Lab. 6 Array Search and Sorting; Input/Output to Text Files

> 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 Bubble Sort

Adapt the implementation of Bubble Sort presented in the slides of class 6, by adding to the results a) the number of bubbles that were considered, and b) the number of bubbles that were swapped. Use the signature

```
function [S,bb,sw] = bubble_sort(V)
```

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

2. Optimise Bubble Sort

In the implementation of Bubble Sort presented in the slides of class 6, the procedure executes exactly **n-1** sweeps (outer loop – **for k = n:-1:2**), each over with decreasingly ranges of the vector (inner loop – **for i = 1:k-1**). However, if the vector is already sorted, or if a prefix of the vector is already sorted, sweeping the bubble does not change the vector any longer, and only wastes time.

Adapt the function presented in the slides of class 6, so that this inefficiency is eliminated, using signature

```
function [S,bb,sw] = bubble_sort_opt(V)
```

Compare the efficiency of this version wrt the previous one, the vectors provided in files **dadosX.txt**.

3. Searching Elements in an Array

Define the two functions below that return **p**, the index of an element of the array **V**) with value **v**. If no such element exists, return 0, and ifS more than one element exists, return the index of one of them. In addition, return also **k**, the number of elements analysed before finishing execution.

The two functions differ on the algorithm used: **search_lin** performs a linear search, whereas **search_bip** performs a bipartite search between indices **lo** and **up**.

```
function [p,k] = search_lin(v, V)
function [p,k] = search_bip(v, V, lo, up)
```

Compare the efficiency of the search of the two algorithms on vectors **V_x** (after their sorting).

> For the following exercises read, into a structure array **S**, the data stored in file "**students.txt**" available in the web site, as done in the previous class)

4. Sort a Structure Array (numeric field)

Define a function with the signature below, that for the structure array given as input, S, returns G, the structure array sorted by the student grades.

```
function G = sort_grades(S)
```

5. Compare two strings

Define the function below that compares two strings str1 and str2

```
function b = before_strings(str1, str2)
```

and return one of the values 0, 1 or 2 depending when whether the two strings are equal (0), str1 is alphabetically before str2 (1), or str2 is alphabetically before str1 (2).

6. Sort a Structure Array (text field)

Define a function with the signature below that, for the structure array given as input, S, returns G, the structure array sorted by the student names.

```
function N = sort_names(S)
```