Assignment No. 8: Iterative vs recursive binary 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)*

Requirements

1. **Implementation of iterative and recursive binary tree traversal in *O(n)* and with *constant additional memory* (5p)**

   You will have to prove your algorithm(s) work on a small-sized input.

2. **Comparative analysis of the *recursive vs iterative* tree traversal from the perspective of the number of operations (2p)**

! Before you start to work on the algorithms evaluation code, make sure you have a correct algorithm!

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 in the [100, 10000] range 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.

3. **Implementation of quicksort hybridization (1p)**

You will have to prove your algorithm(s) work on a small-sized input.

4. **Comparative analysis (between *quicksort and quicksort hibridization*) from the operations and runtime perspective (1p)**

! Before you start to work on the algorithms evaluation code, make sure you have a correct algorithm!

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.

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);
```

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

5. **Determination of an optimal threshold used in hybridization + proof (*graphics/ measurements*) (1p)**

You should vary the threshold value of quicksort hibridization for which insertion sort is applied.

Compare the results from the performance (*number of operations and execution time*) perspective for determination of the optimum threshold. You can use 10.000 as the fixed size of the vector that is being sorted and vary the threshold between [5,50] with an increment of 1 to 5.

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.