# Lab. 7 Efficient Array Sorting

For the following exercises read, into arrays **V_x**, the data stored in files "**dadosX.txt**" available in the web site, as done in the previous class)

### 1. Adapt Merge Sort

Adapt the implementation of Merge Sort presented in the slides of class 7, by including an extra Boolean parameter **dir** specifying if the sorting is done in **increasing / decreasing** order (**dir = 1 or 2**, respectively). Also return the number **c** of comparisons between elements of the vectors. Use signature

```
function [S, c, x] = merge_sort(V, dir)
```

Check the correctness and efficiency of your implementation with vectors **V_x**.

### 2. Adapt Quick Sort

Adapt the implementation of Quick Sort presented in the slides of class 7, by including an extra Boolean parameter **dir** specifying if the sorting is done in **increasing / decreasing** order (**dir = 1 or 2**, respectively). Also add to the results a) the number **a** of accesses to the elements of the vector that were considered, and b) the number **s** of swaps that were made in elements of the array. Use the signature (starting with **lo = 1** and **hi = n**)

```
function [S, a, s] = quick_sort(V, lo, hi, dir)
```

Check the correctness and efficiency of your implementation with vectors **V_x**.

### 3. Assess efficiency of Quick Sort and Merge Sort

Check the efficiency (and correctness) of your implementation of the previous functions if the input vectors are already sorted either in increasing or decreasing order.

### 4. Compare efficiency of Quick Sort and Merge Sort

Compare the results obtained in the previous item, with those obtained with Bubble Sort, as done in the previous lab class.