Assignment No. 3: Analysis & Comparison of Advanced Sorting Methods – Heapsort and Quicksort / QuickSelect

Allocated time: 2 hours

Implementation

You are required to implement **correctly** and **efficiently** *Quicksort and Quick-Select (Randomized-Select)*. You are also required to analyze comparatively of the complexity of *Heapsort* (implemented in Assignment No. 2) and *Quicksort*.

You may find any necessary information and pseudo-code in your course notes, or in the book[1]:

- *Heapsort*: chapter 6 (Heapsort)
- *Quicksort*: chapter 7 (Quicksort)
- *Randomized-Select*: chapter 9

Minimal requirements for grading

- Prepare a demo for each algorithm implemented.
- Interpret the charts and write your observations in the header (block comments) section at the beginning of your *main .cpp* file.
- 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).
- ***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.***

Requirements

1. **QuickSort: implementation (3p)**

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

2. **QuickSort: average case analysis (2p)**

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

You are required to compare the two sorting procedures in the **average** case. Remember that for the **average** case you have to repeat the measurements *m* times (m=5) and report their average; also for the **average** case, make sure you always use the **same** input sequence for the two methods – to make the comparison fair.

This is how the analysis should be performed:
- vary the dimension of the input array (*n*) between [100…10000], with an increment of maximum 500 (we suggest 100).
- for each dimension, generate the appropriate input sequence for the method; run the method, counting the operations (assignments and comparisons, may be counted together).

! Only the assignments and comparisons performed on the input structure and its corresponding auxiliary variables matter.

Generate a chart which compares the two methods under the total number of operations, in the **average** case.

If one of the curves cannot be visualized correctly because the other has a larger growth rate, place that curve on a separate chart as well. Name the chart and curves appropriately.

3. **QuickSort: best and worst case analysis (1p)**

4. **Quicksort and Heapsort: comparative analysis of average case and interpretation (2p)**

5. **Comparative analysis of *one* of the sorting algorithms from L1 (you choose) in iterative vs recursive version. The analysis should be performed based on the *number of operations and the runtime* (2p)**

For the comparative analysis of the iterative vs recursive version pick one of the 3 algorithms from Assignment 1 (bubble sort, insertion or selection). Use the iterative version that you already implemented (corrected, if needed, based on the feedback received from the teacher) and implement the same algorithm in the recursive version.

You must measure the total effort and the running time of the two versions (iterative and recursive) => two charts, each of them comparing the two versions of the algorithm.

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 above example) has to be chosen based on your processor and the compile mode used. We suggest bigger values such as 100 or 1000.

6. **Bonus: QuickSelect - Randomized-Select (0.5p)**

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

For QuickSelect (Randomized-Select) no explicit complexity analysis needs to be performed, only the correctness needs to be demonstrated on sample inputs.