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 3 Due: **14.11.2016**

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*)

Let $X = \{x_1 < x_2 < \ldots < x_n\}$ be n keys and consider a randomized search tree T for X.

a) Consider the node for x_k in T for some k with $1 \leq k \leq n$. What is the expected size of the subtree rooted at that node?

b) What is the probability that the number of ancestors in T of the node for x_1 is n?

c) What is the probability that the number of descendants in T of the node for x_1 is n?

Exercise 2 (*10 points*)

When using randomized search trees it is assumed that the user cannot observe the random priorities that are used in the treap. Show that if the random priorities in a randomized search tree were observable, then a malicious user could make randomized search trees perform badly: For instance, he could generate a sequence of $\Theta(n)$ insertions, deletions, and lookups, starting with an initially empty treap that takes $\Omega(n\sqrt{n})$ time instead of the expected $O(n \log n)$.

Exercise 3 (*10 points*)

In the lecture you saw how to use the function *rand(N)*, that returns a random integer between 1 and N, to choose a pivot element during the execution of *Quicksort*. In this exercise we

want to investigate how we can implement *rand(N)*, if we can call *rand()* only with restricted parameters larger or smaller than N.

a) Show how to simulate a 12 sided dice using a 20 sided dice with only a constant number of dice rolls in expectation.

b) Let M be a natural number much larger than 12. Show how to simulate a 12 sided dice using a M sided dice with only a constant number of dice rolls in expectation.

c) Let N_1 and N_2 be two positive integers and $N_1 \leq N_2$. Show how to implement $rand(N_1)$ using $rand(N_2)$ in expected constant time.

c) Let $\sqrt{N_1} \leq N_2 \leq N_1$. Show how to implement $rand(N_1)$ using $rand(N_2)$ in expected constant time.

Exercise 4 (*10 points*)

The university will open a new hallway with n desks for students to study at. Each desk $i = 1, \ldots, n$ has a laptop stand, situated at distance d_i from the start of the hallway (You can think of the hallway as a line segment). The university intends to provide wifi-service in the hallway by hanging access points in the hallway. Each access point provides wireless service to any laptop stand within distance R of the spot where that access point is placed.

Give a greedy algorithm that determines the smallest number of access points needed to provide wireless service to each laptop stand. Your algorithm should also return the locations where these access points must be hung. You may assume that all numbers are integers.