

UNIVERSIDADE PAULISTA (UNIP)

CIÊNCIAS DA COMPUTAÇÃO

Eric Santos da Silva – RA: F015BA7 Turma CC4P20

Fábio Miguel Naumes – RA: N435CE1 Turma CC4P20

Leonardo Algarve Rezende – RA: N3950A9 Turma CC4P20

Wesley Santos Coelho – RA: N479124 Turma CC4P20

Atividade Prática Supervisionada (APS)

**Sistema de Geoprocessamento de Imagens - Análise de
Performance de Algoritmos de Ordenação de Dados**

São Paulo

2020

ÍNDICE

Objetivo do Trabalho.....	1
Introdução.....	1
Referencial Teórico.....	3
Desenvolvimento.....	8
Resultados e Discussão.....	20
Considerações Finais.....	27
Referências Bibliográficas.....	29
Código Fonte.....	31
Fichas.....	83

Objetivo do Trabalho

O objetivo do trabalho é a criação de um sistema computacional de geoprocessamento, que faça a captação de dados, sejam eles internos ou externos, para o estudo da performance dos algoritmos de ordenação abordados, utilizando a métrica de tempo levado para a ordenação de um vetor em milissegundos.

Introdução

Durante nosso cotidiano nos deparamos com inúmeros problemas, os algoritmos surgem como um tipo de solução específica, mas o que é um algoritmo? Os algoritmos são a descrição passo a passo dos procedimentos necessários para a solução de um problema, sendo muito ligado com o universo lógico, para o melhor entendimento de seu conceito, muitas vezes, ele é comparado a uma receita de bolo.

Quando partimos da simples premissa de ordenar numericamente uma sequência de números, com uma simples pesquisa um imenso leque de soluções possíveis se abre para nós, conhecidos como algoritmos de ordenação eles são o tipo de resolução mais comum na área computacional, pois devido a sua estrutura, podem ser implementados nos mais diversos casos e plataformas, e existem vários tipos deles, cada um com suas particularidades, são alguns dos mais populares: Insertion Sort(Ordenação por Inserção), Selection Sort(Ordenação por Seleção), Bubble Sort(Ordenação por “Bolhas”), Merge Sort(Ordenação por Mistura), Comb Sort(Ordenação por Combos), Quick Sort(Ordenação Rápida), Heap Sort(Ordenação por Pilha), Shell Sort(Ordenação por “Concha”) e entre muitas outras.

Para entendermos as particularidades de cada método de ordenação, segue uma breve descrição do funcionamento de cada um:

- Insertion Sort

O Insertion Sort consiste em percorrer toda a lista da esquerda para a direita, deixando os elementos mais à esquerda em ordem.

- Selection Sort

O Selection Sort consiste em escolher um número a partir do primeiro e compara-lo com os números a sua direita, sempre que um valor menor que o número escolhido é encontrado, ele passa a ser o primeiro elemento, ou seja, o menor número. Quando um número menor que o escolhido é encontrado, ele passa a ser o escolhido.

- Bubble Sort

O Bubble Sort consiste em compararmos um elemento com o elemento seguinte, se o primeiro elemento é menor que o segundo eles permanecem na mesma ordem, porém se o segundo elemento é menor que o primeiro os dois trocam de lugar e as comparações continuam.

- Merge Sort

O Merge Sort consiste na ideia básica de “Dividir e Conquistar” utilizando a recursividade, ele calcula o ponto médio do conjunto, depois o divide em dois subconjuntos e repete o primeiro passo em cada um deles recursivamente e por fim todos os subarranjos são unidos.

- Comb Sort

O Comb Sort consiste na reordenação contínua de pares (que são separados entre si por um intervalo denominado salto, que nada mais é do que a distância para um elemento e o outro). É semelhante ao Bubble Sort mas sua diferença é que o salto pode ser maior que 1, baseando-se no fator de encolhimento (normalmente 1,3).

- Quick Sort

O Quick Sort consiste em escolher um elemento da lista como pivô, depois ordenar os elementos da lista: os números menores que o pivô ficam a esquerda dele enquanto os elementos maiores ficam a sua direita, formando duas sub listas ainda não ordenadas, por fim recursivamente ordenar as duas sublistas.

- Heap Sort

O Heap Sort consiste na utilização de uma estrutura de árvore denominada heap para ordenar os elementos à medida que os insere na heap, quando todas as inserções terminam os elementos são sucessivamente retirados da raiz da heap na ordem solicitada.

- Shell Sort

O Shell Sort consiste em dividir o conjunto de elementos em vários conjuntos menores e depois aplicar a ordenação por inserção neles.

Referencial Teórico

Com a finalidade de nos aprofundarmos tanto no funcionamento quanto na comparação dos algoritmos em si, nos referenciais teóricos abordaremos somente os algoritmos de ordenação discutidos neste semestre, são eles: Bubble Sort, Insertion Sort, Selection Sort e Quick Sort.

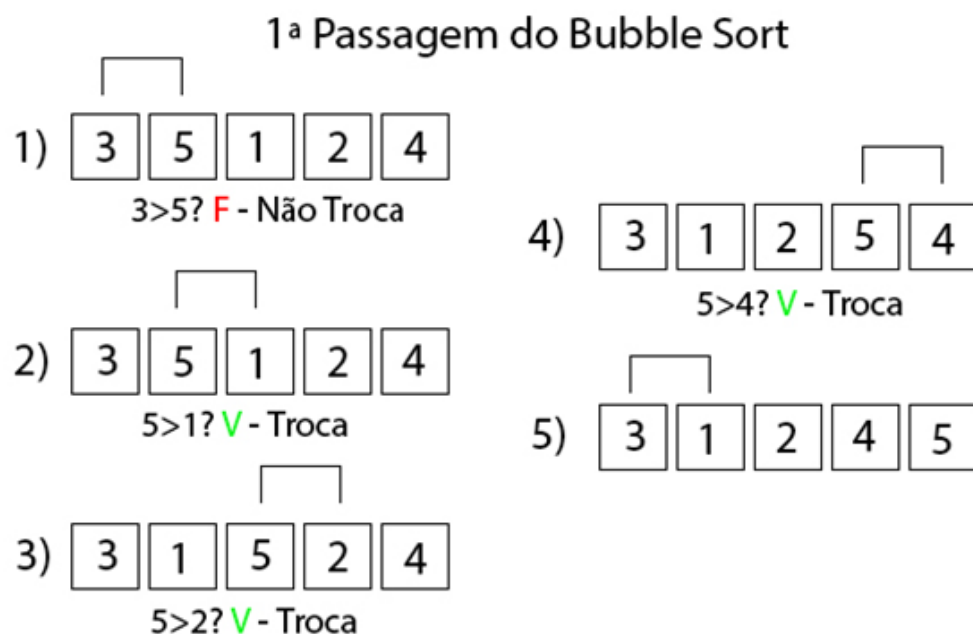
- Bubble Sort

O Bubble Sort é o algoritmo mais simples dentre todos os citados quando se trata de implementação, porém ele não apresenta um bom desempenho em comparação aos outros. Seu conceito é iterar sempre a lista inteira de itens quantas vezes forem necessárias até que os itens estejam corretamente ordenados. Durante

a iteração, o algoritmo deve comparar e ordenar pares de valores, o item iterado e o item logo à sua direita. Para o Bubble Sort, o melhor caso possível é todos os itens estarem na ordem correta, fazendo com que ele percorra o vetor somente uma vez, e o pior caso possível é quando os menores elementos estão todos no final da lista, fazendo com que o algoritmo percorra todos os elementos em uma iteração.

Basicamente podemos contar como vantagens do Bubble Sort a sua fácil implementação e o fato que ele não utiliza armazenamento temporário, alocando menos memória

Exemplo de funcionamento de uma iteração do Bubble Sort:



*Imagem retirada do blog do Profº Davi Guimarães

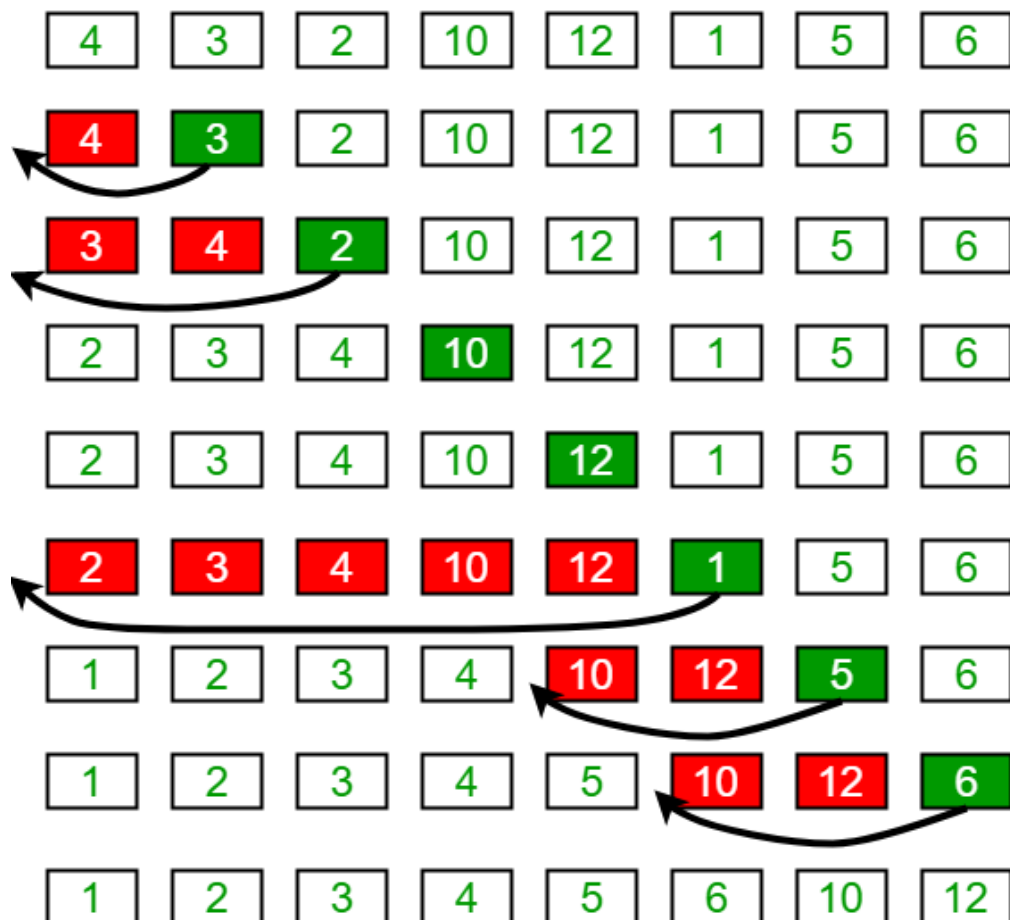
- Insertion Sort

O Insertion Sort é um algoritmo de ordenação que tem um funcionamento parecido com uma mão cheia de cartas quando jogamos baralho, por exemplo imagine que todas as cartas em sua mão estão ordenadas e você acaba de puxar uma nova carta aleatória, para esta carta entrar na ordem junto com as demais é

necessário que você a compare com as outras que estão na mão. O Insertion Sort funciona exatamente assim, os elementos são ordenados à medida que são inseridos no conjunto, sempre começando a comparar pelo último elemento da direita, de forma que os elementos de valor maior são movidos para direita para que o elemento se encaixe no conjunto. As vantagens do Insertion Sort são que ele comporta naturalmente, o que significa que ele trabalha menos quando uma parte do vetor já está ordenada, e que ele não rearranja elementos de mesma chave, sua desvantagem, porém é que sempre é preciso deslocar o vetor para que o elemento seja inserido corretamente no conjunto.

Exemplo de funcionamento do Insertion Sort:

Insertion Sort Execution Example

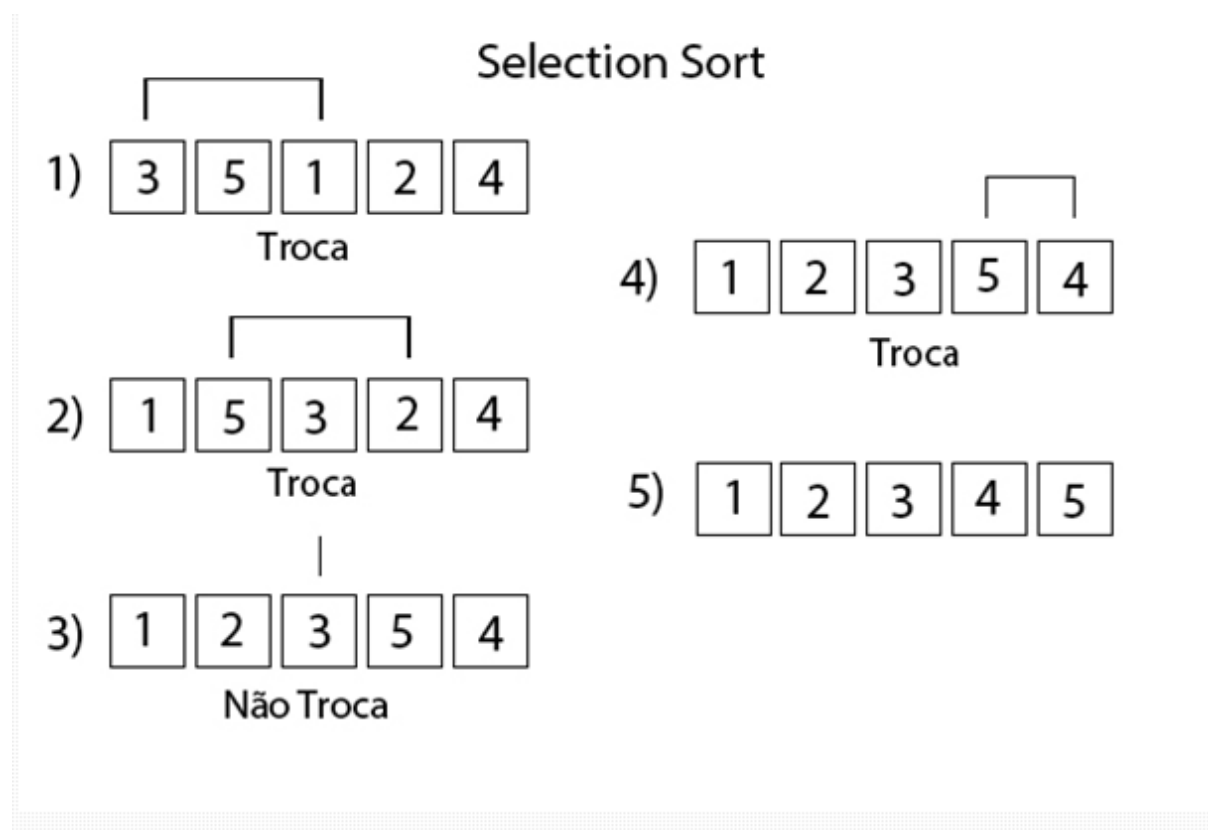


*Imagem retirada de site GeeksforGeeks, artigo Insertion Sort

- Selection Sort

O Selection Sort é um algoritmo que baseia seu funcionamento na ideia de sempre passar o menor valor para a primeira posição, o segundo menor valor para a segunda posição, e assim em diante, percorrendo todos os elementos. A partir do primeiro número, um número é selecionado e comparado com todos os outros números à sua direita, se um número menor é encontrado, este número selecionado ocupa a posição do número encontrado e o próximo a ser selecionado é este número encontrado. No caso de não ser encontrado nenhum número menor que o selecionado, ele ocupará a primeira posição do vetor (ou seja, a primeira posição do primeiro número escolhido). A vantagem do Selection Sort é sua quantidade reduzida de movimentos entre os elementos em comparação com outros algoritmos de ordenação, já suas desvantagens são que não importa se o conjunto já está ordenado ou não, pois o algoritmo ainda mantém seu número de comparações, e o fato que ele não é um algoritmo estável, ou seja, registros com chaves iguais nem sempre manterão a mesma posição relativa do início do processo de ordenação.

Exemplo de funcionamento do Selection Sort:

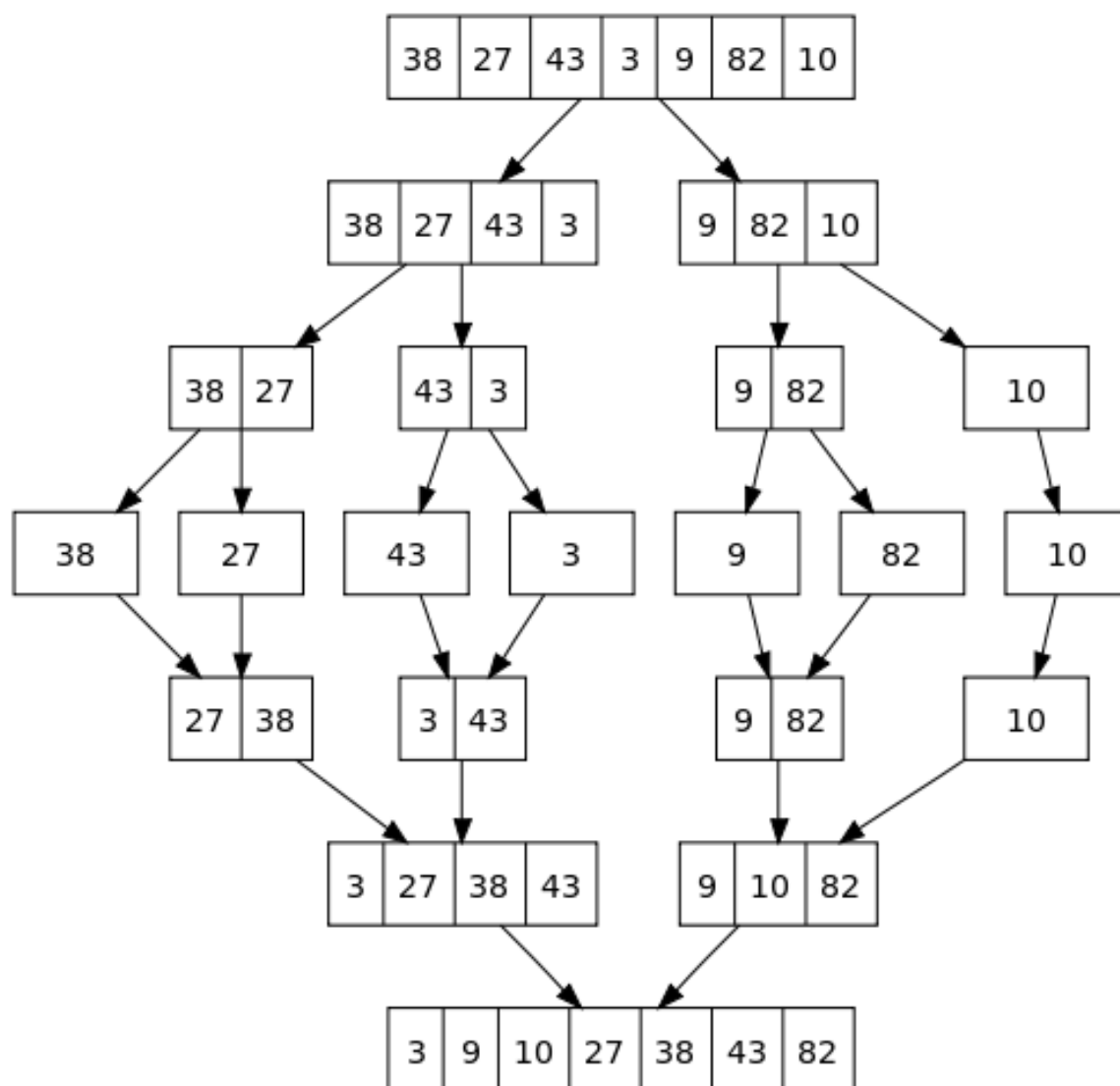


*Imagem retirada do site DevMedia, Artigo Algoritmos de Ordenação:Análise e Comparação

- Quick Sort

Tratando-se da ordenação por comparação, o Quick Sort é o algoritmo mais eficiente. Seu funcionamento consiste na escolha de um elemento denominado pivô, a partir dele o vetor é organizado de forma que todos os elementos menores que ele estejam a sua esquerda e todos os elementos maiores que ele estejam a sua direita, depois disso, com auxílio da recursividade, o mesmo processo ocorre a esquerda e a direita do pivô enquanto o mesmo permanece no lugar e, assim sucessivamente até que todo o vetor esteja ordenado. As vantagens do Quick Sort são a sua rapidez enorme devido a seu laço interno simples, memória auxiliar para a pilha de recursão pequena e alta eficiência, entretanto suas desvantagens são a sua implementação delicada e difícil, fazendo com que enganos menores levem a efeitos indesejados para algumas saídas de dados, e sua instabilidade.

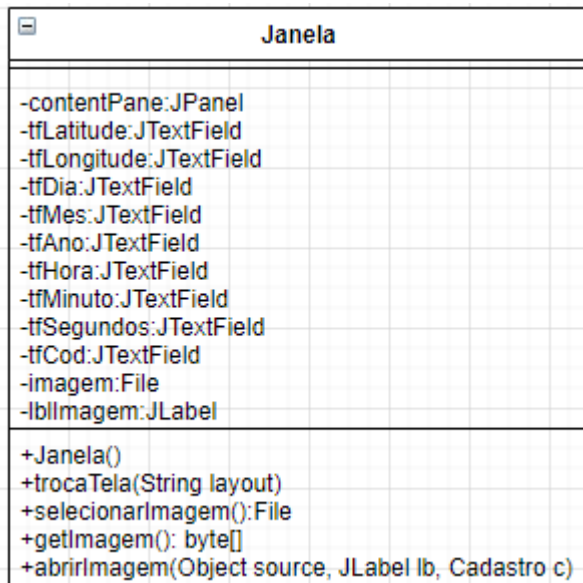
Exemplo de Funcionamento do Quick Sort:



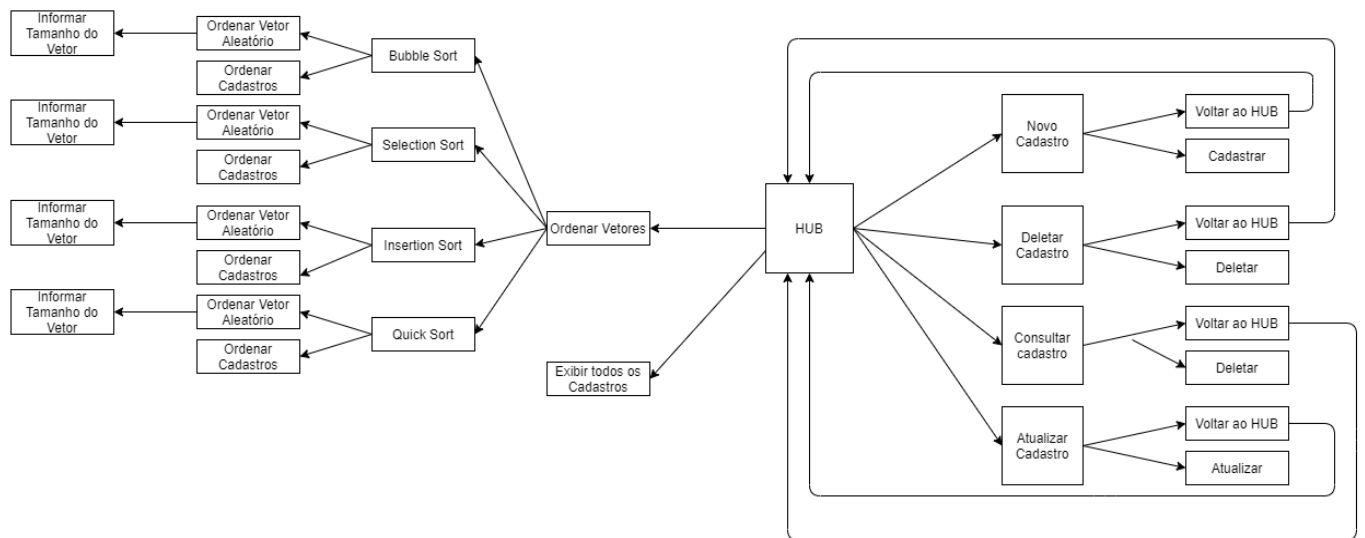
*Imagem retirada do site pioneiros da computação, artigo: p25 John von Neumann

Desenvolvimento

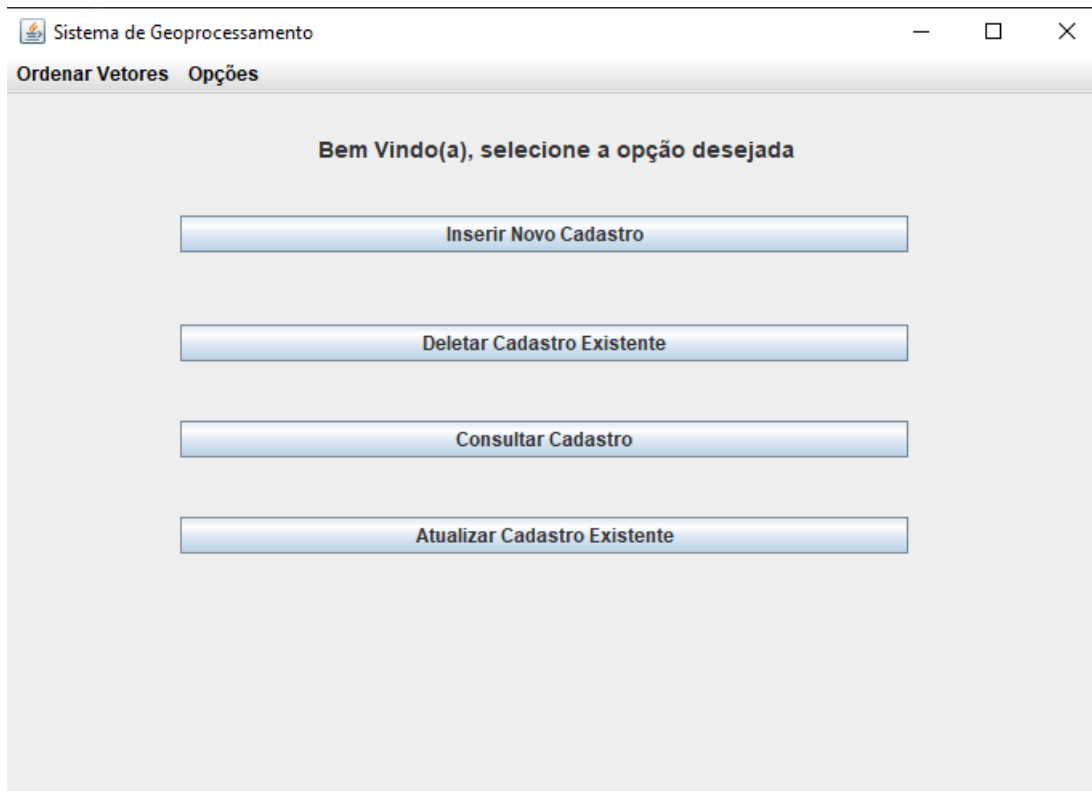
Classe Janela



Tratando-se da Interface do Usuário do sistema, a classe Janela é a responsável por exibir as telas referentes a inserção dos dados pelo usuário, assim como a apresentação destes dados para o usuário. Como a proposta do projeto é o trabalho com imagens, a classe apresenta métodos capazes de carregar uma imagem da máquina do usuário e adiciona-la ao cadastro, sempre apresentando uma miniatura na interface. A troca de telas foi feita através de um CardLayout, contando com as telas: Cadastramento, Deletar, Consultar e Atualizar, além de um menu com as opções de ordenar vetores (sejam eles os próprios cadastros, ou vetores gerados aleatoriamente pela aplicação), exibir todos os cadastros existente no sistema e sair da aplicação.



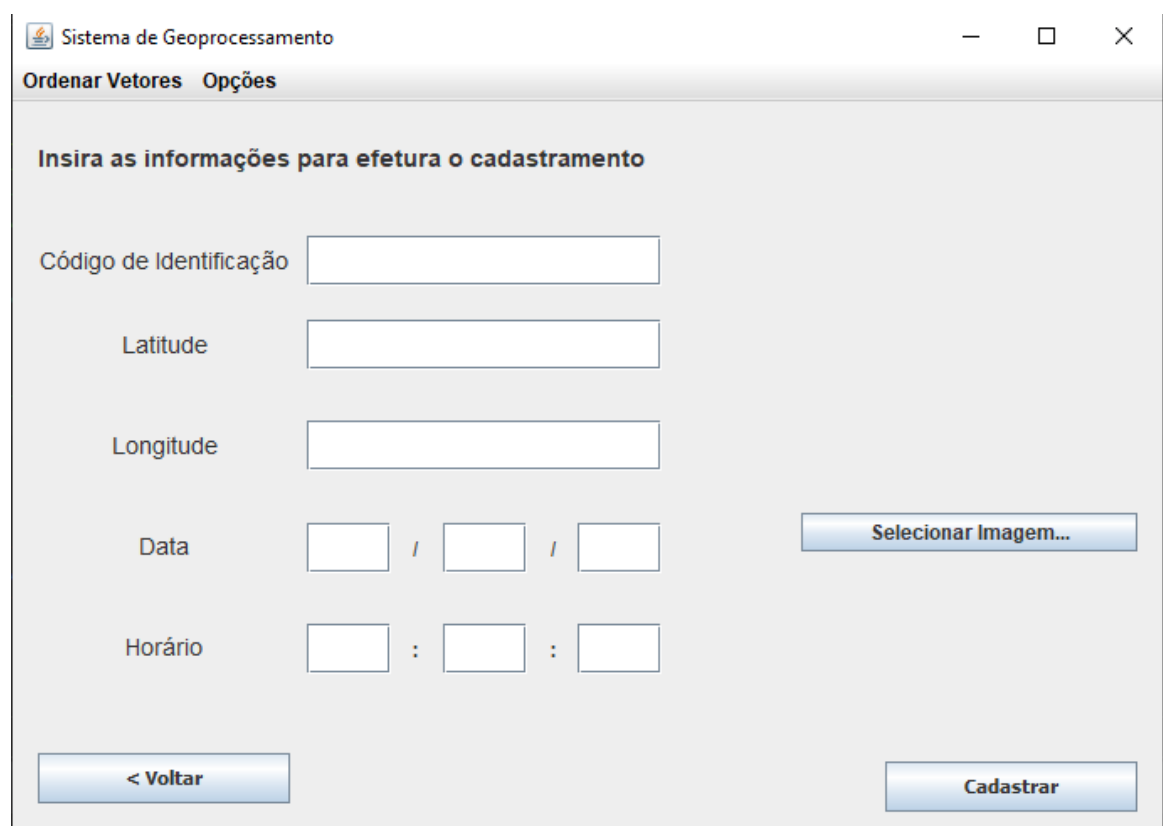
A imagem acima descreve o projeto do fluxo do sistema, a parte central de todo projeto é denominada HUB (em inglês, eixo central), a tela de HUB nos permite acessar todas as partes do sistema através de seus botões e todas as outras telas sempre levam a ela no final, de forma que ela serve como uma orientação para o usuário apresentando a ele todo o leque de opções disponíveis. Tela HUB em funcionamento:



Através da tela HUB, podemos ir para a tela Cadastramento, Deletar, Consultar, Deletar e Atualizar. Nos focando na tela Cadastramento por hora, nesta

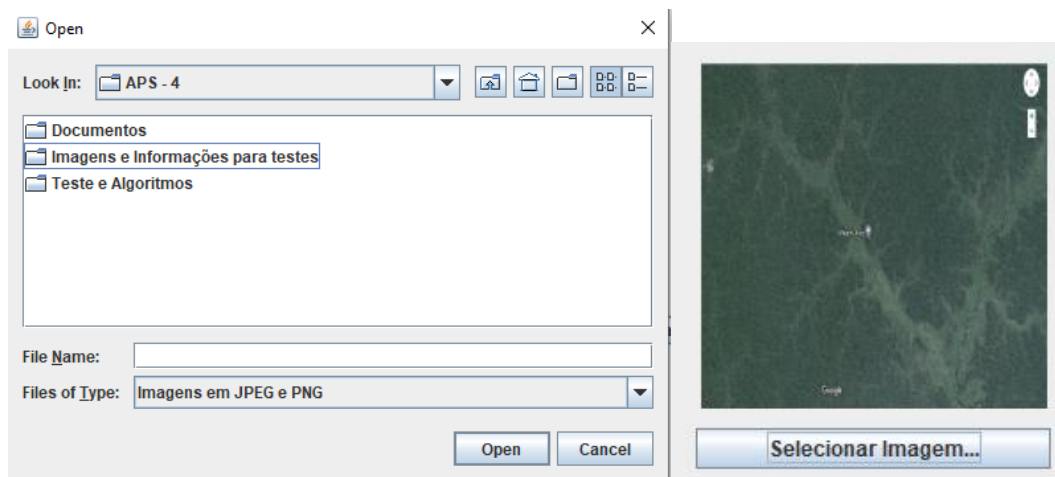
Tela solicitamos que o usuário insira as informações necessárias para um cadastro. Durante a projetagem da aplicação, tomando como base Sistemas de Geoprocessamento profissionais, foi estipulado que os dados necessários para um cadastro são: O código de identificação próprio do cadastro, que servirá para a identificação única do cadastro em questão agindo como uma Chave Primária, o arquivo de imagem a ser armazenado e catalogado, a Latitude e Longitude do local da imagem para serem utilizados como referência, a data e o horário em que a imagem foi coletada. Quando o usuário aperta o botão de selecionar imagem, um JFileChooser é aberto e ele seleciona uma imagem de seu próprio diretório, após selecionada a imagem aparece na tela como uma miniatura. Quando o botão de cadastrar é apertado, são feitas verificações para saber se o usuário selecionou uma imagem e inseriu um código de identificação que ainda não está cadastrado no sistema (visto que estes dois dados são vitais para o bom funcionamento do sistema e a coerência com a proposta do projeto), se o usuário inseriu os dados corretamente o cadastro será efetuado e uma mensagem de sucesso aparecerá.

Tela Cadastramento



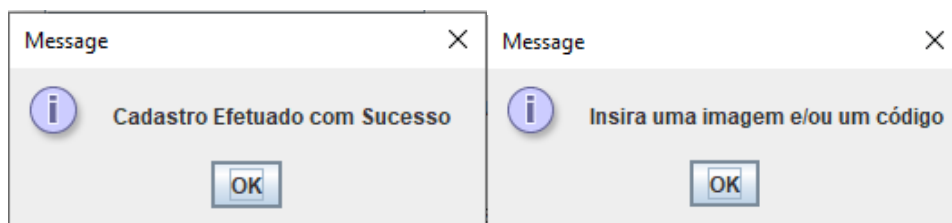
A imagem mostra a janela de cadastro do 'Sistema de Geoprocessamento'. No topo, há uma barra de menu com 'Ordenar Vetores' e 'Opções'. O título principal da tela é 'Insira as informações para efetura o cadastramento'. Abaixo, há campos de entrada para: 'Código de Identificação', 'Latitude', 'Longitude', 'Data' (formato DD/MM/AA) e 'Horário' (formato HH:MM:SS). À direita dos campos de data e horário, há um botão 'Selecionar Imagem...'. Na base da janela, há dois botões: '< Voltar' à esquerda e 'Cadastrar' à direita.

Seleção de Imagem e miniatura de imagem

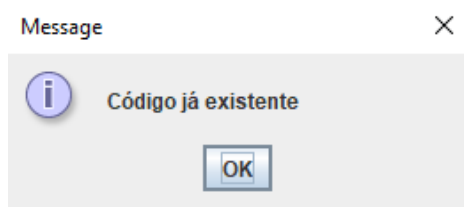


Feedback de sucesso ao cadastrar

Feedback de falta de seleção de imagem e/ou
código



Feedback de cadastro com código idêntico no sistema



Um pouco mais simples se comparada com a tela anterior, a tela Deletar conta com apenas um campo solicitando a inserção do Código de Identificação do cadastro a ser deletado, quando o botão deletar é apertado há uma verificação e identificação do código na lista de cadastros, se o código não constar na lista uma mensagem aparecerá informando o usuário, caso contrário aparecerá uma mensagem pedindo a confirmação do usuário para deletar o cadastro, com a confirmação o cadastro será deletado com sucesso.

Tela Deletar

Sistema de Geoprocessamento

Ordenar Vetores Opções

Insira o Código de Identificação do Cadastro para deleta-lo

Código de Identificação

< Voltar Deletar

Mensagem solicitando confirmação

Feedback de sucesso

Deletar Cadastro

Código encontrado, deseja mesmo deleta-lo?

Confirmar Cancelar

Message

Cadastro Deletado

OK

Feedback de código não encontrado

Message

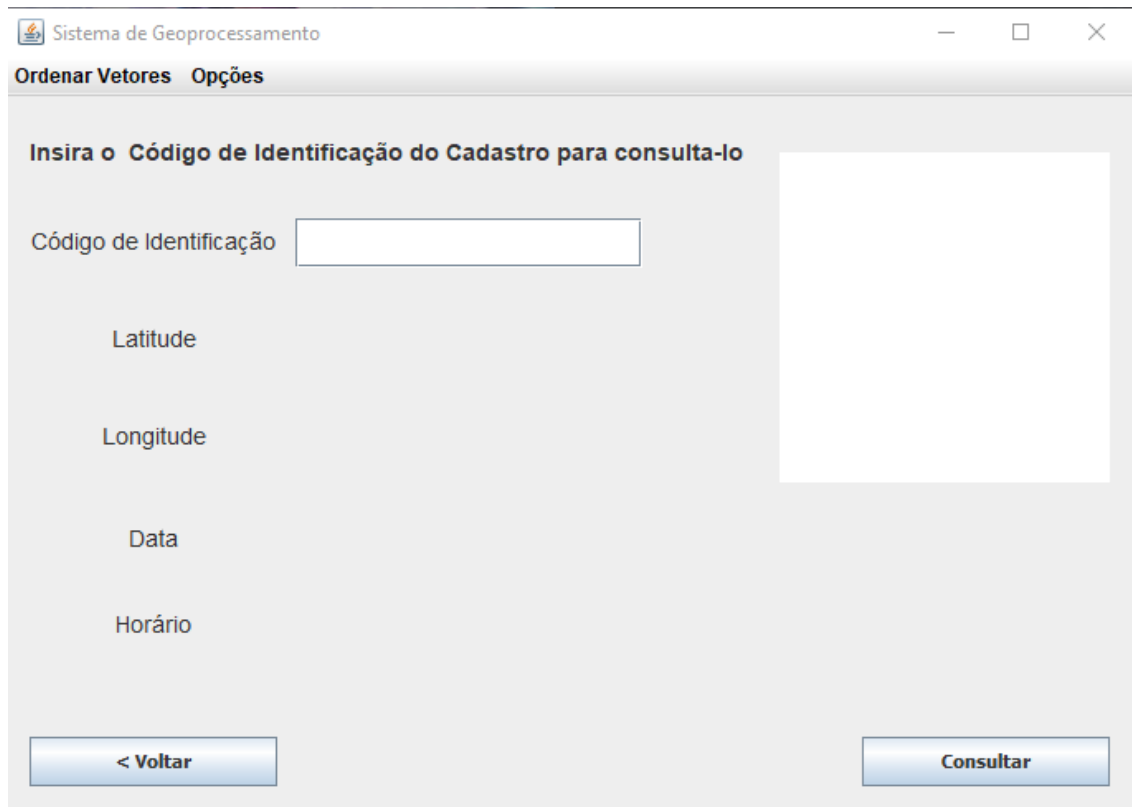
Código não encontrado

OK

A tela Consultar, semelhante a tela Cadastramento, contém apenas um campo de texto solicitando o Código de Identificação do cadastro que se deseja consultar, porém, diferentemente da tela anterior, ele conta com JLabels específicas para apresentar ao usuário os dados solicitados (Latitude, Longitude, Data, Horário

e Imagem). Quando o botão Consultar é apertado, é feita uma verificação e identificação do código na lista de cadastros, quando o código não é encontrado é exibida uma mensagem idêntica ao feedback de código não encontrado da tela Deletar, caso contrário a consulta é realizada com sucesso e as JLabels assumem os valores referentes ao cadastro.

Tela Consulta



Sistema de Geoprocessamento

Ordenar Vetores Opções

Insira o Código de Identificação do Cadastro para consulta-lo

Código de Identificação

Latitude

Longitude

Data

Horário

< Voltar Consultar

Tela Consulta após consulta


Sistema de Geoprocessamento

Ordenar Vetores Opções

Insira o Código de Identificação do Cadastro para consulta-lo

Código de Identificação

Latitude	123N
Longitude	456O
Data	12/05/2020
Horário	12:35:13



A última tela, Atualizar é praticamente idêntica à tela Cadastramento, porém quando o botão atualizar é apertado há uma verificação para saber se o código de identificação inserido pertence a algum dos cadastros existente, em caso negativo uma mensagem idêntica ao feedback de código não encontrado da tela Deletar e da tela Consultar é exibida, em caso positivo a atualização acontecerá e uma mensagem de sucesso na atualização aparecerá.

Cadastro 1 antes da atualização

Sistema de Geoprocessamento

Ordenar Vetores Opções

Insira o Código de Identificação do Cadastro para consulta-lo

Código de Identificação


Latitude 123N

Longitude 456O

Data 12/05/2020

Horário 12:35:13

< Voltar Consultar



Tela Atualizar

Sistema de Geoprocessamento

Ordenar Vetores Opções

Insira as informações para efetuar a atualização no cadastramento

Código de Identificação

Latitude

Longitude

Data / /

Horário : :

Selecionar Imagem...

< Voltar Atualizar

Cadastro 1 após atualização

Sistema de Geoprocessamento

Ordenar Vetores Opções

Insira o Código de Identificação do Cadastro para consulta-lo

Código de Identificação 1

Latitude 987S

Longitude 123E

Data 24/12/2019

Horário 09:11:23

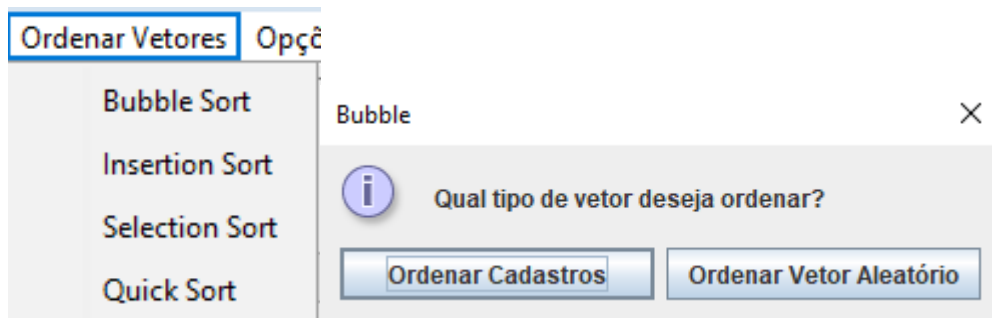
3

< Voltar Consultar

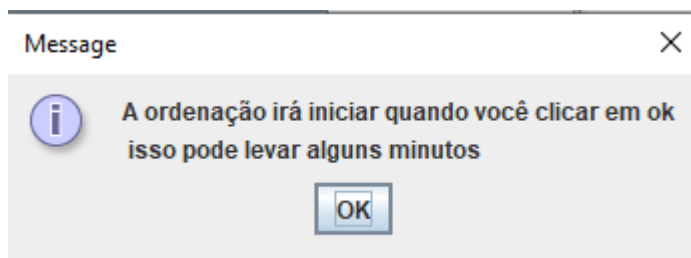
Além das telas, o sistema conta com um menu com as seguintes funções: Ordenar Vetores e Opções. Estas funções de certa forma isoladas do funcionamento principal da aplicação têm serventia na parte analítica do projeto, onde devemos analisar os algoritmos de ordenação. No primeiro item, Ordenar Vetores quando clicado, é aberto um menu em cascata para selecionarmos o método de ordenação desejado entre Bubble Sort, Insertion Sort, Selection Sort e Quick Sort. Após escolhermos, é nos dada a opção através do JOptionPane de escolher se queremos aplicar a ordenação nos cadastros da própria aplicação ou em vetor de tamanho variável com números aleatórios, em ambas as opções no final da operação é mostrada uma mensagem informando os elementos iniciais e finais do vetor, seu tamanho e o tempo decorrido para a ordenação em milissegundos. Se a opção de ordenar os próprios cadastros é selecionada, todos os cadastros existentes são ordenados em ordem numérica baseando-se nos seus códigos de cadastro, agora se a opção de ordenar um vetor de números aleatórios, o tamanho desejado para o vetor é solicitado ao usuário.

Menu em cascata

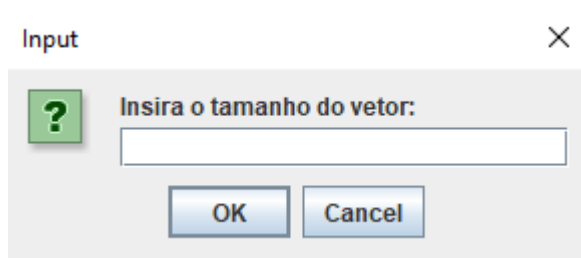
Escolha entre cadastro ou vetor de números aleatórios



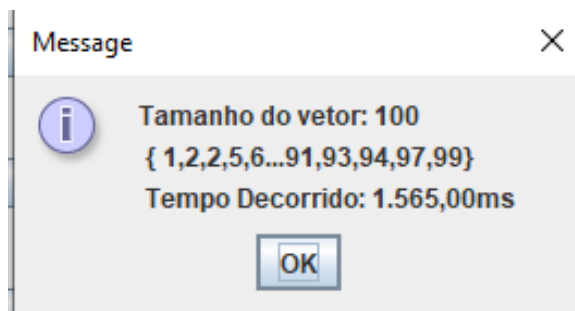
Mensagem de início da ordenação



Solicitação de tamanho do vetor

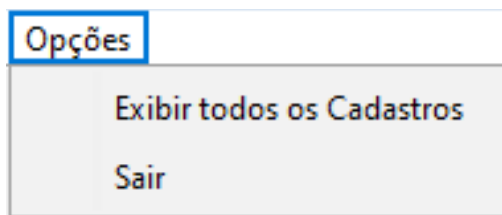


Mensagem com o tamanho do vetor, elementos e tempo decorrido

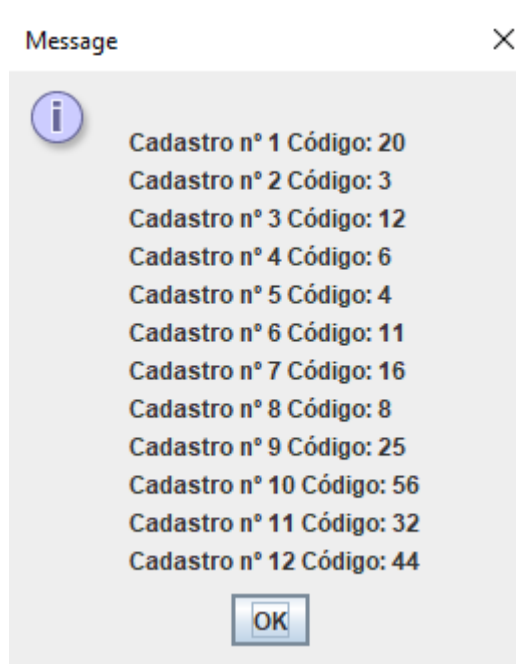


No item de menu Opções é aberto um menu em cascata onde o usuário pode exibir todos os cadastros existentes no sistema em ordem de inserção, esta função é útil para observarmos o resultado final das ordenações ocorridas nos cadastros através do item de menu Ordenar Vetores., ou pode sair da aplicação.

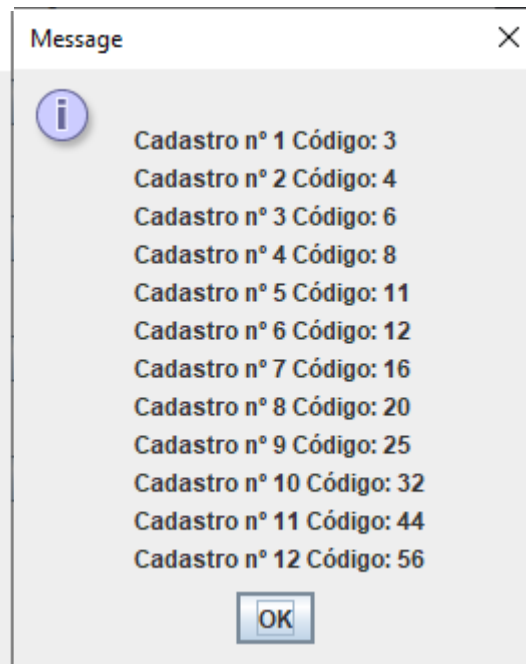
Menu em cascata



Cadastros em ordem de inserção



Cadastros ordenados



Fora a classe Janela temos: a classe Ordenacao, responsável por abrigar os algoritmos de ordenação, vitais para a obtenção de amostras e sua posterior análise; A classe Pesquisa, que abriga os métodos utilizados para verificar a existência de um código nos cadastros existentes e retornar o índice de um cadastro com um código específico; A classe Vetor, que abriga os métodos responsáveis por gerar os vetores de números aleatórios e preenche-los; A classe Cadastro, a classe mais vital do projeto depois de Janela, que abriga os atributos encapsulados com seus getters e setters, todos referentes aos cadastros realizados durante a aplicação; A classe Main, responsável por abrigar o método principal e instanciar a classe Janela.

Resultados e Discussão

Nesta sessão trataremos da análise de alguns dados obtidos através da aplicação, todos referentes ao desempenho dos algoritmos de ordenação em diferentes tamanhos de vetores e casos. Como estamos tratando de desempenho e tempo de ordenação, é importante exibir as configurações da máquina que realizou os testes:

Edição do Windows

Windows 10 Pro

© 2020 Microsoft Corporation. Todos os direitos reservados.

Sistema

Processador: Intel(R) Core(TM) i5-4300M CPU @ 2.60GHz 2.59 GHz

Memória instalada (RAM): 8,00 GB (utilizável: 7,88 GB)

Tipo de sistema: Sistema Operacional de 64 bits, processador com base em x64

Tabela Dados Aleatórios

Os dados da tabela a seguir foram obtidos com a ordenação de vetores com números aleatórios, ou seja, não apresentando nenhuma ordem específica que torne mais fácil ou difícil a ordenação.

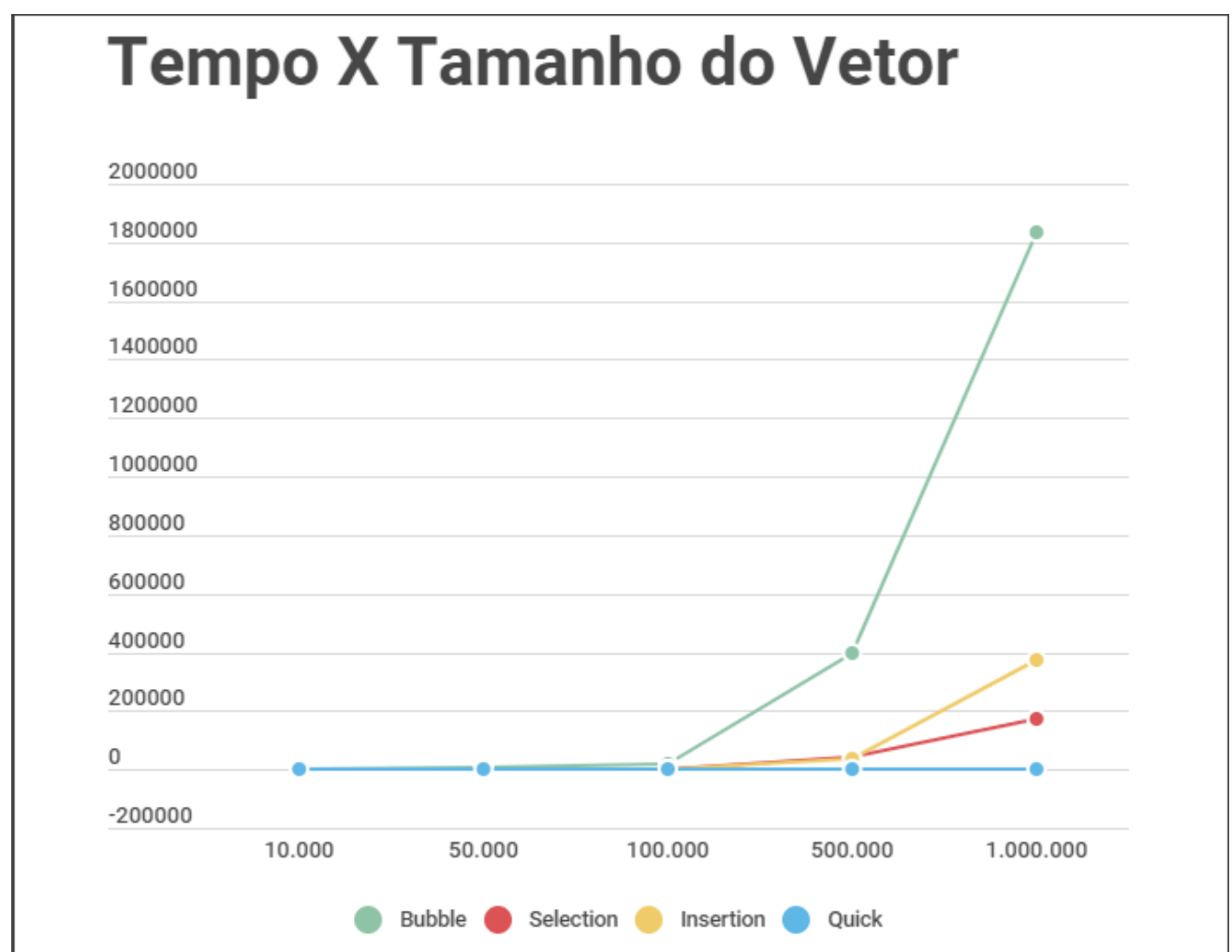
		Algoritmos de Ordenação				
		Bubble	Selection	Insertion	Quick	
Tamanho do Vetor	10.000	189	36	53	5	Tempo de Ordenação em Milissegundos
	50.000	4.369	379	807	7	
	100.000	18.078	1343	1.145	13	
	500.000	394.611	39.538	35.137	65	
	1.000.000	1.836.278	169.727	371.498	145	

Ao compararmos o tempo de ordenação de todos os algoritmos pelo tamanho de vetor, podemos observar um padrão entre eles, com exceção dos vetores de 100.00 e 500.000 elementos onde o Insertion Sort realizou as ordenações mais rapidamente que o Selection Sort, o Bubble Sort sempre é o que demora mais

tempo para realizar sua ordenação, seguido do Insertion Sort, enquanto o Quick Sort sempre é o mais rápido a realizar sua ordenação.

A “recuperação” no vetor de 1.000.000 de elementos do Selection Sort em relação ao Insertion Sort pode dizer respeito a sua característica de afinidade com grandes vetores, vantagem que não está presente no Insertion Sort. Da mesma forma a vantagem do Insertion Sort nos vetores de 100.000 e 500.000 elementos pode dizer respeito a ordem inicial dos elementos, visto que ele pode tirar vantagem desta característica do vetor enquanto o Selection Sort não pode.

Gráfico



O gráfico, apresentando um padrão crescente por se tratar do aumento de elementos do vetor e, conseqüentemente, tempo levado para ordená-los, reforça o padrão abordado na análise da tabela, mostrando o Bubble Sort como o algoritmo com os tempos mais elevados e o Quick Sort com os tempos mais baixos.

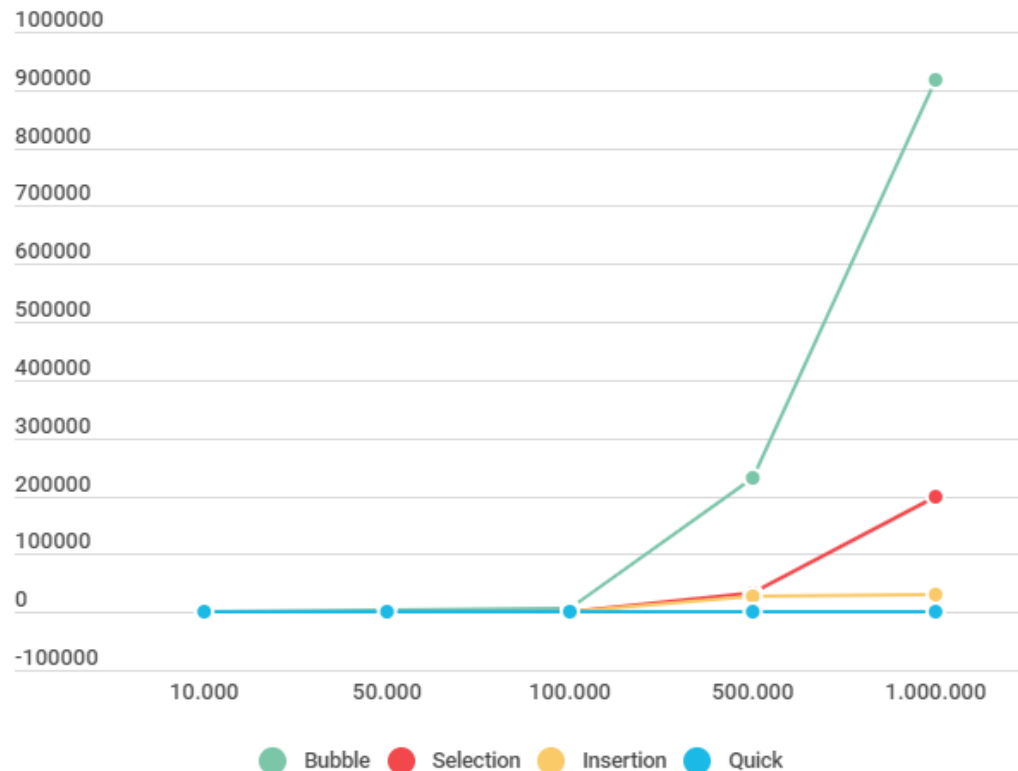
Tabela Dados Semiordenados

Os dados da tabela a seguir foram obtidos com a ordenação de vetores com números ordenados até a metade do vetor, fazendo com que certos algoritmos possam ordenar com mais facilidade que outros.

		Algoritmos de Ordenação				
		Bubble	Selection	Insertion	Quick	
Tamanho do Vetor	10.000	100	37	24	23	Tempo de Ordenação em Milissegundos
	50.000	2.124	324	225	33	
	100.000	7.981	1.186	849	29	
	500.000	230.399	33.410	28.147	113	
	1.000.000	917.592	197.673	29.791	115	

Ao observarmos a tabela, podemos perceber a quebra no padrão que ocorria com os vetores de números aleatórios, apesar de o Bubble e o Quick Sort ainda se destacarem, respectivamente, como o mais lento e mais rápido, diferentemente da tabela anterior o Insertion Sort superou o Selection Sort, com um tempo relativamente mais curto. Tal fato deve-se a característica de comportamento natural do Insertion Sort que permite que ele se beneficie de vetores com partes já ordenadas para poupar tempo no processo. Fora o Insertion Sort, que obteve um ganho considerável de eficiência, o algoritmo mais afetado foi o Bubble Sort que teve grandes diferenças de uma coleta de dados para outra, diminuindo quase pela metade o tempo gasto para ordenação em quase todos os tamanhos de vetores, o que se deve diretamente a semiordenação inicial. O Selection Sort e o Quick Sort não sofreram grandes mudanças, porque, primeiramente o Selection Sort não é afetado pela ordenação inicial, mantendo o mesmo número de comparações, e porque o Quick Sort muda por si só a ordenação dos valores de forma que eles fiquem à direita ou a esquerda de seu pivô.

Tempo X Tamanho do Vetor



O gráfico evidencia a afirmação feita acima com base na tabela, o Insertion Sort sofreu uma redução em seu tempo de ordenação em todos os tamanhos de vetores, enquanto o Selection Sort permaneceu com valores semelhantes, fazendo com que o Insertion Sort se tornasse o segundo algoritmo mais rápido e eficiente neste caso, perdendo apenas para o Quick Sort.

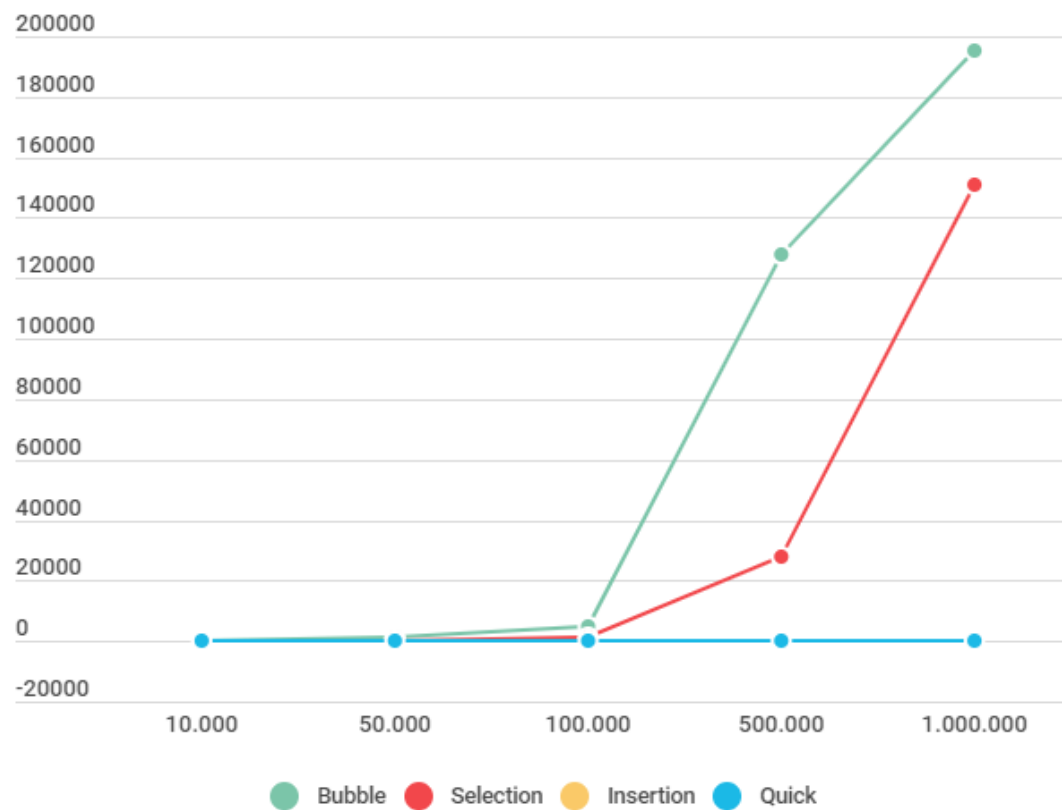
Tabela Dados Ordenados

		Algoritmos de Ordenação				Tempo de Ordenação em Milissegundos
		Bubble	Selection	Insertion	Quick	
Tamanho do Vetor	10.000	62	33	1	35	
	50.000	1.337	294	7	42	
	100.000	5.047	1.398	5	17	
	500.000	128.084	28.070	1	61	
	1.000.000	195.232	150.701	1	119	

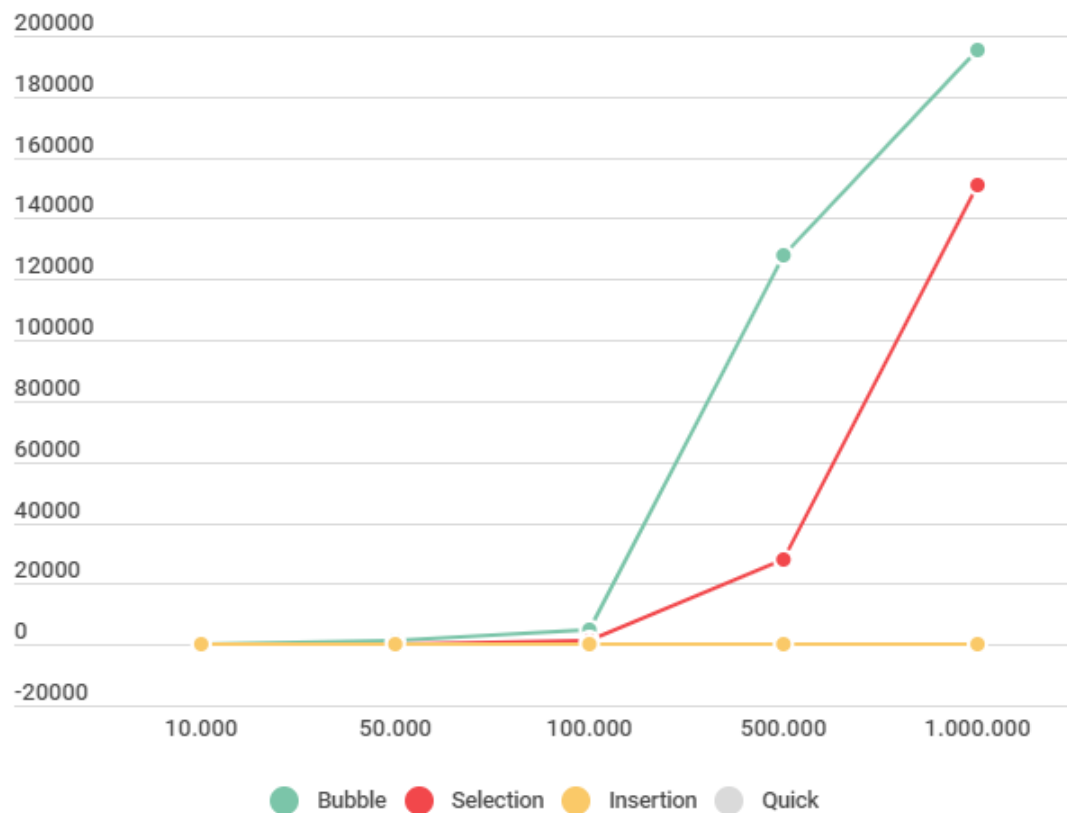
Nesta tabela, podemos observar que todos os algoritmos tiveram uma redução em seu tempo, com exceção do Quick Sort, além disso o mesmo perdeu sua soberania como o mais rápido de todos os algoritmos para o Insertion Sort, pois apesar de manter um tempo semelhante ao das outras tabelas, o Insertion Sort reduziu drasticamente seu tempo de ordenação, beirando o instantâneo. Esta redução se deve novamente ao comportamento natural do Insertion Sort. Apesar dos vetores estarem previamente ordenados, o Bubble Sort e o Selection Sort não alcançaram tempos melhores pois os dois algoritmos contam com um número grande comparações obrigatórias, fazendo com que eles ainda precisem percorrer o vetor mesmo assim.

Gráficos

Tempo X Tamanho do Vetor



Tempo X Tamanho do Vetor

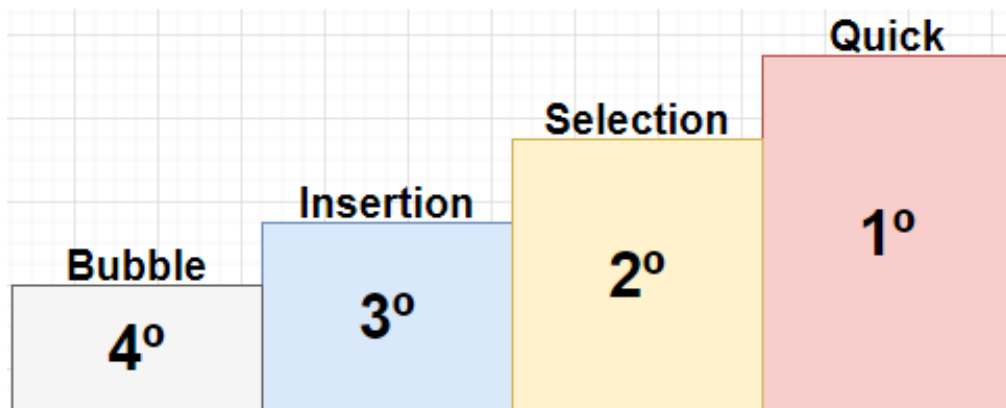


Podemos observar nos gráficos uma drástica mudança em relação ao Insertion Sort, alcançando tempos tão baixos ou menores quanto o Quick Sort, enquanto podemos destacar também uma aproximação do Bubble Sort para o Selection Sort.

Considerações Finais

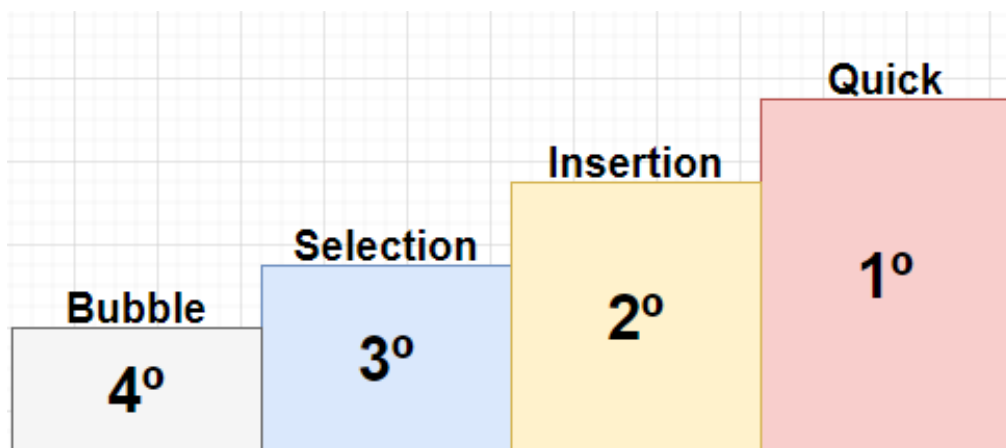
Conforme a exibição dos resultados e sua discussão, foram elaborados os seguintes *rankings*:

Dados Aleatórios



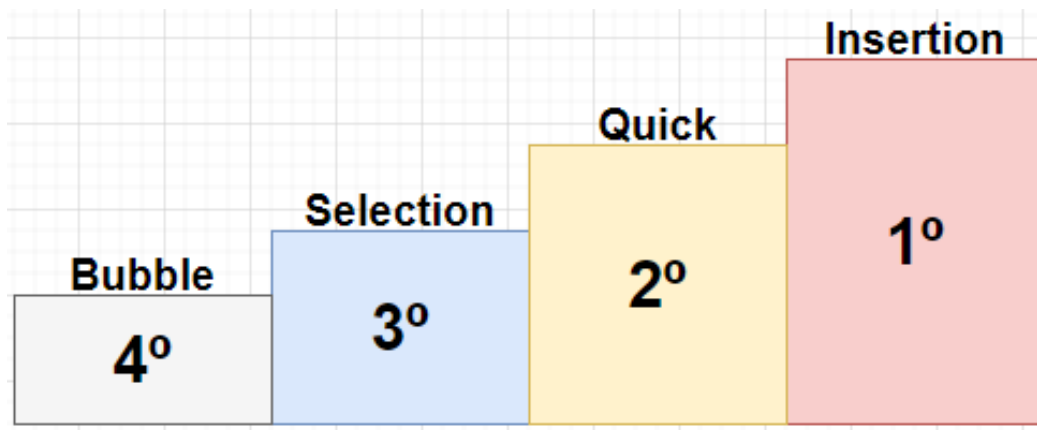
Se tratando de vetores com valores aleatórios, a melhor eficiência apresentada foi a do Quick Sort, enquanto a pior foi a do Bubble Sort. Conforme o *ranking* acima, melhor escolha para este caso: Quick Sort.

Dados Semiordenados



Se tratando de vetores com os dados semiordenados, a melhor eficiência apresentada foi, novamente a do Quick Sort, enquanto a pior novamente também foi a do Bubble Sort. Com vetores semiordenados, o Insertion Sort superou o Selection Sort. Conforme o *ranking* acima, melhor escolha para este caso: Quick Sort.

Dados Ordenados



Se tratando de vetores com os dados semiordenados, a melhor eficiência apresentada foi a do Insertion Sort, enquanto a pior foi, pela terceira vez, a do Bubble Sort. Aqui temos uma virada repentina, com o Insertion Sort superando o Quick Sort(apenas neste caso em específico). Conforme o *ranking* acima, melhor escolha para este caso: Insertion Sort.

Observando os rankings podemos concluir que na maioria dos casos o Quick Sort é a melhor escolha por conta de sua eficiência, visto que a única ocasião em que foi superado, ele, ainda assim, exibiu tempos bem baixos, portanto no desempenho em geral ele é considerado o melhor e mais eficiente algoritmo de ordenação entre os quatro abordados, enquanto o Bubble Sort é o pior e menos eficiente de todos, exibindo os tempos mais elevados em todas os casos analisados.

Referências Bibliográficas

DEVMEDIA. Bruno, 2013. Artigo: Algoritmos de ordenação: análise e comparação.

Disponível em: <<https://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261#:~:text=Os%20mais%20populares%20algoritmos%20de,Heap%20sort%20e%20Shell%20sort>>. Acesso 15 de Novembro de 2020

IME-USP. Paulo Feofiloff, 2018. Artigo: Heapsort.

Disponível em: <<https://www.ime.usp.br/~pf/algoritmos/aulas/hpsrt.html>>. Acesso em 15 de Novembro de 2020

UFG. Hebert Coelho, Nádia Felix. Métodos de Ordenação: Selection, Insertion, Bubble, Merge (Sort)

Disponível em: <http://ww2.inf.ufg.br/~hebert/disc/aed1/AED1_04_ordenacao1.pdf>.

Acesso em 15 de Novembro de 2020

UEMG. Andrew Carlos de Sene Dias, Nayara Almeida Vilela, Walteno Martins Pereira Júnior, 2014. ANÁLISE SOBRE ALGUNS MÉTODOS DE ORDENAÇÃO DE LISTAS:

SELEÇÃO, INSERÇÃO E SHELLSORT

Disponível em: <http://www.waltenomartins.com.br/intercursos_v13n1b.pdf>. Acesso em 15 de Novembro de 2020

Davi Guimarães©. Davi Guimarães, 2014. Artigo: Algoritmos de ordenação: análise e comparação

Disponível em: <http://davitpd.blogspot.com/2014/07/algoritmos-de-ordenacao-analise-e_13.html>. Acesso em 15 de Novembro de 2020

Henrique Braga. Henrique Braga,2018.Algoritmos de Ordenação: Insertion Sort.

Disponível em:<https://medium.com/@henriquebraga_18075/algoritmos-de-ordena%C3%A7%C3%A3o-iii-insertion-sort-bfade66c6bf1>. Acesso em 15 de Novembro de 2020.

GeeksforGeeks. GeeksforGeeks,2020. Artigo: Insertion Sort.

Disponível em: <<https://www.geeksforgeeks.org/insertion-sort/>>. Acesso em 15 de Novembro de 2020.

Henrique Braga. Henrique Braga,2020. Artigo:Algoritmos de Ordenação: Selection Sort.

Disponível em <https://medium.com/@henriquebraga_18075/algoritmos-de-ordena%C3%A7%C3%A3o-ii-selection-sort-8ee4234deb10>. Acesso em: 15 de Novembro de 2020.

Mackenzie Faculdade de Computação e Informática. Daniel Arndt Alves,2013. Estrutura de Dados II Aula 04:Selection Sort.

Disponível em: <<https://www.slideshare.net/DanielArndtAlves/selection-sort-25735943>>.Acesso em: 15 de Novembro de 2020.

Código Fonte

Classe Janela

```
package Frame;
```

```
import java.awt.AlphaComposite;
```

```
import java.awt.CardLayout;
```

```
import Algoritmos.Ordenacao;
```

```
import Algoritmos.Pesquisa;
```

```
import Algoritmos.Vetor;
```

```
import java.awt.Color;
```

```
import java.awt.Font;
```

```
import java.awt.Graphics2D;
```

```
import java.awt.event.ActionEvent;
```

```
import java.awt.event.ActionListener;
```

```
import java.awt.image.BufferedImage;
```

```
import java.io.ByteArrayOutputStream;
```

```
import java.io.File;
```

```
import java.io.IOException;
```

```
import java.text.DecimalFormat;
```

```
import javax.imageio.ImageIO;
```

```
import javax.swing.ImageIcon;

import javax.swing.JButton;

import javax.swing.JFileChooser;

import javax.swing.JFrame;

import javax.swing.JLabel;

import javax.swing.JMenu;

import javax.swing.JMenuBar;

import javax.swing.JMenuItem;

import javax.swing.JOptionPane;

import javax.swing.JPanel;

import javax.swing.JTextField;

import javax.swing.SwingConstants;

import javax.swing.border.EmptyBorder;

import javax.swing.filechooser.FileNameExtensionFilter;


import Entidades.Cadastro;

import Main.Main;

import javax.swing.border.BevelBorder;
```

```
public class Janela extends JFrame {

    public Ordenacao ord = new Ordenacao();

    public Pesquisa p = new Pesquisa();

    private JPanel contentPane;

    private JTextField tfLatitude;
```

```
private JTextField tfLongitude;

private JTextField tfDia;

private JTextField tfMes;

private JTextField tfAno;

private JTextField tfHora;

private JTextField tfMinuto;

private JTextField tfSegundos;

private JTextField tfCod;

private JTextField tfCodDel;

private JTextField tfCodCon;

private JTextField tfCodAtt;

private JTextField tfLatAtt;

private JTextField tfLongAtt;

private JTextField tfDiaAtt;

private JTextField tfMesAtt;

private JTextField tfAnoAtt;

private JTextField tfHoraAtt;

private JTextField tfMinutoAtt;

private JTextField tfSegundoAtt;

private File imagem;

private JLabel lblImagem;

DecimalFormat df = new DecimalFormat("#,###.00");
```

```

public Janela() {

    setTitle("Sistema de Geoprocessamento");

    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    setBounds(100, 100, 700, 500);


    JMenuBar menuBar = new JMenuBar();

    setJMenuBar(menuBar);


    JMenu mnOrdenacao = new JMenu("Ordenar Vetores");

    mnOrdenacao.setForeground(Color.BLACK);

    menuBar.add(mnOrdenacao);


    JMenuItem mntmBubble = new JMenuItem("Bubble Sort");

    mntmBubble.addActionListener(new ActionListener() {

        public void actionPerformed(ActionEvent e) {

            String[] menu = {"Ordenar Cadastros", "Ordenar Vetor Aleatório"};

            int opt = JOptionPane.showOptionDialog(null, "Qual tipo de vetor deseja
ordenar?", "Bubble", JOptionPane.DEFAULT_OPTION,

                JOptionPane.INFORMATION_MESSAGE, null, menu, menu[0]);

            if(opt == 0) {

                String msg = "";

                Vetor vet = new Vetor(Main.CD);

                int v[] = vet.getVetor();

                JOptionPane.showMessageDialog(null, "A ordenação irá iniciar quando
você clicar em ok \n isso pode levar alguns minutos");

```

```

double ini = System.currentTimeMillis();

ord.bubble(v);

double tempo = System.currentTimeMillis() - ini;

Cadastro.ordenaLista(Main.CD, v);

msg += "Tamanho do vetor: " + v.length + "\n Tempo Decorrido: " +
df.format(tempo);

JOptionPane.showMessageDialog(null, msg);

} else if(opt == 1) {

String msg = "";

int tam = Integer.parseInt(JOptionPane.showInputDialog("Insira o
tamanho do vetor:"));

int v[] = null;

Vetor vet = new Vetor();

v = vet.preencherVetorOrd(v, tam);

JOptionPane.showMessageDialog(null, "A ordenação irá iniciar quando
você clicar em ok \n isso pode levar alguns minutos");

double ini = System.currentTimeMillis();

ord.bubble(v);

double tempo = System.currentTimeMillis() - ini;

if(v.length > 6) {

msg += "Tamanho do vetor: " + v.length

+ "\n { " + v[0] + "," + v[1] + "," + v[2] + "," + v[3] + "," + v[4] + "..."+
v[v.length-5] + "," + v[v.length-4] + "," + v[v.length-3] + "," + v[v.length-2] + "," +
v[v.length-1] + "}"

+ "\n Tempo Decorrido: " + df.format(tempo) + "ms";

JOptionPane.showMessageDialog(null, msg);

```

```

    } else if(v.length < 3) {

        msg += "Tamanho de Vetor Insuficiente";

        JOptionPane.showMessageDialog(null, msg);

    } else {

        msg += "Tamanho do vetor: " + v.length

            + "\n { " + v[0] + "," + v[1] + "," + v[2] + "}"

            + "\n Tempo Decorrido: " + df.format(tempo) + "ms";

        JOptionPane.showMessageDialog(null, msg);

    }

}

}

});

mnOrdenacao.add(mntmBubble);


JMenuItem mntmInsertion = new JMenuItem("Insertion Sort");

mntmInsertion.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        String[] menu = {"Ordenar Cadastros", "Ordenar Vetor Aleatório"};

        int opt = JOptionPane.showOptionDialog(null, "Qual tipo de vetor deseja
ordenar?", "Insertion", JOptionPane.DEFAULT_OPTION,

            JOptionPane.INFORMATION_MESSAGE, null, menu, menu[0]);

        if(opt == 0) {

            String msg = "";

            Vetor vet = new Vetor(Main.CD);

```

```

int v[] = vet.getVetor();

JOptionPane.showMessageDialog(null, "A ordenação irá iniciar quando
você clicar em ok \n isso pode levar alguns minutos");

double ini = System.currentTimeMillis();

ord.insertion(v);

double tempo = System.currentTimeMillis() - ini;

Cadastro.ordenaLista(Main.CD, v);

msg += "Tamanho do vetor: " + v.length + "\n Tempo Decorrido: " +
df.format(tempo);

JOptionPane.showMessageDialog(null, msg);

} else if(opt == 1) {

String msg = "";

int tam = Integer.parseInt(JOptionPane.showInputDialog("Insira o
tamanho do vetor:"));

int v[] = null;

Vetor vet = new Vetor();

v = vet.preencherVetorOrd(v, tam);

JOptionPane.showMessageDialog(null, "A ordenação irá iniciar quando
você clicar em ok \n isso pode levar alguns minutos");

double ini = System.currentTimeMillis();

ord.insertion(v);

double tempo = System.currentTimeMillis() - ini;

if(v.length > 6) {

msg += "Tamanho do vetor: " + v.length

```

```
        + "\n { " + v[0] + "," + v[1] + "," + v[2] + "," + v[3] + "," + v[4] + "..."+
v[v.length-5] + "," + v[v.length-4] + "," + v[v.length-3] + "," + v[v.length-2] + "," +
v[v.length-1] + "}"
```

```
        + "\n Tempo Decorrido: " + df.format(tempo) + "ms";
```

```
        JOptionPane.showMessageDialog(null, msg);
```

```
    } else if(v.length < 3) {
```

```
        msg += "Tamanho de Vetor Insuficiente";
```

```
        JOptionPane.showMessageDialog(null, msg);
```

```
    } else {
```

```
        msg += "Tamanho do vetor: " + v.length
```

```
        + "\n { " + v[0] + "," + v[1] + "," + v[2] + "}"
```

```
        + "\n Tempo Decorrido: " + df.format(tempo) + "ms";
```

```
        JOptionPane.showMessageDialog(null, msg);
```

```
    }
```

```
}
```

```
}
```

```
});
```

```
mnOrdenacao.add(mntmInsertion);
```

```
JMenuItem mntmSelection = new JMenuItem("Selection Sort");
```

```
mntmSelection.addActionListener(new ActionListener() {
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        String[] menu = {"Ordenar Cadastros", "Ordenar Vetor Aleatório"};
```



```

int opt = JOptionPane.showOptionDialog(null, "Qual tipo de vetor deseja
ordenar?", "Selection", JOptionPane.DEFAULT_OPTION,

    JOptionPane.INFORMATION_MESSAGE, null, menu, menu[0]);

if(opt == 0) {

    String msg = "";

    Vetor vet = new Vetor(Main.CD);

    int v[] = vet.getVetor();

    JOptionPane.showMessageDialog(null, "A ordenação irá iniciar quando
você clicar em ok \n isso pode levar alguns minutos");

    double ini = System.currentTimeMillis();

    ord.selection(v);

    double tempo = System.currentTimeMillis() - ini;

    Cadastro.ordenaLista(Main.CD, v);

    msg += "Tamanho do vetor: " + v.length + "\n Tempo Decorrido: " +
df.format(tempo);

    JOptionPane.showMessageDialog(null, msg);

} else if(opt == 1) {

    String msg = "";

    int tam = Integer.parseInt(JOptionPane.showInputDialog("Insira o
tamanho do vetor:"));

    int v[] = null;

    Vetor vet = new Vetor();

    v = vet.preencherVetorOrd(v, tam);

    JOptionPane.showMessageDialog(null, "A ordenação irá iniciar quando
você clicar em ok \n isso pode levar alguns minutos");

    double ini = System.currentTimeMillis();

```

```

ord.selection(v);

double tempo = System.currentTimeMillis() - ini;

if(v.length > 6) {

    msg += "Tamanho do vetor: " + v.length

    + "\n { " + v[0] + "," + v[1] + "," + v[2] + "," + v[3] + "," + v[4] + "..."+
v[v.length-5] + "," + v[v.length-4] + "," + v[v.length-3] + "," + v[v.length-2] + "," +
v[v.length-1] + "}"

    + "\n Tempo Decorrido: " + df.format(tempo) + "ms";

    JOptionPane.showMessageDialog(null, msg);

} else if(v.length < 3) {

    msg += "Tamanho de Vetor Insuficiente";

    JOptionPane.showMessageDialog(null, msg);

} else {

    msg += "Tamanho do vetor: " + v.length

    + "\n { " + v[0] + "," + v[1] + "," + v[2] + "}"

    + "\n Tempo Decorrido: " + df.format(tempo) + "ms";

    JOptionPane.showMessageDialog(null, msg);

}

}

}

});

mnOrdenacao.add(mntmSelection);

JMenuItem mntmQuick = new JMenuItem("Quick Sort");

```

```

mntmQuick.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        String[] menu = {"Ordenar Cadastros","Ordenar Vetor Aleatório"};

        int opt = JOptionPane.showOptionDialog(null, "Qual tipo de vetor deseja
ordenar?","Quick",JOptionPane.DEFAULT_OPTION,

            JOptionPane.INFORMATION_MESSAGE, null,menu,menu[0]);

        if(opt == 0) {

            String msg = "";

            Vetor vet = new Vetor(Main.CD);

            int v[] = vet.getVetor();

            JOptionPane.showMessageDialog(null, "A ordenação irá iniciar quando
você clicar em ok \n isso pode levar alguns minutos");

            double ini = System.currentTimeMillis();

            ord.quick(v);

            double tempo = System.currentTimeMillis() - ini;

            Cadastro.ordenaLista(Main.CD, v);

            msg += "Tamanho do vetor: " + v.length + "\n Tempo Decorrido: " +
df.format(tempo);

            JOptionPane.showMessageDialog(null, msg);

        } else if(opt == 1) {

            String msg = "";

            int tam = Integer.parseInt(JOptionPane.showInputDialog("Insira o
tamanho do vetor:"));

            int v[] = null;

            Vetor vet = new Vetor();

```

```
v = vet.preencherVetor(v, tam);
```

```
JOptionPane.showMessageDialog(null, "A ordenação irá iniciar quando  
você clicar em ok \n isso pode levar alguns minutos");
```

```
double ini = System.currentTimeMillis();
```

```
ord.quick(v);
```

```
double fim = System.currentTimeMillis();
```

```
double tempo = fim - ini;
```

```
if(v.length > 6) {
```

```
    msg += "Tamanho do vetor: " + v.length
```

```
    + "\n { " + v[0] + "," + v[1] + "," + v[2] + "," + v[3] + "," + v[4] + "..."+  
v[v.length-5] + "," + v[v.length-4] + "," + v[v.length-3] + "," + v[v.length-2] + "," +  
v[v.length-1] + "}"
```

```
    + "\n Tempo Decorrido: " + tempo + "ms";
```

```
JOptionPane.showMessageDialog(null, msg);
```

```
} else if(v.length < 3) {
```

```
    msg += "Tamanho de Vetor Insuficiente";
```

```
JOptionPane.showMessageDialog(null, msg);
```

```
} else {
```

```
    msg += "Tamanho do vetor: " + v.length
```

```
    + "\n { " + v[0] + "," + v[1] + "," + v[2] + "}"
```

```
    + "\n Tempo Decorrido: " + tempo + "ms";
```

```
JOptionPane.showMessageDialog(null, msg);
```

```
}
```

```

        }

    }

});

mnOrdenacao.add(mntmQuick);


JMenu mnOpcoes = new JMenu("Op\u00E7\u00F5es");
mnOpcoes.setForeground(Color.BLACK);
mnOpcoes.setBackground(Color.WHITE);
menuBar.add(mnOpcoes);


JMenuItem mntmExibirCadastros = new JMenuItem("Exibir todos os
Cadastros");

mntmExibirCadastros.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        Cadastro.exibirCodigosCadastro(Main.CD);

    }

});

mnOpcoes.add(mntmExibirCadastros);


JMenuItem mntmSair = new JMenuItem("Sair");
mntmSair.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        System.exit(0);

    }

});

```

```
mnOpcoes.add(mntmSair);

contentPane = new JPanel();

contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));

setContentPane(contentPane);

contentPane.setLayout(new CardLayout(0, 0));
```

```
JPanel Hub = new JPanel();

contentPane.add(Hub, "HUB");

Hub.setLayout(null);
```

```
JPanel Cadastramento = new JPanel();

contentPane.add(Cadastramento, "CADASTRAMENTO");

Cadastramento.setLayout(null);
```

```
JButton btnNovoCadastro = new JButton("Inserir Novo Cadastro");

btnNovoCadastro.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        trocaTela("CADASTRAMENTO");

    }

});
```

```
btnNovoCadastro.setBounds(103, 71, 454, 23);
```

```
Hub.add(btnNovoCadastro);
```

```
JButton btnDeletarCadastro = new JButton("Deletar Cadastro Existente");
```

```
btnDeletarCadastro.addActionListener(new ActionListener() {
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        trocaTela("DELETAR");
```

```
    }
```

```
});
```

```
btnDeletarCadastro.setBounds(103, 139, 454, 23);
```

```
Hub.add(btnDeletarCadastro);
```

```
JButton btnConsultarCadastro = new JButton("Consultar Cadastro");
```

```
btnConsultarCadastro.addActionListener(new ActionListener() {
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        trocaTela("CONSULTAR");
```

```
    }
```

```
});
```

```
btnConsultarCadastro.setBounds(103, 199, 454, 23);
```

```
Hub.add(btnConsultarCadastro);
```

```
JLabel lblHub = new JLabel("Bem Vindo(a), selecione a op\u00E7\u00E3o  
desejada");
```

```
lblHub.setFont(new Font("Arial", Font.BOLD, 14));
```

```
lblHub.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblHub.setBounds(10, 0, 654, 60);
```

```
Hub.add(lblHub);
```

```
JButton btnAtualizarCadastro = new JButton("Atualizar Cadastro Existente");
```

```
btnAtualizarCadastro.addActionListener(new ActionListener() {
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        trocaTela("ATUALIZAR");
```

```
    }
```

```
});
```

```
btnAtualizarCadastro.setBounds(103, 259, 454, 23);
```

```
Hub.add(btnAtualizarCadastro);
```

```
JLabel lblCadastramento = new JLabel("Insira as informa\u00E7\u00F5es para  
efetura o cadastramento");
```

```
lblCadastramento.setHorizontalAlignment(SwingConstants.LEFT);
```

```
lblCadastramento.setFont(new Font("Arial", Font.BOLD, 14));
```

```
lblCadastramento.setBounds(10, 0, 454, 59);
```

```
Cadastramento.add(lblCadastramento);
```

```
JLabel lblLatitude = new JLabel("Latitude");
```

```
lblLatitude.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblLatitude.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblLatitude.setBounds(10, 125, 150, 30);
```



```
Cadastramento.add(lblLatitude);
```

```
tfLatitude = new JTextField();
```

```
tfLatitude.setBounds(170, 125, 211, 30);
```

```
Cadastramento.add(tfLatitude);
```

```
tfLatitude.setColumns(10);
```

```
JLabel lblLongitude = new JLabel("Longitude");
```

```
lblLongitude.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblLongitude.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblLongitude.setBounds(10, 185, 150, 30);
```

```
Cadastramento.add(lblLongitude);
```

```
tfLongitude = new JTextField();
```

```
tfLongitude.setColumns(10);
```

```
tfLongitude.setBounds(170, 185, 211, 30);
```

```
Cadastramento.add(tfLongitude);
```

```
JLabel lblData = new JLabel("Data");
```

```
lblData.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblData.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblData.setBounds(10, 245, 150, 30);
```

```
Cadastramento.add(lblData);
```

```
tfDia = new JTextField();  
tfDia.setToolTipText("dia");  
tfDia.setBounds(170, 246, 50, 30);  
Cadastramento.add(tfDia);  
tfDia.setColumns(10);
```

```
JLabel lblHora = new JLabel("Hor\u00E1rio");  
lblHora.setHorizontalAlignment(SwingConstants.CENTER);  
lblHora.setFont(new Font("Arial", Font.PLAIN, 14));  
lblHora.setBounds(10, 305, 150, 30);  
Cadastramento.add(lblHora);
```

```
JLabel lblBarra1 = new JLabel("/");  
lblBarra1.setToolTipText("");  
lblBarra1.setHorizontalAlignment(SwingConstants.CENTER);  
lblBarra1.setBounds(300, 246, 33, 30);  
Cadastramento.add(lblBarra1);
```

```
JLabel lblPonto = new JLabel(":");  
lblPonto.setToolTipText("");  
lblPonto.setHorizontalAlignment(SwingConstants.CENTER);  
lblPonto.setBounds(218, 306, 33, 30);  
Cadastramento.add(lblPonto);
```

```
JLabel lblBarra2 = new JLabel("/");  
lblBarra2.setToolTipText("");  
lblBarra2.setHorizontalAlignment(SwingConstants.CENTER);  
lblBarra2.setBounds(218, 246, 33, 30);  
Cadastramento.add(lblBarra2);
```

```
tfMes = new JTextField();  
tfMes.setToolTipText("m\u00EAs");  
tfMes.setColumns(10);  
tfMes.setBounds(251, 246, 50, 30);  
Cadastramento.add(tfMes);
```

```
tfAno = new JTextField();  
tfAno.setToolTipText("ano");  
tfAno.setColumns(10);  
tfAno.setBounds(331, 246, 50, 30);  
Cadastramento.add(tfAno);
```

```
tfHora = new JTextField();  
tfHora.setToolTipText("hora");  
tfHora.setColumns(10);  
tfHora.setBounds(170, 306, 50, 30);  
Cadastramento.add(tfHora);
```

```
tfMinuto = new JTextField();  
tfMinuto.setToolTipText("minuto");  
tfMinuto.setColumns(10);  
tfMinuto.setBounds(251, 306, 50, 30);  
Cadastramento.add(tfMinuto);
```

```
JLabel lblPonto2 = new JLabel(":");  
lblPonto2.setToolTipText("");  
lblPonto2.setHorizontalAlignment(SwingConstants.CENTER);  
lblPonto2.setBounds(300, 306, 33, 30);  
Cadastramento.add(lblPonto2);
```

```
tfSegundos = new JTextField();  
tfSegundos.setToolTipText("segundos");  
tfSegundos.setColumns(10);  
tfSegundos.setBounds(331, 306, 50, 30);  
Cadastramento.add(tfSegundos);
```

```
JLabel lblCodigo = new JLabel("C\u00F3digo de Identifica\u00E7\u00E3o");  
lblCodigo.setHorizontalAlignment(SwingConstants.CENTER);  
lblCodigo.setFont(new Font("Arial", Font.PLAIN, 14));  
lblCodigo.setBounds(10, 75, 150, 30);  
Cadastramento.add(lblCodigo);
```

```
tfCod = new JTextField();

tfCod.setColumns(10);

tfCod.setBounds(170, 75, 211, 30);

Cadastramento.add(tfCod);


JButton btnVoltar = new JButton("< Voltar");

btnVoltar.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        tfCod.setText("");

        tfLatitude.setText("");

        tfLongitude.setText("");

        tfDia.setText("");

        tfMes.setText("");

        tfAno.setText("");

        tfHora.setText("");

        tfMinuto.setText("");

        tfSegundos.setText("");

        lblImagem.setIcon(null);

        imagem = null;

        trocaTela("HUB");

    }

});

btnVoltar.setFont(new Font("Tahoma", Font.BOLD, 11));

btnVoltar.setBounds(10, 383, 150, 30);
```

```

Cadastramento.add(btnVoltar);

JButton btnCadastrar = new JButton("Cadastrar");

btnCadastrar.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        if(p.checaCod(Main.CD, tfCod.getText())) {

            JOptionPane.showMessageDialog(null, "Código já existente");

        } else {

            if(imagem != null && tfCod.getText() != "") {

                Cadastro cad = new
Cadastro(tfCod.getText(),tfLatitude.getText(),tfLongitude.getText(),(tfDia.getText() +
"/"+ tfMes.getText()+ "/" + tfAno.getText()), (tfHora.getText() + ":" + tfMinuto.getText()
+ ":" + tfSegundos.getText()),getImagem());

                Main.CD.add(cad);

                System.out.println(tfCod.getText() + "----" + tfLatitude.getText() + "----" +
tfLongitude.getText() + "----" + (tfDia.getText() + "/" + tfMes.getText()+ "/" +
tfAno.getText()) + "----" + (tfHora.getText() + ":" + tfMinuto.getText() + ":" +
tfSegundos.getText()));

                JOptionPane.showMessageDialog(null, "Cadastro Efetuado com
Sucesso");

                tfCod.setText("");

                tfLatitude.setText("");

                tfLongitude.setText("");

                tfDia.setText("");

                tfMes.setText("");

                tfAno.setText("");

```

```
tfHora.setText("");

tfMinuto.setText("");

tfSegundos.setText("");

lblImagem.setIcon(null);

imagem = null;

} else {

    JOptionPane.showMessageDialog(null, "Insira uma imagem e/ou um
código");

}

}

}

});

btnCadastrar.setFont(new Font("Tahoma", Font.BOLD, 11));

btnCadastrar.setBounds(514, 388, 150, 30);

Cadastramento.add(btnCadastrar);


JPanel Imagem = new JPanel();

Imagem.setBounds(464, 29, 200, 200);

Cadastramento.add(Imagem);


lblImagem = new JLabel("");

Imagem.add(lblImagem);


JButton btnSelecionarImagem = new JButton("Selecionar Imagem...");

btnSelecionarImagem.addActionListener(new ActionListener() {
```

```
public void actionPerformed(ActionEvent e) {

    imagem = selecionarImagem();

    abrirImagem(imagem, lblImagem);

}

});

btnSelecionarImagem.setBounds(464, 240, 200, 23);

Cadastramento.add(btnSelecionarImagem);


JPanel Deletar = new JPanel();

contentPane.add(Deletar, "DELETAR");

Deletar.setLayout(null);


JButton btnVoltar_1 = new JButton("< Voltar");

btnVoltar_1.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        tfCodDel.setText("");

        trocaTela("HUB");

    }

});

btnVoltar_1.setFont(new Font("Tahoma", Font.BOLD, 11));

btnVoltar_1.setBounds(10, 383, 150, 30);

Deletar.add(btnVoltar_1);
```



```

JButton btnDeletar = new JButton("Deletar");

btnDeletar.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        if(p.checaCod(Main.CD, tfCodDel.getText())) {

            int index = p.getIndex(Main.CD,tfCodDel.getText());

            String[] menu = {"Confirmar","Cancelar"};

            int opt = JOptionPane.showOptionDialog(null, "Código encontrado,
deseja mesmo deleta-lo?", "Deletar Cadastro",JOptionPane.DEFAULT_OPTION,

                JOptionPane.INFORMATION_MESSAGE, null,menu,menu[0]);

            if(opt == 0) {

                Main.CD.remove(index);

                JOptionPane.showMessageDialog(null, "Cadastro Deletado");

                tfCodDel.setText("");

            }

        } else {

            JOptionPane.showMessageDialog(null, "Código não encontrado");

        }

    }

});

btnDeletar.setFont(new Font("Tahoma", Font.BOLD, 11));

btnDeletar.setBounds(514, 383, 150, 30);

Deletar.add(btnDeletar);

```

```
JLabel lblDeletar = new JLabel("Insira o C\u00F3digo de  
Identifica\u00E7\u00E3o do Cadastro para deleta-lo");
```

```
lblDeletar.setHorizontalAlignment(SwingConstants.LEFT);
```

```
lblDeletar.setFont(new Font("Arial", Font.BOLD, 14));
```

```
lblDeletar.setBounds(10, 0, 454, 59);
```

```
Deletar.add(lblDeletar);
```

```
JLabel lblCod = new JLabel("C\u00F3digo de Identifica\u00E7\u00E3o");
```

```
lblCod.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblCod.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblCod.setBounds(10, 68, 150, 30);
```

```
Deletar.add(lblCod);
```

```
tfCodDel = new JTextField();
```

```
tfCodDel.setText("");
```

```
tfCodDel.setBounds(171, 69, 210, 30);
```

```
Deletar.add(tfCodDel);
```

```
tfCodDel.setColumns(10);
```

```
JPanel Consultar = new JPanel();
```

```
Consultar.setLayout(null);
```

```
contentPane.add(Consultar, "CONSULTAR");
```

```
JLabel lblConsul = new JLabel("Insira o C\u00F3digo de  
Identifica\u00E7\u00E3o do Cadastro para consulta-lo");
```

```
lblConsul.setHorizontalAlignment(SwingConstants.LEFT);
```

```
lblConsul.setFont(new Font("Arial", Font.BOLD, 14));
```

```
lblConsul.setBounds(10, 0, 454, 59);
```

```
Consultar.add(lblConsul);
```

```
JLabel lblCod_1 = new JLabel("C\u00F3digo de Identifica\u00E7\u00E3o");
```

```
lblCod_1.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblCod_1.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblCod_1.setBounds(10, 68, 150, 30);
```

```
Consultar.add(lblCod_1);
```

```
tfCodCon = new JTextField();
```

```
tfCodCon.setText("");
```

```
tfCodCon.setColumns(10);
```

```
tfCodCon.setBounds(171, 69, 210, 30);
```

```
Consultar.add(tfCodCon);
```

```
JLabel lblLat1 = new JLabel("Latitude");
```

```
lblLat1.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblLat1.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblLat1.setBounds(10, 127, 150, 30);
```

```
Consultar.add(lblLat1);
```

```
JLabel lblLat2 = new JLabel("");  
lblLat2.setForeground(Color.BLACK);  
lblLat2.setBackground(Color.BLACK);  
lblLat2.setHorizontalAlignment(SwingConstants.CENTER);  
lblLat2.setFont(new Font("Arial", Font.PLAIN, 14));  
lblLat2.setBounds(171, 125, 210, 30);  
Consultar.add(lblLat2);
```

```
JLabel lblLong1 = new JLabel("Longitude");  
lblLong1.setHorizontalAlignment(SwingConstants.CENTER);  
lblLong1.setFont(new Font("Arial", Font.PLAIN, 14));  
lblLong1.setBounds(10, 186, 150, 30);  
Consultar.add(lblLong1);
```

```
JLabel lblLong2 = new JLabel("");  
lblLong2.setHorizontalAlignment(SwingConstants.CENTER);  
lblLong2.setForeground(Color.BLACK);  
lblLong2.setFont(new Font("Arial", Font.PLAIN, 14));  
lblLong2.setBackground(Color.BLACK);  
lblLong2.setBounds(171, 184, 210, 30);  
Consultar.add(lblLong2);
```

```
JLabel lblData1 = new JLabel("Data");  
lblData1.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblData1.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblData1.setBounds(10, 248, 150, 30);
```

```
Consultar.add(lblData1);
```

```
JLabel lblData2 = new JLabel("");
```

```
lblData2.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblData2.setForeground(Color.BLACK);
```

```
lblData2.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblData2.setBackground(Color.BLACK);
```

```
lblData2.setBounds(171, 248, 210, 30);
```

```
Consultar.add(lblData2);
```

```
JLabel lblHora1 = new JLabel("Hor\u00E1rio");
```

```
lblHora1.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblHora1.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblHora1.setBounds(10, 300, 150, 30);
```

```
Consultar.add(lblHora1);
```

```
JLabel lblHora2 = new JLabel("");
```

```
lblHora2.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblHora2.setForeground(Color.BLACK);
```

```
lblHora2.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblHora2.setBackground(Color.BLACK);
```

```
lblHora2.setBounds(171, 300, 210, 30);
```

```
Consultar.add(lblHora2);
```

```
JPanel Imagem_1 = new JPanel();
```

```
Imagem_1.setBackground(Color.WHITE);
```

```
Imagem_1.setBounds(464, 29, 200, 200);
```

```
Consultar.add(Imagem_1);
```

```
JLabel lblImagem_1 = new JLabel("");
```

```
Imagem_1.add(lblImagem_1);
```

```
JPanel Atualizar = new JPanel();
```

```
contentPane.add(Atualizar, "ATUALIZAR");
```

```
Atualizar.setLayout(null);
```

```
JLabel lblCadastramentoAtt = new JLabel("Insira as informa\u00E7\u00F5es  
para efetura a atualiza\u00E7\u00E3o no cadastramento");
```

```
lblCadastramentoAtt.setBounds(10, 11, 478, 17);
```

```
lblCadastramentoAtt.setHorizontalAlignment(SwingConstants.LEFT);
```

```
lblCadastramentoAtt.setFont(new Font("Arial", Font.BOLD, 14));
```

```
Atualizar.add(lblCadastramentoAtt);
```

```
JLabel lblCodAtt = new JLabel("C\u00F3digo de Identifica\u00E7\u00E3o");
```

```
lblCodAtt.setFont(new Font("Arial", Font.PLAIN, 14));  
lblCodAtt.setHorizontalAlignment(SwingConstants.CENTER);  
lblCodAtt.setBounds(10, 60, 150, 30);  
Atualizar.add(lblCodAtt);
```

```
tfCodAtt = new JTextField();  
tfCodAtt.setBounds(159, 61, 211, 30);  
Atualizar.add(tfCodAtt);  
tfCodAtt.setColumns(10);
```

```
JLabel lblLatAtt = new JLabel("Latitude");  
lblLatAtt.setHorizontalAlignment(SwingConstants.CENTER);  
lblLatAtt.setFont(new Font("Arial", Font.PLAIN, 14));  
lblLatAtt.setBounds(10, 123, 150, 30);  
Atualizar.add(lblLatAtt);
```

```
tfLatAtt = new JTextField();  
tfLatAtt.setColumns(10);  
tfLatAtt.setBounds(159, 124, 211, 30);  
Atualizar.add(tfLatAtt);
```

```
JLabel lblLongAtt = new JLabel("Longitude");  
lblLongAtt.setHorizontalAlignment(SwingConstants.CENTER);  
lblLongAtt.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblLongAtt.setBounds(10, 182, 150, 30);
```

```
Atualizar.add(lblLongAtt);
```

```
tfLongAtt = new JTextField();
```

```
tfLongAtt.setColumns(10);
```

```
tfLongAtt.setBounds(159, 182, 211, 30);
```

```
Atualizar.add(tfLongAtt);
```

```
JLabel lblDataAtt = new JLabel("Data");
```

```
lblDataAtt.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblDataAtt.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblDataAtt.setBounds(10, 246, 150, 30);
```

```
Atualizar.add(lblDataAtt);
```

```
tfDiaAtt = new JTextField();
```

```
tfDiaAtt.setToolTipText("dia");
```

```
tfDiaAtt.setBounds(159, 247, 50, 30);
```

```
Atualizar.add(tfDiaAtt);
```

```
tfDiaAtt.setColumns(10);
```

```
JLabel lblBarraAtt1 = new JLabel("/");
```

```
lblBarraAtt1.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblBarraAtt1.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblBarraAtt1.setBounds(207, 247, 33, 30);
```



```
Atualizar.add(lblBarraAtt1);
```

```
tfMesAtt = new JTextField();
```

```
tfMesAtt.setToolTipText("m\u00EAs");
```

```
tfMesAtt.setColumns(10);
```

```
tfMesAtt.setBounds(239, 247, 50, 30);
```

```
Atualizar.add(tfMesAtt);
```

```
JLabel lblBarraAtt2 = new JLabel("/");
```

```
lblBarraAtt2.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblBarraAtt2.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblBarraAtt2.setBounds(289, 246, 33, 30);
```

```
Atualizar.add(lblBarraAtt2);
```

```
tfAnoAtt = new JTextField();
```

```
tfAnoAtt.setToolTipText("ano");
```

```
tfAnoAtt.setColumns(10);
```

```
tfAnoAtt.setBounds(320, 247, 50, 30);
```

```
Atualizar.add(tfAnoAtt);
```

```
JLabel lblHoraAtt = new JLabel("Hor\u00E1rio");
```

```
lblHoraAtt.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblHoraAtt.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblHoraAtt.setBounds(10, 307, 150, 30);
```

```
Atualizar.add(lblHoraAtt);
```

```
tfHoraAtt = new JTextField();
```

```
tfHoraAtt.setToolTipText("dia");
```

```
tfHoraAtt.setColumns(10);
```

```
tfHoraAtt.setBounds(159, 308, 50, 30);
```

```
Atualizar.add(tfHoraAtt);
```

```
JLabel lblPontoAtt1 = new JLabel(":");
```

```
lblPontoAtt1.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblPontoAtt1.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblPontoAtt1.setBounds(207, 307, 33, 30);
```

```
Atualizar.add(lblPontoAtt1);
```

```
tfMinutoAtt = new JTextField();
```

```
tfMinutoAtt.setToolTipText("dia");
```

```
tfMinutoAtt.setColumns(10);
```

```
tfMinutoAtt.setBounds(239, 308, 50, 30);
```

```
Atualizar.add(tfMinutoAtt);
```

```
JLabel lblPontoAtt2 = new JLabel(":");
```

```
lblPontoAtt2.setHorizontalAlignment(SwingConstants.CENTER);
```

```
lblPontoAtt2.setFont(new Font("Arial", Font.PLAIN, 14));
```

```
lblPontoAtt2.setBounds(288, 307, 33, 30);
```

```
Atualizar.add(lblPontoAtt2);
```

```
tfSegundoAtt = new JTextField();
```

```
tfSegundoAtt.setToolTipText("ano");
```

```
tfSegundoAtt.setColumns(10);
```

```
tfSegundoAtt.setBounds(320, 307, 50, 30);
```

```
Atualizar.add(tfSegundoAtt);
```

```
JPanel Imagem_2 = new JPanel();
```

```
Imagem_2.setBounds(464, 39, 200, 200);
```

```
Atualizar.add(Imagem_2);
```

```
JLabel lblImagem_2 = new JLabel("");
```

```
Imagem_2.add(lblImagem_2);
```

```
JButton btnSelecionarImagem_1 = new JButton("Selecionar Imagem...");
```

```
btnSelecionarImagem_1.addActionListener(new ActionListener() {
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        imagem = selecionarImagem();
```

```
        abrirImagem(imagem, lblImagem_2);
```

```
    }
```

```
});
```

```
btnSelecionarImagem_1.setBounds(464, 251, 200, 23);
```

```
Atualizar.add(btnSelecionarImagem_1);
```

```
JButton btnVoltarAtt = new JButton("< Voltar");
```

```
btnVoltarAtt.addActionListener(new ActionListener() {
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        tfCodAtt.setText("");
```

```
        tfLatAtt.setText("");
```

```
        tfLongAtt.setText("");
```

```
        tfDiaAtt.setText("");
```

```
        tfMesAtt.setText("");
```

```
        tfAnoAtt.setText("");
```

```
        tfHoraAtt.setText("");
```

```
        tfMinutoAtt.setText("");
```

```
        tfSegundoAtt.setText("");
```

```
        lblImagem_2.setIcon(null);
```

```
        imagem = null;
```

```
        trocaTela("HUB");
```

```
    }
```

```
});
```

```
btnVoltarAtt.setFont(new Font("Tahoma", Font.BOLD, 11));
```

```
btnVoltarAtt.setBounds(10, 388, 150, 30);
```

```
Atualizar.add(btnVoltarAtt);
```

```

JButton btnAtualizar = new JButton("Atualizar");

btnAtualizar.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        if(p.checaCod(Main.CD, tfCodAtt.getText())) {

            if(imagem != null && tfCodAtt.getText() != "") {

                int index = p.getIndex(Main.CD, tfCodAtt.getText());

                Cadastro cad = new
Cadastro(tfCodAtt.getText(),tfLatAtt.getText(),tfLongAtt.getText(),(tfDiaAtt.getText() +
"/"+ tfMesAtt.getText()+ "/" + tfAnoAtt.getText()), (tfHoraAtt.getText() + ":" +
tfMinutoAtt.getText() + ":" + tfSegundoAtt.getText()),getImagem());

                Main.CD.set(index, cad);

                tfCodAtt.setText("");

                tfLatAtt.setText("");

                tfLongAtt.setText("");

                tfDiaAtt.setText("");

                tfMesAtt.setText("");

                tfAnoAtt.setText("");

                tfHoraAtt.setText("");

                tfMinutoAtt.setText("");

                tfSegundoAtt.setText("");

                lblImagem_2.setIcon(null);

                imagem = null;

            } else {

```

```
        JOptionPane.showMessageDialog(null, "Insira uma imagem e/ou um  
código");
```

```
    }
```

```
    } else {
```

```
        JOptionPane.showMessageDialog(null, "Código não encontrado");
```

```
    }
```

```
}
```

```
});
```

```
btnAtualizar.setFont(new Font("Tahoma", Font.BOLD, 11));
```

```
btnAtualizar.setBounds(514, 388, 150, 30);
```

```
Atualizar.add(btnAtualizar);
```

```
JButton btnConsultar = new JButton("Consultar");
```

```
btnConsultar.addActionListener(new ActionListener() {
```

```
    public void actionPerformed(ActionEvent e) {
```

```
        String cod = tfCodCon.getText();
```

```
        System.out.println(cod);
```

```
        if(p.checaCod(Main.CD, cod) ) {
```

```
            int index = p.getIndex(Main.CD, cod);
```

```
            System.out.println("index = " +index);
```

```
            Cadastro c = Main.CD.get(index);
```

```
            System.out.println(c.toString());
```

```
            lblLat2.setText(c.getLat());
```

```
            lblLong2.setText(c.getLon());
```

```

        lblData2.setText(c.getData());

        lblHora2.setText(c.getHora());

        abrirlImagem(c.getBit(),lblImagem_1,c);

    }

    else {

        JOptionPane.showMessageDialog(null, "Código não encontrado");

    }

}

});

btnConsultar.setFont(new Font("Tahoma", Font.BOLD, 11));

btnConsultar.setBounds(514, 383, 150, 30);

Consultar.add(btnConsultar);


JButton btnVoltar_1_1 = new JButton("< Voltar");

btnVoltar_1_1.addActionListener(new ActionListener() {

    public void actionPerformed(ActionEvent e) {

        tfCodCon.setText("");

        lblLat2.setText("");

        lblLong2.setText("");

        lblData2.setText("");

        lblHora2.setText("");

        lblImagem_1.setIcon(null);

        trocaTela("HUB");
    }
});

```

```

    }

});

btnVoltar_1_1.setFont(new Font("Tahoma", Font.BOLD, 11));

btnVoltar_1_1.setBounds(10, 383, 150, 30);

Consultar.add(btnVoltar_1_1);

}

public void trocaTela(String layout) {

    CardLayout card = (CardLayout) contentPane.getLayout();

    card.show(contentPane,layout);

}

public File selecionarImagem() {

    JFileChooser fileChooser = new JFileChooser();

    FileNameExtensionFilter filtro = new FileNameExtensionFilter("Imagens em
JPEG e PNG", "jpg", "png");

    fileChooser.addChoosableFileFilter(filtro);

    fileChooser.setAcceptAllFileFilterUsed(false);

    fileChooser.setDialogType(JFileChooser.OPEN_DIALOG);

    fileChooser.showOpenDialog(this);

    return fileChooser.getSelectedFile();

}

public byte[] getImagem() {

    boolean isPng = false;

```



```
if(imagem != null) {  
    isPng = imagem.getName().endsWith("png");  
    try {  
        BufferedImage image = ImageIO.read(imagem);  
        ByteArrayOutputStream out = new ByteArrayOutputStream();  
        int type = BufferedImage.TYPE_INT_RGB;  
  
        if(isPng) {  
            type = BufferedImage.BITMASK;  
        }  
  
        BufferedImage novalmagem = new BufferedImage(195,195,type);  
        Graphics2D g = novalmagem.createGraphics();  
        g.setComposite(AlphaComposite.Src);  
        g.drawImage(image, 0,0,195,195,null);  
  
        if(isPng) {  
            ImageIO.write(novalmagem,"png",out);  
        } else {  
            ImageIO.write(novalmagem,"jpg",out);  
        }  
  
        out.flush();  
    }  
}
```

```

        byte[] byteArray = out.toByteArray();

        out.close();

        return byteArray;
    } catch (IOException e) {

        e.printStackTrace();

    }
}

return null;
}

```

```

private void abrirlmagem(Object source,JLabel lb) {

    if(source instanceof File) {

        ImageIcon icon = new ImageIcon(imagem.getAbsolutePath());

        icon.setImage(icon.getImage().getScaledInstance(195, 195,100));

        lb.setIcon(icon);

    } else if(source instanceof byte[]) {

        ImageIcon icon = new ImageIcon(getImagem());

        icon.setImage(icon.getImage().getScaledInstance(195, 195,100));

        lb.setIcon(icon);

    }

}

```

```

private void abrirlmagem(Object source,JLabel lb, Cadastro c) {

```

```

if(source instanceof File) {

    ImageIcon icon = new ImageIcon(imagem.getAbsolutePath());

    icon.setImage(icon.getImage().getScaledInstance(195, 195,100));

    lb.setIcon(icon);

} else if(source instanceof byte[]) {

    ImageIcon icon = new ImageIcon(c.getBit());

    icon.setImage(icon.getImage().getScaledInstance(195, 195,100));

    lb.setIcon(icon);

}

}

}

```

Classe Cadastro

```
package Entidades;
```

```
import java.util.ArrayList;
```

```
import javax.swing.JOptionPane;
```

```
import Algoritmos.Pesquisa;
```

```
public class Cadastro {
```

```
    private String cod,lat,lon,data,hora;
```

```
private byte[] bit;
```

```
public Cadastro(String cod, String lat, String lon, String data, String hora, byte[]  
bit) {
```

```
    this.cod = cod;
```

```
    this.lat = lat;
```

```
    this.lon = lon;
```

```
    this.data = data;
```

```
    this.hora = hora;
```

```
    this.bit = bit;
```

```
}
```

```
public String getCod() {
```

```
    return cod;
```

```
}
```

```
public void setCod(String cod) {
```

```
    this.cod = cod;
```

```
}
```

```
public String getLat() {
```

```
    return lat;
```

```
}
```

```
public void setLat(String lat) {  
    this.lat = lat;  
}
```

```
public String getLon() {  
    return lon;  
}
```

```
public void setLon(String lon) {  
    this.lon = lon;  
}
```

```
public String getData() {  
    return data;  
}
```

```
public void setData(String data) {  
    this.data = data;  
}
```

```
public String getHora() {  
    return hora;  
}
```

```
public void setHora(String hora) {
```

```
    this.hora = hora;
```

```
}
```

```
public String toString() {
```

```
    return "cod: " + cod + "/lat: " + lat + "/long: " + lon + "/data: " + data + "/hora: " +  
hora;
```

```
}
```

```
public byte[] getBit() {
```

```
    return bit;
```

```
}
```

```
public void setBit(byte[] bit) {
```

```
    this.bit = bit;
```

```
}
```

```
public static void exhibirCodigosCadastral(ArrayList<Cadastral> c) {
```

```
    String msg = "";
```

```
    for(int i = 0; i < c.size(); i++) {
```

```
        Cadastral cad = c.get(i);
```

```
        msg += ("\\n Cadastral n° " + (i+1) + " Código: " + cad.getCod());
```

```
}
```

```
JOptionPane.showMessageDialog(null, msg);
```

```

    }

    public static void ordenaLista(ArrayList<Cadastro> c,int v[] ) {

        Pesquisa p = new Pesquisa();

        Cadastro aux;

        if(c.size() == v.length) {

            for(int i = 0; i < c.size(); i++) {

                if(Integer.parseInt(c.get(i).getCod()) != v[i] ) {

                    int index = p.getIndex(c, Integer.toString(v[i]));

                    aux = c.get(i);

                    c.set(i, c.get(index));

                    c.set(index, aux);

                }

            }

        }

    }

}

```

Classe Vetor

```
package Algoritmos;
```

```
import java.util.ArrayList;
```

```
import Entidades.Cadastro;
```

```
public class Vetor {
```

```
    private int codigos[];
```

```
    public Vetor(ArrayList<Cadastro> c) {
```

```
        codigos = new int[c.size()];
```

```
        for(int i = 0; i < c.size(); i++) {
```

```
            codigos[i] = Integer.parseInt(c.get(i).getCod());
```

```
        }
```

```
    }
```

```
    public Vetor(){
```

```
    }
```

```
    public int[] getVetor() {
```

```
        return codigos;
```

```
    }
```

```
    public int[] preencherVetor(int v[]) {
```

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



```

        v[i] = (int)(Math.random() * v.length);
    }

    return v;
}

```

```

public int[] preencherVetor(int v[],int tam) {

    v = new int[tam];

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

        v[i] = (int)(Math.random() * tam);

    }

    return v;

}

```

```

public int[] preencherVetorSemiOrd(int v[],int tam) {

    v = new int[tam];

    int random = (int)(Math.random()*(v.length));

    for(int i = 0; i < (v.length/2);i++ ) {

        v[i] = i;

    }

    for(int i =(v.length/2);i < v.length; i++ ) {

        random = (int)(Math.random()*(v.length));

        while(random < (v.length/2)) {

            random = (int)(Math.random()*(v.length));

        }

    }

}

```

```
        v[i] = random;
    }

    return v;
}
```

```
public int[] preencherVetorOrd(int v[],int tam) {

    v = new int[tam];

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

        v[i] = i;

    }


    return v;

}
}
```

Classe Ordenacao

```
package Algoritmos;
```

```
public class Ordenacao {

    //Bubble Sort

    public void bubble(int v[]) {

        int ini = 1, fim, chg;

        while (ini < v.length) {

            fim = v.length - 1;
```

```

while (fim >= ini) {
    if (v[fim-1] > v[fim]) {
        chg    = v[fim-1];
        v[fim-1] = v[fim];
        v[fim]   = chg;
    }
    fim--;
}
ini++;
}
}

```

//Selection Sort

```

public void selection(int v[]) {
    int ini = 0, sch, chg, men;
    while (ini < v.length - 1) {
        chg = ini;
        men = v[ini];
        sch = ini + 1;
        while (sch < v.length) {
            if (v[sch] < men) {
                chg = sch;
                men = v[sch];
            }
        }
    }
}

```

```
        sch++;  
    }  
    v[chg] = v[ini];  
    v[ini] = men;  
    ini++;  
}  
}
```

//Insertion Sort

```
public void insertion(int v []) {  
    int ini = 1, chg, men;  
    while (ini < v.length) {  
        men = v[ini];  
        chg = ini - 1;  
        while (chg >= 0 && men < v[chg]) {  
            v[chg + 1] = v[chg];  
            chg--;  
        }  
        v[chg + 1] = men;  
        ini++;  
    }  
}
```

//Quick Sort

```

public void quick(int v[]) {
    quick(v, 0, v.length - 1);
}

private void quick(int v[], int lef, int rig) {
    int aPiv, aLef, aRig, piv;

    aLef = lef;
    aRig = rig;
    piv = v[lef];

    while (lef < rig) {
        while ((v[rig] >= piv) && (lef < rig)) {
            rig--;
        }
        if (lef != rig) {
            v[lef] = v[rig];
            lef++;
        }
        while ((v[lef] <= piv) && (lef < rig)) {
            lef++;
        }
        if (lef != rig) {
            v[rig] = v[lef];
            rig--;
        }
    }
}

```

```

        v[lef] = piv;

        aPiv = lef;

        if (aLef < aPiv) quick(v, aLef, aPiv - 1);

        if (aRig > aPiv) quick(v, aPiv + 1, aRig);

    }

}

```

Classe Pesquisa

```

package Algoritmos;

import java.util.ArrayList;

import Entidades.Cadastro;

public class Pesquisa {

    public boolean checaCod(ArrayList<Cadastro> cd,String cod) {

        for(int i = 0; i < cd.size();i++) {

            Cadastro c = cd.get(i);

            if(c.getCod().equals(cod)) {

                return true;

            }

        }

        return false;

    }

}

```

```
public int getIndex(ArrayList<Cadastro> cd,String cod) {  
    for(int i = 0; i < cd.size();i++) {  
        Cadastro c = cd.get(i);  
        if(c.getCod().equals(cod)) {  
            return i;  
        }  
    }  
    return -1;  
}  
}
```

Classe Main

```
package Main;
```

```
import java.awt.EventQueue;
```

```
import java.util.ArrayList;
```

```
import Entidades.Cadastro;
```

```
import Frame.Janela;
```

```
public class Main {
```

```
public static ArrayList<Cadastro> CD = new ArrayList<Cadastro>();
```

```
public static void main(String[] args) {
```

```
    EventQueue.invokeLater(new Runnable() {
```

```
        public void run() {
```

```
            try {
```

```
                Janela frame = new Janela();
```

```
                frame.setVisible(true);
```

```
            } catch (Exception e) {
```

```
                e.printStackTrace();
```

```
            }
```

```
        }
```

```
    });
```

```
}
```

```
}
```


Fichas



FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Eric Santos da Silva TURMA: CCAP20 RA: F015BA7

CURSO: Ciências da Computação CAMPUS: Norte SEMESTRE: 4 TURNO: Noturno

CÓDIGO DA ATIVIDADE: 77B1 SEMESTRE: 4 ANO GRADE: Estrutura de Dados

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
23/10	Pesquisa sobre os Algoritmos	3			
24/10	Implementação dos Algoritmos em Teste	5			
25/10	Estágios Iniciais de Criação da UI	6			
29/10	Desenvolvimento da Aplicação	6			
30/10	Criação de Métodos de Input de Dados	5			
01/11	Teste de Funcionamento da Aplicação	3			
01/11	Coleta Primária de Dados	2			
06/11	Tratamento de Erros na Aplicação	8			
07/11	Início da Parte Escrita	5			
07/11	Pesquisa sobre os Algoritmos	2			
08/11	Coleta de Dados Secundárias	7			
13/11	Análise dos Dados Coletados	8			
14/11	Criação de Gráficos e Imagens	5			
15/11	Finalização da Parte Escrita	5			
16/11	Elaboração das Referências Bibliográficas	2			

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: _____	
AVALIAÇÃO: _____	Aprovado ou Reprovado _____
NOTA: _____	
DATA: ____/____/____	
CARIMBO E ASSINATURA DO COORDENADOR DO CURSO _____	



FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Fábio Miguel Naumes

TURMA: CCAP20

RA: N435CE1

CURSO: Ciências da Computação

CAMPUS: Norte

SEMESTRE: 4

TURNO: Noturno

CÓDIGO DA ATIVIDADE: 77B1

SEMESTRE: 4

ANO GRADE: Estrutura de Dados

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
23/10	Pesquisa sobre os Algoritmos	3			
24/10	Implementação dos Algoritmos em Teste	5			
25/10	Estágios Iniciais de Criação da UI	6			
29/10	Desenvolvimento da Aplicação	6			
30/10	Criação de Métodos de Input de Dados	5			
01/11	Teste de Funcionamento da Aplicação	3			
01/11	Coleta Primária de Dados	2			
06/11	Tratamento de Erros na Aplicação	8			
07/11	Início da Parte Escrita	5			
07/11	Pesquisa sobre os Algoritmos	2			
08/11	Coleta de Dados Secundárias	7			
13/11	Análise dos Dados Coletados	8			
14/11	Criação de Gráficos e Imagens	5			
15/11	Finalização da Parte Escrita	5			
16/11	Elaboração das Referências Bibliográficas	2			

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: _____

AValiação: _____ Aprovado ou Reprovado

NOTA: _____

DATA: ____/____/____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO



FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

UNIVERSIDADE PAULISTA

NOME: Leonardo Algarve Rezende

TURMA: CCAP20

RA: N3950A9

CURSO: Ciências da Computação

CAMPUS: Norte

SEMESTRE: 4

TURNO: Noturno

CÓDIGO DA ATIVIDADE: 77B1

SEMESTRE: 4

ANO GRADE: Estrutura de Dados

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
23/10	Pesquisa sobre os Algoritmos	3			
24/10	Implementação dos Algoritmos em Teste	5			
25/10	Estágios Iniciais de Criação da UI	6			
29/10	Desenvolvimento da Aplicação	6			
30/10	Criação de Métodos de Input de Dados	5			
01/11	Teste de Funcionamento da Aplicação	3			
01/11	Coleta Primária de Dados	2			
06/11	Tratamento de Erros na Aplicação	8			
07/11	Início da Parte Escrita	5			
07/11	Pesquisa sobre os Algoritmos	2			
08/11	Coleta de Dados Secundárias	7			
13/11	Análise dos Dados Coletados	8			
14/11	Criação de Gráficos e Imagens	5			
15/11	Finalização da Parte Escrita	5			
16/11	Elaboração das Referências Bibliográficas	2			

(1) Horas atribuídas de acordo com o regulamento das Atividades Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: _____

AValiação: _____

Aprovado ou Reprovado

NOTA: _____

DATA: ____/____/____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO



FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Wesley Santos Coelho

TURMA: CCAP20

RA: N479124

CURSO: Ciências da Computação

CAMPUS: Norte

SEMESTRE: 4

TURNO: Noturno

CÓDIGO DA ATIVIDADE: 77B1

SEMESTRE: 4

ANO GRADE: Estrutura de Dados

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
23/10	Pesquisa sobre os Algoritmos	3			
24/10	Implementação dos Algoritmos em Teste	5			
25/10	Estágios Iniciais de Criação da UI	6			
29/10	Desenvolvimento da Aplicação	6			
30/10	Criação de Métodos de Input de Dados	5			
01/11	Teste de Funcionamento da Aplicação	3			
01/11	Coleta Primária de Dados	2			
06/11	Tratamento de Erros na Aplicação	8			
07/11	Início da Parte Escrita	5			
07/11	Pesquisa sobre os Algoritmos	2			
08/11	Coleta de Dados Secundárias	7			
13/11	Análise dos Dados Coletados	8			
14/11	Criação de Gráficos e Imagens	5			
15/11	Finalização da Parte Escrita	5			
16/11	Elaboração das Referências Bibliográficas	2			

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: _____

AValiação:

Aprovado ou Reprovado

NOTA: _____

DATA: ____/____/____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO