# **Relatório – Implementação da Classe Matriz e Algoritmos de Ordenação**

## **1. Descrição da classe Matriz**

A classe Matriz foi projetada para representar uma matriz bidimensional de inteiros, com dimensões definidas pelo usuário.

### **Atributos**

* linhas (int): quantidade de linhas da matriz.
* colunas (int): quantidade de colunas da matriz.
* dados (int[][]): estrutura que armazena os elementos da matriz.

### **Métodos principais**

* **Construtor Matriz(int linhas, int colunas)** → cria uma matriz vazia (inicializada com zeros).
* **preencherManual(Scanner sc)** → permite ao usuário digitar os valores da matriz, elemento por elemento.
* **preencherAutomatico(int min, int max, Random rng)** → preenche a matriz com valores aleatórios no intervalo [min, max].
* **remover(int linha, int coluna)** → substitui o valor em uma posição específica por 0 (marcando como removido).
* **exibir()** → mostra a matriz no formato tabular, alinhando as colunas para melhor visualização.
* **Ordenação**:  
  + ordenarPorLinhas(boolean crescente) → aplica **Bubble Sort** em cada linha.
  + ordenarPorColunas(boolean crescente) → aplica **Merge Sort** em cada coluna.
  + ordenarMatrizCompleta(boolean crescente) → transforma a matriz em vetor, ordena com **Merge Sort** e reconstrói a matriz.

### **Responsabilidades**

A classe Matriz centraliza todas as operações de:

* armazenamento de dados,
* preenchimento,
* manipulação (remoção),
* visualização (exibição tabular),
* ordenação (diferentes escopos e algoritmos).

## **2. Explicação da lógica de cada funcionalidade**

* **Preenchimento manual**: percorre linha a linha, coluna a coluna, pedindo ao usuário os valores.
* **Preenchimento automático**: usa a classe Random para gerar números entre min e max.
* **Remoção**: acessa a posição [linha][coluna] e substitui por 0.
* **Exibição**: percorre toda a matriz e imprime cada valor com formatação (printf), garantindo que as colunas fiquem alinhadas.
* **Ordenação por linhas**: para cada linha da matriz, aplica o algoritmo **Bubble Sort**.
* **Ordenação por colunas**: extrai cada coluna para um vetor temporário, aplica **Merge Sort** e insere os valores de volta na matriz.
* **Ordenação completa**: “achata” a matriz em um vetor unidimensional, ordena com **Merge Sort**, e reconstrói a matriz em duas dimensões.

## **3. Comparação entre os algoritmos de ordenação**

### **Bubble Sort**

* **Abordagem**: iterativo, compara elementos adjacentes e troca até que todos estejam ordenados.
* **Complexidade**:  
  + Melhor caso: O(n) (quando já está ordenado).
  + Caso médio e pior caso: O(n²).
* **Vantagens**: implementação simples e didática.
* **Desvantagens**: pouco eficiente em matrizes grandes.
* **Uso no projeto**: aplicado em **cada linha** da matriz (normalmente pequena em tamanho).

### **Merge Sort**

* **Abordagem**: recursivo, divide o vetor em duas metades, ordena recursivamente e depois intercala.
* **Complexidade**:  
  + Melhor, médio e pior caso: O(n log n).
* **Vantagens**: desempenho consistente e eficiente em vetores grandes.
* **Desvantagens**: requer memória extra para os vetores auxiliares.
* **Uso no projeto**: aplicado em **colunas** e na **matriz completa** (achatada).

### **Diferenças práticas**

* O **Bubble Sort** é mais indicado quando os vetores são pequenos ou quase ordenados, pois é simples e pode parar cedo.
* O **Merge Sort** é mais indicado para matrizes maiores ou completamente desordenadas, já que garante bom desempenho.

## **4. Conclusões**

**Dificuldades**:

* + Eu tive dificuldades em controlar índices ao ordenar por colunas (extração e reinserção).
  + Implementar corretamente o mergeSort recursivo, garantindo que a ordenação fosse estável e considerando a ordem crescente/decrescente.
  + Tentei evitar que a matriz aparecesse apenas com zeros quando não fosse preenchida.

**Aprendizados**:

* + Reforcei a importância de **estruturas de dados estáticas** (vetores e matrizes) e como manipulá-las.
  + Entendi melhor as diferenças entre algoritmos iterativos (Bubble) e recursivos (Merge), tanto na lógica quanto na eficiência.
  + Percebi a necessidade de organizar o código em **classes separadas**, o que torna o sistema mais modular e fácil de manter.

**Reflexão final**:  
O projeto mostrou na prática como a escolha do algoritmo de ordenação impacta no desempenho e como organizar classes em Java ajuda a estruturar bem as responsabilidades, quanto em organizar o código, deixando ele fácil de entender e manipular ele.

Eu achei melhor fazer um menu, para que eu pudesse fazer as alterações nas matrizes e vizulizá-las da forma que foi proposta.

Para visualizar as matrizes:

MENU MATRIZ

1 - Criar nova matriz

Escolha: 1

Linhas: 3

Colunas: 3

Matriz criada com sucesso!

MENU MATRIZ

3 - Preencher automaticamente

Escolha: 3

Valor mínimo: 1

Valor máximo: 20

Matriz preenchida automaticamente!

MENU MATRIZ

4 - Exibir matriz

Escolha: 4

Matriz atual:

[ 7 2 5 ]

[ 10 3 1 ]

[ 18 6 4 ]