roteiro-11/include/time.h

```c
#ifndef TEMPO_H
#define TEMPO_H

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

typedef long double Time;

Time getCpuTime();
Time formatTime(long int sec, long int usec);
void printElapsedTime(Time start, Time end);

#endif
```

roteiro-11/include/sort-module.h

```c
#ifndef SORT_MODULE_H
#define SORT_MODULE_H

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

#include "./time.h"

typedef int Key;

typedef struct {
    long long comps, swaps;
    Time start, end;
} Statistics;

Statistics* createStatistics();
void destroyStatistics(Statistics** s);
void getElements(Key* arr, int quantity);
int compare(Key a, Key b, int reverse, Statistics* statistics);
void swap(Key* a, Key* b, Statistics* statistics);
void print(Key* arr, int n);

#endif
```

roteiro-11/include/sort.h

```c
#ifndef SORT_H
#define SORT_H

#include "./sort-module.h"

// Sorting essential functions
void shellSort(Key* arr, int n, int reverse, Statistics*
statistics);
void quickSort(Key* arr, int n, int reverse, Statistics*
statistics);
void mergeSort(Key* arr, int n, int reverse, Statistics*
statistics);
void heapSort(Key* arr, int n, int reverse, Statistics*
statistics);
void getOptions(void (**sortFunction)(Key*, int, int,
Statistics*), int* order, int* quantity);

#endif
```

roteiro-11/src/time.c

```c
#include "../include/time.h"

#include <sys/resource.h>

// Returns CPU time in that moment
Time getCpuTime() {
    struct rusage usage;
    getrusage(RUSAGE_SELF, &usage);
    return formatTime(usage.ru_utime.tv_sec, usage.ru_utime.tv_usec);
}

// Join time in seconds and microseconds into a long double variable
Time formatTime(long int sec, long int usec) {
    Time totalTime = sec + ((Time)usec / 1000000.0L);
    return totalTime;
}

// Prints the difference between start to end
void printElapsedTime(Time start, Time end) {
    Time elapsedTime = end - start;
    printf("%Lf", elapsedTime);
}
```

roteiro-11/src/sort-module.c

```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
8   Statistics* createStatistics() {
9       Statistics* new = (Statistics*)malloc(sizeof(Statistics));
10      new->comps = 0;
11      new->swaps = 0;
12      new->start = 0;
13      new->end = 0;
14      return new;
15  }
16
17  // Destroy statistics allocated pointer
18  void destroyStatistics(Statistics** s) {
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++) {
27          scanf("%d", &arr[i]);
28      }
29  }
30
31  // Compare elements (a < b)
32  int compare(Key a, Key b, int reverse, Statistics* statistics)
    {
33      if (statistics != NULL) statistics->comps++;
34      return (!reverse ? (a < b) : (a > b));
35  }
36
37  // Swap elements
38  void swap(Key* a, Key* b, Statistics* statistics) {
39      if (statistics != NULL) statistics->swaps++;
40      Key aux = *a;
41      *a = *b;
42      *b = aux;
43  }
```

```c
// Print all elements in the array
void print(Key* arr, int n) {
    for (int i = 0; i < n; i++) {
        printf("%d ", arr[i]);
    }
    printf("\n");
}
```

roteiro-11/src/sort.c

```c
1   #include "../include/sort.h"
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);
14
15      do {
16          h /= 3;
17          for (int i = h; i < n; i++) {
18              aux = arr[i];
19
20              int j = i;
21              while (j >= h && !compare(arr[j - h], aux, reverse, statistics)) {
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];
36      int i = (low - 1);
37
38      for (int j = low; j < high; j++) {
39          if (!compare(pivot, arr[j], reverse, statistics)) {
40              i++;
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) {
51          int pi = partition(arr, low, high, reverse, statistics);
52          quickSortRecursion(arr, low, pi - 1, reverse, statistics);
53          quickSortRecursion(arr, pi + 1, high, reverse, statistics);
54      }
```

```c
55  }
56
57  void quickSort(Key* arr, int n, int reverse, Statistics* statistics) {
58      quickSortRecursion(arr, 0, n - 1, reverse, statistics);
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
83      while (i < n1 && j < n2) {
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
111 void mergeSortRecursion(Key arr[], int l, int r, int reverse, Statistics*
    statistics) {
```

```c
    if (l < r) {
        int m = l + (r - l) / 2;
        mergeSortRecursion(arr, l, m, reverse, statistics);
        mergeSortRecursion(arr, m + 1, r, reverse, statistics);
        merge(arr, l, m, r, reverse, statistics);
    }
}

void mergeSort(Key* arr, int n, int reverse, Statistics* statistics) {
    mergeSortRecursion(arr, 0, n - 1, reverse, statistics);
}
#pragma endregion

#pragma region HEAP_SORT
void heapify(Key arr[], int n, int i, int reverse, Statistics* statistics) {
    int largest = i;
    int left = 2 * i + 1;
    int right = 2 * i + 2;

    if (left < n && !compare(arr[left], arr[largest], reverse, statistics))
        largest = left;

    if (right < n && !compare(arr[right], arr[largest], reverse, statistics))
        largest = right;

    if (largest != i) {
        swap(&arr[i], &arr[largest], statistics);
        heapify(arr, n, largest, reverse, statistics);
    }
}

void heapSort(Key* arr, int n, int reverse, Statistics* statistics) {
    for (int i = n / 2 - 1; i >= 0; i--)
        heapify(arr, n, i, reverse, statistics);

    for (int i = n - 1; i >= 0; i--) {
        swap(&arr[0], &arr[i], statistics);
        heapify(arr, i, 0, reverse, statistics);
    }
}
#pragma endregion

// Function to exit the program if there is an error
void throwError(char* message) {
    printf("%s - Aborting...\n", message);
    exit(1);
}

// User select the sorting options
void getOptions(void (**sortFunction)(Key*, int, int, Statistics*), int* order,
int* quantity) {
    int option;

    // User selects the algorithm
    printf("1 - Shell Sort\n");
    printf("2 - Quick Sort\n");
    printf("3 - Merge Sort\n");
    printf("4 - Heap Sort\n");
    printf("Sorting Algorithm: ");
```

```c
170        scanf("%d", &option);
171        switch (option) {
172            case 1:
173                *sortFunction = &shellSort;
174                break;
175            case 2:
176                *sortFunction = &quickSort;
177                break;
178            case 3:
179                *sortFunction = &mergeSort;
180                break;
181            case 4:
182                *sortFunction = &heapSort;
183                break;
184            default:
185                throwError("Invalid algorithm option");
186        }
187
188        printf("\n");
189
190        // User selects the order
191        printf("0 - Ascending\n");
192        printf("1 - Descending\n");
193        printf("Order: ");
194        scanf("%d", &option);
195        if (option != 0 && option != 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

```c
#include <stdio.h>
#include <stdlib.h>

#include "../include/sort-module.h"
#include "../include/sort.h"
#include "../include/time.h"

int main() {
    void (*sortFunction)(Key *, int, int, Statistics *);
    int order, quantity;
    getOptions(&sortFunction, &order, &quantity);

    Key *arr = (Key *)malloc(quantity * sizeof(Key));
    getElements(arr, quantity);

    Statistics *s = createStatistics();
    s->start = getCpuTime();
    (*sortFunction)(arr, quantity, order, s);
    s->end = getCpuTime();

    print(arr, quantity);

    printf("\n== Statistics ==\n");
    printf("Elapsed time = ");
    printElapsedTime(s->start, s->end);
    printf("\nComparations = %Ld\n", s->comps);
    printf("Swaps = %Ld\n", s->swaps);

    destroyStatistics(&s);
    free(arr);

    return 0;
}
```

```
gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-11          Q  ≡  _  □  ×

gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$ ./bin/out
1 - Shell Sort
2 - Quick Sort
3 - Merge Sort
4 - Heap Sort
Sorting Algorithm: 1

0 - Ascending
1 - Descending
Order: 0

Number of Elements: 5

3 1 2 5 4
1 2 3 4 5

== Statistics ==
Elapsed time = 0.000003
Comparations = 7
Swaps = 3
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$ ./bin/out
1 - Shell Sort
2 - Quick Sort
3 - Merge Sort
4 - Heap Sort
Sorting Algorithm: 1

0 - Ascending
1 - Descending
Order: 1

Number of Elements: 5

4 5 1 3 2
5 4 3 2 1

== Statistics ==
Elapsed time = 0.000003
Comparations = 7
Swaps = 3
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$
```

```
gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-11          Q  ≡  _  □  ×

gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$ ./bin/out
1 - Shell Sort
2 - Quick Sort
3 - Merge Sort
4 - Heap Sort
Sorting Algorithm: 2

0 - Ascending
1 - Descending
Order: 0

Number of Elements: 5

5 2 4 1 3
1 2 3 4 5

== Statistics ==
Elapsed time = 0.000002
Comparations = 6
Swaps = 5
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$ ./bin/out
1 - Shell Sort
2 - Quick Sort
3 - Merge Sort
4 - Heap Sort
Sorting Algorithm: 2

0 - Ascending
1 - Descending
Order: 1

Number of Elements: 5

3 1 4 2 5
5 4 3 2 1

== Statistics ==
Elapsed time = 0.000003
Comparations = 8
Swaps = 5
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$
```

```
gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-11          Q  ≡  _  □  ×

gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$ ./bin/out
1 - Shell Sort
2 - Quick Sort
3 - Merge Sort
4 - Heap Sort
Sorting Algorithm: 3

0 - Ascending
1 - Descending
Order: 0

Number of Elements: 5

2 5 4 3 1
1 2 3 4 5

== Statistics ==
Elapsed time = 0.000003
Comparations = 7
Swaps = 24
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$ ./bin/out
1 - Shell Sort
2 - Quick Sort
3 - Merge Sort
4 - Heap Sort
Sorting Algorithm: 3

0 - Ascending
1 - Descending
Order: 1

Number of Elements: 5

5 4 3 2 1
5 4 3 2 1

== Statistics ==
Elapsed time = 0.000003
Comparations = 7
Swaps = 24
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$
```

```
gabriel-dp@gabriel-dp: ~/Desktop/dev/c/lab2/roteiro-11          Q  ≡  _  □  ×

gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$ ./bin/out
1 - Shell Sort
2 - Quick Sort
3 - Merge Sort
4 - Heap Sort
Sorting Algorithm: 4

0 - Ascending
1 - Descending
Order: 0

Number of Elements: 5

3 1 2 5 4
1 2 3 4 5

== Statistics ==
Elapsed time = 0.000002
Comparations = 12
Swaps = 10
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$ ./bin/out
1 - Shell Sort
2 - Quick Sort
3 - Merge Sort
4 - Heap Sort
Sorting Algorithm: 4

0 - Ascending
1 - Descending
Order: 1

Number of Elements: 5

4 5 2 1 3
5 4 3 2 1

== Statistics ==
Elapsed time = 0.000004
Comparations = 11
Swaps = 10
gabriel-dp@gabriel-dp:~/Desktop/dev/c/lab2/roteiro-11$
```