AED 1 - Relação Algoritmos de Ordenação

Bruno Tomé[1], **Matheus Calixto**[1]

[1]Instituto Federal de Minas Gerais (IFMG)
São Luiz Gonzaga, s/nº - Formiga / MG - Brasil

ibrunotome@gmail.com, calixtinn@gmail.com

Abstract. *This report is about the relationship between sorting algorithms, showing the time, the number of iterations and comparisons of the best known sorting algorithms.*

Resumo. *Este relatório é sobre a relação entre os algoritmos de ordenação, mostrando o tempo, o número de comparações e iterações dos algoritmos mais conhecidos da ordenação.*

1. Introdução

A proposta de realização deste trabalho é a análise e comparação de tempo, número de iterações e comparações entre 11 algoritmos de ordenação: BubbleSort, SelectionSort, InsertionSort, ShellSort, HeapSort, o IntroSort (Híbrido do QuickSort e InsertionSort) e mais cinco tipos diferentes de QuickSort, alternando a forma de como o elemento pivô é escolhido: Pivô primeiro elemento, pivô último elemento, pivô mediana, pivô randômico e pivô meio.

2. Implementação

2.1. Descrição sobre as decisões de projeto e implementação do programa

A implementação foi feita em dupla, nenhum dos integrantes ficou responsável por uma parte específica, ambos usaram a combinação Pages (ferramenta de edição de texto, com opção de múltiplos editores em tempo real na nuvem) + Skype para codificarmos o algoritmo.

2.2. Descrição das estruturas de dados usadas no programa

Optamos pela utilização de um vetor dinâmico para as massas de entrada, evitando que vários vetores fossem alocados na memória. No corpo principal do algoritmo utilizamos um case, o usuário do programa escolhe a massa de entrada através dele e quando escolhido, um for de 1 a 100 irá chamar as respectivas funções dos 11 algoritmos de ordenação 100 vezes, retornando o tempo médio e número de iterações médias de cada tipo de ordenação.

2.3. Funcionamento das principais funções e procedimentos utilizados

Utilizamos as funções de tempo já implementadas pelo professor Mário e pelo aluno Sávio Cardoso.

Basicamente a maioria das funções implementadas tem o mesmo propósito, receber como parâmetro um vetor dinâmico, o tamanho desse vetor, um contador de iterações e retornar

na própria função o tempo, retornar por referência o contador de iterações e no programa principal é feita a soma das 100 vezes que essa operação é feita, logo depois o resultado é dividido por 100 para chegarmos a média.

A função gerar gera o vetor dinâmico para cada massa de entrada, as funções particao, ordena, particao2, ordena2, particao3, ordena3, particao4, ordena4, particao5, ordena5 são dedicadas aos QuickSort: Meio, primeiro elemento, último elemento, randômico e mediana respectivamente.

2.4. Formato de entrada e saída dos dados

A entrada é feita pelo teclado a partir de um case escolhendo a massa de entrada. A saída é impressa na tela, e os dados são tabelados manualmente.

2.5. Como executar o programa

No terminal com o arquivo do código fonte no Desktop digite o seguinte código:

cd Desktop

fpc aed.pas -oaed.bin

./aed.bin

Depois dessa parte, aparecerá um menu com as 10 opções de massa de entrada, basta escolher uma e o programa rodará 100 vezes para cada um dos métodos de ordenação.

3. Algoritmos de Ordenação

3.1. BubbleSort

O BubbleSort é um algoritmo que percorre o vetor inteiro comparando elementos adjacentes (dois a dois). Os elementos que estão fora de ordem são trocados. O resultado da ordenação se dá repetindo os dois passos acima com os primeiros n-1 elementos, depois com os primeiros n-2 elementos, até que reste apenas um elemento.

Exemplo:

1º = 2 4 1 6 5 3

2º = 2 1 4 6 5 3

3º = 2 1 4 5 6 3

4º = 2 1 4 5 3 6

5º = 1 2 4 5 3 6

6º = 1 2 4 3 5 6

7º = 1 2 3 4 5 6

Complexidade: sempre $O(n^2)$, no melhor, pior e também no caso médio.

3.2. SelectionSort

O SelectionSort é um dos algoritmos mais simples de ordenação. O seu funcionamento se dá da seguinte maneira: Selecionar o menor item do vetor e depois trocá-lo com o item

da primeira posição do vetor. Para concluir a ordenação, basta repetir as duas operações acima com os n-1 itens restantes, depois com os n-2 itens, até que reste apenas 1 elemento.

Exemplo:

1º = 2 4 1 6 5 3

2º = 1 4 2 6 5 3

3º = 1 2 4 6 5 3

4º = 1 2 3 6 5 4

5º = 1 2 3 4 5 6

Complexidade: sempre $O(n^2)$, no melhor, pior e também no caso médio.

3.3. InsertionSort

O método de ordenação por Inserção Direta é o mais rápido entre os outros métodos considerados básicos – Bubblesort e SelectionSort. A principal característica deste método consiste em ordenarmos o arranjo utilizando um sub-arranjo ordenado localizado em seu inicio, e a cada novo passo, acrescentamos a este sub-arranjo mais um elemento, até que atingimos o último elemento do arranjo fazendo assim com que ele se torne ordenado.

Exemplo:

1º = 2 4 1 6 5 3

2º = 1 2 4 6 5 3

3º = 1 2 4 5 6 3

4º = 1 2 3 4 5 6

Complexidade: Melhor caso: $O(n)$ — Pior caso: $O(n^2)$ — Caso Médio: $O(n^2)$

3.4. ShellSort

É uma extensão do algoritmo InsertionSort, ele simplesmente troca elementos perante seu tamanho no intervalo H, depois de completar as trocas, esse intervalo é diminuído e reordenado, a cada iteração ele irá ordenar automaticamente a medida que o intervalo decresce, até que H seja intervalo de 1, então ele será exatamente o InsertionSort.

Exemplo:

Chaves Inicias:

1º = O R D E N A

2º = N A D E O R — Quando H = 4

3º = D A N E O R — Quando H = 2

4º = A D E N O R — Quando H = 1

Complexidade: $O(n * log * n)$

3.5. HeapSort

O HeapSort utiliza uma estrutura de dados chamada heap, para ordenar os elementos a medida que os insere na estrutura. Assim, ao final das inserções, os elementos podem ser

sucessivamente removidos da raiz da heap, na ordem desejada, lembrando-se sempre de manter a propriedade de heap máximo.

A heap pode ser representada como uma árvore (uma árvore binária com propriedades especiais) ou como um vetor. Para uma ordenação crescente, deve ser construído uma heap mínima (o menor elemento fica na raiz). Para uma ordenação decrescente, deve ser construído uma heap máxima (o maior elemento fica na raiz).

Exemplo:

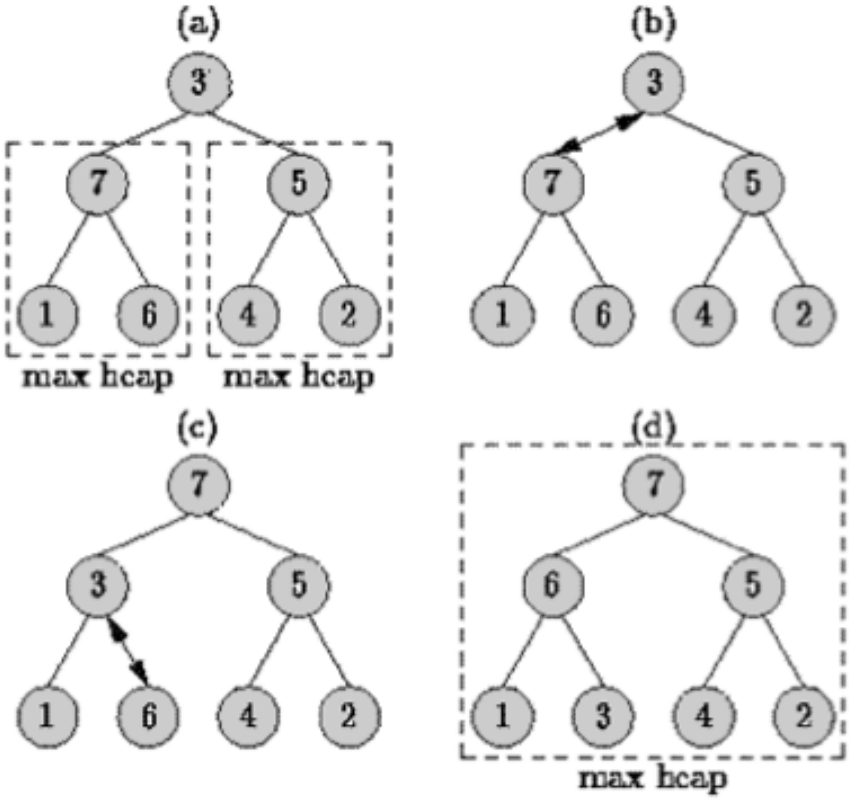

Figura 1. Exemplo Árvore HeapSort

Complexidade: $O(n * log * n)$.

3.6. QuickSort

Este método de classificação foi inventado por Hoare e seu desempenho é o melhor na maioria das vezes. O primeiro elemento do vetor a ser classificada é escolhido como o pivô. Depois da primeira fase da classificação, o pivô ocupa a posição que ocupará quando o vetor estiver completamente classificado. Os registros com valores de chaves menores do que o valor de chave do registro pivô o precedem no vetor e os registros com valores de chaves maiores do que o valor de chave do registro pivô o sucedem no vetor. Cada registro é comparado com o registro pivô e suas posições são permutadas se o valor de chave do registro for maior e o registro preceder o registro pivô, ou se o valor de chave do registro for menor e o registro suceder o registro pivô. A posição do registro pivô no

final de uma fase divide o vetor original em dois subvetores (partições), cada uma delas precisando ser ordenada.

Exemplo:

1º = 2 4 **1** 6 5 3

2º = 1 4 **2** 6 3 3

3º = 1 2 **4** 6 5 3

4º = 1 2 **3** 4 6 5

5º = 1 2 **3** 4 5 6

Complexidade: Melhor caso e caso médio: $O(n * log * n)$ e no pior $O(n^2)$

4. Análise das medidas e comparações realizadas

Sistema e Hardware utilizado para os testes: OS X 10.9.4 — Core i5 3ª Geração 2.5 GHz — 16GB RAM 1600MHz.

Os 11 algoritmos foram testados com as seguintes massas de entrada: 500, 2000, 10000, 30000, 50000, 100000, 150000, 200000, 250000, 300000.

Programa feito com a linguagem pascal, compilado com Free Pascal Compiler (FPC) pelo terminal.

Link para visualizar online (Recomendo, pois é mais fácil visualizá-los lado a lado): http://migre.me/kuyZF

Relação - Tempo em Milisegundos

	BubbleSort	SelectionSort	InsertionSort	ShellSort	HeapSort	QuickSort (Meio)	QuickSort Primeiro	QuickSort Último	QuickSort Randômico	QuickSort Mediana	Introsort
500	0.760	0.380	0.360	0.050	0.010	0.020	0.040	0.070	0.090	0.210	0.120
2000	12.590	4.650	3.590	0.250	0.230	0.320	0.260	0.230	0.320	0.690	0.350
10000	326.700	115.920	87.630	1.920	1.590	1.530	1.550	1.520	1.670	7.220	1.930
30000	2940.960	1048.890	798.950	6.970	5.380	5.030	5.150	5.000	5.440	58.270	6.080
50000	8717.920	3103.700	2395.420	13.220	10.010	8.910	9.390	9.020	9.870	162.980	11.180
100000	34982.460	12546.430	9618.540	30.150	22.200	19.180	19.680	19.440	20.940	663.730	23.430
150000	78561.490	27797.530	21598.880	49.470	35.480	29.970	30.810	29.770	32.360	1738.860	35.890
200000	127967.780	45796.460	34451.330	62.540	43.630	36.920	38.210	36.920	39.780	2970.700	43.980
250000	213059.800	76388.140	58196.930	87.580	60.390	46.770	50.230	50.950	54.490	5040.990	59.800
300000	292707.650	104925.380	78594.010	103.400	70.230	57.700	60.560	57.770	62.230	18448.290	68.340

Figura 2. Tabela: Relação média do tempo em milisegundos de todos os algoritmos

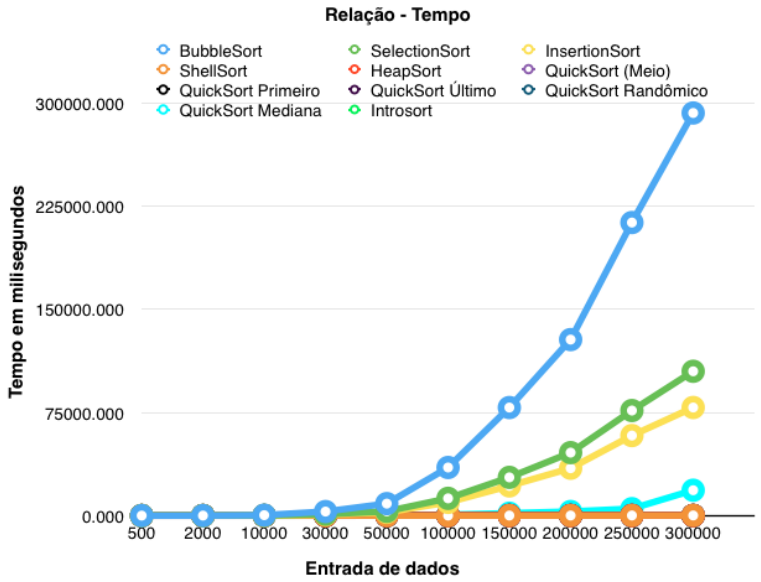

Figura 3. Gráfico: Relação média do tempo em milisegundos de todos os algoritmos

	Relação Trocas										
	BubbleSort	SelectionSort	InsertionSort	ShellSort	HeapSort	QuickSort (Meio)	QuickSort Primeiro	QuickSort Último	QuickSort Randômico	QuickSort Mediana	Introsort
500	62216	500	124928	14281	13667	6368	6563	6552	338	7001	3440
2000	998635	2000	1999455	78311	66638	30053	30881	30881	1354	32683	14697
10000	24994930	10000	49999322	550779	403538	177368	181161	181114	6773	190846	78918
30000	225129944	30000	450289350	2062890	1352416	587394	598015	598385	20323	628648	247749
50000	625029458	50000	1250087814	3816155	2367029	1020566	1039885	1039157	33875	1091396	421294
100000	2499952077	100000	4999998260	8744382	5034149	2156917	2194211	2193995	67738	2302384	865823
150000	5626114976	150000	11252382243	14349637	7807941	3332246	3395707	3393873	101611	3552476	1318141
200000	9998143251	200000	19996501330	20287636	10668163	4544913	4620769	4620338	135480	4839099	1778240
250000	15627054904	250000	31254381907	26325399	13559339	5775798	5867326	5869409	169352	6142144	2240022
300000	22501014802	300000	45002629602	33101660	16515900	7018981	7129765	7131311	203197	7455472	2707181

Figura 4. Tabela: Relação média do número de trocas de todos os algoritmos

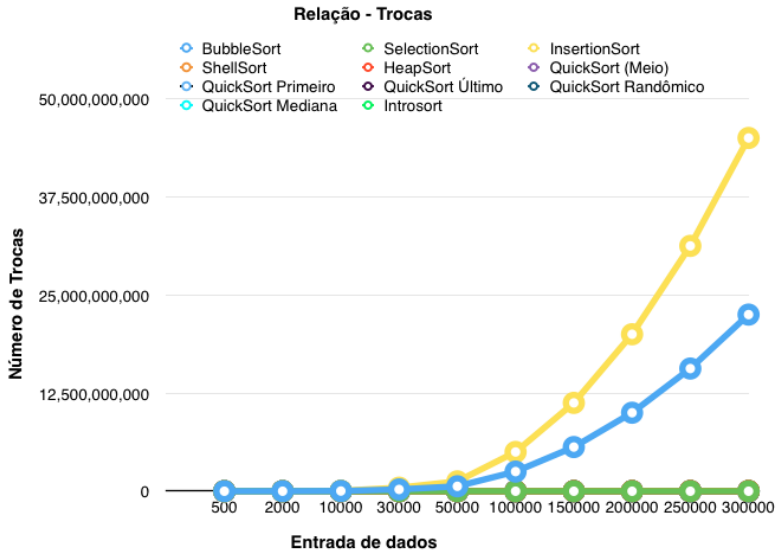

Figura 5. Gráfico: Relação média do número de trocas de todos os algoritmos

	Relação Comparações										
	BubbleSort	SelectionSort	InsertionSort	ShellSort	HeapSort	QuickSort (Meio)	QuickSort Primeiro	QuickSort Último	QuickSort Randômico	QuickSort Mediana	Introsort
500	250	250	250	1000	2500	1000	1500	1500	500	1750	3250
2000	1000	1000	1000	4000	10000	4000	6000	6000	2000	7000	13000
10000	5000	5000	5000	20000	5000	20000	30000	30000	10000	35000	65000
30000	15000	15000	15000	60000	150000	60000	90000	90000	30000	105000	195000
50000	25000	25000	25000	100000	250000	100000	150000	150000	50000	175000	325000
100000	50000	50000	50000	200000	500000	200000	300000	300000	100000	350000	650000
150000	75000	75000	75000	300000	750000	300000	450000	450000	150000	525000	975000
200000	100000	100000	100000	400000	1000000	400000	600000	600000	200000	700000	1300000
250000	125000	125000	125000	500000	1250000	500000	750000	750000	250000	875000	1625000
300000	150000	150000	150000	600000	1500000	600000	900000	900000	300000	1050000	1950000

Figura 6. Tabela: Relação média do número de comparações de todos os algoritmos

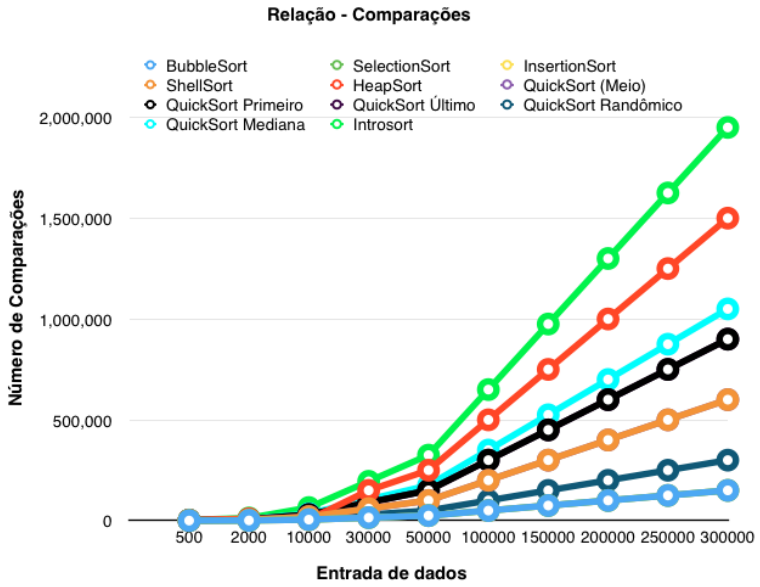

Figura 7. Gráfico: Relação média do número de comparações de todos os algoritmos

4.1. Interpretação

Pelo gráfico podemos observar claramente que os algoritmos BubbleSort, SelectionSort, InsertionSort são quadráticos, pois suas curvas formam um arco. Já os algoritmos ShellSort, HeapSort e QuickSort demonstram uma curva mais constante, o que demonstra sua linearidade como foi demonstrado na análise de algoritmos de Ziviani.

Podemos observar que ambas as comparações de tempo, iterações, e trocas aproximam-se dos resultados obtidos pelos testes de Ziviani.

4.1.1. A quantidade média de comparações e movimentações são métricas representativas do desempenho dos algoritmos analisados?

Sim. O número de trocas depende do estado do vetor (ordenado, parcialmente ordenado, ou completamente desordenado). O que caracteriza um número de trocas diferente em cada vetor. Já as comparações tem um número fixo.

4.1.2. Qual a relacao entre a quantidade média de comparacoes e movimentações e o tempo de execução dos algoritmos analisados?

Quanto maior a complexidade, maior será o custo, ou seja, são diretamente proporcionais.

4.1.3. Em qual ou quais situações são indicados os algoritmos com complexidade quadrática?

Apenas quando a massa de dados é pequena o suficiente para que a ordem quadrática não prejudique o desempenho do algoritmo.

4.1.4. Em qual ou quais situações são indicados os algoritmos com complexidade O(n*log n)?

Em situações onde a massa de dados é muito grande e que se precisa de velocidade.

4.1.5. Os resultados obtidos empiricamente estão em conformidade com os resultados obtidos analiticamente? Explique e justifique seus resultados, principalmente se os resultados empíricos não confirmarem os resultados analíticos.

Sim, os resultados estão bem próximos, salvo algumas exceções, onde o QuickSort com o elemento pivô primeiro e também no elemento pivô último apresentaram leve performance superior ao elemento randômico. Vale lembrar que o tempo necessário para calcular o número de trocas e comparações é acrescentado a função que calcula o tempo, isso certamente atrapalhou um pouco na veracidade dos dados.

4.1.6. Qual a melhor estratégia para a escolha do pivo no algoritmo de ordenação QuickSort?

De acordo com os dados obtidos e também de acordo com a teoria, o elemento pivô do meio demonstrou ser mais eficaz.

4.1.7. Qual o melhor valor para o parâmetro M usado na combinação do algoritmo quicksort com o algoritmo insertion sort? Para determinar o melhor valor de M considere apenas os testes com vetores de tamanhos: 500, 2.000, 10.000, 50.000, 200.000 e 300.000.

Baseado em nossos testes com atribuição do valor de M com 10, 20, 25, 30, 40 e 50, o valor 30 demonstrou ser mais eficaz na maioria dos testes, conforme também vimos empiricamente em aula.

5. Recomendações do uso de cada algoritmo

Com a conclusão desse trabalho prático, podemos garantir que todos os algoritmos de ordenação tem seus pontos fortes e também seus defeitos. Cada um é utilizado em diferentes situações, que depende da massa de entrada, do tempo disponível para a resolução do problema proposto, além do hardware. No entanto a última opção, não é um ponto fundamental a ser analisado, pois, hoje em dia, o problema do custo de memória pode ser contornado devido aos preços que estão mais acessíveis ao usuário.

Algoritmos com complexidade quadrática são de fácil programação e eficientes com pequena quantidade de dados. Do contrário o tempo de execução aumenta bastante, o que o torna inviável para a solução do problema. Nesse caso, é recomendado utilizar dos algoritmos com complexidade n log n, pois conseguem realizar todo o trabalho de ordenação de grandes massas de dados em um tempo significamente curto, dependendo do Hardware. No entanto esses algoritmos são de difícil programação e requerem boa prática para implementá-los.

No desenvolvimento do trabalho houve algumas dificuldades na implementação ao que diz respeito na construção dos algoritmos mais complexos, como o QuickSort e suas variações, e o HeapSort. Mas com pesquisas em diferentes literaturas e através de fontes confiáveis, foram sanadas as dúvidas. Concluindo o trabalho, podemos dizer que foi de grande utilidade para o decorrer do curso. Além de disponibilizar o conhecimento sobre cada algoritmo de ordenação e suas diferentes aplicações, foi aperfeiçoada a prática de programação e do trabalho em grupo.

6. Bibliografia

Livro Projetos de Algoritmos 3ª Edição - Ziviani + Resumos dos algoritmos de ordenação passados pelo professor Mário Luiz Rodrigues

Universidade Tecnológica Federal do Paraná - http://www.ft.unicamp.br/

Algoritmo AllMethods - Prof. Omero Francisco Bertol - UFTPR

Apostilas com conteúdo retirado do livro do Ziviani - Universidade Federal de Minas Gerais - UFMG

7. Listagem do código fonte

```
1  //===== Nome: Bruno Tome      | Matheus Calixto     =====//
2  //===== Matricula: 0011254    | 0011233             =====//
3  //===== ibrunotome@gmail.com | calixtinn@gmail.com =====//
4
5  program AED1;
6  uses sysutils, crt; // para usar as funcoes DateTimeToTimeStamp, now e
       TimeStampToMSecs
7
8  type dynamic = array of longword;
9
10 var
11      tam : longword;
12      vetor : dynamic;
13
14 {
15      Funcao Gerar: Gera um vetor Randomico para que seja testado em
            todos os algoritmos de ordenacao. A cada vez que e chamada, a
            funcao retorna
16      um vetor de elementos diferentes. No entando, um mesmo vetor
            randomico e testado primeiramente em todos os 11 algoritmos, e
             a partir da
17      segunda chamada, e que os elementos sao trocados.
18 }
19
20 function gerar(var qnt : longword):dynamic;
21 var
22      i,temp:longword;
23 begin
24     temp := qnt;
25     setlength(gerar,temp);
26     for i := 1 to temp do
27     begin
28         gerar[i] := random(temp) + 1;
```

```
29      end;
30  end;
31
32  // Procedimento para particionar o vetor ao meio, do algoritmo
        QuickSort.
33
34  procedure Particao(var vet : array of longword; Esq, Dir : longword;
        var i, j : longword; var cont_ite : double);
35  var x, aux: longword;
36
37  begin
38       i := Esq;
39       j := Dir;
40       x := vet[((j + i) div 2 )]; // Obtem o Pivo
41
42       repeat
43            while vet[i] < x do inc(i); cont_ite := cont_ite + 1;
44            while x < vet[j] do dec(j); cont_ite := cont_ite + 1;
45            if i <= j then
46            begin
47                 aux := vet[i];
48                 vet[i] := vet[j];
49                 vet[j] := aux;
50                 cont_ite := cont_ite + 3;
51                 inc(i);
52                 dec(j);
53            end;
54       until i > j;
55  end;
56  // Procedimento para a ordenacao e re-arranjo dos elementos do vetor.
        algoritmo QuickSort (pivo ao meio).
57
58  procedure Ordena( var vet: array of longword; Esq, Dir : longword; var
        cont_ite : double);
59  var i, j : longword;
60
61  begin
62       Particao(vet, Esq, Dir, i, j,cont_ite);
63       if Esq < j then Ordena(vet, Esq, j, cont_ite);
64       if i < Dir then Ordena(vet, i, Dir, cont_ite);
65  end;
66
67  // Procedimento para particionar o vetor, colocando como pivot o
        primeiro elemento do vetor.
68
69  procedure Particao2(var vet : array of longword; Esq, Dir : longword;
        var i, j : longword; var cont_ite : double);
70  var x,aux: longword;
71
72  begin
73       i := Esq;
74       j := Dir;
75       x := vet[i]; // Obtem o Pivo como primeiro elemento
76
77       repeat
78            while vet[i] < x do inc(i); cont_ite := cont_ite + 1;
```

```
79            while x < vet[j] do dec(j); cont_ite := cont_ite + 1;
80            if   i <= j then
81            begin
82                 aux:=vet[i];
83                 vet[i]:=vet[j];
84                 vet[j]:=aux;
85                 cont_ite:=cont_ite+3;
86                 i := i + 1;
87                 j := j - 1;
88            end;
89       until i > j;
90  end;
91
92  // Procedimento para a ordenacao e re-arranjo dos elementos do vetor.
       Algoritmo QuickSort (pivo no primeiro elemento).
93
94  procedure Ordena2( var vet: array of longword; Esq, Dir : longword;var
       cont_ite:double);
95  var i, j : longword;
96
97  begin
98       Particao2(vet, Esq, Dir, i, j,cont_ite);
99       if Esq < j then Ordena2(vet,Esq, j,cont_ite);
100      if i < Dir then Ordena2(vet,i, Dir,cont_ite);
101 end;
102
103 // Procedimento para particionar o vetor, colocando como pivot o ultimo
       elemento do vetor.
104
105 procedure Particao3(var vet : array of longword; Esq, Dir: longword;
       var i, j: longword;var cont_ite:double);
106 var x,aux: longword;
107
108 begin
109      i := Esq;
110      j := Dir;
111      x := vet[j]; // Obtem o Pivo como ultimo elemento
112
113      repeat
114           while vet[i] < x do inc(i); cont_ite := cont_ite + 1;
115           while x < vet[j] do dec(j); cont_ite := cont_ite + 1;
116           if   i <= j then
117           begin
118                aux := vet[i];
119                vet[i] := vet[j];
120                vet[j] := aux;
121                cont_ite := cont_ite + 3;
122                inc(i);
123                dec(j);
124           end;
125      until i > j;
126 end;
127
128 // Procedimento para a ordenacao e re-arranjo dos elementos do vetor.
       Algoritmo QuickSort (pivo no ultimo elemento).
129
```

```
130 procedure Ordena3( var vet: array of longword; Esq, Dir : longword; var
        cont_ite:double);
131 var i, j : longword;
132
133 begin
134      Particao3(vet, Esq, Dir, i, j, cont_ite);
135      if Esq < j then Ordena3(vet, Esq, j, cont_ite);
136      if i < Dir then Ordena3(vet, i, Dir, cont_ite);
137 end;
138
139 // Procedimento para particionar o vetor, onde escolhe-se o pivot a
       partir de um elemento de uma posicao randomica.
140
141 procedure Particao4(var vet:array of longword; Esq, Dir : longword; var
        i : longword; cont_ite : double);
142 var x, indice, aux, j : longword;
143 begin
144     indice := Esq + random(Dir - Esq) + 1;
145     x := vet[indice];
146     aux := vet[indice];
147     vet[indice] := vet[Dir];
148     vet[Dir] := aux; // Pivo Rand mico
149     indice := Dir;
150     i := Esq - 1;
151     j := Esq ;
152     while (j <= Dir - 1) do
153      begin
154           if(vet[j] <= x)then
155          begin
156              i := i+1;
157              aux := vet[i];
158              vet[i] := vet[j];
159              vet[j] := aux;
160                cont_ite := cont_ite + 2;
161          end;
162          j := j + 1;
163      end;
164     aux := vet[i + 1];
165     vet[i+1] := vet[indice];
166     vet[indice] := aux;
167     i := i + 1;
168      cont_ite := cont_ite + 2;
169 end;
170
171 // Procedimento para a ordenacao e re-arranjo dos elementos do vetor.
       Algoritmo QuickSort (pivot randomico).
172
173 procedure Ordena4(var vet:array of longword; Esq, Dir : longword; var
       cont_ite : double);
174 var i:longword;
175 begin
176     if (Esq < Dir) then
177     begin
178         cont_ite := cont_ite + 1;
179         particao4(vet, Esq, Dir, i, cont_ite);
180         Ordena4(vet, Esq, i - 1, cont_ite);
```

```
181         Ordena4(vet, i + 1, Dir, cont_ite);
182     end;
183 end;
184
185 {
186      Procedimento para particionar o vetor, onde escolhe-se o pivot a
            partir da mediana dos 3.
187      Esse processo resume-se em: Escolher o primeiro, ultimo, e o
            elemento central do vetor,
188      ordena-los, e escolher como pivot o elemento central desse novo
            vetor ordenado.
189 }
190
191 procedure Particao5(var vet : array of longword; Esq, Dir: longword;
        var i, j: longword; var cont_ite:double);
192 var x, aux, l, k: longword;
193     vet2 : array of longword;
194
195 begin
196      i := Esq;
197      j := Dir;
198      setlength(vet2, j);
199      vet2[2] := vet[((j+i) div 2)];
200      vet2[1] := vet[i];
201      vet2[3] := vet[j]; // Pivo Mediana
202
203      for l := 1 to 2 do
204      begin
205      for k := (l + 1) to 3 do
206      begin
207           if (vet2[k] < vet2[l]) then
208             begin
209                aux := vet2[l];
210                 vet2[l] := vet2[k];
211                 vet2[k] := aux;
212                 cont_ite := cont_ite + 1;
213             end;
214           end;
215     end;
216
217      x:=vet2[2];
218
219      repeat
220           while vet[i] < x do inc(i); cont_ite := cont_ite + 1;
221           while x < vet[j] do dec(j); cont_ite := cont_ite + 1;
222           if   i <= j then
223           begin
224                aux := vet[i];
225                vet[i] := vet[j];
226                vet[j] := aux;
227                inc(i);
228                dec(j);
229                cont_ite := cont_ite + 3;
230           end;
231      until i > j;
232
```

```
233 end;
234
235 //Procedimento para a ordenacao e re-arranjo dos elementos do vetor.
       Algoritmo QuickSort (pivot Mediana dos 3).
236
237 procedure Ordena5(var vet: array of longword; Esq, Dir : longword; var
       cont_ite:double);
238 var i, j : longword;
239
240 begin
241      Particao5(vet, Esq, Dir, i, j, cont_ite);
242      if Esq < j then Ordena5(vet, Esq, j, cont_ite);
243      if i < Dir then Ordena5(vet, i, Dir, cont_ite);
244 end;
245
246 // Inicio da Fun  o do algoritmo de ordena  o BubbleSort!
247
248 function bubblesort (vet: array of longword; qnt : longword; var
       cont_ite : double): double;
249 var
250      tempoinicio, tempofim, tempototal : double;
251      ts, ts2 : TTimeStamp; // TTimeStamp e um tipo de dado definido na
          unit sysutils.
252      i, j, k : longword;
253      aux : longword;
254
255 begin
256      TS := DateTimeToTimeStamp(now); // armazenda em TS o tempo atual
257      tempoinicio := TimeStampToMSecs(TS); // converte o tempo atual
          para milisegundos
258      k := qnt - 1;
259      for  i := k downto 1 do
260      begin
261           for j := 1 to i do
262           begin
263                if (vet[j] > vet[j+1]) then
264                begin
265                     aux := vet[j];
266                     vet[j] := vet[j+1];
267                     vet[j+1] := aux;
268                     cont_ite := cont_ite + 1;
269                end;
270           end;
271           k := k - 1;
272      end;
273
274      TS2 := DateTimeToTimeStamp(now);
275      tempofim := TimeStampToMSecs(TS2);
276      tempototal := tempofim - tempoinicio; // calcula o tempo gasto na
          execucao da funcao
277      bubblesort := tempototal; // Salva o valor do tempo total na
          fun  o BubbleSort
278 end;
279
280 // Inicio da Funcao do algoritmo de ordenacao SelectionSort!
281
```

```
282 function selectionsort (vet: array of longword; qnt :longword; var
        cont_ite:double): double;
283 var
284      tempoinicio, tempofim, tempototal : double;
285      ts, ts2 : TTimeStamp;
286      i, j,pos : longword;
287      aux : longword;
288
289 begin
290      TS := DateTimeToTimeStamp(now); // armazenda em TS o tempo atual
291      tempoinicio := TimeStampToMSecs(TS); // converte o tempo atual
            para milisegundos
292
293      for i:=0 to qnt-1 do
294      begin
295           pos:=i;
296           for j:= i+1 to qnt do
297           begin
298                if (vet[j] < vet[pos]) then
299                begin
300                     pos:=j;
301                end;
302           end;
303           aux:=vet[pos];
304           vet[pos]:=vet[i];
305           vet[i]:=aux;
306           cont_ite:=cont_ite + 1;
307      end;
308
309      TS2 := DateTimeToTimeStamp(now);
310      tempofim := TimeStampToMSecs(TS2);
311      tempototal := tempofim - tempoinicio; // calcula o tempo gasto na
            execucao da funcao
312      selectionsort := tempototal; // Salva o resultado do tempo total
            na fun o selectionsort
313 end;
314
315 // Inicio da Funcao do algoritmo de ordena o InsertionSort!
316
317 function insertionsort(vet: array of longword; qnt :longword; var
        cont_ite:double): double;
318 var
319      tempoinicio, tempofim, tempototal : double;
320      ts, ts2 : TTimeStamp;
321      i, j: longword;
322      aux : longword;
323
324 begin
325      TS := DateTimeToTimeStamp(now); // armazenda em TS o tempo atual
326      tempoinicio := TimeStampToMSecs(TS); // converte o tempo atual
            para milisegundos
327
328      for i := 2 to qnt do
329      begin
330           aux := vet[i];
331           j := (i - 1);
```

```
332           vet[0] := aux;
333           cont_ite := cont_ite + 1;
334
335           while (aux < vet[ j ]) do
336           begin
337                vet[j + 1] := vet[j];
338                dec(j);
339                cont_ite := cont_ite + 2;
340           end;
341           vet[j + 1] := aux;
342           cont_ite := cont_ite + 1;
343      end;
344
345      TS2 := DateTimeToTimeStamp(now);
346      tempofim := TimeStampToMSecs(TS2);
347      tempototal := tempofim - tempoinicio; // calcula o tempo gasto na
             e x e c u  o da funcao
348      insertionsort := tempototal;
349 end;
350
351 // Inicio da Funcao do algoritmo de ordenacao ShellSort!
352
353 function shellsort(vet: array of longword; qnt :longword; var cont_ite:
        double): double;
354 var
355      tempoinicio, tempofim, tempototal : double;
356      ts, ts2 : TTimeStamp;
357      i, j, h,x: longword;
358
359 begin
360      TS := DateTimeToTimeStamp(now); // armazenda em TS o tempo atual
361      tempoinicio := TimeStampToMSecs(TS); // converte o tempo atual
             para milisegundos
362
363      h := 1;
364      repeat
365           h := 3 * h + 1;
366           cont_ite := cont_ite+1;
367      until h > qnt;
368      repeat
369           h := h div 3;
370           cont_ite := cont_ite + 1;
371           for i := h + 1 to qnt do
372           begin
373                x := vet[i];
374                j := i;
375                cont_ite := cont_ite+2;
376                while (j > h) and (vet[j - h] > x) do
377                begin
378                     vet[j] := vet[j - h];
379                     j := j - h;
380                     cont_ite:=cont_ite+2;
381                end;
382                vet[j] := x;
383                cont_ite := cont_ite+1;
384           end;
```

```
385       until h = 1;
386
387       TS2 := DateTimeToTimeStamp(now);
388       tempofim := TimeStampToMSecs(TS2);
389       tempototal := tempofim - tempoinicio; // calcula o tempo gasto na
              execucao da fun   o
390       shellsort := tempototal;
391  end;
392
393  //In cio da Funcao do algoritmo de ordenacao HeapSort!
394
395  function heapsort (vet: array of longword; qnt :longword; var cont_ite:
         double): double;
396  var
397       tempoinicio, tempofim, tempototal : double;
398       ts, ts2 : TTimeStamp;
399       i, j, x, aux: longword;
400       Esq, Dir : longword;
401
402  begin
403       TS := DateTimeToTimeStamp(now); // armazenda em TS o tempo atual
404       tempoinicio := TimeStampToMSecs(TS); // converte o tempo atual
              para milisegundos
405
406       Esq := (qnt div 2) + 1;
407       cont_ite := cont_ite + 1;
408       Dir := qnt;
409       cont_ite := cont_ite + 1;
410       while Esq > 1 do
411       begin
412            Esq := Esq - 1;
413            i := Esq; j := 2 * i;
414            x := vet[i];
415            cont_ite := cont_ite + 3;
416            while j <= Dir do
417            begin
418                 if (j < Dir) and (vet[j] < vet[j + 1]) then inc(j);
                        cont_ite := cont_ite + 1;
419                 if   x >= vet[j] then break;
420                 vet[i] := vet[j];
421                 i := j; j := 2 * i;
422                 cont_ite := cont_ite + 3;
423            end;
424            vet[i] := x;
425            cont_ite := cont_ite + 1;
426       end;
427
428       while Dir > 1 do
429       begin
430            aux := vet[1];
431            vet[1] := vet[Dir];
432            vet[Dir] := aux;
433            Dir := Dir - 1;
434            i := Esq; j := 2 * i;
435            x := vet[i];
436            cont_ite := cont_ite + 2;
```

```
437           while j <= Dir do
438           begin
439                if (j < Dir) and (vet[j] < vet[j + 1]) then inc(j);
                      cont_ite := cont_ite + 1;
440                if x >= vet[j] then break;
441                vet[i] := vet[j];
442                i := j; j := 2 * i;
443                cont_ite := cont_ite + 2;
444           end;
445           vet[i] := x;
446           cont_ite := cont_ite + 1;
447      end;
448
449      TS2 := DateTimeToTimeStamp(now);
450      tempofim := TimeStampToMSecs(TS2);
451      tempototal := tempofim - tempoinicio; // calcula o tempo gasto na
            execucao da funcao
452      heapsort := tempototal;
453 end;
454
455 // Inicio da Funcao do algoritmo de ordena  o QuickSort (elemento
       Meio)
456
457 function quick_meio(vet: array of longword; qnt : longword; var
       cont_ite : double): double;
458 var
459      tempoinicio, tempofim, tempototal : double;
460      ts, ts2 : TTimeStamp;
461
462 begin
463      TS := DateTimeToTimeStamp(now); // armazenda em TS o tempo atual
464      tempoinicio := TimeStampToMSecs(TS); // converte o tempo atual
            para milisegundos
465
466      Ordena(vet,1,qnt,cont_ite);
467
468
469      TS2 := DateTimeToTimeStamp(now);
470      tempofim := TimeStampToMSecs(TS2);
471      tempototal := tempofim - tempoinicio; // calcula o tempo gasto na
            execu  o da funcao
472      quick_meio := tempototal;
473 end;
474
475 // In cio da Funcao do algoritmo de ordena  o QuickSort (elemento
       Primeiro)
476
477 function quick_primeiro(vet: array of longword; qnt :longword; var
       cont_ite : double): double;
478 var
479      tempoinicio, tempofim, tempototal : double;
480      ts, ts2 : TTimeStamp;
481
482 begin
483      TS := DateTimeToTimeStamp(now); // armazenda em TS o tempo atual
484      tempoinicio := TimeStampToMSecs(TS); // converte o tempo atual
```

```
        para milisegundos
485
486     Ordena2(vet, 1, qnt, cont_ite);
487
488
489     TS2 := DateTimeToTimeStamp(now);
490     tempofim := TimeStampToMSecs(TS2);
491     tempototal := tempofim - tempoinicio; // calcula o tempo gasto na
            execucao da funcao
492     quick_primeiro := tempototal;
493 end;
494
495 // Inicio da Funcao do algoritmo de ordena  o QuickSort (elemento
    Ultimo)
496
497 function quick_ultimo(vet: array of longword; qnt :longword; var
    cont_ite : double): double;
498 var
499     tempoinicio, tempofim, tempototal : double;
500     ts, ts2 : TTimeStamp;
501
502 begin
503     TS := DateTimeToTimeStamp(now); // armazenda em TS o tempo atual
504     tempoinicio := TimeStampToMSecs(TS); // converte o tempo atual
            para milisegundos
505     Ordena3(vet, 1, qnt, cont_ite);
506     TS2 := DateTimeToTimeStamp(now);
507     tempofim := TimeStampToMSecs(TS2);
508     tempototal := tempofim - tempoinicio; // calcula o tempo gasto na
            execu  o da fun  o
509     quick_ultimo := tempototal;
510 end;
511
512 // Inicio da Funcao do algoritmo de ordena  o QuickSort (elemento
    randomico)
513
514 function quick_random(vet: array of longword; qnt :longword;var
    cont_ite:double): double;
515 var
516     tempoinicio, tempofim, tempototal : double;
517     ts, ts2 : TTimeStamp;
518
519 begin
520     TS := DateTimeToTimeStamp(now); // armazenda em TS o tempo atual
521     tempoinicio := TimeStampToMSecs(TS); // converte o tempo atual
            para milisegundos
522     Ordena4(vet, 1, qnt, cont_ite);
523     TS2 := DateTimeToTimeStamp(now);
524     tempofim := TimeStampToMSecs(TS2);
525     tempototal := tempofim - tempoinicio; // calcula o tempo gasto na
            execu  o da fun  o
526     quick_random := tempototal;
527 end;
528
529 // Inicio da Funcao do algoritmo de ordena  o QuickSort (elemento
    Mediana)
```

```
530
531 function quick_mediana(vet: array of longword; qnt :longword;var
        cont_ite:double): double;
532 var
533      tempoinicio, tempofim, tempototal : double;
534      ts, ts2 : TTimeStamp;
535
536 begin
537      TS := DateTimeToTimeStamp(now); // armazenda em TS o tempo atual
538      tempoinicio := TimeStampToMSecs(TS); // converte o tempo atual
            para milisegundos
539      Ordena5(vet, 1, qnt, cont_ite);
540      TS2 := DateTimeToTimeStamp(now);
541      tempofim := TimeStampToMSecs(TS2);
542      tempototal := tempofim - tempoinicio; // calcula o tempo gasto na
            execu  o da fun  o
543      quick_mediana := tempototal;
544 end;
545
546 // Inicio da Funcao do algoritmo de ordenacao QuickSort (Combo com
        Insertion Sort}
547
548 function IntroSort(vet:array of longword; tam:longword; var cont_ite:
        double): double;
549
550      procedure Insercao(var vet:array of longword; tam:longword; var
            cont_ite: double);
551      var
552          i, j, value:longword;
553
554      begin
555          i:=1;
556          while(i < tam) do
557           begin
558           value := vet[i];
559           cont_ite := cont_ite + 1;
560             j := i - 1;
561             while ((j >= 0) and (vet[j] > value))do
562             begin
563                 vet[j + 1] := vet[j];
564                 vet[j] := value;
565                 dec(j);
566             end;
567             inc(i);
568        end;
569      end;
570
571      procedure OrdenaIntro(inicio, fim:longword; var cont_ite: double);
572      var i, j, aux, pivo : longword;
573      begin
574           i := inicio;
575           j := fim;
576           if (i >= j) then aux := 0
577          else
578          begin
579            if ((fim - inicio) < (30)) then Insercao(vet[inicio], (fim-
```

```
            inicio) + 1, cont_ite);
580         // 30 foi o valor escolhido para M (chamar o InsertionSort)
581           pivo := vet[(inicio+fim)div 2];
582          while (i < j) do
583           begin
584             while ((i < j) and (vet[i] < pivo)) do inc(i);
585             while ((i < j) and (vet[j] > pivo)) do dec(j);
586                  if (i < j) then
587                 begin
588                     cont_ite := cont_ite + 1;
589                      aux := vet[i];
590                      vet[i] := vet[j];
591                      vet[j] := aux;
592                      inc(i);
593                      dec(j);
594                     end;
595            end;
596             if (j < i) then
597          begin
598               aux := j;
599               j := i;
600               i := aux;
601         end;
602         OrdenaIntro(inicio, i, cont_ite);
603              if (i = inicio) then
604         begin
605               OrdenaIntro(i + 1, fim, cont_ite);
606           end
607           else
608           begin
609               OrdenaIntro(i, fim, cont_ite);
610           end;
611     end;
612     end;
613
614 var tempoinicio, tempofim, tempototal : double;
615     ts, ts2 : TTimeStamp;
616
617 begin
618      TS := DateTimeToTimeStamp(now); // armazenda em TS o tempo atual
619      tempoinicio := TimeStampToMSecs(TS); // converte o tempo atual
            para milisegundos
620     OrdenaIntro(1, tam, cont_ite);
621      TS2 := DateTimeToTimeStamp(now);
622     tempofim := TimeStampToMSecs(TS2);
623      tempototal := tempofim - tempoinicio; // calcula o tempo gasto na
            e x e c u  o da funcao
624     IntroSort := tempototal;
625 end;
626
627 var opcao: byte;
628      i:longword;
629      soma_bolha, soma_selection, soma_insertion, soma_shell, soma_heap,
            soma_quick_meio,
630      soma_quick_primeiro, soma_quick_ultimo, soma_quick_random,
            soma_quick_mediana, cont_bolha,
```

```
631       cont_selection, cont_insertion, cont_shell, cont_heap,
              cont_quick_m, cont_quick_p,
632       cont_quick_u, cont_quick_r, cont_quick_med, cont_introsort,
              soma_introsort : double;
633  begin
634
635       // Escolha da massa de entradas para a realizacao dos testes de
              cada algoritmo de ordenacao
636
637       clrscr;
638       writeln('Escolha uma massa de entrada: ');
639       writeln;
640       writeln('1 = 500');
641       writeln('2 = 2000');
642       writeln('3 = 10000');
643       writeln('4 = 30000');
644       writeln('5 = 50000');
645       writeln('6 = 100000');
646       writeln('7 = 150000');
647       writeln('8 = 200000');
648       writeln('9 = 250000');
649       writeln('10 = 300000');
650       writeln;
651       write('Op   o: ');
652       read(opcao);
653       writeln;
654       write('Ordenando');
655       delay(500);
656       write('.');
657       delay(500);
658       write('.');
659       delay(500);
660       write('.');
661       writeln;
662
663       {
664            Em cada case ha uma massa de entrada diferente.
665            Primeiramente 500, 2000,
                   10000,30000,50000,100000,150000,200000,250000 e 300000
                   elementos.
666            Sao inicializadas todas as variaveis : Tempo de execucao de
                   cada algoritmo e contadores de iteracoes.
667            Apos inicializadas as variaveis, e executado um For que vai
                   de 1 a 100,
668            testando os algoritmos com diferentes elementos, sendo que em
                    cada passagem
669            o vetor e o mesmo para cada algoritmo. Apos rodados todos os
                   algoritmos 100 vezes,
670            e feita uma media de tempo de cada algoritmo e do numero de
                   iteracoes.
671       }
672
673       case opcao of
674            1 : begin
675                      soma_bolha := 0;
676                      soma_selection := 0;
```

```
677                     soma_insertion := 0;
678                     soma_shell := 0;
679                     soma_heap := 0;
680                     soma_quick_meio := 0;
681                     soma_quick_primeiro := 0;
682                     soma_quick_ultimo := 0;
683                     soma_quick_random := 0;
684                     soma_quick_mediana := 0;
685                     soma_quick_mediana := 0;
686                     soma_introsort := 0;
687                     cont_bolha := 0;
688                     cont_selection := 0;
689                     cont_insertion := 0;
690                     cont_shell := 0;
691                     cont_heap := 0;
692                     cont_quick_m := 0;
693                     cont_quick_p := 0;
694                     cont_quick_u := 0;
695                     cont_quick_r := 0;
696                     cont_quick_med := 0;
697                     cont_introsort := 0;
698                     tam := 500;
699
700                     for i:=1 to 100 do
701                     begin
702                          vetor := gerar(tam);
703                          soma_bolha := soma_bolha + bubblesort(vetor,
                                tam, cont_bolha);
704                          soma_selection := soma_selection +
                                selectionsort(vetor, tam, cont_selection);
705                          soma_insertion := soma_insertion +
                                insertionsort(vetor, tam, cont_insertion);
706                          soma_shell := soma_shell + shellsort(vetor,
                                tam, cont_shell);
707                          soma_heap := soma_heap + heapsort(vetor, tam,
                                cont_heap);
708                          soma_quick_meio := soma_quick_meio +
                                quick_meio(vetor, tam, cont_quick_m);
709                          soma_quick_primeiro:=soma_quick_primeiro +
                                quick_primeiro(vetor, tam, cont_quick_p);
710                          soma_quick_ultimo:=soma_quick_ultimo +
                                quick_ultimo(vetor, tam, cont_quick_u);
711                          soma_quick_random:=soma_quick_random +
                                quick_random(vetor, tam, cont_quick_r);
712                          soma_quick_mediana:=soma_quick_mediana +
                                quick_mediana(vetor, tam, cont_quick_med);
713                          soma_introsort := soma_introsort + IntroSort(
                                vetor, tam, cont_introsort);
714                     end;
715
716                     writeln('Media BubbleSort: ',(soma_bolha/100):0:3,'
                          milisegundos || Iteracoes medias : ',(
                         cont_bolha/100):0:0);
717                     writeln('Media SelectionSort: ',(soma_selection
                         /100):0:3,' milisegundos || Iteracoes medias :
                         ',(cont_selection/100):0:0);
```

```
718                     writeln('Media InsertionSort: ',(soma_insertion
                            /100):0:3,' milisegundos || Iteracoes medias :
                            ',(cont_insertion/100):0:0);
719                     writeln('Media ShellSort: ',(soma_shell/100):0:3,'
                            milisegundos || Iteracoes medias : ',(
                            cont_shell/100):0:0);
720                     writeln('Media HeapSort: ',(soma_heap/100):0:3,'
                            milisegundos || Iteracoes medias : ',(cont_heap
                            /100):0:0);
721                     writeln('Media QuickSort (meio): ',(soma_quick_meio
                            /100):0:3,' milisegundos || Iteracoes medias :
                            ',(cont_quick_m/100):0:0);
722                     writeln('Media QuickSort (primeiro elemento): ',(
                            soma_quick_primeiro/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_p/100):0:0);
723                     writeln('Media QuickSort (ultimo elemento): ',(
                            soma_quick_ultimo/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_u/100):0:0);
724                     writeln('Media QuickSort (elemento randomico): ',(
                            soma_quick_random/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_r/100):0:0);
725                     writeln('Media QuickSort (mediana): ',(
                            soma_quick_mediana/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_med/100):0:0);
726                     writeln('Media IntroSort: ',(soma_introsort/100)
                            :0:3,' milisegundos || Iteracoes medias : ',(
                            cont_introsort/100):0:0);
727                     readln;
728                 end;
729
730             2 : begin
731                     soma_bolha := 0;
732                     soma_selection := 0;
733                     soma_insertion := 0;
734                     soma_shell := 0;
735                     soma_heap := 0;
736                     soma_quick_meio := 0;
737                     soma_quick_primeiro := 0;
738                     soma_quick_ultimo := 0;
739                     soma_quick_random := 0;
740                     soma_quick_mediana := 0;
741                     soma_quick_mediana := 0;
742                     soma_introsort := 0;
743                     cont_bolha := 0;
744                     cont_selection := 0;
745                     cont_insertion := 0;
746                     cont_shell := 0;
747                     cont_heap := 0;
748                     cont_quick_m := 0;
749                     cont_quick_p := 0;
750                     cont_quick_u := 0;
751                     cont_quick_r := 0;
752                     cont_quick_med := 0;
753                     cont_introsort := 0;
754                     tam := 2000;
755
```

```
756                     for i:=1 to 100 do
757                     begin
758                          vetor := gerar(tam);
759                          soma_bolha := soma_bolha + bubblesort(vetor,
                                tam, cont_bolha);
760                          soma_selection := soma_selection +
                                selectionsort(vetor, tam, cont_selection);
761                          soma_insertion := soma_insertion +
                                insertionsort(vetor, tam, cont_insertion);
762                          soma_shell := soma_shell + shellsort(vetor,
                                tam, cont_shell);
763                          soma_heap := soma_heap + heapsort(vetor, tam,
                                cont_heap);
764                          soma_quick_meio := soma_quick_meio +
                                quick_meio(vetor, tam, cont_quick_m);
765                          soma_quick_primeiro:=soma_quick_primeiro +
                                quick_primeiro(vetor, tam, cont_quick_p);
766                          soma_quick_ultimo:=soma_quick_ultimo +
                                quick_ultimo(vetor, tam, cont_quick_u);
767                          soma_quick_random:=soma_quick_random +
                                quick_random(vetor, tam, cont_quick_r);
768                          soma_quick_mediana:=soma_quick_mediana +
                                quick_mediana(vetor, tam, cont_quick_med);
769                          soma_introsort := soma_introsort + IntroSort(
                                vetor, tam, cont_introsort);
770                     end;
771
772                     writeln('Media BubbleSort: ',(soma_bolha/100):0:3,'
                          milisegundos || Iteracoes medias : ',(
                         cont_bolha/100):0:0);
773                     writeln('Media SelectionSort: ',(soma_selection
                         /100):0:3,' milisegundos || Iteracoes medias :
                         ',(cont_selection/100):0:0);
774                     writeln('Media InsertionSort: ',(soma_insertion
                         /100):0:3,' milisegundos || Iteracoes medias :
                         ',(cont_insertion/100):0:0);
775                     writeln('Media ShellSort: ',(soma_shell/100):0:3,'
                         milisegundos || Iteracoes medias : ',(
                         cont_shell/100):0:0);
776                     writeln('Media HeapSort: ',(soma_heap/100):0:3,'
                         milisegundos || Iteracoes medias : ',(cont_heap
                         /100):0:0);
777                     writeln('Media QuickSort (meio): ',(soma_quick_meio
                         /100):0:3,' milisegundos || Iteracoes medias :
                         ',(cont_quick_m/100):0:0);
778                     writeln('Media QuickSort (primeiro elemento): ',(
                         soma_quick_primeiro/100):0:3,' milisegundos ||
                         Iteracoes medias : ',(cont_quick_p/100):0:0);
779                     writeln('Media QuickSort (ultimo elemento): ',(
                         soma_quick_ultimo/100):0:3,' milisegundos ||
                         Iteracoes medias : ',(cont_quick_u/100):0:0);
780                     writeln('Media QuickSort (elemento randomico): ',(
                         soma_quick_random/100):0:3,' milisegundos ||
                         Iteracoes medias : ',(cont_quick_r/100):0:0);
781                     writeln('Media QuickSort (mediana): ',(
                         soma_quick_mediana/100):0:3,' milisegundos ||
```

```
                    Iteracoes medias : ',(cont_quick_med/100):0:0);
782                 writeln('Media IntroSort: ',(soma_introsort/100)
                        :0:3,' milisegundos || Iteracoes medias : ',(
                        cont_introsort/100):0:0);
783                 readln;
784           end;
785
786       3 : begin
787                 soma_bolha := 0;
788                 soma_selection := 0;
789                 soma_insertion := 0;
790                 soma_shell := 0;
791                 soma_heap := 0;
792                 soma_quick_meio := 0;
793                 soma_quick_primeiro := 0;
794                 soma_quick_ultimo := 0;
795                 soma_quick_random := 0;
796                 soma_quick_mediana := 0;
797                 soma_quick_mediana := 0;
798                 soma_introsort := 0;
799                 cont_bolha := 0;
800                 cont_selection := 0;
801                 cont_insertion := 0;
802                 cont_shell := 0;
803                 cont_heap := 0;
804                 cont_quick_m := 0;
805                 cont_quick_p := 0;
806                 cont_quick_u := 0;
807                 cont_quick_r := 0;
808                 cont_quick_med := 0;
809                 cont_introsort := 0;
810                 tam := 10000;
811
812                 for i:=1 to 100 do
813                 begin
814                       vetor := gerar(tam);
815                       soma_bolha := soma_bolha + bubblesort(vetor,
                              tam, cont_bolha);
816                       soma_selection := soma_selection +
                              selectionsort(vetor, tam, cont_selection);
817                       soma_insertion := soma_insertion +
                              insertionsort(vetor, tam, cont_insertion);
818                       soma_shell := soma_shell + shellsort(vetor,
                              tam, cont_shell);
819                       soma_heap := soma_heap + heapsort(vetor, tam,
                              cont_heap);
820                       soma_quick_meio := soma_quick_meio +
                              quick_meio(vetor, tam, cont_quick_m);
821                       soma_quick_primeiro:=soma_quick_primeiro +
                              quick_primeiro(vetor, tam, cont_quick_p);
822                       soma_quick_ultimo:=soma_quick_ultimo +
                              quick_ultimo(vetor, tam, cont_quick_u);
823                       soma_quick_random:=soma_quick_random +
                              quick_random(vetor, tam, cont_quick_r);
824                       soma_quick_mediana:=soma_quick_mediana +
                              quick_mediana(vetor, tam, cont_quick_med);
```

```
825                         soma_introsort := soma_introsort + IntroSort(
                                vetor, tam, cont_introsort);
826                 end;
827
828                 writeln('Media BubbleSort: ',(soma_bolha/100):0:3,'
                        milisegundos || Iteracoes medias : ',(
                        cont_bolha/100):0:0);
829                 writeln('Media SelectionSort: ',(soma_selection
                        /100):0:3,' milisegundos || Iteracoes medias :
                        ',(cont_selection/100):0:0);
830                 writeln('Media InsertionSort: ',(soma_insertion
                        /100):0:3,' milisegundos || Iteracoes medias :
                        ',(cont_insertion/100):0:0);
831                 writeln('Media ShellSort: ',(soma_shell/100):0:3,'
                        milisegundos || Iteracoes medias : ',(
                        cont_shell/100):0:0);
832                 writeln('Media HeapSort: ',(soma_heap/100):0:3,'
                        milisegundos || Iteracoes medias : ',(cont_heap
                        /100):0:0);
833                 writeln('Media QuickSort (meio): ',(soma_quick_meio
                        /100):0:3,' milisegundos || Iteracoes medias :
                        ',(cont_quick_m/100):0:0);
834                 writeln('Media QuickSort (primeiro elemento): ',(
                        soma_quick_primeiro/100):0:3,' milisegundos ||
                        Iteracoes medias : ',(cont_quick_p/100):0:0);
835                 writeln('Media QuickSort (ultimo elemento): ',(
                        soma_quick_ultimo/100):0:3,' milisegundos ||
                        Iteracoes medias : ',(cont_quick_u/100):0:0);
836                 writeln('Media QuickSort (elemento randomico): ',(
                        soma_quick_random/100):0:3,' milisegundos ||
                        Iteracoes medias : ',(cont_quick_r/100):0:0);
837                 writeln('Media QuickSort (mediana): ',(
                        soma_quick_mediana/100):0:3,' milisegundos ||
                        Iteracoes medias : ',(cont_quick_med/100):0:0);
838                 writeln('Media IntroSort: ',(soma_introsort/100)
                        :0:3,' milisegundos || Iteracoes medias : ',(
                        cont_introsort/100):0:0);
839                 readln;
840             end;
841
842         4 : begin
843                 soma_bolha := 0;
844                 soma_selection := 0;
845                 soma_insertion := 0;
846                 soma_shell := 0;
847                 soma_heap := 0;
848                 soma_quick_meio := 0;
849                 soma_quick_primeiro := 0;
850                 soma_quick_ultimo := 0;
851                 soma_quick_random := 0;
852                 soma_quick_mediana := 0;
853                 soma_quick_mediana := 0;
854                 soma_introsort := 0;
855                 cont_bolha := 0;
856                 cont_selection := 0;
857                 cont_insertion := 0;
```

```
858                     cont_shell := 0;
859                     cont_heap := 0;
860                     cont_quick_m := 0;
861                     cont_quick_p := 0;
862                     cont_quick_u := 0;
863                     cont_quick_r := 0;
864                     cont_quick_med := 0;
865                     cont_introsort := 0;
866                     tam := 30000;
867
868                     for i:=1 to 100 do
869                     begin
870                          vetor := gerar(tam);
871                          soma_bolha := soma_bolha + bubblesort(vetor,
                                tam, cont_bolha);
872                          soma_selection := soma_selection +
                                selectionsort(vetor, tam, cont_selection);
873                          soma_insertion := soma_insertion +
                                insertionsort(vetor, tam, cont_insertion);
874                          soma_shell := soma_shell + shellsort(vetor,
                                tam, cont_shell);
875                          soma_heap := soma_heap + heapsort(vetor, tam,
                                cont_heap);
876                          soma_quick_meio := soma_quick_meio +
                                quick_meio(vetor, tam, cont_quick_m);
877                          soma_quick_primeiro:=soma_quick_primeiro +
                                quick_primeiro(vetor, tam, cont_quick_p);
878                          soma_quick_ultimo:=soma_quick_ultimo +
                                quick_ultimo(vetor, tam, cont_quick_u);
879                          soma_quick_random:=soma_quick_random +
                                quick_random(vetor, tam, cont_quick_r);
880                          soma_quick_mediana:=soma_quick_mediana +
                                quick_mediana(vetor, tam, cont_quick_med);
881                          soma_introsort := soma_introsort + IntroSort(
                                vetor, tam, cont_introsort);
882                     end;
883
884                     writeln('Media BubbleSort: ',(soma_bolha/100):0:3,'
                           milisegundos || Iteracoes medias : ',(
                          cont_bolha/100):0:0);
885                     writeln('Media SelectionSort: ',(soma_selection
                          /100):0:3,' milisegundos || Iteracoes medias :
                          ',(cont_selection/100):0:0);
886                     writeln('Media InsertionSort: ',(soma_insertion
                          /100):0:3,' milisegundos || Iteracoes medias :
                          ',(cont_insertion/100):0:0);
887                     writeln('Media ShellSort: ',(soma_shell/100):0:3,'
                          milisegundos || Iteracoes medias : ',(
                          cont_shell/100):0:0);
888                     writeln('Media HeapSort: ',(soma_heap/100):0:3,'
                          milisegundos || Iteracoes medias : ',(cont_heap
                          /100):0:0);
889                     writeln('Media QuickSort (meio): ',(soma_quick_meio
                          /100):0:3,' milisegundos || Iteracoes medias :
                          ',(cont_quick_m/100):0:0);
890                     writeln('Media QuickSort (primeiro elemento): ',(
```

```
                    soma_quick_primeiro/100):0:3,' milisegundos ||
                    Iteracoes medias : ',(cont_quick_p/100):0:0);
891                 writeln('Media QuickSort (ultimo elemento): ',(
                    soma_quick_ultimo/100):0:3,' milisegundos ||
                    Iteracoes medias : ',(cont_quick_u/100):0:0);
892                 writeln('Media QuickSort (elemento randomico): ',(
                    soma_quick_random/100):0:3,' milisegundos ||
                    Iteracoes medias : ',(cont_quick_r/100):0:0);
893                 writeln('Media QuickSort (mediana): ',(
                    soma_quick_mediana/100):0:3,' milisegundos ||
                    Iteracoes medias : ',(cont_quick_med/100):0:0);
894                 writeln('Media IntroSort: ',(soma_introsort/100)
                    :0:3,' milisegundos || Iteracoes medias : ',(
                    cont_introsort/100):0:0);
895                 readln;
896           end;
897
898       5 : begin
899                 soma_bolha := 0;
900                 soma_selection := 0;
901                 soma_insertion := 0;
902                 soma_shell := 0;
903                 soma_heap := 0;
904                 soma_quick_meio := 0;
905                 soma_quick_primeiro := 0;
906                 soma_quick_ultimo := 0;
907                 soma_quick_random := 0;
908                 soma_quick_mediana := 0;
909                 soma_quick_mediana := 0;
910                 soma_introsort := 0;
911                 cont_bolha := 0;
912                 cont_selection := 0;
913                 cont_insertion := 0;
914                 cont_shell := 0;
915                 cont_heap := 0;
916                 cont_quick_m := 0;
917                 cont_quick_p := 0;
918                 cont_quick_u := 0;
919                 cont_quick_r := 0;
920                 cont_quick_med := 0;
921                 cont_introsort := 0;
922                 tam := 50000;
923
924                 for i:=1 to 100 do
925                 begin
926                       vetor := gerar(tam);
927                       soma_bolha := soma_bolha + bubblesort(vetor,
                          tam, cont_bolha);
928                       soma_selection := soma_selection +
                          selectionsort(vetor, tam, cont_selection);
929                       soma_insertion := soma_insertion +
                          insertionsort(vetor, tam, cont_insertion);
930                       soma_shell := soma_shell + shellsort(vetor,
                          tam, cont_shell);
931                       soma_heap := soma_heap + heapsort(vetor, tam,
                          cont_heap);
```

```
932                         soma_quick_meio := soma_quick_meio +
                                quick_meio(vetor, tam, cont_quick_m);
933                         soma_quick_primeiro:=soma_quick_primeiro +
                                quick_primeiro(vetor, tam, cont_quick_p);
934                         soma_quick_ultimo:=soma_quick_ultimo +
                                quick_ultimo(vetor, tam, cont_quick_u);
935                         soma_quick_random:=soma_quick_random +
                                quick_random(vetor, tam, cont_quick_r);
936                         soma_quick_mediana:=soma_quick_mediana +
                                quick_mediana(vetor, tam, cont_quick_med);
937                         soma_introsort := soma_introsort + IntroSort(
                                vetor, tam, cont_introsort);
938                     end;
939
940                     writeln('Media BubbleSort: ',(soma_bolha/100):0:3,'
                            milisegundos || Iteracoes medias : ',(
                            cont_bolha/100):0:0);
941                     writeln('Media SelectionSort: ',(soma_selection
                            /100):0:3,' milisegundos || Iteracoes medias :
                            ',(cont_selection/100):0:0);
942                     writeln('Media InsertionSort: ',(soma_insertion
                            /100):0:3,' milisegundos || Iteracoes medias :
                            ',(cont_insertion/100):0:0);
943                     writeln('Media ShellSort: ',(soma_shell/100):0:3,'
                            milisegundos || Iteracoes medias : ',(
                            cont_shell/100):0:0);
944                     writeln('Media HeapSort: ',(soma_heap/100):0:3,'
                            milisegundos || Iteracoes medias : ',(cont_heap
                            /100):0:0);
945                     writeln('Media QuickSort (meio): ',(soma_quick_meio
                            /100):0:3,' milisegundos || Iteracoes medias :
                            ',(cont_quick_m/100):0:0);
946                     writeln('Media QuickSort (primeiro elemento): ',(
                            soma_quick_primeiro/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_p/100):0:0);
947                     writeln('Media QuickSort (ultimo elemento): ',(
                            soma_quick_ultimo/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_u/100):0:0);
948                     writeln('Media QuickSort (elemento randomico): ',(
                            soma_quick_random/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_r/100):0:0);
949                     writeln('Media QuickSort (mediana): ',(
                            soma_quick_mediana/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_med/100):0:0);
950                     writeln('Media IntroSort: ',(soma_introsort/100)
                            :0:3,' milisegundos || Iteracoes medias : ',(
                            cont_introsort/100):0:0);
951                     readln;
952                 end;
953
954             6 : begin
955                     soma_bolha := 0;
956                     soma_selection := 0;
957                     soma_insertion := 0;
958                     soma_shell := 0;
959                     soma_heap := 0;
```

```
960                     soma_quick_meio := 0;
961                     soma_quick_primeiro := 0;
962                     soma_quick_ultimo := 0;
963                     soma_quick_random := 0;
964                     soma_quick_mediana := 0;
965                     soma_quick_mediana := 0;
966                     soma_introsort := 0;
967                     cont_bolha := 0;
968                     cont_selection := 0;
969                     cont_insertion := 0;
970                     cont_shell := 0;
971                     cont_heap := 0;
972                     cont_quick_m := 0;
973                     cont_quick_p := 0;
974                     cont_quick_u := 0;
975                     cont_quick_r := 0;
976                     cont_quick_med := 0;
977                     cont_introsort := 0;
978                     tam := 100000;
979
980                     for i:=1 to 100 do
981                     begin
982                          vetor := gerar(tam);
983                          soma_bolha := soma_bolha + bubblesort(vetor,
                                tam, cont_bolha);
984                          soma_selection := soma_selection +
                                selectionsort(vetor, tam, cont_selection);
985                          soma_insertion := soma_insertion +
                                insertionsort(vetor, tam, cont_insertion);
986                          soma_shell := soma_shell + shellsort(vetor,
                                tam, cont_shell);
987                          soma_heap := soma_heap + heapsort(vetor, tam,
                                cont_heap);
988                          soma_quick_meio := soma_quick_meio +
                                quick_meio(vetor, tam, cont_quick_m);
989                          soma_quick_primeiro:=soma_quick_primeiro +
                                quick_primeiro(vetor, tam, cont_quick_p);
990                          soma_quick_ultimo:=soma_quick_ultimo +
                                quick_ultimo(vetor, tam, cont_quick_u);
991                          soma_quick_random:=soma_quick_random +
                                quick_random(vetor, tam, cont_quick_r);
992                          soma_quick_mediana:=soma_quick_mediana +
                                quick_mediana(vetor, tam, cont_quick_med);
993                          soma_introsort := soma_introsort + IntroSort(
                                vetor, tam, cont_introsort);
994                     end;
995
996                     writeln('Media BubbleSort: ',(soma_bolha/100):0:3,'
                          milisegundos || Iteracoes medias : ',(
                         cont_bolha/100):0:0);
997                     writeln('Media SelectionSort: ',(soma_selection
                         /100):0:3,' milisegundos || Iteracoes medias :
                         ',(cont_selection/100):0:0);
998                     writeln('Media InsertionSort: ',(soma_insertion
                         /100):0:3,' milisegundos || Iteracoes medias :
                         ',(cont_insertion/100):0:0);
```

```
999                     writeln('Media ShellSort: ',(soma_shell/100):0:3,'
                            milisegundos || Iteracoes medias : ',(
                            cont_shell/100):0:0);
1000                    writeln('Media HeapSort: ',(soma_heap/100):0:3,'
                            milisegundos || Iteracoes medias : ',(cont_heap
                            /100):0:0);
1001                    writeln('Media QuickSort (meio): ',(soma_quick_meio
                            /100):0:3,' milisegundos || Iteracoes medias :
                            ',(cont_quick_m/100):0:0);
1002                    writeln('Media QuickSort (primeiro elemento): ',(
                            soma_quick_primeiro/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_p/100):0:0);
1003                    writeln('Media QuickSort (ultimo elemento): ',(
                            soma_quick_ultimo/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_u/100):0:0);
1004                    writeln('Media QuickSort (elemento randomico): ',(
                            soma_quick_random/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_r/100):0:0);
1005                    writeln('Media QuickSort (mediana): ',(
                            soma_quick_mediana/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_med/100):0:0);
1006                    writeln('Media IntroSort: ',(soma_introsort/100)
                            :0:3,' milisegundos || Iteracoes medias : ',(
                            cont_introsort/100):0:0);
1007                    readln;
1008                end;
1009
1010            7 : begin
1011                    soma_bolha := 0;
1012                    soma_selection := 0;
1013                    soma_insertion := 0;
1014                    soma_shell := 0;
1015                    soma_heap := 0;
1016                    soma_quick_meio := 0;
1017                    soma_quick_primeiro := 0;
1018                    soma_quick_ultimo := 0;
1019                    soma_quick_random := 0;
1020                    soma_quick_mediana := 0;
1021                    soma_quick_mediana := 0;
1022                    soma_introsort := 0;
1023                    cont_bolha := 0;
1024                    cont_selection := 0;
1025                    cont_insertion := 0;
1026                    cont_shell := 0;
1027                    cont_heap := 0;
1028                    cont_quick_m := 0;
1029                    cont_quick_p := 0;
1030                    cont_quick_u := 0;
1031                    cont_quick_r := 0;
1032                    cont_quick_med := 0;
1033                    cont_introsort := 0;
1034                    tam := 150000;
1035
1036                    for i:=1 to 100 do
1037                    begin
1038                         vetor := gerar(tam);
```

```
1039                    soma_bolha := soma_bolha + bubblesort(vetor,
                            tam, cont_bolha);
1040                    soma_selection := soma_selection +
                            selectionsort(vetor, tam, cont_selection);
1041                    soma_insertion := soma_insertion +
                            insertionsort(vetor, tam, cont_insertion);
1042                    soma_shell := soma_shell + shellsort(vetor,
                            tam, cont_shell);
1043                    soma_heap := soma_heap + heapsort(vetor, tam,
                            cont_heap);
1044                    soma_quick_meio := soma_quick_meio +
                            quick_meio(vetor, tam, cont_quick_m);
1045                    soma_quick_primeiro:=soma_quick_primeiro +
                            quick_primeiro(vetor, tam, cont_quick_p);
1046                    soma_quick_ultimo:=soma_quick_ultimo +
                            quick_ultimo(vetor, tam, cont_quick_u);
1047                    soma_quick_random:=soma_quick_random +
                            quick_random(vetor, tam, cont_quick_r);
1048                    soma_quick_mediana:=soma_quick_mediana +
                            quick_mediana(vetor, tam, cont_quick_med);
1049                    soma_introsort := soma_introsort + IntroSort(
                            vetor, tam, cont_introsort);
1050                end;
1051
1052                writeln('Media BubbleSort: ',(soma_bolha/100):0:3,'
                        milisegundos || Iteracoes medias : ',(
                        cont_bolha/100):0:0);
1053                writeln('Media SelectionSort: ',(soma_selection
                        /100):0:3,' milisegundos || Iteracoes medias :
                        ',(cont_selection/100):0:0);
1054                writeln('Media InsertionSort: ',(soma_insertion
                        /100):0:3,' milisegundos || Iteracoes medias :
                        ',(cont_insertion/100):0:0);
1055                writeln('Media ShellSort: ',(soma_shell/100):0:3,'
                        milisegundos || Iteracoes medias : ',(
                        cont_shell/100):0:0);
1056                writeln('Media HeapSort: ',(soma_heap/100):0:3,'
                        milisegundos || Iteracoes medias : ',(cont_heap
                        /100):0:0);
1057                writeln('Media QuickSort (meio): ',(soma_quick_meio
                        /100):0:3,' milisegundos || Iteracoes medias :
                        ',(cont_quick_m/100):0:0);
1058                writeln('Media QuickSort (primeiro elemento): ',(
                        soma_quick_primeiro/100):0:3,' milisegundos ||
                        Iteracoes medias : ',(cont_quick_p/100):0:0);
1059                writeln('Media QuickSort (ultimo elemento): ',(
                        soma_quick_ultimo/100):0:3,' milisegundos ||
                        Iteracoes medias : ',(cont_quick_u/100):0:0);
1060                writeln('Media QuickSort (elemento randomico): ',(
                        soma_quick_random/100):0:3,' milisegundos ||
                        Iteracoes medias : ',(cont_quick_r/100):0:0);
1061                writeln('Media QuickSort (mediana): ',(
                        soma_quick_mediana/100):0:3,' milisegundos ||
                        Iteracoes medias : ',(cont_quick_med/100):0:0);
1062                writeln('Media IntroSort: ',(soma_introsort/100)
                        :0:3,' milisegundos || Iteracoes medias : ',(
```

```
                    cont_introsort/100):0:0);
1063                readln;
1064           end;
1065
1066       8 : begin
1067                soma_bolha := 0;
1068                soma_selection := 0;
1069                soma_insertion := 0;
1070                soma_shell := 0;
1071                soma_heap := 0;
1072                soma_quick_meio := 0;
1073                soma_quick_primeiro := 0;
1074                soma_quick_ultimo := 0;
1075                soma_quick_random := 0;
1076                soma_quick_mediana := 0;
1077                soma_quick_mediana := 0;
1078                soma_introsort := 0;
1079                cont_bolha := 0;
1080                cont_selection := 0;
1081                cont_insertion := 0;
1082                cont_shell := 0;
1083                cont_heap := 0;
1084                cont_quick_m := 0;
1085                cont_quick_p := 0;
1086                cont_quick_u := 0;
1087                cont_quick_r := 0;
1088                cont_quick_med := 0;
1089                cont_introsort := 0;
1090                tam := 200000;
1091
1092                for i:=1 to 100 do
1093                begin
1094                     vetor := gerar(tam);
1095                     soma_bolha := soma_bolha + bubblesort(vetor,
                         tam, cont_bolha);
1096                     soma_selection := soma_selection +
                         selectionsort(vetor, tam, cont_selection);
1097                     soma_insertion := soma_insertion +
                         insertionsort(vetor, tam, cont_insertion);
1098                     soma_shell := soma_shell + shellsort(vetor,
                         tam, cont_shell);
1099                     soma_heap := soma_heap + heapsort(vetor, tam,
                         cont_heap);
1100                     soma_quick_meio := soma_quick_meio +
                         quick_meio(vetor, tam, cont_quick_m);
1101                     soma_quick_primeiro:=soma_quick_primeiro +
                         quick_primeiro(vetor, tam, cont_quick_p);
1102                     soma_quick_ultimo:=soma_quick_ultimo +
                         quick_ultimo(vetor, tam, cont_quick_u);
1103                     soma_quick_random:=soma_quick_random +
                         quick_random(vetor, tam, cont_quick_r);
1104                     soma_quick_mediana:=soma_quick_mediana +
                         quick_mediana(vetor, tam, cont_quick_med);
1105                     soma_introsort := soma_introsort + IntroSort(
                         vetor, tam, cont_introsort);
1106                end;
```

```
1107
1108                    writeln('Media BubbleSort: ',(soma_bolha/100):0:3,'
                            milisegundos || Iteracoes medias : ',(
                            cont_bolha/100):0:0);
1109                    writeln('Media SelectionSort: ',(soma_selection
                            /100):0:3,' milisegundos || Iteracoes medias :
                            ',(cont_selection/100):0:0);
1110                    writeln('Media InsertionSort: ',(soma_insertion
                            /100):0:3,' milisegundos || Iteracoes medias :
                            ',(cont_insertion/100):0:0);
1111                    writeln('Media ShellSort: ',(soma_shell/100):0:3,'
                            milisegundos || Iteracoes medias : ',(
                            cont_shell/100):0:0);
1112                    writeln('Media HeapSort: ',(soma_heap/100):0:3,'
                            milisegundos || Iteracoes medias : ',(cont_heap
                            /100):0:0);
1113                    writeln('Media QuickSort (meio): ',(soma_quick_meio
                            /100):0:3,' milisegundos || Iteracoes medias :
                            ',(cont_quick_m/100):0:0);
1114                    writeln('Media QuickSort (primeiro elemento): ',(
                            soma_quick_primeiro/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_p/100):0:0);
1115                    writeln('Media QuickSort (ultimo elemento): ',(
                            soma_quick_ultimo/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_u/100):0:0);
1116                    writeln('Media QuickSort (elemento randomico): ',(
                            soma_quick_random/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_r/100):0:0);
1117                    writeln('Media QuickSort (mediana): ',(
                            soma_quick_mediana/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_med/100):0:0);
1118                    writeln('Media IntroSort: ',(soma_introsort/100)
                            :0:3,' milisegundos || Iteracoes medias : ',(
                            cont_introsort/100):0:0);
1119                    readln;
1120                end;
1121
1122            9 : begin
1123                    soma_bolha := 0;
1124                    soma_selection := 0;
1125                    soma_insertion := 0;
1126                    soma_shell := 0;
1127                    soma_heap := 0;
1128                    soma_quick_meio := 0;
1129                    soma_quick_primeiro := 0;
1130                    soma_quick_ultimo := 0;
1131                    soma_quick_random := 0;
1132                    soma_quick_mediana := 0;
1133                    soma_quick_mediana := 0;
1134                    soma_introsort := 0;
1135                    cont_bolha := 0;
1136                    cont_selection := 0;
1137                    cont_insertion := 0;
1138                    cont_shell := 0;
1139                    cont_heap := 0;
1140                    cont_quick_m := 0;
```

```
1141                    cont_quick_p := 0;
1142                    cont_quick_u := 0;
1143                    cont_quick_r := 0;
1144                    cont_quick_med := 0;
1145                    cont_introsort := 0;
1146                    tam := 250000;
1147
1148                    for i:=1 to 100 do
1149                    begin
1150                         vetor := gerar(tam);
1151                         soma_bolha := soma_bolha + bubblesort(vetor,
                                tam, cont_bolha);
1152                         soma_selection := soma_selection +
                                selectionsort(vetor, tam, cont_selection);
1153                         soma_insertion := soma_insertion +
                                insertionsort(vetor, tam, cont_insertion);
1154                         soma_shell := soma_shell + shellsort(vetor,
                                tam, cont_shell);
1155                         soma_heap := soma_heap + heapsort(vetor, tam,
                                cont_heap);
1156                         soma_quick_meio := soma_quick_meio +
                                quick_meio(vetor, tam, cont_quick_m);
1157                         soma_quick_primeiro:=soma_quick_primeiro +
                                quick_primeiro(vetor, tam, cont_quick_p);
1158                         soma_quick_ultimo:=soma_quick_ultimo +
                                quick_ultimo(vetor, tam, cont_quick_u);
1159                         soma_quick_random:=soma_quick_random +
                                quick_random(vetor, tam, cont_quick_r);
1160                         soma_quick_mediana:=soma_quick_mediana +
                                quick_mediana(vetor, tam, cont_quick_med);
1161                         soma_introsort := soma_introsort + IntroSort(
                                vetor, tam, cont_introsort);
1162                    end;
1163
1164                    writeln('Media BubbleSort: ',(soma_bolha/100):0:3,'
                         milisegundos || Iteracoes medias : ',(
                        cont_bolha/100):0:0);
1165                    writeln('Media SelectionSort: ',(soma_selection
                        /100):0:3,' milisegundos || Iteracoes medias :
                        ',(cont_selection/100):0:0);
1166                    writeln('Media InsertionSort: ',(soma_insertion
                        /100):0:3,' milisegundos || Iteracoes medias :
                        ',(cont_insertion/100):0:0);
1167                    writeln('Media ShellSort: ',(soma_shell/100):0:3,'
                        milisegundos || Iteracoes medias : ',(
                        cont_shell/100):0:0);
1168                    writeln('Media HeapSort: ',(soma_heap/100):0:3,'
                        milisegundos || Iteracoes medias : ',(cont_heap
                        /100):0:0);
1169                    writeln('Media QuickSort (meio): ',(soma_quick_meio
                        /100):0:3,' milisegundos || Iteracoes medias :
                        ',(cont_quick_m/100):0:0);
1170                    writeln('Media QuickSort (primeiro elemento): ',(
                        soma_quick_primeiro/100):0:3,' milisegundos ||
                        Iteracoes medias : ',(cont_quick_p/100):0:0);
1171                    writeln('Media QuickSort (ultimo elemento): ',(
```

```
                    soma_quick_ultimo/100):0:3,' milisegundos ||
                    Iteracoes medias : ',(cont_quick_u/100):0:0);
1172                writeln('Media QuickSort (elemento randomico): ',(
                    soma_quick_random/100):0:3,' milisegundos ||
                    Iteracoes medias : ',(cont_quick_r/100):0:0);
1173                writeln('Media QuickSort (mediana): ',(
                    soma_quick_mediana/100):0:3,' milisegundos ||
                    Iteracoes medias : ',(cont_quick_med/100):0:0);
1174                writeln('Media IntroSort: ',(soma_introsort/100)
                    :0:3,' milisegundos || Iteracoes medias : ',(
                    cont_introsort/100):0:0);
1175                readln;
1176           end;
1177
1178       10 : begin
1179                soma_bolha := 0;
1180                soma_selection := 0;
1181                soma_insertion := 0;
1182                soma_shell := 0;
1183                soma_heap := 0;
1184                soma_quick_meio := 0;
1185                soma_quick_primeiro := 0;
1186                soma_quick_ultimo := 0;
1187                soma_quick_random := 0;
1188                soma_quick_mediana := 0;
1189                soma_quick_mediana := 0;
1190                soma_introsort := 0;
1191                cont_bolha := 0;
1192                cont_selection := 0;
1193                cont_insertion := 0;
1194                cont_shell := 0;
1195                cont_heap := 0;
1196                cont_quick_m := 0;
1197                cont_quick_p := 0;
1198                cont_quick_u := 0;
1199                cont_quick_r := 0;
1200                cont_quick_med := 0;
1201                cont_introsort := 0;
1202                tam := 300000;
1203
1204                for i:=1 to 100 do
1205                begin
1206                     vetor := gerar(tam);
1207                     soma_bolha := soma_bolha + bubblesort(vetor,
                         tam, cont_bolha);
1208                     soma_selection := soma_selection +
                         selectionsort(vetor, tam, cont_selection);
1209                     soma_insertion := soma_insertion +
                         insertionsort(vetor, tam, cont_insertion);
1210                     soma_shell := soma_shell + shellsort(vetor,
                         tam, cont_shell);
1211                     soma_heap := soma_heap + heapsort(vetor, tam,
                         cont_heap);
1212                     soma_quick_meio := soma_quick_meio +
                         quick_meio(vetor, tam, cont_quick_m);
1213                     soma_quick_primeiro:=soma_quick_primeiro +
```

```
                                quick_primeiro(vetor, tam, cont_quick_p);
1214                        soma_quick_ultimo:=soma_quick_ultimo +
                                quick_ultimo(vetor, tam, cont_quick_u);
1215                        soma_quick_random:=soma_quick_random +
                                quick_random(vetor, tam, cont_quick_r);
1216                        soma_quick_mediana:=soma_quick_mediana +
                                quick_mediana(vetor, tam, cont_quick_med);
1217                        soma_introsort := soma_introsort + IntroSort(
                                vetor, tam, cont_introsort);
1218                    end;
1219
1220                    writeln('Media BubbleSort: ',(soma_bolha/100):0:3,'
                            milisegundos || Iteracoes medias : ',(
                            cont_bolha/100):0:0);
1221                    writeln('Media SelectionSort: ',(soma_selection
                            /100):0:3,' milisegundos || Iteracoes medias :
                            ',(cont_selection/100):0:0);
1222                    writeln('Media InsertionSort: ',(soma_insertion
                            /100):0:3,' milisegundos || Iteracoes medias :
                            ',(cont_insertion/100):0:0);
1223                    writeln('Media ShellSort: ',(soma_shell/100):0:3,'
                            milisegundos || Iteracoes medias : ',(
                            cont_shell/100):0:0);
1224                    writeln('Media HeapSort: ',(soma_heap/100):0:3,'
                            milisegundos || Iteracoes medias : ',(cont_heap
                            /100):0:0);
1225                    writeln('Media QuickSort (meio): ',(soma_quick_meio
                            /100):0:3,' milisegundos || Iteracoes medias :
                            ',(cont_quick_m/100):0:0);
1226                    writeln('Media QuickSort (primeiro elemento): ',(
                            soma_quick_primeiro/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_p/100):0:0);
1227                    writeln('Media QuickSort (ultimo elemento): ',(
                            soma_quick_ultimo/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_u/100):0:0);
1228                    writeln('Media QuickSort (elemento randomico): ',(
                            soma_quick_random/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_r/100):0:0);
1229                    writeln('Media QuickSort (mediana): ',(
                            soma_quick_mediana/100):0:3,' milisegundos ||
                            Iteracoes medias : ',(cont_quick_med/100):0:0);
1230                    writeln('Media IntroSort: ',(soma_introsort/100)
                            :0:3,' milisegundos || Iteracoes medias : ',(
                            cont_introsort/100):0:0);
1231                    readln;
1232                end;
1233        end;
1234    readln;
1235    end.
```

aed.pas