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 12 Due: **6.2.2017**

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 all exercise sheets to be admitted to the final exam.

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 this exercise, we will study the behavior of hollow heaps, and in particular, the consequences of a slight modification to the `DecreaseKey` operation. Throughout, we assume that n is a large number that is a power of 2.

a) (2 points) Consider a sequence of $n + 1$ insertions followed by one deletion. Describe the structure of the resulting hollow heap and, in particular, the ranks.

In hollow heaps, we essentially do the following to decrease the key of an item e, stored in node u, to a new value k: Create a new node v and move e to v (leaving u hollow). Set the key of v to k, move all but two of the children of u (and their subtrees) to v, and set the rank of v to $\text{rank}(u) - 2$. (Here, we ignore the boundary case that $\text{rank}(u) < 2$.

Consider the following modification of the `DecreaseKey` operation: we do not move any children of u to v and set the rank of v to 0. The rest remains the same. This essentially means that we do a `Delete` followed by an `Insert`.

b) (3 points) Consider the hollow heap after $n + 1$ insertions followed by one deletion, as in a). On each child of the root pointed to by the `MinPointer`, we perform a modified *DecreaseKey* operation. Afterwards, we perform a `Delete` operation on this root. What is the worst-case running time of this sequence?

c) (5 points) Keeping the modified `DecreaseKey` operation in mind, expand the ideas of b) to construct, for *any* (large enough) number m, a sequence of m operations that takes $\Omega(m \log n)$ time. What does this say about the amortized cost of `Insert`, `Delete`, and the modified `DecreaseKey` operations?

Exercise 2 (*10 points*) Splay trees are dynamic binary search trees that use tree rotations (just as in treaps) to keep the tree balanced. We always rotate along the parent of the current node, except in the special case when the current node is a left child of a left child or a right child of a right child, in which case we rotate along the grandparent of the current node first. We call this procedure *splaying.*

The `Insert` operation will insert a new node at the correct place in the tree and then *splay* the new node upwards until it becomes the root of the tree. The `Find` operation will look for the item as usual in binary search trees, and *splay* the node where the search ended (regardless of whether it contains the requested key) upwards until it becomes the root of the tree.

Now suppose that we did not use the special case, and instead of splaying in the `Insert` and `Find` operations, performed the following procedure:

```
1: procedure MoveToRoot(v)
2:     while parent($v$) != Null do
3:         rotate along parent($v$)
```

Prove that for *any* (large enough) m, there is a sequence of m `Insert` and `Find` operations that require $\Omega(mn)$ time to execute. What does this say about the amortized cost of `Insert` and `Find` operations?

Exercise 3 (*10 points*)

A *red-black tree* is a binary search tree with the following properties:

- Every node is colored either red or black.
- The root is black.
- Every leaf is colored black.
- If a node is red, then both its children are black.
- The number of black nodes encountered on the (unique) path from a leaf to the root is the same for all leafs.

Give a lower- and upper-bound on the height of a red-black tree with n items.

Exercise 4 (*10 points*)

In the lecture we covered Cuckoo Hashing. Recall that here we use two tables T_1 and T_2 of the same size and hash functions h_1 and h_2. When inserting a new key x, we first try to put x at position $h_1(x)$ in T_1. If this leads to a collision, then the previously stored key y is moved to position $h_2(y)$ in T_2. If this leads to another collision, then the next key is again inserted at the appropriate position in T_1, and so on. In some cases, this procedure continues forever, i.e. the same configuration appears after some steps of moving the keys around to dissolve collisions.

a) (5 points) Consider two tables of size 5 each and two hash functions $h_1(k) = k \mod 5$ and $h_2(k) = \lfloor \frac{k}{5} \rfloor \mod 5$. Insert the keys $27, 2, 32$ in this order into initially empty hash tables, and show the result.

b) (5 points) Find another key such that its insertion leads to an infinite sequence of key displacements.