ESO207A/211: Data Structures and Algorithms

Homework *4* **Due: Oct 17 2014**

Instructions.

1. Start each problem on a new sheet. Write your name, Roll No., the course number, the problem number, the date and the names of any students with whom you collaborated. Note that you may collaborate with other students but you have to write your own answers.
2. For questions in which algorithms are asked for, your answer should take the form of a short write-up. The first paragraph should summarize the problem you are solving and what your results are (time/space complexity as appropriate). The body of the essay should provide the following:
   (a) A description of the algorithm in English and/or pseudo-code, where, helpful.
   (b) At least one worked example or diagram to show more precisely how your algorithm works.
   (c) A proof/argument of the correctness of the algorithm.
   (d) An analysis of the running time of the algorithm.

   Remember, your goal is to communicate. *Full marks will be given only to correct solutions which are described clearly.* Convoluted and unclear descriptions will receive *low marks.*

Problem 1. *Binary search tree.* An alternative method of performing an inorder tree walk of an n-node binary search tree finds the minimum element in the tree by calling TREE-MINIMUM and then making $n-1$ calls to TREE-SUCCESSOR. Prove that this algorithm runs in time $\Theta(n)$.

Problem 2. *Rotations.*

1. Show that in every n-node binary search tree, there are exactly $n-1$ rotations (left + right).
2. Give an algorithm that transforms an arbitrary n-node binary search tree into a right-going chain using at most $O(n)$ rotations.

Problem 3. *Treaps.*

A treap is a binary search tree where each node x has two fields, $x.key$ and $x.priority$. The *priority* value of a node is often a random number. We assume that all priorities are distinct and all keys are also distinct. The nodes of a treap satisfy the binary-search-tree property for keys and the min-heap property for priorities. See Figure 1 for an example.

1. If v is in the left subtree of u then $v.key < u.\ key$.
2. If v is the right subtree of u then $v.key > u.key$.
3. If v is a child of u, then $v.priority > u.priority$.

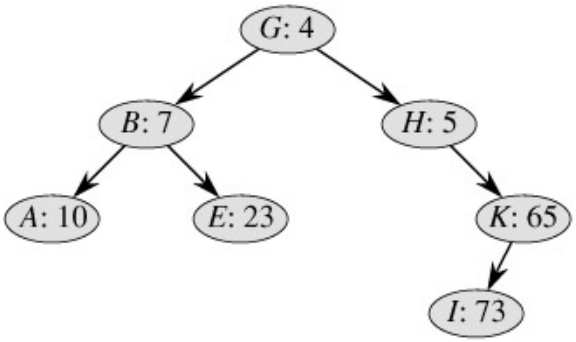

Figure 1: A treap. Each node x is labeled with $x.key : x.priority$. For example, the root has key G and priority 4.

It helps to think of treaps in the following way. Suppose that we insert nodes $x_1, x_2, \ldots, x_n$, with associated keys into a treap. Then the resulting treap is the tree that would be formed if the nodes had been insserted into a normal binary search tree in the order given by their priorities, i.e., $x_i.priority < x_j.priority$ means that we had inserted x_i before x_j.

a Show that given a set of nodes $x_1, x_2, \ldots, x_n$, with associated keys and priorities, all distinct, the treap associated with these nodes is unique.

To insert a new node, we first assign to the node a new random priority. Then, we call the insertion algorithm, called Treap-Insert depicted in Figure 2.

b Explain how Treap-Insert works. Explain the idea in English and give pseudo-code. (*Hint*: Insert as in a normal binary-search-tree and then perform rotations to restore min-heap property.)

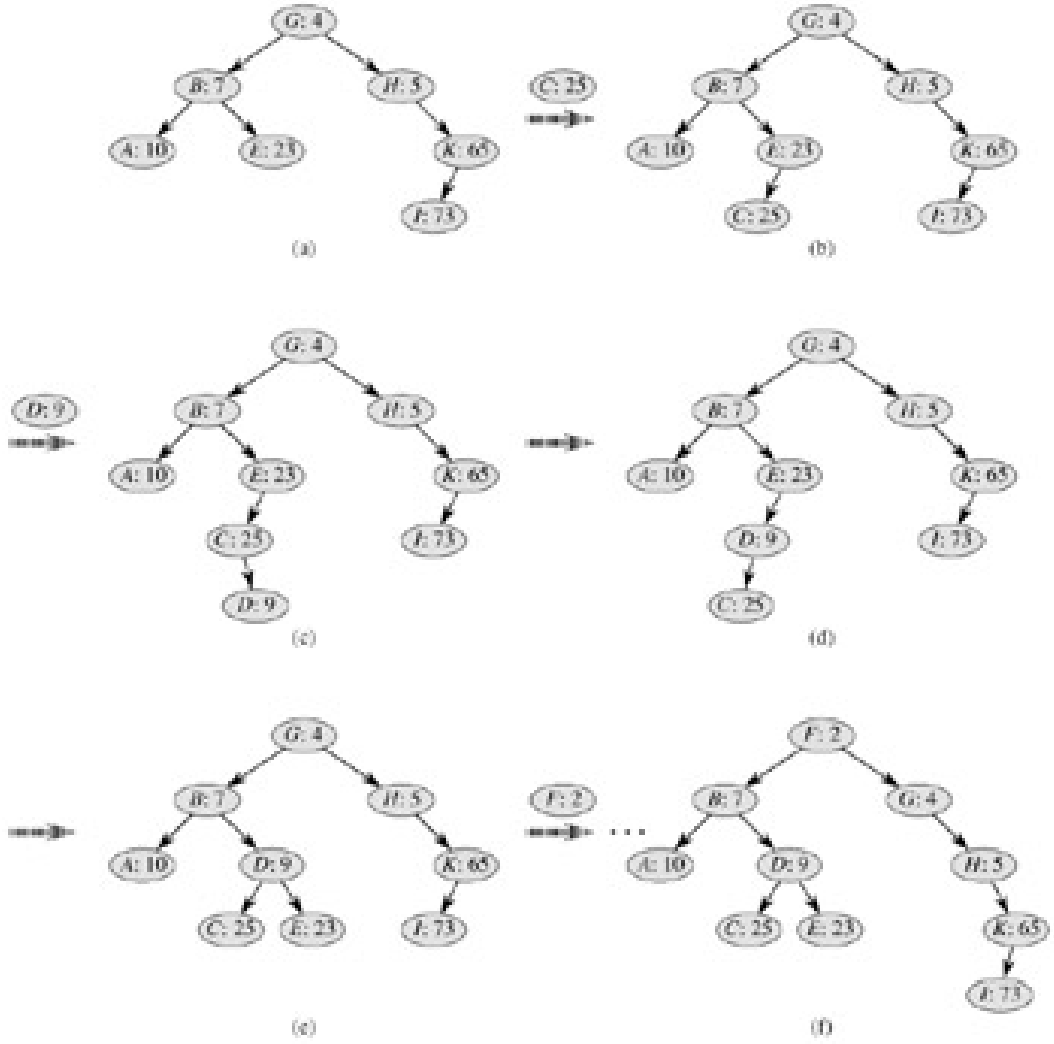

Figure 2: The operation of TREAP-INSERT. (a) the original treap, prior to insertion. (b) The treap after insrting a node with key C and priority 25. (c)-(d) Intermediate stages when inserting a node with key D and priority 9. (e) The treap after the insertions of parts (c) and (d) is done. (f) The treap after inserting a node with key F and priority 2.