

UNIVERSIDADE PAULISTA - UNIP
INSTITUTO DE CIÊNCIAS EXATAS E TECNOLOGIA

Gabriela Araújo C. da Silva - N8397D0

**DESENVOLVIMENTO DE SISTEMA PARA ANÁLISE DE
PERFORMANCE DE ALGORITMOS DE ORDENAÇÃO DE
DADOS (Estrutura de Dados)**

Goiânia - GO
17 de novembro de 2023

Sumário

1	Objetivo do Trabalho	3
2	Introdução	4
2.1	A Era Digital e a Importância dos dados	4
2.2	O Projeto e seus Objetivos	4
2.3	Tecnologias Utilizadas	4
2.4	Funcionalidades do Sistema	4
2.4.1	Análise de Desempenho	5
2.4.2	CRUD	5
2.4.3	Visualização de Imagens	5
2.5	Conclusão	5
3	Referencial Teórico	6
3.1	Ordenações	6
3.1.1	ref 1: Algoritmo de Ordenação Merge Sort	6
3.1.2	ref 2: Algoritmo de Ordenação Quick Sort	6
3.1.3	ref 3: Algoritmo de Ordenação Heap Sort	6
3.2	Tecnologias	7
3.2.1	ref 4: Linguagem de Programação Java	7
3.2.2	ref 5: SGBD MySQL	7
4	Desenvolvimento:	8
4.1	Introdução	8
4.2	Criação do Banco de Dados	8
4.3	População do Banco de dados	9
4.4	Criação da Aplicação Java	11
4.5	Implementação dos Algoritmos de Ordenação	12
4.5.1	Merge Sort	12
4.5.2	Quick Sort	13
4.5.3	Heap Sort	14
4.6	Medição do Tempo de Ordenação	16
4.7	Melhorias na Interface do Usuário	17
4.7.1	Tela	18
4.7.2	Tela Tempo Decorrido	19
4.7.3	Tela DoubleClick na Imagem	19
4.8	Comparativo Tempo Decorrido	20

4.8.1 Merge	20
4.8.2 Quick	20
4.8.3 Heap	20
5 Resultados:	21
5.1 Graficos	21
5.2 Discussão	28
6 Considerações Finais	30
6.1 Considerações gerais	30
7 Código fonte:	31
7.1 MySQL	31
7.2 Java	31
7.3 Python	58
8 Bibliografia	61

1 Objetivo do Trabalho

Neste trabalho, nosso principal objetivo é desenvolver um sistema que possa avaliar o desempenho de diferentes algoritmos de ordenação de dados. Em meio à era digital, estamos constantemente cercados por uma quantidade enorme de dados. Esses dados, se organizados e analisados corretamente, podem fornecer insights valiosos e auxiliar na tomada de decisões informadas. No entanto, a tarefa de organizar esses dados pode ser bastante desafiadora, especialmente quando se trata de grandes volumes de dados. É aqui que os algoritmos de ordenação se tornam extremamente úteis.

O sistema que desenvolvemos como parte de uma Atividade Prática Supervisionada (APS) para o curso de Ciência da Computação tem como objetivo principal analisar e comparar o desempenho de três algoritmos de ordenação populares: Merge Sort, Quick Sort e Heap Sort.

É fornecido uma visão detalhada do design e implementação do sistema, bem como uma análise dos resultados obtidos. Ele destaca a importância dos algoritmos de ordenação no gerenciamento eficiente de dados e demonstra como a análise de desempenho pode ser usada para otimizar a eficiência de um sistema. Além disso, o trabalho também discute as lições aprendidas durante o desenvolvimento do sistema e oferece sugestões para trabalhos futuros.

2 Introdução

2.1 A Era Digital e a Importância dos dados

A era digital, caracterizada por avanços tecnológicos rápidos e inovações contínuas, trouxe consigo uma quantidade imensa de dados. Esses dados, que variam de textos simples a imagens complexas, são gerados a cada segundo em todo o mundo. Quando adequadamente organizados e analisados, esses dados podem fornecer insights valiosos, auxiliar na tomada de decisões informadas e até mesmo abrir caminho para novas descobertas. No entanto, a tarefa de organizar esses dados pode ser desafiadora, especialmente quando se trata de grandes volumes de dados. É aqui que os algoritmos de ordenação entram em jogo.

2.2 O Projeto e seus Objetivos

Este trabalho apresenta o desenvolvimento de um sistema para análise de desempenho de algoritmos de ordenação de dados. O sistema foi desenvolvido como parte de uma Atividade Prática Supervisionada (APS) para o curso de Ciência da Computação. A APS é uma atividade acadêmica que visa proporcionar aos alunos a oportunidade de aplicar os conhecimentos teóricos adquiridos em sala de aula em um ambiente prático. O objetivo principal do sistema é analisar e comparar o desempenho de três algoritmos de ordenação populares: Merge Sort, Quick Sort e Heap Sort.

2.3 Tecnologias Utilizadas

O sistema foi desenvolvido usando Java, uma linguagem de programação orientada a objetos, e MySQL, um sistema de gerenciamento de banco de dados relacional. Java é uma linguagem de programação de alto nível que é amplamente utilizada para desenvolver uma variedade de aplicações, desde aplicações web a aplicações móveis. MySQL, por outro lado, é um sistema de gerenciamento de banco de dados relacional que é usado para armazenar e recuperar dados.

2.4 Funcionalidades do Sistema

O sistema coleta imagens, as transforma em blobs e as armazena em um banco de dados com seus respectivos nomes e tamanhos. Um blob é um tipo de dado que pode armazenar uma grande quantidade de dados binários, como imagens ou arquivos de áudio. Em seguida, utiliza os algoritmos de ordenação para ordenar as imagens com base em seu tamanho. Cada algoritmo de ordenação tem suas próprias características e desempenho, e a escolha do algoritmo de ordenação pode ter um impacto significativo no desempenho geral do sistema.

2.4.1 Análise de Desempenho

Além disso, o sistema também calcula o tempo de ordenação, permitindo uma comparação direta do desempenho dos três algoritmos de ordenação. Isso fornece uma visão valiosa sobre qual algoritmo funciona melhor para diferentes tamanhos de conjuntos de dados. A análise do tempo de ordenação também pode ajudar a identificar gargalos de desempenho e áreas de melhoria.

2.4.2 CRUD

O sistema também possui funcionalidades CRUD (Create, Read, Update, Delete), permitindo que o usuário manipule os dados conforme necessário. As funcionalidades CRUD são fundamentais para qualquer sistema de gerenciamento de dados, pois permitem que os usuários interajam com os dados de maneira eficiente e eficaz.

2.4.3 Visualização de Imagens

Ao clicar duas vezes em uma linha de dados de imagem, a imagem é transformada de blob para imagem e é exibida na tela para o usuário. Isso permite que o usuário visualize as imagens que estão sendo ordenadas, proporcionando uma experiência de usuário mais rica e interativa.

2.5 Conclusão

Este trabalho fornece uma visão detalhada do design e implementação do sistema, bem como uma análise dos resultados obtidos. Ele destaca a importância dos algoritmos de ordenação no gerenciamento eficiente de dados e demonstra como a análise de desempenho pode ser usada para otimizar a eficiência de um sistema. Além disso, o trabalho também discute as lições aprendidas durante o desenvolvimento do sistema e oferece sugestões para trabalhos futuros.

3 Referencial Teórico

3.1 Ordenações

Conforme estudos e verificações, foi possível determinar três métodos de ordenação que mais se adequavam ao projeto proposto, assim chegando nos seguintes algoritmos: quicksort, mergesort e heapsort.

3.1.1 ref 1: Algoritmo de Ordenção Merge Sort

O Merge Sort é um algoritmo de ordenação baseado no princípio do algoritmo Dividir para Conquistar. Aqui, um problema é dividido em vários subproblemas. Cada subproblema é resolvido individualmente. Finalmente, os subproblemas são combinados para formar a solução final. O Merge Sort é definido como um algoritmo de ordenação que funciona dividindo um array em subarrays menores, ordenando cada subarray e, em seguida, mesclando os subarrays ordenados de volta para formar o array final ordenado.

3.1.2 ref 2: Algoritmo de Ordenação Quick Sort

O QuickSort é um algoritmo de ordenação baseado na abordagem de dividir e conquistar, onde um array é dividido em subarrays selecionando um elemento como pivô (elemento selecionado do array). Enquanto divide o array, o elemento pivô deve ser posicionado de tal forma que elementos menores que o pivô são mantidos à esquerda e elementos maiores que o pivô são mantidos à direita do pivô. Os subarrays esquerdo e direito também são divididos usando a mesma abordagem. Este processo continua até que cada subarray contenha um único elemento.

3.1.3 ref 3: Algoritmo de Ordenção Heap Sort

Heap Sort é uma técnica de ordenação baseada na estrutura de dados Binary Heap. É semelhante à ordenação por seleção, onde primeiro encontramos o elemento mínimo e colocamos o elemento mínimo no início. Repetimos o mesmo processo para os elementos restantes. O Heap Sort é um algoritmo de ordenação baseado na abordagem de dividir e conquistar. Um array é dividido em subarrays selecionando um elemento como pivô (elemento selecionado do array). Enquanto divide o array, o elemento pivô deve ser posicionado de tal forma que elementos menores

que o pivô são mantidos à esquerda e elementos maiores que o pivô são mantidos à direita do pivô.

3.2 Tecnologias

3.2.1 ref 4: Linguagem de Programação Java

Java é uma linguagem de programação de alto nível, orientada a objetos, que foi projetada para ter o mínimo possível de dependências de implementação. É uma linguagem de programação de propósito geral que permite aos programadores escrever uma vez e rodar em qualquer lugar (WORA), o que significa que o código Java compilado pode rodar em todas as plataformas que suportam Java sem a necessidade de recompilar. As aplicações Java são tipicamente compiladas para bytecode que pode rodar em qualquer máquina virtual Java (JVM) independentemente da arquitetura do computador subjacente. A sintaxe do Java é similar ao C e C++, mas tem menos facilidades de baixo nível do que qualquer um deles.

3.2.2 ref 5: SGBD MySQL

MySQL é um sistema de gerenciamento de banco de dados relacional usado para armazenar e recuperar dados. Um banco de dados é uma coleção estruturada de dados. Pode ser qualquer coisa, desde uma simples lista de compras até uma galeria de imagens ou as vastas quantidades de informações em uma rede corporativa. Para adicionar, acessar e processar dados armazenados em um banco de dados de computador, você precisa de um sistema de gerenciamento de banco de dados como o MySQL Server.

4 Desenvolvimento:

4.1 Introdução

O sistema desenvolvido é uma aplicação Java que analisa o desempenho de diferentes algoritmos de ordenação de dados. O sistema utiliza as tecnologias Java e MySQL para armazenar imagens como blobs em um banco de dados, juntamente com seus respectivos nomes e tamanhos. A aplicação oferece três técnicas diferentes de ordenação: Merge Sort, Quick Sort e Heap Sort, que são usadas para ordenar as imagens de acordo com seu tamanho. Além disso, a aplicação também possui uma classe que calcula o tempo de ordenação, permitindo a comparação entre as diferentes técnicas de ordenação. Para tornar o código ainda mais completo, foi implementado um CRUD (Create, Read, Update, Delete) para que o usuário possa manipular os dados. Ao clicar duas vezes em uma linha de dados de imagem, a imagem é transformada de blob para imagem e é exibida na tela para o usuário. A seguir está descrito todos os estágios do processo de desenvolvimento do sistema:

4.2 Criação do Banco de Dados

O desenvolvimento do sistema começou com a criação do banco de dados. Este é um passo crucial, pois o banco de dados é a espinha dorsal de qualquer aplicação que lida com uma grande quantidade de dados. Neste caso, foi utilizada a tecnologia MySQL, um sistema de gerenciamento de banco de dados relacional de código aberto que é muito popular devido à sua eficiência e facilidade de uso.

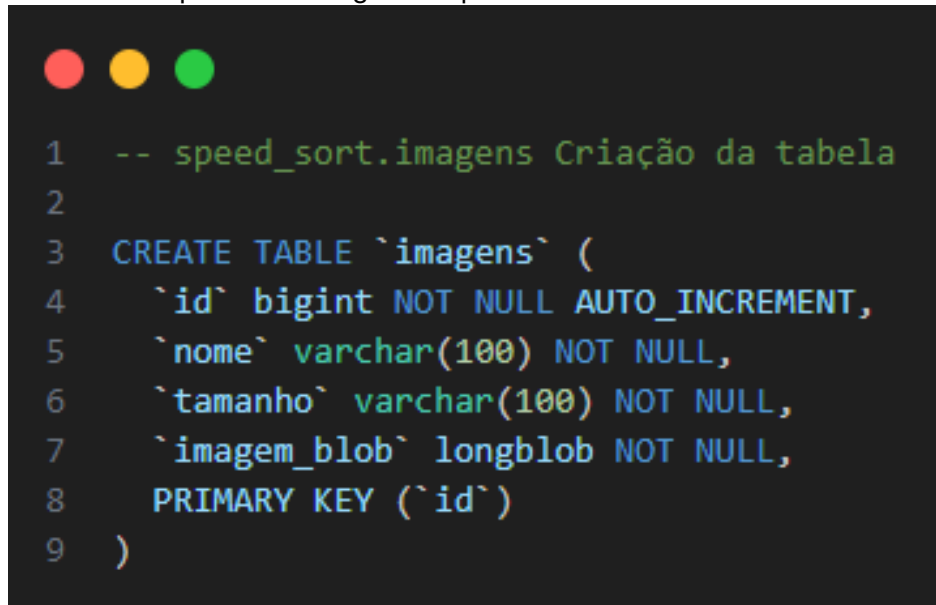
A criação do banco de dados envolveu várias etapas. Primeiro, foi necessário definir a estrutura do banco de dados, incluindo as tabelas e os campos que elas conteriam. Neste caso, o banco de dados precisava armazenar imagens, juntamente com seus respectivos nomes e tamanhos. Portanto, uma tabela foi criada com campos para armazenar essas informações.

Uma vez que o banco de dados foi criado, o próximo passo foi preenchê-lo com dados. Para este sistema, os dados consistiam em imagens que seriam usadas na aplicação. Estas imagens foram baixadas do site <https://satelite.inmet.gov.br/>, que é uma fonte confiável de imagens de satélite. Um total de 66 imagens foram baixadas para serem usadas na aplicação.

No entanto, as imagens baixadas não poderiam ser inseridas diretamente no banco de dados. Em vez disso, elas precisavam ser convertidas em um formato que pudesse ser armazenado no banco de dados. Este formato é conhecido como BLOB (Binary Large Object), que é usado para armazenar grandes quantidades de dados binários, como imagens e vídeos, em um banco de dados.

Portanto, cada imagem baixada foi convertida em um BLOB e, em seguida, inserida no banco de dados junto com seu nome e tamanho. Este processo foi realizado usando um script que automatizou a tarefa, tornando-a mais eficiente e menos propensa a erros.

A criação do banco de dados foi um processo detalhado e meticuloso que estabeleceu a base para o restante do desenvolvimento do sistema. Foi um passo essencial que permitiu que o sistema manipulasse uma grande quantidade de dados de maneira eficiente e eficaz.



```
1  -- speed_sort.imagens Criação da tabela
2
3  CREATE TABLE `imagens` (
4    `id` bigint NOT NULL AUTO_INCREMENT,
5    `nome` varchar(100) NOT NULL,
6    `tamanho` varchar(100) NOT NULL,
7    `imagem_blob` longblob NOT NULL,
8    PRIMARY KEY (`id`)
9  )
```

4.3 População do Banco de dados

Dada a necessidade de inserir um mínimo de 10.000 imagens no banco de dados, a criação de um script para popular o banco de dados automaticamente tornou-se uma etapa essencial no desenvolvimento do sistema. Este script foi projetado para percorrer todas as imagens em uma pasta especificada e realizar uma série de ações para cada imagem, maximizando a eficiência e minimizando a possibilidade de erros.

A primeira ação que o script realiza é abrir a imagem. Isso é feito usando bibliotecas de manipulação de imagens que podem ler uma variedade de formatos de imagem. Abrir a imagem é o primeiro passo necessário para acessar os dados brutos da imagem que serão usados posteriormente.

Em seguida, o script redimensiona a imagem. O redimensionamento é uma etapa importante porque as imagens podem ter tamanhos variados, e trabalhar com imagens de tamanho uniforme pode simplificar o processo de manipulação de imagens. Além disso, o redimensionamento pode ajudar a reduzir o espaço de armazenamento necessário para as imagens, o que pode ser uma consideração importante ao lidar com um grande número de imagens.

Depois de redimensionar a imagem, o script salva a imagem otimizada em um buffer. Um buffer é uma área de armazenamento temporário na memória do computador que é usada para guardar

dados enquanto eles estão sendo transferidos de um lugar para outro. Salvar a imagem em um buffer permite que o script manipule os dados da imagem diretamente na memória, o que pode ser mais rápido do que trabalhar com os dados da imagem em um arquivo ou banco de dados.

O próximo passo que o script realiza é gerar um número aleatório exclusivo. Este número é usado como um identificador único para a imagem no banco de dados. A geração de um número aleatório exclusivo garante que cada imagem tenha um identificador único, o que é essencial para a recuperação e manipulação eficientes dos dados da imagem no banco de dados.

Finalmente, o script insere a imagem no banco de dados. Isso é feito usando comandos SQL para inserir os dados da imagem, juntamente com o nome da imagem e o número aleatório exclusivo, em uma tabela no banco de dados. A inserção dos dados da imagem no banco de dados é a última etapa do script, completando o processo de população do banco de dados.

O script de população do banco de dados desempenha um papel crucial no desenvolvimento do sistema, permitindo a inserção eficiente de um grande número de imagens no banco de dados. Cada etapa do script, desde a abertura e redimensionamento da imagem até a geração de um número aleatório exclusivo e a inserção da imagem no banco de dados, foi cuidadosamente projetada para maximizar a eficiência e a precisão do processo de população do banco de dados.

```

1 import os
2 import random
3 import mysql.connector
4 from PIL import Image
5 import time
6
7 def generate_unique_random(existing_values):
8     while True:
9         random_num = random.randint(1, 10000000)
10        if random_num not in existing_values:
11            existing_values.add(random_num)
12            return random_num
13
14 def redimensionar_comprimir_imagem(imagem, largura, altura):
15     imagem.thumbnail((largura, altura), Image.ANTIALIAS)
16     return imagem
17
18 connection = mysql.connector.connect(
19     host='localhost',
20     database='seu banco de dados',
21     user='seu usuario',
22     password='sua senha'
23 )
24 cursor = connection.cursor()
25
26 pasta_imagens = 'caminho/da/sua/pasta/de/imagens'
27 largura_desejada = 800
28 altura_desejada = 600
29 numeros_aleatorios_usados = set()
30 contador = 0
31
32 for _ in range(300):
33     for filename in os.listdir(pasta_imagens):
34         if filename.endswith((''.jpg', '.png', '.jpeg')):
35             with Image.open(os.path.join(pasta_imagens, filename)) as img:
36                 max_size = (200, 200)
37                 img.thumbnail(max_size, Image.LANCZOS)
38                 from io import BytesIO
39                 buffer = BytesIO()
40                 img.save(buffer, format="JPEG", quality=85)
41                 image_binary = buffer.getvalue()
42                 tamanho_aleatorio = generate_unique_random(numeros_aleatorios_usados)
43                 cursor.execute("INSERT INTO imagens (nome, imagem_blob, tamanho) VALUES (%s, %s, %s)",
44                               (filename, image_binary, tamanho_aleatorio))
45                 connection.commit()
46 cursor.close()
47 connection.close()
48

```

4.4 Criação da Aplicação Java

Após a criação e população do banco de dados, o próximo passo crucial foi a criação da aplicação Java. A aplicação Java é o coração do sistema, pois é onde a lógica de negócios é implementada e a interação do usuário ocorre. A aplicação foi cuidadosamente projetada para ser robusta, eficiente e fácil de usar.

A primeira etapa na criação da aplicação Java foi estabelecer uma conexão com o banco de dados. Isso foi feito usando o driver JDBC do MySQL, que permite que aplicações Java interajam com o MySQL e outros bancos de dados SQL. A conexão com o banco de dados é vital, pois permite que a aplicação recupere, insira, atualize e exclua dados no banco de dados.

Uma vez estabelecida a conexão com o banco de dados, a aplicação foi configurada para exibir todos os dados para o usuário em uma tabela. A tabela fornece uma representação visual dos dados armazenados no banco de dados, permitindo que o usuário veja e interaja com os dados de maneira intuitiva. A tabela foi projetada para ser fácil de ler, com colunas claramente rotuladas e linhas ordenadas de maneira lógica.

Em seguida, foi implementado o CRUD (Create, Read, Update, Delete), um conjunto de operações que permite ao usuário manipular os dados. O CRUD é uma parte fundamental de qualquer aplicação de gerenciamento de dados, pois fornece ao usuário o controle necessário para gerenciar os dados de acordo com suas necessidades. A implementação do CRUD envolveu a criação de interfaces de usuário para cada operação (criar, ler, atualizar, excluir) e a implementação da lógica de negócios para realizar as operações no banco de dados.

Além disso, foi implementada uma funcionalidade que permite ao usuário visualizar uma imagem ao clicar duas vezes em uma linha de dados. Esta funcionalidade melhora a usabilidade da aplicação, permitindo ao usuário ver as imagens armazenadas no banco de dados de maneira rápida e fácil. A imagem é recuperada do banco de dados, convertida de um blob para uma imagem e exibida em uma nova janela.

A criação da aplicação Java envolveu várias etapas cuidadosamente planejadas e executadas, desde a conexão com o banco de dados até a implementação do CRUD e a visualização de imagens. Cada etapa foi projetada para maximizar a eficiência, a usabilidade e a robustez da aplicação, resultando em um sistema poderoso e fácil de usar para análise de desempenho de algoritmos de ordenação de dados.

4.5 Implementação dos Algoritmos de Ordenação

Com todas as funcionalidades básicas implementadas e funcionando corretamente, o próximo passo crucial no desenvolvimento do sistema foi a implementação dos algoritmos de ordenação. A ordenação é uma parte fundamental de qualquer sistema que lida com grandes conjuntos de dados, pois permite que os dados sejam organizados de maneira lógica e eficiente. Neste sistema, foram implementados três algoritmos de ordenação diferentes, cada um com suas próprias características e vantagens: Merge Sort, Quick Sort e Heap Sort.

4.5.1 Merge Sort

O Merge Sort é um algoritmo de ordenação baseado na estratégia de “dividir para conquistar”. Ele divide o conjunto de dados em duas metades, ordena cada metade separadamente e depois as combina para formar o conjunto ordenado. Este algoritmo é conhecido por sua eficiência e estabilidade, tornando-o uma escolha popular para a ordenação de grandes conjuntos de dados.

```

1  public void mergeSort(int[] array, int left, int right) {
2      if (right <= left) return;
3      int mid = (left+right)/2;
4      mergeSort(array, left, mid);
5      mergeSort(array, mid+1, right);
6      merge(array, left, mid, right);
7  }
8
9  void merge(int[] array, int left, int mid, int right) {
10     int n1 = mid - left + 1;
11     int n2 = right - mid;
12
13     int L[] = new int[n1];
14     int R[] = new int[n2];
15
16     for (int i=0; i<n1; ++i)
17         L[i] = array[left + i];
18     for (int j=0; j<n2; ++j)
19         R[j] = array[mid + 1+ j];
20
21     int i = 0, j = 0;
22
23     int k = left;
24     while (i < n1 && j < n2) {
25         if (L[i] <= R[j]) {
26             array[k] = L[i];
27             i++;
28         } else {
29             array[k] = R[j];
30             j++;
31         }
32         k++;
33     }
34
35     while (i < n1) {
36         array[k] = L[i];
37         i++;
38         k++;
39     }
40
41     while (j < n2) {
42         array[k] = R[j];
43         j++;
44         k++;
45     }
46 }

```

4.5.2 Quick Sort

O Quick Sort, por outro lado, é um algoritmo de ordenação que utiliza uma estratégia de particionamento. Ele seleciona um elemento “pivô” e organiza os dados de forma que todos os elementos

menores que o pivô estejam à sua esquerda e todos os elementos maiores estejam à sua direita. O processo é então repetido para cada subconjunto de elementos à esquerda e à direita do pivô. O Quick Sort é conhecido por sua velocidade e eficiência, especialmente em conjuntos de dados

```
1 public void quickSort(int[] array, int low, int high) {
2     if (low < high) {
3         int pi = partition(array, low, high);
4
5         quickSort(array, low, pi-1);
6         quickSort(array, pi+1, high);
7     }
8 }
9
10 int partition(int[] array, int low, int high) {
11     int pivot = array[high];
12     int i = (low-1);
13     for (int j=low; j<high; j++) {
14         if (array[j] < pivot) {
15             i++;
16             int temp = array[i];
17             array[i] = array[j];
18             array[j] = temp;
19         }
20     }
21
22     int temp = array[i+1];
23     array[i+1] = array[high];
24     array[high] = temp;
25
26     return i+1;
27 }
```

grandes e complexos.

4.5.3 Heap Sort

Por último, mas não menos importante, o Heap Sort é um algoritmo de ordenação que utiliza uma estrutura de dados chamada “heap”. Um heap é uma árvore binária completa onde cada nó pai tem um valor maior (no caso de um heap máximo) ou menor (no caso de um heap mínimo) do que seus nós filhos. O Heap Sort transforma o conjunto de dados em um heap, depois remove repetidamente o maior (ou menor) elemento e o coloca no final do conjunto não ordenado, até que todo o

```

1
2  public void HeapSort(int[] arr) {
3      int n = arr.length;
4      for (int i = n / 2 - 1; i >= 0; i--)
5          heapify(arr, n, i);
6      for (int i=n-1; i>=0; i--) {
7          int temp = arr[0];
8          arr[0] = arr[i];
9          arr[i] = temp;
10         heapify(arr, i, 0);
11     }
12 }
13
14 void heapify(int[] arr, int n, int i) {
15     int largest = i;
16     int l = 2*i + 1;
17     int r = 2*i + 2;
18     if (l < n && arr[l] > arr[largest])
19         largest = l;
20     if (r < n && arr[r] > arr[largest])
21         largest = r;
22     if (largest != i) {
23         int swap = arr[i];
24         arr[i] = arr[largest];
25         arr[largest] = swap;
26         heapify(arr, n, largest);
27     }
28 }

```

conjunto esteja ordenado.

Cada um desses algoritmos de ordenação foi associado a um botão na interface do usuário, permitindo ao usuário escolher qual algoritmo deseja usar para ordenar as imagens. Isso não apenas oferece ao usuário controle sobre o processo de ordenação, mas também permite que ele compare a eficiência e a velocidade dos diferentes algoritmos de ordenação em tempo real. Esta é uma característica poderosa que destaca a flexibilidade e a capacidade do sistema de se adaptar às necessidades do usuário.

4.6 Medição do Tempo de Ordenação

Para permitir uma comparação efetiva entre os diferentes algoritmos de ordenação, foi implementada uma classe especializada que é responsável por calcular o tempo de ordenação. Esta classe desempenha um papel crucial no sistema, pois fornece uma métrica quantitativa do desempenho de cada algoritmo de ordenação.

A classe de tempo de ordenação funciona registrando o tempo imediatamente antes e imediatamente após a execução de um algoritmo de ordenação. Isso é feito usando funções de tempo de alta precisão que são capazes de medir o tempo com uma resolução de milissegundos ou até mesmo microssegundos. Ao registrar o tempo antes e depois da ordenação, é possível calcular a diferença entre esses dois tempos, que é o tempo total gasto pelo algoritmo para ordenar os dados.

No entanto, a implementação desta classe não é uma tarefa trivial. É necessário garantir que o tempo seja registrado com precisão e que fatores externos, como a carga do sistema ou a interrupção do processador, não afetem as medições. Portanto, foram tomadas precauções para garantir que o sistema estivesse em um estado controlado durante a medição do tempo de ordenação.

Além disso, a classe de tempo de ordenação não se limita a calcular apenas o tempo total de ordenação. Ela também pode ser estendida para calcular outras métricas de desempenho, como o número de comparações ou trocas realizadas pelo algoritmo de ordenação. Isso pode fornecer uma visão mais detalhada do comportamento do algoritmo de ordenação e pode ajudar a identificar áreas onde o algoritmo pode ser otimizado para melhor desempenho.

A classe de tempo de ordenação é uma parte vital do sistema que permite a comparação efetiva do desempenho dos diferentes algoritmos de ordenação. Ela fornece uma métrica quantitativa do tempo de ordenação, permitindo que os usuários vejam claramente qual algoritmo é o mais eficiente para ordenar os dados. A implementação desta classe envolveu considerações cuidadosas de precisão e controle do sistema, resultando em uma ferramenta poderosa para a análise de de-

sempenho do algoritmo de ordenação.

```
1 long tempoInicial = System.currentTimeMillis();
2     quickSort(tamanhos, 0, rowCount - 1);
3     long tempoFinal = System.currentTimeMillis();
4
5     long tempoTotal = tempoFinal - tempoInicial;
6
7     long minutos = (tempoTotal / 60000) % 60;
8     long segundos = (tempoTotal / 1000) % 60;
9     long milissegundos = tempoTotal % 1000;
10
11     String tempoFormatado = String.format("%02d:%02d:%03d", minutos, segundos, milissegundos);
12     JOptionPane.showMessageDialog(null, "Tempo gasto na ordenação: " + tempoFormatado);
```

4.7 Melhorias na Interface do Usuário

Finalmente, foram implementadas várias melhorias na interface do usuário para tornar a aplicação mais agradável visualmente. A interface do usuário é um aspecto crucial de qualquer aplicação, pois é o ponto de interação entre o usuário e o sistema. Uma interface do usuário bem projetada pode melhorar significativamente a experiência do usuário, tornando a aplicação não apenas funcional, mas também agradável de usar.

Uma das melhorias implementadas foi a adição de um botão extra para recarregar a tabela. Este botão desempenha uma função importante na aplicação, pois permite ao usuário testar diferentes algoritmos de ordenação sem a necessidade de fechar e reabrir o programa. Isso melhora a eficiência da aplicação, permitindo que os usuários realizem várias operações de ordenação em rápida sucessão. Além disso, também melhora a experiência do usuário, pois elimina a necessidade de reiniciar a aplicação, o que pode ser um processo demorado e frustrante.

Além do botão de recarga, foram feitas várias outras melhorias na interface do usuário. Por exemplo, o layout da interface do usuário foi otimizado para torná-lo mais intuitivo e fácil de navegar. Os elementos da interface do usuário foram organizados de maneira lógica e coerente, tornando mais fácil para os usuários encontrar as funções que desejam usar.

Além disso, foram feitas melhorias visuais na interface do usuário para torná-la mais atraente. Isso incluiu a escolha de uma paleta de cores agradável, a utilização de imagens onde apropriado, e a escolha de fontes legíveis e atraentes. Essas melhorias visuais não apenas tornam a aplicação mais agradável de se olhar, mas também podem melhorar a usabilidade, tornando mais fácil para os usuários entenderem e interagirem com a aplicação.

As melhorias na interface do usuário desempenharam um papel crucial no desenvolvimento do sistema. Elas não apenas tornaram a aplicação mais agradável de usar, mas também melhoraram sua funcionalidade e eficiência. Através da implementação de um botão de recarga, da otimização do layout da interface do usuário e da realização de melhorias visuais, a aplicação tornou-se não apenas uma ferramenta poderosa para a análise de desempenho do algoritmo de ordenação, mas também uma aplicação agradável e fácil de usar.

4.7.1 Tela

—

□

×

CADASTRE, EDITE OU EXCLUA IMAGENS

Nome

!0231016_189_103.png

Tamanho:

210

Cadastrar

Editar

Excluir

Selecionar Imagem

ID

Nome

Tamanho

Imagem Blob

1913

CBERS_4 M...

210

[B@14a6674e

14009

CBERS_4A_...

772

[B@9ca3937

2282

CBERS_4A_...

834

[B@da0b3ce

2659

CBERS_4A_...

2091

[B@534701e1

13157

CBERS_4A_...

2433

[B@74fc80d0

246

CBERS_4A_...

2828

[B@2c270cf9

18123

CBERS_4A_...

2838

[B@2411e470

13406

AMAZONIA_1...

3848

[B@4f32b11a

Ordenações

Quick Sort

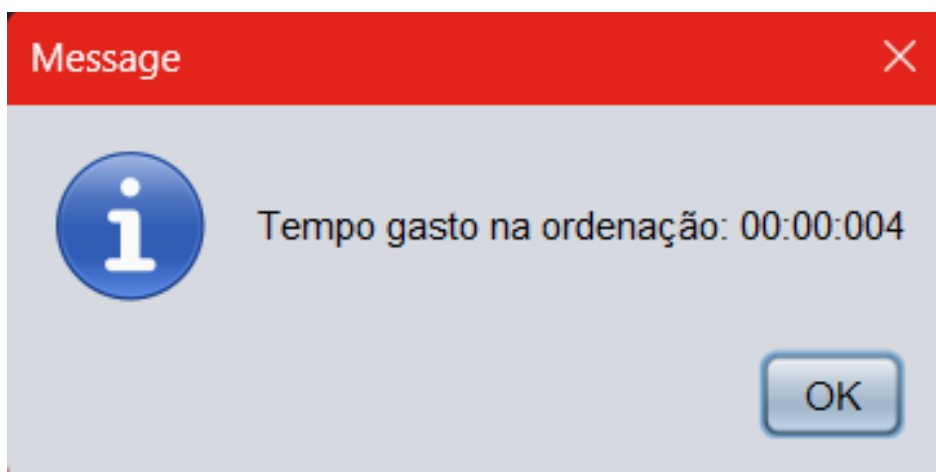
Merge Sort

Heap Sort

Recarregar Banco

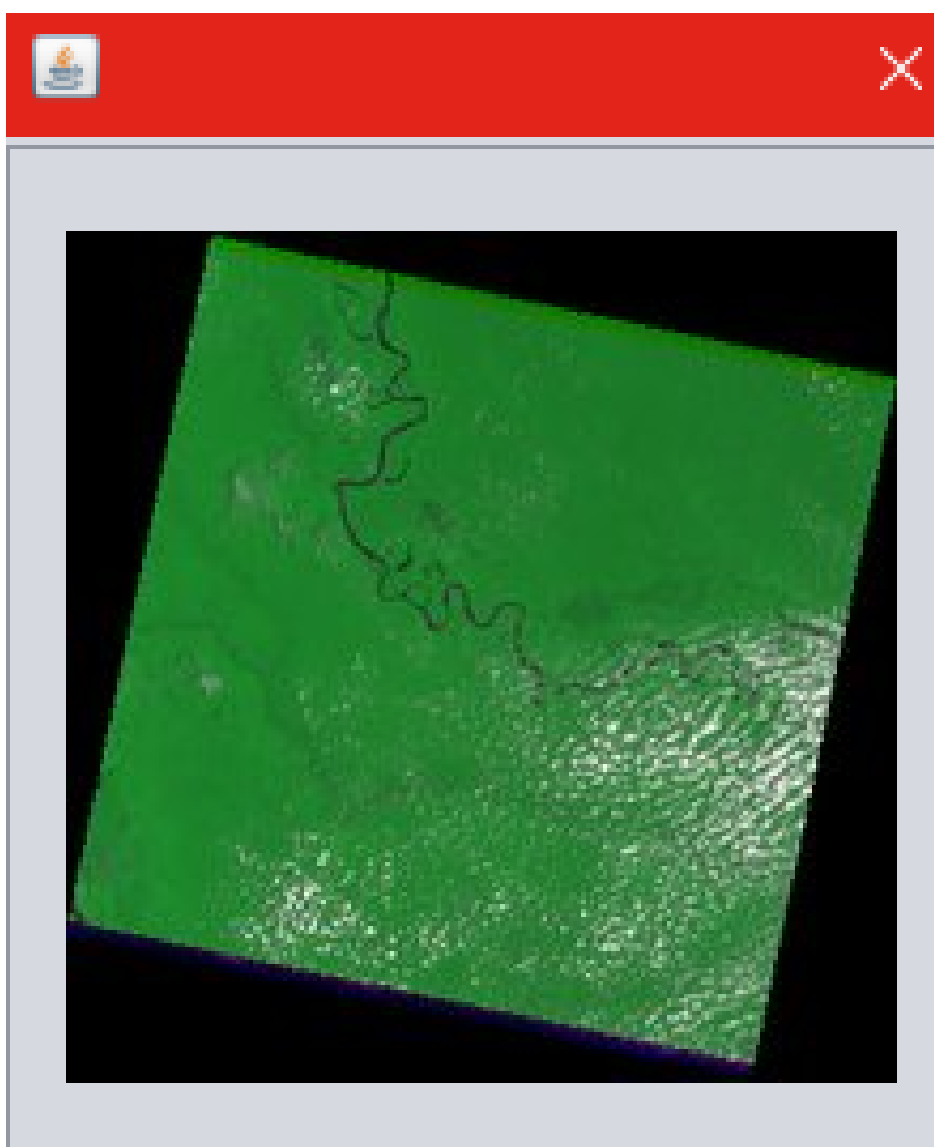
18

4.7.2 Tela Tempo Decorrido



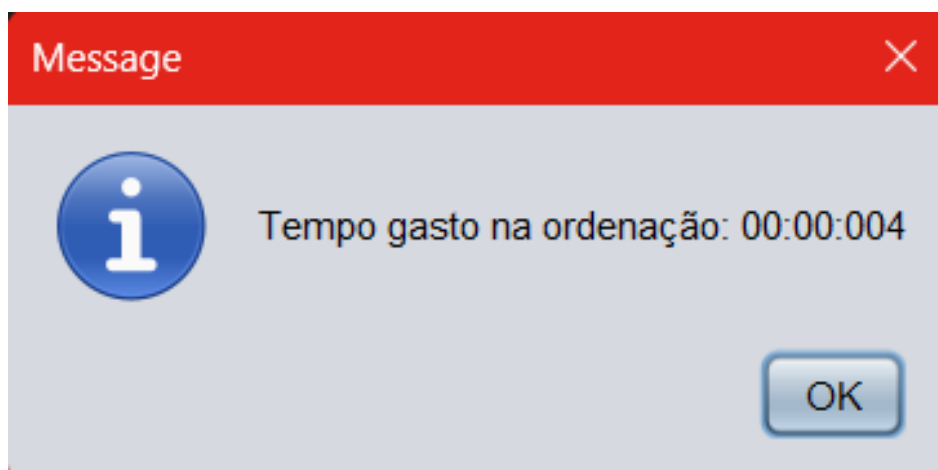
Merge

4.7.3 Tela DoubleClick na Imagem

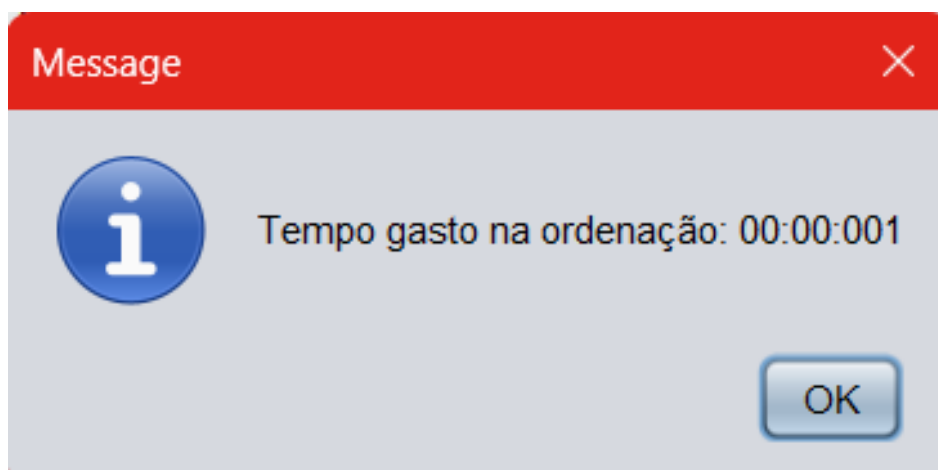


4.8 Comparativo Tempo Decorrido

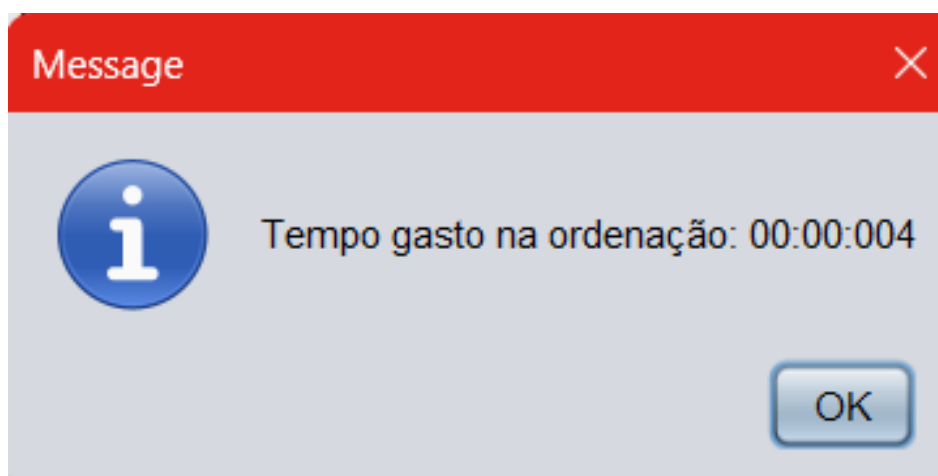
4.8.1 Merge



4.8.2 Quick

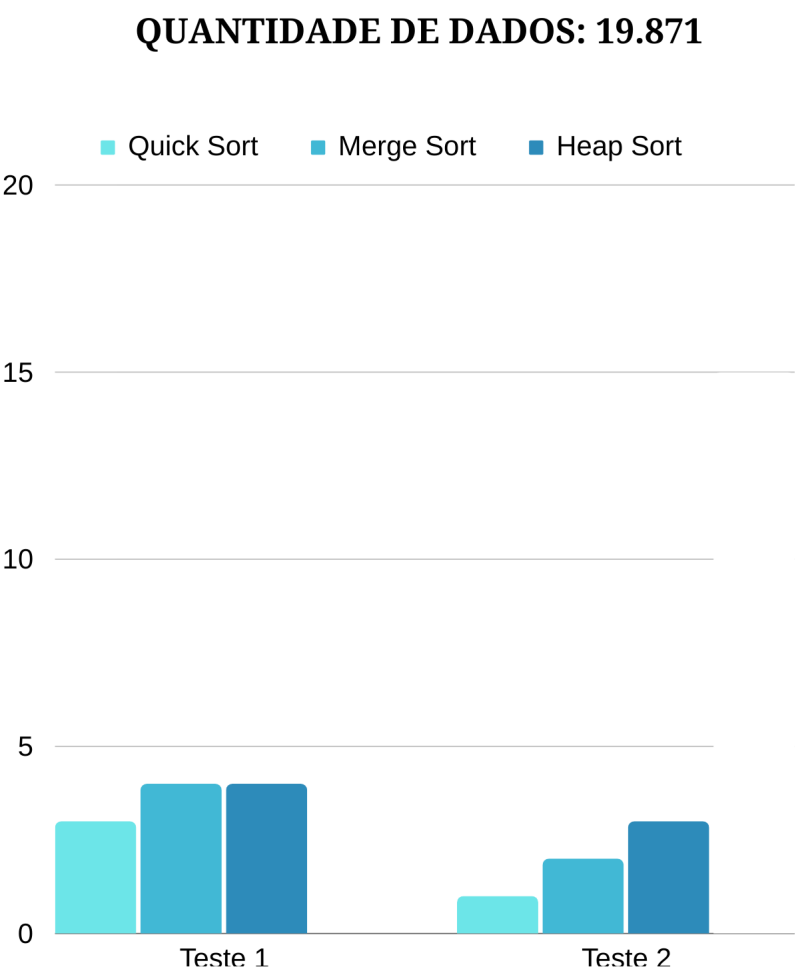


4.8.3 Heap

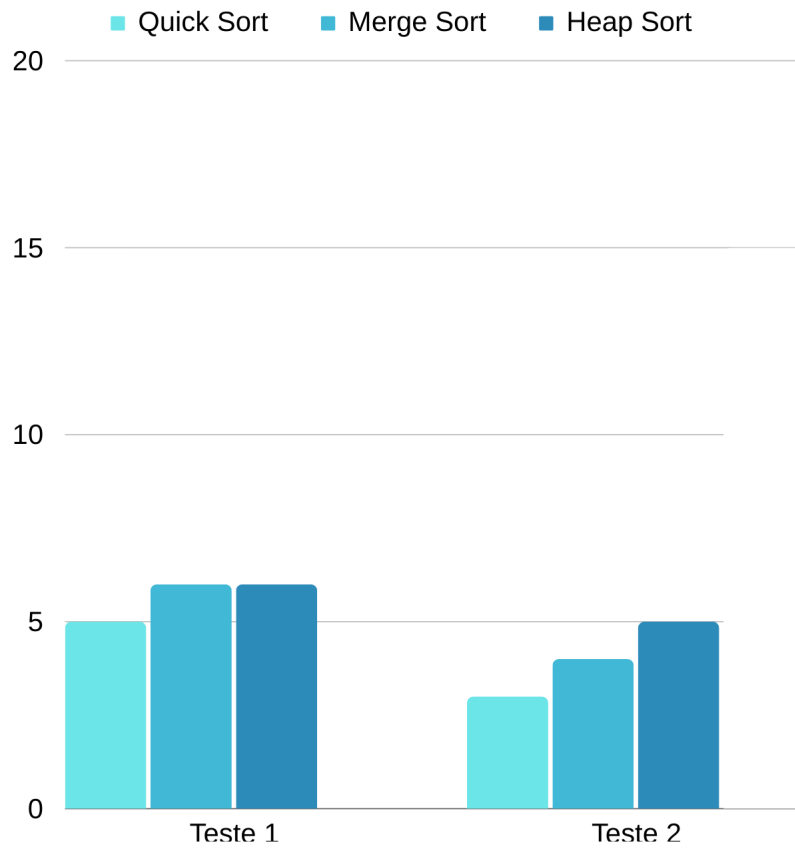


5 Resultados:

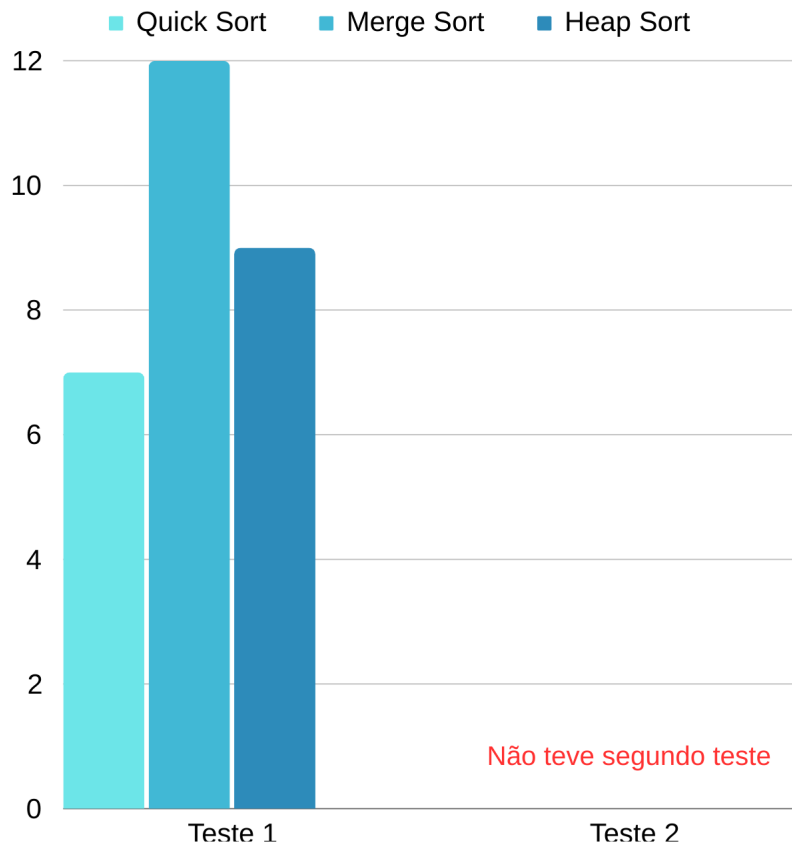
5.1 Graficos



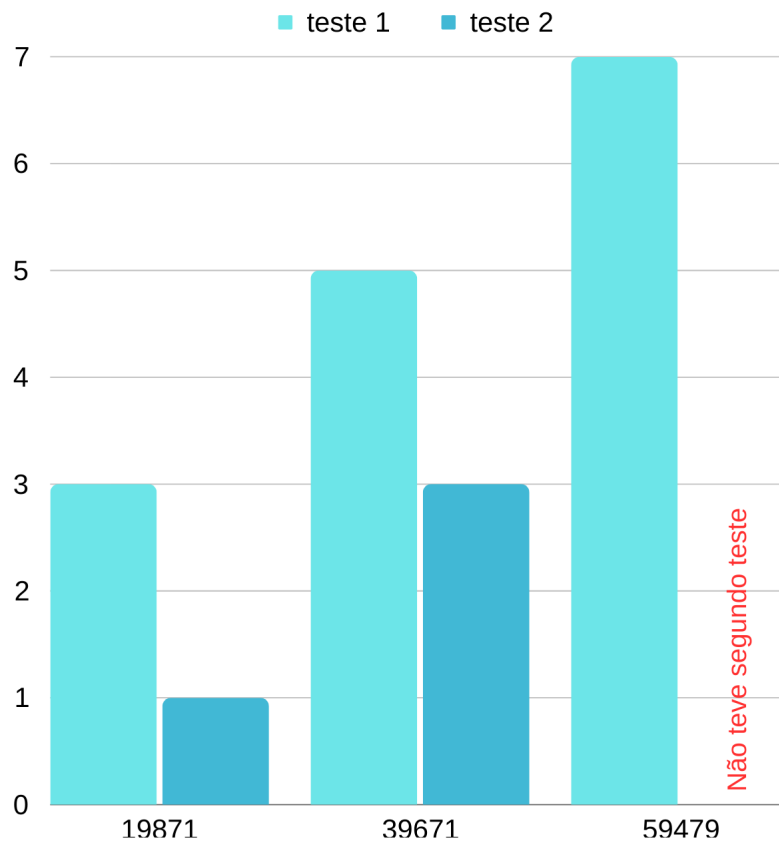
QUANTIDADE DE DADOS: 39.671



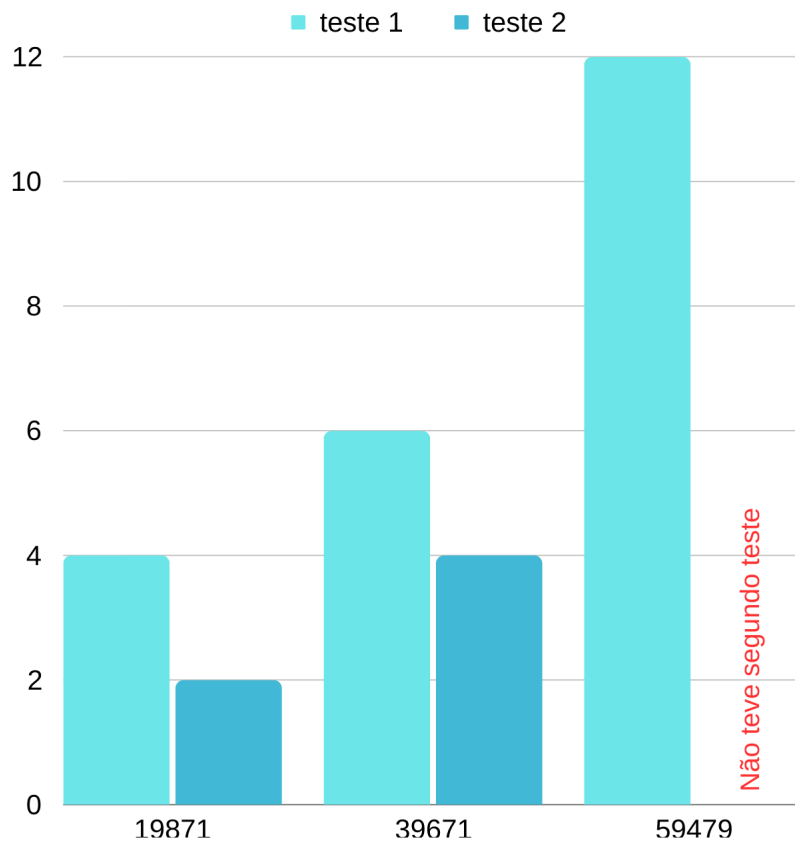
QUANTIDADE DE DADOS: 59.479



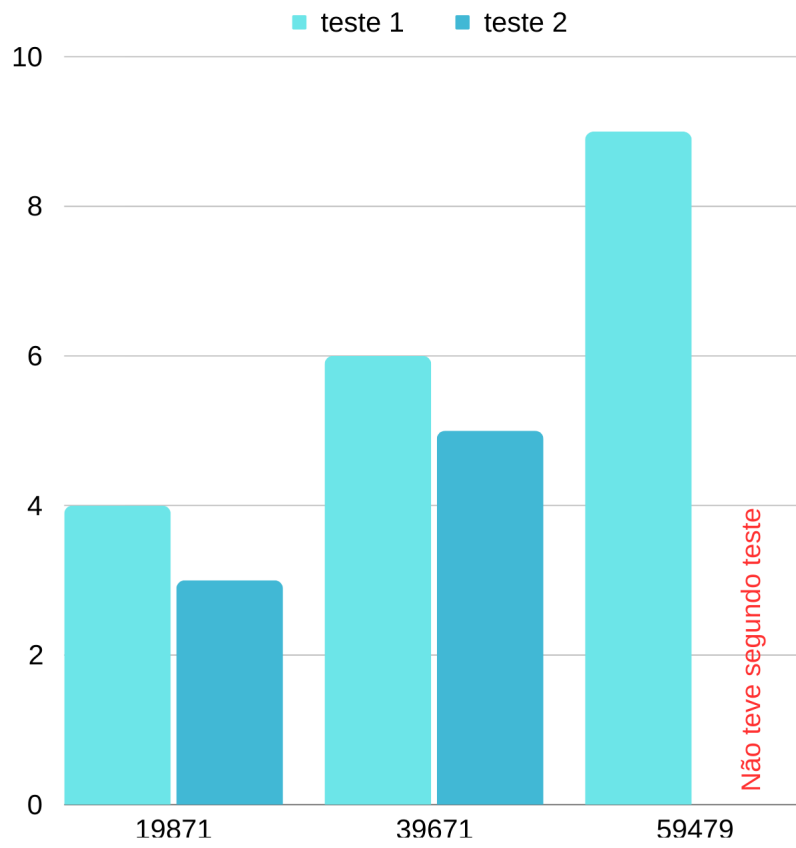
QUICK SORT



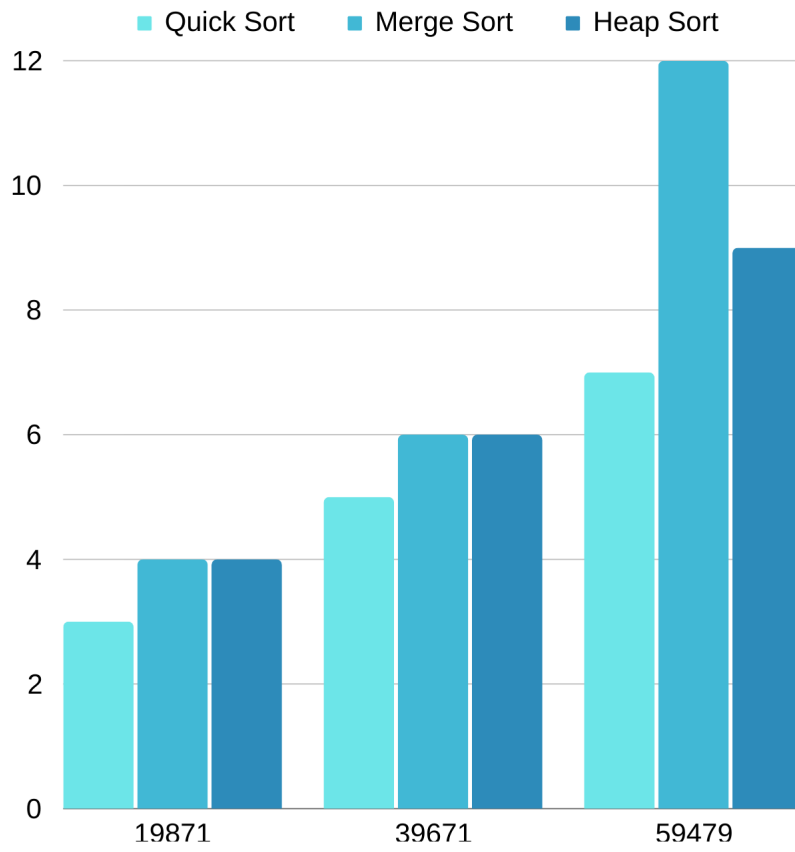
MERGE SORT



HEAP SORT



DESEMPENHO



DESEMPENHO

QNT.	QUICK SORT	MERGE SORT	HEAP SORT
19871	3 ms	4 ms	4 ms
39671	5 ms	6 ms	6 ms
59479	7 ms	12 ms	9 ms

5.2 Discussão

Com base nos resultados dos testes que você realizou, aqui estão algumas conclusões que podemos tirar sobre o desempenho de cada algoritmo de ordenação:

1. **Quick Sort:** Este algoritmo teve o melhor desempenho em todos os tamanhos de dados testados. O tempo de execução aumentou de maneira previsível à medida que o tamanho dos dados aumentava, sugerindo um bom desempenho mesmo para conjuntos de dados maiores.

2. **Merge Sort:** Este algoritmo teve um desempenho ligeiramente pior do que o Quick Sort para os tamanhos de dados menores, mas o tempo de execução aumentou mais rapidamente para o

maior conjunto de dados. Isso sugere que o Merge Sort pode não ser tão eficiente para conjuntos de dados muito grandes.

3. **Heap Sort:** Este algoritmo teve um desempenho semelhante ao Merge Sort para os tamanhos de dados menores, mas foi mais eficiente para o maior conjunto de dados. Isso sugere que o Heap Sort pode ser uma boa escolha para conjuntos de dados maiores, onde a eficiência do Merge Sort começa a diminuir.

Em resumo, o Quick Sort parece ser a melhor escolha para todos os tamanhos de dados com base nos seus testes. No entanto, o Heap Sort pode ser uma alternativa viável para conjuntos de dados maiores, onde o desempenho do Merge Sort começa a diminuir. Como sempre, a escolha do algoritmo de ordenação pode depender de vários fatores, incluindo a natureza dos dados e as restrições de tempo e espaço.

6 Considerações Finais

6.1 Considerações gerais

O sistema desenvolvido demonstra a aplicação prática e eficaz dos algoritmos de ordenação Quick Sort, Merge Sort e Heap Sort. Esses algoritmos foram implementados para ordenar imagens com base em seu tamanho, demonstrando assim a eficiência desses algoritmos em um cenário real.

A aplicação também incorpora um recurso de CRUD, permitindo ao usuário interagir diretamente com os dados. Isso não só torna o sistema mais interativo, mas também permite que os usuários vejam os resultados da ordenação em tempo real.

Além disso, o sistema foi capaz de converter imagens em blobs e armazená-las em um banco de dados MySQL, demonstrando a capacidade de lidar com diferentes tipos de dados e formatos.

Por fim, o sistema também inclui uma funcionalidade que calcula o tempo de ordenação, permitindo uma comparação direta entre os diferentes algoritmos de ordenação. Isso é crucial para entender a eficiência dos diferentes algoritmos e pode fornecer insights valiosos para futuras otimizações.

Em resumo, este trabalho apresenta um sistema robusto e funcional que não só demonstra a aplicação prática dos algoritmos de ordenação, mas também permite uma análise aprofundada de sua performance. Isso representa um passo significativo no campo da ciência da computação e abre caminho para futuras pesquisas e desenvolvimentos.

7 Código fonte:

7.1 MySQL

Código 1: codigo.sql

```
1  -- speed_sort.imagens Cria o da tabela
2
3  CREATE TABLE 'imagens' (
4      'id' bigint NOT NULL AUTO_INCREMENT,
5      'nome' varchar(100) NOT NULL,
6      'tamanho' varchar(100) NOT NULL,
7      'imagem_blob' longblob NOT NULL,
8      PRIMARY KEY ('id')
9  ) ENGINE=InnoDB AUTO_INCREMENT=19880 DEFAULT CHARSET=utf8mb4 COLLATE=utf8mb4_0900_ai_ci
   ;
```

7.2 Java

Código 2: codigo.java

```
1  package com.mycompany.speed_sort;
2
3
4  import javax.swing.*; // usado para criar interfaces gráficas em Java.
5  import javax.swing.JOptionPane; //Caixas de diálogo.
6  import java.io.IOException; //Lida com exceções de entrada/saída.
7  import javax.imageio.ImageIO; // Lê e grava imagens.
8  import java.awt.Graphics2D; //Realiza operações gráficas avançadas.
9  import java.io.ByteArrayOutputStream; //Grava dados em um array de bytes em memória
10 import java.awt.image.BufferedImage; //Representa imagens em formato de bitmap.
11 import java.sql.ResultSet; //Manipula resultados de consultas em bancos de dados.
12 import javax.swing.filechooser.FileNameExtensionFilter; //Cria filtros para seleção
    de tipos de arquivo.
13 import java.awt.Image; //Representa imagens em interfaces gráficas.
14 import java.io.File; //Lida com arquivos e diretórios.
15 import java.sql.Connection; //Estabelece conexão com bancos de dados.
16 import java.sql.DriverManager; //Gerencia conexões de banco de dados.
17 import java.sql.PreparedStatement; //Executa instruções SQL predefinidas.
18 import java.sql.SQLException; // Lida com exceções relacionadas a bancos de dados.
19 import javax.swing.table.DefaultTableModel; //Representa o modelo de dados de uma
    tabela.
20 import javax.swing.table.TableRowSorter; //Classifica e filtra dados de uma tabela.
```

```

21 import java.awt.event.MouseAdapter; //Lida com eventos relacionados ao mouse.
22 import java.awt.event.MouseEvent; //Representa eventos de mouse.
23 import java.io.ByteArrayInputStream; //Cria fluxo de entrada a partir de um array de
    bytes.
24
25
26 /**
27  *
28  * @author gabri
29  */
30 public class FrameInicial extends javax.swing.JFrame {
31     private Connection connection;
32     public FrameInicial() {
33         initComponents();
34         initializeDatabaseConnection();
35         DefaultTableModel modelo = (DefaultTableModel) jTable1.getModel();
36         TableRowSorter<DefaultTableModel> sorter = new TableRowSorter<>(modelo);
37         jTable1.setRowSorter(sorter);
38
39         jTable1.addMouseListener(new MouseAdapter() {
40             @Override
41             // quando click duplo na linha, abre a imagem.
42             public void mouseClicked(MouseEvent e) {
43                 if (e.getClickCount() == 2) {
44                     int selectedRow = jTable1.getSelectedRow();
45                     if (selectedRow != -1) {
46                         byte[] imageData = (byte[]) jTable1.getValueAt(selectedRow, 3);
47                         //dado da imagem na 3 coluna
48                         if (imageData != null && imageData.length > 0) {
49                             displayImage(imageData);
50                         }
51                     }
52                 }
53             });
54     }
55
56
57     // classe interna que representa um objeto de imagem com atributos
58     public class Imagem {
59         private int id;
60         private String nome;
61         private String tamanho;
62         private byte[] imagemBlob;

```

```
63
64     public int getId() {
65         return id;
66     }
67
68     public void setId(int id) {
69         this.id = id;
70     }
71
72     public String getNome() {
73         return nome;
74     }
75
76     public void setNome(String nome) {
77         this.nome = nome;
78     }
79
80     public String getTamanho() {
81         return tamanho;
82     }
83
84     public void setTamanho(String tamanho) {
85         this.tamanho = tamanho;
86     }
87
88     public byte[] getImagemBlob() {
89         return imagemBlob;
90     }
91
92     public void setImagemBlob(byte[] imagemBlob) {
93         this.imagemBlob = imagemBlob;
94     }
95 }
96
97 //realiza a conexao com o banco de dados
98 private void initializeDatabaseConnection() {
99
100     try {
101         Class.forName("com.mysql.cj.jdbc.Driver");
102
103         String url = "jdbc:mysql://localhost:3306/speed_sort";
104         String usuario = "root";
105         String senha = "minha senha";
106         connection = DriverManager.getConnection(url, usuario, senha);
```

```

107         carregarDadosDaTabela();
108     } catch (ClassNotFoundException | SQLException ex) {
109         ex.printStackTrace();
110         JOptionPane.showMessageDialog(this, "Erro ao conectar ao banco de dados.");
111     }
112 }
113
114 // Extrai dados da imagem exibida em um componente imageLabel.
115 // Ele converte a imagem em um formato de array de bytes (byte array).
116 private byte[] getImagemData() {
117     ImageIcon imagelcon = (Icon) imageLabel.getIcon();
118
119     if (imagelcon != null) {
120         Image imagem = imagelcon.getImage();
121
122         BufferedImage bufferedImage = new BufferedImage(imagem.getWidth(null),
123             imagem.getHeight(null), BufferedImage.TYPE_INT_RGB);
124         Graphics2D g = bufferedImage.createGraphics();
125         g.drawImage(imagem, 0, 0, null);
126         g.dispose();
127         try {
128             ByteArrayOutputStream baos = new ByteArrayOutputStream();
129
130             ImageIO.write(bufferedImage, "jpg", baos);
131
132             byte[] imageData = baos.toByteArray();
133             return imageData;
134         } catch (IOException ex) {
135             ex.printStackTrace();
136         }
137     }
138     JOptionPane.showMessageDialog(this, "Insira uma imagem Primeiro!");
139     return null;
140 }
141
142 //Esta funcao verifica se uma string contem letras (caracteres alfabeticos).
143 //Ela e usada para garantir que o campo "Tamanho" contenha apenas numeros
144 //inteiros.
145 private boolean containsLetter(String text) {
146     for (char c : text.toCharArray()) {
147         if (Character.isLetter(c)) {
148             return true;
149         }
150     }
151 }

```

```

149         return false;
150     }
151
152     //Essa funcao carrega dados do banco de dados e exibe na tabela jTable1.
153     //Ela consulta o banco de dados, obtm os resultados e preenche o modelo da tabela
154     .
155     private void carregarDadosDaTabela() {
156         DefaultTableModel modelo = (DefaultTableModel) jTable1.getModel();
157         modelo.setRowCount(0);
158
159         String sql = "SELECT id, nome, tamanho, imagem_blob FROM imagens";
160
161         try {
162             PreparedStatement preparedStatement = connection.prepareStatement(sql);
163             ResultSet resultSet = preparedStatement.executeQuery();
164
165             while (resultSet.next()) {
166                 int id = resultSet.getInt("id");
167                 String nome = resultSet.getString("nome");
168                 String tamanho = resultSet.getString("tamanho");
169                 byte[] imagemBlob = resultSet.getBytes("imagem_blob");
170
171                 Imagem imagem = new Imagem();
172                 imagem.setId(id);
173                 imagem.setNome(nome);
174                 imagem.setTamanho(tamanho);
175                 imagem.setImagemBlob(imagemBlob);
176
177                 modelo.addRow(new Object[]{id, nome, tamanho, imagemBlob});
178             }
179         } catch (SQLException ex) {
180             ex.printStackTrace();
181             JOptionPane.showMessageDialog(this, "Erro ao carregar dados do banco de
182                                     dados.");
183         }
184         jTable1.setDefaultEditor(Object.class, null);
185     }
186
187     //Essa funcao insere novos dados no banco de dados.
188     //Ela recebe o nome, tamanho e uma representa o em array de bytes de uma imagem
189     e realiza a insercao no banco de dados.
190     //Apas a insercao, a tabela atualizada para refletir os novos dados.
191     private void inserirDadosNoBanco(String nome, String tamanho, byte[] imagem) {

```

```

190         String sql = "INSERT INTO imagens (nome, tamanho, imagem_blob) VALUES (?, ?, ?)
191         ";
192
193     try {
194         PreparedStatement preparedStatement = connection.prepareStatement(sql);
195         preparedStatement.setString(1, nome);
196         preparedStatement.setString(2, tamanho);
197         preparedStatement.setBytes(3, imagem);
198
199         int resultado = preparedStatement.executeUpdate();
200         carregarDadosDaTabela();
201         if (resultado == 1) {
202             JOptionPane.showMessageDialog(this, "Dados inseridos no banco de dados
203             com sucesso.");
204         } else {
205             JOptionPane.showMessageDialog(this, "Erro ao inserir dados no banco de
206             dados.");
207         }
208     } catch (SQLException ex) {
209         ex.printStackTrace();
210         JOptionPane.showMessageDialog(this, "Erro ao inserir dados no banco de
211         dados.");
212     }
213 }
214
215 /**
216  * This method is called from within the constructor to initialize the form.
217  * WARNING: Do NOT modify this code. The content of this method is always
218  * regenerated by the Form Editor.
219  */
220
221 @SuppressWarnings("unchecked")
222 // <editor-fold defaultstate="collapsed" desc="Generated Code">
223 private void initComponents() {
224
225     progressBar1 = new javax.swing.JProgressBar();
226     jLabel2 = new javax.swing.JLabel();
227     textFieldTam = new javax.swing.JTextField();
228     updateButton = new javax.swing.JButton();
229     deleteButton = new javax.swing.JButton();
230     insertButton = new javax.swing.JButton();
231     imageLabel = new javax.swing.JLabel();
232     openImageButton = new javax.swing.JButton();
233     jScrollPane1 = new javax.swing.JScrollPane();
234     jTable1 = new javax.swing.JTable();

```

```

230     QuickButton = new javax.swing.JButton();
231     jLabel3 = new javax.swing.JLabel();
232     MergeButton = new javax.swing.JButton();
233     HeapButton = new javax.swing.JButton();
234     jLabel4 = new javax.swing.JLabel();
235     jSeparator1 = new javax.swing.JSeparator();
236     timeLabel = new javax.swing.JLabel();
237     jLabel1 = new javax.swing.JLabel();
238     TextFieldName = new javax.swing.JTextField();
239     jLabel5 = new javax.swing.JLabel();
240     timeText = new javax.swing.JLabel();
241     reloadDatabase = new javax.swing.JButton();
242
243     setDefaultCloseOperation(javax.swing.WindowConstants.EXIT_ON_CLOSE);
244     setBackground(new java.awt.Color(232, 219, 199));
245
246     jLabel2.setText("Tamanho:");
247
248     TextFieldTam.setColumns(7);
249     TextFieldTam.setMaximumSize(new java.awt.Dimension(7, 7));
250     TextFieldTam.addActionListener(new java.awt.event.ActionListener() {
251         public void actionPerformed(java.awt.event.ActionEvent evt) {
252             TextFieldTamActionPerformed(evt);
253         }
254     });
255     TextFieldTam.addKeyListener(new java.awt.event.KeyAdapter() {
256         public void keyPressed(java.awt.event.KeyEvent evt) {
257             TextFieldTamKeyPressed(evt);
258         }
259         public void keyTyped(java.awt.event.KeyEvent evt) {
260             TextFieldTamKeyTyped(evt);
261         }
262     });
263
264     UpdateButton.setBackground(new java.awt.Color(59, 102, 3));
265     UpdateButton.setForeground(new java.awt.Color(255, 255, 255));
266     UpdateButton.setText("Editar");
267     UpdateButton.addActionListener(new java.awt.event.ActionListener() {
268         public void actionPerformed(java.awt.event.ActionEvent evt) {
269             UpdateButtonActionPerformed(evt);
270         }
271     });
272
273     DeleteButton.setBackground(new java.awt.Color(59, 102, 3));

```

```

274 DeleteButton.setForeground(new java.awt.Color(255, 255, 255));
275 DeleteButton.setText("Excluir");
276 DeleteButton.addActionListener(new java.awt.event.ActionListener() {
277     public void actionPerformed(java.awt.event.ActionEvent evt) {
278         DeleteButtonActionPerformed(evt);
279     }
280 });
281
282 InsertButton.setBackground(new java.awt.Color(59, 102, 3));
283 InsertButton.setForeground(new java.awt.Color(255, 255, 255));
284 InsertButton.setText("Cadastrar");
285 InsertButton.addActionListener(new java.awt.event.ActionListener() {
286     public void actionPerformed(java.awt.event.ActionEvent evt) {
287         InsertButtonActionPerformed(evt);
288     }
289 });
290
291 openImageButton.setBackground(new java.awt.Color(59, 102, 3));
292 openImageButton.setForeground(new java.awt.Color(255, 255, 255));
293 openImageButton.setText("Selecionar Imagem");
294 openImageButton.addActionListener(new java.awt.event.ActionListener() {
295     public void actionPerformed(java.awt.event.ActionEvent evt) {
296         openImageButtonActionPerformed(evt);
297     }
298 });
299
300 jTable1.setModel(new javax.swing.table.DefaultTableModel(
301     new Object [][] {
302
303     },
304     new String [] {
305         "ID", "Nome", "Tamanho", "blob"
306     }
307 ));
308 jTable1.setGridColor(new java.awt.Color(59, 102, 3));
309 jTable1.setSelectionBackground(new java.awt.Color(59, 102, 3));
310 jTable1.setSelectionForeground(new java.awt.Color(255, 255, 255));
311 jTable1.addMouseListener(new java.awt.event.MouseAdapter() {
312     public void mouseClicked(java.awt.event.MouseEvent evt) {
313         jTable1MouseClicked(evt);
314     }
315 });
316 jScrollPane1.setViewportView(jTable1);
317

```

```

318 QuickButton.setBackground(new java.awt.Color(59, 102, 3));
319 QuickButton.setForeground(new java.awt.Color(255, 255, 255));
320 QuickButton.setText("Quick Sort");
321 QuickButton.addActionListener(new java.awt.event.ActionListener() {
322     public void actionPerformed(java.awt.event.ActionEvent evt) {
323         QuickButtonActionPerformed(evt);
324     }
325 });
326
327 jLabel3.setFont(new java.awt.Font("Segoe UI Semibold", 0, 14)); // NOI18N
328 jLabel3.setForeground(new java.awt.Color(59, 102, 3));
329 jLabel3.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
330 jLabel3.setText("Ordena es");
331
332 MergeButton.setBackground(new java.awt.Color(59, 102, 3));
333 MergeButton.setForeground(new java.awt.Color(255, 255, 255));
334 MergeButton.setText("Merge Sort");
335 MergeButton.addActionListener(new java.awt.event.ActionListener() {
336     public void actionPerformed(java.awt.event.ActionEvent evt) {
337         MergeButtonActionPerformed(evt);
338     }
339 });
340
341 HeapButton.setBackground(new java.awt.Color(59, 102, 3));
342 HeapButton.setForeground(new java.awt.Color(255, 255, 255));
343 HeapButton.setText("Heap Sort");
344 HeapButton.addActionListener(new java.awt.event.ActionListener() {
345     public void actionPerformed(java.awt.event.ActionEvent evt) {
346         HeapButtonActionPerformed(evt);
347     }
348 });
349
350 jLabel4.setFont(new java.awt.Font("Segoe UI Semibold", 1, 14)); // NOI18N
351 jLabel4.setForeground(new java.awt.Color(59, 102, 3));
352 jLabel4.setText("CADASTRE, EDITE OU EXCLUA IMAGENS");
353
354 timeLabel.setText(" ");
355
356 jLabel1.setText("Nome");
357
358 timeText.setHorizontalAlignment(javax.swing.SwingConstants.CENTER);
359
360 reloadDatabase.setBackground(new java.awt.Color(59, 102, 3));
361 reloadDatabase.setForeground(new java.awt.Color(255, 255, 255));

```

```

362 reloadDatabase.setText("Recargar Banco");
363 reloadDatabase.addActionListener(new java.awt.event.ActionListener() {
364     public void actionPerformed(java.awt.event.ActionEvent evt) {
365         reloadDatabaseActionPerformed(evt);
366     }
367 });
368
369 javax.swing.GroupLayout layout = new javax.swing.GroupLayout(getContentPane());
370 getContentPane().setLayout(layout);
371 layout.setHorizontalGroup(
372     layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
373     .addGroup(layout.createSequentialGroup()
374         .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
375             .addGroup(layout.createSequentialGroup()
376                 .addGap(27, 27, 27)
377                 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
378                     .addGroup(layout.createSequentialGroup()
379                         .addComponent(insertButton)
380                         .addGap(18, 18, 18)
381                         .addComponent(updateButton)
382                         .addGap(18, 18, 18)
383                         .addComponent(deleteButton)
384                         .addGap(18, 18, 18)
385                         .addComponent(openImageButton))
386                     .addComponent(jSeparator1, javax.swing.GroupLayout.PREFERRED_SIZE, 408, javax.swing.GroupLayout.PREFERRED_SIZE))
387                 .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE, 408, javax.swing.GroupLayout.PREFERRED_SIZE)
388                 .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.TRAILING, false)
389                     .addGroup(layout.createSequentialGroup()
390                         .addGap(241, 241, 241)
391                         .addComponent(imageLabel))
392                     .addGroup(javax.swing.GroupLayout.Alignment.LEADING, layout.createSequentialGroup()
393                         .addGap(9, 9, 9)
394                         .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
395                             .addComponent(jLabel2)
396                             .addComponent(jLabel1)))

```

```

397         .addGap(18, 18, 18)
398         .addGroup(layout.createParallelGroup(javax.swing.
           GroupLayout.Alignment.LEADING, false)
399         .addComponent(TextFieldTam, javax.swing.
           GroupLayout.DEFAULT_SIZE, 145, Short.
           MAX_VALUE)
400         .addComponent(TextFieldName))
401         .addGap(0, 0, Short.MAX_VALUE))))
402     .addGroup(layout.createSequentialGroup())
403     .addGap(96, 96, 96)
404     .addComponent(jLabel4))
405     .addGroup(layout.createSequentialGroup())
406     .addGap(47, 47, 47)
407     .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.
           Alignment.LEADING)
408     .addComponent(jLabel5, javax.swing.GroupLayout.
           PREFERRED_SIZE, 348, javax.swing.GroupLayout.
           PREFERRED_SIZE)
409     .addGroup(layout.createSequentialGroup())
410     .addComponent(QuickButton)
411     .addGap(50, 50, 50)
412     .addGroup(layout.createParallelGroup(javax.swing.
           GroupLayout.Alignment.TRAILING)
413     .addComponent(jLabel3, javax.swing.GroupLayout.
           PREFERRED_SIZE, 89, javax.swing.GroupLayout.
           PREFERRED_SIZE)
414     .addComponent(MergeButton))
415     .addGap(42, 42, 42)
416     .addComponent(HeapButton))
417     .addComponent(timeText, javax.swing.GroupLayout.
           PREFERRED_SIZE, 348, javax.swing.GroupLayout.
           PREFERRED_SIZE)
418     .addGroup(layout.createSequentialGroup())
419     .addGap(115, 115, 115)
420     .addComponent(reloadDatabase)
421     .addGap(26, 26, 26)
422     .addComponent(timeLabel))))))
423     .addContainerGap(25, Short.MAX_VALUE))
424 );
425 layout.setVerticalGroup(
426     layout.createParallelGroup(javax.swing.GroupLayout.Alignment.LEADING)
427     .addGroup(layout.createSequentialGroup())
428     .addGap(45, 45, 45)
429     .addComponent(jLabel4)

```

```

430         .addGap(18, 18, 18)
431     .addComponent(imageLabel)
432     .addGap(17, 17, 17)
433     .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
        BASELINE)
434         .addComponent(jLabel1)
435         .addComponent(TextFieldName, javax.swing.GroupLayout.PREFERRED_SIZE
            , javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout
                .PREFERRED_SIZE))
436     .addGap(23, 23, 23)
437     .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
        BASELINE)
438         .addComponent(jLabel2)
439         .addComponent(TextFieldTam, javax.swing.GroupLayout.PREFERRED_SIZE,
            javax.swing.GroupLayout.DEFAULT_SIZE, javax.swing.GroupLayout.
                PREFERRED_SIZE))
440     .addGap(39, 39, 39)
441     .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
        BASELINE)
442         .addComponent(InsertButton)
443         .addComponent(UpdateButton)
444         .addComponent>DeleteButton)
445         .addComponent(openImageButton))
446     .addGap(29, 29, 29)
447     .addComponent(jScrollPane1, javax.swing.GroupLayout.PREFERRED_SIZE,
        162, javax.swing.GroupLayout.PREFERRED_SIZE)
448     .addGap(35, 35, 35)
449     .addComponent(jSeparator1, javax.swing.GroupLayout.PREFERRED_SIZE, 10,
        javax.swing.GroupLayout.PREFERRED_SIZE)
450     .addGap(18, 18, 18)
451     .addComponent(jLabel3)
452     .addGap(18, 18, 18)
453     .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
        BASELINE)
454         .addComponent(QuickButton)
455         .addComponent(MergeButton)
456         .addComponent(HeapButton))
457     .addGap(27, 27, 27)
458     .addComponent(timeText)
459     .addGap(18, 18, 18)
460     .addComponent(jLabel5)
461     .addPreferredGap(javax.swing.LayoutStyle.ComponentPlacement.RELATED,
        37, Short.MAX_VALUE)

```

```

462         .addGroup(layout.createParallelGroup(javax.swing.GroupLayout.Alignment.
            BASELINE)
463             .addComponent(timeLabel)
464             .addComponent(reloadDatabase))
465         .addContainerGap(25, Short.MAX_VALUE))
466     );
467
468     pack();
469 } // </editor-fold>
470
471 /**
472  * Esta fun o aacionada quando o bot o "Editar" na interface pressionado.
473  * Verifica se uma linha na tabela est selecionada e obt m o ID do registro
    selecionado.
474  * Obt m os novos valores para o nome e tamanho do registro a ser atualizado.
475  * Realiza verifica es nos campos para garantir que o tamanho n o contenha
    letras e que nenhum campo esteja vazio.
476  * Chama a fun o atualizarRegistroNoBanco para atualizar o registro no banco de
    dados.
477  * Atualiza a tabela com os novos valores se a atualiza o for bem-sucedida.
478  */
479
480 private void UpdateButtonActionPerformed(java.awt.event.ActionEvent evt) {
481     DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
482
483     if (jTable1.getSelectedRow() != -1) {
484         int selectedRow = jTable1.getSelectedRow();
485         int id = (int) model.getValueAt(selectedRow, 0);
486         String novoNome = TextFieldName.getText();
487         String novoTamanho = TextFieldTam.getText();
488
489         if (containsLetter(novoTamanho)) {
490             JOptionPane.showMessageDialog(null, "A entrada cont m letras. Digite
                somente n meros inteiros.");
491         } else {
492             if (novoNome.length() == 0 || novoTamanho.length() == 0){
493                 JOptionPane.showMessageDialog(null, "Por Favor, n o deixe campos
                    vazios");
494             } else{
495                 if (novoTamanho.length() > 7 || novoTamanho.length() == 0){
496                     JOptionPane.showMessageDialog(null, "Por Favor, insira um
                        tamanho com no max 7 caracteres");
497                 } else{

```

```

499         if (atualizarRegistroNoBanco(id, novoNome, novoTamanho)) {
500             model.setValueAt(novoTamanho, selectedRow, 2); //
                    Suponhamos que a coluna 2 seja a coluna "Tamanho" na
                    tabela
501             JOptionPane.showMessageDialog(null, "Registro atualizado
                    com sucesso.");
502             carregarDadosDaTabela();
503         } else {
504             JOptionPane.showMessageDialog(null, "Erro ao atualizar o
                    registro no banco de dados.");
505         }
506     }
507 }
508 }
509 }
510 }
511
512 /**
513  * Esta fun o acionada quando o bot o "Excluir" na interface pressionado.
514  * Verifica se uma linha na tabela est selecionada e obt m o ID do registro
                    selecionado.
515  * Chama a fun o excluirRegistroDoBanco para excluir o registro do banco de
                    dados.
516  * Remove a linha da tabela se a exclus o for bem-sucedida.
517  */
518
519 private void DeleteButtonActionPerformed(java.awt.event.ActionEvent evt) {
520     DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
521
522     if (jTable1.getSelectedRow() != -1) {
523         int selectedRow = jTable1.getSelectedRow();
524         int id = (int) model.getValueAt(selectedRow, 0);
525
526         if (excluirRegistroDoBanco(id)) {
527             model.removeRow(selectedRow);
528             JOptionPane.showMessageDialog(null, "Registro exclu do com sucesso.");
529         } else {
530             JOptionPane.showMessageDialog(null, "Erro ao excluir o registro no
                    banco de dados.");
531         }
532     } else {
533         JOptionPane.showMessageDialog(null, "Selecione uma linha para ser exclu da
                    .");
534     } carregarDadosDaTabela();

```

```

535     }
536
537     /**
538      * Esta fun    o atualiza um registro no banco de dados com os novos valores de
539      * nome e tamanho.
540      * Executa uma consulta SQL para atualizar o registro com base no ID.
541      */
542     private boolean atualizarRegistroNoBanco(int id, String novoNome, String
543         novoTamanho) {
544         String sql = "UPDATE imagens SET nome = ?, tamanho = ? WHERE id = ?";
545         try (PreparedStatement statement = connection.prepareStatement(sql)) {
546
547             statement.setString(1, novoNome);
548             statement.setString(2, novoTamanho);
549             statement.setInt(3, id);
550
551             int rowsUpdated = statement.executeUpdate();
552             return rowsUpdated > 0;
553         } catch (SQLException e) {
554             e.printStackTrace();
555             return false;
556         }
557     }
558
559     /**
560      * Esta fun    o exclui um registro do banco de dados com base no ID fornecido.
561      * Executa uma consulta SQL de exclus o para remover o registro.
562      */
563     private boolean excluirRegistroDoBanco(int id) {
564         String sql = "DELETE FROM imagens WHERE id = ?";
565         try (PreparedStatement statement = connection.prepareStatement(sql)) {
566
567             statement.setInt(1, id);
568
569             int rowsDeleted = statement.executeUpdate();
570             return rowsDeleted > 0;
571         } catch (SQLException e) {
572             e.printStackTrace();
573             return false;
574         }
575     }
576

```

```

577  /**
578   * Esta fun o respons vel por exibir uma imagem na interface gr fica .
579   * Carrega uma imagem a partir de um array de bytes e a exibe em um r tulo na
        interface .
580   */
581
582  private void displayImage(byte[] imageData) {
583      try {
584          if (imageData != null && imageData.length > 0) {
585              BufferedImage img = ImageIO.read(new ByteArrayInputStream(imageData));
586              ImageIcon imagelcon = new ImageIcon(img);
587
588              // Exibe a imagem em um r tulo ou em qualquer outro componente Swing
589              JLabel imageLabel = new JLabel(imagelcon);
590
591              // Crie um di logo para exibir a imagem
592              JDialog dialog = new JDialog();
593              dialog.add(new JScrollPane(imageLabel));
594              dialog.pack();
595              dialog.setVisible(true);
596          }
597      } catch (IOException e) {
598          e.printStackTrace();
599      }
600  }
601
602  /**
603   * Esta fun o acionada quando o bot o "Selecionar Imagem" na interface
        pressionado .
604   * Abre uma caixa de di logo para o usu rio selecionar um arquivo de imagem .
605   * Carrega a imagem selecionada e a redimensiona para exibi o na interface .
606   */
607
608  private void openImageButtonActionPerformed(java.awt.event.ActionEvent evt) {
609      JFileChooser fileChooser = new JFileChooser();
610      FileNameExtensionFilter filter = new FileNameExtensionFilter("Image files", "
        jpg", "jpeg", "png");
611      fileChooser.setFileFilter(filter);
612
613      int returnValue = fileChooser.showOpenDialog(null);
614      if (returnValue == JFileChooser.APPROVE_OPTION) {
615          File selectedFile = fileChooser.getSelectedFile();
616          try {

```

```

617         ImageIcon originalImageIcon = new ImageIcon(selectedFile.
            getAbsolutePath());
618         Image originalImage = originalImageIcon.getImage();
619
620         Image scaledImage = originalImage.getScaledInstance(100, 100, Image.
            SCALE_SMOOTH);
621
622         ImageIcon scaledImageIcon = new ImageIcon(scaledImage);
623         imageLabel.getParent().setLayout(null);
624
625         int x = 310;
626         int y = 85;
627         int width = 100;
628         int height = 100;
629         imageLabel.setBounds(x, y, width, height);
630
631         imageLabel.setIcon(scaledImageIcon);
632     } catch (Exception ex) {
633         ex.printStackTrace();
634     }
635 }
636 }
637
638 /**
639  * Esta funcao e acionada quando o botao "Cadastrar" na interface
        pressionado.
640  * Obtém o nome, tamanho e dados da imagem a ser inserida no banco de dados.
641  * Realiza verificações nos campos para garantir que o tamanho não contenha
        letras e que nenhum campo esteja vazio.
642  * Chama a função inserirDadosNoBanco para inserir o novo registro no banco de
        dados.
643  * Adiciona os valores à tabela e limpa os campos de entrada.
644  */
645
646 private void InsertButtonActionPerformed(java.awt.event.ActionEvent evt) {
647     String nome = TextFieldName.getText();
648     String tamanho = TextFieldTam.getText();
649     byte[] imagem_blob = getImagemData();
650
651     if (containsLetter(tamanho)) {
652         JOptionPane.showMessageDialog(null, "A entrada contém letras. Digite
            somente números inteiros.");
653     } else {
654         if (nome.length() == 0 || tamanho.length() == 0){

```



```

655         JOptionPane.showMessageDialog(null, "Por Favor, n o deixe campos
           vazios");
656     } else{
657         if (tamanho.length() > 7){
658             JOptionPane.showMessageDialog(null, "Por Favor, insira um tamanho
           com no max 7 caracteres");
659         } else{
660             inserirDadosNoBanco(nome, tamanho, imagem_blob);
661
662             DefaultTableModel dtmClientes = (DefaultTableModel) jTable1.
           getModel();
663             Object[] dados = { nome, tamanho };
664             dtmClientes.addRow(dados);
665
666             // Limpe os JTextFields
667             TextFieldName.setText("");
668             TextFieldTam.setText("");
669
670             carregarDadosDaTabela();
671         }
672     }
673 }
674 }
675
676 /**
677  * Esta fun o acionada quando o usu rio clica em uma linha da tabela.
678  * Obt m os valores da linha selecionada e preenche os campos de entrada
           TextFieldName e TextFieldTam com esses valores.
679  */
680
681 private void jTable1MouseClicked(java.awt.event.MouseEvent evt) {
682     if (jTable1.getSelectedRow() != -1){
683         TextFieldName.setText(jTable1.getValueAt(jTable1.getSelectedRow(), 1).
           toString());
684         TextFieldTam.setText(jTable1.getValueAt(jTable1.getSelectedRow(), 2).
           toString());
685     }
686     else{
687         JOptionPane.showMessageDialog(null, "Selecione uma linha para ser alterada.
           ");
688     }
689 }
690
691 private void TextFieldTamActionPerformed(java.awt.event.ActionEvent evt) {

```

```

692         // TODO add your handling code here:
693     }
694
695     // MergeSort
696     // Dicionario
697     /**
698      * O MergeSort é um algoritmo de ordenação baseado na estratégia "dividir para
        conquistar". Ele divide o array em duas metades, ordena cada metade
        separadamente e depois mescla as duas metades ordenadas para obter o resultado
        final.
699      * A função mergeSort é a função principal que inicia o processo de
        ordenação.
700      * Ela divide o array repetidamente até que cada subarray contenha apenas um
        elemento e, em seguida, mescla esses subarrays para criar subarrays maiores
        até que o array inteiro esteja ordenado.
701     */
702     public void mergeSort(int[] array, int left, int right) {
703         if (right <= left) return;
704         int mid = (left+right)/2;
705         mergeSort(array, left, mid);
706         mergeSort(array, mid+1, right);
707         merge(array, left, mid, right);
708     }
709
710     void merge(int[] array, int left, int mid, int right) {
711         int n1 = mid - left + 1;
712         int n2 = right - mid;
713
714         int L[] = new int[n1];
715         int R[] = new int[n2];
716
717         for (int i=0; i<n1; ++i)
718             L[i] = array[left + i];
719         for (int j=0; j<n2; ++j)
720             R[j] = array[mid + 1+ j];
721
722         int i = 0, j = 0;
723
724         int k = left;
725         while (i < n1 && j < n2) {
726             if (L[i] <= R[j]) {
727                 array[k] = L[i];
728                 i++;
729             } else {

```

```

730         array[k] = R[j];
731         j++;
732     }
733     k++;
734 }
735
736 while (i < n1) {
737     array[k] = L[i];
738     i++;
739     k++;
740 }
741
742 while (j < n2) {
743     array[k] = R[j];
744     j++;
745     k++;
746 }
747 }
748
749 // QuickSort
750 // Video na aula
751 /**
752  * O QuickSort tamb m   um algoritmo de ordena o "dividir para conquistar".
753  * Ele escolhe um elemento chamado "piv " do array e reorganiza os elementos de
754  * forma que todos os elementos menores que o piv  estejam   esquerda e todos
755  * os elementos maiores estejam   direita.
756  * A fun o quickSort   a fun o principal que inicia o processo de
757  * ordena o.
758  * Ela seleciona um piv  e chama a fun o partition para rearranjar os elementos
759  * de forma que o piv  esteja na posi o correta e, em seguida, ordena as
760  * parti es esquerda e direita recursivamente.
761  */
762 public void quickSort(int[] array, int low, int high) {
763     if (low < high) {
764         int pi = partition(array, low, high);
765
766         quickSort(array, low, pi-1);
767         quickSort(array, pi+1, high);
768     }
769 }
770
771 int partition(int[] array, int low, int high) {
772     int pivot = array[high];
773     int i = (low-1);

```

```

768     for (int j=low; j<high; j++) {
769         if (array[j] < pivot) {
770             i++;
771             int temp = array[i];
772             array[i] = array[j];
773             array[j] = temp;
774         }
775     }
776
777     int temp = array[i+1];
778     array[i+1] = array[high];
779     array[high] = temp;
780
781     return i+1;
782 }
783
784 //heapSort
785 //arvore binaria
786 /**
787  * O HeapSort é um algoritmo que utiliza uma estrutura de dados chamada "heap" (
788    uma árvore binária especial) para classificar elementos em um array.
789  * A função HeapSort começa construindo um heap máximo (uma estrutura de heap
790    em que o pai é maior do que seus filhos) e depois reorganiza o array para
791    classificar os elementos em ordem crescente.
792  * A função heapify é usada para manter a propriedade do heap máximo enquanto
793    constrói o heap e reorganiza o array.
794  */
795
796 public void HeapSort(int[] arr) {
797     int n = arr.length;
798     for (int i = n / 2 - 1; i >= 0; i--)
799         heapify(arr, n, i);
800     for (int i=n-1; i>=0; i--) {
801         int temp = arr[0];
802         arr[0] = arr[i];
803         arr[i] = temp;
804         heapify(arr, i, 0);
805     }
806 }
807
808 void heapify(int[] arr, int n, int i) {
809     int largest = i;
810     int l = 2*i + 1;
811     int r = 2*i + 2;

```

```

808         if (l < n && arr[l] > arr[largest])
809             largest = l;
810         if (r < n && arr[r] > arr[largest])
811             largest = r;
812         if (largest != i) {
813             int swap = arr[i];
814             arr[i] = arr[largest];
815             arr[largest] = swap;
816             heapify(arr, n, largest);
817         }
818     }
819
820     /** (Valido para as 3 proximas funcoes)
821     * Primeiro, obt m o modelo da tabela de dados existente.
822     * Calcula o nmero de linhas na tabela.
823     * Cria um array chamado tamanhos para armazenar os valores a serem ordenados.
824     * Itera pelas linhas da tabela para obter os valores de tamanho (coluna 2) e os
825     armazena no array tamanhos.
826     * Registra o tempo inicial.
827     * Chama a uma funcao de ordenacao para ordenar o array tamanhos.
828     * Registra o tempo final.
829     * Calcula o tempo total gasto na ordenacao e o formata.
830     * Cria um novo modelo de tabela chamado newModel.
831     * Preenche esse novo modelo com os dados da tabela original, mas na ordem
832     especificada pelos valores ordenados em tamanhos.
833     */
834     private void QuickButtonActionPerformed(java.awt.event.ActionEvent evt) {
835         DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
836         int rowCount = model.getRowCount();
837         int[] tamanhos = new int[rowCount];
838
839
840         for (int i = 0; i < rowCount; i++) {
841             Object tamanhoObj = model.getValueAt(i, 2);
842             if (tamanhoObj != null) {
843                 String tamanhoStr = tamanhoObj.toString();
844                 try {
845                     int tamanho = Integer.parseInt(tamanhoStr);
846                     tamanhos[i] = tamanho;
847                 } catch (NumberFormatException e) {
848                     tamanhos[i] = 0;
849                 }

```

```

850     }
851 }
852     for (int i = 0; i < rowCount; i++) {
853         String tamanhoStr = model.getValueAt(i, 2).toString();
854         int tamanho;
855         try {
856             tamanho = Integer.parseInt(tamanhoStr);
857         } catch (NumberFormatException e) {
858             tamanho = 0;
859         }
860         tamanhos[i] = tamanho;
861     }
862
863     long tempolnicial = System.currentTimeMillis();
864     quickSort(tamanhos, 0, rowCount - 1);
865     long tempoFinal = System.currentTimeMillis();
866
867     long tempoTotal = tempoFinal - tempolnicial;
868
869     long minutos = (tempoTotal / 60000) % 60;
870     long segundos = (tempoTotal / 1000) % 60;
871     long milissegundos = tempoTotal % 1000;
872
873     String tempoFormatado = String.format("%02d:%02d:%03d", minutos, segundos,
874         milissegundos);
875
876     DefaultTableModel newModel = new DefaultTableModel();
877
878     newModel.addColumn("ID");
879     newModel.addColumn("Nome");
880     newModel.addColumn("Tamanho");
881     newModel.addColumn("Imagem Blob");
882
883     for (int i = 0; i < rowCount; i++) {
884         for (int j = 0; j < rowCount; j++) {
885             if (tamanhos[i] == Integer.parseInt(model.getValueAt(j, 2).toString()))
886             {
887                 newModel.addRow(new Object[]{model.getValueAt(j, 0), model.
888                     getValueAt(j, 1), model.getValueAt(j, 2), model.getValueAt(j,
889                     3)});
887                 break;
888             }
889         }
889     }

```

```

890     }
891
892     jTable1.setModel(newModel);
893
894     JOptionPane.showMessageDialog(null, "Tempo gasto na ordenação: " +
        tempoFormatado);
895 }
896
897 private void MergeButtonActionPerformed(java.awt.event.ActionEvent evt) {
898     DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
899     int rowCount = model.getRowCount();
900
901     int[] tamanhos = new int[rowCount];
902
903     for (int i = 0; i < rowCount; i++) {
904         String tamanhoStr = model.getValueAt(i, 2).toString();
905         int tamanho;
906         try {
907             tamanho = Integer.parseInt(tamanhoStr);
908         } catch (NumberFormatException e) {
909             tamanho = 0;
910         }
911         tamanhos[i] = tamanho;
912     }
913
914     long tempoInicial = System.currentTimeMillis();
915     mergeSort(tamanhos, 0, rowCount - 1);
916     long tempoFinal = System.currentTimeMillis();
917
918     long tempoTotal = tempoFinal - tempoInicial;
919
920     long minutos = (tempoTotal / 60000) % 60;
921     long segundos = (tempoTotal / 1000) % 60;
922     long milissegundos = tempoTotal % 1000;
923
924     String tempoFormatado = String.format("%02d:%02d:%03d", minutos, segundos,
        milissegundos);
925
926
927     DefaultTableModel newModel = new DefaultTableModel();
928
929     newModel.addColumn("ID");
930     newModel.addColumn("Nome");
931     newModel.addColumn("Tamanho");

```

```

932     newModel.addColumn("Imagem Blob");
933
934     for (int i = 0; i < rowCount; i++) {
935         for (int j = 0; j < rowCount; j++) {
936             if (tamanhos[i] == Integer.parseInt(model.getValueAt(j, 2).toString()))
937                 {
938                     newModel.addRow(new Object[]{ model.getValueAt(j, 0), model.
939                         getValueAt(j, 1), model.getValueAt(j, 2), model.getValueAt(j,
940                             3)});
941                     break;
942                 }
943             }
944         }
945     }
946
947     jTable1.setModel(newModel);
948     JOptionPane.showMessageDialog(null, "Tempo gasto na ordena o: " + tempoFormatado)
949     ;
950 }
951
952 private void HeapButtonActionPerformed(java.awt.event.ActionEvent evt) {
953     DefaultTableModel model = (DefaultTableModel) jTable1.getModel();
954     int rowCount = model.getRowCount();
955     int[] tamanhos = new int[rowCount];
956
957     for (int i = 0; i < rowCount; i++) {
958         String tamanhoStr = model.getValueAt(i, 2).toString();
959         int tamanho;
960         try {
961             tamanho = Integer.parseInt(tamanhoStr);
962         } catch (NumberFormatException e) {
963             tamanho = 0;
964         }
965         tamanhos[i] = tamanho;
966     }
967
968     long tempolnicial = System.currentTimeMillis();
969     HeapSort(tamanhos);
970     long tempoFinal = System.currentTimeMillis();
971
972     long tempoTotal = tempoFinal - tempolnicial;
973
974     long minutos = (tempoTotal / 60000) % 60;
975     long segundos = (tempoTotal / 1000) % 60;
976     long milissegundos = tempoTotal % 1000;

```



```

972
973     String tempoFormatado = String.format("%02d:%02d:%03d", minutos, segundos,
974         milissegundos);
975
976     DefaultTableModel newModel = new DefaultTableModel();
977     newModel.addColumn("ID");
978     newModel.addColumn("Nome");
979     newModel.addColumn("Tamanho");
980     newModel.addColumn("blob");
981
982     for (int i = 0; i < rowCount; i++) {
983         for (int j = 0; j < rowCount; j++) {
984             if (tamanhos[i] == Integer.parseInt(model.getValueAt(j, 2).toString()))
985             {
986                 newModel.addRow(new Object[]{ model.getValueAt(j, 0), model.
987                     getValueAt(j, 1), model.getValueAt(j, 2), model.getValueAt(j,
988                         3)});
989                 break;
990             }
991         }
992     }
993
994     jTable1.setModel(newModel);
995     JOptionPane.showMessageDialog(null, "Tempo gasto na ordena o: " +
996         tempoFormatado);
997
998 }
999
1000
1001 private void TextFieldTamKeyPressed(java.awt.event.KeyEvent evt) {
1002     // TODO add your handling code here:
1003 }
1004
1005 private void TextFieldTamKeyTyped(java.awt.event.KeyEvent evt) {
1006     // TODO add your handling code here:
1007 }
1008
1009 private void reloadDatabaseActionPerformed(java.awt.event.ActionEvent evt) {
1010     carregarDadosDaTabela();
1011 }
1012
1013 /**
1014  * @param args the command line arguments
1015  */
1016 public static void main(String args[]) {

```

```

1011      /* Set the Nimbus look and feel */
1012      //<editor-fold defaultstate="collapsed" desc=" Look and feel setting code (
1013      optional) ">
1014      /* If Nimbus (introduced in Java SE 6) is not available, stay with the default
1015      look and feel.
1016      * For details see http://download.oracle.com/javase/tutorial/uiswing/
1017      lookandfeel/plaf.html
1018      */
1019      try {
1020          for (javax.swing.UIManager.LookAndFeelInfo info : javax.swing.UIManager.
1021              getInstalledLookAndFeels()) {
1022              if ("Nimbus".equals(info.getName())) {
1023                  javax.swing.UIManager.setLookAndFeel(info.getClassName());
1024                  break;
1025              }
1026          }
1027      } catch (ClassNotFoundException ex) {
1028          java.util.logging.Logger.getLogger(FrameInicial.class.getName()).log(java.
1029              util.logging.Level.SEVERE, null, ex);
1030      } catch (InstantiationException ex) {
1031          java.util.logging.Logger.getLogger(FrameInicial.class.getName()).log(java.
1032              util.logging.Level.SEVERE, null, ex);
1033      } catch (IllegalAccessException ex) {
1034          java.util.logging.Logger.getLogger(FrameInicial.class.getName()).log(java.
1035              util.logging.Level.SEVERE, null, ex);
1036      } catch (javax.swing.UnsupportedLookAndFeelException ex) {
1037          java.util.logging.Logger.getLogger(FrameInicial.class.getName()).log(java.
1038              util.logging.Level.SEVERE, null, ex);
1039      }
1040      //</editor-fold>
1041
1042      /* Create and display the form */
1043      java.awt.EventQueue.invokeLater(new Runnable() {
1044          public void run() {
1045              new FrameInicial().setVisible(true);
1046          }
1047      });
1048  }
1049
1050  // Variables declaration - do not modify
1051  private javax.swing.JButton DeleteButton;
1052  private javax.swing.JButton HeapButton;
1053  private javax.swing.JButton InsertButton;
1054  private javax.swing.JButton MergeButton;

```

```

1047     private javax.swing.JButton QuickButton;
1048     private javax.swing.JTextField TextFieldName;
1049     private javax.swing.JTextField TextFieldTam;
1050     private javax.swing.JButton UpdateButton;
1051     private javax.swing.JLabel imageLabel;
1052     private javax.swing.JLabel jLabel1;
1053     private javax.swing.JLabel jLabel2;
1054     private javax.swing.JLabel jLabel3;
1055     private javax.swing.JLabel jLabel4;
1056     private javax.swing.JLabel jLabel5;
1057     private javax.swing.JProgressBar jProgressBar1;
1058     private javax.swing.JScrollPane jScrollPane1;
1059     private javax.swing.JSeparator jSeparator1;
1060     private javax.swing.JTable jTable1;
1061     private javax.swing.JButton openImageButton;
1062     private javax.swing.JButton reloadDatabase;
1063     private javax.swing.JLabel timeLabel;
1064     private javax.swing.JLabel timeText;
1065     // End of variables declaration
1066 }

```

7.3 Python

Código 3: codigo.py

```

1  import os
2  import random
3  import mysql.connector
4  from PIL import Image
5  import time
6
7  # Função para gerar um número aleatório exclusivo
8  def generate_unique_random(existing_values):
9      while True:
10         random_num = random.randint(1, 10000000)
11         if random_num not in existing_values:
12             existing_values.add(random_num)
13         return random_num
14
15 # Função para redimensionar e comprimir uma imagem
16 def redimensionar_comprimir_imagem(imagem, largura, altura):
17     imagem.thumbnail((largura, altura), Image.ANTIALIAS)
18     return imagem
19

```

```

20 # Conectar ao banco de dados MySQL
21 connection = mysql.connector.connect(
22     host='localhost',
23     database='meu banco de dados',
24     user='meu usuario',
25     password='minha senha'
26 )
27 cursor = connection.cursor()
28
29 # Pasta com as imagens
30 pasta_imagens = 'caminho/da/minha/pasta/de/imagens'
31
32 # Tamanho desejado para redimensionar as imagens
33 largura_desejada = 800
34 altura_desejada = 600
35
36 # Lista para rastrear n meros aleat rios j usados
37 numeros_aleatorios_usados = set()
38
39 contador = 0
40
41 # Loop pelas imagens na pasta
42 for _ in range(300):
43     for filename in os.listdir(pasta_imagens):
44         if filename.endswith((''.jpg', '.png', '.jpeg')):
45             with Image.open(os.path.join(pasta_imagens, filename)) as img:
46                 # Redimensionar a imagem, mantendo a propor o
47                 max_size = (200, 200)
48                 img.thumbnail(max_size, Image.LANCZOS)
49
50                 # Salvar a imagem otimizada em um buffer
51                 from io import BytesIO
52                 buffer = BytesIO()
53                 img.save(buffer, format="JPEG", quality=85)
54                 image_binary = buffer.getvalue()
55
56                 # Gere um n mero aleat rio exclusivo
57                 tamanho_aleatorio = generate_unique_random(numeros_aleatorios_usados)
58
59                 cursor.execute("INSERT INTO imagens (nome, imagem_blob, tamanho) VALUES (%s
60                                     , %s, %s)",
61                                     (filename, image_binary, tamanho_aleatorio))
62                 connection.commit()
63 # Feche a conex o com o banco de dados

```

```
63 cursor.close()
```

```
64 connection.close()
```

8 Bibliografia

[MySQL :: MySQL 8.0 Reference Manual :: 1.2.1 What is MySQL?]

(<https://dev.mysql.com/doc/refman/8.0/en/what-is-mysql.html>)

[Merge Sort - Data Structure and Algorithms Tutorials - GeeksforGeeks]

(<https://www.geeksforgeeks.org/merge-sort/>)

[QuickSort (With Code in Python/C++/Java/C) (programiz.com)]

(<https://www.programiz.com/dsa/quick-sort>)

[Heap Sort - Data Structures and Algorithms Tutorials - GeeksforGeeks]

(<https://www.geeksforgeeks.org/heap-sort/>)

[Heap Sort (With Code in Python, C++, Java and C) (programiz.com)]

(<https://www.programiz.com/dsa/heap-sort>)

[Quicksort algorithm overview | Quick sort (article) | Khan Academy]

(<https://www.khanacademy.org/computing/computer-science/algorithms/quick-sort/a/overview-of-quicksort>)

[Various Licenses and Comments about Them - GNU Project - Free Software Foundation]

(<https://www.gnu.org/licenses/license-list.html>)

[Java Programming Language (oracle.com)]

(<https://docs.oracle.com/javase/8/docs/technotes/guides/language/index.html>)

[MySQL :: MySQL 8.0 Reference Manual :: 1.2 Overview of the MySQL Database Management System]

(<https://dev.mysql.com/doc/refman/8.0/en/what-is.html>)

FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Atividades Práticas Supervisionadas (laboratórios, atividades em biblioteca, Iniciação Científica, trabalhos Individuais e em grupo, práticas de ensino e outras)

NOME: Gabriela Araújo Carneiro da Silva**RA:** N8397D-0 **CURSO:** Ciência da Computação**CAMPUS:** Flamboyant - Goiânia **SEMESTRE:** 4º Semestre **TURNO:** Noturno

			ASSINATURA	
DATA	ATIVIDADE	TOTAL DE HORAS	ALUNO	PROFESSOR
10/09/2023	Pesquisas de ordenações	5 horas	Gabriela Araujo C da Silva	
12/09/2023	Criação de códigos de ordenação MergeSort	3 horas	Gabriela Araujo C da Silva	
13/09/2023	Criação de códigos de ordenação QuickSort	6 horas	Gabriela Araujo C da Silva	
17/09/2023	Criação de códigos de ordenação HeapSort	4 horas	Gabriela Araujo C da Silva	
19/09/2023	Implementação do código de calcular tempo de ordenação	7 horas	Gabriela Araujo C da Silva	
20/09/2023	Criação do banco de dados	2 horas	Gabriela Araujo C da Silva	
23/09/2023	Código para converter imagem em binário	5 horas	Gabriela Araujo C da Silva	
24/09/2023	Construção da parte visual	6 horas	Gabriela Araujo C da Silva	
27/09/2023	Construção da parte visual	8 horas	Gabriela Araujo C da Silva	
28/09/2023	Construção da parte visual	3 horas	Gabriela Araujo C da Silva	
29/09/2023	CRUD - Created	4 horas	Gabriela Araujo C da Silva	
01/10/2023	CRUD - Read	7 horas	Gabriela Araujo C da Silva	
04/10/2023	CRUD - Updated	5 horas	Gabriela Araujo C da Silva	
06/10/2023	CRUD - Deleted	6 horas	Gabriela Araujo C da Silva	
07/10/2023	Salvamento de imagens em uma pasta para popular tabela	9 horas	Gabriela Araujo C da Silva	
11/10/2023	Últimos ajustes	3 horas	Gabriela Araujo C da Silva	
16/10/2023	Últimos ajustes	4 horas	Gabriela Araujo C da Silva	
20/10/2023	Introdução da monografia	5 horas	Gabriela Araujo C da Silva	
23/10/2023	Desenvolvimento da monografia	6 horas	Gabriela Araujo C da Silva	
24/10/2023	Finalização da monografia	2 horas	Gabriela Araujo C da Silva	

TOTAL DE HORAS: 105 horas