**CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA**

**FACULDADE DE TECNOLOGIA DE CARAPICUÍBA**

**CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS**

**PROJETO 1**

**ALGORÍTMOS ELEMENTARES DE ORDENAÇÃO**

Prof.ª ANDREIA CRISTINA GRISOLIO MACHION

1430481723057 - Angela Vieira Xaves

1430481923024 - Bruno de Oliveira Lopes

1430482021033 - Natalia Aparecida Silva de Souza

1430481823045 – Renan da Silva Ferreira

**CARAPICUÍBA**

**2021**

**Algoritmo de Ordenação**

import java.util.Random;

import java.util.ArrayList;

public class Ordenacao {

private int [] vetor;

public Ordenacao(long num\_elems){

vetor = new int[(int)num\_elems];

preencherVetor();

}

public boolean isSorted(){

for (int i = 1; i < vetor.length; i++){

if (vetor[i-1] > vetor[i]){

return false;

}

}

return true;

}

public void preencherVetor(){

Random gerador = new Random();

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

vetor[i] = gerador.nextInt(10);

}

}

public void bubbleSort(){

int tmp;

for (int i = vetor.length-1;i > 1; i--){

/\* i decresce pois a cada iteracao, uma posicao a mais no final ja esta

perfeitamente ordenada \*/

for (int j = 1; j < i+1; j++){

/\* inverte a posicao de quaisquer dois elementos adjacentes que nao

estejam ordenados entre si. Isso resulta que o maior elemento da parte

ainda nao ordenada do vetor seja levado ate a parte ordenada do vetor,

na sua respectiva posicao final \*/

if (vetor[j-1] > vetor[j]){

tmp = vetor[j-1];

vetor[j-1] = vetor[j];

vetor[j] = tmp;

}

}

}

}

public void insertionSort(){

int copia, j;

for (int i = 1; i < vetor.length; ++i){

/\* i representa ate onde vai, no momento, o "novo vetor" (eh o mesmo

vetor mas uma regiao ja processada sera considerada separadamente da

ainda nao processada)\*/

copia = vetor[i]; // vai inserir o elemento da posicao i no "novo vetor"

for (j = i-1; j >= 0 && vetor[j] > copia; --j){

/\* ate encontrar a posicao correta para vetor[i], vai abrindo espaco,

avancando uma posicao cada elemento\*/

vetor[j+1] = vetor[j];

}

// posicao j+1 eh a correta para o elemento vetor[i] que foi copiado antes

vetor[j+1] = copia;

}

}

public void selectionSort(){

int minimo, tmp;

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

/\* a cada iteracao, nao olha mais uma posicao a esquerda do vetor

(todas as iguais ou menores que o valor atual de i)\*/

minimo = i;

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

// seleciona o menor valor ainda nao selecionado do vetor

if (vetor[j] < vetor[minimo]) {

minimo = j;

}

}

/\* Insere o valor selecionado, trocando a posicao com o que estava anteriormente la \*/

tmp = vetor[i];

vetor[i] = vetor[minimo];

vetor[minimo] = tmp;

}

}

public int [] getVetor(){

return this.vetor;

}

public static void main(String [] argv){

Benchmark bench;

long comeco, fim, tamanho;

final long TAMANHO = 100000;

tamanho = TAMANHO;

System.out.println("BubbleSort:");

for (int i = 0; i < 4; i++){

Ordenacao o = new Ordenacao(tamanho);

bench = new Benchmark();

for (int j = 0; j < 5; j++){

comeco = System.nanoTime() / 1000000;

o.bubbleSort();

fim = System.nanoTime()/ 1000000;

bench.inserirTempo(fim-comeco);

o.preencherVetor();

}

System.out.println(tamanho);

bench.printResultados();

System.out.println();

tamanho \*= 2;

}

System.out.println("\n");

tamanho = TAMANHO;

System.out.println("InsertionSort:");

for (int i = 0; i < 4; i++){

Ordenacao o = new Ordenacao(tamanho);

bench = new Benchmark();

for (int j = 0; j < 5; j++){

comeco = System.nanoTime()/ 1000000;

o.insertionSort();

fim = System.nanoTime()/ 1000000;

bench.inserirTempo(fim-comeco);

o.preencherVetor();

}

System.out.println(tamanho);

bench.printResultados();

System.out.println();

tamanho \*= 2;

}

System.out.println("\n");

tamanho = TAMANHO;

System.out.println("SelectionSort:");

for (int i = 0; i < 4; i++){

Ordenacao o = new Ordenacao(tamanho);

bench = new Benchmark();

for (int j = 0; j < 5; j++){

comeco = System.nanoTime()/ 1000000;

o.selectionSort();

fim = System.nanoTime()/ 1000000;

bench.inserirTempo(fim-comeco);

o.preencherVetor();

}

System.out.println(tamanho);

bench.printResultados();

System.out.println();

tamanho \*= 2;

}

System.out.println("\n");

}

}

class Benchmark{

private ArrayList<Long> resultados;

private long media;

public Benchmark(){

resultados = new ArrayList<Long>();

media = 0;

}

public void inserirTempo(long tempo){

resultados.add(tempo);

calcMedia();

}

public void calcMedia(){

long soma = 0;

for (Long i : resultados){

soma += i;

}

media = soma / resultados.size();

}

public void printResultados(){

for (Long i : resultados){

System.out.print("" + i + "\t");

}

System.out.println();

System.out.println(media);

}

public long getMedia(){

return media;

}

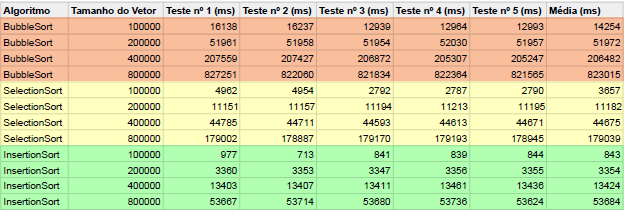
public ArrayList<Long> getResultados(){

return resultados;

}

}

**Tabela de Testes realizados**



**Conclusão:**

Através do experimento foi possível concluir que há uma diferença claríssima em desempenho entre os algoritmos, em ordem do menos para o mais eficiente, BubbleSort, SelectionSort e InsertionSort. Em nossa implementação em Java o SelectionSort se mostrou, em média, 4,4 vezes mais rápido que o BubbleSort, e o InsertionSort 3,6 vezes mais eficiente que o SelectionSort e em relação ao BubbleSort, 15,9 vezes mais eficiente. A ineficiência do BubbleSort deve-se principalmente ao grande número de comparações e trocas que o algoritmo faz na sua execução, bem maior que para os outros algoritmos.

Mesmo havendo essa diferença em nível de desempenho entre os algoritmos, em relação ao crescimento do conjunto de dados todos apresentaram um comportamento quadrático: a cada vez que dobrava o tamanho do vetor a ser ordenado o tempo gasto pelos algoritmos aumentou, em média, 3,9 vezes para o BubbleSort, 3,7 vezes para o SelectionSort e 4 vezes para o InsertionSort.

Em conclusão, mesmo entre algoritmos de uma mesma complexidade assintótica pode haver fatores que resultam em grandes diferenças de tempo de execução.

Fonte: https://www.ime.usp.br/~pf/algoritmos/index.html.