UNIVERSIDADE PAULISTA

B22816-4 MARCIO FERNANDES CRUZ
B54ECG-9 MAURICIO JOSE F OLIVEIRA
T618FA-6 RENATO GOULART RODRIGUES
B20309-9 RODOLFO YURI DE A FONTANA

Aplicação de métodos de ordenação em imagens geradas por satélites

SÃO PAULO

2013

B22816-4 MARCIO FERNANDES CRUZ

B54ECG-9 MAURICIO JOSE F OLIVEIRA

T618FA-6 RENATO GOULART RODRIGUES

B20309-9 RODOLFO YURI DE A FONTANA

Aplicação de métodos de ordenação em imagens geradas por satélites

Dados obtidos através dos métodos e aplicações estudados em aulas e com pesquisas para fins de obtenção de nota para o curso de Ciência da Computação 2/3° Semestre (Noturno) da Universidade Paulista (UNIP) sendo entrega ao Professor/Orientador

Ciência da Computação 2° e 3º Semestre – Noturno

Professor Alan.

SÃO PAULO

2013

Guia de normalização para apresentação de trabalhos acadêmicos da Universidade Paulista. / Ana Lúcia E. Pires... [et al]. – São Paulo, 2012

.

SUMÁRIO

OBJETIVO DO TRABALHO

Este trabalho consiste em analisar alguns métodos de ordenação clássicos da Ciência da Computação, aplicados a ordenação em ordem crescente em objetos de imagens de satélite.

Estas imagens estão gravadas em disco e, foi criado um sistema computacional criando três vetores de imagens, sendo: vetores ordenado em ordem crescente, decrescente e embaralhado.

Em função de várias faixas de quantidade de objetos de imagens de satélite, foi-se aplicado seis métodos de ordenação, obtendo comparativos de eficiência em tempo.

Estes comparativos foram apresentados em três arquivos HTML, sendo um arquivo para imagens ordenada em forma crescente, decrescente e embaralhada.

Resumo

Os sistemas computacionais devem ser eficientes na obtenção dos vários tipos de Informações armazenadas. E, para isso, de uma forma ou de outra, estar informações devem estar ordenadas.

Seja um banco de dados, um gerenciador de arquivos ou mesmo um mecanismo de pesquisa de páginas de internet todos, sem exceção aplicam métodos de ordenação. Estes métodos estão encapsulados na visão do usuário final que geralmente sequer conhece a teoria por trás da busca de determinado item.

É atribuição da Ciência da Computação o estudo dos métodos de ordenação bem, como, o desenvolvimento de novos métodos, conforme for necessário, para atender a necessidade de algum modelo computacional.

Este trabalho descreve alguns métodos de ordenação, suas vantagens e desvantagens e apresenta um software desenvolvido em Java, onde o usuário pode lançar faixas de quantidades para que, consiga comparar a eficiência dos métodos em sequências embaralhadas, em ordem crescente e, no pior, ocaso, na ordem decrescente.

Os objetos a serem ordenados são 10.000 imagens catalogadas de satélite, da região amazônica e, o sistema computador criado, trabalha em função da ordenação destas imagens armazenadas em disco e, apresenta o resultado em páginas HTML com tabelas e gráficos de comparação.

ABSTRACT

The Computational system be efficient in obtain of many types of stored information. And, for that, one way or another, this information must be in order. Is a database, one file manager or one search engine of all webpages, without exception applies ordering in methods, these methods are encapsulated in vision of the final user, that usually even know the theory behind of search for a particular item.

That allocation of Computer Science, The study of ordering methods well, like, the development of new methods, as needed, to meet the necessity of any computational model.

this academy job describe some ordering methods, their advantages and disadvantages and presents a software developed in java, where the user can launch ranges of quantities for can compare the efficiency of methods in shuffled sequences, in ascending order and descending order.

the object to be ordering are 10.000 cataloged images of the amazon region obtained by satellite, and the computer system developed, works in function of the ordering of these stored images on disk, and show the result in HTML pages with tables and comparison graphics.

INTRODUÇÃO

A ordenação é um processo para organizar um conjunto de objetos em uma ordem crescente ou decrescente para que, em outro momento, a busca de determinada informação seja feita de maneira eficiente.

Para OLIVEIRA et. al (1999a) é de grande importância para os profissionais que atuam com Informática manter as informações ordenadas seguindo uma classificação previamente definida.

Na evolução da Computação foram e estão sendo desenvolvidas várias técnicas para ordenação, estas agrupadas em dois grandes grupos que são definidos como métodos de ordenação externa e interna.

Os métodos de ordenação externa se diferem do de ordenação interna pelo fato que, para se fazer o processo de ordenação, devido ao volume de dados, não é possível utilizar a memória principal do computador, no caso, definida como memória RAM ou volátil. Enquanto isso, os métodos de ordenação necessários conseguem efetuar o processo de ordenação utilização por completo servindo-se apenas da memória principal do computador.

Neste trabalho são discutidos os métodos de ordenação interno e, 'segundo OLIVEIRA et. al (2002), faz-se necessário agrupá-los em alguns grupos:

- Troca;
- Seleção;
- Inserção;
- Partição;
- Distribuição;
- Posição;
- Híbrido.

Este grupo de métodos tem suas características próprias e, cada um, agrupa uma ou mais técnicas de ordenação'.

Não podemos afirmar que existe uma técnica universal que pode ser utilizada para ordenação, pois, para cada situação em específica, deve ser aplicado um método. E de suma importância que o cientista da computação saiba propor, dentre as várias técnicas, uma que seja eficiente para resolver um determinado problema e, dependendo do problema a ser resolvido, faz-se necessário a criação ou utilização de algum modelo híbrido.

O grupo de técnicas de ordenação mais fácil de ser implementados é o de Troca. Para percorrer os conjuntos de dados, utiliza-se a técnica de iteração. De acordo com

OLIVEIRA et. al (1999[a]), estes tem a característica de serem feitos sucessivas trocas entre dois elementos da lista. Neste trabalho é abordado a técnica Bubble Short.

No grupo de métodos de ordenação do tipo Seleção, da mesma forma que o grupo de Troca, utiliza-se iteração para percorrer o conjunto de objetos. E, a cada iteração, é armazenado o índice do menor elemento que, é feito a passagem para a posição de contagem sequencial que é acrescida de um (um), a cada varredura no conjunto de objetos. Neste trabalho foi escolhida a técnica *Selection Sort.*

Segundo OLIVEIRA et. al (2002), o grupo de métodos chamados Inserção consiste em ordenar um subvetor localizado em seu início e, a cada novo passo, acrescenta-se a este subvetor, mais um elemento. Embora mais complexo de ser implementado e geralmente mais eficiente que os métodos anteriores listados, ainda ele utiliza a iteração para percorrer os objetos. Neste trabalho a técnica apresentada deste grupo é a *Insertion Sort.*

O grupo de Partição tem a característica de utilizar um recurso mais avançado do que a iteração, que é a recursividade. Seguindo a definição de SWAIT JR. (1991), este método consiste em particionar ou dividir os dados de um conjunto em dois ou mais subconjuntos e, ordenar cada um destes e, depois agrupar novamente. Neste trabalho são utilizados duas técnicas deste grupo, que são o *Merge Sort* e o *Quick Sort*.

Segundo VILLAS et. al (1993), o grupo de técnicas do grupo de Distribuição tem a característica de tratar *n* vezes os elementos de uma lista linear, criando ao final uma lista linear ordenada. O número de ciclos (n) é igual ao número de caracteres da chave. A técnica implementada neste trabalho é o *Radix Sort* que, não utiliza nenhuma forma de recursividade e, todas as comparações são feitas por iterações comuns, porém, utilizando-se de artifícios da computação como operador *bit a bit*.

O grupo de técnicas chamado de Posição, segundo OLIVEIRA et. al (2002), trabalha com as chaves de um conjunto de objetos, considerando que estas chaves são compostas por inteiros positivos.

Finalizando, temos o grupo de técnicas chamado de Híbrido que segundo SWAIT JR. (1991), tem a característica de combinar dois ou mais técnicas anteriormente apresentadas.

TEMA ESCOLHIDO

De acordo com a normalização interna da Universidade Paulista (UNIP), estudos disciplinares APS (Atividades Praticas Supervisionadas). Aplicação de métodos de ordenação em imagens geradas por satélites.

1. DISSERTAÇÃO

Este trabalho se baseia na ordenação em forma crescente de 10.000 imagens capturadas de satélite e apresentação dos dados comparativos de eficiência em tempo por faixas de quantidades de imagens processadas. Dentre inúmeros métodos existentes, foi implementado seis (seis) técnicas de ordenação extraídos da literatura de Ciência da Computação. Estas técnicas foram implementadas seguindo exatamente a definição do funcionamento dos mesmos. No final o software abre três páginas HTML que mostra os comparativos dos resultados. Abaixo são descritos os passos do desenvolvimento do artefato computacional.

1.1 Obtenção de imagens do satélite

O processo de obtenção das mais de 10.000 imagens não foi algo executado só por um processo ou software. Abaixo está enumerado o processo completo que indica desde a obtenção das latitudes dos limites da região da Amazônia brasileira até a gravação dos arquivos de imagem em disco.

1.1.1 Obtenção do arquivo KMZ

Arquivos KMZ contêm informações sobre regiões geográficas da Terra. E, o primeiro passo foi obter um arquivo deste tipo de toda a região amazônica. Foi acessado o *site* ***http://www.gmapas.com/poligonos-ibge/poligonos-ibge-municipios-amazonas*** em 29/04/2013, 19h e foi feito *download* do arquivo KMZ de todos os municípios da Unidade Federativa da Amazônia.

1.1.2 Obtenção das coordenadas geográficas

Através do software Global Mapper v.13.00 (***www.bluemarblegeo.com/***) e, foi aberto o arquivo KMZ obtido. Na opção *View->Properties*, este software nos deu informações importantes sobre os limites de Latitude e Longitude de onde devemos extrair as imagens. No caso, o software nos deu os números, conforme escrito abaixo:

- UPPER LEFT X=-76.4045834439
- UPPER LEFT Y=2.5482475125
- LOWER RIGHT X=-53.4945328561

- LOWER RIGHT Y=-10.1196628125
- WEST LONGITUDE=76° 24' 16.5004" W
- NORTH LATITUDE=2° 32' 53.6910" N
- EAST LONGITUDE=53° 29' 40.3183" W
- SOUTH LATITUDE=10° 07' 10.7861" S
- PIXEL SIZE X=0.0224608 arc degrees / pixel
- PIXEL SIZE Y=0.0224608 arc degrees / pixel
- SCALE=1:9433500
- ENCLOSED AREA=3588559 sq km
- VIEW PIXEL SIZE=1020 x 564

1.1.3 Obtenção dos arquivos 10.000 arquivos de imagem

Através do software Universal Maps Downloader (**http://www.softonpc.com/umd/**) foi definido as opções:

- Maps type: Yahoo Sattelite Maps
- Zoom Level: 12
- Left Longitude: -76.4045834439
- Right Longitude: -53.4945328561
- Top Latitude: 2.5482475125
- Bottom Latitude: -10.1196628125

Executando a opção *Start*, durante 4 horas, foi feito download de aproximadamente 10.500 arquivos de formato “JPG” de tamanho de aproximadamente 16 Kb da região Amazônica, conforme um exemplo abaixo. Estes arquivos foram armazenados numa pasta chamada imagens Satelite, de onde, o aplicativo vai varrer estas imagens para proceder à ordenação.

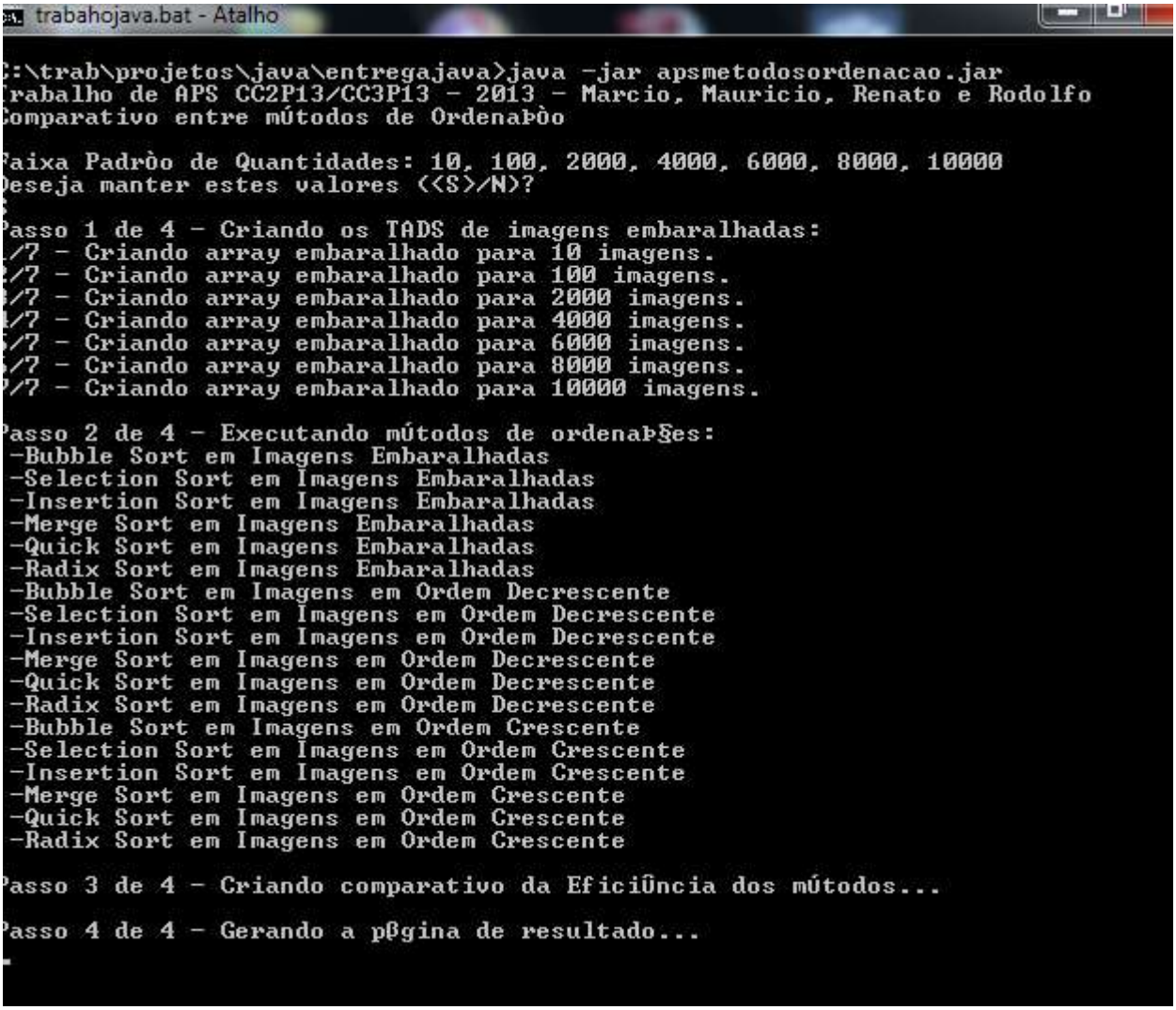
Imagem ys_589_1003_11.jpg

1.1.4 Funcionamento do sistema computacional

O programa está encapsulado em um arquivo jar e, este pode ser executado por qualquer computador que tenha uma JRE a partir da 1.7.

O aplicativo deve ser acessado via *console* e, apresenta uma imagem conforme abaixo:

Java-jar apsmetodosordenacao.jar



O sistema sugere uma faixa padrão de quantidades aos usuários e, o usuário pode clicar <ENTER> para continuar a execução ou teclar **N <ENTER>** e. Definir suas próprias faixas de quantidade para comparativo de resultados.

Foi deixada em aberto a possibilidade de o usuário lançar suas próprias quantidades para que, possa fazer um estudo dos métodos de ordenação em função de três tipos de vetores:

- Imagens de satélite embaralhadas;
- Imagens de satélite em ordem decrescente;
- Imagens de satélite em ordem crescente;

No final do processo o sistema abre três páginas HTML, com três quadros cada uma, sendo:

1.1.4.1 Aplicação de métodos de Ordenação em Imagens <<Ordem>>:

3.1.4.1

Aplicação de Métodos de Ordenação em Imagens Embaralhadas

Qtd. Imagens	5	10	15	20
Bubble Sort Tipo: Por Troca	20 µs	13 µs	19 µs	40 µs
Selection Sort Tipo: Por Seleção	23 µs	13 µs	24 µs	47 µs
Insertion Sort Tipo: Por Inserção	23 µs	13 µs	15 µs	32 µs
Merge Sort Tipo: Por Partição	34 µs	14 µs	21 µs	107 µs
Quick Sort Tipo: Por Partição	47 µs	13 µs	23 µs	30 µs
Radix Sort Tipo: Por Distribuicao	40 µs	37 µs	56 µs	59 µs

Eficiência dos métodos em tempo de processamento:

1.1.4.2 Gráfico entre

Eficiência dos métodos em Tempo de Processamento

5 imagens	10 imagens	15 imagens	20 imagens
Bubble Sort ↑ 15%	Bubble Sort	Insertion Sort ↑ 27%	Quick Sort ↑ 7%
Selection Sort	Insertion Sort	Bubble Sort ↑ 11%	Insertion Sort ↑ 25%
Insertion Sort ↑ 48%	Quick Sort	Merge Sort ↑ 10%	Bubble Sort ↑ 18%
Merge Sort ↑ 18%	Selection Sort ↑ 8%	Quick Sort ↑ 4%	Selection Sort ↑ 26%
Radix Sort ↑ 18%	Merge Sort ↑ 164%	Selection Sort ↑ 133%	Radix Sort ↑ 81%
Quick Sort	Radix Sort	Radix Sort	Merge Sort

quantidade de imagens e tempo de processamento

1.1.5 Estrutura dos pacotes e código-fontes

O sistema computacional foi dividido em sub-pacotes, a fim de facilitar a manutenção e ser escalonável para que, no futuro, possa ser implementado vários métodos.

1.1.5.1 Pacote de tipos enumerados

Organizamos alguns tipos enumerados em quatro classes. Estas são utilizadas em toda a estrutura do sistema.

1.1.5.1.1 Classificação de métodos

As técnicas de ordenação utilizadas no trabalho são do tipo Inserção, Partição, Seleção, Troca e Distribuição. Esta classe facilita a organização das técnicas no código-fonte.

1.1.5.1.2 Critério de Ordem

Todas as técnicas de ordenação são aplicadas em conjunto de objetos em ordem crescente, decrescente e embaralhada.

1.1.5.1.3 Tipo de Exibição dos gráficos

O sistema apresenta gráfico apenas por tempo de processamento, porém, o sistema é escalonável para que, em uma nova versão possa ser feita contagem por número de movimentações ou de comparações. Assim, teremos outras formas de comparações de resultado além de tempo.

1.1.5.1.4 Tipo de métodos de ordenação

O sistema tem implementado os seguintes métodos de Ordenação: Bubble Sort, Selection Sort, Insertion Sort, Merge Sort, Quick Sort e Radix Sort.

Esta classe organiza os métodos existentes e facilita a implementação de novas técnicas no futuro.

1.1.5.2 Pacote dos objetos de imagem

1.1.5.3 Classe ImagemSatelite

Esta classe representa cada arquivo de imagem JPG em disco, na criação do objeto, passa-se o nome do arquivo e, a classe gera uma chave, única, que representará a imagem no processo de ordenação.

1.1.5.4 Classe ArrayImagens

Esta classe recebe como parâmetro na criação o caminho do disco onde contém as imagens de satélite capturadas. Varre os arquivos em discos e cria três conjuntos de imagens de determinado diretório:

- Conjunto de imagens embaralhadas;
- Conjunto de imagens em ordem crescente;
- Conjunto de imagens em ordem decrescente.

1.1.5.3 Pacote das classes de ordenação

Neste pacote estão implementadas as seis técnicas criadas neste trabalho para apresentar os resultados de eficiência de ordenação em vetores de imagens crescentes, decrescentes e embaralhadas.

Existe uma classe principal chamada Ordenação que possui instruções comuns

que são compartilhadas com qualquer método. Recebe como parâmetro de entrada um vetor de objetos, não importa a ordem.

Esta classe ordenação tem um único método publico, chamado processar e, quem instancia uma classe do tipo ordenação, chama este método e tem seu resultado dos dados processados.

A classe que estende ordenação, pelo contrato, é obrigada a implementar um método chamado Ordenar () e, a lógica do negócio, de cada método, é feito dentro desta implementação.

Foram estendidas as seguintes classes de ordenação e, cada uma, faz o mesmo processo de ordenação de um objeto de imagens, sempre em ordem crescente. As técnicas implementadas foram:

- Bubble Sort;
- Insertion Sort;
- Merge Sort;
- Quick Sort;
- Radix Sort;
- Selection Sort.

Na classe abstrata Ordenacao, existe um método que é chamado para testar se o método implementado fez a ordenação correta, este método é o testarOrdenacaoArray (). Se por um acaso alguma nova implementação não fizer corretamente a ordenação, o sistema computacional levanta uma exceção ao desenvolvedor alertando do problema.

Os métodos de implementação mais simples são o Selection Sort, Bubble Sort e o Insertion Sort. Todos eles utilizam a técnica básica de iteração (loop) para percorrer o conjunto de objetos. Eles são eficientes para conjuntos de poucas imagens.

Os métodos de implementação com maior nível de complexidade são o Merge Sort e o Quick Sort, que utilizam o conceito de “dividir para conquistar” e, utilizam da técnica de recursividade, ou seja, o método implementado em Java chama ele mesmo para atingir o objetivo final da ordenação. Estes métodos se mostraram muito eficientes para conjunto de objetos de grande quantidade.

O método mais difícil a ser implementado foi o Radix Sort, pois se utiliza de recursos de operação bit a bit que não são técnicas triviais de implementação. Mesmo sendo o mais complexo a ser implementado, mesmo para grandes quantidades de imagens, não foi tão eficiente como o Quick Sort.

1.1.5.4 Pacote de apresentação dos resultados

O trabalho se preocupou em apresentar ao usuário os resultados da ordenação de uma forma prática e de fácil entendimento. Para isso, foram criadas cinco classes especiais para isso, definidas como: Resultado, ListaResultado, Grafico, AuxiliarHTML e ApresentacaoResultado.

Este conjunto de classes deste pacote apresenta no final do processamento três páginas de resultado, com comparativos de desempenho por quantidade de imagens e mostra as melhores técnicas para cada situação. A primeira página representa os resultados no processamento de imagens em ordem embaralhada. A segunda, os resultados no processamento em ordem crescente e, a terceira, em ordem decrescente.

Para a criação dos gráficos, utilizamos o software livre voltado para a plataforma Java chamado JfreeChart.

Para a criação do HTML, foi gerado HTML pelo próprio programa, utilizando-se técnicas de CSS para deixar zebrado as linhas.

A classe que faz a apresentação dos resultados e comparativos é a ApresentacaoResultado.

2. RADIX SORT

A origem desse algoritmo remonta aos tempos em que ocorriam tabulações de dados através de cartões perfumados, onde se utilizavam maquinas apropriadas para calculo de dados. Posteriormente, este tipo de cartões foi utilizado como entrada de dados em computadores digitais.

Para entender a estratégia desse método, será considerada a analogia para a ordenação de um conjunto de cartões. Sendo assim, é necessária a separação física desses cartões, onde eles são classificados por um campo numérico (chave) de um ou mais algoritmos.

Todos os cartões que terminassem em zero eram armazenados no escaninho de numero zero, os de numero um no escaninho um, e assim por diante. Depois disso, eram reorganizados em função do algarismo da dezena (penúltimo), e assim por diante, até sua ordenação.

Para implementar este algoritmo, utiliza-se um vetor de índices, indicando o numero do escaninho, que aponta para listas circulares, organizadas como pilhas.

2.1 Posição

Este método trabalha com as chaves de um vetor considerando que estas chaves soa compostas por números unteiros positivos entre 1 (Um) e X, com **n** elementos. Serão analisados os algoritimos:

Contagem;
Calculo de endereços;
Indireta.

2.2 Contagem

> A estratégia do método de ordenação por contagem consiste em determinar para cada valor em determinado endereço do vetor, o número de elementos que são menores que este valor, colocando-os na posição correta, tomando cuidado para não inserir valores iguais na mesma posição, Apud (Lopes 1999).

Este método Ultiliza duas áreas complementares para seu funcionamento, 1 (Uma) para armazenar os elementos ordenados e outra contendo n elementos. A ordenação por contagem estável, porque os números com menos valor aparecem na mesma ordem na área ordenada:

Exemplo

Considerando o vetor:

Origem [05,06,01,01,08,05]

O primeiro passo é criar o vetor Y, por exemplo, de forma que seu tamanho seja igual ao maior elemento do Vetor original . Para este exemplo, o maior elemento é o 08. Inicia-se o vetor Y inserindo valor 0 para cada posição:

Y [0,0,0,0,0,0,0,0]

O próximo passo é determinar o vetor Y [I], de modo a indicar um número de elementos iguais a I, ou seja, faz a contagem de ocorrência do vetor origem [I], colocando em Y [I]nas suas devidas posições :

Origem [**05**,06,01,01,08,05]	Y [0,0,0,0,0,0,0,0]
Origem [05,**06**,01,01,08,05]	[0,0,0,0,**1**,0,0,0]
Origem [05,06,**01**,01,08,05]	[0,0,0,0,1,**1**,0,0]
Origem [05,06,01,**01**,08,05]	[**1**,0,0,0,1,1,0,0]
Origem [05,06,01,01,**08**,05]	[**2**,0,0,0,1,1,0,0]
Origem [05,06,01,01,08,**05**]	[2,0,0,0,1,1,0,**1**]
	[2,0,0,0,**2**,1,0,1]

Agora ,se normaliza de forma que o vetor Y [I] contem o número de elementos menores ou iguais a I, começando da Segunda posição até o final do vetor, fazendo Y [I] receber Y [I] + Y [I - 1]:

Y [**02**,**0**,0,0,02,01,0,01]
[02,**02**,**0**,0,02,01,0,01]
[02,02,**02**,**0**,02,01,0,01]
[02,02,02,**02**,**02**,01,0,01]
[02,02,02,02,**04**,**01**,0,01]
[02,02,02,02,04,**05**,**0**,01]
[02,02,02,02,04,05,**05**,**01**]
[02,02,02,02,04,05,05,06]

Por fim,coloca cada elemento de origem [ ]em sua posição correta da seguinte forma:

Pos recebe origem [I] ;

Aux [Y [pos]] recebe pos;

Y [pos] recebe Y [pos] menos 1 (Um)

3. MERGE SORT

O algoritmo Merge Sort , também conhecido por intercalação,é um algoritmo recursivo que tem como finalidade combinar duas listas já ordenadas. O algoritmo quebra o vetor original em dois outros de tamanhos menores, recursivamente,até obter vetores de tamanho 1, depois retorna da recurção combinado os resultados.

Segundo VILLAS et al. (1993), esta técnica consiste em determinar, a cada passo, a menor chave dentre os vetores ordenados, e armazenar as informações correspondentes a essa chave no vetor resultante. A cada menor chave determinada, utilizando uma variável auxiliar para guarda-lá.

Existe a possibilidade de uma chave ser encontrada em um ou mais vetores. Nesse caso, as informações armazenadas no vetor resultante serão as existentes no último vetor onde ela foi encontrada. Contudo,essa decissão pode variar de acordo com a necessidade. Numa sitação, se pode acumular algum campo totalizador e gravá-la como variar de acordo com uma única ocorrência da chave em questão.Para o vetor resultante, reserva-se uma área igual á soma das áreas dos vetores ordenados, prevendo a possibilidade de não haver nenhuma chave duplicada. (ibidem).

Um dos principais problemas com o merge sort é que ele faz uso de um vetor auxiliar. O primeiro passo é dividir o vetor em blocos unutarios :

[70] [12] [20] [16] [10] [08] [05]

Depois, é feita a composição de pares de blocos,ordemando-os internamente, usando o metodo de troca:

[70,12] [20,16] [10,08] [05]

[12,70] [16,20] [08,10] [05]

Agora, o processo se repete com a composição de pares de blocos e ordenação interna. Detalhando a ordenação dentro do bloco, o processo de comparação se faz da seguinte forma:Utiliza-se dois vetores, um de origem e um auxiliar de destino, compare-se cada elemento inicial de ambos os blocos do vetor de origem, colocando o menor dele no de auxiliar.

VETOR ORIGEM [**12,**70] [**16**,20]

VETOR AUXILIAR [12,]

Segunda comparação:

VETOR ORIGEM [**70**] [**16**,20]

VETOR AUXILIAR [12,16]

Terçeira comparação

VETOR ORIGEM [**70**] [**20**]

VETOR AUXILIAR [12,16,20]

Neste exemplo, restou apenas 1 (Um) elemento, sendo que ele é transferido para o vetor auxiliar, resultando no bloco ordenado:

VETOR ORIGEM [**70**]

VETOR AUXILIAR [12,16,20,70]

Ao observar o vetor inteiro, há ainda três blocos:
[12,16,20,70] [08,10,] [05]

Começa-se a ordenação comparando 2 (Dois) blocos ultilizando 2 (Dois) vetores, 1 (Um) de origem e 1(Um) auxiliar de destino compara-se cada elemento inicial de ambos os blocos do vetor de origem, colocando o menor dele no auxiliar:
VETOR ORIGEM [**12**,16,20,70] [**08**,10]

VETOR AUXILIAR [08]

Como restou apenas 1 (Um) bloco, ele é transferido para o vetor auxiliar, resultando no bloco ordenado:

VETOR ORIGEM [12,16,20,70]

VETOR AUXILIAR [08,10,12,16,20,70]

Agora o vetor inteiro está da seguinte forma:

[08,10,12,16,20,70] [05]

Observa-se que ainda existem 2 (Dois) blocos. Faz-se, então ,as comparações resultando no vetor ordenado :

VETOR ORIGEM [**08**,10,12,16,20,70] [**05**]

VETOR AUXILIAR [05,08,10,12,16,20,70]

Insertion sort

Insertion sort (ordenação por inserção) é conhecido por ser um tipo de ordenação de implementação básica, esse método costuma ser o mais eficiente em pequenas quantidades de objetos e em métodos básicos (Troca e Inserção), mas para muitos objetos não é aconselhável porque há métodos mais eficientes.

4.1 Como funciona

Ele funciona da seguinte maneira ele analisa um vetor com elementos indo da esquerda para a direita e conforme ele vai analisando o vetor ele vai deixando os elementos de forma ordenada pré-definida, ou seja, crescente ou decrescente. Esse tipo de ordenação é o que vale ao mesmo que uma pessoa ordena cartas em um jogo de pôquer.

Exemplo:

7 – 5 – 9 – 6 –8

O método verifica qual dos elementos é o menor entre eles, após achar esse elemento ele efetua uma troca de posição com o primeiro elemento da sequência.

5 – 7 – 9 – 6 – 8

Então o método faz novamente a mesma operação igual à mencionada acima, só que dessa vez o elemento a ser usado como parâmetro é o segundo elemento.

5 – 6 – 9 – 7 – 8

Novamente o método volta a fazer o mesmo procedimento até que não existam elementos menores que o elemento anterior.

5 – 6 – 7 – 8 – 9

Outro exemplo desse método é escolher o 1° elemento e ir testando elemento por elemento até o último elemento e assim ir trocando cada vez que encontrar um elemento menor que o escolhido (1° elemento). Em seguida se efetua a mesma operação até o último elemento a ser ordenado e assim finaliza o processo.

4.2 **Tempo de execução:**

O seu tempo de execução depende de algumas coisas:

A entrada: Se o início já estiver ordenado o Insertion fara o mínimo de trabalho porque já não necessita mais iniciar um looping interno.

Tamanho da entrada: seu tempo para ordenar 1000 elementos é diferente de ordenar seis.

Quando estamos falando de tempo de execução queremos sempre que seja o máximo de tempo possível, porque assim temos a garantia de que ele não deixara passar nada. Assim temos alguns tipos de analise:

Pior caso: quando a ordem do array está invertida.

Assim o laço interno executará n-1vezes, aonde o “n” é o valor da variável no laço externo.

Melhor caso: outro algoritmo pior poderá trabalhar mais rápido para uma entrada especificada e assim esse tipo de análise não prove garantia a seu tempo de execução.

Então o tempo de execução de pior caso podemos dizer que depende da maquina em que a ordenação esta sendo executada.

4.3 Característica:

Numero de trocas e comparações menores entre a ordenação.

Esse método tem a fama por ser um método mais fácil de programar pelo fato de usar a interação de códigos da linguagem básica como o While, for, do while e entre outros tornando ainda mais fácil a interação do programador com a construção do método.

5. Quick sort

Quick sort é um algoritmo de comparação não estável, rápido e mais eficiente que seus concorrentes e conta com a estratégia do ditado popular “dividir para conquistar”.

Desenvolvido em 1960 pelo inglês Charles Antony Richard Hoare (Conhecido como Tony Hoare ou C.A.R. Hoare) quando fez uma visita na Universidade de Moscovo, ele criou o ‘Quick sort’ tentando traduzir um dicionário de inglês para russo, ao ordenar as palavras para reduzir o problema original em subproblemas para serem resolvidos de forma mais fácil e rápido. A versão final do algoritmo só saiu em 1962 após vários aperfeiçoamentos.

5.1 Como funciona

O seu funcionamento tem como função rearranjar os elementos de modo que os menores precedam os maiores, após isso ele divide em duas sublistas (pivô) e trata cada uma de uma vez até que os elementos fiquem ordenados (Otimização de casos específicos).

5.2 Passo a Passo:

1- Caso base ou não: ou seja, se é lista unitária ou vazia, nesse caso a entrada já é comum e ordenada.

2- Escolher qualquer elemento da lista pivô, que normalmente é sempre o primeiro elemento da lista de sequência a ser ordenada.

3- Reorganização da lista de forma que os elementos menores que o pivô fique de um lado e do outro lado os que são maiores. Esse processo é mais conhecido como particionamento.

4- Recursivamente ordene as sublistas dos elementos maiores e menores

5- Combinam-se as duas listas ordenadas e o pivô

A recursão são listas sempre ordenadas de tamanho zero ou um. Esse processo é finito, por causa de cada iteração aonde um elemento é colocado na última posição que é o elemento que não será manipulado na próxima iteração.

Pior caso de execução: quando os números de elementos a ser ordenados são muito grandes a execução desse método fica muito lenta mesmo levando em conta o tipo de maquina a ser executada.

6. BUBBLE SORT

Bubble sort, ou ordenação por flutuação (literalmente "por bolha"), é um algoritmo de ordenação dos mais simples. A ideia é percorrer o vector diversas vezes, a cada passagem fazendo flutuar para o topo o maior elemento da sequência. Essa movimentação lembra a forma como as bolhas em um tanque de água procuram seu próprio nível, e disso vem o nome do algoritmo.

> 'O problema consiste em (ordenação de array), troca de valores num vetor até que este se encontre ordenado. Os valores que serão mostrados neste trabalho, a ordenação se dará por ordem crescente' Segundo Furtado (2000).

Buble sort é a troca de valores entre posições consecutivas, fazendo com que os vetores mais altos (ou mais baixos) "borbulhem" para o final do vetor.

Bubble sort tem de pior caso e média complexidade tanto, onde n é o número de itens a serem ordenados. Existem muitos algoritmos de ordenação com substancialmente melhor a complexidade de pior caso, ou média. Mesmo outros algoritmos de ordenação, como a ordenação por inserção, tendem a ter melhor desempenho do que bubble sort. Portanto, a bolha de classificação não é um algoritmo de ordenação prático quando N é grande.

A única vantagem significativa que bubble sort tem sobre a maioria das outras implementações, mesmo quicksort, mas não a ordenação por inserção, é que a capacidade de detectar que a lista é ordenada de forma eficiente é embutida no algoritmo. Desempenho de bubble sort sobre uma lista já classificada (melhor caso). Por outro lado, a maioria dos outros algoritmos, mesmo aqueles com melhor média complexidade do caso, realizar todo o seu processo de classificação no conjunto e, portanto, são mais complexas. No entanto, não só tipo de inserção tem esse mecanismo também, mas também um melhor desempenho de uma lista que está substancialmente ordenada (tendo um pequeno número de inversões).

7. SELECTION SORT

Tipo de seleção é um algoritmo de ordenação, especificamente um enlace comparação espécie. Ele tem complexidade de tempo, tornando-o ineficiente em grandes listas, e, geralmente, um desempenho pior do que a semelhante tipo de inserção. Seleção espécie é conhecida por sua simplicidade, e tem vantagens de desempenho sobre algoritmos mais complicados em determinadas situações, particularmente onde a memória auxiliar é limitada.

O algoritmo divide a lista de entrada em duas partes: a sublista de itens já ordenados, que é construído em cima da esquerda para a direita na frente (à esquerda) da lista, e os sub-lista de itens restantes a serem classificados, que ocupam o resto da lista. Inicialmente, a sub-lista ordenada está vazio e sublist indiferenciados é a lista de entrada inteira. O algoritmo prossegue, encontrando o menor (ou maior, dependendo de ordem de classificação) elemento da sub-lista não ordenada, trocá-lo com o elemento indiferenciado mais à esquerda (colocando-o em ordem de classificação), e movendo-se os limites sublista um elemento para a direita. O algoritmo de seleção direta ou linear é simples e de fácil entendimento.

'Segundo Oliveira (1999b), esse método tem por finalidade de descobrir, a cada iteração, qual é o menor elemento do vetor (elemento este que passará a ocupar a sua posição definitiva no vetor ordenado)'. No inicio do procedimento, o primeiro elemento, o pivô, é considerado o menor. Em seguida, por comparações sucessivas entre o pivô e os demais elementos do vetor, buscando-se o menor entre todos. Se não existir, é porque o elemento encontrado troca de lugar com o pivô. Se não existir, é porque o primeiro elemento pivô, já é o menor. Em seguida, parte-se para o elemento seguinte e repete-se a operação, até a análise do penúltimo elemento do vetor.

8. CONSIDERAÇÕES FINAIS

O trabalho apresenta o comparativo de processamento em tempo de seis diferentes técnicas de ordenação, aplicados a três conjuntos diferentes de objetos, sendo eles ordenados em ordem crescente, decrescente e embaralhados.

A conclusão que o trabalho nos mostrou indica que não podemos eleger um método universal de ordenação. Para cada tipo de situação, deve-se aplicar um diferente método, a fim de conseguir uma ordenação num menor tempo possível.

8.1 Conjunto de objetos embaralhados

8.1.1 Tempo de processamento em micros segundos

Qtd. Imagens	10	100	2000	4000	6000	8000	10000
Bubble Sort Tipo: Por Troca	24 µs	1.020 µs	30.239 µs	86.023 µs	172.032 µs	395.098 µs	639.370 µs
Selection Sort Tipo: Por Seleção	27 µs	1.102 µs	26.652 µs	85.123 µs	175.769 µs	419.151 µs	500.843 µs
Insertion Sort Tipo: Por Inserção	26 µs	511 µs	15.234 µs	31.256 µs	51.063 µs	98.174 µs	154.502 µs
Merge Sort Tipo: Por Partição	37 µs	183 µs	6.589 µs	11.652 µs	2.654 µs	2.475 µs	3.286 µs
Quick Sort Tipo: Por Partição	29 µs	136 µs	4.014 µs	12.172 µs	8.047 µs	1.606 µs	2.206 µs
Radix Sort Tipo: Por Distribuicao	58 µs	210 µs	4.551 µs	10.922 µs	14.208 µs	6.211 µs	4.522 µs

8.1.2 Eficiência dos métodos em tempo de processamento

10 imagens	100 imagens	2000 imagens	4000 imagens	6000 imagens	8000 imagens	10000 imagens
Bubble Sort ↑ 8%	Quick Sort ↑ 35%	Quick Sort ↑ 13%	Radix Sort ↑ 7%	Merge Sort ↑ 203%	Quick Sort ↑ 54%	Quick Sort ↑ 49%
Insertion Sort ↑ 4%	Merge Sort ↑ 15%	Radix Sort ↑ 45%	Merge Sort ↑ 4%	Quick Sort ↑ 77%	Merge Sort ↑ 151%	Merge Sort ↑ 38%
Selection Sort ↑ 7%	Radix Sort ↑ 143%	Merge Sort ↑ 131%	Quick Sort ↑ 157%	Radix Sort ↑ 259%	Radix Sort ↑ 1.481%	Radix Sort ↑ 3.317%
Quick Sort ↑ 28%	Insertion Sort ↑ 100%	Insertion Sort ↑ 75%	Insertion Sort ↑ 172%	Insertion Sort ↑ 237%	Insertion Sort ↑ 302%	Insertion Sort ↑ 224%
Merge Sort ↑ 57%	Bubble Sort ↑ 8%	Selection Sort ↑ 13%	Selection Sort ↑ 1%	Bubble Sort ↑ 2%	Bubble Sort ↑ 6%	Selection Sort ↑ 28%
Radix Sort	Selection Sort	Bubble Sort	Bubble Sort	Selection Sort	Selection Sort	Bubble Sort

8.1.3 Conclusão:

Para o conjunto de imagens embaralhadas, os métodos de implementação simples que utilizam a iteração como o Bubble Sort, é mais eficiente, porém, a partir de 100 imagens, os métodos mais complexos como Radix Sort e Quick Sort, apresentam eficiência de até 80% maior comparado aos métodos mais simples.

8.2 Conjuntos de imagens em ordem decrescente

8.2.1 Tempo de processamento em micros segundos

Qtd. Imagens	10	100	2000	4000	6000	8000	10000
Bubble Sort Tipo: Por Troca	5 µs	30 µs	12.051 µs	52.211 µs	103.013 µs	184.827 µs	297.352 µs
Selection Sort Tipo: Por Seleção	2 µs	26 µs	10.966 µs	43.997 µs	98.944 µs	179.976 µs	292.437 µs
Insertion Sort Tipo: Por Inserção	4 µs	28 µs	11.108 µs	44.309 µs	98.876 µs	179.533 µs	288.483 µs
Merge Sort Tipo: Por Partição	10 µs	17 µs	352 µs	723 µs	920 µs	1.290 µs	1.632 µs
Quick Sort Tipo: Por Partição	5 µs	8 µs	168 µs	361 µs	569 µs	603 µs	815 µs
Radix Sort Tipo: Por Distribuicao	88 µs	35 µs	455 µs	855 µs	1.090 µs	1.484 µs	4.781 µs

8.2.2 Eficiência dos métodos em tempo de processamento

10 imagens	100 imagens	2000 imagens	4000 imagens	6000 imagens	8000 imagens	10000 imagens
Selection Sort ↑ 100%	Quick Sort ↑ 113%	Quick Sort ↑ 110%	Quick Sort ↑ 100%	Quick Sort ↑ 62%	Quick Sort ↑ 114%	Quick Sort ↑ 100%
Insertion Sort ↑ 25%	Merge Sort ↑ 53%	Merge Sort ↑ 29%	Merge Sort ↑ 18%	Merge Sort ↑ 18%	Merge Sort ↑ 15%	Merge Sort ↑ 193%
Quick Sort	Selection Sort ↑ 8%	Radix Sort ↑ 2.310%	Radix Sort ↑ 5.046%	Radix Sort ↑ 8.971%	Radix Sort ↑ 11.998%	Radix Sort ↑ 5.934%
Bubble Sort ↑ 100%	Insertion Sort ↑ 7%	Selection Sort ↑ 1%	Selection Sort ↑ 1%	Insertion Sort ↑ 0%	Insertion Sort ↑ 0%	Insertion Sort ↑ 1%
Merge Sort ↑ 780%	Bubble Sort ↑ 17%	Insertion Sort ↑ 8%	Insertion Sort ↑ 18%	Selection Sort ↑ 4%	Selection Sort ↑ 3%	Selection Sort ↑ 2%
Radix Sort	Radix Sort	Bubble Sort	Bubble Sort	Bubble Sort	Bubble Sort	Bubble Sort

8.2.3 Conclusão:

Considerado neste trabalho o pior caso para ordenação, pelo fato dos objetos estarem completamente em ordem reversa, os métodos de implementação Selection Sort é 100% mais eficiente que o Insertion Sort em quantidades baixa de imagens, porém, a partir de 100 imagens, o Quick Sort tem eficiência de até 113% acima do segundo colocado, o também eficiente Merge Sort.

8.3 Conjuntos de imagens em ordem crescente

8.3.1 Tempo de processamento em micros segundos

Qtd. Imagens	10	100	2000	4000	6000	8000	10000
Bubble Sort Tipo: Por Troca	8 µs	5 µs	93 µs	208 µs	304 µs	329 µs	288 µs
Selection Sort Tipo: Por Seleção	3 µs	16 µs	8.970 µs	31.582 µs	69.701 µs	126.626 µs	203.290 µs
Insertion Sort Tipo: Por Inserção	3 µs	7 µs	131 µs	322 µs	365 µs	471 µs	814 µs
Merge Sort Tipo: Por Partição	12 µs	18 µs	364 µs	1.065 µs	1.062 µs	1.018 µs	1.518 µs
Quick Sort Tipo: Por Partição	5 µs	8 µs	147 µs	346 µs	568 µs	618 µs	738 µs
Radix Sort Tipo: Por Distribuição	6 µs	52 µs	474 µs	714 µs	964 µs	1.302 µs	2.088 µs

8.3.2 Eficiência dos métodos em tempo de processamento

10 imagens	100 imagens	2000 imagens	4000 imagens	6000 imagens	8000 imagens	10000 imagens
Selection Sort	Bubble Sort ↑ 40%	Bubble Sort ↑ 41%	Bubble Sort ↑ 55%	Bubble Sort ↑ 20%	Bubble Sort ↑ 43%	Bubble Sort ↑ 156%
Insertion Sort ↑ 67%	Insertion Sort ↑ 14%	Insertion Sort ↑ 12%	Insertion Sort ↑ 7%	Insertion Sort ↑ 56%	Insertion Sort ↑ 31%	Quick Sort ↑ 10%
Quick Sort ↑ 20%	Quick Sort ↑ 100%	Quick Sort ↑ 148%	Quick Sort ↑ 106%	Quick Sort ↑ 70%	Quick Sort ↑ 65%	Insertion Sort ↑ 86%
Radix Sort ↑ 33%	Selection Sort ↑ 13%	Merge Sort ↑ 30%	Radix Sort ↑ 49%	Radix Sort ↑ 10%	Merge Sort ↑ 28%	Merge Sort ↑ 38%
Bubble Sort ↑ 50%	Merge Sort ↑ 189%	Radix Sort ↑ 1.792%	Merge Sort ↑ 2.865%	Merge Sort ↑ 6.463%	Radix Sort ↑ 9.625%	Radix Sort ↑ 9.636%
Merge Sort	Radix Sort	Selection Sort	Selection Sort	Selection Sort	Selection Sort	Selection Sort

8.3.3 Conclusão:

Considerado neste trabalho o melhor caso, pelo fato dos objetos estarem totalmente ordenados, os métodos mais simples são os que têm melhor eficiência. Independente da quantidade de objetos a serem verificados, a técnica mais simples de todas, no caso o Bubble Sort, é o mais eficiente, mantendo eficiência de até 156% acima do Quick Sort, onde este se utiliza de recursão ao invés de simples iteração como os métodos mais simples.

9. ESTRUTURA DO PROGRAMA

CÓDIGO FONTE

Pacote br.marciofcruz.apsmetodos.ordenacao.imagem

CLASSE IMAGEMSATELITE

```
package br.marciofcruz.apsmetodosordenacao.imagem;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Arrays;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

import br.marciofcruz.apsmetodosordenacao.tipos.TipoCriterioOrdem;

/**
 * Esta classe representa, para cada quantidade de imagens, um array ordenado e
 * outro aleatório
 *
 * @author B22816-4 Marcio Fernandes Cruz
 * @see java.lang.Object
 * @version 1.00
 * @since 1.0
 */

public class ArrayImagens {

	private int qtdeImagens;
	private ImagemSatelite[] crescente;
	private ImagemSatelite[] embaralhado;
	private ImagemSatelite[] decrescente;

	public ArrayImagens(int qtdeImagens, String path)
				throws FileNotFoundException {
		this.qtdeImagens = qtdeImagens;

		ImagemSatelite[] objetos = CriarArrayImagens(path, "JPG", qtdeImagens);

		this.crescente = objetos.clone();
		this.embaralhado = crescenteToEmbaralhado(objetos.clone());
		this.decrescente = crescenteToReverso(objetos.clone());
	}
```

```
public ImagemSatelite[] getImagensOrdem(TipoCriterioOrdem tipoCriterioOrdem) {
		switch (tipoCriterioOrdem) {
		case crescente:
			return crescente;
		case decrescente:
			return decrescente;
		case embaralhado:
			return embaralhado;
		default:
			return embaralhado;
		}
}

public int qtdeImagens() {
		return qtdeImagens;
}

private ImagemSatelite[] crescenteToEmbaralhado(ImagemSatelite[] origem) {
		ImagemSatelite[] retorno = new ImagemSatelite[origem.length];

		List<ImagemSatelite> lista = Arrays.asList(origem);

		Collections.shuffle(lista);

		int cont = 0;

		Iterator<ImagemSatelite> iterator = lista.iterator();
		while (iterator.hasNext()) {
			retorno[cont] = iterator.next();
			cont++;
		}

		return retorno;
}

/**
 * Retorna o array em ordem reversa
 */
private ImagemSatelite[] crescenteToReverso(ImagemSatelite[] origem) {
		ImagemSatelite[] retorno = new ImagemSatelite[origem.length];

		List<ImagemSatelite> lista = Arrays.asList(origem);

		Collections.reverse(lista);

		int cont = 0;

		Iterator<ImagemSatelite> iterator = lista.iterator();
		while (iterator.hasNext()) {
			retorno[cont] = iterator.next();
```

```
                cont++;
            }

            return retorno;
        }

        /**
         * Busca as imagens de array em determinado diretório e cria um array de retorno
         */

        private ImagemSatelite[] CriarArrayImagens(String pastaImagens, String extensaoImagem,
                int numeroImagens) throws FileNotFoundException {

            int quantidadeImagensCapturadas = 0;

            File diretorioImagens = new File(pastaImagens);

            String[] listaArquivos = diretorioImagens.list();

            String extensao = "." + extensaoImagem.toUpperCase();

            ImagemSatelite[] imagensDeSatelite = new ImagemSatelite[numeroImagens];

            for (int i = 1; (i < listaArquivos.length) && (i <= numeroImagens); i++) {

                if (listaArquivos[i].toString().toUpperCase().endsWith(extensao)) {

                    imagensDeSatelite[quantidadeImagensCapturadas] = new
ImagemSatelite(pastaImagens+"\\"+listaArquivos[i].toString());

                    quantidadeImagensCapturadas++;
                }
            }
            return imagensDeSatelite;
        }

}
```

CLASSE ARRAYIMAGENS

```
package br.marciofcruz.apsmetodosordenacao.imagem;

import java.io.File;
import java.io.FileNotFoundException;
import java.util.Arrays;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

import br.marciofcruz.apsmetodosordenacao.tipos.TipoCriterioOrdem;

/**
 * Esta classe representa, para cada quantidade de imagens, um array ordenado e
```

```
 * outro aleatório
 *
 * @author B22816-4 Marcio Fernandes Cruz
 * @see java.lang.Object
 * @version 1.00
 * @since 1.0
 */

public class ArrayImagens {

	private int qtdeImagens;
	private ImagemSatelite[] crescente;
	private ImagemSatelite[] embaralhado;
	private ImagemSatelite[] decrescente;

	public ArrayImagens(int qtdeImagens, String path)
			throws FileNotFoundException {
		this.qtdeImagens = qtdeImagens;

		ImagemSatelite[] objetos = CriarArrayImagens(path, "JPG", qtdeImagens);

		this.crescente = objetos.clone();
		this.embaralhado = crescenteToEmbaralhado(objetos.clone());
		this.decrescente = crescenteToReverso(objetos.clone());
	}

	public ImagemSatelite[] getImagensOrdem(TipoCriterioOrdem tipoCriterioOrdem) {
		switch (tipoCriterioOrdem) {
		case crescente:
			return crescente;
		case decrescente:
			return decrescente;
		case embaralhado:
			return embaralhado;
		default:
			return embaralhado;
		}
	}

	public int qtdeImagens() {
		return qtdeImagens;
	}

	private ImagemSatelite[] crescenteToEmbaralhado(ImagemSatelite[] origem) {
		ImagemSatelite[] retorno = new ImagemSatelite[origem.length];

		List<ImagemSatelite> lista = Arrays.asList(origem);

		Collections.shuffle(lista);

		int cont = 0;

		Iterator<ImagemSatelite> iterator = lista.iterator();
		while (iterator.hasNext()) {
			retorno[cont] = iterator.next();
			cont++;
		}

		return retorno;
	}

	/**
	 * Retorna o array em ordem reversa
	 */
	private ImagemSatelite[] crescenteToReverso(ImagemSatelite[] origem) {
		ImagemSatelite[] retorno = new ImagemSatelite[origem.length];
```

```
                List<ImagemSatelite> lista = Arrays.asList(origem);

                Collections.reverse(lista);

                int cont = 0;

                Iterator<ImagemSatelite> iterator = lista.iterator();
                while (iterator.hasNext()) {
                        retorno[cont] = iterator.next();
                        cont++;
                }

                return retorno;
        }

        /**
         * Busca as imagens de array em determinado diretório e cria um array de retorno
         */

        private ImagemSatelite[] CriarArrayImagens(String pastaImagens, String extensaoImagem,
                        int numeroImagens) throws FileNotFoundException {

                int quantidadeImagensCapturadas = 0;

                File diretorioImagens = new File(pastaImagens);

                String[] listaArquivos = diretorioImagens.list();

                String extensao = "." + extensaoImagem.toUpperCase();

                ImagemSatelite[] imagensDeSatelite = new ImagemSatelite[numeroImagens];

                for (int i = 1; (i < listaArquivos.length) && (i <= numeroImagens); i++) {

                        if (listaArquivos[i].toString().toUpperCase().endsWith(extensao)) {

                                imagensDeSatelite[quantidadeImagensCapturadas] = new
ImagemSatelite(pastaImagens+"\\"+listaArquivos[i].toString());

                                quantidadeImagensCapturadas++;
                        }
                }
                return imagensDeSatelite;
        }

}
```

Pacote br.marciofcruz.apsmetodos.ordenacao.tipos

CLASSE CLASSIFICACAOMETODO

```
package br.marciofcruz.apsmetodosordenacao.tipos;

import java.security.InvalidAlgorithmParameterException;

public enum ClassificacaoMetodo {
        porTroca, porSelecao, porInsercao, porParticao, porPosicao, porDistribuicao, nenhum;
```

```
	static public String getNome(ClassificacaoMetodo classificacaoMetodo) throws InvalidAlgorithmParameterException {
		switch (classificacaoMetodo) {
		case porInsercao:
			return "Por Inserção";
		case porParticao:
			return "Por Partição";
		case porPosicao:
			return "por Posição";
		case porSelecao:
			return "Por Seleção";
		case porTroca:
			return "Por Troca";
		case porDistribuicao:
			return "Por Distribuicao";
		default:
			throw new InvalidAlgorithmParameterException("Parâmetro Inválido");
		}
	}

}
```

CLASSE TIPOCRITERIOORDEM

```
package br.marciofcruz.apsmetodosordenacao.tipos;

import java.security.InvalidAlgorithmParameterException;

public enum TipoCriterioOrdem {
	embaralhado, decrescente, crescente;

	static public String getNome(TipoCriterioOrdem tipoCriterioOrdem) throws InvalidAlgorithmParameterException {
		switch (tipoCriterioOrdem) {
		case crescente:
			return "Imagens em Ordem Crescente";
		case decrescente:
			return "Imagens em Ordem Decrescente";
		case embaralhado:
			return "Imagens Embaralhadas";
		default:
			throw new InvalidAlgorithmParameterException("Tipo de ordenação não definido");

		}
	}

}
```

CLASSE TIPOEXIBICOAGRAFICO

```
package br.marciofcruz.apsmetodosordenacao.tipos;
```

```
import java.awt.Color;
import java.security.InvalidAlgorithmParameterException;

public enum TipoExibicaoGrafico {
	tempoProcessamento;

	public static String getDescricao(TipoExibicaoGrafico tipo)
			throws InvalidAlgorithmParameterException {
		switch (tipo) {
		case tempoProcessamento:
			return "Tempo de Processamento";
		default:
			throw new InvalidAlgorithmParameterException("Argumento inválido");

		}
	}

	public static Color getCor(TipoExibicaoGrafico tipo)
			throws InvalidAlgorithmParameterException {
		switch (tipo) {
		case tempoProcessamento:
			return Color.blue;
		default:
			throw new InvalidAlgorithmParameterException("Argumento inválido");

		}

	}
}
```

CLASSE TIPOMETODOORDENACAO

```
package br.marciofcruz.apsmetodosordenacao.tipos;

import java.security.InvalidAlgorithmParameterException;

import br.marciofcruz.apsmetodosordenacao.ordenacao.*;

public enum TipoMetodoOrdenacao {
	bubbleSort, selectionSort, /*heapSort,*/ insertionSort, mergeSort, quickSort, radixSort;

	static public String getNome(TipoMetodoOrdenacao tipoMetodoOrdenacao) {
		switch (tipoMetodoOrdenacao) {
		case bubbleSort:
			return "Bubble Sort";
		case selectionSort:
			return "Selection Sort";
		/*case heapSort:
			return "Heap Sort"; */
		case insertionSort:
			return "Insertion Sort";
		case mergeSort:
			return "Merge Sort";
		case quickSort:
			return "Quick Sort";
```

```
		case radixSort:
			return "Radix Sort";
		default:
			return null;
		}
	}

	static public Ordenacao getInstanciaDe(
			TipoMetodoOrdenacao tipoMetodoOrdenacao) {
		switch (tipoMetodoOrdenacao) {
		case bubbleSort:
			return new BubbleSort();
		case selectionSort:
			return new SelectionSort();
		/*case heapSort:
			return new HeapSort(); */
		case insertionSort:
			return new InsertionSort();
		case mergeSort:
			return new MergeSort();
		case quickSort:
			return new QuickSort();
		case radixSort:
			return new RadixSort();
		default:
			return null;
		}
	}

	static public TipoMetodoOrdenacao getTipoPorNome(String nomeClasse)
			throws InvalidAlgorithmParameterException {

		switch (nomeClasse) {
		case "marciofcruz.apsmetodosordenacao.ordenacao.BubbleSort":
			return TipoMetodoOrdenacao.bubbleSort;
		case "marciofcruz.apsmetodosordenacao.ordenacao.SelectionSort":
			return TipoMetodoOrdenacao.selectionSort;
		/*case "marciofcruz.apsmetodosordenacao.ordenacao.HeapSort":
			return TipoMetodoOrdenacao.heapSort; */
		case "marciofcruz.apsmetodosordenacao.ordenacao.InsertionSort":
			return TipoMetodoOrdenacao.insertionSort;
		case "marciofcruz.apsmetodosordenacao.ordenacao.MergeSort":
			return TipoMetodoOrdenacao.mergeSort;
		case "marciofcruz.apsmetodosordenacao.ordenacao.QuickSort":
			return TipoMetodoOrdenacao.quickSort;
		case "marciofcruz.apsmetodosordenacao.ordenacao.RadixSort":
			return TipoMetodoOrdenacao.radixSort;
		default:
			throw new InvalidAlgorithmParameterException(
					"Nâo é possível traduzir o tipo de método de ordenação da classe: "
							+ nomeClasse);
```

```
                }

        }

        static public ClassificacaoMetodo getTipoClassificacaoMetodo(
                        TipoMetodoOrdenacao tipoMetodoOrdenacao)
                        throws InvalidAlgorithmParameterException {
                switch (tipoMetodoOrdenacao) {
                case bubbleSort:
                        return ClassificacaoMetodo.porTroca;
                case insertionSort:
                        return ClassificacaoMetodo.porInsercao;
                case selectionSort:
                        return ClassificacaoMetodo.porSelecao;
                /*case heapSort:
                        return ClassificacaoMetodo.porSelecao; */
                case mergeSort:
                        return ClassificacaoMetodo.porParticao;
                case quickSort:
                        return ClassificacaoMetodo.porParticao;
                case radixSort:
                        return ClassificacaoMetodo.porDistribuicao;

                default:
                        throw new InvalidAlgorithmParameterException("Parâmetro inválido");
                }
        }

}
```

Pacote br.marciofcruz.apsmetodosordenacao.eficiencia

CLASSE METODO TEMPOPROCESSAMENTO

```
package br.marciofcruz.apsmetodosordenacao.eficiencia;

import br.marciofcruz.apsmetodosordenacao.tipos.TipoMetodoOrdenacao;

public class MetodoTempoProcessamento implements
                Comparable<MetodoTempoProcessamento> {

        private TipoMetodoOrdenacao tipoMetodoOrdenacao;
        private long tempoProcessamentoemNanoSegundos = 0;
        private double eficienciaEmPorcentagem;

        public MetodoTempoProcessamento(TipoMetodoOrdenacao tipoMetodoOrdenacao,
                        long tempoProcessamentoemNanoSegundos) {
```

```
                this.tipoMetodoOrdenacao = tipoMetodoOrdenacao;
                this.tempoProcessamentoemNanoSegundos = tempoProcessamentoemNanoSegundos;

        }

        public TipoMetodoOrdenacao getTipoMetodoOrdenacao() {
                return tipoMetodoOrdenacao;
        }

        public int getTempoProcessamentoemMicroSegundos() {
                return (int) tempoProcessamentoemNanoSegundos / 1000;
        }

        public long getTempoProcessamentoemNanoSegundos() {
                return tempoProcessamentoemNanoSegundos;
        }

        @Override
        public int hashCode() {
                return (int) (tempoProcessamentoemNanoSegundos / 1000);
        }

        @Override
        public int compareTo(MetodoTempoProcessamento o) {
                if (this.getTempoProcessamentoemNanoSegundos() < o
                                .getTempoProcessamentoemNanoSegundos()) {
                        return -1;
                } else if (this.getTempoProcessamentoemNanoSegundos() < o
                                .getTempoProcessamentoemNanoSegundos()) {
                        return 1;

                } else {
                        return 0;
                }

        }

        /**
         * @return the eficienciaEmPorcentagem
         */
        public double getEficienciaEmPorcentagem() {
                return eficienciaEmPorcentagem;
        }

        /**
         * @param eficienciaEmPorcentagem the eficienciaEmPorcentagem to set
         */
        public void setEficienciaEmPorcentagem(double eficienciaEmPorcentagem) {
                this.eficienciaEmPorcentagem = eficienciaEmPorcentagem;
        }
}
```

CLASSE EFICIENCIA

```
package br.marciofcruz.apsmetodosordenacao.eficiencia;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;
import java.util.List;

import br.marciofcruz.apsmetodosordenacao.tipos.TipoMetodoOrdenacao;

public class Eficiencia {

	private int quantidadeImagens;
	private List<MetodoTempoProcessamento> listaMetodoTempoProcessamento = new ArrayList<>();

	public Eficiencia(int quantidadeImagens) {
		this.quantidadeImagens = quantidadeImagens;
	}

	public int getQuantidadeImagens() {
		return quantidadeImagens;
	}

	public List<MetodoTempoProcessamento> getListaMetodoTempoProcessamento() {
		return listaMetodoTempoProcessamento;
	}

	public void adicionar(TipoMetodoOrdenacao tipoMetodoOrdenacao,
			long tempoProcessamentoemNanoSegundos) {

		listaMetodoTempoProcessamento.add(new MetodoTempoProcessamento(
				tipoMetodoOrdenacao, tempoProcessamentoemNanoSegundos));

		setDadosEficiencia();
	}

	/*
	 * Este método ordena o array e verifica a eficiência com o registro anterior
	 */
	private void setDadosEficiencia() {
		Collections.sort(listaMetodoTempoProcessamento);

		MetodoTempoProcessamento anterior = null;
		Iterator<MetodoTempoProcessamento> iterator = listaMetodoTempoProcessamento.iterator();

		while(iterator.hasNext()) {
			MetodoTempoProcessamento atual = iterator.next();
```

```
                        if (anterior != null) {
                                double porcentagem = 0;

                                if (atual.getTempoProcessamentoemNanoSegundos() != 0) {
                                        double tempoAnterior =
(double)anterior.getTempoProcessamentoemMicroSegundos();
                                        double tempoAtual =
(double)atual.getTempoProcessamentoemMicroSegundos();

                                        porcentagem = ((tempoAtual/tempoAnterior)-1)*100;

                                        anterior.setEficienciaEmPorcentagem(porcentagem);
                                }
                        }

                        anterior = atual;
                }
        }

}
```

Classe ListaEficiencia

```
package br.marciofcruz.apsmetodosordenacao.eficiencia;

import java.security.InvalidAlgorithmParameterException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

public class ListaEficiencia {

        private List<Eficiencia> listaEficiencia = new ArrayList<>();

        public List<Eficiencia> getListaEficiencia() {
                return listaEficiencia;
        }

        public void adicionar(Eficiencia eficiencia) {
                listaEficiencia.add(eficiencia);
        }

        public Eficiencia getEficienciaPorQuantidadeImagens(int quantidadeImagens) throws
InvalidAlgorithmParameterException {
                Iterator<Eficiencia> iterator = listaEficiencia.iterator();

                while(iterator.hasNext()) {
                        Eficiencia retorno = iterator.next();

                        if (retorno.getQuantidadeImagens() == quantidadeImagens) {
                                return retorno;
```

```
                        }
                }

                throw new InvalidAlgorithmParameterException("Argumento inválido");
        }

}
```

Pacote br.marciofcruz.apsmetodosordenacao.apresentacao

CLASSE RESULTADO

```
package br.marciofcruz.apsmetodosordenacao.apresentacao;

import br.marciofcruz.apsmetodosordenacao.tipos.TipoMetodoOrdenacao;

/**
 * Esta classe representa os números alcançados através do processamento de determinado método em
 * função de determinada quantidade de imagens
 * @author B22816-4 Marcio Fernandes Cruz
 * @see java.lang.Object
 * @version 1.00
 * @since 1.0
 */

public class Resultado {

        private TipoMetodoOrdenacao tipoMetodoOrdenacao;
        private int quantidadeImagens;
        private long tempoemNs;


        public TipoMetodoOrdenacao getTipoMetodoOrdenacao() {
                return tipoMetodoOrdenacao;
        }

        @Override
        public String toString() {
                return "Resultado [nomeMetodo=" + TipoMetodoOrdenacao.getNome(tipoMetodoOrdenacao)+ ",
quantidadeImagens="
                                + quantidadeImagens +", tempoemNs=" + tempoemNs + "]";
        }

        public int getQuantidadeImagens() {
                return quantidadeImagens;
        }
```

```
	public Resultado(TipoMetodoOrdenacao tipoMetodoOrdenacao, int quantidadeImagens) {
		this.tipoMetodoOrdenacao = tipoMetodoOrdenacao;
		this.quantidadeImagens = quantidadeImagens;

		tempoemNs = 0;
	}

	public long getTempoemNanoSegundo() {
		return tempoemNs;
	}

	public long getTempoEmMicroSegundo() {
		return tempoemNs/1000;
	}


	public void setTempoemNs(long tempoemNs) {
		this.tempoemNs = tempoemNs;
	}
}
```

CLASSE LISTARESULTADO

```
package br.marciofcruz.apsmetodosordenacao.apresentacao;

import java.security.InvalidAlgorithmParameterException;
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;

import br.marciofcruz.apsmetodosordenacao.tipos.TipoMetodoOrdenacao;

public class ListaResultado {

	private List<Resultado> resultados = new ArrayList<>();

	public List<Resultado> getResultados() {
		return resultados;
	}

	public void adicionar(final Resultado resultado) {
		resultados.add(resultado);
	}

	public Resultado getRetornoPorQuantidadeImagens(TipoMetodoOrdenacao tipoMetodoOrdenacao, int
quantidadeImagens) throws InvalidAlgorithmParameterException {
		Iterator<Resultado> iterator = resultados.iterator();

		while(iterator.hasNext()) {
```

```
                        Resultado retorno = iterator.next();

                        if (retorno.getQuantidadeImagens() == quantidadeImagens &&
                                retorno.getTipoMetodoOrdenacao() == tipoMetodoOrdenacao) {
                                return retorno;
                        }
                }

                throw new InvalidAlgorithmParameterException("Argumento inválido");
        }
}
```

CLASSE GRAFICO

```
package br.marciofcruz.apsmetodosordenacao.apresentacao;

import java.io.FileOutputStream;
import java.io.IOException;
import java.security.InvalidAlgorithmParameterException;
import java.util.Iterator;


import org.jfree.chart.ChartFactory;
import org.jfree.chart.ChartUtilities;
import org.jfree.chart.JFreeChart;
import org.jfree.chart.plot.PlotOrientation;
import org.jfree.chart.title.TextTitle;
import org.jfree.data.category.DefaultCategoryDataset;

import br.marciofcruz.apsmetodosordenacao.tipos.TipoCriterioOrdem;
import br.marciofcruz.apsmetodosordenacao.tipos.TipoExibicaoGrafico;
import br.marciofcruz.apsmetodosordenacao.tipos.TipoMetodoOrdenacao;

/**
 * Esta classe gera um arquivo PNG com gráficos, afim de ser inserido na
 * apresentação HTML
 *
 * @author B22816-4 Marcio Fernandes Cruz
 * @see java.lang.Object
 * @version 1.00
 * @since 1.0
 */

public class Grafico {

        private ListaResultado listaResultado;
        private TipoExibicaoGrafico tipoExibicaoGrafico;
        private TipoCriterioOrdem tipoCriterioOrdem;
        private String nomeEixoY;
```

```
private String getTitulo() throws InvalidAlgorithmParameterException {
		StringBuilder nome = new StringBuilder();

		nome.append(TipoExibicaoGrafico.getDescricao(tipoExibicaoGrafico));
		nome.append(" em ");
		nome.append(TipoCriterioOrdem.getNome(tipoCriterioOrdem));

		return nome.toString();
}

public String getNomeArquivo() {
		StringBuilder nome = new StringBuilder();

		switch (tipoExibicaoGrafico) {
		case tempoProcessamento:
			nome.append("tempoprocessamento");
			break;
		default:
			nome.append("tempoprocessamento");
			break;
		}

		switch (tipoCriterioOrdem) {
		case crescente:
			nome.append("crescente");
			break;
		case decrescente:
			nome.append("decrescente");
			break;
		case embaralhado:
			nome.append("embaralhado");
			break;
		default:
			nome.append("embaralhado");
			break;

		}

		nome.append(".png");

		return nome.toString();
}

private void gerarGrafico() throws InvalidAlgorithmParameterException,
			IOException {
		DefaultCategoryDataset ds = new DefaultCategoryDataset();

		Iterator<Resultado> iterator = listaResultado.getResultados()
					.iterator();
```

```
		while (iterator.hasNext()) {
			Resultado resultado = iterator.next();
			double dado = -1;

			switch (tipoExibicaoGrafico) {
			case tempoProcessamento:
				dado = Math.round(resultado.getTempoEmMicroSegundo());
				break;
			default:
				throw new InvalidAlgorithmParameterException(
						"Parâmetro não permitido");
			}

			ds.addValue(dado, TipoMetodoOrdenacao.getNome(resultado
					.getTipoMetodoOrdenacao()), Integer.toString(resultado
					.getQuantidadeImagens()));

		}

		JFreeChart grafico = ChartFactory.createLineChart(getTitulo(),
				"Imagens", nomeEixoY, ds, PlotOrientation.VERTICAL, true, true,
				false);

		grafico.setBorderVisible(true);
		grafico.fireChartChanged();

		TextTitle titulo = grafico.getTitle();
		titulo.setExpandToFitSpace(true);
		titulo.setPaint(TipoExibicaoGrafico.getCor(tipoExibicaoGrafico));

		FileOutputStream arquivo = new FileOutputStream(getNomeArquivo());
		ChartUtilities.writeChartAsPNG(arquivo, grafico, 900, 400);

	}

	public Grafico(ListaResultado listaResultado,
			TipoExibicaoGrafico tipoExibicaoGrafico,
			TipoCriterioOrdem tipoCriterioOrdem)
			throws InvalidAlgorithmParameterException, IOException {

		this.tipoCriterioOrdem = tipoCriterioOrdem;
		this.listaResultado = listaResultado;
		this.tipoExibicaoGrafico = tipoExibicaoGrafico;

		switch (tipoExibicaoGrafico) {
		case tempoProcessamento:
			nomeEixoY = "Tempo em µs (microsegundos)";
			break;
		default:
			throw new InvalidAlgorithmParameterException(
					"Parâmetro não permitido");
```

```
                }

                gerarGrafico();
        }

}
```

CLASSE AUXILIARHTML

```
package br.marciofcruz.apsmetodosordenacao.apresentacao;

import java.security.InvalidAlgorithmParameterException;
import java.text.DecimalFormat;
import java.text.NumberFormat;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Iterator;
import java.util.List;
import java.util.Properties;

import br.marciofcruz.apsmetodosordenacao.eficiencia.Eficiencia;
import br.marciofcruz.apsmetodosordenacao.eficiencia.ListaEficiencia;
import br.marciofcruz.apsmetodosordenacao.eficiencia.MetodoTempoProcessamento;
import br.marciofcruz.apsmetodosordenacao.tipos.ClassificacaoMetodo;
import br.marciofcruz.apsmetodosordenacao.tipos.TipoCriterioOrdem;
import br.marciofcruz.apsmetodosordenacao.tipos.TipoMetodoOrdenacao;

/**
 * Esta classe tem métodos auxiliares para a criação de um arquivo HTML de
 * apresentação dos resultados
 *
 * @author B22816-4 Marcio Fernandes Cruz
 * @see java.lang.Object
 * @version 1.00
 * @since 1.0
 */

public class AuxiliarHTML {

        private Date dataProcessamento;
        private StringBuilder texto;
        private int[] faixaDeQuantidades;

        public AuxiliarHTML(StringBuilder texto, int[] faixaDeQuantidades,
                        Date dataProcessamento) {
                this.texto = texto;
                this.faixaDeQuantidades = faixaDeQuantidades;
                this.dataProcessamento = dataProcessamento;
        }
```

```
public StringBuilder getTexto() {
	return texto;
}

public void adicionar(StringBuilder dado) {
	texto.append(dado.toString());
}

public void adicionar(String dado) {
	texto.append(dado);
}

public void inserirImagem(List<Grafico> graficos) {

	StringBuilder auxiliar = new StringBuilder();

	Iterator<Grafico> iterator = graficos.iterator();

	auxiliar.append("<p>");

	while (iterator.hasNext()) {
		auxiliar.append("<center><img src=\"");
		auxiliar.append(iterator.next().getNomeArquivo());
		auxiliar.append("\" alt=\"Resultados\"></center>\n");
	}

	auxiliar.append("</p>\n");

	adicionar(auxiliar);
}

public void imprimirTabelaComparativoMetodos(String tituloRelatorio,
		TipoCriterioOrdem tipoCriterioOrdem, List<Resultado> lista)
		throws InvalidAlgorithmParameterException {

	boolean impressaoImpar = true;

	StringBuilder auxiliar = new StringBuilder();

	auxiliar.append("<p>");
	auxiliar.append("<br><hr><center><font size=\"5\" color=#000099><strong>");
	auxiliar.append(tituloRelatorio);
	auxiliar.append("</strong></font></center>");

	auxiliar.append("<div class=\"datagrid\"><table>\n");

	auxiliar.append("<thead>\n");
	auxiliar.append("<th>");
	auxiliar.append("<strong>Qtd. Imagens</strong>");
	auxiliar.append("</th>\n");
```

```
		for (int i = 0; i < faixaDeQuantidades.length; i++) {
			auxiliar.append("<th>");
			auxiliar.append(faixaDeQuantidades[i]);
			auxiliar.append("</th>\n");
		}

		auxiliar.append("</tr></thead>\n");
		auxiliar.append("<tbody>\n");

		TipoMetodoOrdenacao metodoAnterior = null;

		Iterator<Resultado> iterator = lista.iterator();

		while (iterator.hasNext()) {
			Resultado resultado = iterator.next();

			if (metodoAnterior != resultado.getTipoMetodoOrdenacao()) {
				if (impressaoImpar) {
					auxiliar.append("<tr>\n");
				} else {
					auxiliar.append("<tr class=\"alt\">\n");
				}

				auxiliar.append("<td>");
				auxiliar.append("<font size=\"4\">");
				auxiliar.append(TipoMetodoOrdenacao.getNome(resultado
						.getTipoMetodoOrdenacao()));
				auxiliar.append("</font>");
				auxiliar.append("</br>");
				auxiliar.append("Tipo: ");

	auxiliar.append(ClassificacaoMetodo.getNome(TipoMetodoOrdenacao.getTipoClassificacaoMetodo(resultado.getTipo
MetodoOrdenacao())));
				auxiliar.append("</td>\n");

				metodoAnterior = resultado.getTipoMetodoOrdenacao();

				impressaoImpar = !impressaoImpar;
			}

			NumberFormat format = new DecimalFormat("###,###.###");

			auxiliar.append("<td>");

			auxiliar.append("<strong>");
			auxiliar.append(format.format(Math.round(resultado
					.getTempoEmMicroSegundo())));
			auxiliar.append(" µs</strong></br>");

			auxiliar.append("</td>\n");
```

```
                if (metodoAnterior != resultado.getTipoMetodoOrdenacao()) {
                        auxiliar.append("</tr>\n");
                }

                metodoAnterior = resultado.getTipoMetodoOrdenacao();
        }

        auxiliar.append("</tbody></table></div>\n");
        auxiliar.append("</p>");

        adicionar(auxiliar);
}

public void imprimirTabelaEficiencia(ListaEficiencia listaEficiencia)
                throws InvalidAlgorithmParameterException {
        StringBuilder auxiliar = new StringBuilder();

        auxiliar.append("<p>");
        auxiliar.append("<br><hr><center><font size=\"5\" color=#000099><strong>");
        auxiliar.append("Eficiência dos métodos em Tempo de Processamento");
        auxiliar.append("</strong></font></center>");

        auxiliar.append("<div class=\"datagrid\"><table>\n");

        auxiliar.append("<thead>\n");

        for (int i = 0; i < faixaDeQuantidades.length; i++) {
                auxiliar.append("<th>");
                auxiliar.append(faixaDeQuantidades[i]);
                auxiliar.append(" imagens</th>\n");
        }

        auxiliar.append("</tr></thead>\n");
        auxiliar.append("<tbody>\n");

        DecimalFormat format = new DecimalFormat("###,###,###,##0.##");

        auxiliar.append("<tr>\n");
        for (int i = 0; i < faixaDeQuantidades.length; i++) {
                auxiliar.append("<td align=\"left\">");

                Eficiencia eficiencia = listaEficiencia
                                .getEficienciaPorQuantidadeImagens(faixaDeQuantidades[i]);

                Iterator<MetodoTempoProcessamento> iterator = eficiencia
                                .getListaMetodoTempoProcessamento().iterator();
                while (iterator.hasNext()) {
                        MetodoTempoProcessamento metodoTempoProcessamento = iterator
                                        .next();
```

```
                                auxiliar.append(TipoMetodoOrdenacao
                                                .getNome(metodoTempoProcessamento
                                                                .getTipoMetodoOrdenacao()));

                                double porcentagem = metodoTempoProcessamento
                                                .getEficienciaEmPorcentagem();

                                porcentagem = Math.round(porcentagem);

                                if (metodoTempoProcessamento.getEficienciaEmPorcentagem() != 0) {
                                        auxiliar.append("</br>");
                                        auxiliar.append("&#09;&uarr; ");
                                        auxiliar.append(format.format(porcentagem));
                                        auxiliar.append("%");
                                }
                                else
                                {
                                        auxiliar.append("</br>");
                                }

                                auxiliar.append("</br></br>");
                        }

                        auxiliar.append("</td>\n");
                }
                auxiliar.append("</tr>\n");

                auxiliar.append("</tbody></table></div>\n");
                auxiliar.append("</p>");

                adicionar(auxiliar);
        }

        public void inserirDadosProcessamento() {
                StringBuilder auxiliar = new StringBuilder();

                Properties p = System.getProperties();
                Runtime runtime = Runtime.getRuntime();

                // Imprimir Informações do ambiente no HTML
                auxiliar.append("</br>");
                auxiliar.append("<font size=\"4\" color=#000099>");
                auxiliar.append("<strong>Data do Processamento:</strong>");
                auxiliar.append(new SimpleDateFormat("dd/MM/yyyy - E - HH:mm:ss")
                                .format(dataProcessamento).toString());
                auxiliar.append("<br>\n");

                auxiliar.append("</br><strong>Especificação da VM: </strong>");
                auxiliar.append(p.getProperty("java.vm.specification.name"));

                auxiliar.append("</br><strong>Arquitetura: </strong>");
```

```
                auxiliar.append(p.getProperty("sun.arch.data.model"));
                auxiliar.append(" bits");

                auxiliar.append("</br><strong>Versão da VM: </strong>");
                auxiliar.append(p.getProperty("java.vm.specification.version"));

                auxiliar.append("</br><strong>Fabricante da VM: </strong>");
                auxiliar.append(p.getProperty("java.vendor"));

                auxiliar.append("</br><strong>CPU: </strong>");
                auxiliar.append(p.getProperty("sun.cpu.isalist"));

                auxiliar.append("</br><strong>Total de CPUs: </strong>");
                auxiliar.append(runtime.availableProcessors());

                auxiliar.append("</br><strong>Sistema Operacional: </strong>");
                auxiliar.append(System.getProperty("os.name"));

                auxiliar.append("</br>");
                auxiliar.append("</br>");

                auxiliar.append("<strong>Trabalho de APS para Estrutura de Dados 2013</strong><br>\n");
                auxiliar.append("<strong>Campus:</strong> Unip/Marquês<br><br>\n");

                auxiliar.append("<strong>Aluno 1:</strong> B22816-4 MARCIO FERNANDES CRUZ<br>\n");
                auxiliar.append("<strong>Aluno 2:</strong> B54ECG-9 MAURICIO JOSE F OLIVEIRA<br>\n");
                auxiliar.append("<strong>Aluno 3:</strong> T618FA-6 RENATO GOULART RODRIGUES<br>\n");
                auxiliar.append("<strong>Aluno 4:</strong> B20309-9 RODOLFO YURI DE A FONTANA<br>\n");
                auxiliar.append("</font>");

                adicionar(auxiliar);
        }

        public void inicioHTML(String tituloRelatorio) {
                StringBuilder auxiliar = new StringBuilder();

                auxiliar.append("<!DOCTYPE html PUBLIC \"-//W3C//DTD HTML 4.01//EN\">\n");
                auxiliar.append("<html> <head><title>");
                auxiliar.append(tituloRelatorio);
                auxiliar.append(" CC2P13");

                auxiliar.append("</title>\n");
                auxiliar.append("<style type=\"text/css\">\n");
                auxiliar.append(".datagrid table { border-collapse: collapse; text-align: center; width: 100%; }\n");
                auxiliar.append(".datagrid {font: normal 12px/150% Arial, Helvetica, sans-serif; background: #fff; overflow:
hidden; }\n");
                auxiliar.append(".datagrid table td, .datagrid table th { padding: 3px 10px; }\n");
                auxiliar.append(".datagrid table thead th {background:-webkit-gradient( linear, left top, left bottom, color-
stop(0.05, #006699), color-stop(1, #00557F) );\n");
                auxiliar.append("background:-moz-linear-gradient( center top, #006699 5%, #00557F
100% );filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#006699', endColorstr='#00557F');\n");
```

```
                    auxiliar.append("background-color:#006699; color:#FFFFFF; font-size: 12px; font-weight: bold; border-left:
1px solid #0070A8; }\n");
                    auxiliar.append(".datagrid table thead th:first-child { border: none; }.datagrid table tbody td { color: #00496B;
border-left: 1px solid #E1EEF4;font-size: 12px;font-weight: normal; }\n");
                    auxiliar.append(".datagrid table tbody .alt td { background: #E1EEF4; color: #00496B; }.datagrid table tbody
td:first-child { border-left: none; }\n");
                    auxiliar.append(".datagrid table tbody tr:last-child td { border-bottom: none; }.datagrid table tfoot td div
{ border-top: 1px solid #006699;background: #E1EEF4;}\n");
                    auxiliar.append(".datagrid table tfoot td { padding: 0; font-size: 12px } .datagrid table tfoot td div{ padding:
2px; }\n");
                    auxiliar.append(".datagrid table tfoot td ul { margin: 0; padding:0; list-style: none; text-align: right; }\n");
                    auxiliar.append(".datagrid table tfoot  li { display: inline; }\n");
                    auxiliar.append(".datagrid table tfoot li a { text-decoration: none; display: inline-block;  padding: 2px 8px;
margin: 1px;color: #FFFFFF;border: 1px solid #006699;-webkit-border-radius: 3px; -moz-border-radius: 3px; border-radius: 3px;
background:-webkit-gradient( linear, left top, left bottom, color-stop(0.05, #006699), color-stop(1, #00557F) );background:-moz-
linear-gradient( center top, #006699 5%, #00557F
100% );filter:progid:DXImageTransform.Microsoft.gradient(startColorstr='#006699', endColorstr='#00557F');background-
color:#006699; }\n");
                    auxiliar.append(".datagrid table tfoot ul.active,\n");
                    auxiliar.append(".datagrid table tfoot ul a:hover { text-decoration: none;border-color: #006699; color:
#FFFFFF; background: none; background-color:#00557F;}\n");
                    auxiliar.append("</style>\n");

                    auxiliar.append("</head>\n");

                    auxiliar.append("<body>\n");

                    adicionar(auxiliar);
            }

}
```

CLASSE APRESENTAÇÃORESULTADO

```
package br.marciofcruz.apsmetodosordenacao.apresentacao;

import java.awt.Desktop;
import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.security.InvalidAlgorithmParameterException;
import java.util.ArrayList;
import java.util.Date;
import java.util.HashMap;
import java.util.List;

import br.marciofcruz.apsmetodosordenacao.eficiencia.ListaEficiencia;
import br.marciofcruz.apsmetodosordenacao.tipos.TipoCriterioOrdem;
```

```
import br.marciofcruz.apsmetodosordenacao.tipos.TipoExibicaoGrafico;

public class ApresentacaoResultado {

	private int[] faixaDeQuantidades;
	private HashMap<TipoCriterioOrdem, ListaResultado> mapaResultado;
	private HashMap<TipoCriterioOrdem, ListaEficiencia> mapaEficiencia;

	public ApresentacaoResultado(int[] faixaDeQuantidades,
				HashMap<TipoCriterioOrdem, ListaResultado> mapaResultado,
				HashMap<TipoCriterioOrdem, ListaEficiencia> mapaEficiencia) {
		super();

		this.faixaDeQuantidades = faixaDeQuantidades;
		this.mapaResultado = mapaResultado;
		this.mapaEficiencia = mapaEficiencia;
	}

	private String getNomeArquivoHTML(TipoCriterioOrdem tipoCriterioOrdem)
				throws InvalidAlgorithmParameterException {
		switch (tipoCriterioOrdem) {
		case crescente:
			return "resultadoordemcrescente.html";
		case decrescente:
			return "resultadoordemdecrescente.html";
		case embaralhado:
			return "resultadoordemembaralhado.html";
		default:
			throw new InvalidAlgorithmParameterException("Argumento inválido");
		}
	}

	private String getTituloRelatorio(TipoCriterioOrdem tipoCriterioOrdem)
				throws InvalidAlgorithmParameterException {
		switch (tipoCriterioOrdem) {
		case crescente:
			return "Aplicação de Métodos de Ordenação em Imagens em Ordem Crescente";
		case decrescente:
			return "Aplicação de Métodos de Ordenação em Imagens em Ordem Decrescente";
		case embaralhado:
			return "Aplicação de Métodos de Ordenação em Imagens Embaralhadas";
		default:
			throw new InvalidAlgorithmParameterException("Argumento inválido");
		}
	}


	private List<Grafico> gerarGrafico(TipoCriterioOrdem tipoCriterioOrdem)
				throws InvalidAlgorithmParameterException, IOException {

		List<Grafico> retorno = new ArrayList<>();
```

```
                for (TipoExibicaoGrafico tipoExibicaoGrafico : TipoExibicaoGrafico
                                .values()) {
                        retorno.add(new Grafico(mapaResultado.get(tipoCriterioOrdem),
                                        tipoExibicaoGrafico, tipoCriterioOrdem));
                }

                return retorno;
        }

        private void gerarResultadosHTML(Date dataProcessamento, TipoCriterioOrdem tipoCriterioOrdem, List<Grafico>
graficos)
                        throws IOException, InvalidAlgorithmParameterException {
                StringBuilder texto = new StringBuilder();

                AuxiliarHTML html = new AuxiliarHTML(texto, faixaDeQuantidades, dataProcessamento);

                String tituloRelatorio = getTituloRelatorio(tipoCriterioOrdem);

                html.inicioHTML(tituloRelatorio);

                ListaResultado lista = mapaResultado.get(tipoCriterioOrdem);

                List<Resultado> resultados = lista.getResultados();

                html.imprimirTabelaComparativoMetodos(tituloRelatorio, tipoCriterioOrdem, resultados);

                html.imprimirTabelaEficiencia(mapaEficiencia.get(tipoCriterioOrdem));

                html.inserirImagem(graficos);

                html.inserirDadosProcessamento();

                html.adicionar("</body></html>");

                texto = html.getTexto();

                File f = new File(getNomeArquivoHTML(tipoCriterioOrdem));
                BufferedWriter bw = new BufferedWriter(new FileWriter(f));

                bw.write(texto.toString());

                bw.close();

                Desktop d = Desktop.getDesktop();
                try {
                        d.browse(f.toURI());
                } catch (IOException e) {
                        System.out.println(e);
                }
        }
```

```
	public void apresentarResultados()
				throws InvalidAlgorithmParameterException, IOException {

		Date dataProcessamento = new Date();

		for (TipoCriterioOrdem tipoCriterioOrdem : TipoCriterioOrdem.values()) {

				List<Grafico> graficos = gerarGrafico(tipoCriterioOrdem);

				gerarResultadosHTML(dataProcessamento, tipoCriterioOrdem, graficos);
		}
	}

}
```

Pacote br.marciofcruz.apsmetodosordenacao.ordenacao

CLASSE ABSTRATA ORDENAÇÃO

```
package br.marciofcruz.apsmetodosordenacao.ordenacao;

/**
 * Esta é uma classe abstrata que será utilizada por vários métodos de ordenação a serem implementados
 * @author B22816-4 Marcio Fernandes Cruz
 * @see java.lang.Object
 * @version 1.00
 * @since 1.00
 */

import java.security.InvalidAlgorithmParameterException;
import java.util.Arrays;


import br.marciofcruz.apsmetodosordenacao.apresentacao.Resultado;
import br.marciofcruz.apsmetodosordenacao.imagem.ImagemSatelite;
import br.marciofcruz.apsmetodosordenacao.tipos.TipoMetodoOrdenacao;

public abstract class Ordenacao {

	protected ImagemSatelite[] imagens;
	private String nomeClasse = "";
	private Resultado resultado;
	private int tamanho;

	/**
	 * Construtor Ordenacao O construtor recebe o array de imagens que deve ser
	 * ordenado
	 *
```

```
 * @param imagens
 * @throws InvalidAlgorithmParameterException
 */

public Ordenacao() {

        String nomeCompletoClasse = getClass().toString();
        this.nomeClasse = nomeCompletoClasse.substring(
                        nomeCompletoClasse.indexOf(".") + 1,
                        nomeCompletoClasse.length());
}

public int getTamanho() {
        return tamanho;
}

/**
 * Este método vai ser chamado por quem instanciar determinado método de
 * ordenção Já possui todo o controle contagem dos indices do resultado como
 * tempo, comparaçóes, etc
 * @throws InvalidAlgorithmParameterException
 */

public Resultado processar(ImagemSatelite[] imagens) throws InvalidAlgorithmParameterException {
        this.imagens = imagens.clone();
        this.tamanho = imagens.length;
        resultado = new Resultado(TipoMetodoOrdenacao.getTipoPorNome(nomeClasse), imagens.length);

        long tempoInicial, tempoFinal;

        tempoInicial = System.nanoTime();
        Ordenar(); // método será definido na classe que
                                                                        // extende Ordenação
        tempoFinal = System.nanoTime();

        resultado.setTempoemNs(tempoFinal-tempoInicial);

        testarOrdenacaoArray();

        return resultado;
}

/**
 * Este método deve ser implementado para quem estender este classe Como
 * este método é utilizado internamente nesta classe abstrata, repassamos
 * como retorno o resultado
 *
 * @param resultado
 * @return resultado
 */
```

```
	protected abstract void Ordenar();

	/*
	 * Depois de ordenar, temos que chamar este método para certificar que o
	 * método implementado fez a ordenação crescente corretamente
	 */
	private void testarOrdenacaoArray() {
		int ultimaChave = -1;

		for (int i = 0; i < imagens.length; i++) {
			if (imagens[i].hashCode() < ultimaChave) {
				throw new AssertionError(
						"Método de ordenação foi aplicado mas, array de imagens não está
ordenado corretamente: "
+ "\nMétodo: "
+
TipoMetodoOrdenacao.getNome(resultado.getTipoMetodoOrdenacao())
+ "\nQtd. Imagens: "
+ resultado.getQuantidadeImagens());

			}

			ultimaChave = imagens[i].hashCode();
		}
	}

	@Override
	public String toString() {
		return nomeClasse + " " + "Itens: " + resultado.getQuantidadeImagens()
				+ "\t" + "Tempo em ns: " + resultado.getTempoEmMicroSegundo() + "\t"+
				Arrays.toString(imagens);

	}
}
```

CLASSE RADIXSORT

```
package br.marciofcruz.apsmetodosordenacao.ordenacao;

import br.marciofcruz.apsmetodosordenacao.imagem.ImagemSatelite;

public class RadixSort extends Ordenacao {

	private int bits = 4;

	@Override
	protected void Ordenar() {
		ImagemSatelite[] imagensClone = new ImagemSatelite[getTamanho()];
		ImagemSatelite[] imagensCloneOrig = imagensClone;

		int direitaReferencia = 0;
```

```
                for (int mascara = ~(-1 << bits); mascara != 0; mascara <<= bits, direitaReferencia += bits) {

                        int[] contaLista = new int[1 << bits];

                        for (int posicao = 0; posicao < getTamanho(); posicao++) {
                                int chave = (imagens[posicao].hashCode() & mascara) >> direitaReferencia;

                                ++contaLista[chave];
                        }

                        for (int posicao = 1; posicao < contaLista.length; posicao++) {
                                contaLista[posicao] += contaLista[posicao - 1];
                        }

                        for (int posicao = getTamanho() - 1; posicao >= 0; posicao--) {

                                int chave = (imagens[posicao].hashCode() & mascara) >> direitaReferencia;

                                --contaLista[chave];

                                imagensClone[contaLista[chave]] = imagens[posicao];
                        }

                        ImagemSatelite[] swap = imagensClone;
                        imagensClone = imagens;
                        imagens = swap;
                }

                if (imagens == imagensCloneOrig) {
                        System.arraycopy(imagens, 0, imagensClone, 0, getTamanho());
                }
        }
}
```

CLASSE INSERTIONSORT

```
package br.marciofcruz.apsmetodosordenacao.ordenacao;

import br.marciofcruz.apsmetodosordenacao.imagem.ImagemSatelite;

/**
 * Esta classe implementa a definição da técnica InsertionSort Esta estrutura de
 * dados utilizamos um sub-aranjo para controlar a inserção
 *
 * Algoritmo do tipo: Inserção.
 *
 * @author B22816-4 Marcio Fernandes Cruz
 * @see br.marciofcruz.apsmetodosordenacao.ordenacao.Ordenacao
 * @version 1.00
 * @since 1.00
 */
```

```
public class InsertionSort extends Ordenacao {

	@Override
	protected void Ordenar() {
		if (imagens.length > 1) {

			short ponteiroSubArranjo = 1;

			while (ponteiroSubArranjo < getTamanho()) {

				// varrer do final até o início do sub-arranjo
				short ponteiroAuxiliar = ponteiroSubArranjo;

				boolean sairLoop = false;

				for (short i = (short) (ponteiroAuxiliar-1); i >= 0 && !sairLoop; i--) {

					if (imagens[ponteiroAuxiliar].hashCode() < imagens[i]
							.hashCode()) {
						ImagemSatelite swap = imagens[i];
						imagens[i] = imagens[ponteiroAuxiliar];
						imagens[ponteiroAuxiliar] = swap;

						ponteiroAuxiliar = i;
					}
					else
					{
						sairLoop = true;
					}
				}

				ponteiroSubArranjo++;
			}
		}
	}

}
```

CLASSE QUICKSORT

```
package br.marciofcruz.apsmetodosordenacao.ordenacao;

import br.marciofcruz.apsmetodosordenacao.imagem.ImagemSatelite;

/**
 * Esta classe implementa a definição da técnica Quick Sort, que é uma técnica
 * que utiliza o conceito de dividir para conquistar e, por isso, seu tipo é de
 * particionamento. Sua desvantagem é que temos que ter outra estrutura de dados
 * instanciada, no caso, um array auxiliar
 *
```

```
 * Algoritmo do tipo: Particionamento
 *
 * @author B22816-4 Marcio Fernandes Cruz
 * @see br.marciofcruz.apsmetodosordenacao.ordenacao.Ordenacao
 * @version 1.00
 * @since 1.00
 */

public class QuickSort extends Ordenacao {

        private void quickSort(int indiceInicio, int indiceFim) {
                int menorQue = indiceInicio;

    int maiorQue = indiceFim;

    ImagemSatelite swap;

    ImagemSatelite meio = imagens[(indiceInicio + indiceFim) / 2];

    while (menorQue < maiorQue) {

       while (imagens[menorQue].hashCode() < meio.hashCode()) {
          menorQue++;
       }

       while (imagens[maiorQue].hashCode() > meio.hashCode()) {
          maiorQue--;
       }

       if (menorQue <= maiorQue) {

          swap = imagens[menorQue];

          imagens[menorQue] = imagens[maiorQue];

          imagens[maiorQue] = swap;

          menorQue++;

          maiorQue--;
       }

    }

    if (indiceInicio < maiorQue) {

       quickSort(indiceInicio, maiorQue);

    }

    if (menorQue < indiceFim) {
```

```
        quickSort(menorQue, indiceFim);

    }

        }

        @Override
        protected void  Ordenar() {
                quickSort(0, getTamanho()-1);
        }

}
```

CLASSE SELECTIONSORT

```
package br.marciofcruz.apsmetodosordenacao.ordenacao;

import br.marciofcruz.apsmetodosordenacao.imagem.ImagemSatelite;

/**
 * Esta classe implementa a definição da técnica SelectionSort Esta estrutura de
 * dados define encontrar a menor chave a partir de um ponteiro que devemos
 * aumentar a cada troca
 *
 * Algoritmo do tipo: Seleção. Análise de complexidade de algoritmo:
 *
 * @author B22816-4 Marcio Fernandes Cruz
 * @see br.marciofcruz.apsmetodosordenacao.ordenacao.Ordenacao
 * @version 1.00
 * @since 1.00
 */

public class SelectionSort extends Ordenacao {

        @Override
        protected void Ordenar() {
                short pivo = 0;

                short tamanho = (short) getTamanho();

                while (pivo < tamanho) {

                        for (short i = (short) (pivo + 1); i < tamanho; i++) {

                                if (imagens[pivo].hashCode() > imagens[i].hashCode()) {

                                        ImagemSatelite swap = imagens[pivo];
                                        imagens[pivo] = imagens[i];
                                        imagens[i] = swap;
                                }
```

```
                        }

                        pivo++;
                }
        }

}
```

CLASSE MERGESORT

```
package br.marciofcruz.apsmetodosordenacao.ordenacao;

import br.marciofcruz.apsmetodosordenacao.imagem.ImagemSatelite;

/**
 * Esta classe implementa a definição da técnica Merge Sort, que é uma técnica que utiliza
 * o conceito de dividir para conquistar e, por isso, seu tipo é de particionamento.
 * Sua desvantagem é que temos que ter outra estrutura de dados instanciada, no caso, um array
 * auxiliar
 *
 * Algoritmo do tipo: Particionamento
 *
 * @author B22816-4 Marcio Fernandes Cruz
 * @see br.marciofcruz.apsmetodosordenacao.ordenacao.Ordenacao
 * @version 1.00
 * @since 1.00
 */

public class MergeSort extends Ordenacao {

        private ImagemSatelite[] arrayAuxiliar;

        private void mergeSort(int indiceInicio, int indiceFim) {
                if (indiceInicio < indiceFim) {
                        int indiceMetade = indiceInicio + (indiceFim - indiceInicio) / 2;

                        mergeSort(indiceInicio, indiceMetade); // ordenando primeira metade
                        mergeSort(indiceMetade + 1, indiceFim); // Ordenando a segunda metade

                        // combinando as partes
                        combinar(indiceInicio, indiceMetade, indiceFim);
                }
        }

        private void combinar(int indiceInicio, int indiceMetade, int indiceFim) {

                // Copiar as partes para os arrays
                for (int i = indiceInicio; i <= indiceFim; i++) {
                        arrayAuxiliar[i] = imagens[i];
                }
```

```
                int i = indiceInicio;
                int j = indiceMetade + 1;
                int k = indiceInicio;

                // Copiar os valores menores do primeiro array esquerdo ou direito para
                // o array ImagemSatelite
                while (i <= indiceMetade && j <= indiceFim) {
                        if (arrayAuxiliar[i].hashCode() <= arrayAuxiliar[j].hashCode()) {
                                imagens[k] = arrayAuxiliar[i];
                                i++;
                        } else {
                                imagens[k] = arrayAuxiliar[j];
                                j++;
                        }

                        k++;
                }

                // copiar o resto do array da primeira parte até o Array Original
                while (i <= indiceMetade) {
                        imagens[k] = arrayAuxiliar[i];
                        k++;
                        i++;
                }

        }

        @Override
        protected void Ordenar() {
                arrayAuxiliar = new ImagemSatelite[getTamanho()]; // criando array auxiliar

                mergeSort(0, getTamanho() - 1);
        }

}
```

CLASSE BUBBLESORT

```
package br.marciofcruz.apsmetodosordenacao.ordenacao;

import br.marciofcruz.apsmetodosordenacao.imagem.ImagemSatelite;

/**
 * Esta classe implementa a definição da técnica BubbleSort Esta estrutura de
 * dados define que devemos trocar valores em posições subsequentes. E,
 * repetimos este processo para todos os itens do Array.
 *
 * Algoritmo do tipo: Troca.
 *
 * @author B22816-4 Marcio Fernandes Cruz
```

```
 * @see br.marciofcruz.apsmetodosordenacao.ordenacao.Ordenacao
 * @version 1.00
 * @since 1.00
 */

public class BubbleSort extends Ordenacao {

	@Override
	protected void Ordenar() {

		short limite = (short) getTamanho();

		boolean houveTroca = true;

		while (limite > 0 && houveTroca) {

			houveTroca = false;

			for (short i = 0; i < limite-1; i++) {

				if (imagens[i + 1].hashCode() < imagens[i].hashCode()) {
					ImagemSatelite swap = imagens[i];
					imagens[i] = imagens[i + 1];
					imagens[i + 1] = swap;

					houveTroca = true;
				}
			}

			limite--;
		}
	}

}
```

Pacote br.marciofcruz.apsmetodosordenacao.main

CLASSE PRINCIPAL

```
package br.marciofcruz.apsmetodosordenacao.main;

/**
 * Esta classe é a principal do sistema e faz as seguintes fases do processo:
 * 1 - Obtenção das imagens ordenadas de determinado diretório
 * 2 - Alimentação de N faixas de quantidade de imagens ordenadas e embaralhadas
 * 3 - Execução dos métodos de ordenação nas várias faixas dos arrays embaralhados
```

APRESENTAÇÃO DOS RESULTADOS

```
 * @author B22816-4 Marcio Fernandes Cruz
 * @see java.lang.Object
 * @version 1.00
 * @since 1.00
 */

import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.IOException;
import java.io.InputStreamReader;
import java.security.InvalidAlgorithmParameterException;
import java.util.ArrayList;
import java.util.Collections;
import java.util.HashMap;
import java.util.Iterator;
import java.util.List;
import java.util.Scanner;

import br.marciofcruz.apsmetodosordenacao.apresentacao.ApresentacaoResultado;
import br.marciofcruz.apsmetodosordenacao.apresentacao.ListaResultado;
import br.marciofcruz.apsmetodosordenacao.apresentacao.Resultado;
import br.marciofcruz.apsmetodosordenacao.eficiencia.Eficiencia;
import br.marciofcruz.apsmetodosordenacao.eficiencia.ListaEficiencia;
import br.marciofcruz.apsmetodosordenacao.imagem.ArrayImagens;
import br.marciofcruz.apsmetodosordenacao.tipos.TipoCriterioOrdem;
import br.marciofcruz.apsmetodosordenacao.tipos.TipoMetodoOrdenacao;

public class Principal {

	/**
	 * Vamos fazer teste com N faixas de quantidade de imagens
	 */

	final static int limiteQuantidade = 10000;

	private static int[] faixaPadrao = { 10, 100, 2000, 4000, 6000, 8000, 10000};
	private static int[] faixaDeQuantidades;

	private HashMap<TipoCriterioOrdem, ListaResultado> mapaResultado = new HashMap<>();
	private HashMap<TipoCriterioOrdem, ListaEficiencia> mapaEficiencia = new HashMap<>();

	private static Scanner in;

	private ArrayImagens[] criarArrayImagens() throws FileNotFoundException {

		String path = System.getProperty("user.dir") + "\\imagensSatelite";

		ArrayImagens[] lista = new ArrayImagens[faixaDeQuantidades.length];
		for (short i = 0; i < faixaDeQuantidades.length; i++) {
			String mensagem = Integer.toString(i + 1) + "/"
					+ Integer.toString(faixaDeQuantidades.length)
					+ " - Criando array embaralhado para "
					+ faixaDeQuantidades[i] + " imagens.";
			System.out.println(mensagem);

			lista[i] = new ArrayImagens(faixaDeQuantidades[i], path);
		}

		return lista;
	}

	private void executarProcessoOrdenacao(ArrayImagens[] lista)
			throws InvalidAlgorithmParameterException {
		for (TipoCriterioOrdem ordem : TipoCriterioOrdem.values()) {

			ListaResultado listaResultado = new ListaResultado();

			for (TipoMetodoOrdenacao metodo : TipoMetodoOrdenacao.values()) {

				System.out.println(" -" + TipoMetodoOrdenacao.getNome(metodo)
						+ " em " + TipoCriterioOrdem.getNome(ordem));
```

```
                                        for (int i = 0; i < faixaDeQuantidades.length; i++) {
                                                listaResultado.adicionar(TipoMetodoOrdenacao
                                                                .getInstanciaDe(metodo).processar(
                                                                                lista[i].getImagensOrdem(ordem)));
                                        }
                                }

                                mapaResultado.put(ordem, listaResultado);
                        }

                }

                private static String listaFaixaValores(int[] faixa) {
                        StringBuilder s = new StringBuilder();

                        for (int i = 0; i < faixa.length; i++) {
                                s.append(faixa[i]);

                                if (i != faixa.length - 1) {
                                        s.append(", ");
                                }
                        }

                        return s.toString();
                }

                private void gerarMapaEficienciaMetodos()
                                throws InvalidAlgorithmParameterException {

                        for (TipoCriterioOrdem tipoCriterioOrdem : TipoCriterioOrdem.values()) {

                                ListaEficiencia listaEficiencia = new ListaEficiencia();

                                for (int i = 0; i < faixaDeQuantidades.length; i++) {

                                        Eficiencia eficiencia = new Eficiencia(faixaDeQuantidades[i]);

                                        for (TipoMetodoOrdenacao tipoMetodoOrdenacao : TipoMetodoOrdenacao
                                                        .values()) {

                                                ListaResultado listaResultado = mapaResultado
                                                                .get(tipoCriterioOrdem);
                                                Resultado resultado = listaResultado
                                                                .getRetornoPorQuantidadeImagens(
                                                                                tipoMetodoOrdenacao,
faixaDeQuantidades[i]);

                                                eficiencia.adicionar(tipoMetodoOrdenacao,
                                                                resultado.getTempoemNanoSegundo());
                                        }

                                        listaEficiencia.adicionar(eficiencia);
                                }

                                mapaEficiencia.put(tipoCriterioOrdem, listaEficiencia);
                        }
                }

                public void processar() throws IOException,
                                InvalidAlgorithmParameterException, InstantiationException,
                                IllegalAccessException {
                        System.out
                                        .println("Passo 1 de 4 - Criando os TADS de imagens embaralhadas:");
                        ArrayImagens[] lista = this.criarArrayImagens();

                        System.out
                                        .println("\nPasso 2 de 4 - Executando métodos de ordenações:");
                        executarProcessoOrdenacao(lista);

                        System.out
                                        .println("\nPasso 3 de 4 - Criando comparativo da Eficiência dos métodos...");
                        gerarMapaEficienciaMetodos();

                        System.out.println("\nPasso 4 de 4 - Gerando a página de resultado...");
                        ApresentacaoResultado apresentacao = new ApresentacaoResultado(
```

```
                                faixaDeQuantidades, mapaResultado, mapaEficiencia);
        apresentacao.apresentarResultados();
}

private static int[] lerNovaFaixaQuantidade() {

        System.out.println("Limite de quantidade: " + limiteQuantidade);

        List<Integer> lista = new ArrayList<>();

        int numero = 0;

        do {
                System.out.println("Digite a faixa de quantidades ou \"0\" para finalizar:");

                in = new Scanner(System.in);

                numero = in.nextInt();

                if (numero > limiteQuantidade || numero <= 1) {
                        System.out.println("Quantidade deve ser entre 2 e "
                                        + limiteQuantidade);
                } else {
                        if (lista.contains(numero)) {
                                System.out.println("Elmento já foi inserido");
                        } else {
                                lista.add(numero);
                                Collections.sort(lista);

                                System.out.println(lista);
                        }
                }

        } while (numero != 0);

        int[] retorno = new int[lista.size()];
        int posicao = 0;
        Iterator<Integer> iterator = lista.iterator();
        while (iterator.hasNext()) {
                retorno[posicao] = iterator.next();
                posicao++;
        }

        return retorno;
}

private static void definirOpcaoFaixa() throws IOException {

        StringBuilder s = new StringBuilder();

        s.append("Trabalho de APS CC2P13/CC3P13 - 2013 - Marcio, Mauricio, Renato e Rodolfo\n");
        s.append("Comparativo entre métodos de Ordenação\n\n");

        s.append("Faixa Padrão de Quantidades: ");
        s.append(listaFaixaValores(faixaPadrao));

        s.append("\n");
        s.append("Deseja manter estes valores (<S>/N)?");

        System.out.println(s.toString());

        String opcao = null;
        BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
        try {
                opcao = br.readLine();
        } catch (IOException ioe) {
                System.out.println("Erro ao ler string");
        }

        opcao = opcao.toUpperCase();

        switch (opcao) {
        case "S":
                faixaDeQuantidades = faixaPadrao;
                break;
        case "N":
```

```
                        System.out.println("\n\n");
                        faixaDeQuantidades = lerNovaFaixaQuantidade();
                        break;
                default:
                        faixaDeQuantidades = faixaPadrao;
                }
        }

        public static void main(String[] args)
                        throws InvalidAlgorithmParameterException, IOException,
                        InstantiationException, IllegalAccessException {
                Principal principal = new Principal();

                Principal.definirOpcaoFaixa();

                principal.processar();
        }

}
```

10 RELATORIO COM AS LINLHAS DE CODIGO DAS PAGINAS

public enum = criação de classe de tipo enumerados
throw new = lança uma exeção
switch = multipla seleção;
for = iteração;
if =comparação simples;
public abstract class = definição de classe abastrata
import = importação de pacotes
protected = modificador de acesso
private = modificador de acesso
/**....*/ = Comentário JavaDoc
System.nanoTime() = Retorno de tempo em nanosegundos
protected abstract void = definição de método abstrato;
ImagemSatelite[] = array de objetos do tipo ImagemSatelite;
System.arraycopy = faz cópia de array de objetos;
Character.isDigit(char c) = método estático para verificar se digito é um número;
Integer.parseInt = método estático para converter se String para Inteiro;
throws = checked exception;
(short) (int i) = cast de inteiro para short;
Collections.sort(List lista) = ordena uma lista em ordem crescente
Collections.shuffle(List lista) = embaralha uma lista de objetos
Collections.reverse(Lista lista) = deixa uma lista em ordem decrescente
Iterator = recurso para fazer um for each em uma lista de objetos
int[] faixaPadrao = { 10, 100} = array de constantes
HashMap = mapa de objetos
Scanner = classe especializada em ler dados do usuário via teclado;
new = criação de objetos e referência do mesmo a um denominador (nome de variavel);
static = criação de métodos estáticos (não precisa a classe estar instanciada para ser chamado);
Scanner.nextInt() = obriga a ler inteiros do teclado, caso não ler, carrega exceção;
lista.size() = retorna o tamanho de uma lista de objetos, equivalente a length() de um vetor de objetos de tipos primitivos;

StringBuilder = Pelo fato da classe String ser imutável, instanciamos esta classe para alimentarmos uma sequência de caracteres e, depois, no final, convertemos para String.

BIBLIOGRAFIAS

http://pt.wikipedia.org/wiki/Insertion_sort
http://pt.wikipedia.org/wiki/Quicksort
http://www.dsc.ufcg.edu.br/~pet/jornal/abril2013/materias/historia_da_computacao.html
http://amigonerd.net/exatas/informatica/algoritmos-de-ordenacao
https://en.wikipedia.org/wiki/Radix_sort
http://www.youtube.com/watch?v=xhr26ia4k38
http://en.wikipedia.org/wiki/Merge_sort
http://www.cse.iitk.ac.in/users/dsrkg/cs210/applets/sortingII/mergeSort/mergeSort.html
https://www.youtube.com/watch?v=PKCMMSXQyJE
Livro: Métodos de Ordenação Interna (implementação,Análise e desempenho em Delphi)
Autores: Àlvaro Borges de Oliveira
Adriana Prada
Reginaldo Rubens da Silva