Assessed Exercise 1

This lab sheet contains material based on Lecture 7. This exercise is for submission using Moodle and counts for 10% of the total assessment mark for this course.

This exercise is worth a total of 30 points.

The deadline for submission is Friday 11 February 2022 at 4:30pm.

Exercise

This exercise has three parts. The first involves implementing four variants of the quicksort algorithm in Java, the second involves running some empirical studies to compare the performance of each version, and the third involves the definition and implementation in Java of an algorithm to compute adversaries for quicksort.

Submission

Submit the Java sources of your implementations and a short (maximum 3 pages) report briefly describing what you have done in each part of the exercise. Your report should include a heading stating your full name and matriculation number and clear instructions on how to run your code.

Part 1

Implement the following algorithms in Java:

a) The pseudocode for QUICKSORT introduced in Lecture 7 (slide 14), including procedure PARTITION implementing right-most element pivot selection (slide 13). [6]

b) A variant of QUICKSORT which returns without sorting subarrays with fewer than k elements and then uses INSERTION-SORT to sort the entire nearly-sorted array (Lecture 7 slide 24). [2]

c) A variant of QUICKSORT using the median-of-three partitioning scheme. [2]

d) 3-WAY-QUICKSORT (Lecture 7, slide 25). Explain in the report how your implementation operates. [10]

Part 2

Compare all four versions of the algorithm you implemented in Part 1 (try experimenting with different cutoff values in 1b) using `TimeSortingAlgorithms` you implemented in Lab 1. Use datasets provided on Moodle under Lab/Files. Also compare with the running times of `InsertionSort` and `MergeSort`. Report and explain your findings. Hint: use `TestSortingAlgorithms` from Lab 1 to test the correctness of your algorithms. [4]

Part 3

Define an algorithm to generate pathological input sequences for the quicksort algorithm, that is instances that will require quadratic time to be sorted. Consider a median-of-three partitioning scheme (left, middle, right). Also, we assume duplicates are **not** allowed in the input array. Explain in the report how your algorithm operates and implement it in Java.

[6]