

**Ciência Da Computação**

**3**ª **Periodo**

**“DESENVOLVIMENTO DE SISTEMA PARA ANÁLISE DE PERFORMANCE DE ALGORITMOS DE ORDENAÇÃO DE DADOS”**

**Índice**

Objetivo Trabalho\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_3 Introdução\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_4

Referencial Teórico\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_6

Desenvolvimento\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_15

Resultados e Discussão\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_25

Considerações Finais\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_28

Referências Bibliográficas\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_29

Código Fonte\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_30

Fichas APS\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_43

**Objetivo do Trabalho**

Esse trabalho tem como objetivo apresentar o que são métodos de ordenação, quais são os mais conhecidos, demonstrar como funcionam, aplicá-los em um programa, obter os resultados de desempenho e discutir sobre eles.

**Introdução**

Ordenação: Tornar mais simples, rápida e viável a recuperação de uma determinada informação, num conjunto grande de informações.

Algoritmo de ordenação em ciência da computação é um algoritmo que coloca os elementos de uma dada sequência em certa ordem, em outras palavras, efetua sua ordenação completa ou parcial. Existem várias razões para se ordenar uma sequência. Uma delas é a possibilidade se acessar seus dados de modo mais eficiente.

Não existe um método de ordenação considerado universalmente superior a todos os outros. É necessário analisar o problema e, com base nas características dos dados, decidirem qual o método que melhor se aplica a ele.

Alguns aspectos de medida de eficiência: tempo para codificação do programa de ordenação; tempo de máquina para a execução do programa; espaço de memória necessário. Normalmente o tempo gasto é medido pelo número de operações críticas, ao problema, que são efetuadas. Nesse caso as operações críticas são: comparações de chaves; movimentação de registros ou de ponteiros; troca entre dois registros. A medida é tomada pela análise do melhor caso, do pior caso e do caso médio. O resultado é uma fórmula em função de n (número de registros do arquivo).

Na realidade o tempo gasto não depende exclusivamente do tamanho do arquivo, mas também de outros aspectos, por exemplo: se existe uma pré ordenação no arquivo ou não.

Abaixo são apresentados três dos algoritmos de ordenação mais conhecidos.

**Bubble Sort**

É o método mais intuitivo e simples de ser implementado, porém é o menos eﬁciente.

**Selection Sort**

Neste tipo de ordenação, se seleciona o menor elemento do subvetor não ordenado e troca com o primeiro elemento deste subvetor

**Quick Sort**

O quicksort é um algoritmo de ordenação considerado entre os mais rápidos e eficientes.

O ambiente de desenvolvimento do programa será o Visual Studio, utilizando a linguagem C#.

**Referencial Teórico**

A escolha de um método de ordenação deve – se, a necessidade a qual o software requer, disponibilizando-se de uma biblioteca já estabelecida, porém não fixa, ou seja, podem surgir várias outras, o desenvolvedor pode optar por diversas formas de ordenar. Porém alguns métodos são mais simples e outros mais complexos ou mais eficientes e menos eficientes.

Existem vários desenvolvedores e criadores de métodos e lógicas, mas podemos colocar em destaque James Gosling, que agrega grande conhecimento e influência, assim como experiência, sendo referência na área tecnológica, ajudando a inovar e produzir novos conceitos.

Ordenar corresponde ao processo de rearranjar um conjunto de objeto sem ordem ascendente ou descendente. O objetivo principal da ordenação é facilitar a recuperação posterior de itens do conjunto ordenado. O objetivo principal da ordenação é facilitar a recuperação posterior de itens do conjunto ordenado.

Serão apresentados agora os métodos de ordenação Bubble Sort, Selection Sort e Quick Sort.

Bubble Sort

Bubble sort é um simples e bem conhecido algoritmo de ordenação. Ele é usado mais para prática inicial, para ajudar na introdução aos algoritmos de ordenação. Bubble sort pertence aos O (n²) algoritmos de ordenação, o que o torna bastante ineficiente para ordenar grandes volumes de dados. Bubble sort é estável e adaptativo.

Algoritmo

1. Compara cada par de elementos adjacentes do início do vetor e, se eles estiverem na ordem reversa, troca eles de lugar.
2. Se ao menos uma troca foi feita, repete o primeiro passo.

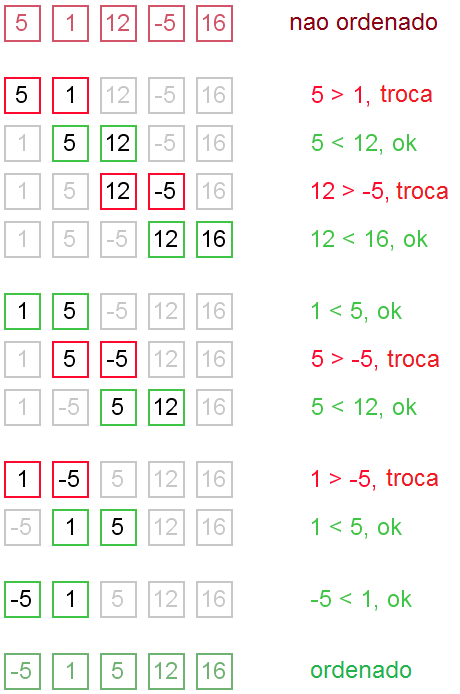
Você pode imaginar como se em cada passo grandes bolhas flutuam até a superfície e ficam lá. Em um momento, quando nenhuma bolha flutua, a ordenação para. Vejamos um exemplo do algoritmo ordenando um vetor para termos uma idéia mais clara sobre o bubble sort.

Análise de Complexidade

Em casos de complexidade mediana ou alta, o bubble sort possui desempenho de O (n²). Assim como, ele efetua O (n²) trocas nos piores casos. Bubble sort é adaptativo. Isso significa que em vetores quase ordenados seu desempenho estimado é O(n). Evite implementações que não confiram se o vetor já está ordenado em cada passo (qualquer troca feita). Essa checagem é necessária, em ordem para preservar propriedades adaptativas.

Exemplo:

Ordenando {5, 1, 12, -5, 16} usando bubble sort.



Ordenação por seleção

Ordenação por seleção é um dos algoritmos de ordenação O (n²), o que o torna bem ineficiente para ordenar grandes volumes de dados. Ordenação por seleção é notável por sua simples programação e pode superar em performance outros métodos de ordenação em certas situações.

Algoritmo

A ideia do algoritmo é bem simples. O vetor é imaginariamente dividido em duas partes, ordenada e não ordenada. No começo, a ordenada está vazia, enquanto a não ordenada contém todo o vetor. A cada passo, o algoritmo acha o elemento mínimo na parte não ordenada e adiciona ele para o final da parte ordenada. Quando a parte não ordenada se torna vazia, o algoritmo encerra.

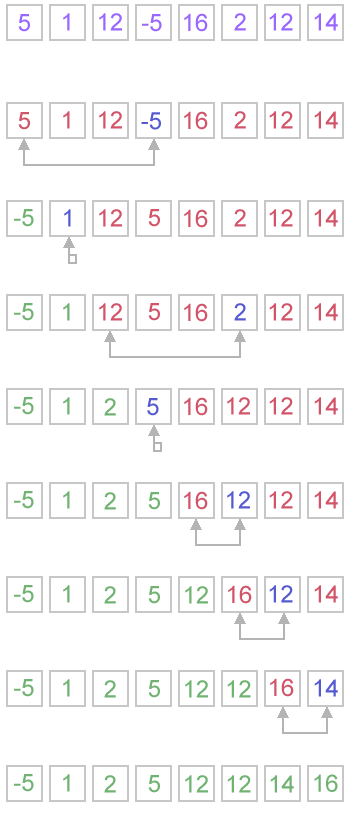
Análise de Complexidade

Ordenação por seleção acaba quando a parte não ordenada se torna vazia. Como nós sabemos, em cada passo o número de elementos não ordenados diminui por 1. Portanto, ordenação por seleção faz n passos (n é o número de elementos no vetor) antes de terminar. Cada passo requer achar um mínimo na parte não ordenada. Assim sendo, n + (n - 1) + (n - 2) + ... + 1, resulta em O (n²) números de comparações. Números de trocas podem variar de 0 (no caso de um vetor ordenado) a n - 1 (no caso do vetor ser ordenado em ordem reversa), que resulta em O(n) números de trocas. Em geral a complexidade do algoritmo é O (n²).

Em fato, ordenação por seleção requer (n – 1) números de trocas na maioria, tornando-o bastante eficiente em situações, onde a operação de obter é significantemente maior do que a opção de ler.

Exemplo:

Ordenando {5, 1, 12, -5, 16, 2, 12, 14} usando ordenação por seleção.



Quicksort

Quicksort é um rápido algoritmo de ordenação, que é usado não só para o meio educacional, mas altamente aplicado em prática. Em média, ele tem O(n log n) de complexidade, tornando o quicksort adequado para ordenar grandes volumes de dados.

A ideia do algoritmo é bem simples e quando você perceber, você poderá escrever quicksort tão rápido quando o bubble sort.

Algoritmo

A estratégia de dividir e conquistar é usada no quicksort. Abaixo o processo de recursão é descrito:

1. Escolha um valor pivô: Nós pegamos o valor do elemento do meio como valor pivô, mas pode ser qualquer valor, desde que esteja entre os valores a serem ordenados.

2. Particione: Reordena os elementos em tal método que, todos os elementos que forem menos que o pivô vão para a parte esquerda do vetor e todos os elementos maiores que o pivô, vão para a parte direita do vetor. Valores iguais ao pivô podem ficar em qualquer parte do vetor. Nota-se que, o vetor pode ser dividido em partes não iguais.

3. Ordene ambas as partes: Aplique o algoritmo quicksort recursivamente para as partes esquerdas e direitas.

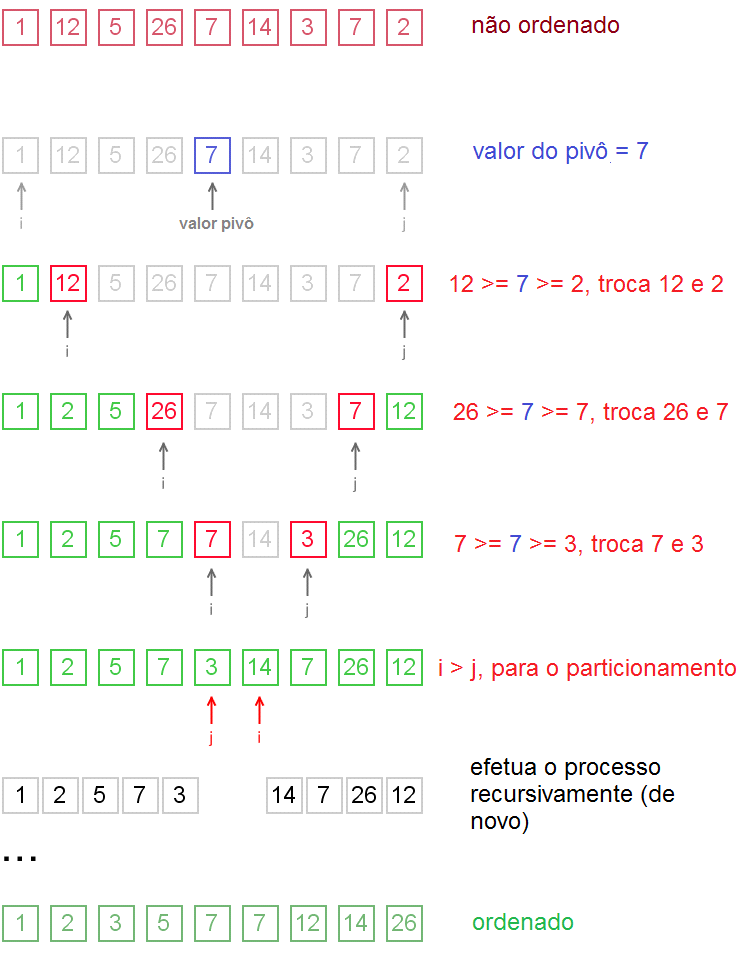
Algoritmo de particionamento em detalhe

Há dois índices, **i** e **j**, e bem no início do algoritmo **i** aponta para o primeiro elemento no vetor e **j** aponta para o último. Em seguida, o algoritmo move o **i** para frente, até um elemento com valor maior ou igual ao pivô é encontrado. O índice **j** é movido para trás, até que um elemento com valor inferior ou igual ao pivô seja encontrado. Se **i<=j** então eles são trocados e **i** passa para a próxima posição (**i+1**), **j** passa para a posição anterior (**j-1**). O algoritmo termina quando **i** se torna maior do que **j**.

Após o particionamento, todos os valores anteriores ao elemento **i** são inferiores ou iguais ao pivô e todos os valores posteriores ao elemento **j** são maiores ou iguais ao pivô.

Exemplo:

Ordenando {1, 12, 5, 26, 7, 14, 3, 7, 2} usando quicksort.



Por que funciona?

No algoritmo de particionamento o vetor é dividido em duas partes e todo elemento **a** da parte esquerda é inferior ou igual a todos os elementos **b** da parte direita. **A** e **b** também satisfazem **a <= pivô <= b**. Após o encerramento do processo de recursão, ambas as partes se tornam ordenadas e, levando em conta todos os argumentos apresentados acima, todo o vetor é ordenado.

Análise da Complexidade

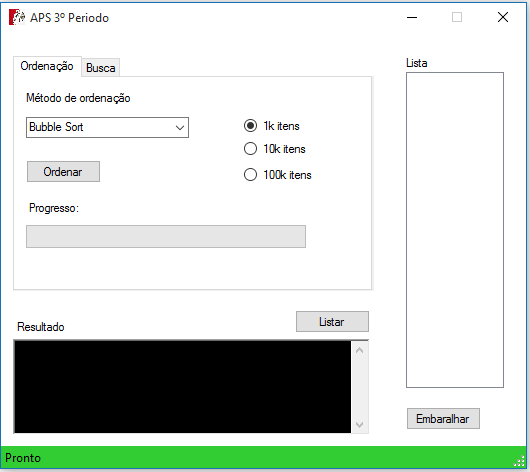
Em média, o quicksort possui O (n log n) de complexidade, mas nos piores casos, o quicksort roda em O (n²) de tempo, mas nos mais "práticos" dados ele funciona bem e superior à outros O (n log n) algoritmos de ordenação.

Observações extras

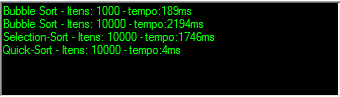
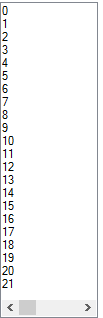
É recomendado que o algoritmo de partição seja usado como uma função separada, por exemplo, a utilização de duas funções, uma de partição e outra de ordenação que utiliza a de partição.

**Desenvolvimento**

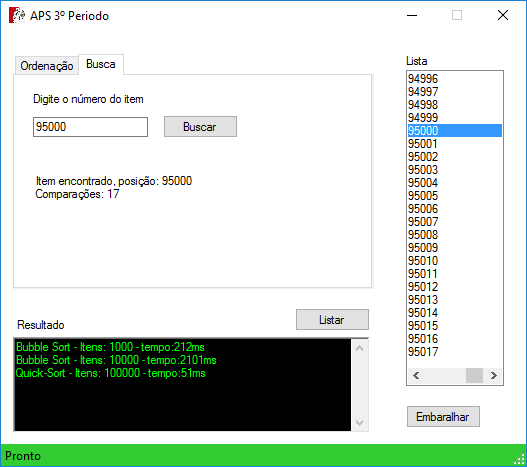
Neste trabalho os dados para ordenamentos são gerados aleatoriamente assim que o programa é iniciado. A partir daí na interface gráfica o usuário pode escolher o tamanho da lista e o método de ordenação que deseja usar.



Ao clicar em ‘Ordenar’ vetor é ordenado, o resultado é mostrado na caixa de resultado e são listados os itens do vetor na listBox:



Na parte de busca o usuário pode realizar a busca binária digitando o item que deseja encontrar, a posição do item na lista ordenada será indicada na listBox



**Resultados e Discussão**

Os resultados que serão apresentados a seguir foram obtidos através de um computador com processador Intel core i5 3.10Ghz x 4.

**Dados Aleatórios**

|  |  |  |  |
| --- | --- | --- | --- |
|  | |  |  |
|  |  |  |  |
| **Tempo de ordenação(ms)** | | | |
|  |  |  |  |
| **Método** | **Mil** | **Dez mil** | **Cem mil** |
| Bubble Sort | 276 | 2163 | 130.922 |
| Selection Sort | 168 | 1736 | 73.065 |
| Quick Sort | 5 | 41 | 51 |

**Dados ordenados**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tempo de ordenação(ms)** | | | |
|  |  |  |  |
| **Método** | **Mil** | **Dez mil** | **Cem mil** |
| Bubble Sort | 164 | 1789 | 68.013 |
| Selection Sort | 167 | 1796 | 71.536 |
| Quick Sort | 11 | 44 | 410 |

**Dados pré-ordenados (Metade ordenada)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Tempo de ordenação(ms)** | | | |
|  |  |  |  |
| **Método** | **Mil** | **Dez mil** | **Cem mil** |
| Bubble Sort | 170 | 1854 | 78.479 |
| Selection Sort | 166 | 1771 | 69.787 |
| Quick Sort | 9 | 1007 | 11.052 |

Através dos testes realizados podemos perceber pelos gráficos acima que o método de ordenação Quick Sort é o mais rápido de todos, enquanto o Bubble Sort é o mais lento, nos casos em que o tamanho foi de 1000 itens, obtivemos resultados próximos para os três métodos testados. Nos casos em que o tamanho do vetor é de 10.000 e os dados já estão ordenados o método Selection Sort e o Bubble Sort tem resultados bastante parecidos e quando o vetor já está ordenando e possui 100.000 itens o método bubble sort é mais rápido que o Selection, levando 68 segundos contra 71 segundos do Selection.

**Considerações Finais**

No trabalho foi apresentado o que são métodos de ordenação, alguns dos mais conhecidos, foi explicado como funcionam, seus desempenhos e foram também aplicados na prática em um programa. Em relação aos desempenhos dos métodos, o quick sort é o algoritmo mais eficiente que existe para uma grande variedade de situações, é recursivo, o que demanda uma pequena quantidade de memória adicional.

**Referências Bibliográficas**

http://csharpbrasil.com.br/algoritmos-de-ordenacao-em-csharp/

https://pt.wikipedia.org/wiki/Algoritmo\_de\_ordenacao

https://msdn.microsoft.com/pt-br/library/system.diagnostics.stopwatch(v=vs.90).aspx

http://www.devmedia.com.br/algoritmos-de-ordenacao-analise-e-comparacao/28261

http://homepages.dcc.ufmg.br/~cunha/teaching/20121/aeds2/complexity.pdf

**Código Fonte**

**Classe de ordenação ‘Ordenação.cs’.**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Diagnostics;

namespace APS3periodo

{

public delegate void Progress(int val);

class Ordenacao

{

public event Progress onProgressUpdate;

int progress = 0;

public Stopwatch sw = new Stopwatch();

#region Bubble Sort

public List<int> bubbleSort(List<int> l)

{

int tamanho = l.Count;

int comparacoes = 0;

int trocas = 0;

progress = 0;

for (int i = tamanho - 1; i >= 1; i--)

{

progress++;

onProgressUpdate(progress);

for (int j = 0; j < i; j++)

{

comparacoes++;

if (l[j] > l[j + 1])

{

int aux = l[j];

l[j] = l[j + 1];

l[j + 1] = aux;

trocas++;

}

}

}

return l;

}

#endregion

#region quick sort

public List<int> QuickSort(List<int> input, int left, int right)

{

if (left < right)

{

int q = Partition(input, left, right);

QuickSort(input, left, q - 1);

QuickSort(input, q + 1, right);

}

return input;

}

private int Partition(List<int> input, int left, int right)

{

int pivot = input[right];

int temp;

int i = left;

for (int j = left; j < right; j++)

{

if (input[j] <= pivot)

{

temp = input[j];

input[j] = input[i];

input[i] = temp;

i++;

}

}

input[right] = input[i];

input[i] = pivot;

return i;

}

#endregion

#region Selection Sort

public List<int> selectionSort(List<int> vetor)

{

int min, aux;

progress = 0;

for (int i = 0; i < vetor.Count - 1; i++)

{

min = i;

progress++;

onProgressUpdate(progress);

for (int j = i + 1; j < vetor.Count; j++)

if (vetor[j] < vetor[min])

min = j;

if (min != i)

{

aux = vetor[min];

vetor[min] = vetor[i];

vetor[i] = aux;

}

}

return vetor;

}

#endregion

}

}

**Classe de busca binária ‘busca.cs’**

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace APS3periodo

{

class Busca

{

static int c = 0;

public int Comparacoes

{

get { return c; }

}

public int BuscaBinaria\_(List<int> l, int chave)

{

int meio;

int inicio = 0;

int fim = l.Capacity - 1;

c = 0;

do

{

c++;

meio = (int)(inicio + fim) / 2;

if (l[meio] == chave)

{

//Retorna a posição do número na seqüencia.

return meio;

}

else if (chave > l[meio])

inicio = meio + 1;

else

fim = meio - 1;

}

while (inicio <= fim);

//Caso o retorno for -1, então o número não existe na sequencia.

return -1;

}

}

}

**Classe do Form**

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

using System.Diagnostics;

namespace APS3periodo

{

public partial class Form1 : Form

{

static int size;//Tamanho da lista

List<int> l1 = new List<int>(size);

bool lista\_ordenada = false;

Ordenacao ord = new Ordenacao();

string tipoOrd = "";

public Form1()

{

InitializeComponent();

}

private void btnOrdenar\_Click(object sender, EventArgs e)

{

statusStrip1.BackColor = Color.Red;

tamanho\_lista();

pBarOrd.Maximum = size;

PreencheLista(l1);

Esable();

toolStripStatusLabel1.Text = "Ordenando items";

switch (cBoxMetOrd.Text)

{

case "Bubble Sort":

tipoOrd = "Bubble Sort";

break;

case "Selection-Sort":

tipoOrd = "Selection-Sort";

break;

case "Quick-Sort":

tipoOrd = "Quick-Sort";

break;

default:

break;

}

bwOrdenacao.RunWorkerAsync();

}

private void Form1\_Load(object sender, EventArgs e)

{

tamanho\_lista();

statusStrip1.BackColor = Color.LimeGreen;

toolStripStatusLabel1.Text = "Preenchendo Lista";

PreencheLista(l1);

cBoxMetOrd.SelectedIndex = 0;

ord.onProgressUpdate += new Progress(progresso); //Inscreve o metodo progresso() no evento onProgressUpdate

pBarOrd.Maximum = size;

}

/// <summary>

/// Lista os itens na OrdList

/// </summary>

/// <param name="l"></param>

void Listar(List<int> l)

{

OrdList.Items.Clear();

foreach (var item in l)

{

OrdList.Items.Add(item);

}

}

/// <summary>

/// Bloqueia ou desbloqueia botões de acão

/// </summary>

void Esable()

{

bool bt1 = btnOrdenar.Enabled;

bool bt2 = btnPreencheLista.Enabled;

bool bt3 = btnListar.Enabled;

if (bt1 && bt2 && bt3)

{

statusStrip1.BackColor = Color.DarkGray;

btnOrdenar.Enabled = false;

btnPreencheLista.Enabled = false;

btnListar.Enabled = false;

}

else

{

statusStrip1.BackColor = Color.LimeGreen;

btnOrdenar.Enabled = true;

btnPreencheLista.Enabled = true;

btnListar.Enabled = true;

}

}

/// <summary>

/// define o tamanho da lista

/// </summary>

void tamanho\_lista()

{

if (rb1000.Checked)

{

size = 1000;

}

else if (rb10k.Checked)

{

size = 10000;

}

else

{

size = 100000;

}

}

/// <summary>

///

/// Preenche o Vetor com numeros aleatórios de 0 a 100.000

/// </summary>

void PreencheLista(List<int> l)

{

lista\_ordenada = false;

Random r = new Random();

int rand = 0;

l1.Clear();

for (int i = 0; i < size; i++)

{

l1.Add(i);

}

//Faz o 'embaralhamento' dos itens

for (int i = 0; i < size; i++)

{

rand = r.Next(size);

int aux = l1[i];

l1[i] = l1[rand];

l1[rand] = aux;

}

toolStripStatusLabel1.Text = "Pronto";

}

private void btnPreencheLista\_Click(object sender, EventArgs e)

{

toolStripStatusLabel1.Text = "Preenchendo Lista";

OrdList.Items.Clear();

PreencheLista(l1);

Listar(l1);

}

/// <summary>

/// Passa valores para a BW.progres\_change que atualiza componentes no form, é disparada com o evento Ordenacao.onProgressUpdate

/// </summary>

/// <param name="val"></param>

void progresso(int val)

{

bwOrdenacao.ReportProgress(val);

}

#region Bworker Ordenação

private void bwOrdenacao\_DoWork(object sender, DoWorkEventArgs e)

{

ord.sw.Start();

switch (tipoOrd)

{

case "Bubble Sort":

ord.bubbleSort(l1);

break;

case "Selection-Sort":

ord.selectionSort(l1);

break;

case "Quick-Sort":

ord.QuickSort(l1, 0, l1.Count - 1);

break;

default:

break;

}

}

private void bwOrdenacao\_ProgressChanged(object sender, ProgressChangedEventArgs e)

{

lblOrdProgress.Text = String.Format("{0}{1}{2}{3}{4}", "Progresso: ", e.ProgressPercentage + 1, " de ", size, " itens ordenados");

if (pBarOrd.Value == size)

{

pBarOrd.Value = 0;

}

pBarOrd.Value = e.ProgressPercentage;

}

private void bwOrdenacao\_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)

{

ord.sw.Stop();

toolStripStatusLabel1.Text = "Pronto";

richTextBox1.Text += tipoOrd + " - Itens: " + size + " - tempo:" + ord.sw.ElapsedMilliseconds + "ms" + "\n";

ord.sw.Reset();

OrdList.Items.Clear();

lista\_ordenada = true;

pBarOrd.Value = 0;

Listar(l1);

Esable();

}

#endregion

private void btnBuscar\_Click(object sender, EventArgs e)

{

if (txtBusca.Text == "")

{

lblResutBusca.Text = "Campo vazio, digite um item a buscar!";

return;

}

if (!lista\_ordenada)

{

lblResutBusca.Text = "Lista não está ordenada, não é possivel realizar busca binária!";

return;

}

Busca b1 = new Busca();

int key = Convert.ToInt32(txtBusca.Text);

int pos = b1.BuscaBinaria\_(l1, key);

if (pos == -1)

lblResutBusca.Text = "Item não encontrado, tente outro item";

else

lblResutBusca.Text = "Item encontrado, posição: " + pos + "\n" +

"Comparações: " + b1.Comparacoes;

OrdList.SelectedIndex = pos;

}

private void btnListar\_Click(object sender, EventArgs e)

{

Listar(l1);

}

}

}