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 4 Due: **21.11.2016**

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*)

In class you have seen two theorems to bound the running time of divide and conquer algorithms, namely the Master Theorem and the Akra-Bazzi Theorem.

Theorem 1 (Master Theorem)**.** *Let $a \geq 1, b > 1$ be constants, let $g(n)$ be a function and let $T(n)$ be defined on the non-negative integers by the recurrence*

$$T(n) = aT(\frac{n}{b}) + g(n),$$

where we interpret $\frac{n}{b}$ to either mean $\lfloor \frac{n}{b} \rfloor$ or $\lceil \frac{n}{b} \rceil$. Let $p := log_b a$, then $T(n)$ can be bounded asymptotically as follows.

1. *If $g(n) = O(n^{p-\epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^p)$.*
2. *If $g(n) = \Theta(n^p)$, then $T(n) = \Theta(n^p \log n)$.*
3. *If $g(n) = \Omega(n^{p+\epsilon})$ for some constant $\epsilon > 0$, and if $ag(\frac{n}{b}) \leq cg(n)$ for some constant $c < 1$ and all sufficently large n, then $T(n) = \Theta(g(n))$.*

Show that the Akra-Bazzi Theorem is more general by using it to derive the Master Theorem.

Exercise 2 (*10 points*)

Given a sequence $S = (x_1, \ldots, x_n)$ of n keys and a positive integer k, such that $1 < k < n$, we say that S has a k-fraction, if there is a key that appears more than $\frac{n}{k}$ times in S.

a) Give an $O(n \log n)$ time algorithm that decides whether S has a k-fraction.

b) Give an $O(n \log k)$ time algorithm for this problem.

   Hint: One can compute the median element of S in linear time.

Exercise 3 (*10 points*)

Recall that Interval Scheduling is the following problem. The input is a set of n requests; request i is given by its start time $s(i)$ and finish time $f(i)$. Two distinct requests i, j are compatible if $f(j) \leq s(i)$ or $f(i) \leq s(j)$. The problem is to find a largest set of compatible requests.

One possible greedy algorithm for interval scheduling always selects the shortest interval that is compatible with each request in the set of currently selected intervals. In class, we showed that this strategy does not always produce an optimal solution. In fact, the example exhibited that an optimal solution selects 2 intervals, but this greedy algorithm selects only 1, a multiplicative factor 2 difference. Prove that the shortest-interval greedy algorithm always returns a solution to the interval scheduling problem that schedules at least half the number of request that an optimal solution would.

Exercise 4 (*10 points*)

A palindromic subsequence of a string is a subsequence that reads the same forwards and backwards. Design an $O(n^2)$ time algorithm that computes the longest palindromic subsequence of a given string of length n, e.g., using dynamic programming.