



Universidade Federal de São João del Rei
Departamento de Ciência da Computação
Curso de Ciência da Computação

Roteiro 10

Adélson de Oliveira Carmo Júnior
212050019

1 Algoritmos de Ordenação

1.1 Reimplantação

Código

```
1 #ifndef ORDENACAO_PT1
2 #define ORDENACAO_PT1
3
4 int* copiaVetor(int* v, int n);
5 void imprimeVetor(int* v, int n);
6 void preencheAleatorio(int* v, int n,
    int ini, int fim);
7 void troca(int* a, int *b);
8 void criaHeap(int *v, int pai, int fim);
9 void HeapSort(int *v, int n);
10 void Merge(int *v, int ini, int meio,
    int fim);
11 void MergeSort(int *v, int ini, int
    fim);
12 int particao(int *v, int ini, int fim);
13 void QuickSort(int *v, int ini, int
    fim, int n);
14 int getComp();
15 int getMov();
16 void setComp(int valor);
17 void setMov(int valor);
18
19 #endif
    codigos/questao11/questao11.h
```

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include "questao11.h"
5
6 //Medidas de Complexidade
7 int comp; //Num. de comparacoes
8 int mov; //Num. de movimentacoes
9
10 int getComp(){
11     return comp;
12 }
13
14 int getMov(){
15     return mov;
16 }
17
18 void setComp(int valor){
19     comp = valor;
20 }
21
22 void setMov(int valor){
23     mov = valor;
24 }
25
26 int* copiaVetor(int* v, int n){
27     int i;
28     int *v2;
29     v2 = (int*) malloc (n*sizeof(int));
30     for(i=0; i<n; i++) v2[i] = v[i];
31     return v2;
32 }
33
34 void imprimeVetor(int* v, int n){
35     int i, prim = 1;
36     printf("[");
37     for(i=0; i<n; i++)
38         if(prim){ printf("%d", v[i]);
39             prim = 0; }
40         else printf(", %d", v[i]);
41     printf("]");
42 }
```

```

40     printf("\n");
41 }
42
43 void preencheAleatorio(int* v, int n,
44     int ini, int fim){
45     int i;
46     for(i=0; i<n; i++){
47         v[i] = ini + rand() % (fim-ini
48             + 1);
49         //v[i] = (n-i); //Para o pior
50         caso
51     }
52 }
53 void troca(int* a, int *b){
54     int aux = *a;
55     *a = *b;
56     *b = aux;
57 }
58 void criaHeap(int *v, int pai, int fim){
59     int aux = v[pai];
60     int filho = 2*pai + 1;
61     while(filho <= fim){
62         if(filho < fim)
63             if(v[filho] < v[filho+1])
64                 filho++;
65         if(aux < v[filho]){
66             v[pai] = v[filho];
67             pai = filho;
68             filho = 2*pai + 1;
69         }else filho = fim + 1;
70     }
71     v[pai] = aux;
72 }
73 void HeapSort(int *v, int n){
74     int i;
75     for(i=(n-1)/2; i>=0; i--){
76         criaHeap(v, i, n-1);
77     }
78     for(i=n-1; i>=1; i--){
79         troca(&v[0], &v[i]);
80         criaHeap(v, 0, i-1);
81     }
82 }
83 void Merge(int *v, int ini, int meio,
84     int fim){
85     int tam = fim-ini+1;
86     //Vetor Auxiliar - A
87     int *A = (int*) malloc
88         (tam*sizeof(int));
89     int i = ini, j = meio+1, k = 0;
90     while (i<=meio && j<=fim) {
91         if (v[i] < v[j]){ A[k] = v[i];
92             i++; }
93         else { A[k] = v[j]; j++; }
94         k++;
95     }
96     while (i<=meio) { A[k] = v[i];
97         i++; k++; }
98     while (j<=fim) { A[k] = v[j];
99         j++; k++; }
100 }
101 void MergeSort(int *v, int ini, int
102     fim){
103     if(ini < fim ){
104         int meio = (ini + fim)/2;
105         MergeSort(v, ini, meio);
106         MergeSort(v, meio+1, fim);
107         Merge(v, ini, meio, fim);
108     }
109 }
110 int particao(int *v, int ini, int fim){
111     int i = ini, j = fim;
112     int pivo = v[(ini+fim)/2];
113     while (1) {
114         comp++;
115         while(v[i] < pivo){ i++;
116             comp++; } //procura algum >=
117             pivo do lado esquerdo
118         while(v[j] > pivo){ j--;
119             comp++; } //procura algum <=
120             pivo do lado direito
121     }
122     if(i<j){
123         troca(&v[i], &v[j]);
124         //troca os elementos
125         encontrados
126         mov++;
127         i++;
128         j--;
129     }else
130         return j; //retorna o local
131         onde foi feita a particao
132 }
133 void QuickSort(int *v, int ini, int
134     fim, int n){
135     if(ini < fim ){
136         int q = particao(v, ini, fim);
137         //printf("Parts: (%d, %d) e
138             (%d, %d): ", ini, q, q+1,
139             fim);
140         //imprimeVetor(v, n);
141         QuickSort(v, ini, q, n);
142         QuickSort(v, q+1, fim, n);
143     }
144 }

```

codigos/questao11/questao11.c

```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include "questao11.h"
5
6  int main(){
7
8      //Atribuicoes iniciais
9      srand(time(NULL));
10     setComp(0);
11     setMov(0);
12     clock_t t;
13
14     int *v1, *v2, *v3;
15     int n;
16
17     /* Recebe valores e preenche */
18     printf("Digite o tamanho do
19           vetor:\n");
20     scanf("%d", &n);
21     v1 = (int*) malloc (n*sizeof(int));
22     preencheAleatorio(v1, n, 1, 100);
23     imprimeVetor(v1, n);
24
25     v2 = copiaVetor(v1, n);
26     v3 = copiaVetor(v1, n);
27
28     /* MergeSort */
29     t = clock();
30     MergeSort(v1, 0, n-1);
31     t = clock() - t;
32     printf("\nInformacoes da Ordenacao
33           por MergeSort:\n");
34     printf("Tempo Execucao: %f
35           seconds.\n",
36           ((float)t)/CLOCKS_PER_SEC);
37     printf("Comparacoes: %d\n",
38           getComp());
39     printf("Movimentacoes: %d\n",
40           getMov());
41     printf("Memoria (bytes): %ld\n",
42           n*sizeof(int));
43
44     imprimeVetor(v1, n);
45
46     /* HeapSort */
47     setComp(0);
48     setMov(0);
49     t = clock();
50     HeapSort(v2, n);
51     t = clock() - t;
52     printf("\nInformacoes da Ordenacao
53           por HeapSort:\n");
54     printf("Tempo Execucao: %f
55           seconds.\n",
56           ((float)t)/CLOCKS_PER_SEC);
57     printf("Comparacoes: %d\n",
58           getComp());
59     printf("Movimentacoes: %d\n",
60           getMov());
61     printf("Memoria (bytes): %ld\n",
62           n*sizeof(int));
63
64     imprimeVetor(v2, n);
65
66     /* QuickSort */
67     setComp(0);
68     setMov(0);
69     t = clock();
70     QuickSort(v3, 0, n-1, n);
71     t = clock() - t;
72     printf("\nInformacoes da Ordenacao
73           por QuickSort:\n");
74     printf("Tempo Execucao: %f
75           seconds.\n",
76           ((float)t)/CLOCKS_PER_SEC);
77     printf("Comparacoes: %d\n",
78           getComp());
79     printf("Movimentacoes: %d\n",
80           getMov());
81     printf("Memoria (bytes): %ld\n",
82           n*sizeof(int));
83
84     imprimeVetor(v3, n);
85
86     free(v1);
87     free(v2);
88     free(v3);
89
90     return 0;
91 }

```

codigos/questao11/main.c

```

1  all: questao11.o
2     gcc questao11.o main.c -o main
3
4  questao11.o: questao11.h questao11.c
5     gcc -c questao11.c

```

```

6
7  clean:
8     rm -f questao11.o main

```

codigos/questao11/Makefile

```

Digite o tamanho do vetor:
10
[73, 86, 11, 37, 14, 71, 16, 31, 85, 37]

Informacoes da Ordenacao por MergeSort:
Tempo Execucao: 0.000024 seconds.
Comparacoes: 0
Movimentacoes: 0
Memoria (bytes): 40
[11, 14, 16, 31, 37, 37, 71, 73, 85, 86]

```

Figura 1: Questão 1.1 - Saída 1

```

Informacoes da Ordenacao por HeapSort:
Tempo Execucao: 0.000034 seconds.
Comparacoes: 0
Movimentacoes: 0
Memoria (bytes): 40
[11, 14, 16, 31, 37, 37, 71, 73, 85, 86]

Informacoes da Ordenacao por QuickSort:
Tempo Execucao: 0.000014 seconds.
Comparacoes: 50
Movimentacoes: 9
Memoria (bytes): 40
[11, 14, 16, 31, 37, 37, 71, 73, 85, 86]

```

Figura 2: Questão 1.1 - Saída 2

1.2 Aplicando modificações para decrescer

Código

```

1 #ifndef ORDENACAO_PT2
2 #define ORDENACAO_PT2
3
4 int* copiaVetor(int* v, int n);
5 void imprimeVetor(int* v, int n);
6 void preencheAleatorio(int* v, int n,
7     int ini, int fim);
8 void troca(int* a, int* b);
9 void criaHeap(int* v, int pai, int fim);
10 void HeapSort(int* v, int n);
11 void Merge(int* v, int ini, int meio,
12     int fim);
13 void MergeSort(int* v, int ini, int
14     fim);
15
16 int particao(int* v, int ini, int fim);
17 void QuickSort(int* v, int ini, int
18     fim, int n);
19 int getComp();
20 int getMov();
21 void setComp(int valor);
22 void setMov(int valor);
23
24 #endif
25
26 //Medidas de Complexidade
27 int comp; //Num. de comparacoes
28 int mov; //Num. de movimentacoes
29
30 int getComp(){
31     return comp;
32 }
33
34 int getMov(){
35     return mov;
36 }
37
38 void setComp(int valor){
39     comp = valor;
40 }
41
42 void setMov(int valor){
43     mov = valor;
44 }
45
46 int* copiaVetor(int* v, int n){
47     int i;
48     int* v2;
49     v2 = (int*) malloc (n*sizeof(int));
50     for(i=0; i<n; i++) v2[i] = v[i];
51     return v2;
52 }
53
54 void imprimeVetor(int* v, int n){
55     int i, prim = 1;
56     printf("[");
57     for(i=0; i<n; i++)
58         if(prim){ printf("%d", v[i]);
59             prim = 0; }
60     else printf(", %d", v[i]);
61     printf("]\n");
62 }

```

codigos/questao12/questao12.h

```

42
43 void preencheAleatorio(int* v, int n,
    int ini, int fim){
44     int i;
45     for(i=0; i<n; i++){
46         v[i] = ini + rand() % (fim-ini
            + 1);
47         //v[i] = (n-i); //Para o pior
            caso
48     }
49 }
50
51 void troca(int* a, int *b){
52     int aux = *a;
53     *a = *b;
54     *b = aux;
55 }
56
57 void criaHeap(int *v, int pai, int fim){
58     int aux = v[pai];
59     int filho = 2*pai + 1;
60     while(filho <= fim){
61         if(filho < fim && v[filho] >
            v[filho+1])
62             filho++;
63         if(aux < v[filho]){
64             break;
65         } else {
66             v[pai] = v[filho];
67             pai = filho;
68             filho = 2*pai + 1;
69         }
70     }
71     v[pai] = aux;
72 }
73
74
75 void HeapSort(int *v, int n){
76     int i;
77     for(i=(n-1)/2; i>=0; i--){
78         criaHeap(v, i, n-1);
79     }
80     for(i=n-1; i>=1; i--){
81         troca(&v[0], &v[i]);
82         criaHeap(v, 0, i-1);
83     }
84 }
85 void Merge(int *v, int ini, int meio,
    int fim){
86     int tam = fim-ini+1;
87     //Vetor Auxiliar - A
88     int *A = (int*) malloc
        (tam*sizeof(int));
89     int i = ini, j = meio+1, k = 0;
90     while (i<=meio && j<=fim) {
91         if (v[i] > v[j]){ A[k] = v[i];
            i++; }
92         else { A[k] = v[j]; j++; }
93         k++;
94     }
95     while (i<=meio) { A[k] = v[i];
        i++; k++; }
96     while (j<=fim) { A[k] = v[j]; j++;
        k++; }
97     for(i = ini, k=0; i<=fim; i++, k++)
        v[i] = A[k];
98     free(A);
99 }
100
101 void MergeSort(int *v, int ini, int
    fim){
102     if(ini < fim ){
103         int meio = (ini + fim)/2;
104         MergeSort(v, ini, meio);
105         MergeSort(v, meio+1, fim);
106         Merge(v, ini, meio, fim);
107     }
108 }
109
110 int particao(int *v, int ini, int fim){
111     int i = ini, j = fim;
112     int pivo = v[(ini+fim)/2];
113     while (1) {
114         comp++;
115         while(v[i] > pivo){ i++;
            comp++; } //procura algum >=
            pivo do lado esquerdo
116
117         comp++;
118         while(v[j] < pivo){ j--;
            comp++;} //procura algum <=
            pivo do lado direito
119
120         if(i<j){
121             troca(&v[i], &v[j]);
122             //troca os elementos
            encontrados
123             mov++;
124             i++;
125             j--;
126         }else
            return j; //retorna o local
            onde foi feita a particao
127     }
128 }
129
130 void QuickSort(int *v, int ini, int
    fim, int n){
131     if(ini < fim ){
132         int q = particao(v, ini, fim);
133         //printf("Parts: (%d, %d) e
            (%d, %d): ", ini, q, q+1,
            fim);
134         //imprimeVetor(v, n);
135         QuickSort(v, ini, q, n);
136         QuickSort(v, q+1, fim, n);
137     }
138 }

```

codigos/questao12/questao12.c

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include "questao12.h"
5
6 int main(){
7
8     //Atribuicoes iniciais
9     srand(time(NULL));
10    setComp(0);
11    setMov(0);
12    clock_t t;
13
14    int *v1, *v2, *v3;
15    int n;
16
17    /* Recebe valores e preenche */
18    printf("Digite o tamanho do
19           vetor:\n");
20    scanf("%d", &n);
21    v1 = (int*) malloc (n*sizeof(int));
22    preencheAleatorio(v1, n, 1, 100);
23    imprimeVetor(v1, n);
24
25    v2 = copiaVetor(v1, n);
26    v3 = copiaVetor(v1, n);
27
28    /* MergeSort */
29    t = clock();
30    MergeSort(v1, 0, n-1);
31    t = clock() - t;
32    printf("\nInformacoes da Ordenacao
33           por MergeSort:\n");
34    printf("Tempo Execucao: %f
35           seconds.\n",
36           ((float)t)/CLOCKS_PER_SEC);
37    printf("Comparacoes: %d\n",
38           getComp());
39    printf("Movimentacoes: %d\n",
40           getMov());
41    printf("Memoria (bytes): %ld\n",
42           n*sizeof(int));
43
44    imprimeVetor(v1, n);
45
46    /* HeapSort */
47    setComp(0);
48    setMov(0);
49    t = clock();
50    HeapSort(v2, n);
51    t = clock() - t;
52    printf("\nInformacoes da Ordenacao
53           por HeapSort:\n");
54    printf("Tempo Execucao: %f
55           seconds.\n",
56           ((float)t)/CLOCKS_PER_SEC);
57    printf("Comparacoes: %d\n",
58           getComp());
59    printf("Movimentacoes: %d\n",
60           getMov());
61    printf("Memoria (bytes): %ld\n",
62           n*sizeof(int));
63
64    imprimeVetor(v2, n);
65
66    /* QuickSort */
67    setComp(0);
68    setMov(0);
69    t = clock();
70    QuickSort(v3, 0, n-1, n);
71    t = clock() - t;
72    printf("\nInformacoes da Ordenacao
73           por QuickSort:\n");
74    printf("Tempo Execucao: %f
75           seconds.\n",
76           ((float)t)/CLOCKS_PER_SEC);
77    printf("Comparacoes: %d\n",
78           getComp());
79    printf("Movimentacoes: %d\n",
80           getMov());
81    printf("Memoria (bytes): %ld\n",
82           n*sizeof(int));
83
84    imprimeVetor(v3, n);
85
86    free(v1);
87    free(v2);
88    free(v3);
89
90    return 0;
91 }

```

codigos/questao12/main.c

```

1 all: questao12.o
2     gcc questao12.o main.c -o main
3
4 questao12.o: questao12.h questao12.c
5     gcc -c questao12.c

```

```

6
7 clean:
8     rm -f questao12.o main

```

codigos/questao12/Makefile

```

Informacoes da Ordenacao por HeapSort:
Tempo Execucao: 0.000016 seconds.
Comparacoes: 0
Movimentacoes: 0
Memoria (bytes): 40
[98, 87, 84, 83, 73, 54, 25, 11, 10, 5]

Informacoes da Ordenacao por QuickSort:
Tempo Execucao: 0.000014 seconds.
Comparacoes: 60
Movimentacoes: 8
Memoria (bytes): 40
[98, 87, 84, 83, 73, 54, 25, 11, 10, 5]

```

Figura 3: Questão 2.1 - Saída 1

```

Digite o tamanho do vetor:
10
[98, 73, 54, 11, 10, 5, 83, 25, 87, 84]

Informacoes da Ordenacao por MergeSort:
Tempo Execucao: 0.000023 seconds.
Comparacoes: 0
Movimentacoes: 0
Memoria (bytes): 40
[98, 87, 84, 83, 73, 54, 25, 11, 10, 5]

```

Figura 4: Questão 2.1 - Saída 2

1.3 Utilizando entradas grandes

Código

```

1 #ifndef ORDENACAO_PT3
2 #define ORDENACAO_PT3
3
4 int* copiaVetor(int* v, int n);
5 void imprimeVetor(int* v, int n);
6 void preencheAleatorio(int* v, int n,
7     int ini, int fim);
8 void troca(int* a, int* b);
9 void criaHeap(int* v, int pai, int fim);
10 void HeapSort(int* v, int n);
11 void Merge(int* v, int ini, int meio,
12     int fim);
13 void MergeSort(int* v, int ini, int
14     fim);
15
16 int particao(int* v, int ini, int fim);
17 void QuickSort(int* v, int ini, int
18     fim, int n);
19 int getComp();
20 int getMov();
21 void setComp(int valor);
22 void setMov(int valor);
23 #endif
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41

```

codigos/questao13/questao13.h

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <time.h>
4 #include "questao13.h"
5
6 //Medidas de Complexidade
7 int comp; //Num. de comparacoes
8 int mov; //Num. de movimentacoes
9
10 int getComp(){
11     return comp;
12 }
13
14 int getMov(){
15     return mov;
16 }
17
18 void setComp(int valor){
19     comp = valor;
20 }
21
22 void setMov(int valor){
23     mov = valor;
24 }
25
26 int* copiaVetor(int* v, int n){
27     int i;
28     int* v2;
29     v2 = (int*) malloc (n*sizeof(int));
30     for(i=0; i<n; i++) v2[i] = v[i];
31     return v2;
32 }
33
34 void imprimeVetor(int* v, int n){
35     int i, prim = 1;
36     printf("[");
37     for(i=0; i<n; i++)
38         if(prim){ printf("%d", v[i]);
39             prim = 0; }
40     else printf(", %d", v[i]);
41     printf("]\n");

```

```

42
43 void preencheAleatorio(int* v, int n,
    int ini, int fim){
44     int i;
45     for(i=0; i<n; i++){
46         v[i] = ini + rand() % (fim-ini
            + 1);
47         //v[i] = (n-i); //Para o pior
            caso
48     }
49 }
50
51 void troca(int* a, int *b){
52     int aux = *a;
53     *a = *b;
54     *b = aux;
55 }
56
57 void criaHeap(int *v, int pai, int fim){
58     int aux = v[pai];
59     int filho = 2*pai + 1;
60     while(filho <= fim){
61         if(filho < fim)
62             if(v[filho] < v[filho+1])
63                 filho++;
64         if(aux < v[filho]){
65             v[pai] = v[filho];
66             pai = filho;
67             filho = 2*pai + 1;
68         }else filho = fim + 1;
69     }
70     v[pai] = aux;
71 }
72
73 void HeapSort(int *v, int n){
74     int i;
75     for(i=(n-1)/2; i>=0; i--)
76         criaHeap(v, i, n-1);
77     for(i=n-1; i>=1; i--){
78         troca(&v[0], &v[i]);
79         criaHeap(v, 0, i-1);
80     }
81 }
82
83 void Merge(int *v, int ini, int meio,
    int fim){
84     int tam = fim-ini+1;
85     //Vetor Auxiliar - A
86     int *A = (int*) malloc
        (tam*sizeof(int));
87     int i = ini, j = meio+1, k = 0;
88     while (i<=meio && j<=fim) {
89         if (v[i] < v[j]){ A[k] = v[i];
            i++; }
90         else { A[k] = v[j]; j++; }
91         k++;
92     }
93     while (i<=meio) { A[k] = v[i];
        i++; k++; }
94     while (j<=fim) { A[k] = v[j]; j++;
        k++; }
95     for(i = ini, k=0; i<=fim; i++, k++)
        v[i] = A[k];
96     free(A);
97 }
98
99 void MergeSort(int *v, int ini, int
    fim){
100     if(ini < fim ){
101         int meio = (ini + fim)/2;
102         MergeSort(v, ini, meio);
103         MergeSort(v, meio+1, fim);
104         Merge(v, ini, meio, fim);
105     }
106 }
107
108 int particao(int *v, int ini, int fim){
109     int i = ini, j = fim;
110     int pivo = v[(ini+fim)/2];
111     while (1) {
112         comp++;
113         while(v[i] < pivo){ i++;
            comp++; } //procura algum >=
            pivo do lado esquerdo
114
115         comp++;
116         while(v[j] > pivo){ j--;
            comp++;} //procura algum <=
            pivo do lado direito
117
118         if(i<j){
119             troca(&v[i], &v[j]);
            //troca os elementos
            encontrados
120
121             mov++;
122             i++;
123             j--;
124         }else
            return j; //retorna o local
            onde foi feita a particao
125     }
126 }
127
128 void QuickSort(int *v, int ini, int
    fim, int n){
129     if(ini < fim ){
130         int q = particao(v, ini, fim);
131         //printf("Parts: (%d, %d) e
            (%d, %d): ", ini, q, q+1,
            fim);
132         //imprimeVetor(v, n);
133         QuickSort(v, ini, q, n);
134         QuickSort(v, q+1, fim, n);
135     }
136 }

```

codigos/questao13/questao13.c


```

1  #include <stdio.h>
2  #include <stdlib.h>
3  #include <time.h>
4  #include "questao13.h"
5
6  int main(int argc, char *argv[]){
7
8      //Atribuicoes iniciais
9      setComp(0);
10     setMov(0);
11     clock_t t;
12
13     int n;
14
15     /* Leitura de arquivo e
16        verificacoes */
17     if (argc != 2) {
18         fprintf(stderr, "Uso: %s
19             <nome_do_arquivo>\n",
20                 argv[0]);
21         return 1;
22     }
23
24     FILE *arquivo = fopen(argv[1], "r");
25
26     if (arquivo == NULL) {
27         fprintf(stderr, "Erro ao abrir
28             o arquivo %s.\n", argv[1]);
29         return 1;
30     }
31
32     /* Pega o tamanho do arquivo e cria
33        um vetor com esse tamanho */
34     fscanf(arquivo, "%d", &n);
35
36     int *v1 = (int*)malloc(n *
37         sizeof(int));
38     int *v2 = (int*)malloc(n *
39         sizeof(int));
40     int *v3 = (int*)malloc(n *
41         sizeof(int));
42
43     /* Confere se o vetor nao esta
44        vazio */
45     if (v1 == NULL) {
46         fprintf(stderr, "Erro ao alocar
47             memoria para o vetor.\n");
48         fclose(arquivo);
49         return 1;
50     }
51
52     /* Copia os dados do arquivo para
53        esse vetor */
54     for (int i = 0; i < n; i++) {
55         fscanf(arquivo, "%d", &v1[i]);
56     }
57
58     v2 = copiaVetor(v1, n);
59     v3 = copiaVetor(v1, n);
60
61     /* MergeSort */
62     t = clock();
63     MergeSort(v1, 0, n-1);
64     t = clock() - t;
65     printf("\nInformacoes da Ordenacao
66         por MergeSort:\n");
67     printf("Tempo Execucao: %f
68         seconds.\n",
69         ((float)t)/CLOCKS_PER_SEC);
70     printf("Comparacoes: %d\n",
71         getComp());
72     printf("Movimentacoes: %d\n",
73         getMov());
74     printf("Memoria (bytes): %ld\n",
75         n*sizeof(int));
76
77     /* HeapSort */
78     setComp(0);
79     setMov(0);
80     t = clock();
81     HeapSort(v2, n);
82     t = clock() - t;
83     printf("\nInformacoes da Ordenacao
84         por HeapSort:\n");
85     printf("Tempo Execucao: %f
86         seconds.\n",
87         ((float)t)/CLOCKS_PER_SEC);
88     printf("Comparacoes: %d\n",
89         getComp());
90     printf("Movimentacoes: %d\n",
91         getMov());
92     printf("Memoria (bytes): %ld\n",
93         n*sizeof(int));
94
95     /* QuickSort */
96     setComp(0);
97     setMov(0);
98     t = clock();
99     QuickSort(v3, 0, n-1, n);
100    t = clock() - t;
101    printf("\nInformacoes da Ordenacao
102        por QuickSort:\n");
103    printf("Tempo Execucao: %f
104        seconds.\n",
105        ((float)t)/CLOCKS_PER_SEC);
106    printf("Comparacoes: %d\n",
107        getComp());
108    printf("Movimentacoes: %d\n",
109        getMov());
110    printf("Memoria (bytes): %ld\n",
111        n*sizeof(int));
112
113    free(v1);
114    free(v2);
115    free(v3);
116
117    return 0;
118 }

```

codigos/questao13/main.c

```

1 all: questao13.o
2   gcc questao13.o main.c -o main
3
4 questao13.o: questao13.h questao13.c
5   gcc -c questao13.c

```

```

6
7 clean:
8   rm -f questao13.o main

```

codigos/questao13/Makefile

Saída:

Devido à expressiva quantidade de arquivos de entrada e considerando o tempo significativo que seria demandado para processar um milhão de elementos, optou-se por utilizar exclusivamente os conjuntos que continham o segundo maior número de elementos, totalizando cem mil números.

```

Informacoes da Ordenacao por MergeSort:
Tempo Execucao: 0.025097 seconds.
Comparacoes: 0
Movimentacoes: 0
Memoria (bytes): 400000

Informacoes da Ordenacao por HeapSort:
Tempo Execucao: 0.024440 seconds.
Comparacoes: 0
Movimentacoes: 0
Memoria (bytes): 400000

Informacoes da Ordenacao por QuickSort:
Tempo Execucao: 0.007771 seconds.
Comparacoes: 1768928
Movimentacoes: 50000
Memoria (bytes): 400000

```

Figura 5: Questão 2.1 - Ordem decrescente

```

Informacoes da Ordenacao por MergeSort:
Tempo Execucao: 0.041002 seconds.
Comparacoes: 0
Movimentacoes: 0
Memoria (bytes): 400000

Informacoes da Ordenacao por HeapSort:
Tempo Execucao: 0.037094 seconds.
Comparacoes: 0
Movimentacoes: 0
Memoria (bytes): 400000

Informacoes da Ordenacao por QuickSort:
Tempo Execucao: 0.027562 seconds.
Comparacoes: 2524473
Movimentacoes: 422822
Memoria (bytes): 400000

```

Figura 6: Questão 2.1 - Ordem misturada

```

Informacoes da Ordenacao por MergeSort:
Tempo Execucao: 0.022847 seconds.
Comparacoes: 0
Movimentacoes: 0
Memoria (bytes): 400000

Informacoes da Ordenacao por HeapSort:
Tempo Execucao: 0.024006 seconds.
Comparacoes: 0
Movimentacoes: 0
Memoria (bytes): 400000

Informacoes da Ordenacao por QuickSort:
Tempo Execucao: 0.007720 seconds.
Comparacoes: 1768927
Movimentacoes: 0
Memoria (bytes): 400000

```

Figura 7: Questão 3.1 - Ordem crescente

```

Informacoes da Ordenacao por MergeSort:
Tempo Execucao: 0.025112 seconds.
Comparacoes: 0
Movimentacoes: 0
Memoria (bytes): 400000

Informacoes da Ordenacao por HeapSort:
Tempo Execucao: 0.023525 seconds.
Comparacoes: 0
Movimentacoes: 0
Memoria (bytes): 400000

Informacoes da Ordenacao por QuickSort:
Tempo Execucao: 0.007370 seconds.
Comparacoes: 1768927
Movimentacoes: 26
Memoria (bytes): 400000

```

Figura 8: Questão 3.1 - Quase em ordem crescente