Lab. 6 Efficient Array Sorting

1. Read Lists from files

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

```
def read_list_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. Adapt Merge Sort

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

```
def merge_sort_info(L, inc):
```

that includes an extra **True / False** Boolean parameter **inc**, specifying whether the sorting of list V is done in **increasing / decreasing** order, respectively.

The function returns a triple (`S, nc, tm`), where

- `S` is the sorted list, and
- `nc` is the number of comparisons made (e.g. when merging two sorted lists)
- `tm` is the process time (Note: use `process_time()` function from module `time`)

Check the correctness of your implementation with lists **L_x**.

3. 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, inc):
```

that includes an extra **True / False** Boolean parameter **inc**, specifying whether the sorting of list V is done in **increasing / decreasing** order, respectively.

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()**.

4. Assess efficiency of Quick Sort and Merge Sort

Check the efficiency (and correctness) of your implementation of the previous functions

a. For large lists (**L_1000**, **L_10000** and **L_100000**);
b. When the input lists are already sorted either in increasing or decreasing order.

5. Graphics

Specify a function with signature

```
def draw_complexity(n, xlin, ylin):
```

to draw a graph allowing the comparison of complexities O(n), O(log(n), o(n^2) and O(n log(n)), where

- `n` defines the range of `x` values (note: from 1 to n); and
- `xlin, ylin` are Booleans that specify whether the axes use a linear or logarithmic scale.

Hint: Use commands `xscale('linear')/yscale('linear')` or `yxscale('log')/yscale('log')` from library `mathplotlib.pyplot` to specify the type of scales to be used in the `x` and `y` axes.