TEL AVIV UNIVERSITY
Department of Computer Science
0368.4281 – Advanced topics in data structures
Spring Semester, 2000/2001

Homework 2, May 2, 2001

Due on May 16 in my mailbox (since our semester is going to be over soon without we even noticed it began I would like this date to be firm this time).

Heaps

Question no. 3 from homework 1 wasn't phrased very well and came out confusing. Here is a second attempt. You are welcome to think about it again and submit a new answer. I am sorry about that.

1. 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. For example, average cost of $\Omega(\sqrt{n})$ per operation on long enough sequences is quite easy to get.)

Trees

2. We perform a sequence of m operations on an heterogenous 2-4 finger search tree (i.e. a finger search tree without level links). Out of these m operations k are concatenations where the ith concatenation concatenates two trees such that the smaller among them contains n_i elements. The other $m-k$ operations are find, insert, delete, and split. The j-th operation among these operations is performed on an element at distance d_j from the beginning of a list containing n_j elements. Prove that using the implementations of the operations as described in class one can perform the sequence in

$$O(\sum_{i=1}^{k} \log(n_i) + \sum_{j=1}^{m-k} \log(\min\{d_j, n_j - d_j\}))$$

time.

3. Improve the bound in the previous question to

$$O(k + \sum_{j=1}^{m-k} \log(\min\{d_j, n_j - d_j\}))$$

time.

4. Describe an algorithm that merges two sorted lists of lengths m and n where $m \leq n$. We represent the lists by finger 2-4 trees. Your algorithm should run in $O(m \log(\frac{n}{m}))$ time. The algorithm should produce a finger 2-4 tree representing the list resulting from the merge.

5. In this question you'd fill in some details skipped in class regarding the Jordan sorting algorithm.

a) Prove that $T(p, s)$ define by $T(p, s) = 0$ if $p = 0$ and otherwise

$$T(p, s) = \max_{0 \leq i < p, 0 \leq d \leq s} \{T(i, d+1) + T(p-i-1, s-d+1) + O(1 + \log(\min\{d, s-d\} + 1))\}$$

is $O(p + s)$.

b) What would be the running time of the Jordan sorting algorithm if we represent each list of siblings in the upper and lower trees simply as a circular doubly linked list without a finger search tree on top. Prove your answer.