Assignment No. 8: Iterative vs recursive tree traversal. Quicksort hybridization. Analysis & Comparison of running time

Allocated time: 2 hours

Implementation

You are required to implement **correctly** and **efficiently** *iterative* and *recursive* binary tree traversal, as well as a hibrid *Quicksort.*

You may find any necessary information and pseudo-code in your course and seminar notes

- *Recursive and iterative binary tree traversal*
- *Hybridization for quicksort using iterative insertion sort - in quicksort, for array sizes < threshold, insertion sort should be used (use quicksort implementation from assignment 3 and insertion sort from first assignment)*

Thresholds

Threshold	Requirements
5	Implementation of iterative and recursive binary tree traversal + demo
7	Comparative analysis of the recursive vs iterative tree traversal from the operations perspective
9	Quicksort hybridization. Demo and comparative analysis from the operations and runtime perspective.
10	Determination of an optimal threshold used in hybridisation + proof (graphics/ measurements)

Evaluation

! Before you start to work on the algorithms evaluation code, make sure you have a correct algorithm! You will have to prove your algorithm(s) work on a small-sized input.

1. In the comparative analysis of the iterative vs recursive version, you have to count only the print key operations, varying the no of nodes from the tree between between [100…10000], with an increment of maximum 500 (we suggest 100).

   For binary tree construction you can start from an array with a variable size and pick a random node as root.

2. For quicksort hybridization you have to use the iterative insertion sort from the first assignment if the size of the vector is small (we suggest using insertion sort if the vector has less than 30 elements). Compare runtime and the number of operations (assignments + comparisons) for quicksort implemented in the third assignment with the hybrid one.

   You should vary the size of the array for which insertion sort is applied and compare the results from the performance perspective for determination of the optimum threshold. You can use 10.000 as the size of the vector that is being sorted.

   For measuring the runtime you can use Profiler similar to the example below.

   ```
   profiler.startTimer("your_function", current_size);
   for(int test=0; test<nr_tests; ++test) {
      your_function(array, current_size);
   }
   profiler.stopTimer("your_function", current_size);
   ```

   The number of tests (*nr_tests* from the example) has to be chosen based on your processor and the compile mode used. We suggest bigger values such as 100 or 1000.

   When you are measuring the execution time make sure all the processes that are not critical are stopped.

3. Prepare a demo for each algorithm implemented.

4. We do not accept assignments without code indentation and with code not organized in functions (for example where the entire code is in the main function).

5. ***The points from the requirements correspond to a correct and complete solution, quality of interpretation from the block comment and the correct answer to the questions from the teacher.***