max planck institut
informatik

SAARLAND
UNIVERSITY
COMPUTER SCIENCE

Karl Bringmann, Erik Jan van Leeuwen **Winter 2016/2017**

Exercises for Algorithms and Data Structures

http://www.mpi-inf.mpg.de/departments/algorithms-complexity/teaching/winter16/algorithms-and-data-structures/

Exercise Sheet 2 Due: **7.11.2016**

The homework must be handed in on Monday before the lecture. You may collaborate with other students on finding the solutions for this problem set, but every student must hand in a writeup in their own words. We also expect you to state your collaborators and sources (books, papers, course notes, web pages, etc.) that you used to arrive at your solutions.

You need to collect at least 50% of all points on the first six exercise sheets, and at least 50% of all points on the remaining exercise sheets.

Whenever you are asked to design an algorithm in this exercise sheet you have to give a proof of its correctness as well as an asymptotic upper bound on its worst case running time.

Exercise 1 (*10 points*)

Consider the RAM model introduced in the first lecture of the course. One of our assumptions was that each memory cell can store up to $c \log n$ bits, where n is the input size and c is some positive constant. Let c' be a constant larger than c.

a) Show how to store numbers that have size $c' \log n$ bits.

b) Show how to do multiplication and addition of two integers of size $c' \log n$ bits in time $O(1)$.

Similar statements hold for the other RAM operations. This shows that the exact constant in the cell size does not matter much.

Exercise 2 (*10 points*)

Consider a data structure with n entries, for some integer n. We will think of each entry as being either 0 or 1. The data structure supports five methods: *initialize()*, *setToOne(i)*, *setToZero(i)*, *isSetToOne(i)*, and *isSetToZero(i)*, for $1 \le i \le n$. The method *initialize()* sets all entries to 0, the method *setToOne(i)* sets the i-th entry to 1, etc. A naive implementation of this data structure uses an array A with n entries, which uses $O(n)$ time for *initialize()* and $O(1)$ time for the other methods. Develop an implementation for the data structure and its methods that requires $O(1)$ time for all five methods.

Hint: For example, use two arrays that store pointers to each others cells, and a counter storing the current number of 1s.

Exercise 3 (*10 points*)

You are given arrays A of size n and B of size m as an input, where $m \leq n$. Both arrays store integers that are sorted by their value. In this exercise we want to compute the elements that appear in both arrays.

a) Give an algorithm with worst case running time $O(n + m)$.

b) Give an algorithm with worst case running time $O(m \log n)$. Are there inputs for which this algorithm runs faster than the algorithm from part a)?

c) Combine the algorithms from parts a) and b) to an algorithm with worst case running time $O(m \log \frac{n}{m})$. This is the fastest known algorithm for this problem.

Exercise 4 (*10 points*)

You are given an array A of n distinct entries. Show how to generate a random permutation of its entries in time $O(n)$. Show that the generated permutation is equally likely among all $n!$ possible permutations.