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

1. Read Vectors from files

Read a file with several numbers, one per line, into a list of numbers (a vector). Use the following signature

```
def read_vector_from_file (fname):
```

that should return a list of numbers.

Test your code with files "**dados_X.txt**", for different values of X, available in the web site, to yield the corresponding lists **L_X**.

2. Reverse Bubble Sort

Adapt the Bubble Sort function presented in the slides of class 6, with a function with signature

```
def bubble_sort (V, inc):
```

that returns the list sorted in increasing or decreasing order, depending on the value of the Boolean parameter inc.

Analyse the results obtained with the lists **L_x** obtained from the files, and name the results as **S_X** or **R_X**, depending on the sorting order.

3. Adapt Bubble Sort

Adapt the Bubble Sort function above in the slides of class 6, with a function with signature

```
def bubble_sort_info(V, inc):
```

that returns a triple (`S, nb, ns`), where

- `S` is the sorted list, and
- `nb` is the number of bubbles that were considered, and
- `ns` is the number of bubbles that were swapped.

Analyse the results obtained with the lists **L_x**.

4. 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 in range(n-1:1:-1)`:), each over with decreasingly ranges of the vector (inner loop – `for i in range(0,k):`). 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

```
def bubble_sort_info_opt(V, inc):
```

Compare the efficiency of this version wrt the previous one, on lists **L_X**.

5. Searching Elements in an Array

Adapt functions **find_seq_for(x, S)** , **find_seq_sorted(x, S)** and **find_between(x, S)**, to return a pair **(p, c)** where

- `p` is the first position in list V where x is found (p = -1, if x is not present in the list), and
- `c` is the number of comparisons between x and elements of the list.

Compare the efficiency of the search of the three algorithms on lists **S_x**, obtained from sorting the corresponding lists **L_X** (use a arbitrary numbers x, belonging, or not, to lists S_x).

6. Input a Table of Dictionaries

Remind exercise 1 of last lab, where you defined a function with signature

```
def read_table(fname):
```

that returns a table (a list of dictionaries), **T**, with the data stored in file with **fname**. The first line of the file indicates the title of the table, the second the names of the fields, separated by semi-colons ("*;*"), and the subsequent lines the table data (also separated by semi-colons ("*;*").

Read file **students.txt** available in the website, and create a table **T** (a list of dictionaries)

7. Sort a Table of Dictionaries by a 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 in increasing order.

```
function N = sort_table_names(T)
```

Suggestion: Adapt bubble sort for dictionaries, as done in the previous question, but now comparing fields of dictionaries (which are strings).

8. Sort a Table of Dictionaries by a Numeric Field

Define a function with the signature below, that for the list of dictionaries, **A**, given as input, returns **G**, list of dictionaries sorted by the student grades in decreasing order.

```
def sort_table_grades(A)
```

Suggestion: Adapt bubble sort for dictionaries, as before, but make sure that, for the sake of comparisons, you convert the grades "rep" and "freq" into virtual grades 0 and 5, respectively, so the grade "rep" is less than grade "freq" that is less than any approval grade.