Compulsory assignment 1 - INF 102 - Autumn 2016

Deadline: **30 September 2016, 10:00**

Organizational notes

This compulsory assignment is an individual task, however you are allowed to work together with at most 1 other student. If you do, remember to write down both your name and the name of team member on everything you submit (code + answer text). In addition to your own code, you may use the entire Java standard library, the booksite and the code provided in the github repository associated with this course.

The assignments are pass or fail. If you have made a serious attempt but the result is still not sufficient, you get feedback and a new (short, final) deadline. You need to pass all (3) assignments to be admitted to the final exam.

Your solutions (including all source code and textual solutions in PDF format) must be submitted to the automatic submission system accessible through the course before Sep 30th 2016, 10am. Instructions on how to submit will follow shortly. Independent of using the submission system, you should always keep a backup of your solutions for safety and later reference.

If you have any questions related to the exercises, send an email to:
`gunnar.schulze@ii.uib.no`
In addition you have the opportunity to ask some questions in the Friday review session and at the workshops.

1 Simple calculator

Write a program that can evaluate simple arithmetic expressions of the following form:

```
"1+2=*3+4==" (=21)
```
(1)

In this notation, the = operator indicates that the previous operation should be evaluated before proceeding. Thus, in the above example, the total is calculated by first adding 1 and 2 (3), then adding 3 and 4 (7) and finally multiplying the sums to yield 21. You may assume that the only operators used are: +, $*$ and = and that the input is given as a single string of the form just described.

2 Triplicates in four lists

Given four lists of N names, devise a linearithmic (O(N*log(N))) algorithm to determine if there is any name common to *at least* three of the four lists, and if so, return the lexicographically first such name. Implement your algorithm in Java and do a (theoretical) runtime analysis that explains how you arrive at the linearithmic order of growth.

3 Quicksort, empirically

Perform doubling experiments on sorting an integer array using Quicksort and a modified Quicksort version, where the initial shuffling of the input array is removed. E.g. choose N, the input array size as 5K, 10K, 20K, 40K, 80K, 160K, 320K and measure the time taken for both programs to complete. (Hint: Use *Stopwatch* provided in the *algs4.jar* archive).

Make a plot comparing the runtime (Y-axis) against the input size (X-axis) based on these experiments. Compute the logs of the input sizes and of the run times and make a log-log plot. Estimate the slope (a) and intercept for the log-log plot such that the runtime (Y) can be described as $Y \approx aX - b$.

Perform `LinearRegression` (provided in algs4.jar) to estimate a and b.

Find a function $f(N)$ such that $log(f(N)) = a * log(N) - b$. What do the constants a and b and variable N mean in this context?

Compare the two versions of Quicksort. Does the shuffling procedure influence the runtime? If so, how?

4 Countsort

Implement `Countsort` of integers 0..99 (use an array of length 100). Design a client program that runs your implementation of `Countsort` on a large 1M integer array with values ranging from 0-99.

Use your plotting procedure, developed in exercise 3 to describe the runtime of this algorithm for different values of N (length of input array) while keeping the number of possible values constant (0-99). You can for example perform doubling experiments as in exercise 3.

Compare this runtime to a linearithmic profile. Which Big-Oh runtime does `Countsort` assume when $N >> 100$?

5 Exam exercise

Design (and solve) an interesting exercise that you think could be part of the final exam. Submit both the exercise text and the solution (possibly a small program). The task should be solvable given the lectures/book content covered in the course so far (Chapters 1.3-2.5 from the textbook).