Data Structures - Assignment no. 5, April 11, 2007

Remarks:

- **Write both your name and your ID number very clearly on the top of the exercise. Write your exercises in pen, or in clearly visible pencil. Please write *very* clearly.**
- **Recall that 80% of the theoretical exercises must be submitted. The exercises can and must be worked on and submitted alone.**
- **Give correctness and complexity proofs for every algorithm you write.**
- **For every question where you are required to write pseudo-code, also explain your solution in words.**

1. Insert the keys 5, then 9 and then 2 to the heap depicted in Figure 1. Then perform delete-min four times. Now draw the resulting heap.

2. Show how to modify a 2-4+ search tree, in order to get a data structure that supports the operations *delete-min* and *insert* in $O(\log n)$ time, and *find-min* in $O(1)$ time. What are the disadvantages of such an implementation of a heap, compared to the standard implementation?

3. Describe an algorithm that is given two heaps, both of size n, and returns a heap that contains all elements of both heaps. (Assume that no key appears more than once in the input). Try to make the algorithm as asymptotically efficient as possible. (Hint: the solution is very easy, and can be described in one or two lines).

4. Describe an algorithm that prints the k smallest elements in a Heap. You can assume that the heap is represented as an array or as a tree, whichever is more comfortable for you. As usual, you may also assume that no key appears more than once in the heap. The algorithm should take $O(k \log k)$ time. The algortihm should not modify the heap. Give: (i) pseudo-code; (ii) an explanation of the algorithm; (iii) an explanation why it is correct; and (iv) an explanation why the running time is indeed $O(k \log k)$.
   Note: Observe that getting an algorithm that runs in time $O(k \log n)$, where n is the size of the heap, is easy – just perform k delete-mins. (In order to avoid modifying the heap, you need to undo your actions, which takes another $O(k \log n)$ time).

5. Describe an algorithm that solves the following problem. You are given k sorted lists $A_1, \ldots, A_k$, each of length n. The output should be one sorted list which contains the keys of all input lists. The algorithm should take $O(nk \log k)$ time. You may assume that no key appears more than once in the input. Give: (i) an explanation of the algorithm; (ii) an explanation why it is correct; and (iii) an explanation why the running time is indeed $O(nk \log k)$.
   Hint: The merge procedure that is used as a subroutine in *merge-sort* (which you learned in the course "extended introduction to CS") answers this question for $k = 2$ in time $O(n)$.

6. Consider an implementation of Fibonacci heaps without cascading cuts (all other details are as shown in class, the only difference is that delete and decrease-key just cut the subtree and do not continue with cascading cuts). For any large enough m show a sequence of m operations on heaps of size at most n such that the average cost of an operation is as high as possible. (By m large enough we mean larger even than some function of n.)

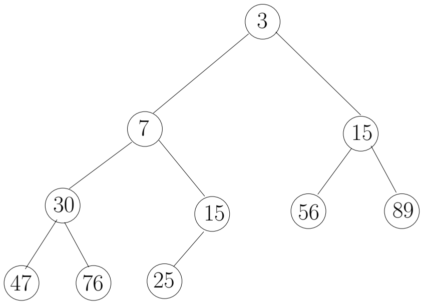

Figure 1: A Heap.