Lab. 8x – Overview of Past Exercises

0. Read Vectors from files

Use the function previously defined to read files "**dados_X.txt**", for different values of X, available in the web site and obtain the corresponding lists lists **L_X**.

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

1. 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 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(L)`.

To check the efficiency of the sorting, the function returns a quadruple (**`S, nb, ns, tm`**), where

- **`S`** is the sorted list, and
- **`nc`** is the number of comparisons made, and
- **`ns`** is the number of swaps made while partitioning the list.
- **`tm`** is the process time (Note: use **`process_time()`** function from module **`time`**).

2. Adapt Quick Sort

Adapt the implementation of Quick Sort presented in the slides of class 7, with a function with signature

`def quick_sort_info(L)`.

The function returns a quadruple (**`S, nb, ns, tm`**), where

- **`S`** is the sorted list, and
- **`nc`** is the number of comparisons made, and
- **`ns`** is the number of swaps made while partitioning the list.
- **`tm`** is the process time (Note: use **`process_time()`** function from module **`time`**)

Check the correctness of your implementation with lists **L_x.copy()**.

3. Searching Elements in an Array

Adapt functions **find_unsorted(x, L)** , **find_sorted(x, L)**, that return a pair **(p, c)** where

- **`p`** is the first position in list L 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.

The first function assumes the list L is not sorted, whereas the second assumes L to be sorted and adopts a divide-and conquer algorithm.

4. Monte-Carlo Simulation of a Simple Queue System

Assume a server whose service time follows an Erlang distribution (2,3) with customers arriving according to an exponential distribution of mean 5. Compare the number of accepted and rejected customers in case they may queue or not waiting for the server to be available. The queue gas q positions for customers to stay on waiting. Use and adapt (for the case with a queue), the state transition table below.

State Variables	**event (at time t)**	**variables update**			**next event (s)**	
		state variables	**monitoring variables**			
busy	**type**	**busy**	**n_accepted**	**n_rejected**	**type**	**time**
FALSE	arrival	-> TRUE	+1	=	arrival leaving	t + exp(l) t + Er(n,l)
TRUE	arrival	=	=	+1	arrival	t + exp(l)
TRUE	leaving	-> FALSE	=	=	-	-