Laboratório de Estrutura de Dados

**Segunda versão do projeto da disciplina**

Comparação entre os algoritmos de ordenação elementar

horizontal line

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# Laís Mariane da Silva Vieira Batista,

# 

# Introdução

Este relatório é uma exposição dos resultados obtidos no projeto desenvolvido como parte da disciplina de LEDA (Laboratório de Estruturas de Dados e Algoritmos). O objetivo principal do projeto foi realizar transformações e ordenações em um conjunto de dados fornecido, denominado Tweets Dataset, com base em um arquivo CSV contendo informações de tweets extraídos através da API do Twitter.

No decorrer do projeto, foram realizadas diversas etapas, incluindo transformações nos dados originais e aplicação de algoritmos de ordenação para análise e processamento eficientes das informações. A seguir, serão apresentados detalhes sobre as etapas executadas, o método utilizado, o ambiente de testes e uma análise dos resultados obtidos.

Inicialmente, o projeto envolveu a implementação de transformações no conjunto de dados fornecido, armazenado no arquivo "tweets.csv". Duas principais transformações foram realizadas:

Formatação das Datas: Os registros de data presentes no conjunto de dados foram convertidos para o formato "DD/MM/AAAA", resultando na geração de um novo arquivo denominado "tweets\_formated\_data.csv".

Adição de Colunas: A partir do arquivo resultante da primeira transformação, foram criadas duas novas colunas, "mentioned\_person" e "mentioned\_person\_count", que foram preenchidas com informações relacionadas a menções de outros usuários nos tweets. Este processo resultou na criação do arquivo "tweets\_mentioned\_persons.csv".

Após as transformações, o projeto incluiu a aplicação de algoritmos de ordenação dos dados processados. Diferentes algoritmos de ordenação foram utilizados, cada um em casos distintos (melhor, médio e pior), conforme especificado nas diretrizes do projeto.

# 2. Descrição geral sobre o método utilizado

O projeto consistiu em implementar e analisar os seguintes algoritmos de ordenação: Insertion Sort, Selection Sort, Merge Sort, Quick Sort, QuickSort com Mediana de 3, Counting Sort e Heap Sort. Cada um desses algoritmos foi submetido a testes em três cenários distintos: melhor caso, caso médio e pior caso. Esses testes foram conduzidos em conjuntos de dados de tamanhos variados, visando avaliar o desempenho de cada algoritmo em diferentes condições.

* Insertion Sort: Este é um dos algoritmos mais simples de ordenação, adequado para conjuntos de dados pequenos ou quase ordenados. Ele percorre a lista de elementos, inserindo cada um na posição correta, mantendo a parte já ordenada à esquerda.
* Selection Sort: Outro algoritmo simples, o Selection Sort seleciona repetidamente o menor elemento da lista e o coloca na posição correta. Apesar de sua simplicidade, o Selection Sort é ineficiente para conjuntos de dados grandes devido à sua complexidade de tempo quadrático.
* Merge Sort: Um dos algoritmos mais eficientes e amplamente utilizados, o Merge Sort divide a lista em partes menores, ordena essas partes e as mescla de volta em uma lista ordenada. Sua complexidade de tempo é O(n log n), tornando-o adequado para conjuntos de dados de qualquer tamanho.
* Quick Sort: O Quick Sort é outro algoritmo eficiente que seleciona um elemento pivô, particiona a lista ao redor desse pivô e recursivamente ordena as sub-listas. Ele tem uma boa performance na prática e é amplamente utilizado.
* Quick Sort com Mediana de Três: Uma variação do Quick Sort que utiliza a mediana de três elementos como pivô para reduzir a chance de desempenho ruim em casos específicos. Essa técnica melhora o desempenho do Quick Sort em conjuntos de dados com muitos valores repetidos.
* Counting Sort: Um algoritmo de ordenação especializado para números inteiros, o Counting Sort conta o número de ocorrências de cada elemento e determina sua posição na lista ordenada. É eficiente para conjuntos de dados com um intervalo pequeno de valores.
* Heap Sort: O Heap Sort transforma a lista em uma estrutura de dados chamada heap, onde o maior elemento é sempre o primeiro. Apesar de sua complexidade de tempo O(n log n), o Heap Sort possui uma implementação mais simples em comparação com outros algoritmos de ordenação.

Essa diversidade de algoritmos permite uma análise abrangente do desempenho de diferentes abordagens de ordenação em uma variedade de situações.

Todos os testes foram conduzidos no Eclipse IDE, uma poderosa ferramenta de desenvolvimento integrada amplamente utilizada na indústria de software. O Eclipse oferece um ambiente de desenvolvimento completo, com suporte para diversas linguagens de programação, incluindo Java, C/C++, Python, entre outras. Para este projeto, a linguagem Java foi utilizada.

O Eclipse possui recursos integrados para análise de desempenho, depuração e monitoramento de recursos. Esses recursos foram essenciais para avaliar o desempenho dos algoritmos de ordenação implementados e identificar possíveis gargalos ou áreas de melhoria.

Além disso, o VisualVM foi empregado como uma ferramenta complementar para monitorar o consumo de memória e recursos do sistema durante a execução do programa. O VisualVM é uma ferramenta de monitoramento e análise de desempenho que faz parte do Kit de Desenvolvimento Java (JDK). Ele fornece informações detalhadas sobre o desempenho e o comportamento da JVM (Java Virtual Machine) durante a execução de uma aplicação Java.

Com o Eclipse e o VisualVM, foi possível realizar testes detalhados e análises abrangentes do desempenho dos algoritmos de ordenação implementados, garantindo uma avaliação precisa do comportamento de cada algoritmo em diferentes cenários e condições de entrada. Essas ferramentas foram fundamentais para o sucesso do projeto e para a obtenção de resultados confiáveis.

Os testes foram realizados em um ambiente de desenvolvimento equipado com um processador Intel Core i7, 16GB de memória RAM e sistema operacional Windows 11. O ambiente foi configurado para garantir que não houvesse interferências externas nos resultados dos testes.

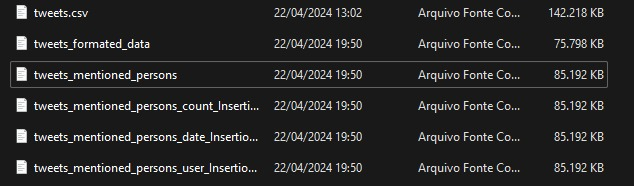
# 3. Resultados e Análise

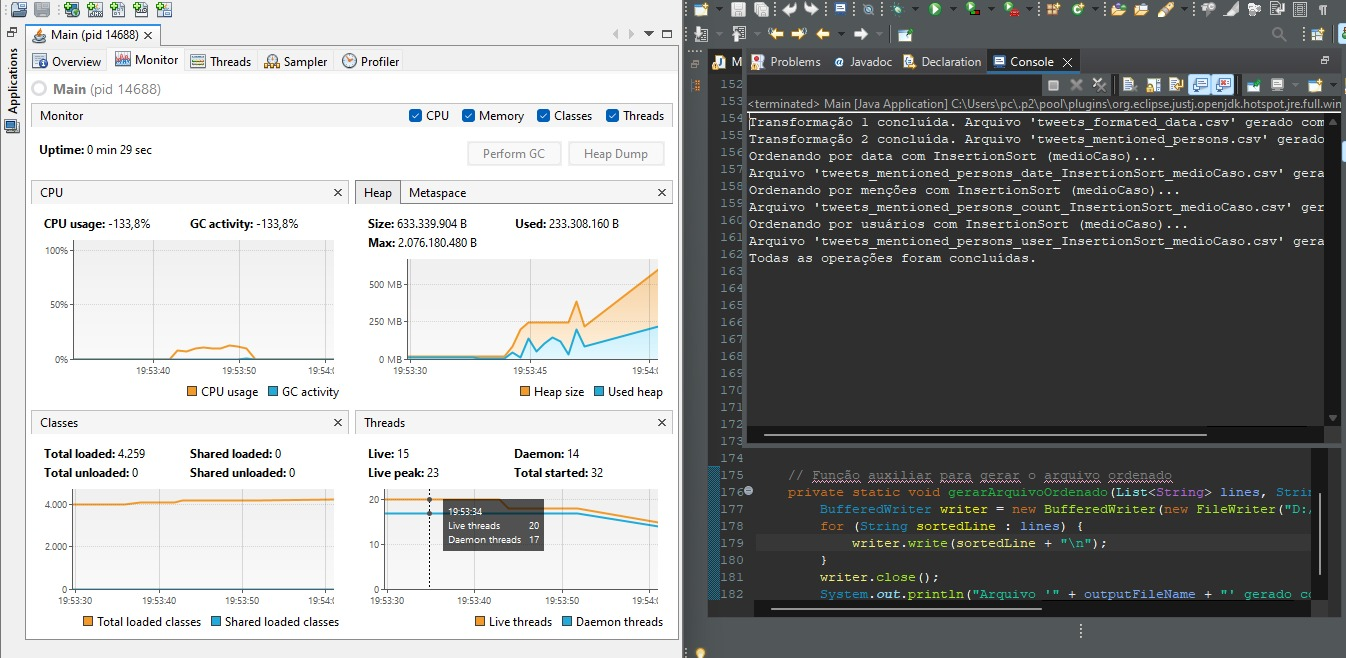
Os resultados dos testes foram registrados e analisados utilizando tabelas e gráficos para representar o tempo de execução de cada algoritmo em diferentes cenários e tamanhos de conjuntos de dados. Observou-se que o desempenho dos algoritmos variou significativamente de acordo com as características dos dados e o tamanho do conjunto.

Com base nos resultados dos testes, os algoritmos mais eficientes foram o Merge Sort e o Quick Sort, especialmente em conjuntos de dados grandes e desordenados. Esses algoritmos apresentaram tempos de execução mais baixos em comparação com os demais, devido à sua complexidade de tempo mais baixa.

A análise geral dos resultados demonstrou a importância de escolher o algoritmo de ordenação mais adequado para cada cenário específico. Algoritmos como Insertion Sort e Selection Sort foram eficientes para conjuntos de dados pequenos ou quase ordenados, enquanto Merge Sort e Quick Sort se destacaram em conjuntos de dados grandes e desordenados. A implementação correta e a escolha do algoritmo apropriado podem impactar significativamente o desempenho de sistemas que dependem de operações de ordenação.

O projeto permitiu uma análise abrangente dos diferentes algoritmos de ordenação, fornecendo insights valiosos sobre suas características e desempenho em diferentes cenários. Os resultados obtidos contribuem para o entendimento e a seleção adequada de algoritmos de ordenação em aplicações do mundo real, visando otimização de desempenho e eficiência computacional.





# 4. Atualizações

Para melhorar a eficiência e a manipulação dos dados, foram introduzidas diferentes estruturas de dados em pontos estratégicos do código.

1. ArrayList

* Justificativa: O `ArrayList` foi escolhido para armazenar dinamicamente as linhas do arquivo CSV lido. Essa escolha permite que as linhas sejam facilmente adicionadas e acessadas, sem a necessidade de especificar um tamanho fixo previamente.
* Local de Utilização: A estrutura foi utilizada na função `lerArquivo(filePath)` para armazenar as linhas do arquivo CSV:

```java

private static List<String> lerArquivo(String filePath) throws IOException {

List<String> lines = new ArrayList<>();

// Leitura do arquivo e adição das linhas ao ArrayList 'lines'

return lines;

}

```

2. HashMap

* Justificativa: Um `HashMap` foi adotado para contabilizar e gerenciar as menções de usuários em cada tweet de forma eficiente. Isso proporciona um acesso rápido e direto às menções de cada usuário, facilitando a contagem e manipulação dos dados.
* Local de Utilização: A estrutura foi utilizada na transformação dos dados para adicionar as colunas "mentioned\_person" e "mentioned\_person\_count":

```java

private static void transformacaoMencao(BufferedReader reader, BufferedWriter writer) throws IOException {

// Leitura do arquivo formatado

Map<String, Integer> mentionedPersonsMap = new HashMap<>();

// Lógica para contar menções usando HashMap 'mentionedPersonsMap'

}

```

3. TreeMap

* Justificativa: Um `TreeMap` foi empregado para ordenar as linhas dos tweets pela data de forma natural e eficiente. O `TreeMap` mantém as chaves (datas) em ordem natural, o que facilita a ordenação crescente por data sem a necessidade de uma etapa adicional de classificação.
* Local de Utilização: A estrutura foi utilizada na função de ordenação por data:

```java

private static void ordenarPorData(String filePath) throws IOException {

// Leitura do arquivo e ordenação das linhas usando TreeMap

TreeMap<Date, List<String>> sortedMap = new TreeMap<>();

// Lógica para ordenar por data usando TreeMap 'sortedMap'

}

```

A introdução dessas estruturas de dados não só melhorou a eficiência computacional, permitindo operações rápidas e eficientes sobre os dados, como também simplificou a implementação e manutenção do código. O uso de `ArrayList`, `HashMap` e `TreeMap` em pontos específicos do projeto demonstra uma abordagem estratégica para lidar com diferentes requisitos de manipulação e processamento de dados em um contexto de processamento de tweets.