Análise de Algoritmos - Ordenação

Gustavo de Souza Silva
Guilherme de Souza Silva
Arthur Xavier
Schumaiquer Souto

Faculdade de Computação
Universidade Federal de Uberlândia

27 de junho de 2017

Lista de Figuras

Lista de Tabelas

Lista de Listagens

Sumário

Capítulo 1

Introdução

Este relatŕio tem como objetivo fazer a análise de diversos algoritmos já conhecidos de ordenação. O intúito desde trabalho é comprovar que as provas matemáticas realmente acontecem em um ambiente real de execução.

1.1 Codificação

O arquivo vetor.c mantém todas as funções a respeito do vetor, como geração, preenchimento, etc.

Listagem 1.1: Arquivo referente ao vetor

```
1  #include <stdlib.h>
2  #include <stdio.h>
3
4  #include "vetor.h"
5
6  #define MAX(x,y) ( \
7      { __auto_type __x = (x); __auto_type __y = (y); \
8        __x > __y ? __x : __y; })
9
10 #define TROCA(v, i, j, temp) (  \
11     { (temp) = v[(i)];      \
12       v[(i)] = v[(j)];      \
13       v[(j)] = (temp);})
14
15 /*
16 typedef enum ordem {ALEATORIO, CRESCENTE, DECRESCENTE} Ordem;
17 typedef enum modificador {TOTALMENTE, PARCIALMENTE} Modificador;
18 typedef int Percentual;
19 */
20
21 double rand_double(double min, double max)
22 { // Retorna números em ponto flutuante aleatórios uniformemente
23   // distribuídos no intervalo fechado [min,max].
24     return min + (rand() / (RAND_MAX / (max-min)));
25 }
26
27 int rand_int(int min, int max){
```

```
28   // Retorna números inteiros aleatórios uniformemente distribuídos
29   // no intervalo fechado [min,max].
30   // Para maiores informações:
31   // https://stackoverflow.com/questions/2509679/how-to-generate-a-random-
         number-from-within-a-range
32   unsigned long num_baldes = (unsigned long) max-min+1;
33   if (num_baldes<1){
34     fprintf(stderr, "Intervalo invalido\n");
35     exit(-1);
36   }
37   unsigned long num_rand  = (unsigned long) RAND_MAX+1;
38   unsigned long tam_balde = num_rand / num_baldes;
39   unsigned long defeito   = num_rand % num_baldes;
40   long x;
41   do
42     x = random();
43   while (num_rand - defeito <= (unsigned long)x);
44   return x / tam_balde + min;
45 }
46
47
48 static void inline preenche_vetor_int(int * v, int n, int k, int q, int r,
       int incr){
49   int i, j;
50       i=0;
51       while (i < n) {
52         for(j=i; j < i+q; j++)
53           v[j] = k;
54
55         i = i + q;
56         if (r > 0) {
57           v[j] = k;
58           i = i + 1;
59           r = r - 1;
60         }
61         k = k + incr;
62       }
63 }
64
65
66 int * gera_vetor_int(int n, Modificador c, Ordem o, Percentual p,
67                      int minimo, int maximo){
68   int i,j; // índices
69   int a = maximo - minimo + 1; // amplitude do intervalo
70   int q = n / a;  // número mínimo de valores repetidos
71   int r = n % a;  // r elementos terão o número (q+1) valores repetidos
72   int k;          // valor do elemento atualmente sob consideração
73   int * v;        // vetor[0..n-1] a ser preenchido
74
75   CONFIRME(n >= 1, "O número de elementos deve ser estritamente positivo.\
         n");
76   CONFIRME(maximo >= minimo, "O valor máximo deve ser maior que o mínimo.\
         n");
77   CONFIRME(0 <= p && p <= 100, "O percentual deve estar entre [0,100]\n");
78
79   v = (int *) calloc(n, sizeof(int)); // aloca um vetor com n inteiros
80   CONFIRME(v != NULL, "calloc falhou\n");
81
82   switch (o) {
```

```
83      case CRESCENTE:
84        preenche_vetor_int(v, n, minimo, q, r, 1);
85        break;
86      case DECRESCENTE:
87        preenche_vetor_int(v, n, maximo, q, r, -1);
88        break;
89      case ALEATORIO:
90        for(i=0; i<n; i++) v[i] = rand_int(minimo,maximo);
91        break;
92      default: CONFIRME(false, "Ordem Inválida\n");
93    }
94
95    switch (c) {
96    case PARCIALMENTE:
97      q = (p * n) / 200;
98      for(i=0;i<q;i++)
99        TROCA(v,i,n-i-1,k);
100     break;
101   case TOTALMENTE: break;
102   default: CONFIRME(false, "Modificador do vetor desconhecido");
103   }
104
105   return v;
106 }
107
108
109 static void inline preenche_vetor_double(double * v, int n, double inicial
      ,
110                                          double delta, double sinal)
111 {
112   int i;
113   for(i=0;i<n;i++)
114     v[i]= inicial + sinal*i*delta;
115 }
116
117
118 double * gera_vetor_double(int n, Modificador c, Ordem o, Percentual p,
119                            double minimo, double maximo){
120   int i; // índice
121   double a = maximo - minimo;     // amplitude do intervalo
122   double delta;
123   double * v;       // vetor[0..n-1] a ser preenchido
124   double temp;
125   int q;
126
127   CONFIRME(n >= 1, "O número de elementos deve ser estritamente positivo.\
        n");
128   CONFIRME(maximo >= minimo, "O valor máximo deve ser maior que o mínimo.\
        n");
129   CONFIRME(0 <= p && p <= 100, "O percentual deve estar entre [0,100]\n");
130
131   delta =  a / MAX(n-1.0, 1.0);    // incremento nos elementos do vetor
132   v = (double *) calloc(n, sizeof(double)); // aloca um vetor com n
        doubles
133   CONFIRME(v != NULL, "callocfalhou\n");
134
135   switch (o) {
136     case CRESCENTE:
137       preenche_vetor_double(v, n, minimo, delta, 1);
```

```
138       break;
139     case DECRESCENTE:
140       preenche_vetor_double(v, n, maximo, delta, -1);
141       break;
142     case ALEATORIO:
143       for(i=0; i<n; i++) v[i] = rand_double(minimo,maximo);
144       break;
145     default: CONFIRME(false, "Ordem Inválida\n");
146   }
147 
148   switch (c) {
149   case PARCIALMENTE:
150     q = (p * n) / 200;
151     for(i=0;i<q;i++)
152       TROCA(v,i,n-i-1,temp);
153     break;
154   case TOTALMENTE: break;
155   default: CONFIRME(false, "Modificador do vetor desconhecido");
156   }
157 
158   return v;
159 }
160 
161 void escreva_vetor_int(int * v, int n, char * arq){
162   int i;
163   FILE* fd = NULL;
164 
165   fd = fopen(arq, "w");
166   CONFIRME(fd!= NULL, "escreva_vetor_int: fopen falhou\n");
167 
168   // Na primeira linha está o número de elementos
169   fprintf(fd, "%d\n", n);
170   for(i=0; i<n; i++)
171     fprintf(fd, "%d\n", v[i]);
172   fclose(fd);
173 }
174 
175 void escreva_vetor_double(double * v, int n, char * arq){
176   int i;
177   FILE* fd = NULL;
178 
179   fd = fopen(arq, "w");
180   CONFIRME(fd!= NULL, "escreva_vetor_double: fopen falhou\n");
181 
182   // Na primeira linha está o número de elementos
183   fprintf(fd, "%d\n", n);
184   for(i=0; i<n; i++)
185     fprintf(fd, "%f\n", v[i]);
186   fclose(fd);
187 }
188 
189 int * leia_vetor_int(char * arq, int * n){
190   int i;
191   FILE* fd = NULL;
192   int * v;
193 
194   fd = fopen(arq, "r");
195   CONFIRME(fd!= NULL, "leia_vetor_int: fopen falhou\n");
196 
```

```
197   // Leia o número de elementos do vetor
198   CONFIRME(fscanf(fd, "%d\n", n) == 1,
199            "leia_vetor_int: erro ao ler o número de elementos do vetor\n")
                 ;
200
201   v = (int *) calloc(*n, sizeof(int)); // aloca um vetor com n inteiros
202   CONFIRME(v != NULL, "leia_vetor_int: calloc falhou\n");
203
204   i=0;
205   while(fscanf(fd, "%d\n", &v[i]) == 1) i++;
206   fclose(fd);
207
208   return v;
209 }
210
211 double * leia_vetor_double(char * arq, int * n){
212   int i;
213   FILE* fd = NULL;
214   double * v;
215
216   fd = fopen(arq, "r");
217   CONFIRME(fd!= NULL, "leia_vetor_int: fopen falhou\n");
218
219   // Leia o número de elementos do vetor
220   CONFIRME(fscanf(fd, "%d\n", n) == 1,
221            "leia_vetor_double: erro ao ler o número de elementos do vetor\
                 n");
222
223
224  // Aloca um vetor com n doubles
225   v = (double *) calloc(*n, sizeof(double));
226   CONFIRME(v != NULL, "leia_vetor_int: calloc falhou\n");
227
228   i=0;
229   while(fscanf(fd, "%lf\n", &v[i]) == 1) i++;
230   fclose(fd);
231
232   return v;
233 }
234
235 bool esta_ordenado_int(Ordem o, int * v, int n){
236   int i;
237
238   CONFIRME(n > 0,
239            "estaOrdenado_int: o número de elementos deve ser maior que
                 zero.\n");
240   if (n == 1) return true;
241   switch (o) {
242     case CRESCENTE:
243       for(i=0;i<n;i++)
244         if (v[i-1] > v[i])
245           return false;
246       break;
247     case DECRESCENTE:
248       for(i=0;i<n;i++)
249         if (v[i-1] < v[i])
250           return false;
251       break;
252     default: CONFIRME(false, "estaOrdenado_int: Ordem Inválida\n");
```

```
253   }
254   return true;
255 }
256
257
258 bool esta_ordenado_double(Ordem o, double * v, int n){
259   int i;
260
261   CONFIRME(n > 0,
262            "estaOrdenado_double: o número de elementos deve ser maior que
                 zero.\n");
263   if (n == 1) return true;
264   switch (o) {
265     case CRESCENTE:
266       for(i=1;i<n;i++)
267         if (v[i-1] > v[i]){
268           printf("valor V[%d] = %lf eh maior que V[%d] = %lf",i-1,v[i-1],i
                  ,v[i]);
269           return false;
270         }
271       break;
272     case DECRESCENTE:
273       for(i=1;i<n;i++)
274         if (v[i-1] < v[i]){
275           printf("valor V[%d] = %lf eh menor que V[%d] = %lf",i-1,v[i-1],i
                  ,v[i]);
276           return false;
277           }
278       break;
279     default: CONFIRME(false, "estaOrdenado_double: Ordem Inválida\n");
280   }
281   return true;
282 }
283
284 void imprime_vetor_int(int * v, int n){
285   int i;
286
287   for(i=0; i < n; i++)
288     printf("v[%d] = %d\n", i, v[i]);
289   printf("\n");
290 }
291
292 void imprime_vetor_double(double * v, int n){
293   int i;
294
295   for(i=0; i < n; i++)
296     printf("v[%d] = %lf\n", i, v[i]);
297   printf("\n");
298 }
```

Este arquivo serve para gerar os vetores e salva-los em arquivos.

Listagem 1.2: Geração dos vetores

```
1 #include <stdlib.h>
2 #include <stdio.h>
3 #include <string.h>
4 #include <math.h>
5 #include <sys/types.h>
```

```c
#include <sys/stat.h>
#include <unistd.h>

#include "vetor.h"

#define POT2(n) (1 << (n))


void gera_e_salva_vet(int n, Modificador m, Ordem o, Percentual p){
  int * v = NULL;
  char nome_do_arquivo[64];
  char sufixo[10];

  switch (o){
    case ALEATORIO:
      sprintf(nome_do_arquivo, "vIntAleatorio_%d", n);
      break;
    case CRESCENTE:
      sprintf(nome_do_arquivo, "vIntCrescente_%d", n);
      break;
    case DECRESCENTE:
      sprintf(nome_do_arquivo, "vIntDecrescente_%d", n);
      break;
    default: CONFIRME(false,
                      "gera_e_salva_vet: Ordenação desconhecida");
  }

  if (p > 0)
    sprintf(sufixo, "_P%2d.dat", p);
  else
    strcpy(sufixo, ".dat");

  v = gera_vetor_int(n, m, o, p, 1, n);
  strcat(nome_do_arquivo, sufixo);
  escreva_vetor_int(v, n, nome_do_arquivo);
  free(v);
}


int main(int argc, char *argv[]){
  int n = 0;
  int p = 0;
  char diretorio[256];

  struct stat st = {0};


  if (argc == 2)
    strcpy(diretorio, argv[1]);
  else
    strcpy(diretorio, "./vetores");

  if (stat(diretorio, &st) == -1) { // se o diretorio não existir,
    mkdir(diretorio, 0700);         // crie um
  }

  CONFIRME(chdir(diretorio) == 0, "Erro ao mudar de diretório");

  for(n = POT2(4); n <= POT2(14); n <<= 1){
```

```
65      gera_e_salva_vet(n, TOTALMENTE, ALEATORIO,    0);
66      gera_e_salva_vet(n, TOTALMENTE, CRESCENTE,    0);
67      gera_e_salva_vet(n, TOTALMENTE, DECRESCENTE, 0);
68
69      for(p=10; p <= 50; p += 10){
70        gera_e_salva_vet(n, PARCIALMENTE, CRESCENTE,    p);
71        gera_e_salva_vet(n, PARCIALMENTE, DECRESCENTE, p);
72      }
73      printf("Vetores para n = %d gerados.\n", n);
74    }
75
76    CONFIRME(chdir("..") == 0, "Erro ao mudar de diretório");
77
78    exit(0);
79  }
```

Este arquivo contém os algoritmos de ordenação pedidos.

Listagem 1.3: Métodos de ordenação

```
1  #include <stdio.h>
2  #include <stdlib.h>
3  #include "vetor.h"
4  #include <math.h>
5  void intercala(int * v,int p, int q, int r);
6  static void inline troca(int *A, int i, int j){
7    int temp;
8    temp = A[i];
9    A[i] = A[j];
10   A[j] = temp;
11 }
12
13 void ordena_por_bolha(int *A, int n){
14   int i,j;
15
16   if (n<2) return;
17
18   for(i=0; i<n; i++)
19     for(j=0; j<n-1; j++)
20       if (A[j] > A[j+1])
21         troca(A,j,j+1);
22 }
23
24
25 void ordena_por_shell(int *A, int n){
26   // Sequência de lacunas de Marcin Ciura
27   // Ref: https://en.wikipedia.org/wiki/Shellsort
28   int lacunas[] = {701, 301, 132, 57, 23, 10, 4, 1};
29   int *lacuna;
30   int i, j, temp;
31
32   for(lacuna=lacunas; *lacuna > 0; lacuna++){
33     for(i=*lacuna; i < n; i++){
34       // adicione A[i] aos elementos que foram ordenados
35       // guarde A[i] em temp e crie um espaço na posição i
36       temp = A[i];
37       // Desloque os elementos previamente ordenados até
38       // que a posição correta para A[i] seja encontrada
39       for(j=i; j >= *lacuna && A[j - *lacuna] > temp; j -= *lacuna){
```

```
40          A[j] = A[j - *lacuna];
41        }
42        // Coloque temp (o A[i] original) em sua posição correta
43        A[j] = temp;
44      }
45    }
46 }
47
48 void ordena_intercala(int * v,int p,int r)
49 {
50     int q;
51     if (p < r) {
52         q = floor ((p + r) / 2); // retorna o chão dessa operação
53         ordena_intercala (v, p, q);
54         ordena_intercala(v, q + 1, r);
55         intercala(v, p, q, r);
56     }
57 }
58
59 void intercala(int * v,int p, int q, int r)
60 {
61     int *B = calloc((r+1),sizeof(int));
62     int i,k,j;
63     for (i = p; i<=q; i++)
64         B[i] = v[i];
65     for (j = (q + 1); j<= r;j++) {
66         B[(r + q + 1 - j)] = v[j];
67     }
68     i = p;
69     j = r;
70     for(k = p; k<=r; k++) {
71             if (B[i] <= B[j]) {
72                 v[k] = B[i];
73                 i++;
74             } else {
75                 v[k] = B[j];
76                 j--;
77             }
78     }
79   free(B);
80 }
81
82 void insertion(int *v, int tam)
83 {
84     int chave,i,j;
85     for(j=1;j<tam;j++)
86     {
87         chave = v[j];
88         i = j - 1;
89         while (i >= 0 && v[i] > chave)
90         {
91             v[i+1] = v[i];
92             i = i-1;
93         }
94   v[i+1] = chave;
95     }
96 }
97
98 void heap(int *a, int n) {
```

```
99      int i = n / 2, pai, filho, t;
100     for (;;) {
101         if (i > 0) {
102             i--;
103             t = a[i];
104         } else {
105             n--;
106             if (n == 0) return;
107             t = a[n];
108             a[n] = a[0];
109         }
110         pai = i;
111         filho = i * 2 + 1;
112         while (filho < n) {
113             if ((filho + 1 < n)  &&  (a[filho + 1] > a[filho]))
114                 filho++;
115             if (a[filho] > t) {
116                 a[pai] = a[filho];
117                 pai = filho;
118                 filho = pai * 2 + 1;
119             } else {
120                 break;
121             }
122         }
123         a[pai] = t;
124     }
125 }
126
127 void quick(int *vetor, int inicio, int fim){
128
129     int pivo, aux, i, j, meio;
130
131     i = inicio;
132     j = fim;
133
134     meio = (int) ((i + j) / 2);
135     pivo = vetor[meio];
136
137     do{
138         while (vetor[i] < pivo) i = i + 1;
139         while (vetor[j] > pivo) j = j - 1;
140
141         if(i <= j){
142             aux = vetor[i];
143             vetor[i] = vetor[j];
144             vetor[j] = aux;
145             i = i + 1;
146             j = j - 1;
147         }
148     }while(j > i);
149
150     if(inicio < j) quick(vetor, inicio, j);
151     if(i < fim) quick(vetor, i, fim);
152 }
153
154 void radix(int *vetor, int tamanho){
155     int i;
156     int *b;
157     int maior = vetor[0];
```

```
158     int exp = 1;
159
160     b = (int *)calloc(tamanho, sizeof(int));
161
162     for (i = 0; i < tamanho; i++) {
163         if (vetor[i] > maior)
164             maior = vetor[i];
165     }
166
167     while (maior/exp > 0) {
168         int bucket[10] = { 0 };
169         for (i = 0; i < tamanho; i++)
170             bucket[(vetor[i] / exp) % 10]++;
171         for (i = 1; i < 10; i++)
172             bucket[i] += bucket[i - 1];
173         for (i = tamanho - 1; i >= 0; i--)
174             b[--bucket[(vetor[i] / exp) % 10]] = vetor[i];
175         for (i = 0; i < tamanho; i++)
176             vetor[i] = b[i];
177         exp *= 10;
178     }
179     free(b);
180 }
181
182 void coutingsort(int *A, int tamanho){
183     int k = tamanho;
184     int aux;
185     int *C = (int*)calloc(k+1,sizeof(int));
186     int *B = (int*)malloc(tamanho*sizeof(int));
187
188     for(int j = 0;j<tamanho;j++){
189         C[A[j]]++;
190     }
191     for(int i=1;i<=k;i++){
192         C[i] = C[i] + C[i-1];
193     }
194     for(int j=0;j<tamanho;j++){
195         B[C[A[j]]-1] = A[j];
196         C[A[j]]--;
197     }
198     for(int i=0;i<tamanho;i++){
199         A[i] = B[i];
200     }
201 }
202
203 void insertiondouble(double *v, int tam)
204 {
205     int i,j;
206     double chave;
207     for(j=1;j<tam;j++)
208     {
209         chave = v[j];
210         i = j - 1;
211         while (i >= 0 && v[i] > chave)
212         {
213             v[i+1] = v[i];
214             i = i-1;
215         }
216   v[i+1] = chave;
```

```
217     }
218 }
219
220 void bucketsort(double *A,int tamanho){
221     bucket *C = (bucket*)malloc(10*sizeof(bucket));
222     int j,i;
223     for(int i=0;i<10;i++){ //Inicialização dos topos dos baldes
224         C[i].topo = 0.0;
225         C[i].balde = (double*)malloc((int)(tamanho)*sizeof(double));
226     }
227     for(i = 0;i<tamanho;i++){ //Verifica em que balde o elem deve ficar
228         j = 10-1;
229         while(1){
230             if(j<0){
231                 break;
232             }
233             if(A[i]>=j*10){
234                 C[j].balde[C[j].topo] = A[i];
235                 (C[j].topo)++;
236                 break;
237             }
238             j--;
239         }
240     }
241     for(i=0;i<10;i++){ //ordena os baldes
242         if(C[i].topo){
243             insertiondouble(C[i].balde,C[i].topo);
244         }
245     }
246     i=0;
247     for(j=0;j<10;j++){ //coloca os elementos dos baldes de volta no vetor
248         for(int k=0;k<C[j].topo;k++){
249             A[i]=C[j].balde[k];
250             i++;
251         }
252     }
253     for(i=0;i<10;i++){
254         free(C[i].balde);
255     }
256     free(C);
257 }
```

O arquivo ensaios.c serve para automatizar e calcular os tempos de cada método de ordenação.

Listagem 1.4: Automatização dos experimentos

```
1  #include <stdlib.h>
2  #include <stdio.h>
3  #include <string.h>
4  #include <stdint.h>
5  #include <time.h>
6  #include <float.h>
7  #include <math.h>
8  #include <sys/types.h>
9  #include <sys/stat.h>
10 #include <unistd.h>
11
12 #include "vetor.h"
```

```
13 #include "ordena.h"
14
15 #define BILHAO 1000000000L
16
17 #define CRONOMETRA(funcao,vetor,n) {                              \
18    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &inicio);              \
19    funcao(vetor,0,n);                                               \
20    clock_gettime(CLOCK_PROCESS_CPUTIME_ID, &fim);                 \
21    tempo_de_cpu_aux = BILHAO * (fim.tv_sec - inicio.tv_sec) +     \
22                   fim.tv_nsec - inicio.tv_nsec;                  \
23    }
24
25 int main(int argc, char *argv[]){
26   int * v = NULL;
27   int n = 0;
28   uint64_t tempo_de_cpu_aux = 0;
29   int tamanho = 0,count = 0;
30   //clock_t inicio, fim;
31   struct timespec inicio, fim;
32   uint64_t tempo_de_cpu = 0.0;
33   char msg[256];
34   char nome_do_arquivo[128];
35   char **arquivos;
36   int k=0,h = 0;
37   arquivos = (char**)malloc(200*sizeof(char*));
38   for(int i=0;i<200;i++){
39     arquivos[i] = (char*)malloc(128*sizeof(char));
40   }
41
42   for(int i=0;i<11;i++){
43     sprintf(nome_do_arquivo,"vetores/vIntAleatorio_%d.dat",(int)pow(2,i
         +4%15));
44     strcpy(arquivos[k], nome_do_arquivo);
45     k++;
46   }
47   for(int i=0;i<11;i++){
48     sprintf(nome_do_arquivo,"vetores/vIntCrescente_%d.dat",(int)pow(2,i
         +4%15));
49     strcpy(arquivos[k], nome_do_arquivo);
50     k++;
51
52   }
53   for(int i=0;i<11;i++){
54     sprintf(nome_do_arquivo,"vetores/vIntCrescente_%d_P10.dat",(int)pow(2,
         i+4%15));
55     strcpy(arquivos[k], nome_do_arquivo);
56     k++;
57   }
58   for(int i=0;i<11;i++){
59     sprintf(nome_do_arquivo,"vetores/vIntCrescente_%d_P20.dat",(int)pow(2,
         i+4%15));
60     strcpy(arquivos[k], nome_do_arquivo);
61     k++;
62   }
63   for(int i=0;i<11;i++){
64     sprintf(nome_do_arquivo,"vetores/vIntCrescente_%d_P30.dat",(int)pow(2,
         i+4%15));
65     strcpy(arquivos[k], nome_do_arquivo);
66     k++;
```

```
67    }
68    for(int i=0;i<11;i++){
69      sprintf(nome_do_arquivo,"vetores/vIntCrescente_%d_P40.dat",(int)pow(2,
          i+4%15));
70      strcpy(arquivos[k], nome_do_arquivo);
71      k++;
72    }
73    for(int i=0;i<11;i++){
74      sprintf(nome_do_arquivo,"vetores/vIntCrescente_%d_P50.dat",(int)pow(2,
          i+4%15));
75      strcpy(arquivos[k], nome_do_arquivo);
76      k++;
77    }
78    for(int i=0;i<11;i++){
79      sprintf(nome_do_arquivo,"vetores/vIntDecrescente_%d.dat",(int)pow(2,i
          +4%15));
80      strcpy(arquivos[k], nome_do_arquivo);
81      k++;
82    }
83    for(int i=0;i<11;i++){
84      sprintf(nome_do_arquivo,"vetores/vIntDecrescente_%d_P10.dat",(int)pow
          (2,i+4%15));
85      strcpy(arquivos[k], nome_do_arquivo);
86      k++;
87    }
88    for(int i=0;i<11;i++){
89      sprintf(nome_do_arquivo,"vetores/vIntDecrescente_%d_P20.dat",(int)pow
          (2,i+4%15));
90      strcpy(arquivos[k], nome_do_arquivo);
91      k++;
92    }
93
94    for(int i=0;i<11;i++){
95        sprintf(nome_do_arquivo,"vetores/vIntDecrescente_%d_P30.dat",(int)
            pow(2,i+4%15));
96      strcpy(arquivos[k], nome_do_arquivo);
97      k++;
98    }
99    for(int i=0;i<11;i++){
100     sprintf(nome_do_arquivo,"vetores/vIntDecrescente_%d_P40.dat",(int)pow
          (2,i+4%15));
101     strcpy(arquivos[k], nome_do_arquivo);
102     k++;
103   }
104   for(int i=0;i<11;i++){
105     sprintf(nome_do_arquivo,"vetores/vIntDecrescente_%d_P50.dat",(int)pow
          (2,i+4%15));
106     strcpy(arquivos[k], nome_do_arquivo);
107     k++;
108   }
109     printf("%d\n",k);
110   //strcpy(nome_do_arquivo, "vetores/vIntCrescente_131072.dat");
111   // Leia o vetor a partir do arquivo
112   //v = leia_vetor_int(nome_do_arquivo, &n);
113   printf("%s\n",arquivos[11]);
114   for(int i=0;i<k;i++){
115     tempo_de_cpu = 0.0;
116     if(h > 10){
117         h = 0;
```

```
118     }
119     for(int j=0;j<3;j++){
120         v = leia_vetor_int(arquivos[i],&n);
121         tamanho = (int)pow(2,h+4%15);
122         /*inicio = clock();
123         //ordena_por_bolha(v,n);
124         insertion(v,tamanho);
125         fim = clock();*/
126   CRONOMETRA(ordena_intercala, v, tamanho-1);
127         //tempo_de_cpu += ((double) (fim - inicio)) / CLOCKS_PER_SEC;
128   tempo_de_cpu += tempo_de_cpu_aux;
129     }
130     if(esta_ordenado_int(CRESCENTE,v,n) && count < 11){
131         printf("Tempo do vetor aleatorio tamanho %d: %llu\n",tamanho,(long
              long unsigned int)tempo_de_cpu/(uint64_t) 3);
132     }
133     else if(esta_ordenado_int(CRESCENTE,v,n) && count < 22){
134         printf("Tempo do vetor Crescente tamanho %d: %llu\n",tamanho,(long
              long unsigned int)tempo_de_cpu/(uint64_t) 3);
135     }
136     else if(esta_ordenado_int(CRESCENTE,v,n) && count < 33){
137         printf("Tempo do vetor Crescente P10 tamanho %d: %llu\n",tamanho,(
              long long unsigned int)tempo_de_cpu/(uint64_t) 3);
138     }
139     else if(esta_ordenado_int(CRESCENTE,v,n) && count < 44){
140         printf("Tempo do vetor Crescente P20 tamanho %d: %llu\n",tamanho,(
              long long unsigned int)tempo_de_cpu/(uint64_t) 3);
141     }
142     else if(esta_ordenado_int(CRESCENTE,v,n) && count < 55){
143         printf("Tempo do vetor Crescente P30 tamanho %d: %llu\n",tamanho,(
              long long unsigned int)tempo_de_cpu/(uint64_t) 3);
144     }
145     else if(esta_ordenado_int(CRESCENTE,v,n) && count < 66){
146         printf("Tempo do vetor Crescente P40 tamanho %d: %llu\n",tamanho,(
              long long unsigned int)tempo_de_cpu/(uint64_t) 3);
147     }
148     else if(esta_ordenado_int(CRESCENTE,v,n) && count < 77){
149         printf("Tempo do vetor Crescente P50 tamanho %d: %llu\n",tamanho,(
              long long unsigned int)tempo_de_cpu/(uint64_t) 3);
150     }
151     else if(esta_ordenado_int(CRESCENTE,v,n) && count < 88){
152         printf("Tempo do vetor Decrescente tamanho %d: %llu\n",tamanho,(
              long long unsigned int)tempo_de_cpu/(uint64_t) 3);
153     }
154     else if(esta_ordenado_int(CRESCENTE,v,n) && count < 99){
155         printf("Tempo do vetor Decrescente P10 tamanho %d: %llu\n",tamanho
              ,(long long unsigned int)tempo_de_cpu/(uint64_t) 3);
156     }
157     else if(esta_ordenado_int(CRESCENTE,v,n) && count < 110){
158         printf("Tempo do vetor Decrescente P20 tamanho %d: %llu\n",tamanho
              ,(long long unsigned int)tempo_de_cpu/(uint64_t) 3);
159     }
160     else if(esta_ordenado_int(CRESCENTE,v,n) && count < 121){
161         printf("Tempo do vetor Decrescente P30 tamanho %d: %llu\n",tamanho
              ,(long long unsigned int)tempo_de_cpu/(uint64_t) 3);
162     }
163     else if(esta_ordenado_int(CRESCENTE,v,n) && count < 132){
164         printf("Tempo do vetor Decrescente P40 tamanho %d: %llu\n",tamanho
              ,(long long unsigned int)tempo_de_cpu/(uint64_t) 3);
```

```
165     }
166     else if(esta_ordenado_int(CRESCENTE,v,n) && count < 143){
167         printf("Tempo do vetor Decrescente P50 tamanho %d: %llu\n",tamanho
                ,(long long unsigned int)tempo_de_cpu/(uint64_t) 3);
168     }
169     else{
170         printf("Erro em ordenção do vetor %d, arquivo %s\n",i,arquivos[i])
                ;
171     }
172     h++;
173     count++;
174   }
175   //imprime_vetor_int(v,16384);
176   free(v);
177   exit(0);
178 }
```

1.1.1 Comandos

Os seguintes passos devem ser seguidos para criação dos vetores que serão utilizados no experimento: 1 - Compilar o arquivo vetor.c;

```
> gcc -O3 -c vetor.c
```

2 - Compilar o programar que gera os vetores e os coloca no diretório determinado;

```
> gcc -O3 vetor.o gera_vets.c -o gera_vets.exe
```

3 - Para usá-lo digite

```
> ./gera_vets.exe
```

Os passos a seguir são para execução do experimento> 1 - Verifique a existência do diretório contendo os vetores, e então digite o seguinte comando:

```
> gcc -O3 -c ordena.c
```

2 - Agora é necessário compilar o arquivo de ensaio e tudo que será utilizado

```
> gcc -O3 vetor.o ordena.o ensaios.c -o ensaios.exe
```

3 - Para executar digite:

```
> ./ensaios.exe
```

Capítulo 2

Gráficos de Funções Matemáticas

Colocar os gráficos de n nlogn e talz.

Capítulo 3

Insertion Sort

Texto sobre o insertion

3.1 Insertion Sort - Vetor Aleatório

Um pequeno texto falando sobre o vetor totalmente aleatório e o insertion.

Tabela 3.1: *Insertion Sort com Vetor aleatório*

Número de Elementos	**Tempo de execução em nanosegundos**
16	592
32	623
64	1330
128	3921
256	13475
512	49717
1024	181720
2048	709142
4096	2818906
8192	11332358
16384	44220895

3.1.1 Gráfico Insertion sort - Vetor Aletório

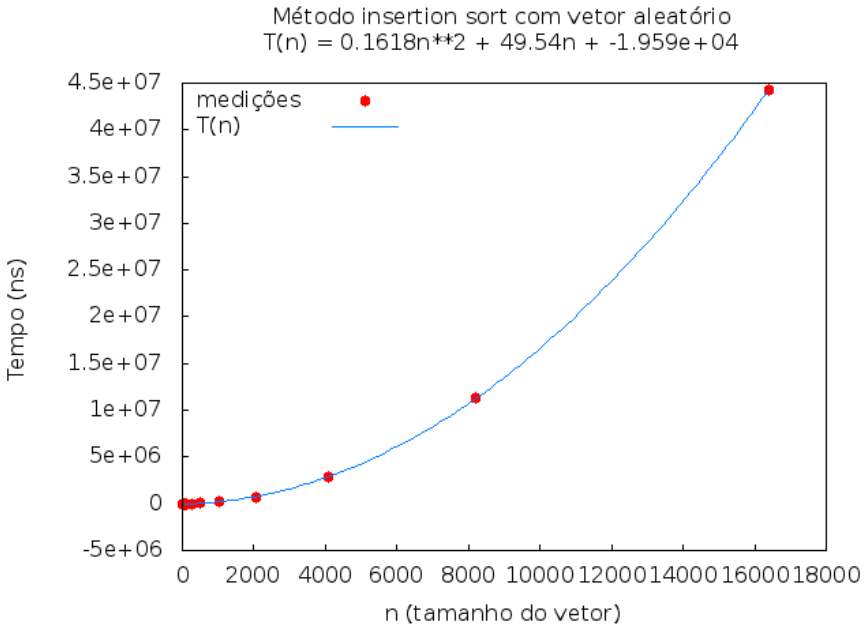

Figura 3.1: *Gráfico Insertion Sort - Vetor Aleatorio*

3.2 Insertion Sort - Vetor Crescente

Pequeno texto sobre o vetor

Tabela 3.2: *Insertion Sort com Vetor ordenado em ordem crescente*

Número de Elementos	Tempo de execução em nanosegundos
16	330
32	359
64	366
128	441
256	653
512	1151
1024	1616
2048	3006
4096	5551
8192	11105
16384	21993

3.2.1 Grafico Insertio sort - Vetor Crescente

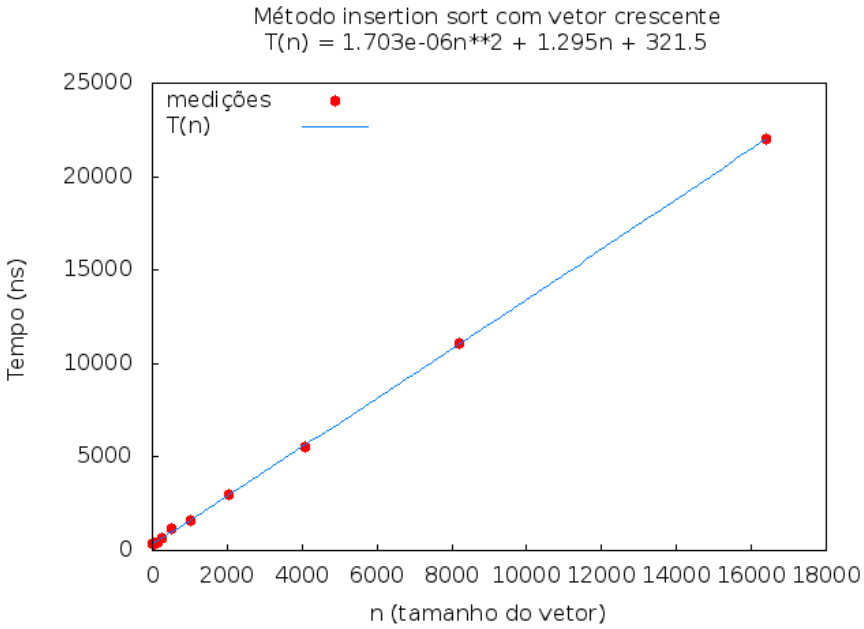

Figura 3.2: *Gráfico Insertion Sort - Vetor Crescente*

Capítulo 4

Merge Sort