

1.1/Heap.h

```
1
2 #ifndef HEAP_H
3 #define HEAP_H
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <time.h>
8
9 //Medidas de Complexidade
10 int comp; //Num. de comparacoes
11 int mov; //Num. de movimentacoes
12
13 int* copiaVetorHeap(int* v, int n){
14     int i;
15     int *v2;
16     v2 = (int*) malloc (n*sizeof(int));
17     for(i=0; i<n; i++) v2[i] = v[i];
18     return v2;
19 }
20 void imprimeVetorHeap(int* v, int n){
21     int i, prim = 1;
22     printf("[");
23     for(i=0; i<n; i++)
24         if(prim){ printf("%d", v[i]); prim = 0; }
25         else printf(", %d", v[i]);
26     printf("]\n");
27 }
28
29 void preencheAleatorioHeap(int* v, int n, int ini, int fim){
30     int i;
31     for(i=0; i<n; i++){
32         v[i] = ini + rand() % (fim-ini + 1);
33         //v[i] = (n-i); //Para o pior caso
34     }
35 }
36
37 void trocaHeap(int* a, int *b){
38     int aux = *a;
39     *a = *b;
40     *b = aux;
41 }
42
43 void criaHeap(int *v, int pai, int fim){
44     int aux = v[pai];
45     int filho = 2*pai + 1;
46     while(filho <= fim){
47         if(filho < fim)
48             if(v[filho] < v[filho+1])
49                 filho++;
50         if(aux < v[filho]){
51             v[pai] = v[filho];
52             pai = filho;
53             filho = 2*pai + 1;
54         }else filho = fim + 1;
55     }
56     v[pai] = aux;
57 }
```

```
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

```
1  #include <stdio.h>
2  #include "Heap.h"
3  #include "Merge.h"
4  #include "Quick.h"
5  #include "Shell.h"
6
7
8  int main(){
9
10     int n;
11     printf("Quantidade de elementos a serem ordenados: ");
12     scanf("%d", &n);
13
14
15     int vetor[n];
16     printf("Digite os %d inteiros a serem ordenados:\n", n);
17     for (int i = 0; i < n; i++) {
18         scanf("%d", &vetor[i]);
19     }
20
21     int heap[n], merge[n], quick[n], shell[n];
22     for (int i = 0; i < n; i++) {
23         heap[i] = vetor[i];
24         merge[i] = vetor[i];
25         quick[i] = vetor[i];
26         shell[i] = vetor[i];
27     }
28
29     HeapSort(heap, n);
30     MergeSort(merge, 0, n-1);
31     QuickSort(quick, 0, n-1, n);
32     ShellSort(shell, n);
33
34     printf("\nValores ordenados pelo Heap Sort: ");
35     for (int i = 0; i < n; i++) {
36         printf("%d ", heap[i]);
37     }
38
39     printf("\nValores ordenados pelo Merge Sort: ");
40     for (int i = 0; i < n; i++) {
41         printf("%d ", merge[i]);
42     }
43
44     printf("\nValores ordenados pelo Quick Sort: ");
45     for (int i = 0; i < n; i++) {
46         printf("%d ", quick[i]);
47     }
48
49     printf("\nValores ordenados pelo Shell Sort: ");
50     for (int i = 0; i < n; i++) {
51         printf("%d ", shell[i]);
52     }
53
54     printf("\n");
55
56
57     return 0;
```

```
58 | }  
59 |
```

1.1/Merge.h

```

1  /*----- File: Merge.c -----+
2  |Merge Sort                      |
3  |                               |
4  |                               |
5  | Implementado por Guilherme C. Pena em 08/11/2023 |
6  +-----+ */
7  #ifndef MERGE_H
8  #define MERGE_H
9
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <time.h>
14
15 //Medidas de Complexidade
16 int comp; //Num. de comparacoes
17 int mov; //Num. de movimentacoes
18
19 int* copiaVetorMerge(int* v, int n){
20     int i;
21     int *v2;
22     v2 = (int*) malloc (n*sizeof(int));
23     for(i=0; i<n; i++) v2[i] = v[i];
24     return v2;
25 }
26 void imprimeVetorMerge(int* v, int n){
27     int i, prim = 1;
28     printf("[");
29     for(i=0; i<n; i++)
30         if(prim){ printf("%d", v[i]); prim = 0; }
31         else printf(", %d", v[i]);
32     printf("]\n");
33 }
34
35 void preencheAleatorioMerge(int* v, int n, int ini, int fim){
36     int i;
37     for(i=0; i<n; i++)
38         v[i] = ini + rand() % (fim-ini + 1);
39 }
40
41 void trocaMerge(int* a, int *b){
42     int aux = *a;
43     *a = *b;
44     *b = aux;
45 }
46
47 void Merge(int *v, int ini, int meio, int fim){
48     int tam = fim-ini+1;
49     //Vetor Auxiliar - A
50     int *A = (int*) malloc (tam*sizeof(int));
51     int i = ini, j = meio+1, k = 0;
52     while (i<=meio && j<=fim) {
53         if (v[i] < v[j]){ A[k] = v[i]; i++; }
54         else { A[k] = v[j]; j++; }
55         k++;
56     }
57     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

```

1  /*----- File: Quick.c -----+
2  |Quick Sort                      |
3  |                               |
4  |                               |
5  | Implementado por Guilherme C. Pena em 14/11/2023 |
6  +-----+ */
7  #ifndef QUICK_H
8  #define QUICK_H
9
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <time.h>
14
15 //Medidas de Complexidade
16 int comp; //Num. de comparacoes
17 int mov; //Num. de movimentacoes
18
19 int* copiaVetorQuick(int* v, int n){
20     int i;
21     int *v2;
22     v2 = (int*) malloc (n*sizeof(int));
23     for(i=0; i<n; i++) v2[i] = v[i];
24     return v2;
25 }
26 void imprimeVetorQuick(int* v, int n){
27     int i, prim = 1;
28     printf("[");
29     for(i=0; i<n; i++)
30         if(prim){ printf("%d", v[i]); prim = 0; }
31         else printf(", %d", v[i]);
32     printf("]\n");
33 }
34
35 void preencheAleatorioQuick(int* v, int n, int ini, int fim){
36     int i;
37     for(i=0; i<n; i++){
38         v[i] = ini + rand() % (fim-ini + 1);
39         //v[i] = (n-i); //Para o pior caso
40     }
41 }
42
43 void trocaQuick(int* a, int *b){
44     int aux = *a;
45     *a = *b;
46     *b = aux;
47 }
48
49 //Versao do livro
50 int particiona(int *v, int ini, int fim){
51     int esq, dir, pivo, aux;
52     esq = ini; dir = fim;
53     pivo = v[ini];
54     while(esq < dir){
55         while(esq <= fim && v[esq] <= pivo) esq++;
56         while(dir >= 0 && v[dir] > pivo) dir--;
57         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

```

1  /*----- File: Shell.c -----+
2  |Shell Sort                               |
3  |                                         |
4  |                                         |
5  | Implementado por Guilherme C. Pena em 20/11/2023 |
6  +-----+ */
7
8  #ifndef SHELL_H
9  #define SHELL_H
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <time.h>
14
15 //Medidas de Complexidade
16 int comp; //Num. de comparacoes
17 int mov; //Num. de movimentacoes
18
19 int* copiaVetorShell(int* v, int n){
20     int i;
21     int *v2;
22     v2 = (int*) malloc (n*sizeof(int));
23     for(i=0; i<n; i++) v2[i] = v[i];
24     return v2;
25 }
26 void imprimeVetorShell(int* v, int n){
27     int i, prim = 1;
28     printf("[");
29     for(i=0; i<n; i++)
30         if(prim){ printf("%d", v[i]); prim = 0; }
31         else printf(", %d", v[i]);
32     printf("]\n");
33 }
34
35 void preencheAleatorioShell(int* v, int n, int ini, int fim){
36     int i;
37     for(i=0; i<n; i++)
38         v[i] = ini + rand() % (fim-ini + 1);
39 }
40
41 void trocaShell(int* a, int *b){
42     int aux = *a;
43     *a = *b;
44     *b = aux;
45 }
46
47 void ShellSort(int *v, int n) {
48     int i, j, atual;
49     int h = 1;
50     while(h < n) h = 3*h+1;
51     while (h > 0) {
52         for(i = h; i < n; i++) {
53             atual = v[i];
54             j = i;
55             while (j > h-1 && atual <= v[j - h]) {
56                 v[j] = v[j - h];
57                 j = j - h;

```

```
58 |     }  
59 |     v[j] = atual;  
60 |     }  
61 |     h = h/3;  
62 | }  
63 | }  
64 |  
65 | #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

```
1
2 #ifndef HEAP_H
3 #define HEAP_H
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <time.h>
8
9 //Medidas de Complexidade
10 int comp; //Num. de comparacoes
11 int mov; //Num. de movimentacoes
12
13 int* copiaVetorHeap(int* v, int n){
14     int i;
15     int *v2;
16     v2 = (int*) malloc (n*sizeof(int));
17     for(i=0; i<n; i++) v2[i] = v[i];
18     return v2;
19 }
20 void imprimeVetorHeap(int* v, int n){
21     int i, prim = 1;
22     printf("[");
23     for(i=0; i<n; i++)
24         if(prim){ printf("%d", v[i]); prim = 0; }
25         else printf(", %d", v[i]);
26     printf("]\n");
27 }
28
29 void preencheAleatorioHeap(int* v, int n, int ini, int fim){
30     int i;
31     for(i=0; i<n; i++){
32         v[i] = ini + rand() % (fim-ini + 1);
33         //v[i] = (n-i); //Para o pior caso
34     }
35 }
36
37 void trocaHeap(int* a, int *b){
38     int aux = *a;
39     *a = *b;
40     *b = aux;
41 }
42
43
44
45 void criaHeap(int *v, int pai, int fim){
46     int aux = v[pai];
47     int filho = 2*pai + 1;
48     while(filho <= fim){
49         if(filho < fim && v[filho] > v[filho+1]){
50             filho++;
51         }
52         if(aux > v[filho]){
53             v[pai] = v[filho];
54             pai = filho;
55             filho = 2*pai + 1;
56         }else {
57             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

```
1  #include <stdio.h>
2  #include "Heap.h"
3  #include "Merge.h"
4  #include "Quick.h"
5  #include "Shell.h"
6
7
8  int main(){
9
10     int n;
11     printf("Quantidade de elementos a serem ordenados: ");
12     scanf("%d", &n);
13
14
15     int vetor[n];
16     printf("Digite os %d inteiros a serem ordenados:\n", n);
17     for (int i = 0; i < n; i++) {
18         scanf("%d", &vetor[i]);
19     }
20
21     int heap[n], merge[n], quick[n], shell[n];
22     for (int i = 0; i < n; i++) {
23         heap[i] = vetor[i];
24         merge[i] = vetor[i];
25         quick[i] = vetor[i];
26         shell[i] = vetor[i];
27     }
28
29     HeapSort(heap, n);
30     MergeSort(merge, 0, n-1);
31     QuickSort(quick, 0, n-1, n);
32     ShellSort(shell, n);
33
34     printf("\nValores ordenados pelo Heap Sort: ");
35     for (int i = 0; i < n; i++) {
36         printf("%d ", heap[i]);
37     }
38
39     printf("\nValores ordenados pelo Merge Sort: ");
40     for (int i = 0; i < n; i++) {
41         printf("%d ", merge[i]);
42     }
43
44     printf("\nValores ordenados pelo Quick Sort: ");
45     for (int i = 0; i < n; i++) {
46         printf("%d ", quick[i]);
47     }
48
49     printf("\nValores ordenados pelo Shell Sort: ");
50     for (int i = 0; i < n; i++) {
51         printf("%d ", shell[i]);
52     }
53
54     printf("\n");
55
56
57     return 0;
```

```
58 | }  
59 |
```

1.2/Merge.h

```
1
2 #ifndef MERGE_H
3 #define MERGE_H
4
5
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <time.h>
9
10 //Medidas de Complexidade
11 int comp; //Num. de comparacoes
12 int mov; //Num. de movimentacoes
13
14 int* copiaVetorMerge(int* v, int n){
15     int i;
16     int *v2;
17     v2 = (int*) malloc (n*sizeof(int));
18     for(i=0; i<n; i++) v2[i] = v[i];
19     return v2;
20 }
21 void imprimeVetorMerge(int* v, int n){
22     int i, prim = 1;
23     printf("[");
24     for(i=0; i<n; i++)
25         if(prim){ printf("%d", v[i]); prim = 0; }
26         else printf(", %d", v[i]);
27     printf("]\n");
28 }
29
30 void preencheAleatorioMerge(int* v, int n, int ini, int fim){
31     int i;
32     for(i=0; i<n; i++)
33         v[i] = ini + rand() % (fim-ini + 1);
34 }
35
36 void trocaMerge(int* a, int *b){
37     int aux = *a;
38     *a = *b;
39     *b = aux;
40 }
41
42 void Merge(int *v, int ini, int meio, int fim){
43     int tam = fim - ini + 1;
44     int *A = (int*) malloc(tam * sizeof(int));
45     int i = ini, j = meio + 1, k = 0;
46
47     while (i <= meio && j <= fim) {
48         if (v[i] > v[j]) {
49             A[k] = v[i];
50             i++;
51         } else {
52             A[k] = v[j];
53             j++;
54         }
55         k++;
56     }
57 }
```



```
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

```
1
2 #ifndef QUICK_H
3 #define QUICK_H
4
5
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <time.h>
9
10 //Medidas de Complexidade
11 int comp; //Num. de comparacoes
12 int mov; //Num. de movimentacoes
13
14 int* copiaVetorQuick(int* v, int n){
15     int i;
16     int *v2;
17     v2 = (int*) malloc (n*sizeof(int));
18     for(i=0; i<n; i++) v2[i] = v[i];
19     return v2;
20 }
21 void imprimeVetorQuick(int* v, int n){
22     int i, prim = 1;
23     printf("[");
24     for(i=0; i<n; i++)
25         if(prim){ printf("%d", v[i]); prim = 0; }
26         else printf(", %d", v[i]);
27     printf("]\n");
28 }
29
30 void preencheAleatorioQuick(int* v, int n, int ini, int fim){
31     int i;
32     for(i=0; i<n; i++){
33         v[i] = ini + rand() % (fim-ini + 1);
34         //v[i] = (n-i); //Para o pior caso
35     }
36 }
37
38 void trocaQuick(int* a, int *b){
39     int aux = *a;
40     *a = *b;
41     *b = aux;
42 }
43
44 //Versao do livro
45 int particiona(int *v, int ini, int fim){
46     int esq, dir, pivo, aux;
47     esq = ini; dir = fim;
48     pivo = v[ini];
49     while(esq < dir){
50         while(esq <= fim && v[esq] <= pivo) esq++;
51         while(dir >= 0 && v[dir] > pivo) dir--;
52         if(esq < dir) trocaQuick(&v[esq], &v[dir]);
53     }
54     v[ini] = v[dir];
55     v[dir] = pivo;
56     return dir;
57 }
```

```
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
decrecente
65
66         comp++;
67         while(v[j] < pivo){ j--; comp++;} // Modificado para ordenar em ordem
decrecente
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 partiç o
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

```
1
2
3 #ifndef SHELL_H
4 #define SHELL_H
5
6 #include <stdio.h>
7 #include <stdlib.h>
8 #include <time.h>
9
10 //Medidas de Complexidade
11 int comp; //Num. de comparacoes
12 int mov; //Num. de movimentacoes
13
14 int* copiaVetorShell(int* v, int n){
15     int i;
16     int *v2;
17     v2 = (int*) malloc (n*sizeof(int));
18     for(i=0; i<n; i++) v2[i] = v[i];
19     return v2;
20 }
21 void imprimeVetorShell(int* v, int n){
22     int i, prim = 1;
23     printf("[");
24     for(i=0; i<n; i++)
25         if(prim){ printf("%d", v[i]); prim = 0; }
26         else printf(", %d", v[i]);
27     printf("]\n");
28 }
29
30 void preencheAleatorioShell(int* v, int n, int ini, int fim){
31     int i;
32     for(i=0; i<n; i++)
33         v[i] = ini + rand() % (fim-ini + 1);
34 }
35
36 void trocaShell(int* a, int *b){
37     int aux = *a;
38     *a = *b;
39     *b = aux;
40 }
41
42
43 void ShellSort(int *v, int n) {
44     int i, j, atual;
45     int h = 1;
46     while(h < n) h = 3*h+1;
47     while (h > 0) {
48         for(i = h; i < n; i++) {
49             atual = v[i];
50             j = i;
51             while (j > h-1 && atual >= v[j - h]) {
52                 v[j] = v[j - h];
53                 j = j - h;
54             }
55             v[j] = atual;
56         }
57         h = h/3;
```

```
58 |     }  
59 | }  
60 |  
61 | #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

```
1  #ifndef DESEMPENHO_H
2  #define DESEMPENHO_H
3
4  #include "Heap.h"
5  #include "Merge.h"
6  #include "Quick.h"
7  #include "Shell.h"
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <string.h>
11 #include <time.h>
12
13 void medirDesempenho(int *valores, int tamanho, char *tipoOrdenacao) {
14     int *copiaValores = (int *)malloc(tamanho * sizeof(int));
15     for (int i = 0; i < tamanho; ++i) {
16         copiaValores[i] = valores[i];
17     }
18
19     clock_t inicio, fim;
20     double tempoExecucao;
21     long comp = 0, mov = 0;
22
23     inicio = clock();
24     if (strcmp(tipoOrdenacao, "Heap") == 0) {
25         HeapSort(copiaValores, tamanho, &comp, &mov);
26     } else if (strcmp(tipoOrdenacao, "Merge") == 0) {
27         MergeSort(copiaValores, 0, tamanho-1, &comp, &mov);
28     } else if (strcmp(tipoOrdenacao, "Quick") == 0) {
29         QuickSort(copiaValores, 0, tamanho-1, tamanho, &comp, &mov);
30     } else if (strcmp(tipoOrdenacao, "Shell") == 0) {
31         ShellSort(copiaValores, tamanho, &comp, &mov);
32     }
33
34     else {
35         printf("Algoritmo de ordenação não está disponível\n");
36         free(copiaValores);
37         return;
38     }
39     fim = clock();
40
41     tempoExecucao = ((double)(fim - inicio)) / CLOCKS_PER_SEC;
42
43     printf("Tipo de ordenacao: %s\n", tipoOrdenacao);
44     printf("Tempo de execucao: %f segundos\n", tempoExecucao);
45     printf("Numero de comparações: %ld\n", comp);
46     printf("Numero de movimentações: %ld\n\n", mov);
47
48     free(copiaValores);
49 }
50
51
52 #endif
```

1.3/Heap.h

```
1
2 #ifndef HEAP_H
3 #define HEAP_H
4
5 #include <stdio.h>
6 #include <stdlib.h>
7 #include <time.h>
8
9
10 int* copiaVetorHeap(int* v, int n){
11     int i;
12     int *v2;
13     v2 = (int*) malloc (n*sizeof(int));
14     for(i=0; i<n; i++) v2[i] = v[i];
15     return v2;
16 }
17 void imprimeVetorHeap(int* v, int n){
18     int i, prim = 1;
19     printf("[");
20     for(i=0; i<n; i++)
21         if(prim){ printf("%d", v[i]); prim = 0; }
22         else printf(", %d", v[i]);
23     printf("]\n");
24 }
25
26 void preencheAleatorioHeap(int* v, int n, int ini, int fim){
27     int i;
28     for(i=0; i<n; i++){
29         v[i] = ini + rand() % (fim-ini + 1);
30         //v[i] = (n-i); //Para o pior caso
31     }
32 }
33
34 void trocaHeap(int* a, int *b){
35     int aux = *a;
36     *a = *b;
37     *b = aux;
38 }
39
40 void criaHeap(int *v, int pai, int fim, long* comp, long* mov){
41     int aux = v[pai];
42     int filho = 2*pai + 1;
43     while(filho <= fim){
44         if(filho < fim && v[filho] < v[filho+1]){
45             filho++;
46         }
47         (*comp)++; // Contabiliza a comparação
48
49         if(aux < v[filho]){
50             v[pai] = v[filho];
51             pai = filho;
52             filho = 2*pai + 1;
53             (*mov)++; // Contabiliza a movimentação
54         } else {
55             filho = fim + 1;
56         }
57     }
```



```
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     }
66     for(i=n-1; i>=1; i--){
67         trocaHeap(&v[0], &v[i]);
68         criaHeap(v, 0, i-1, comp, mov);
69     }
70 }
71
72 #endif
```

1.3/main.c

```
1
2 #include <stdio.h>
3 #include <stdlib.h>
4 #include "desempenho.h"
5 #include "Heap.h"
6 #include "Merge.h"
7 #include "Quick.h"
8 #include "Shell.h"
9
10 int main(int argc, char *argv[]) {
11     if (argc != 2) {
12         printf("Uso: %s <nome_do_arquivo>\n", argv[0]);
13         return 1;
14     }
15
16     FILE *arquivo = fopen(argv[1], "r");
17     if (arquivo == NULL) {
18         printf("Erro ao abrir o arquivo.\n");
19         return 1;
20     }
21
22     int tamanho;
23     fscanf(arquivo, "%d", &tamanho);
24
25     int *valores = (int *)malloc(tamanho * sizeof(int));
26
27     for (int i = 0; i < tamanho; i++) {
28         fscanf(arquivo, "%d", &valores[i]);
29     }
30
31     fclose(arquivo);
32
33
34     medirDesempenho(valores, tamanho, "Heap");
35     medirDesempenho(valores, tamanho, "Merge");
36     medirDesempenho(valores, tamanho, "Quick");
37     medirDesempenho(valores, tamanho, "Shell");
38
39
40     free(valores);
41
42     return 0;
43 }
```

1.3/Merge.h

```

1  /*----- File: Merge.c -----+
2  |Merge Sort                      |
3  |                               |
4  |                               |
5  | Implementado por Guilherme C. Pena em 08/11/2023 |
6  +-----+ */
7  #ifndef MERGE_H
8  #define MERGE_H
9
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <time.h>
14
15
16 int* copiaVetorMerge(int* v, int n){
17     int i;
18     int *v2;
19     v2 = (int*) malloc (n*sizeof(int));
20     for(i=0; i<n; i++) v2[i] = v[i];
21     return v2;
22 }
23
24 void imprimeVetorMerge(int* v, int n){
25     int i, prim = 1;
26     printf("[");
27     for(i=0; i<n; i++)
28         if(prim){ printf("%d", v[i]); prim = 0; }
29         else printf(", %d", v[i]);
30     printf("]\n");
31 }
32
33 void preencheAleatorioMerge(int* v, int n, int ini, int fim){
34     int i;
35     for(i=0; i<n; i++)
36         v[i] = ini + rand() % (fim-ini + 1);
37 }
38
39 void trocaMerge(int* a, int *b, long* mov){
40     int aux = *a;
41     *a = *b;
42     *b = aux;
43     (*mov)++;
44 }
45
46 void Merge(int *v, int ini, int meio, int fim, long* comp, long* mov){
47     int tam = fim - ini + 1;
48     // Vetor Auxiliar - A
49     int *A = (int*) malloc(tam * sizeof(int));
50     int i = ini, j = meio + 1, k = 0;
51
52     while (i <= meio && j <= fim) {
53         (*comp)++;
54         if (v[i] < v[j]) {
55             A[k] = v[i];
56             i++;
57         } 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

```

1  /*----- File: Quick.c -----+
2  |Quick Sort                      |
3  |                               |
4  |                               |
5  | Implementado por Guilherme C. Pena em 14/11/2023 |
6  +-----+ */
7  #ifndef QUICK_H
8  #define QUICK_H
9
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <time.h>
14
15
16 int* copiaVetorQuick(int* v, int n){
17     int i;
18     int *v2;
19     v2 = (int*) malloc (n*sizeof(int));
20     for(i=0; i<n; i++) v2[i] = v[i];
21     return v2;
22 }
23 void imprimeVetorQuick(int* v, int n){
24     int i, prim = 1;
25     printf("[");
26     for(i=0; i<n; i++)
27         if(prim){ printf("%d", v[i]); prim = 0; }
28         else printf(", %d", v[i]);
29     printf("]\n");
30 }
31
32 void preencheAleatorioQuick(int* v, int n, int ini, int fim){
33     int i;
34     for(i=0; i<n; i++){
35         v[i] = ini + rand() % (fim-ini + 1);
36         //v[i] = (n-i); //Para o pior caso
37     }
38 }
39
40 void trocaQuick(int* a, int *b){
41     int aux = *a;
42     *a = *b;
43     *b = aux;
44 }
45
46 int particao(int *v, int ini, int fim, long *comp, long *mov){
47     int i = ini, j = fim;
48     int pivo = v[(ini+fim)/2];
49     while (1) {
50
51         while(v[i] < pivo){ i++; (*comp)++; } // Incrementa comparação
52
53         while(v[j] > pivo){ j--; (*comp)++; } // Incrementa comparação
54
55         if(i < j){
56             (*comp)++;
57             (*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

```

1  /*----- File: Shell.c -----+
2  |Shell Sort                               |
3  |                                         |
4  |                                         |
5  | Implementado por Guilherme C. Pena em 20/11/2023 |
6  +-----+ */
7
8  #ifndef SHELL_H
9  #define SHELL_H
10
11 #include <stdio.h>
12 #include <stdlib.h>
13 #include <time.h>
14
15
16 int* copiaVetorShell(int* v, int n){
17     int i;
18     int *v2;
19     v2 = (int*) malloc (n*sizeof(int));
20     for(i=0; i<n; i++) v2[i] = v[i];
21     return v2;
22 }
23
24 void imprimeVetorShell(int* v, int n){
25     int i, prim = 1;
26     printf("[");
27     for(i=0; i<n; i++)
28         if(prim){ printf("%d", v[i]); prim = 0; }
29         else printf(", %d", v[i]);
30     printf("]\n");
31 }
32
33 void preencheAleatorioShell(int* v, int n, int ini, int fim){
34     int i;
35     for(i=0; i<n; i++)
36         v[i] = ini + rand() % (fim-ini + 1);
37 }
38
39 void trocaShell(int* a, int *b, long* mov){
40     int aux = *a;
41     *a = *b;
42     *b = aux;
43     (*mov)++;
44 }
45
46 void ShellSort(int *v, int n, long* comp, long* mov) {
47     int i, j, atual;
48     int h = 1;
49     while(h < n) h = 3*h+1;
50     while (h > 0) {
51         for(i = h; i < n; i++) {
52             atual = v[i];
53             j = i;
54             while (j > h-1 && atual <= v[j - h]) {
55                 v[j] = v[j - h];
56                 j = j - h;
57                 (*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

[lucascosta@fedora 1.3]\$

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]\$ █