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 Reimplentação

Código

```h
#ifndef ORDENACAO_PT1
#define ORDENACAO_PT1

int* copiaVetor(int* v, int n);
void imprimeVetor(int* v, int n);
void preencheAleatorio(int* v, int n, int ini, int fim);
void troca(int* a, int *b);
void criaHeap(int *v, int pai, int fim);
void HeapSort(int *v, int n);
void Merge(int *v, int ini, int meio, int fim);
void MergeSort(int *v, int ini, int fim);
int particao(int *v, int ini, int fim);
void QuickSort(int *v, int ini, int fim, int n);
int getComp();
int getMov();
void setComp(int valor);
void setMov(int valor);

#endif
```

codigos/questao11/questao11.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "questao11.h"

//Medidas de Complexidade
int comp; //Num. de comparacoes
int mov; //Num. de movimentacoes

int getComp(){
    return comp;
}

int getMov(){
    return mov;
}

void setComp(int valor){
    comp = valor;
}

void setMov(int valor){
    mov = valor;
}

int* copiaVetor(int* v, int n){
    int i;
    int *v2;
    v2 = (int*) malloc (n*sizeof(int));
    for(i=0; i<n; i++) v2[i] = v[i];
    return v2;
}

void imprimeVetor(int* v, int n){
    int i, prim = 1;
    printf("[");
    for(i=0; i<n; i++)
        if(prim){ printf("%d", v[i]);
            prim = 0; }
        else printf(", %d", v[i]);
```

```
40      printf("]\n");
41  }
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             mov++;
121             i++;
122             j--;
123         }else
124             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/questao11/questao11.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "questao11.h"

int main(){

    //Atribuicoes iniciais
    srand(time(NULL));
    setComp(0);
    setMov(0);
    clock_t t;

    int *v1, *v2, *v3;
    int n;

    /* Recebe valores e preenche */
    printf("Digite o tamanho do vetor:\n");
    scanf("%d", &n);
    v1 = (int*) malloc (n*sizeof(int));

    preencheAleatorio(v1, n, 1, 100);
    imprimeVetor(v1, n);

    v2 = copiaVetor(v1, n);
    v3 = copiaVetor(v1, n);

    /* MergeSort */
    t = clock();
    MergeSort(v1, 0, n-1);
    t = clock() - t;
    printf("\nInformacoes da Ordenacao por MergeSort:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", getComp());
    printf("Movimentacoes: %d\n", getMov());
    printf("Memoria (bytes): %ld\n", n*sizeof(int));

    imprimeVetor(v1, n);

    /* HeapSort */
    setComp(0);
    setMov(0);
    t = clock();
    HeapSort(v2, n);
    t = clock() - t;
    printf("\nInformacoes da Ordenacao por HeapSort:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", getComp());
    printf("Movimentacoes: %d\n", getMov());
    printf("Memoria (bytes): %ld\n", n*sizeof(int));

    imprimeVetor(v2, n);

    /* QuickSort */
    setComp(0);
    setMov(0);
    t = clock();
    QuickSort(v3, 0, n-1, n);
    t = clock() - t;
    printf("\nInformacoes da Ordenacao por QuickSort:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", getComp());
    printf("Movimentacoes: %d\n", getMov());
    printf("Memoria (bytes): %ld\n", n*sizeof(int));

    imprimeVetor(v3, n);

    free(v1);
    free(v2);
    free(v3);

    return 0;
}
```

codigos/questao11/main.c

```
all: questao11.o
  gcc questao11.o main.c -o main

questao11.o: questao11.h questao11.c
  gcc -c questao11.c

clean:
  rm -f questao11.o main
```

codigos/questao11/Makefile

Saída

```
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

```c
#ifndef ORDENACAO_PT2
#define ORDENACAO_PT2

int* copiaVetor(int* v, int n);
void imprimeVetor(int* v, int n);
void preencheAleatorio(int* v, int n, int ini, int fim);
void troca(int* a, int *b);
void criaHeap(int *v, int pai, int fim);
void HeapSort(int *v, int n);
void Merge(int *v, int ini, int meio, int fim);
void MergeSort(int *v, int ini, int fim);
int particao(int *v, int ini, int fim);
void QuickSort(int *v, int ini, int fim, int n);
int getComp();
int getMov();
void setComp(int valor);
void setMov(int valor);

#endif
```

codigos/questao12/questao12.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "questao12.h"

//Medidas de Complexidade
int comp; //Num. de comparacoes
int mov; //Num. de movimentacoes

int getComp(){
    return comp;
}

int getMov(){
    return mov;
}

void setComp(int valor){
    comp = valor;
}

void setMov(int valor){
    mov = valor;
}

int* copiaVetor(int* v, int n){
    int i;
    int *v2;
    v2 = (int*) malloc (n*sizeof(int));
    for(i=0; i<n; i++) v2[i] = v[i];
    return v2;
}

void imprimeVetor(int* v, int n){
    int i, prim = 1;
    printf("[");
    for(i=0; i<n; i++)
        if(prim){ printf("%d", v[i]); prim = 0; }
        else printf(", %d", v[i]);
    printf("]\n");
}
```

```c

void preencheAleatorio(int* v, int n, int ini, int fim){
    int i;
    for(i=0; i<n; i++){
        v[i] = ini + rand() % (fim-ini + 1);
        //v[i] = (n-i); //Para o pior caso
    }
}

void troca(int* a, int *b){
  int aux = *a;
  *a = *b;
  *b = aux;
}

void criaHeap(int *v, int pai, int fim){
    int aux = v[pai];
    int filho = 2*pai + 1;
    while(filho <= fim){
        if(filho < fim && v[filho] > v[filho+1])
            filho++;
        if(aux < v[filho]){
            break;
        } else {
            v[pai] = v[filho];
            pai = filho;
            filho = 2*pai + 1;
        }
    }
    v[pai] = aux;
}


void HeapSort(int *v, int n){
    int i;
    for(i=(n-1)/2; i>=0; i--)
        criaHeap(v, i, n-1);
    for(i=n-1; i>=1; i--){
        troca(&v[0], &v[i]);
        criaHeap(v, 0, i-1);
    }
}

void Merge(int *v, int ini, int meio, int fim){
    int tam = fim-ini+1;
    //Vetor Auxiliar - A
    int *A = (int*) malloc (tam*sizeof(int));
    int i = ini, j = meio+1, k = 0;
    while (i<=meio && j<=fim) {
        if (v[i] > v[j]){ A[k] = v[i]; i++; }
        else { A[k] = v[j]; j++; }
        k++;
    }
    while (i<=meio) { A[k] = v[i]; i++;  k++; }
    while (j<=fim) { A[k] = v[j];  j++; k++; }
    for(i = ini, k=0; i<=fim; i++, k++)
        v[i] = A[k];
    free(A);
}

void MergeSort(int *v, int ini, int fim){
    if(ini < fim ){
        int meio = (ini + fim)/2;
        MergeSort(v, ini, meio);
        MergeSort(v, meio+1, fim);
        Merge(v, ini, meio, fim);
    }
}

int particao(int *v, int ini, int fim){
    int i = ini, j = fim;
    int pivo = v[(ini+fim)/2];
    while (1) {
        comp++;
        while(v[i] > pivo){ i++; comp++; } //procura algum >= pivo do lado esquerdo

        comp++;
        while(v[j] < pivo){ j--; comp++;} //procura algum <= pivo do lado direito

        if(i<j){
            troca(&v[i], &v[j]); //troca os elementos encontrados
            mov++;
            i++;
            j--;
        }else
            return j; //retorna o local onde foi feita a particao
    }
}

void QuickSort(int *v, int ini, int fim, int n){
    if(ini < fim ){
        int q = particao(v, ini, fim);
        //printf("Parts: (%d, %d) e (%d, %d): ", ini, q, q+1, fim);
        //imprimeVetor(v, n);
        QuickSort(v, ini, q, n);
        QuickSort(v, q+1, fim, n);
    }
}
```

codigos/questao12/questao12.c

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "questao12.h"

int main(){

    //Atribuicoes iniciais
    srand(time(NULL));
    setComp(0);
    setMov(0);
    clock_t t;

    int *v1, *v2, *v3;
    int n;

    /* Recebe valores e preenche */
    printf("Digite o tamanho do vetor:\n");
    scanf("%d", &n);
    v1 = (int*) malloc (n*sizeof(int));

    preencheAleatorio(v1, n, 1, 100);
    imprimeVetor(v1, n);

    v2 = copiaVetor(v1, n);
    v3 = copiaVetor(v1, n);

    /* MergeSort */
    t = clock();
    MergeSort(v1, 0, n-1);
    t = clock() - t;
    printf("\nInformacoes da Ordenacao por MergeSort:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", getComp());
    printf("Movimentacoes: %d\n", getMov());
    printf("Memoria (bytes): %ld\n", n*sizeof(int));

    imprimeVetor(v1, n);

    /* HeapSort */
    setComp(0);
    setMov(0);
    t = clock();
    HeapSort(v2, n);
    t = clock() - t;
    printf("\nInformacoes da Ordenacao por HeapSort:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", getComp());
    printf("Movimentacoes: %d\n", getMov());
    printf("Memoria (bytes): %ld\n", n*sizeof(int));

    imprimeVetor(v2, n);

    /* QuickSort */
    setComp(0);
    setMov(0);
    t = clock();
    QuickSort(v3, 0, n-1, n);
    t = clock() - t;
    printf("\nInformacoes da Ordenacao por QuickSort:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", getComp());
    printf("Movimentacoes: %d\n", getMov());
    printf("Memoria (bytes): %ld\n", n*sizeof(int));

    imprimeVetor(v3, n);

    free(v1);
    free(v2);
    free(v3);

    return 0;
}
```

codigos/questao12/main.c

```
all: questao12.o
  gcc questao12.o main.c -o main

questao12.o: questao12.h questao12.c
  gcc -c questao12.c

clean:
  rm -f questao12.o main
```

codigos/questao12/Makefile

Saída

```
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

```c
#ifndef ORDENACAO_PT3
#define ORDENACAO_PT3

int* copiaVetor(int* v, int n);
void imprimeVetor(int* v, int n);
void preencheAleatorio(int* v, int n, int ini, int fim);
void troca(int* a, int *b);
void criaHeap(int *v, int pai, int fim);
void HeapSort(int *v, int n);
void Merge(int *v, int ini, int meio, int fim);
void MergeSort(int *v, int ini, int fim);
int particao(int *v, int ini, int fim);
void QuickSort(int *v, int ini, int fim, int n);
int getComp();
int getMov();
void setComp(int valor);
void setMov(int valor);

#endif
```

codigos/questao13/questao13.h

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "questao13.h"

//Medidas de Complexidade
int comp; //Num. de comparacoes
int mov; //Num. de movimentacoes

int getComp(){
    return comp;
}

int getMov(){
    return mov;
}

void setComp(int valor){
    comp = valor;
}

void setMov(int valor){
    mov = valor;
}

int* copiaVetor(int* v, int n){
    int i;
    int *v2;
    v2 = (int*) malloc (n*sizeof(int));
    for(i=0; i<n; i++) v2[i] = v[i];
    return v2;
}

void imprimeVetor(int* v, int n){
    int i, prim = 1;
    printf("[");
    for(i=0; i<n; i++)
        if(prim){ printf("%d", v[i]); prim = 0; }
        else printf(", %d", v[i]);
    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            mov++;
121            i++;
122            j--;
123        }else
124            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

```c
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "questao13.h"

int main(int argc, char *argv[]){

    //Atribuicoes iniciais
    setComp(0);
    setMov(0);
    clock_t t;

    int n;

    /* Leitura de arquivo e verificacoes */
    if (argc != 2) {
        fprintf(stderr, "Uso: %s <nome_do_arquivo>\n", argv[0]);
        return 1;
    }

    FILE *arquivo = fopen(argv[1], "r");

    if (arquivo == NULL) {
        fprintf(stderr, "Erro ao abrir o arquivo %s.\n", argv[1]);
        return 1;
    }

    /* Pega o tamanho do arquivo e cria um vetor com esse tamanho */
    fscanf(arquivo, "%d", &n);

    int *v1 = (int*)malloc(n * sizeof(int));
    int *v2 = (int*)malloc(n * sizeof(int));
    int *v3 = (int*)malloc(n * sizeof(int));

    /* Confere  se o vetor nao esta vazio */
    if (v1 == NULL) {
        fprintf(stderr, "Erro ao alocar memoria para o vetor.\n");
        fclose(arquivo);
        return 1;
    }

    /* Copia os dados do arquivo para esse vetor */
    for (int i = 0; i < n; i++) {
        fscanf(arquivo, "%d", &v1[i]);
    }

    v2 = copiaVetor(v1, n);
    v3 = copiaVetor(v1, n);

    /* MergeSort */
    t = clock();
    MergeSort(v1, 0, n-1);
    t = clock() - t;
    printf("\nInformacoes da Ordenacao por MergeSort:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", getComp());
    printf("Movimentacoes: %d\n", getMov());
    printf("Memoria (bytes): %ld\n", n*sizeof(int));

    /* HeapSort */
    setComp(0);
    setMov(0);
    t = clock();
    HeapSort(v2, n);
    t = clock() - t;
    printf("\nInformacoes da Ordenacao por HeapSort:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", getComp());
    printf("Movimentacoes: %d\n", getMov());
    printf("Memoria (bytes): %ld\n", n*sizeof(int));

    /* QuickSort */
    setComp(0);
    setMov(0);
    t = clock();
    QuickSort(v3, 0, n-1, n);
    t = clock() - t;
    printf("\nInformacoes da Ordenacao por QuickSort:\n");
    printf("Tempo Execucao:  %f seconds.\n", ((float)t)/CLOCKS_PER_SEC);
    printf("Comparacoes: %d\n", getComp());
    printf("Movimentacoes: %d\n", getMov());
    printf("Memoria (bytes): %ld\n", n*sizeof(int));

    free(v1);
    free(v2);
    free(v3);

    return 0;
}
```

codigos/questao13/main.c

```
all: questao13.o
  gcc questao13.o main.c -o main

questao13.o: questao13.h questao13.c
  gcc -c questao13.c

clean:
  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