**1.1/Heap.h**

```c

#ifndef HEAP_H
#define HEAP_H

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

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

int* copiaVetorHeap(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 imprimeVetorHeap(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");
}

void preencheAleatorioHeap(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 trocaHeap(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)
            if(v[filho] < v[filho+1])
                filho++;
        if(aux < v[filho]){
            v[pai] = v[filho];
            pai = filho;
            filho = 2*pai + 1;
        }else filho = fim + 1;
    }
    v[pai] = aux;
}
```

```
58
59 void HeapSort(int *v, int n){
60     int i;
61     for(i=(n-1)/2; i>=0; i--)
62         criaHeap(v, i, n-1);
63     for(i=n-1; i>=1; i--){
64         trocaHeap(&v[0], &v[i]);
65         criaHeap(v, 0, i-1);
66     }
67 }
68
69
70 #endif
```

**1.1/main.c**

```c
#include <stdio.h>
#include "Heap.h"
#include "Merge.h"
#include "Quick.h"
#include "Shell.h"


int main(){

   int n;
   printf("Quantidade de elementos a serem ordenados: ");
   scanf("%d", &n);


   int vetor[n];
   printf("Digite os %d inteiros a serem ordenados:\n", n);
   for (int i = 0; i < n; i++) {
       scanf("%d", &vetor[i]);
   }

   int heap[n], merge[n], quick[n], shell[n];
   for (int i = 0; i < n; i++) {
       heap[i] = vetor[i];
       merge[i] = vetor[i];
       quick[i] = vetor[i];
       shell[i] = vetor[i];
   }

   HeapSort(heap, n);
   MergeSort(merge, 0, n-1);
   QuickSort(quick, 0, n-1, n);
   ShellSort(shell, n);

   printf("\nValores ordenados pelo Heap Sort: ");
   for (int i = 0; i < n; i++) {
       printf("%d ", heap[i]);
   }

   printf("\nValores ordenados pelo Merge Sort: ");
   for (int i = 0; i < n; i++) {
       printf("%d ", merge[i]);
   }

   printf("\nValores ordenados pelo Quick Sort: ");
   for (int i = 0; i < n; i++) {
       printf("%d ", quick[i]);
   }

   printf("\nValores ordenados pelo Shell Sort: ");
   for (int i = 0; i < n; i++) {
       printf("%d ", shell[i]);
   }

   printf("\n");


   return 0;
```

```
58 }
59
```

**1.1/Merge.h**

```c
/*---------------- File: Merge.c  ----------------------+
|Merge Sort                                             |
|                                                       |
|                                                       |
| Implementado por Guilherme C. Pena em 08/11/2023      |
+-------------------------------------------------------+ */
#ifndef MERGE_H
#define MERGE_H


#include <stdio.h>
#include <stdlib.h>
#include <time.h>

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

int* copiaVetorMerge(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 imprimeVetorMerge(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");
}

void preencheAleatorioMerge(int* v, int n, int ini, int fim){
    int i;
    for(i=0; i<n; i++)
        v[i] = ini + rand() % (fim-ini + 1);
}

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

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++; }
```

```
58      while (j<=fim) { A[k] = v[j];  j++;  k++; }
59      for(i = ini, k=0; i<=fim; i++, k++) v[i] = A[k];
60      free(A);
61  }
62
63  void MergeSort(int *v, int ini, int fim){
64      if(ini < fim ){
65          int meio = (ini + fim)/2;
66          MergeSort(v, ini, meio);
67          MergeSort(v, meio+1, fim);
68          Merge(v, ini, meio, fim);
69      }
70  }
71
72
73  #endif
```

## 1.1/Quick.h

```c
/*---------------- File: Quick.c  ----------------------+
|Quick Sort                                             |
|                                                       |
|                                                       |
| Implementado por Guilherme C. Pena em 14/11/2023      |
+-------------------------------------------------------+ */
#ifndef QUICK_H
#define QUICK_H


#include <stdio.h>
#include <stdlib.h>
#include <time.h>

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

int* copiaVetorQuick(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 imprimeVetorQuick(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");
}

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

//Versao do livro
int particiona(int *v, int ini, int fim){
    int esq, dir, pivo, aux;
    esq = ini; dir = fim;
    pivo = v[ini];
    while(esq < dir){
        while(esq <= fim && v[esq] <= pivo) esq++;
        while(dir >= 0 && v[dir] > pivo) dir--;
        if(esq < dir) trocaQuick(&v[esq], &v[dir]);
```

```
58      }
59      v[ini] = v[dir];
60      v[dir] = pivo;
61      return dir;
62  }
63
64  int particao(int *v, int ini, int fim){
65      int i = ini, j = fim;
66      int pivo = v[(ini+fim)/2];
67      while (1) {
68          comp++;
69          while(v[i] < pivo){ i++; comp++; } //procura algum >= pivo do lado esquerdo
70
71          comp++;
72          while(v[j] > pivo){ j--; comp++;} //procura algum <= pivo do lado direito
73
74          if(i<j){
75              trocaQuick(&v[i], &v[j]); //troca os elementos encontrados
76              mov++;
77              i++;
78              j--;
79          }else
80              return j; //retorna o local onde foi feita a particao
81      }
82  }
83
84  void QuickSort(int *v, int ini, int fim, int n){
85      if(ini < fim ){
86          int q = particao(v, ini, fim);
87          //printf("Parts: (%d, %d) e (%d, %d): ", ini, q, q+1, fim);
88          //imprimeVetor(v, n);
89          QuickSort(v, ini, q, n);
90          QuickSort(v, q+1, fim, n);
91      }
92  }
93
94  #endif
```

**1.1/Shell.h**

```c
/*---------------- File: Shell.c  ------------------+
|Shell Sort                                         |
|                                                                  |
|                                                                  |
| Implementado por Guilherme C. Pena em 20/11/2023      |
+---------------------------------------------------+ */

#ifndef SHELL_H
#define SHELL_H

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

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

int* copiaVetorShell(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 imprimeVetorShell(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");
}

void preencheAleatorioShell(int* v, int n, int ini, int fim){
    int i;
    for(i=0; i<n; i++)
        v[i] = ini + rand() % (fim-ini + 1);
}

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

void ShellSort(int *v, int n) {
  int i, j, atual;
  int h = 1;
  while(h < n) h = 3*h+1;
  while (h > 0) {
    for(i = h; i < n; i++) {
      atual = v[i];
      j = i;
      while (j > h-1 && atual <= v[j - h]) {
        v[j] = v[j - h];
        j = j - h;
```

```
      }
      v[j] = atual;
    }
    h = h/3;
  }
}

#endif
```

```
[lucascosta@fedora 1.1]$ gcc main.c -o main
[lucascosta@fedora 1.1]$ ./main
Quantidade de elementos a serem ordenados: 5
Digite os 5 inteiros a serem ordenados:
75
4
16
2
37

Valores ordenados pelo Heap Sort: 2 4 16 37 75
Valores ordenados pelo Merge Sort: 2 4 16 37 75
Valores ordenados pelo Quick Sort: 2 4 16 37 75
Valores ordenados pelo Shell Sort: 2 4 16 37 75
[lucascosta@fedora 1.1]$
```

**1.2/Heap.h**

```

#ifndef HEAP_H
#define HEAP_H

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

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

int* copiaVetorHeap(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 imprimeVetorHeap(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");
}

void preencheAleatorioHeap(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 trocaHeap(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]){
            v[pai] = v[filho];
            pai = filho;
            filho = 2*pai + 1;
        }else {
            filho = fim + 1;
```

```
58          }
59      }
60      v[pai] = aux;
61  }
62
63
64  void HeapSort(int *v, int n){
65      int i;
66      for(i=(n-1)/2; i>=0; i--)
67          criaHeap(v, i, n-1);
68      for(i=n-1; i>=1; i--){
69          trocaHeap(&v[0], &v[i]);
70          criaHeap(v, 0, i-1);
71      }
72  }
73
74
75  #endif
```

**1.2/main.c**

```c
#include <stdio.h>
#include "Heap.h"
#include "Merge.h"
#include "Quick.h"
#include "Shell.h"


int main(){

   int n;
   printf("Quantidade de elementos a serem ordenados: ");
   scanf("%d", &n);


   int vetor[n];
   printf("Digite os %d inteiros a serem ordenados:\n", n);
   for (int i = 0; i < n; i++) {
       scanf("%d", &vetor[i]);
   }

   int heap[n], merge[n], quick[n], shell[n];
   for (int i = 0; i < n; i++) {
       heap[i] = vetor[i];
       merge[i] = vetor[i];
       quick[i] = vetor[i];
       shell[i] = vetor[i];
   }

   HeapSort(heap, n);
   MergeSort(merge, 0, n-1);
   QuickSort(quick, 0, n-1, n);
   ShellSort(shell, n);

   printf("\nValores ordenados pelo Heap Sort: ");
   for (int i = 0; i < n; i++) {
       printf("%d ", heap[i]);
   }

   printf("\nValores ordenados pelo Merge Sort: ");
   for (int i = 0; i < n; i++) {
       printf("%d ", merge[i]);
   }

   printf("\nValores ordenados pelo Quick Sort: ");
   for (int i = 0; i < n; i++) {
       printf("%d ", quick[i]);
   }

   printf("\nValores ordenados pelo Shell Sort: ");
   for (int i = 0; i < n; i++) {
       printf("%d ", shell[i]);
   }

   printf("\n");


   return 0;
```

```
58 }
59
```

**1.2/Merge.h**

```

#ifndef MERGE_H
#define MERGE_H


#include <stdio.h>
#include <stdlib.h>
#include <time.h>

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

int* copiaVetorMerge(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 imprimeVetorMerge(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");
}

void preencheAleatorioMerge(int* v, int n, int ini, int fim){
    int i;
    for(i=0; i<n; i++)
        v[i] = ini + rand() % (fim-ini + 1);
}

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

void Merge(int *v, int ini, int meio, int fim){
    int tam = fim - ini + 1;
    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++;
    }

```

```
58      while (i <= meio) {
59          A[k] = v[i];
60          i++;
61          k++;
62      }
63
64      while (j <= fim) {
65          A[k] = v[j];
66          j++;
67          k++;
68      }
69
70      for (i = ini, k = 0; i <= fim; i++, k++) {
71          v[i] = A[k];
72      }
73
74      free(A);
75  }
76
77
78  void MergeSort(int *v, int ini, int fim){
79      if(ini < fim ){
80          int meio = (ini + fim)/2;
81          MergeSort(v, ini, meio);
82          MergeSort(v, meio+1, fim);
83          Merge(v, ini, meio, fim);
84      }
85  }
86
87
88  #endif
```

**1.2/Quick.h**

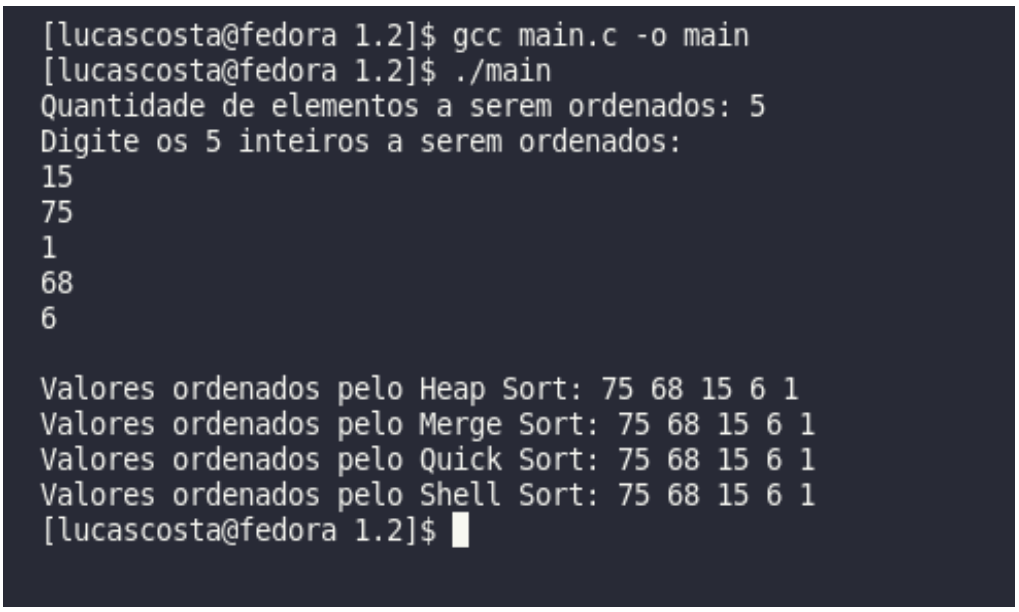
```c

#ifndef QUICK_H
#define QUICK_H


#include <stdio.h>
#include <stdlib.h>
#include <time.h>

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

int* copiaVetorQuick(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 imprimeVetorQuick(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");
}

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

//Versao do livro
int particiona(int *v, int ini, int fim){
    int esq, dir, pivo, aux;
    esq = ini; dir = fim;
    pivo = v[ini];
    while(esq < dir){
        while(esq <= fim && v[esq] <= pivo) esq++;
        while(dir >= 0 && v[dir] > pivo) dir--;
        if(esq < dir) trocaQuick(&v[esq], &v[dir]);
    }
    v[ini] = v[dir];
    v[dir] = pivo;
    return dir;
}
```

```
58
59  int particao(int *v, int ini, int fim){
60      int i = ini, j = fim;
61      int pivo = v[(ini+fim)/2];
62      while (1) {
63          comp++;
64          while(v[i] > pivo){ i++; comp++; } // Modificado para ordenar em ordem
    decrescente
65
66          comp++;
67          while(v[j] < pivo){ j--; comp++;} // Modificado para ordenar em ordem
    decrescente
68
69          if(i<j){
70              trocaQuick(&v[i], &v[j]); // Troca os elementos encontrados
71              mov++;
72              i++;
73              j--;
74          }else
75              return j; // Retorna o local onde foi feita a partição
76      }
77  }
78
79  void QuickSort(int *v, int ini, int fim, int n){
80      if(ini < fim ){
81          int q = particao(v, ini, fim);
82          //printf("Parts: (%d, %d) e (%d, %d): ", ini, q, q+1, fim);
83          //imprimeVetor(v, n);
84          QuickSort(v, ini, q, n);
85          QuickSort(v, q+1, fim, n);
86      }
87  }
88
89  #endif
```

**1.2/Shell.h**

```c


#ifndef SHELL_H
#define SHELL_H

#include <stdio.h>
#include <stdlib.h>
#include <time.h>

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

int* copiaVetorShell(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 imprimeVetorShell(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");
}

void preencheAleatorioShell(int* v, int n, int ini, int fim){
    int i;
    for(i=0; i<n; i++)
        v[i] = ini + rand() % (fim-ini + 1);
}

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


void ShellSort(int *v, int n) {
  int i, j, atual;
  int h = 1;
  while(h < n) h = 3*h+1;
  while (h > 0) {
    for(i = h; i < n; i++) {
      atual = v[i];
      j = i;
      while (j > h-1 && atual >= v[j - h]) {
        v[j] = v[j - h];
        j = j - h;
      }
      v[j] = atual;
    }
    h = h/3;
```

```
  }
}

#endif
```


[lucascosta@fedora 1.2]$ gcc main.c -o main
[lucascosta@fedora 1.2]$ ./main
Quantidade de elementos a serem ordenados: 5
Digite os 5 inteiros a serem ordenados:
15
75
1
68
6
Valores ordenados pelo Heap Sort: 75 68 15 6 1
Valores ordenados pelo Merge Sort: 75 68 15 6 1
Valores ordenados pelo Quick Sort: 75 68 15 6 1
Valores ordenados pelo Shell Sort: 75 68 15 6 1
[lucascosta@fedora 1.2]$

**1.3/desempenho.h**

```c
#ifndef DESEMPENHO_H
#define DESEMPENHO_H

#include "Heap.h"
#include "Merge.h"
#include "Quick.h"
#include "Shell.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>

void medirDesempenho(int *valores, int tamanho, char *tipoOrdenacao) {
    int *copiaValores = (int *)malloc(tamanho * sizeof(int));
    for (int i = 0; i < tamanho; ++i) {
        copiaValores[i] = valores[i];
    }

    clock_t inicio, fim;
    double tempoExecucao;
    long comp = 0, mov = 0;

    inicio = clock();
    if (strcmp(tipoOrdenacao, "Heap") == 0) {
        HeapSort(copiaValores, tamanho, &comp, &mov);
    } else if (strcmp(tipoOrdenacao, "Merge") == 0) {
        MergeSort(copiaValores, 0, tamanho-1, &comp, &mov);
    } else if (strcmp(tipoOrdenacao, "Quick") == 0) {
        QuickSort(copiaValores, 0, tamanho-1, tamanho, &comp, &mov);
    } else if (strcmp(tipoOrdenacao, "Shell") == 0) {
        ShellSort(copiaValores, tamanho, &comp, &mov);
    }

    else {
        printf("Algoritmo de ordenação não está disponível\n");
        free(copiaValores);
        return;
    }
    fim = clock();

    tempoExecucao = ((double)(fim - inicio)) / CLOCKS_PER_SEC;

    printf("Tipo de ordenacao: %s\n", tipoOrdenacao);
    printf("Tempo de execucao: %f segundos\n", tempoExecucao);
    printf("Numero de comparações: %ld\n", comp);
    printf("Numero de movimentações: %ld\n\n", mov);

    free(copiaValores);
}


#endif
```

**1.3/Heap.h**

```

#ifndef HEAP_H
#define HEAP_H

#include <stdio.h>
#include <stdlib.h>
#include <time.h>


int* copiaVetorHeap(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 imprimeVetorHeap(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");
}

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

void criaHeap(int *v, int pai, int fim, long* comp, long* mov){
    int aux = v[pai];
    int filho = 2*pai + 1;
    while(filho <= fim){
        if(filho < fim && v[filho] < v[filho+1]){
            filho++;
        }
        (*comp)++; // Contabiliza a comparação

        if(aux < v[filho]){
            v[pai] = v[filho];
            pai = filho;
            filho = 2*pai + 1;
            (*mov)++; // Contabiliza a movimentação
        } else {
            filho = fim + 1;
        }
    }
```

```
58      v[pai] = aux;
59  }
60
61  void HeapSort(int *v, int n, long* comp, long* mov){
62      int i;
63      for(i=(n-1)/2; i>=0; i--)
64          criaHeap(v, i, n-1, comp, mov);
65      for(i=n-1; i>=1; i--){
66          trocaHeap(&v[0], &v[i]);
67          criaHeap(v, 0, i-1, comp, mov);
68      }
69  }
70
71
72  #endif
```

**1.3/main.c**

```c

#include <stdio.h>
#include <stdlib.h>
#include "desempenho.h"
#include "Heap.h"
#include "Merge.h"
#include "Quick.h"
#include "Shell.h"

int main(int argc, char *argv[]) {
    if (argc != 2) {
        printf("Uso: %s <nome_do_arquivo>\n", argv[0]);
        return 1;
    }

    FILE *arquivo = fopen(argv[1], "r");
    if (arquivo == NULL) {
        printf("Erro ao abrir o arquivo.\n");
        return 1;
    }

    int tamanho;
    fscanf(arquivo, "%d", &tamanho);

    int *valores = (int *)malloc(tamanho * sizeof(int));

    for (int i = 0; i < tamanho; i++) {
        fscanf(arquivo, "%d", &valores[i]);
    }

    fclose(arquivo);


    medirDesempenho(valores, tamanho, "Heap");
    medirDesempenho(valores, tamanho, "Merge");
    medirDesempenho(valores, tamanho, "Quick");
    medirDesempenho(valores, tamanho, "Shell");


    free(valores);

    return 0;
}
```

**1.3/Merge.h**

```c
/*---------------- File: Merge.c  ----------------------+
|Merge Sort                                             |
|                                                       |
|                                                       |
| Implementado por Guilherme C. Pena em 08/11/2023      |
+-------------------------------------------------------+ */
#ifndef MERGE_H
#define MERGE_H


#include <stdio.h>
#include <stdlib.h>
#include <time.h>


int* copiaVetorMerge(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 imprimeVetorMerge(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");
}

void preencheAleatorioMerge(int* v, int n, int ini, int fim){
    int i;
    for(i=0; i<n; i++)
        v[i] = ini + rand() % (fim-ini + 1);
}

void trocaMerge(int* a, int *b, long* mov){
    int aux = *a;
    *a = *b;
    *b = aux;
    (*mov)++;
}

void Merge(int *v, int ini, int meio, int fim, long* comp, long* mov){
    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) {
        (*comp)++;
        if (v[i] < v[j]) {
            A[k] = v[i];
            i++;
        } else {
```

```
58              A[k] = v[j];
59              j++;
60          }
61          k++;
62      }
63
64      while (i <= meio) {
65          A[k] = v[i];
66          i++;
67          k++;
68      }
69
70      while (j <= fim) {
71          A[k] = v[j];
72          j++;
73          k++;
74      }
75
76      for (i = ini, k = 0; i <= fim; i++, k++) {
77          v[i] = A[k];
78          (*mov)++;
79      }
80
81      free(A);
82  }
83
84  void MergeSort(int *v, int ini, int fim, long* comp, long* mov){
85      if(ini < fim ){
86          int meio = (ini + fim)/2;
87          MergeSort(v, ini, meio, comp, mov);
88          MergeSort(v, meio+1, fim, comp, mov);
89          Merge(v, ini, meio, fim, comp, mov);
90      }
91  }
92
93  #endif
```

**1.3/Quick.h**

```c
/*---------------- File: Quick.c  ----------------------+
|Quick Sort                                             |
|                                                       |
|                                                       |
| Implementado por Guilherme C. Pena em 14/11/2023      |
+-------------------------------------------------------+ */
#ifndef QUICK_H
#define QUICK_H


#include <stdio.h>
#include <stdlib.h>
#include <time.h>


int* copiaVetorQuick(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 imprimeVetorQuick(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");
}

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

int particao(int *v, int ini, int fim, long *comp, long *mov){
    int i = ini, j = fim;
    int pivo = v[(ini+fim)/2];
    while (1) {

        while(v[i] < pivo){ i++; (*comp)++; } // Incrementa comparação

        while(v[j] > pivo){ j--; (*comp)++; } // Incrementa comparação

        if(i < j){
            (*comp)++;
            (*mov)++;
```

```
58              trocaQuick(&v[i], &v[j]); // Troca os elementos encontrados
59              i++;
60              j--;
61          } else {
62              return j; // Retorna o local onde foi feita a partição
63          }
64      }
65  }
66
67  void QuickSort(int *v, int ini, int fim, int n, long *comp, long *mov){
68      if(ini < fim ){
69          int q = particao(v, ini, fim, comp, mov);
70          QuickSort(v, ini, q, n, comp, mov);
71          QuickSort(v, q+1, fim, n, comp, mov);
72      }
73  }
74
75  #endif
```

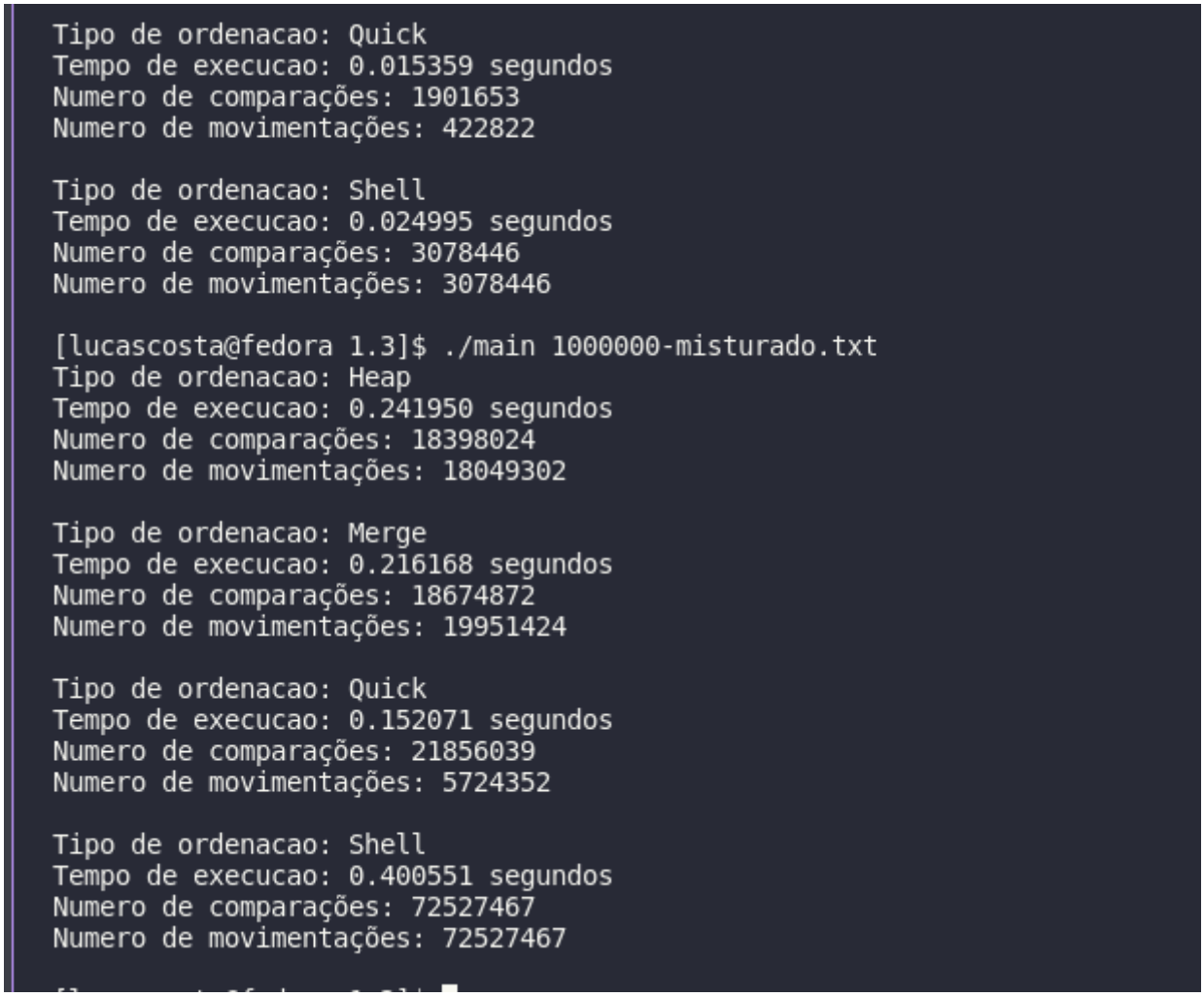
## 1.3/Shell.h

```c
/*---------------- File: Shell.c  ------------------+
|Shell Sort                                         |
|                                                                  |
|                                                                  |
| Implementado por Guilherme C. Pena em 20/11/2023      |
+---------------------------------------------------+ */

#ifndef SHELL_H
#define SHELL_H

#include <stdio.h>
#include <stdlib.h>
#include <time.h>


int* copiaVetorShell(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 imprimeVetorShell(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");
}

void preencheAleatorioShell(int* v, int n, int ini, int fim){
    int i;
    for(i=0; i<n; i++)
        v[i] = ini + rand() % (fim-ini + 1);
}

void trocaShell(int* a, int *b, long* mov){
    int aux = *a;
    *a = *b;
    *b = aux;
    (*mov)++;
}

void ShellSort(int *v, int n, long* comp, long* mov) {
  int i, j, atual;
  int h = 1;
  while(h < n) h = 3*h+1;
  while (h > 0) {
    for(i = h; i < n; i++) {
      atual = v[i];
      j = i;
      while (j > h-1 && atual <= v[j - h]) {
        v[j] = v[j - h];
        j = j - h;
        (*comp)++;
```

```
58          (*mov)++;
59        }
60        v[j] = atual;
61      }
62      h = h/3;
63    }
64  }
65
66
67  #endif
```

MISTURADO

```
[lucascosta@fedora 1.3]$ ./main 100-misturado.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.000042 segundos
Numero de comparações: 516
Numero de movimentações: 477

Tipo de ordenacao: Merge
Tempo de execucao: 0.000066 segundos
Numero de comparações: 542
Numero de movimentações: 672

Tipo de ordenacao: Quick
Tempo de execucao: 0.000037 segundos
Numero de comparações: 575
Numero de movimentações: 164

Tipo de ordenacao: Shell
Tempo de execucao: 0.000030 segundos
Numero de comparações: 451
Numero de movimentações: 451

[lucascosta@fedora 1.3]$ ./main 1000-misturado.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.000479 segundos
Numero de comparações: 8441
Numero de movimentações: 8106

Tipo de ordenacao: Merge
Tempo de execucao: 0.000561 segundos
Numero de comparações: 8735
Numero de movimentações: 9976

Tipo de ordenacao: Quick
Tempo de execucao: 0.000417 segundos
Numero de comparações: 9588
Numero de movimentações: 2414

Tipo de ordenacao: Shell
Tempo de execucao: 0.000444 segundos
Numero de comparações: 9436
Numero de movimentações: 9436

[lucascosta@fedora 1.3]$ ./main 10000-misturado.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.005935 segundos
Numero de comparações: 117779
Numero de movimentações: 114395

Tipo de ordenacao: Merge
Tempo de execucao: 0.007007 segundos
Numero de comparações: 120541
Numero de movimentações: 133616
```


Tipo de ordenacao: Quick
Tempo de execucao: 0.015359 segundos
Numero de comparações: 1901653
Numero de movimentações: 422822
Tipo de ordenacao: Shell
Tempo de execucao: 0.024995 segundos
Numero de comparações: 3078446
Numero de movimentações: 3078446
[lucascosta@fedora 1.3]$ ./main 1000000-misturado.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.241950 segundos
Numero de comparações: 18398024
Numero de movimentações: 18049302
Tipo de ordenacao: Merge
Tempo de execucao: 0.216168 segundos
Numero de comparações: 18674872
Numero de movimentações: 19951424
Tipo de ordenacao: Quick
Tempo de execucao: 0.152071 segundos
Numero de comparações: 21856039
Numero de movimentações: 5724352
Tipo de ordenacao: Shell
Tempo de execucao: 0.400551 segundos
Numero de comparações: 72527467
Numero de movimentações: 72527467

ORDENADO

```
[lucascosta@fedora 1.3]$ ./main 100-ordenado.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.000039 segundos
Numero de comparações: 547
Numero de movimentações: 541

Tipo de ordenacao: Merge
Tempo de execucao: 0.000041 segundos
Numero de comparações: 356
Numero de movimentações: 672

Tipo de ordenacao: Quick
Tempo de execucao: 0.000021 segundos
Numero de comparações: 573
Numero de movimentações: 0

Tipo de ordenacao: Shell
Tempo de execucao: 0.000007 segundos
Numero de comparações: 0
Numero de movimentações: 0

[lucascosta@fedora 1.3]$ ./main 1000-ordenado.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.000378 segundos
Numero de comparações: 8813
Numero de movimentações: 8709

Tipo de ordenacao: Merge
Tempo de execucao: 0.000321 segundos
Numero de comparações: 5044
Numero de movimentações: 9976

Tipo de ordenacao: Quick
Tempo de execucao: 0.000126 segundos
Numero de comparações: 8977
Numero de movimentações: 0

Tipo de ordenacao: Shell
Tempo de execucao: 0.000074 segundos
Numero de comparações: 0
Numero de movimentações: 0

[lucascosta@fedora 1.3]$ ./main 10000-ordenado.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.001334 segundos
Numero de comparações: 122288
Numero de movimentações: 121957

Tipo de ordenacao: Merge
Tempo de execucao: 0.001114 segundos
Numero de comparações: 69008
Numero de movimentações: 133616
```

```
Tipo de ordenacao: Quick
Tempo de execucao: 0.000444 segundos
Numero de comparações: 123617
Numero de movimentações: 0

Tipo de ordenacao: Shell
Tempo de execucao: 0.000294 segundos
Numero de comparações: 0
Numero de movimentações: 0

[lucascosta@fedora 1.3]$ ./main 100000-ordenado.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.015531 segundos
Numero de comparações: 1556441
Numero de movimentações: 1550855

Tipo de ordenacao: Merge
Tempo de execucao: 0.013502 segundos
Numero de comparações: 853904
Numero de movimentações: 1668928

Tipo de ordenacao: Quick
Tempo de execucao: 0.004894 segundos
Numero de comparações: 1568929
Numero de movimentações: 0

Tipo de ordenacao: Shell
Tempo de execucao: 0.003691 segundos
Numero de comparações: 0
Numero de movimentações: 0

[lucascosta@fedora 1.3]$ ./main 1000000-ordenado.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.147113 segundos
Numero de comparações: 18864660
Numero de movimentações: 18787793

Tipo de ordenacao: Merge
Tempo de execucao: 0.128875 segundos
Numero de comparações: 10066432
Numero de movimentações: 19951424

Tipo de ordenacao: Quick
Tempo de execucao: 0.058471 segundos
Numero de comparações: 18951425
Numero de movimentações: 0

Tipo de ordenacao: Shell
Tempo de execucao: 0.044743 segundos
Numero de comparações: 0
Numero de movimentações: 0
```

CONTRARIO

```
[lucascosta@fedora 1.3]$ ./main 100-contrario.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.000035 segundos
Numero de comparações: 478
Numero de movimentações: 417

Tipo de ordenacao: Merge
Tempo de execucao: 0.000034 segundos
Numero de comparações: 316
Numero de movimentações: 672

Tipo de ordenacao: Quick
Tempo de execucao: 0.000016 segundos
Numero de comparações: 524
Numero de movimentações: 50

Tipo de ordenacao: Shell
Tempo de execucao: 0.000013 segundos
Numero de comparações: 230
Numero de movimentações: 230

[lucascosta@fedora 1.3]$ ./main 1000-contrario.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.000370 segundos
Numero de comparações: 7991
Numero de movimentações: 7317

Tipo de ordenacao: Merge
Tempo de execucao: 0.000328 segundos
Numero de comparações: 4932
Numero de movimentações: 9976

Tipo de ordenacao: Quick
Tempo de execucao: 0.000129 segundos
Numero de comparações: 8478
Numero de movimentações: 500

Tipo de ordenacao: Shell
Tempo de execucao: 0.000135 segundos
Numero de comparações: 3920
Numero de movimentações: 3920

[lucascosta@fedora 1.3]$ ./main 10000-contrario.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.003321 segundos
Numero de comparações: 113360
Numero de movimentações: 106697

Tipo de ordenacao: Merge
Tempo de execucao: 0.001166 segundos
Numero de comparações: 64608
Numero de movimentações: 133616
```

.

```
Tipo de ordenacao: Quick
Tempo de execucao: 0.000459 segundos
Numero de comparações: 118618
Numero de movimentações: 5000

Tipo de ordenacao: Shell
Tempo de execucao: 0.000512 segundos
Numero de comparações: 53704
Numero de movimentações: 53704

[lucascosta@fedora 1.3]$ ./main 100000-contrario.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.015510 segundos
Numero de comparações: 1463377
Numero de movimentações: 1397435

Tipo de ordenacao: Merge
Tempo de execucao: 0.012349 segundos
Numero de comparações: 815024
Numero de movimentações: 1668928

Tipo de ordenacao: Quick
Tempo de execucao: 0.004790 segundos
Numero de comparações: 1518930
Numero de movimentações: 50000

Tipo de ordenacao: Shell
Tempo de execucao: 0.006454 segundos
Numero de comparações: 619654
Numero de movimentações: 619654

[lucascosta@fedora 1.3]$ ./main 1000000-contrario.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.150025 segundos
Numero de comparações: 18001491
Numero de movimentações: 17333409

Tipo de ordenacao: Merge
Tempo de execucao: 0.131240 segundos
Numero de comparações: 9884992
Numero de movimentações: 19951424

Tipo de ordenacao: Quick
Tempo de execucao: 0.059825 segundos
Numero de comparações: 18451426
Numero de movimentações: 500000

Tipo de ordenacao: Shell
Tempo de execucao: 0.072160 segundos
Numero de comparações: 6245384
Numero de movimentações: 6245384
```

.

QUASE-ORDENADO

```
[lucascosta@fedora 1.3]$ ./main 100-quaseordenado.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.000040 segundos
Numero de comparações: 545
Numero de movimentações: 536

Tipo de ordenacao: Merge
Tempo de execucao: 0.000045 segundos
Numero de comparações: 462
Numero de movimentações: 672

Tipo de ordenacao: Quick
Tempo de execucao: 0.000019 segundos
Numero de comparações: 568
Numero de movimentações: 9

Tipo de ordenacao: Shell
Tempo de execucao: 0.000020 segundos
Numero de comparações: 207
Numero de movimentações: 207

[lucascosta@fedora 1.3]$ ./main 1000-quaseordenado.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.000387 segundos
Numero de comparações: 8801
Numero de movimentações: 8676

Tipo de ordenacao: Merge
Tempo de execucao: 0.000358 segundos
Numero de comparações: 6709
Numero de movimentações: 9976

Tipo de ordenacao: Quick
Tempo de execucao: 0.000140 segundos
Numero de comparações: 8962
Numero de movimentações: 15

Tipo de ordenacao: Shell
Tempo de execucao: 0.000196 segundos
Numero de comparações: 2563
Numero de movimentações: 2563

[lucascosta@fedora 1.3]$ ./main 10000-quaseordenado.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.003767 segundos
Numero de comparações: 122275
Numero de movimentações: 121845

Tipo de ordenacao: Merge
Tempo de execucao: 0.001089 segundos
Numero de comparações: 85149
Numero de movimentações: 133616
```

.

```
Tipo de ordenacao: Quick
Tempo de execucao: 0.000427 segundos
Numero de comparações: 123601
Numero de movimentações: 16

Tipo de ordenacao: Shell
Tempo de execucao: 0.000629 segundos
Numero de comparações: 31498
Numero de movimentações: 31498

[lucascosta@fedora 1.3]$ ./main 100000-quaseordenado.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.015504 segundos
Numero de comparações: 1556425
Numero de movimentações: 1550657

Tipo de ordenacao: Merge
Tempo de execucao: 0.013457 segundos
Numero de comparações: 910980
Numero de movimentações: 1668928

Tipo de ordenacao: Quick
Tempo de execucao: 0.004715 segundos
Numero de comparações: 1568903
Numero de movimentações: 26

Tipo de ordenacao: Shell
Tempo de execucao: 0.005291 segundos
Numero de comparações: 108386
Numero de movimentações: 108386

[lucascosta@fedora 1.3]$ ./main 1000000-quaseordenado.txt
Tipo de ordenacao: Heap
Tempo de execucao: 0.146159 segundos
Numero de comparações: 18863195
Numero de movimentações: 18788703

Tipo de ordenacao: Merge
Tempo de execucao: 0.129805 segundos
Numero de comparações: 10131686
Numero de movimentações: 19951424

Tipo de ordenacao: Quick
Tempo de execucao: 0.059043 segundos
Numero de comparações: 18951397
Numero de movimentações: 28

Tipo de ordenacao: Shell
Tempo de execucao: 0.045392 segundos
Numero de comparações: 110062
Numero de movimentações: 110062

[lucascosta@fedora 1.3]$
```