Project instructions

Implement, test and compare array sorting methods.

Group I methods:

- Insertion sort
- Selection sort
- Bubble sort

Group II methods:

- Quicksort
- Shellsort
- Heapsort

Arrays with values of type int in the range <-100; 100> are the subject of sorting. Perform three types of sorting tests in ascending order for each method, depending on the preparation of the input data.

- Test1 - For randomly generated data
- Test2 - For data sorted in reverse order (descending)
- Test3 - For data sorted properly (ascending)

We load the sorted data from a file named **data** (we obtain it with our own sorting program, which writes the results to it).

Analysis of the results for each test (Test1 - Test3) should include graphs of the dependence of sorting time on array size

- One summary chart for the 3 primary methods,
- One summary chart for the second group of methods,
- One chart comparing all Group I and Group II methods

That sums up to a total of 9 charts - 3 for each data set.

To study the times, you can use the system function *time* or *gettimeofday.* We call it in the program just before and just after sorting and take into account the difference of the obtained times.

We choose the maximum size of the array and the length of the list so that the sorting time does not exceed 15 minutes. **The program asks the user for the size of the array to be sorted**.

An electronic report (pdf file) should present the algorithms (pseudo-codes or block diagrams of the methods) and their computational complexity compared with the results obtained (an attempt to approximate the obtained diagrams with appropriate complexity functions).

The program should include sorting functions and a minimal main function in which the user enters the size of the array and the program reads the data from the file ‘data’ and calls all the sorting functions. Please organize the way in which all the tests are to be carried out as you wish (preparing the data file, collecting the times, processing the results).

The organization of the project implementation is up to you.

It is best to prepare a large file with randomly generated numbers and before each sorting (except for tests for sorted data) read from it the appropriate number of data to be sorted (this will vary to get points on the graph for values on the ox axis from 0 to about a million).

The way the data is written to the file can be formatted or binary -- that's up to you, but the program should be able to read from a ‘data’ file in which consecutive numbers are separated by spaces or a transition to a new line -- reading format %d.

Program functionality:

1. read n (array size)
2. declaration of an n-element array
3. read by format (%d) n numbers from a file named ‘data’
4. call sorting functions
5. output a sorted array

Points 3 - 5 must be implementable for each algorithm -- it can be a menu of choices with repeatable calculations for all sorting functions or rigidly 6 calls.