

**Computabilidade & Complexidade de Algoritmos**

**Algoritmos de Ordenação**

**Dr. Ivan Carlos Alcântara de Oliveira**

**São Paulo**

**2018**



**1609506-5 Daniel Alves Teixeira**

**1612369-7 Bruno Soares Coello Rodriguez**

**1619803-4 Matheus Veloso Tavares**

**1623757-9 Lucas Siqueira Machado**

**1628247-7 William José Silva Dos Santos**

**1652167-6 Marcus Vinicius Silva Dos Santos**

**São Paulo**

**2018**

**Sumário**

3.Descrição .............................................................................. 4

2.Tabela O-Grande................................................................... 5

3. Código Fonte ....................................................................... 6

4. Primeiro Experimento ........................................................ 25

5. Segundo Experimento......................................................... 29

6. Considerações Finais........................................................... 32

**Descrição**

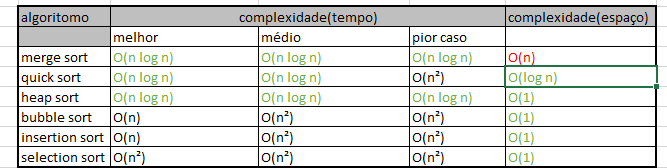
As simulações a serem realizadas neste trabalho envolvem o assunto de algoritmos de ordenação.

O sistema permite utilizar métodos de ordenação de conjuntos aleatórios, ordem crescente e decrescente, permitindo a comparação dos mesmos para saber qual melhor tempo x método. Dentre os métodos existentes utilizamos os seguintes:

* *Método Bolha*
* *Método Seleção Direta*
* *Método Inserção Direta*
* *Método MergeSort*
* *Método HeapSort*
* *Método QuickSort*

No início dos tempos a aplicação foi desenvolvida em Java, com os métodos citados acima, cada algoritmo tem seu determinado tempo de complexidade.

**TABELA O-GRANDE**



**Código Fonte**

import java.lang.\*;

import java.io.\*;

import java.util.\*;

import javax.swing.\*;

public class OrdenaCompleto implements Serializable {

private int conj[];

private Integer qtdeNumeros;

public int getQtdeNumeros(){

return qtdeNumeros.intValue();

}

private static String montaMenu(){

String menu = "\n1 - Gera Números Aleatórios\n2 - Gera Números em Ordem Crescente\n3 - Gera Números em OrdEm Decrescente\n4 - Ordena Bolha\n5 - Ordena Seleção Direta\n6 - Ordena Insercao Direta\n7 - Ordena MergeSort\n8 - Ordena HeapSort\n9 - Ordena QuickSort\n10 - Ordena Todos Metodos \n11 - finaliza\n\nopcao:";

return menu;

}

public void geraArquivo(){

try{

String pathName = JOptionPane.showInputDialog("Digite o nome do Arquivo: " );

File f = new File(pathName);

if (!f.exists()){

f.createNewFile();

}

OutputStream out = new FileOutputStream(f);

// associar ao arquivo o ObjectOutputStream

ObjectOutputStream s = new ObjectOutputStream(out);

// Armazena o total de números aleatórios gerados

qtdeNumeros

= new Integer(Integer.parseInt(JOptionPane.showInputDialog("Digite a Quantidade de Números Aleatórios:" )));

s.writeObject(qtdeNumeros);

// Cria um gerador de números randômico

Random numero = new Random();

// serializar e transferir o objeto

Integer n;

for(int i = 0; i < qtdeNumeros.intValue(); i++){

n = new Integer(numero.nextInt());

System.out.println("gera " + n.intValue());

s.writeObject(n);

}

}catch(Exception e){

System.out.println("Erro na Criação do Arquivo");

}

}

public void geraArquivoOrdenadoCrescente(){

try{

String pathName = JOptionPane.showInputDialog("Digite o nome do Arquivo: " );

File f = new File(pathName);

if (!f.exists()){

f.createNewFile();

}

OutputStream out = new FileOutputStream(f);

// associar ao arquivo o ObjectOutputStream

ObjectOutputStream s = new ObjectOutputStream(out);

// Armazena o total de números aleatórios gerados

qtdeNumeros

= new Integer(Integer.parseInt(JOptionPane.showInputDialog("Digite a Quantidade de Números Aleatórios:" )));

s.writeObject(qtdeNumeros);

// serializar e transferir o objeto

Integer n;

for(int i = 0; i < qtdeNumeros.intValue(); i++){

n = new Integer(i);

System.out.println("gera " + n.intValue());

s.writeObject(n);

}

}catch(Exception e){

System.out.println("Erro na Criação do Arquivo");

}

}

public void geraArquivoOrdenadoDecrescente(){

try{

String pathName = JOptionPane.showInputDialog("Digite o nome do Arquivo: " );

File f = new File(pathName);

if (!f.exists()){

f.createNewFile();

}

OutputStream out = new FileOutputStream(f);

// associar ao arquivo o ObjectOutputStream

ObjectOutputStream s = new ObjectOutputStream(out);

// Armazena o total de números aleatórios gerados

qtdeNumeros

= new Integer(Integer.parseInt(JOptionPane.showInputDialog("Digite a Quantidade de Números Aleatórios:" )));

s.writeObject(qtdeNumeros);

// serializar e transferir o objeto

Integer n;

for(int i = qtdeNumeros.intValue(); i > 0; i--){

n = new Integer(i);

System.out.println("gera " + n.intValue());

s.writeObject(n);

}

}catch(Exception e){

System.out.println("Erro na Criação do Arquivo");

}

}

public void leArquivo(){

try{

String pathName = JOptionPane.showInputDialog("Digite o nome do Arquivo: " );

File f = new File(pathName);

if (f.exists()){

InputStream in = new FileInputStream(f);

// associar ao arquivo o ObjectInputStream

ObjectInputStream s = new ObjectInputStream(in);

// Armazena o total de números aleatórios gerados

qtdeNumeros = (Integer) s.readObject();

conj = new int[qtdeNumeros.intValue()];

// serializar e transferir o objeto

for(int i = 0; i < qtdeNumeros.intValue(); i++){

Integer n = (Integer) s.readObject();

conj[i] = n.intValue();

System.out.println("valor " + n.intValue());

}

}

}catch(Exception e){

System.out.println("Erro na leitura do Arquivo");

}

}

public void bolha(int conjunto[], int qtde){

int aux,i,j;

for (i=0;i<qtde-1;i++)

for (j = 0; j < qtde - i-1; j++)

if ( conjunto[j] > conjunto[j+1] ){

// permuta conjunto [j] com conjunto [j+1]

aux = conjunto[j];

conjunto[j] = conjunto[j+1];

conjunto [j+1] = aux;

}

}

public void bolha(){

bolha(conj, qtdeNumeros.intValue());

}

public void selecao(){

selecao(conj, qtdeNumeros.intValue());

}

public void selecao(int conjunto[], int qtde)

{

int i,indConjunto,j,valor;

for (i = qtde -1; i > 0; i--){

valor = conjunto[0];

indConjunto = 0;

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

if ( conjunto[j] > valor){

valor = conjunto[j];

indConjunto = j;

}

}

conjunto[ indConjunto ] = conjunto [ i ];

conjunto[ i ] = valor;

}

}

public void intercala(int p,int q,int r, int conjunto[]){

int i, j, k;

int w[]= new int[r-p];

i = p; j = q; k = 0;

while (i < q && j < r) {

if (conjunto[i] <= conjunto[j]) w[k++] = conjunto[i++];

else w[k++] = conjunto[j++];

}

while (i < q) w[k++] = conjunto[i++];

for (i = p; i < j; ++i) conjunto[i] = w[i-p];

}

public void mergeSort(){

mergeSort(conj,qtdeNumeros.intValue());

}

public void mergeSort( int conjunto[],int qtde){

int p, r;

int b = 1;

while (b < qtde){

p = 0;

while (p + b < qtde){

r = p + 2\*b;

if (r > qtde) r = qtde;

intercala(p, p+b, r, conjunto);

p = p + 2\*b;

}

b= 2\*b;

}

}

public void peneira(int p,int m,int conjunto[]){

int x = conjunto[p];

while (2\*p <= m){

int f = 2\*p;

if (f < m && conjunto[f] < conjunto[f+1]) ++f;

if (x >= conjunto[f]) break;

conjunto[p] = conjunto[f];

p = f;

}

conjunto[p] = x;

}

public void heapSort(){

int aux[] = new int [qtdeNumeros.intValue()+1];

// Fazendo uma cópia deslocada

for (int i = 0; i < qtdeNumeros.intValue(); i++) aux [i+1] = conj [i];

heapSort(aux, qtdeNumeros.intValue());

// Retornando ao vetor original

for (int i = 0; i < qtdeNumeros.intValue(); i++) conj [i] = aux [i+1];

// Verifica se vetor esta realmente ordenado

int i;

for ( i = 0; i < qtdeNumeros.intValue()-1; i++)

if (!(conj[i] <= conj[i+1])) break;

System.out.println("Indice: " + i);

}

public void heapSort(int conjunto[],int qtde){

int p, m, x;

for (p = qtde/2; p >= 1; --p)

peneira (p, qtde, conjunto);

for (m = qtde; m >= 2; --m) {

x = conjunto[1];

conjunto[1] = conjunto[m];

conjunto[m] = x;

peneira (1, m-1, conjunto);

}

}

public void quickSort( int conjunto[ ], int inicio, int fim){

int pivo, trab, i, j;

i = inicio;

j = fim;

pivo = conjunto [ (i + j) / 2 ];

do{

while (conjunto [i] < pivo) i++;

while (conjunto [j] > pivo) j--;

if (i <= j) {

trab = conjunto[i];

conjunto[i] = conjunto[j];

conjunto[j] = trab;

i = i + 1;

j = j - 1;

}

} while ( i <= j );

if (inicio < j) quickSort( conjunto, inicio, j);

if (fim > i ) quickSort ( conjunto, i, fim);

}

public void quickSort(){

quickSort(conj,0, qtdeNumeros.intValue()-1);

}

public void insercaoDireta( int conjunto[], int n ){

int j, // Percorre o segmento desordenado

i, // Percorre o segmento ordenado

chave; // Elemento a ser inserido no conjunto ordenado

for(i=1; i< n; i++){

chave = conjunto[i];

j = i-1;

// Insere Conjunto[j] na seq. Ord. conjunto[0..j-1]

while ((j>=0) && (conjunto[j]>chave)){

conjunto[j+1] = conjunto[j];

j--;

}

conjunto[j+1] = chave;

}

}

public void insercaoDireta(){

insercaoDireta(conj, qtdeNumeros.intValue());

}

public static void main(String args[]){

OrdenaCompleto o = new OrdenaCompleto();

long antes, depois;

int op = 0;

do {

switch( Integer.parseInt(JOptionPane.showInputDialog(montaMenu()))){

// 1 - Gera números aleatórios

case 1: o.geraArquivo(); break;

// 2 - Gera números aleatórios em Ordem Crescente

case 2: o.geraArquivoOrdenadoCrescente(); break;

// 3 - Gera números aleatórios em Ordem Decrescente

case 3: o.geraArquivoOrdenadoDecrescente(); break;

// 4 - Ordena pelo método da bolha

case 4: o.leArquivo();

antes = System.currentTimeMillis();

o.bolha();

depois = System.currentTimeMillis();

JOptionPane.showMessageDialog( null, "tempo de execução para o conjunto de "+ o.getQtdeNumeros()+ " dados: " + (depois-antes) + " milisegundos");

break;

// 5 - Ordena pelo método da selecao direta

case 5: o.leArquivo();

antes = System.currentTimeMillis();

o.selecao();

depois = System.currentTimeMillis();

JOptionPane.showMessageDialog( null, "tempo de execução para o conjunto de "+ o.getQtdeNumeros()+ " dados: " + (depois-antes) + " milisegundos");

break;

// 6 - Ordena pelo método da inserção direta

case 6: o.leArquivo();

antes = System.currentTimeMillis();

o.insercaoDireta();

depois = System.currentTimeMillis();

JOptionPane.showMessageDialog( null, "tempo de execução para o conjunto de "+ o.getQtdeNumeros()+ " dados: " + (depois-antes) + " milisegundos");

break;

// 7 - Ordena pelo método MergeSort

case 7: o.leArquivo();

antes = System.currentTimeMillis();

o.mergeSort();

depois = System.currentTimeMillis();

JOptionPane.showMessageDialog( null, "tempo de execução para o conjunto de "+ o.getQtdeNumeros()+ " dados: " + (depois-antes) + " milisegundos");

break;

// 8 - Ordena pelo método HeapSort

case 8: o.leArquivo();

antes = System.currentTimeMillis();

o.heapSort();

depois = System.currentTimeMillis();

JOptionPane.showMessageDialog( null, "tempo de execução para o conjunto de "+ o.getQtdeNumeros()+ " dados: " + (depois-antes) + " milisegundos");

break;

// 9 - Ordena pelo método Quicksort

case 9: o.leArquivo();

antes = System.currentTimeMillis();

o.quickSort();

depois = System.currentTimeMillis();

JOptionPane.showMessageDialog( null, "tempo de execução para o conjunto de "+ o.getQtdeNumeros()+ " dados: " + (depois-antes) + " milisegundos");

break;

//10 - Ordena todos

case 10:

o.leArquivo();

String texto = "";

//Bolha

antes = System.currentTimeMillis();

o.bolha();

depois = System.currentTimeMillis();

texto += "bolha "+ o.getQtdeNumeros()+ " dados: " + (depois-antes) + " milisegundos\n";

//selecao

antes = System.currentTimeMillis();

o.selecao();

depois = System.currentTimeMillis();

texto += "selecao "+ o.getQtdeNumeros()+ " dados: " + (depois-antes) + " milisegundos\n";

//insercaoDireta

antes = System.currentTimeMillis();

o.insercaoDireta();

depois = System.currentTimeMillis();

texto += "insercaoDireta "+ o.getQtdeNumeros()+ " dados: " + (depois-antes) + " milisegundos\n";

//mergeSort

antes = System.currentTimeMillis();

o.mergeSort();

depois = System.currentTimeMillis();

texto += "mergeSort "+ o.getQtdeNumeros()+ " dados: " + (depois-antes) + " milisegundos\n";

//heapSort

antes = System.currentTimeMillis();

o.heapSort();

depois = System.currentTimeMillis();

texto += "heapSort "+ o.getQtdeNumeros()+ " dados: " + (depois-antes) + " milisegundos\n";

//quickSort

antes = System.currentTimeMillis();

o.quickSort();

depois = System.currentTimeMillis();

texto += "quickSort "+ o.getQtdeNumeros()+ " dados: " + (depois-antes) + " milisegundos\n";

JOptionPane.showMessageDialog( null, texto);

break;

default: op = 11;

}

}while ( op != 11);

System.exit(0);

}}

**1º Experimento**

Informações do hardware:

Processador : Intel Core i5 8250U 1.8ghz   
Memoria: 8GB GDDR4  
HD: 1000GB  
Placa de Vídeo: Nvidia MX110 2GB

Sistema Operacional:Windows 10

**Análise crítica/discussão dos resultados**

Considerando a Comparação do Conjunto de 1000000 ( 1 Milhão de Elementos)

Analisando os gráficos obstem-se que utilizando o método da bolha, em ordem decrescente obtivemos o resultado mais rápido (316621 milissegundos) em relação ao conjunto de ordem crescente (640016 milissegundos) e o aleatório (3404146 milissegundos).

Analisando os gráficos obstem-se que utilizando o método heap sort, em ordem decrescente (78 milissegundos) e crescente (78 milissegundos) se obteve o mesmo tempo, porem os mesmos foram menores do que o aleatório (85 milissegundos).

Analisando os gráficos obstem-se que utilizando o método inserção direta, em todos métodos obteve o mesmo tempo (0 milissegundos).

Analisando os gráficos obstem-se que utilizando o método mergeSort, em ordem crescente (62) foi o mais rápido comparado aos outros conjuntos aleatórios (109) e descrente (78).

Analisando os gráficos obstem-se que utilizando o método quick sort em ordem decrescente (0) foi mais rápido que os conjuntos crescentes (16) e aleatório (14).

Analisando os gráficos obstem-se que utilizando o método seleção, em ordem decrescente (166345 milissegundos) foi o mais eficiente em relação aos demais conjuntos onde no crescente (445874) e no aleatório (446493).

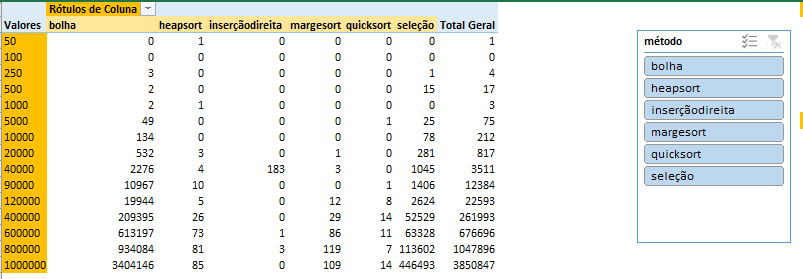


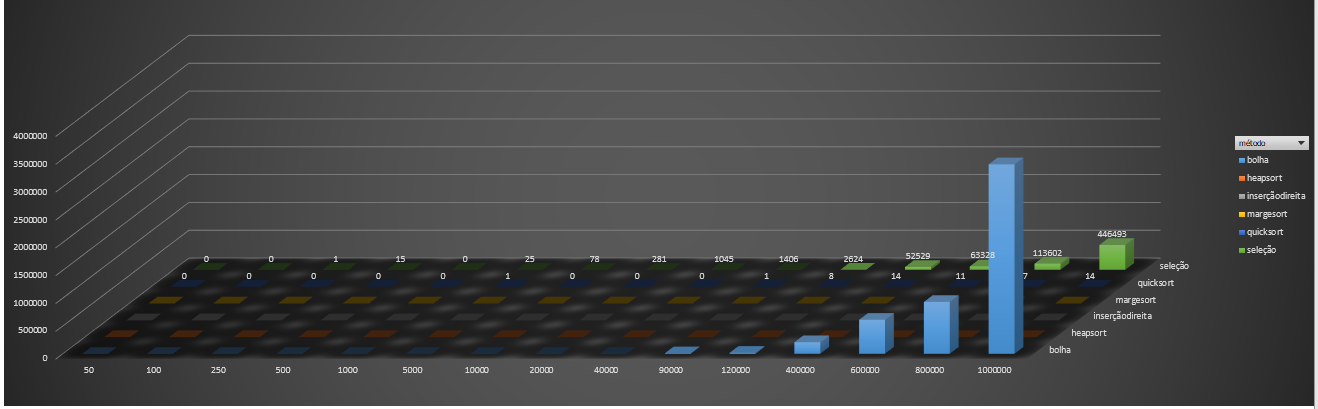
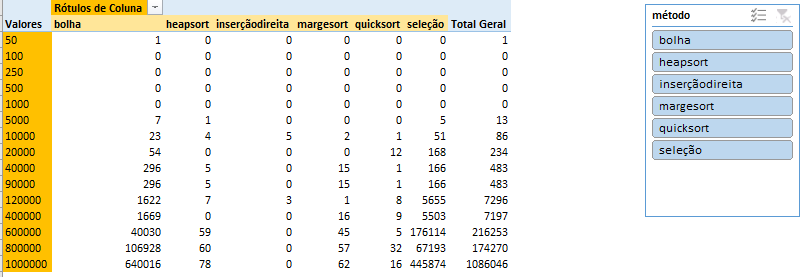
Figura 1Tabela: Ordenação Aleatória

Figura 2Gráfico: Ordenação Aleatório



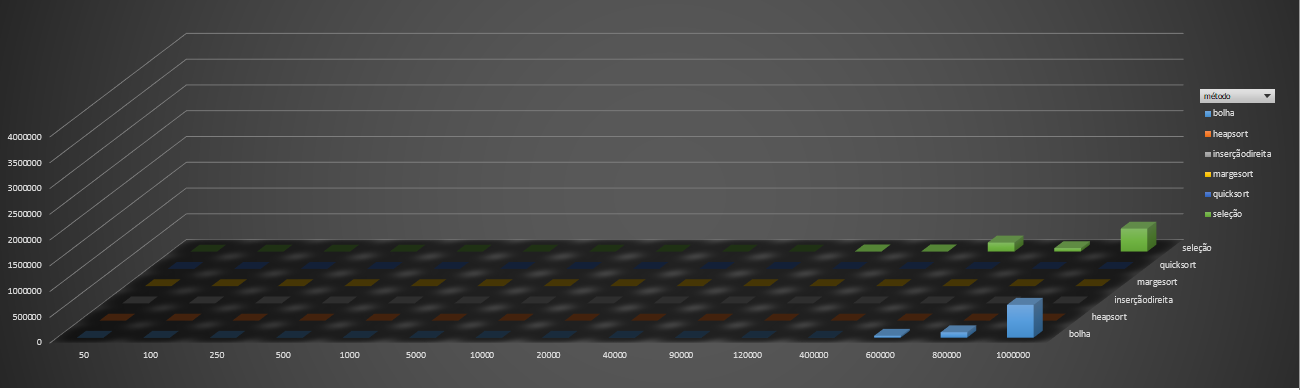
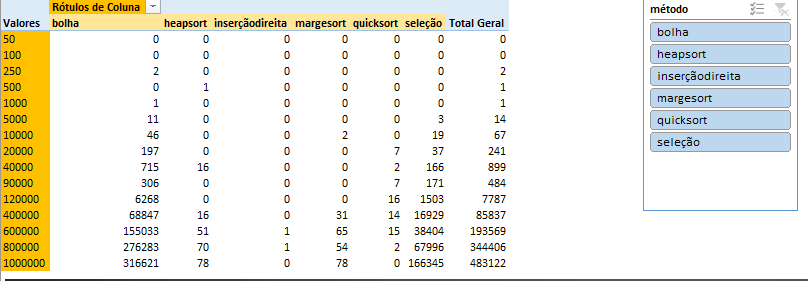
Figura 3Tabela: Ordem Crescente

Figura 4Gráfico: Ordem Crescente



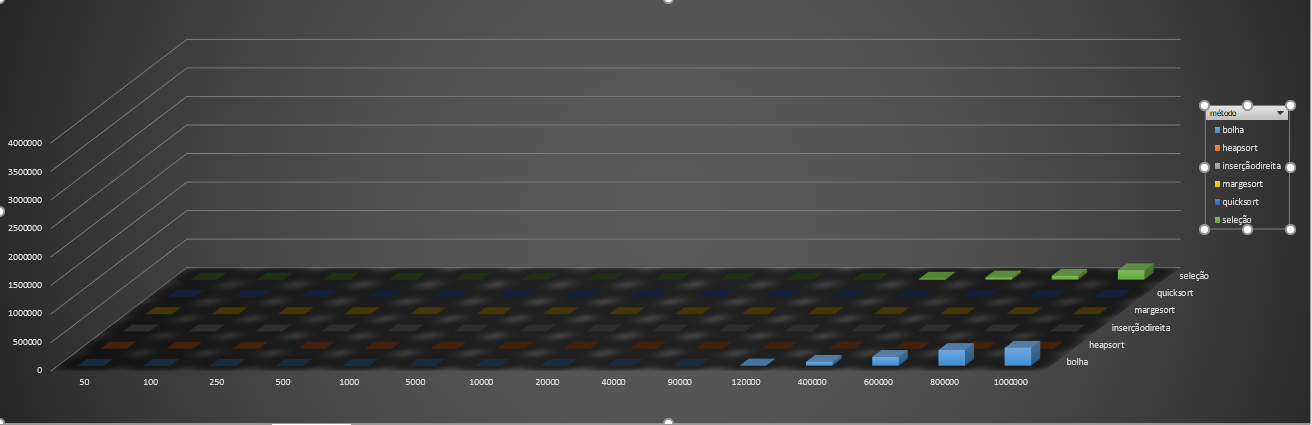
Figura 5Tabela: Ordem Decrescente

Figura 6Gráfico: Ordem Decrescente

**2º Experimento**

Informações do Hardware:  
Processador: Pentium n3700

Memoria: 4GB GDDR3

HD: 500GB  
Windows 8.1

**Análise crítica/discussão dos resultados**

Considerando a Comparação do Conjunto de 1000000 ( 1 Milhão de Elementos)

Analisando os gráficos obstem-se que utilizando o método da bolha;

Ordem crescente (614653) foi o mais eficiente em relação ao conjunto decrescente (1241494), porem o de pior caso foi o aleatório (2027957).

Analisando os gráficos obstem-se que utilizando o método heap sort;

O conjunto aleatório teve uma eficiência maior (219) em seguida ordem decrescente (235) e o pior crescente (250).

Analisando os gráficos obstem-se que utilizando o método inserção direta.

Em todos conjuntos obtivemos mesmo tempo de execução (0)

Analisando os gráficos obstem-se que utilizando o método mergeSort;

O mais eficiente é a ordem decrescente (187), após o aleatório (188) e então o crescente (219).

Analisando os gráficos obstem-se que utilizando o método quick sort;

O mais eficiente foi o crescente (46), em seguida o decrescente (47) e o pior caso o aleatório (62).

Analisando os gráficos obstem-se que utilizando o método seleção;

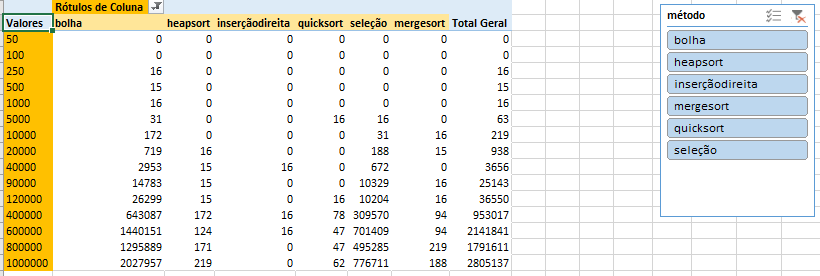
O mais eficiente foi em ordem decrescente (522819) em seguida a ordem crescente com (527850) e o pior caso foi o aleatório (776711).

Figura 7Tabela: Ordenação Aleatória

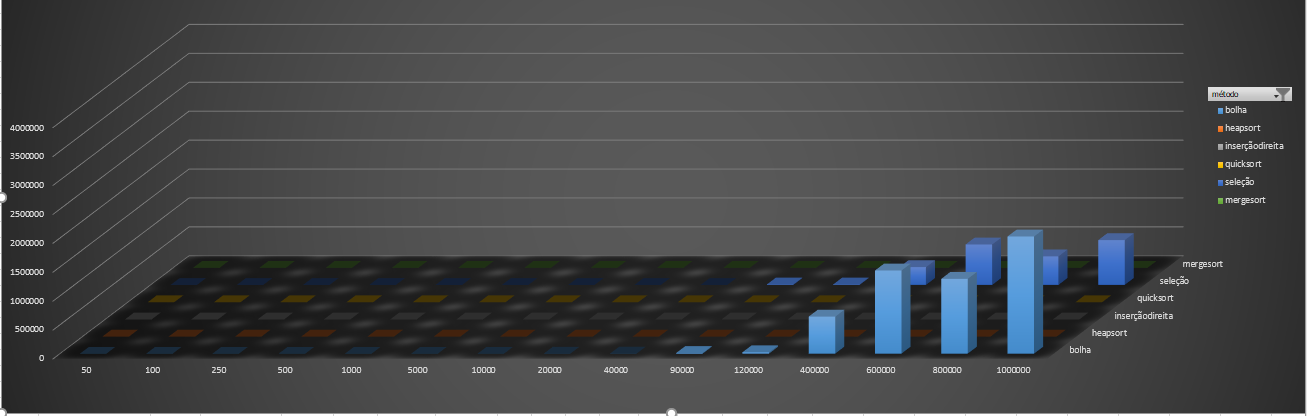


Figura 8Gráfico: Ordenação Aleatória

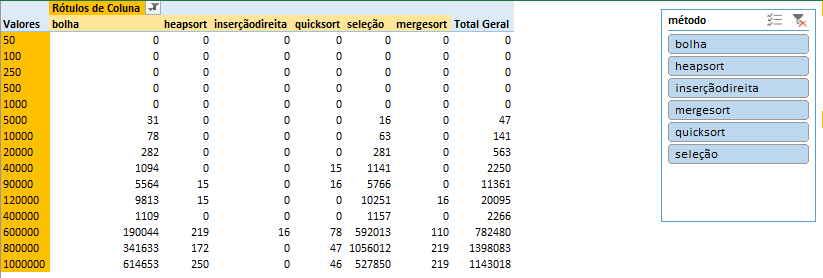


Figura 9Tabela: Ordem Crescente

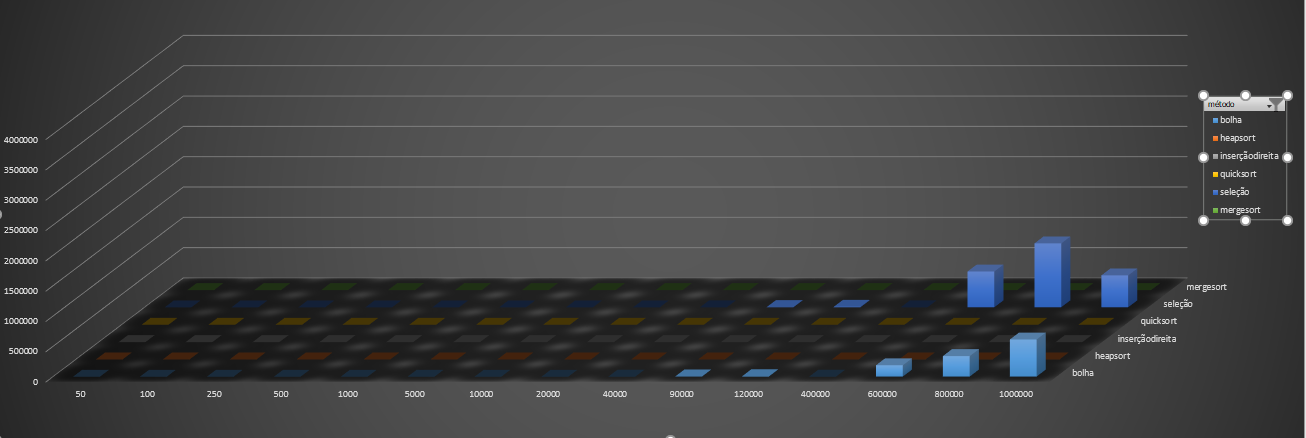


Figura 10Gráfico: Ordem Crescente

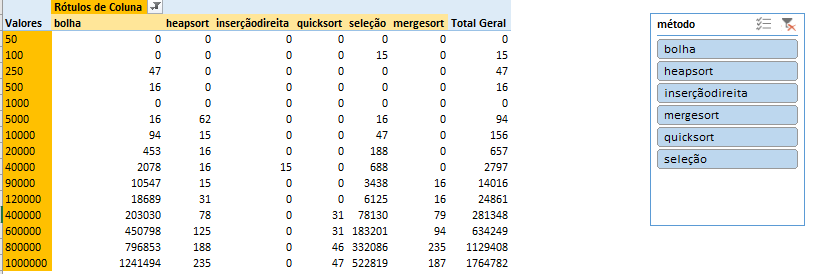


Figura 11Tabela: Ordem Decrescente

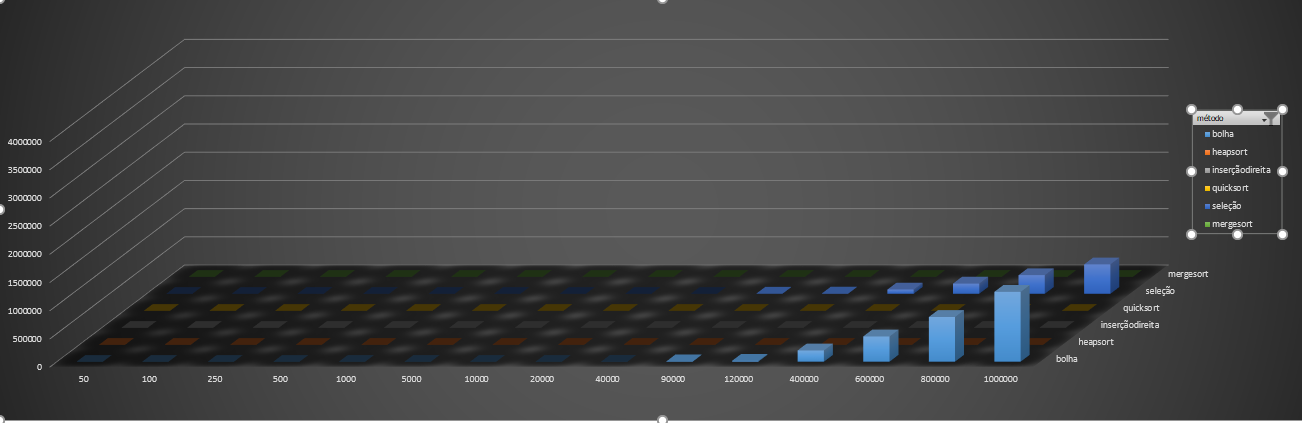


Figura 12Gráfico: Ordem Decrescente

**Considerações Finais**

Com base nos experimentos realizados, conclui-se que nem sempre um computador com o hardware de configuração superior, a um computador de configuração inferior seja mais eficiente, ressaltando o Método de ordenação em que seja Utilizado. Portanto levando em conta a complexidade de cada algoritmo e seu respectivo T(n), foi possível notar uma determinada diferença em crescimento do tempo de execução de mesmos métodos em diferentes configurações de hardware, entretanto houve uma variação significativa de aumento em tempo, em diferentes tamanhos de conjuntos, no mesmo método, porém em computadores de configurações diferentes. Mas no geral percebemos que nos conjuntos de 1000000 (1 Milhão de Elementos), a diferença foi muito notável, por exemplo no primeiro experimento obtivemos o tempo de (14 milissegundos) utilizando o Método QuickSort e no segundo experimento, em um computador de hardware inferior ao do primeiro experimento utilizando também o método QuickSort obtivemos ( 62 milissegundos ).