Laboratório de Estrutura de Dados

Primeira versão do projeto da disciplina

Comparação entre os algoritmos de ordenação elementar

Aluno:

Bruno da Silva Barbosa

1. Introdução

Este relatório corresponde ao relato dos resultados obtidos no projeto da disciplina de EDA e LEDA que tem como finalidade a implantação e análise de algoritmos de ordenação utilizando os dados da Google play store. Os algoritmos de ordenação usados foram os 7 estudados em sala de aula (Selection Sort, Insertion Sort, Merge Sort, Quicksort, QuickSort com Mediana de 3, Counting, e HeapSort).

Para executar o projeto, é necessário que seja feito o download do dataset da google play no kaggle e em seguida efetuar as transformações e salvar cada uma com seu nome específico.

A Partir dos resultados obtidos no final do projeto, foi possível analisar e perceber a diferença entre o algoritmos, onde alguns se comportam melhor do que outros. Alguns se comportaram de forma diferente do ideal por não conseguirem analisar os dados de forma eficiente e outros algoritmos desenvolveram um comportamento muito melhor por conseguirem analisar uma quantidade maior de dados de forma mais eficiente, sendo possível afirmar que algoritmos como o mergesort e heapsort possuem mais eficiência que o selection sort.

2. Descrição geral sobre o método utilizado

Para executar o teste no projeto foi feito a transformação indicada pelo professor onde teria que converter o formato da data para o padrão proposto e gerar um arquivo chamado "googleplaystore_date.csv" e em seguida filtrar o dataset pelo gênero e salvar com o nome de "googleplaystore_genres.csv" e salvar ambos na pasta onde se encontra o Main.java do projeto.

Os testes no projeto foram feitos utilizando o Visual Studio Code e ao executar o projeto é exibido um menu simples para que seja feita a escolha de um método de ordenação entre os 7 ou se deseja que todos os testes sejam feitos. Ao escolher um algoritmo para ser ordenado o sistema irá executar todo o processo de ordenação de acordo com o tipo de caso (bom, médio e ruim) e o campo a ser ordenado(App, Ratings, Installs e Last Update). Por fim ele imprimirá uma tabela com o tempo(ms) e gera um arquivo.csv ordenado e nomeado de acordo com o campo e caso.

Os gráficos inseridos foram feitos utilizando o google sheets juntamente com o google slides, já os gráficos de consumo consumo de memória foram capturados através do VisualVM.

Descrição geral do ambiente de testes

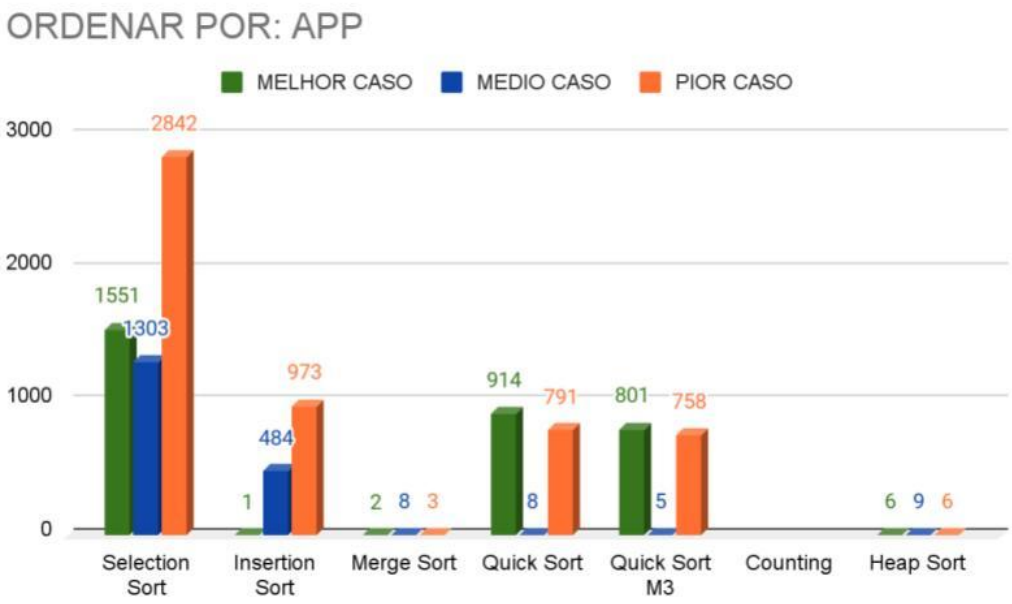
Para desenvolver e executar esse projeto, foi utilizada uma máquina com as seguintes especificações:

- Processador: AMD Ryzen 5 3600 6-Core Processor | 3.59 GHz
- Placa de vídeo: AMD Radeon RX 550
- Memoria ram: 16 GB , DDR4, 3000MHz;
- SSD: 512GB
- Sistema operacional: Windows 11

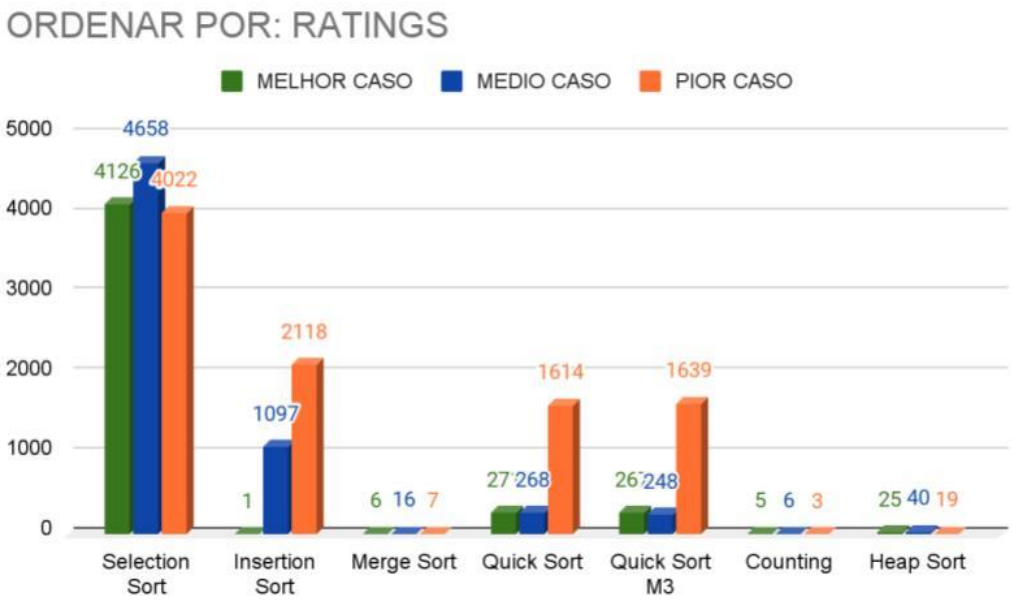
3. Resultados e Análise

Para demonstrar os resultados obtidos foram feitos 4 gráficos de acordo com o campo que foi ordenado (App, Ratings, Installs, Last Update), juntamente com uma tabela com todos os dados obtidos, exibindo o tempo de execução (em milissegundos) que foi obtido em cada ordenação e caso. Foi utilizado o Google Slides e o Google Sheets para criação dos gráficos e tabela.

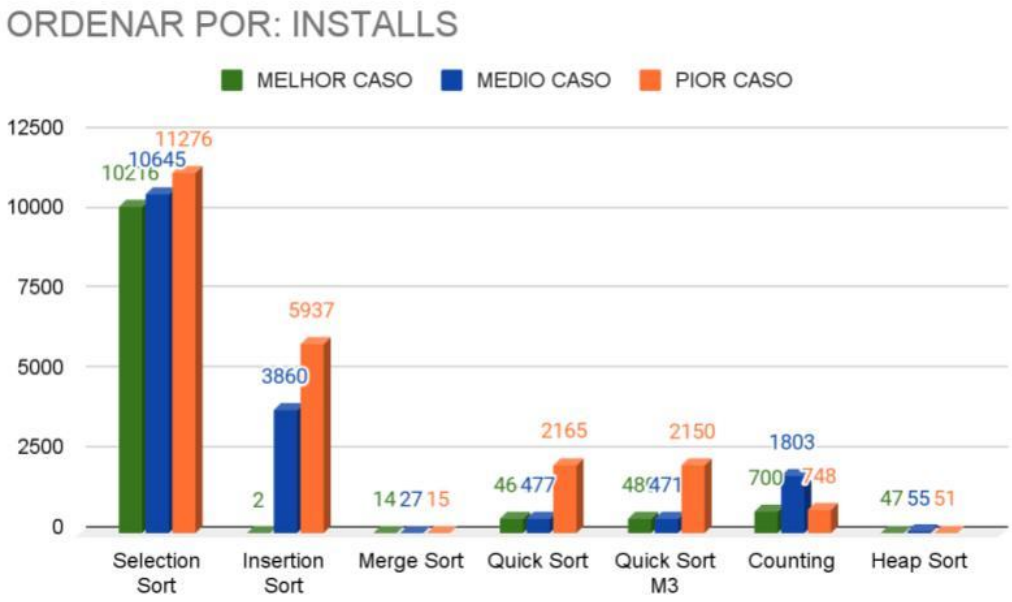
- Ordenação pelos nomes (APP)



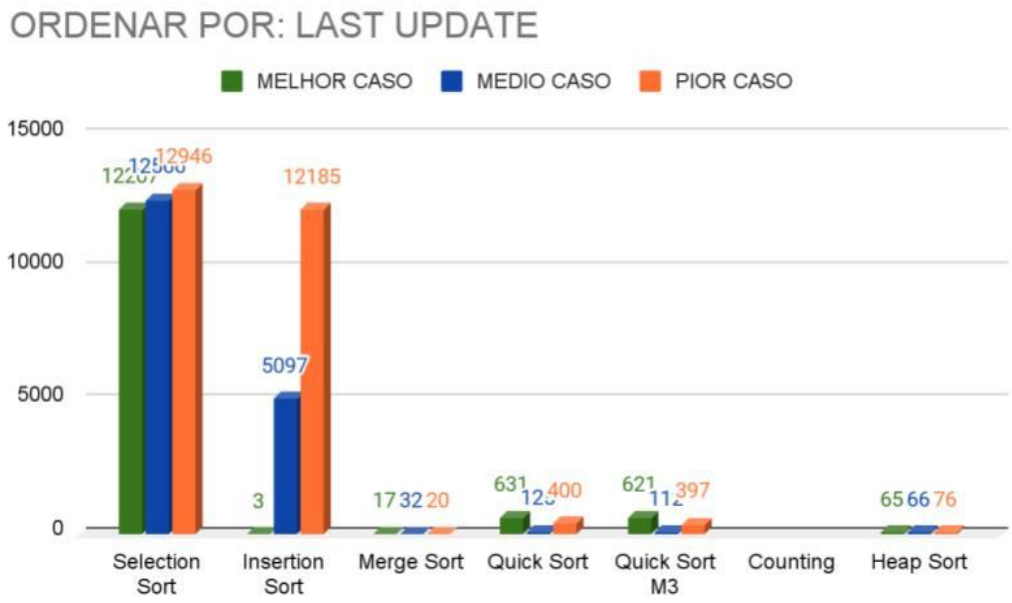
- Ordenação pelas avaliações (RATINGS)



- Ordenação pelas instalações (INSTALLS)

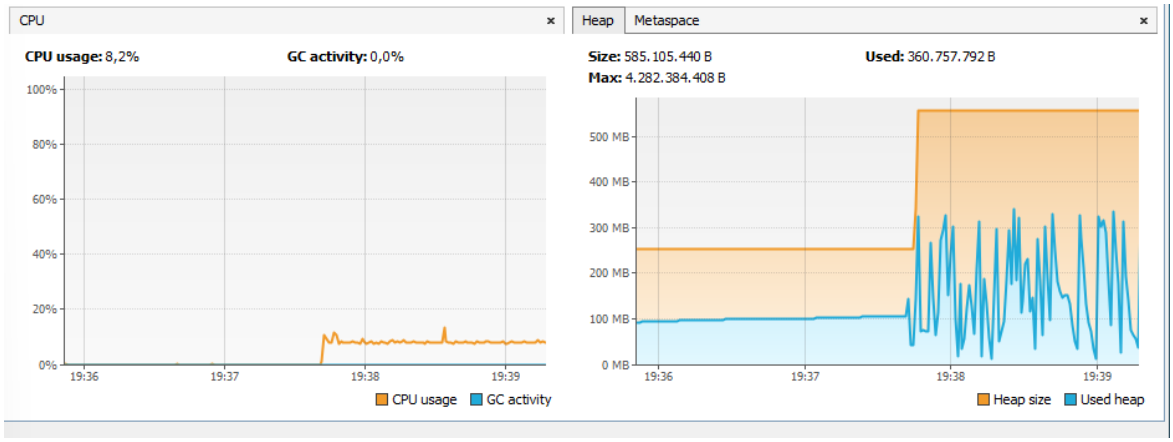


- Ordenação pela última atualização (LAST UPDATE)

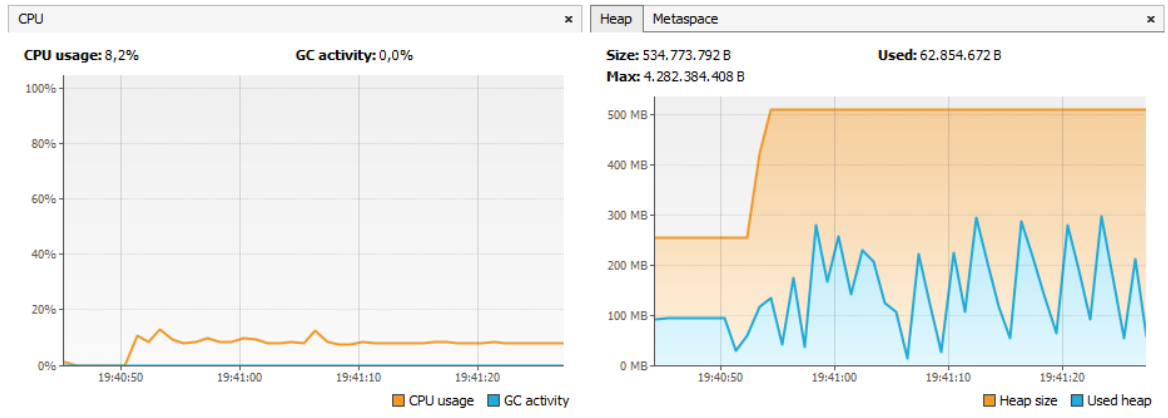


- Uso de CPU e memória

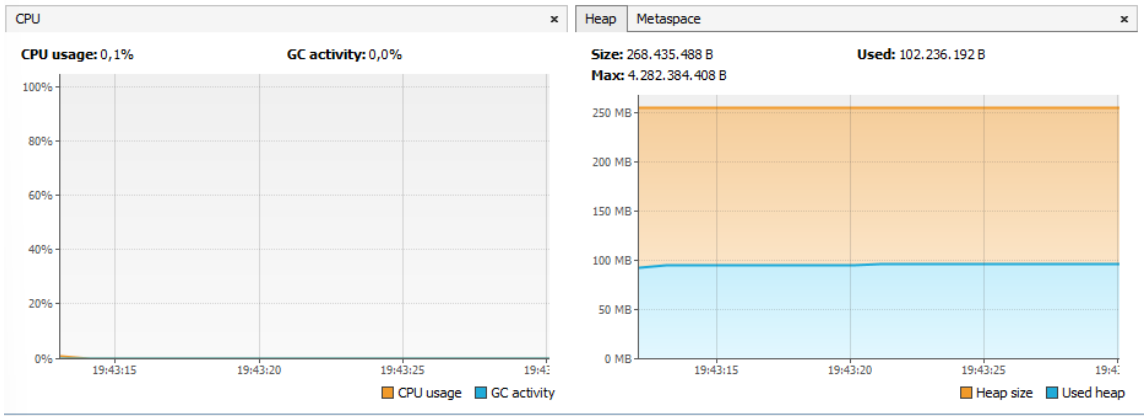
Selection Sort:



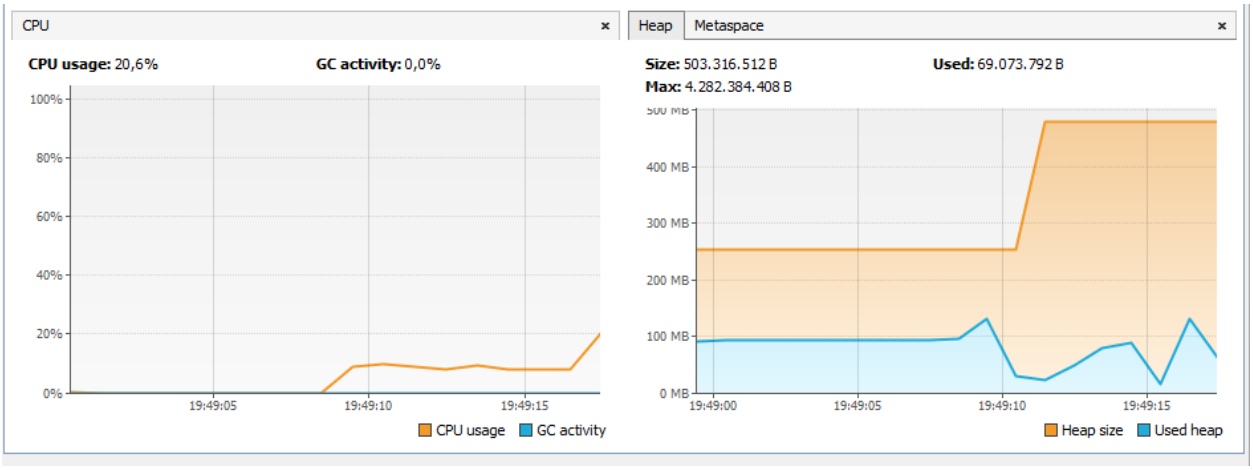
Insertion Sort



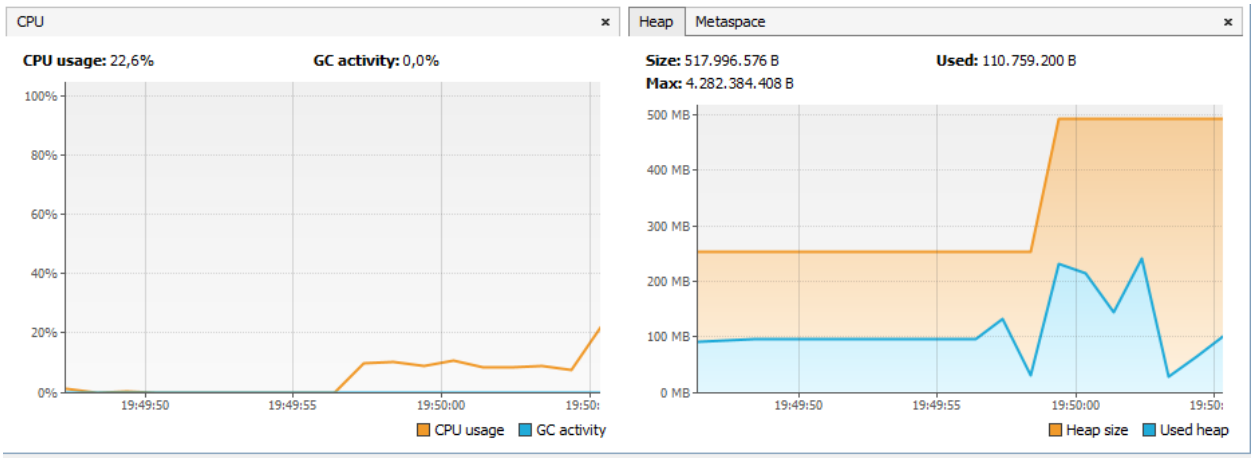
Merge Sort



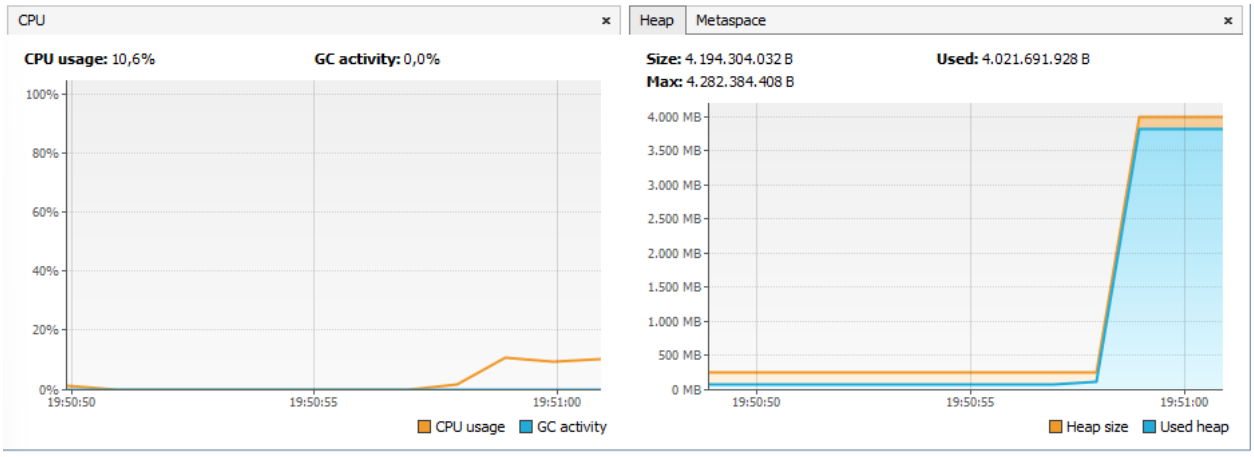
Quick Sort



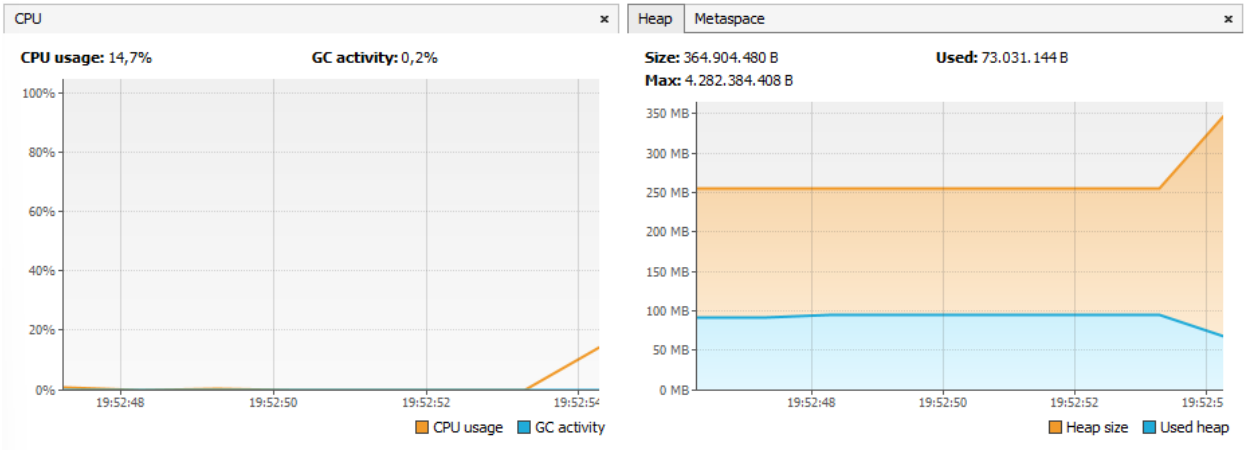
Quick Sort com mediana de 3



Counting Sort



Heap Sort


CPU
CPU usage: 14,7%
GC activity: 0,2%
100%
80%
60%
40%
20%
0%
19:52:48
19:52:50
19:52:52
CPU usage
GC activity
Heap
Metaspace
Size: 364.904.480 B
Used: 73.031.144 B
Max: 4.282.384.408 B
350 MB
300 MB
250 MB
200 MB
150 MB
100 MB
50 MB
0 MB
19:52:48
19:52:50
19:52:52
Heap size
Used heap

- Tabela com os tempos de execução obtidos na execução do projeto:

	Ordenação por Nome (APP)			Ordenação por Avaliações (RATINGS)		
	MELHOR CASO	MEDIO CASO	PIOR CASO	MELHOR CASO	MEDIO CASO	PIOR CASO
Selection Sort	1551 ms	1303 ms	2842 ms	4126 ms	4658 ms	4022 ms
Insertion Sort	1 ms	484 ms	973 ms	1 ms	1097 ms	2118 ms
Merge Sort	2 ms	8 ms	3 ms	6 ms	16 ms	7 ms
Quick Sort	914 ms	8 ms	791 ms	271 ms	268 ms	1614 ms
Quick Sort M3	801 ms	5 ms	758 ms	267 ms	248 ms	1639 ms
Counting	-	-	-	5 ms	6 ms	3 ms
Heap Sort	6 ms	9 ms	6 ms	25 ms	40 ms	19 ms

	Ordenação por Instalações (INSTALLS)			Ordenação por Ultima Atualização (LAST UPDATE)		
	MELHOR CASO	MEDIO CASO	PIOR CASO	MELHOR CASO	MEDIO CASO	PIOR CASO
Selection Sort	10216 ms	10645 ms	11276 ms	12207 ms	12566 ms	12946 ms
Insertion Sort	2 ms	3860 ms	5937 ms	3 ms	5097 ms	12185 ms
Merge Sort	14 ms	27 ms	15 ms	17 ms	32 ms	20 ms
Quick Sort	464 ms	477 ms	2165 ms	631 ms	125 ms	400 ms
Quick Sort M3	480 ms	471 ms	2150 ms	621 ms	112 ms	397 ms
Counting	700 ms	1803 ms	748 ms	-	-	-
Heap Sort	47 ms	55 ms	51 ms	65 ms	66 ms	76 ms

Análise dos dados obtidos

- **Análise do tempo de execução**

Ao analisar os gráficos de acordo com o caso, método de ordenação e campo ordenado foi possível visualizar que alguns algoritmos que exigem menos comparações como o mergesort e o heapsort que se comportaram de forma extremamente eficiente, ao ordenar valores em string ou numéricos em todos os casos. Por outro lado, alguns algoritmos não tiveram um desempenho tão bom como é o caso do Selection sort que se comportou mal em todos os casos. O quicksort e o quicksort com mediana de 3 se comportaram de forma satisfatória mas deixaram a desejar no tempo de ordenação do nome dos apps. O insertion sort teve um ótimo desempenho ao ordenar no melhor caso, já nos outros casos ele sofreu um aumento significativo no seu tempo de execução. O counting sort foi implementado apenas em dados que podiam ser considerados como inteiro, como é o caso das avaliações e número de instalações, mesmo assim foi possível obter um desempenho excelente, chegando a ser até melhor que o heapsort e o mergesort ao ordenar as avaliações.

- **Análise do uso de CPU e memória**

Com a análise dos gráficos de consumo e memória de cada ordenação contido no VisualVM é possível observar que o Mergesort e o Heapsort se saíram superiores aos outros, tanto no consumo de CPU quanto no de memória, já os outros algoritmos se saíram quase iguais com uma média de 8 a 10% de uso da CPU e um consumo de memória entre 100 e 400 MB, com exceção do counting sort que teve um tempo de execução muito eficiente, mas para isso acontecer foi necessário um alto consumo de memória, chegando a utilizar pouco mais de 3500 MB.

A Partir desses dados foi possível perceber o quão eficiente alguns algoritmos são em comparação a outros ao ordenar uma grande quantidade de dados, chegando até a ser espantoso a comparação, onde o mergesort ordenou o campo last update em apenas 20ms e o selection fez o mesmo trabalho em 12946 ms