Advanced Data Structures (Jan-Apr 2012)

Problem Sheet 1 *Due Date: Jan 23, 2012*

1. Recall that a *treap* is a data structure which is a binary tree that stores in every internal node an ordered pair (k, p) ((key, priority) pair) such that the (key) values of the first coordinate satisfy the 'binary search tree' property and the (priority) values of the second coordinate satisfy the 'min heap' property (i.e. the priority of the parent $\leq$ the priority of the children).

   Consider the following algorithm to build a treap given n (k, p) pairs where all the key values and the priority values are distinct.

   Sort the given pairs in increasing order of the first coordinates (keys). Starting from an empty treap, insert the next item (from this sorted order) with pair (k, p) as follows. First insert it as a right child of the last node (leaf) of the right most path. Note that the binary search property on the keys is already satisfied. Then, as long as the priority value of its parent is larger than its priority value, left rotate at its parent. This process stops if the priority value of the parent is smaller than the given item's priority value or the given item reaches the root.

   Clearly the algorithm performs $O(n \log n)$ key comparisons. Show that the algorithm performs $O(n)$ priority comparisons.

   (Hint: Try and show that the amorized number of priority comparisons in each insert is $O(1)$ using a potential argument.)

2. Consider the analysis of the move to front heuristic we discussed in class. Obtain the competitive ratio under the cost model where every exchange (with an adjacent element) costs 1. I.e. in the model, an algorithm for access (or insert and delete) can only swap an element with an adjacent element, though it can perform any number of them, each at a cost of 1. i.e. there are no free exchanges.