roteiro-11/include/time.h

```
1 #ifndef TEMPO H
2 #define TEMPO_H
3
4 #include <stdio.h>
5
   #include <stdlib.h>
6
7
   typedef long double Time;
8
   Time getCpuTime();
9
   Time formatTime(long int sec, long int usec);
10
   void printElapsedTime(Time start, Time end);
11
12
13 #endif
```

roteiro-11/include/sort-module.h

```
1 #ifndef SORT MODULE H
2 #define SORT MODULE H
 3
 4 #include <stdio.h>
 5
   #include <stdlib.h>
   #include "./time.h"
 7
 8
   typedef int Key;
9
10
   typedef struct {
11
       long long comps, swaps;
12
       Time start, end;
13
   } Statistics;
14
15
16
   Statistics* createStatistics();
17
   void destroyStatistics(Statistics** s);
   void getElements(Key* arr, int quantity);
18
   int compare(Key a, Key b, int reverse, Statistics* statistics);
19
   void swap(Key* a, Key* b, Statistics* statistics);
20
   void print(Key* arr, int n);
21
22
23 #endif
```

roteiro-11/include/sort.h

```
1
   #ifndef SORT H
2
   #define SORT H
 3
4
   #include "./sort-module.h"
 5
6
   // Sorting essential functions
   void shellSort(Key* arr, int n, int reverse, Statistics*
7
   statistics);
   void quickSort(Key* arr, int n, int reverse, Statistics*
8
   statistics);
9
   void mergeSort(Key* arr, int n, int reverse, Statistics*
   statistics);
   void heapSort(Key* arr, int n, int reverse, Statistics*
10
   statistics);
   void getOptions(void (**sortFunction)(Key*, int, int,
11
   Statistics*), int* order, int* quantity);
12
13 #endif
```

roteiro-11/src/time.c

```
1 #include "../include/time.h"
3
   #include <sys/resource.h>
4
   // Returns CPU time in that moment
   Time getCpuTime() {
7
       struct rusage usage;
       getrusage(RUSAGE SELF, &usage);
9
       return formatTime(usage.ru utime.tv sec, usage.ru utime.tv usec);
10
11
   // Join time in seconds and microseconds into a long double variable
12
   Time formatTime(long int sec, long int usec) {
       Time totalTime = sec + ((Time)usec / 1000000.0L);
14
15
       return totalTime;
16 }
17
18 // Prints the difference between start to end
   void printElapsedTime(Time start, Time end) {
19
20
       Time elapsedTime = end - start;
21
       printf("%Lf", elapsedTime);
22 }
23
```

roteiro-11/src/sort-module.c

```
1 #include "./sort-module.h"
 2
 3 #include <stdio.h>
 4 #include <stdlib.h>
 5 #include <string.h>
 6
 7
   // Allocate a new statistics pointer
   Statistics* createStatistics() {
 8
        Statistics* new = (Statistics*)malloc(sizeof(Statistics));
 9
10
        new->comps = 0;
11
        new->swaps = 0;
12
        new->start = 0;
13
        new->end = 0;
14
        return new;
15
   }
16
   // Destroy statistics allocated pointer
17
   void destroyStatistics(Statistics** s) {
18
19
        if (*s == NULL) return;
20
        free(*s):
21
        *s = NULL;
22
   }
23
24
   // Get elements from user input
25
   void getElements(Key* arr, int quantity) {
26
        for (int i = 0; i < quantity; i++) {
            scanf("%d", &arr[i]);
27
28
        }
29
   }
30
31
   // Compare elements (a < b)</pre>
   int compare(Key a, Key b, int reverse, Statistics* statistics)
32
    {
33
        if (statistics != NULL) statistics->comps++;
34
        return (!reverse ? (a < b) : (a > b));
35
   }
36
37
   // Swap elements
   void swap(Key* a, Key* b, Statistics* statistics) {
38
39
        if (statistics != NULL) statistics->swaps++;
40
        Key aux = *a;
        *a = *b:
41
42
        *b = aux;
43
   }
```

```
44
45
   // Print all elements in the array
   void print(Key* arr, int n) {
46
47
        for (int i = 0; i < n; i++) {</pre>
            printf("%d ", arr[i]);
48
49
        }
        printf("\n");
50
51
   }
52
```

roteiro-11/src/sort.c

```
#include "../include/sort.h"
 1
 2
 3
   #include <stdio.h>
 4
   #include <stdlib.h>
 5
 6
   #pragma region SHELL SORT
 7
   void shellSort(Key* arr, int n, int reverse, Statistics* statistics) {
 8
        Key aux;
 9
        int h = 1;
10
11
        do {
12
            h = 3 * h + 1;
13
        } while (h < n);</pre>
14
15
        do {
16
            h /= 3;
17
            for (int i = h; i < n; i++) {
18
                aux = arr[i];
19
20
                int j = i;
                while (j \ge h \& ext{lem} (arr[j - h], aux, reverse, statistics)) {
21
22
                    arr[j] = arr[j - h];
23
                    statistics->swaps++;
24
                     j -= h;
25
                }
26
27
                arr[j] = aux;
28
29
        } while (h > 1);
30
31
   #pragma endregion
32
33
   #pragma region QUICK SORT
34
   Key partition(Key* arr, int low, int high, int reverse, Statistics* statistics)
35
        Key pivot = arr[high];
        int i = (low - 1);
36
37
38
        for (int j = low; j < high; j++) {
39
            if (!compare(pivot, arr[j], reverse, statistics)) {
40
41
                swap(&arr[i], &arr[j], statistics);
42
            }
43
        }
44
45
        swap(&arr[i + 1], &arr[high], statistics);
46
        return (i + 1);
47
    }
48
49
   void quickSortRecursion(Key* arr, int low, int high, int reverse, Statistics*
    statistics) {
50
        if (low < high) {</pre>
51
            int pi = partition(arr, low, high, reverse, statistics);
            quickSortRecursion(arr, low, pi - 1, reverse, statistics);
52
53
            quickSortRecursion(arr, pi + 1, high, reverse, statistics);
54
        }
```

```
55
    }
 56
57
    void quickSort(Key* arr, int n, int reverse, Statistics* statistics) {
         quickSortRecursion(arr, 0, n - 1, reverse, statistics);
 58
 59
 60
    #pragma endregion
 61
 62
    #pragma region MERGE SORT
 63
     void merge(Key arr[], int l, int m, int r, int reverse, Statistics* statistics)
 64
         int i, j, k;
 65
         int n1 = m - l + 1;
 66
         int n2 = r - m;
 67
 68
         int L[n1], R[n2];
 69
 70
         for (i = 0; i < n1; i++) {
71
             L[i] = arr[l + i];
 72
             statistics->swaps++;
 73
74
         for (j = 0; j < n2; j++) {
 75
             R[j] = arr[m + 1 + j];
 76
             statistics->swaps++;
77
         }
 78
 79
         i = 0;
 80
         j = 0;
 81
         k = l;
 82
         while (i < n1 && j < n2) {
 83
 84
             if (compare(L[i], R[j], reverse, statistics)) {
 85
                 arr[k] = L[i];
 86
                 statistics->swaps++;
 87
                 i++;
88
             } else {
 89
                 arr[k] = R[j];
 90
                 statistics->swaps++;
 91
                 j++;
 92
 93
             k++;
 94
         }
 95
 96
         while (i < n1) {
 97
             arr[k] = L[i];
 98
             statistics->swaps++;
 99
             i++;
100
             k++;
101
         }
102
103
         while (j < n2) {
104
             arr[k] = R[j];
105
             statistics->swaps++;
106
             j++;
107
             k++;
108
         }
109
     }
110
     void mergeSortRecursion(Key arr[], int l, int r, int reverse, Statistics*
111
     statistics) {
```

```
112
         if (l < r) {
             int m = l + (r - l) / 2;
113
114
             mergeSortRecursion(arr, l, m, reverse, statistics);
115
             mergeSortRecursion(arr, m + 1, r, reverse, statistics);
116
             merge(arr, l, m, r, reverse, statistics);
117
         }
118
     }
119
120
     void mergeSort(Key* arr, int n, int reverse, Statistics* statistics) {
121
         mergeSortRecursion(arr, 0, n - 1, reverse, statistics);
122
123
    #pragma endregion
124
125
    #pragma region HEAP SORT
     void heapify(Key arr[], int n, int i, int reverse, Statistics* statistics) {
126
127
         int largest = i;
         int left = 2 * i + 1;
128
129
         int right = 2 * i + 2;
130
131
         if (left < n && !compare(arr[left], arr[largest], reverse, statistics))</pre>
132
             largest = left;
133
134
         if (right < n && !compare(arr[right], arr[largest], reverse, statistics))</pre>
135
             largest = right;
136
137
         if (largest != i) {
138
             swap(&arr[i], &arr[largest], statistics);
139
             heapify(arr, n, largest, reverse, statistics);
140
         }
141
     }
142
143
     void heapSort(Key* arr, int n, int reverse, Statistics* statistics) {
144
         for (int i = n / 2 - 1; i >= 0; i - -)
             heapify(arr, n, i, reverse, statistics);
145
146
147
         for (int i = n - 1; i \ge 0; i - -) {
148
             swap(&arr[0], &arr[i], statistics);
149
             heapify(arr, i, 0, reverse, statistics);
150
         }
151
152
    #pragma endregion
153
154
    // Function to exit the program if there is an error
155
    void throwError(char* message) {
156
         printf("%s - Aborting...\n", message);
157
         exit(1);
158
    }
159
160
     // User select the sorting options
     void getOptions(void (**sortFunction)(Key*, int, int, Statistics*), int* order,
161
     int* quantity) {
         int option;
162
163
164
         // User selects the algorithm
165
         printf("1 - Shell Sort\n");
166
         printf("2 - Quick Sort\n");
167
         printf("3 - Merge Sort\n");
         printf("4 - Heap Sort\n");
168
169
         printf("Sorting Algorithm: ");
```

```
170
         scanf("%d", &option);
171
         switch (option) {
172
             case 1:
173
                 *sortFunction = &shellSort;
                 break;
174
175
             case 2:
176
                 *sortFunction = &quickSort;
177
                 break;
178
             case 3:
179
                 *sortFunction = &mergeSort;
180
181
             case 4:
182
                 *sortFunction = &heapSort;
183
                 break;
184
             default:
185
                 throwError("Invalid algorithm option");
186
         }
187
188
         printf("\n");
189
         // User selects the order
190
191
         printf("0 - Ascending\n");
192
         printf("1 - Descending\n");
193
         printf("Order: ");
         scanf("%d", &option);
194
195
         if (option != 0 \&\& option <math>!= 1)
196
             throwError("Invalid order option");
197
198
         *order = option;
199
200
         printf("\n");
201
202
         // User enters the quantity
203
         printf("Number of Elements: ");
204
         scanf("%d", quantity);
205
         if (quantity < 0)
206
             throwError("Invalid number");
207
208
         printf("\n");
209
     }
210
```

roteiro-11/src/main.c

```
1 #include <stdio.h>
 2
   #include <stdlib.h>
 3
   #include "../include/sort-module.h"
 4
 5
   #include "../include/sort.h"
   #include "../include/time.h"
 7
 8
   int main() {
        void (*sortFunction)(Key *, int, int, Statistics *);
 9
        int order, quantity;
10
        getOptions(&sortFunction, &order, &quantity);
11
12
        Key *arr = (Key *)malloc(quantity * sizeof(Key));
13
14
        getElements(arr, quantity);
15
16
        Statistics *s = createStatistics();
17
        s->start = getCpuTime();
18
        (*sortFunction)(arr, quantity, order, s);
19
        s->end = getCpuTime();
20
21
        print(arr, quantity);
22
23
        printf("\n== Statistics ==\n");
24
        printf("Elapsed time = ");
25
        printElapsedTime(s->start, s->end);
        printf("\nComparations = %Ld\n", s->comps);
26
        printf("Swaps = %Ld\n", s->swaps);
27
28
29
        destroyStatistics(&s);
30
        free(arr);
31
32
        return 0;
33
   }
```

```
gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-11 Q =
                                                                                                       gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-11 Q = -
 - Shell Sort
                                                                                             - Shell Sort
2 - Ouick Sort
                                                                                             - Ouick Sort
3 - Merge Sort
                                                                                           3 - Merge Sort
4 - Heap Sort
                                                                                             - Heap Sort
Sorting Algorithm: 1
                                                                                           Sorting Algorithm: 2
0 - Ascending
                                                                                             - Ascending
1 - Descending
                                                                                             - Descending
Order: 0
                                                                                           Order: 0
Number of Elements: 5
                                                                                           Number of Elements: 5
3 1 2 5 4
                                                                                           5 2 4 1 3
 1 2 3 4 5
                                                                                           1 2 3 4 5
== Statistics ==
                                                                                            == Statistics ==
Elapsed time = 0.000003
                                                                                           Elapsed time = 0.000002
                                                                                           Comparations = 6
Comparations = 7
Swaps = 3
                                                                                           Swaps = 5
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$ ./bin/out
                                                                                            gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$ ./bin/out
 l - Shell Sort
                                                                                             - Shell Sort
 - Ouick Sort
                                                                                             - Ouick Sort
 - Merge Sort
                                                                                             - Merge Sort
4 - Heap Sort
                                                                                             - Heap Sort
Sorting Algorithm: 1
                                                                                           Sorting Algorithm: 2
0 - Ascendina
                                                                                           0 - Ascendina
1 - Descending
                                                                                             - Descending
Order: 1
                                                                                           Order: 1
Number of Elements: 5
                                                                                            Number of Elements: 5
4 5 1 3 2
                                                                                           3 1 4 2 5
                                                                                           5 4 3 2 1
5 4 3 2 1
== Statistics ==
                                                                                            == Statistics ==
                                                                                           Elapsed time = 0.000003
Elapsed time = 0.000003
                                                                                            Comparations = 8
Comparations = 7
Swaps = 3
                                                                                            Swaps = 5
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$
                                                                                             abriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$
            gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-11 🔍 🚊 🖃
                                                                                                        gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-11 🔍 🗏 🖃
 gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$ ./bin/out
                                                                                             abriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$ ./bin/out
 - Shell Sort
                                                                                              - Shell Sort
 - Ouick Sort
                                                                                             - Quick Sort
3 - Merge Sort
                                                                                             - Merge Sort
 - Heap Sort
                                                                                             - Heap Sort
Sorting Algorithm: 3
                                                                                           Sorting Algorithm: 4
0 - Ascending
                                                                                             - Ascending
1 - Descending
                                                                                           1 - Descending
Order: 0
                                                                                           Order: 0
Number of Elements: 5
                                                                                           Number of Elements: 5
 5 4 3 1
                                                                                             1 2 5 4
 2 3 4 5
                                                                                             2 3 4 5
 == Statistics ==
                                                                                            == Statistics ==
Elapsed time = 0.000003
                                                                                           Elapsed time = 0.000002
Comparations = 7
                                                                                           Comparations = 12
Swaps = 24
                                                                                            Swaps = 10
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$ ./bin/out
1 - Shell Sort
                                                                                            gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$ ./bin/out
1 - Shell Sort
 - Quick Sort
                                                                                             - Quick Sort
3 - Merge Sort
                                                                                             - Merge Sort
4 - Heap Sort
                                                                                           4 - Heap Sort
Sorting Algorithm: 3
                                                                                           Sorting Algorithm: 4
0 - Ascending

    Ascending

 - Descending
                                                                                             - Descending
Order: 1
                                                                                           Order: 1
Number of Elements: 5
                                                                                           Number of Elements: 5
5 4 3 2 1
                                                                                           4 5 2 1 3
                                                                                           5 4 3 2 1
5 4 3 2 1
== Statistics ==
                                                                                            == Statistics ==
Elapsed time = 0.000003
                                                                                           Elapsed time = 0.000004
                                                                                            Comparations = 11
Comparations = 7
Swaps = 24
                                                                                            Swaps = 10
 gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$
                                                                                             abriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$
```

