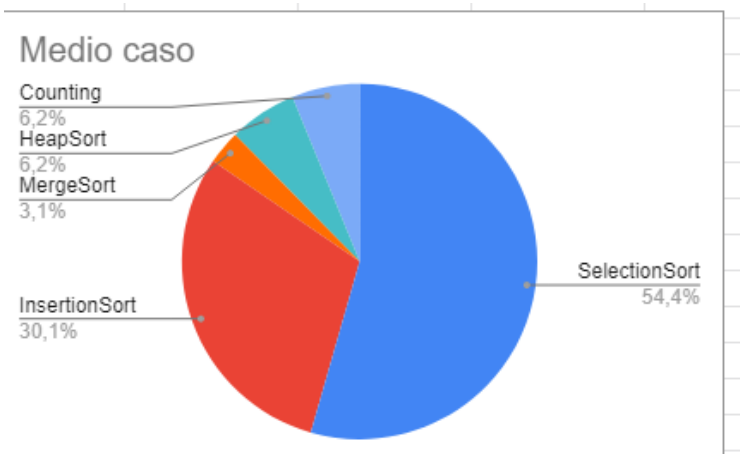


Figura 2: No caso médio, foram o SelectionSort e InsertionSort os mais demorados com 3000

Para o pior caso das ordenações, o HeapSort e Counting tiveram um ótimo desempenho. Observando o gráfico abaixo, o SelectSort novamente aparece como um algoritmo de ordenação custoso de tempo.

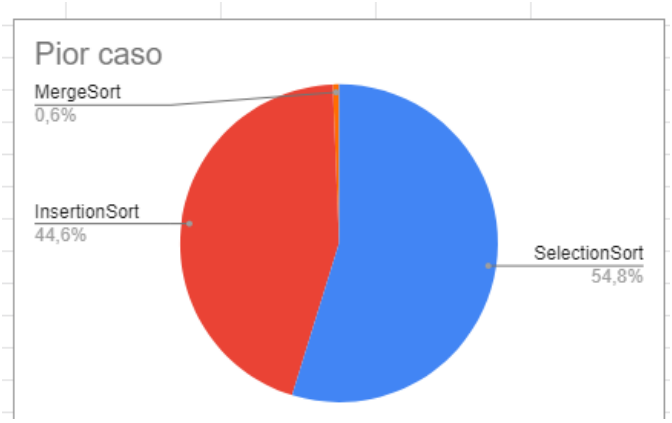


Figura 3: No pior caso, o SelectionSort e InsertionSort foram os mais demorados com 3000 entradas

Concluindo pelos TimeMillis(), o melhor algoritmo de ordenação para esse caso é o MergeSort por ser um algoritmo estável O(n*log*n) Abaixo tem o resumo de todos os dados coletados para a análise de Length:

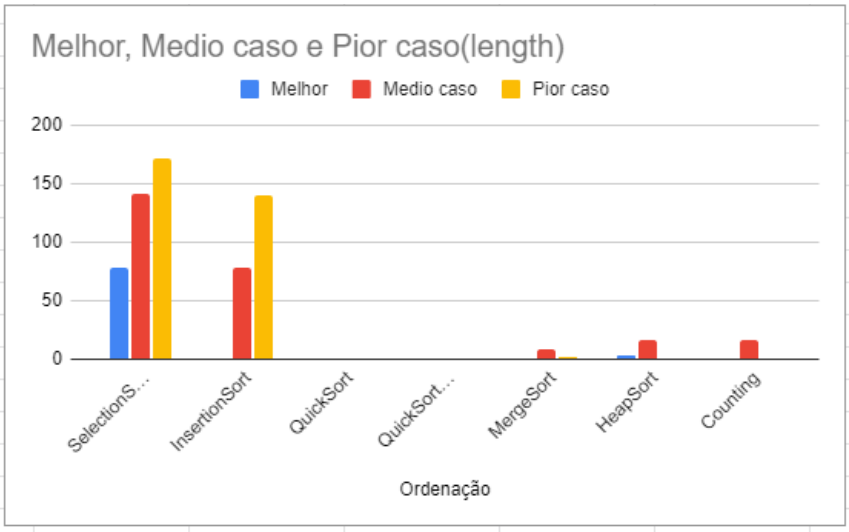


Figura 4: Análise de todos algoritmos exceto o QuickSort e QuickSort mediana de 3

Para os casos dos dados de mês-ano, pode-se observar que essa foi a mais trabalhosa de ordenada pelos algoritmos por envolver variáveis que precisam de conversão e se repetirem consideravelmente.

Tabela 3: Tempo de ordenação pelo mês-ano com 30 entradas

Ordenação	Melhor caso	Medio caso	Pior caso
SelectionSort	16	16	15
InsertionSort	47	16	15
QuickSort	62	16	94
QuickSort Media	0	16	78
MergeSort	12	31	32
HeapSort	15	16	63

O QuickMedian se destacou para essa análise com 30 variáveis para o melhor caso. O MergerSort ficou com 12 ms. Deste modo, o que mais se destacou nesse teste, olhando

pelo gráfico de pizza foram Insertion e QuickSort com valores próximos comparados às outras ordenações e com piores desempenhos mesmo nos melhores casos.

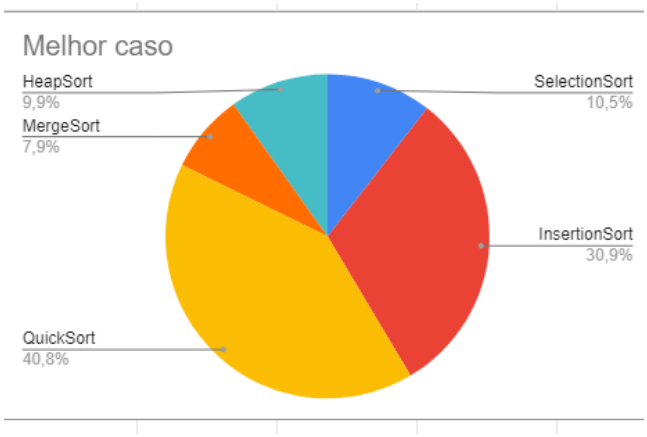


Figura 5: No caso do melhor caso, os que demoraram mais form o insertionSort e quickSort com as 30 entradas

Já observando o Médio caso, os algoritmos tiveram desempenho similares.

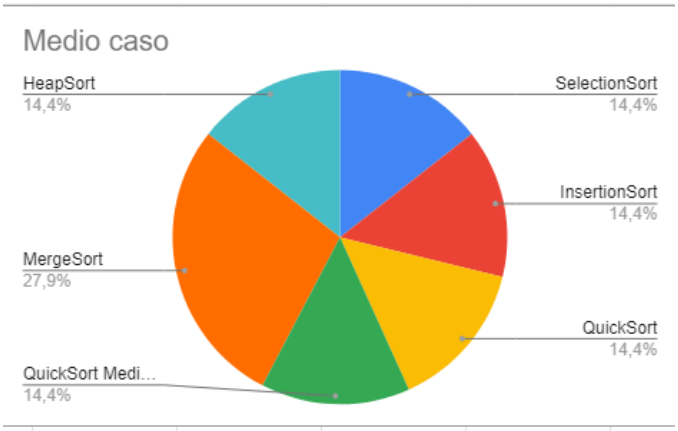


Figura 6: Médio caso, a ordenação mais demorada foi o MergeSort com 30 entradas

Seguindo os resultados para pior caso, o QuickSort demorou mais.

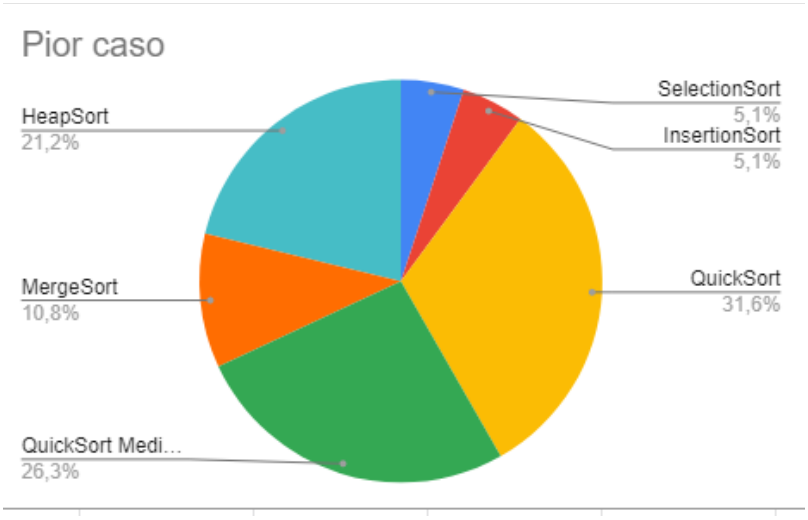


Figura 7: Médio caso, a ordenação mais demorada pelo mês-ano foi o MergeSort com 30 entradas

Os resultados da análise das ordenações do mês-ano com 30 entradas estão abaixo.

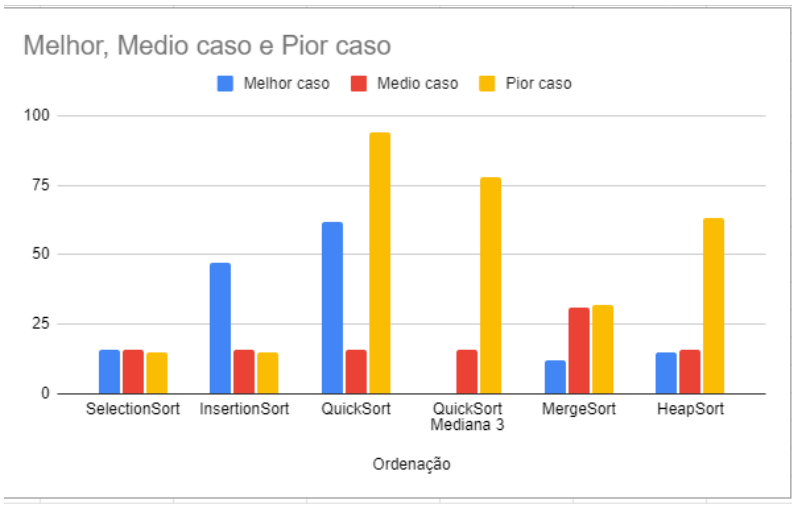


Figura 8: Representação dos casos com 30 entradas

Com 3000 entradas, fica claro o aumento exponencial que o insertion fez nessa comparação em relação ao pior caso. O resultado do tempo foi tão grande que quase não é possível ver graficamente as outras ordenações

Tabela 4: Tempo gasto gerado por 3000 entradas

Ordenação	Melhor caso	Medio caso	Pior caso
SelectionSort	16	15	16
InsertionSort	156	155414	334921
QuickSort	-	-	-
QuickSort Media	-	-	-
MergeSort	1895	1597	1776
HeapSort	921	1023	1214

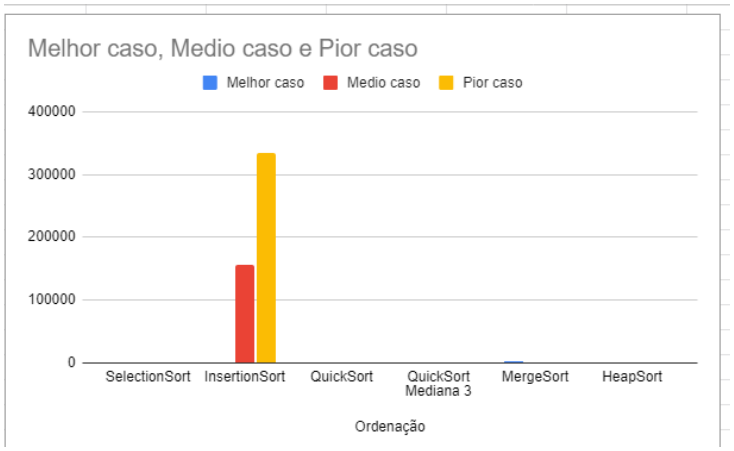


Figura 9: Representação das ordenações nos casos de 3000 entradas

Para 30 variáveis com QuickSort e QuickSortMedian, todas as ordenações foram feitas com o mesmo tempo para o melhor caso e o pior caso.

Tabela 5: Tempo gasto pelas ordenações de data nos casos de 30 entradas

Ordenação	Melhor	Medio caso	Pior caso
SelectionSort	0	0	0
InsertionSort	0	15	0
QuickSort	0	0	0
QuickSort Media	0	0	0
MergeSort	0	3	0
HeapSort	0	0	0

Na representação abaixo podemos notar que desta vez saiu mais demorado foi o insertionSort e o MergeSort no médio caso

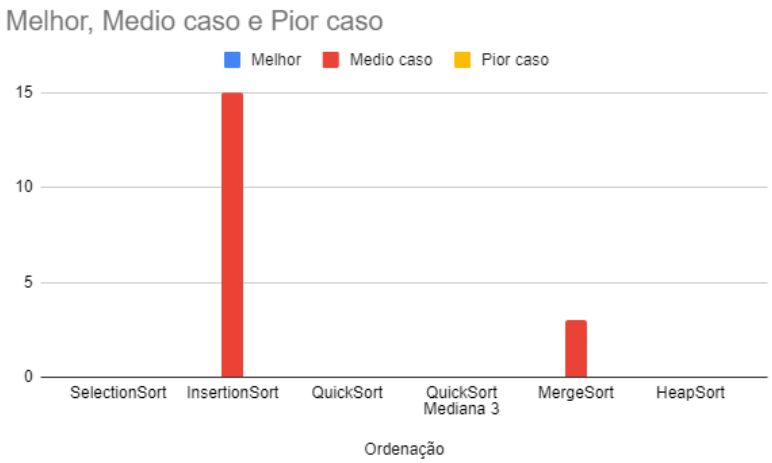


Figura 10: Representação gráfica da tabela 5

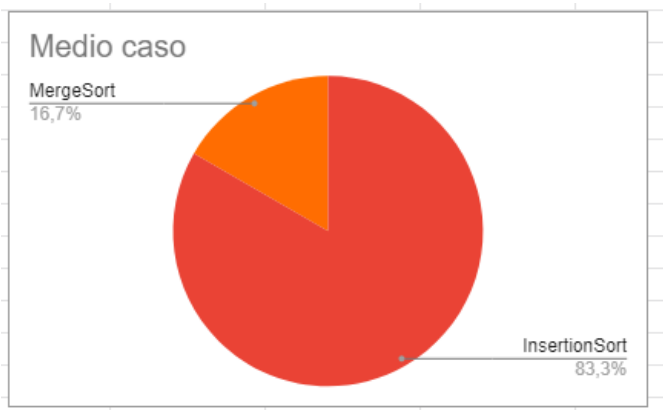


Figura 11: Representação gráfica de pizza em porcentagem

Analisando os dados com 3000 entradas, as ordenações geraram os seguintes tempo:

Tabela 6: representação do tempo gasto em milissegundos pela ordenação de data

Ordenação	Melhor	Medio caso	Pior caso
SelectionSort	0	0	0
InsertionSort	0	203	437
QuickSort	-	-	-
QuickSort Media	-	-	-
MergeSort	129	167	189
HeapSort	0	0	2

Desta vez, o insertionSort e o mergeSort foram os mais demorados, mas mesmo com esses dados, o MergeSort apresentou os tipos de casos mais ou menos semelhantes

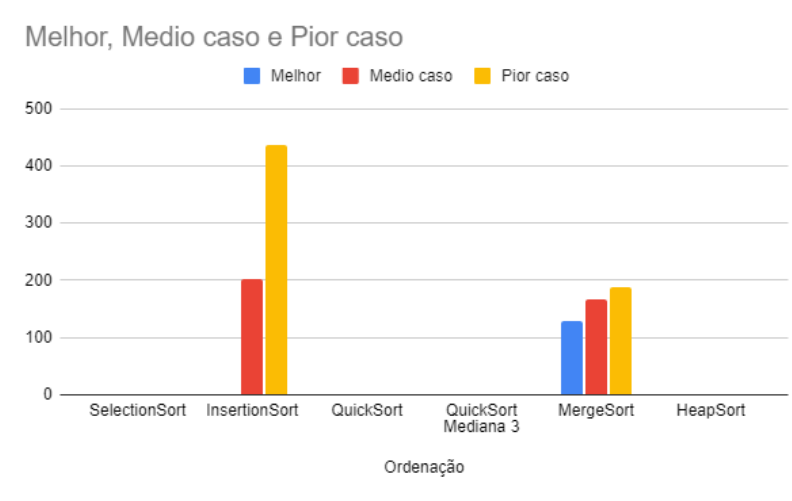


Figura 12: Representação do tempo gasto em milissegundos pela ordenação de data

Com isso, se colocarmos para ordenar todas as linhas do arquivo supondo com um certo comportamento teríamos os seguintes resultados:

Tabela 7: Representação dos cálculos de complexidade

Ordenação	Melhor caso	Medio caso	Pior caso
SelectionSort	O(n²)	O(n²)	O(n²)
InsertionSort	O(n)	O(n²)	O(n²)
QuickSort	O(n*log n)	O(n²)	O(n²)
QuickSort Mediana 3			
MergeSort	O(n*log n)	O(n*log n)	O(n*log n)
HeapSort	O(n*log n)	O(n*log n)	O(n*log n)
Counting	O(n+k)	O(n+k)	O(n+k)

Tabela 8: Supondo possível resultado com todas as entradas da planilha

Ordenação	Melhor caso	Medio caso	Pior caso
SelectionSort	448736534884	448736534884	448736534884
InsertionSort	669878	448736534884	448736534884
QuickSort	3902706,358	448736534884	448736534884
QuickSort Media	-	-	-
MergeSort	3902706,358	3902706,358	3902706,358
HeapSort	3902706,358	3902706,358	3902706,358
Counting	669878	669878	669878

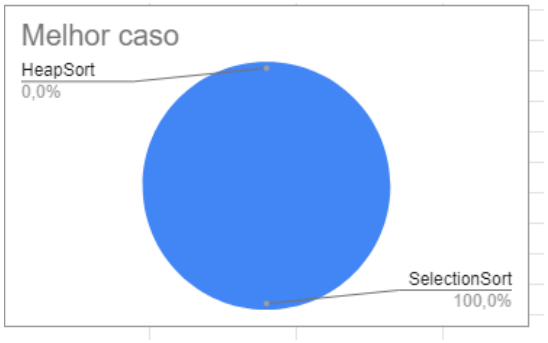


Figura 11: Representação gráfica de pizza no melhor caso

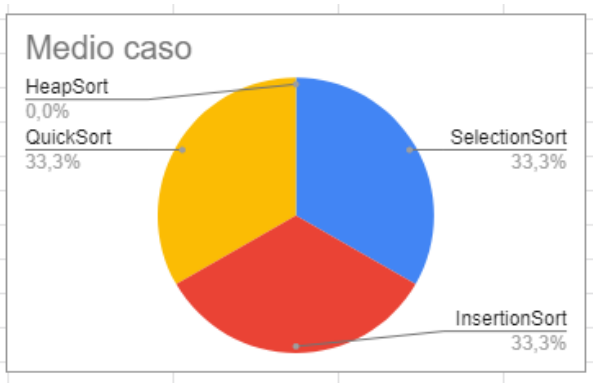


Figura 11: Representação gráfica de pizza caso médio

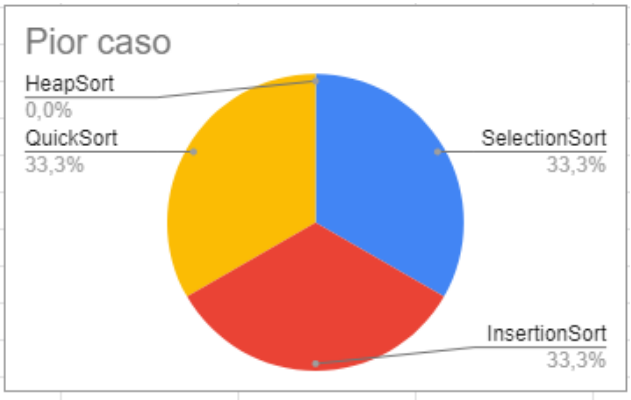
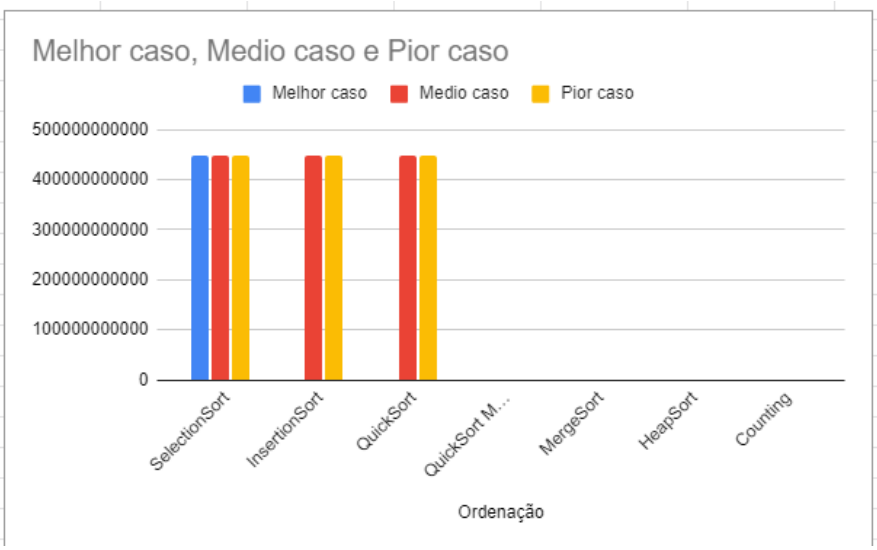


Figura 11: Representação gráfica de pizza no pior caso

Tabela 7: Representação dos cálculos de complexidade



Portanto, podemos concluir que não tem um algoritmo correto para todas as situações, pois nem sempre é necessário um algoritmo potente de ordenação para poucas variáveis de entrada. No entanto, para grandes volumes de dados é muito importante a análise das estruturas de ordenações uma vez que a área de tecnologia se expande cada vez mais gerando novos dados diariamente.

O SelectionSort e o InsertSort foram os que mais se destacam como os mais demorados nos testes. Já o HeapSort e CountingSort (ordenamento por tipo inteiro), se mostraram ótimos candidatos como os mais rápidos.