

# CS381 Homework 4

Out: Oct. 26, Thursday

Due: Nov. 9, Thursday, at the beginning of (or before) class  
Submissions will not be accepted afterwards.

## Instructions:

**Read the course policy (including academic dishonesty) in the course webpage at <http://www.cs.purdue.edu/homes/gopal/cs381>.**

Text refers to the Introduction to Algorithms (second edition) book.

**Justify your answers. Show appropriate work.**

**Reading:** Chapters 12, 13, and 6 of Text.

## Problem 1

You are given an arbitrary binary search tree  $T$  with a certain number of keys. The procedure for deleting a key is as discussed in class: (1) if the node is a leaf then just drop it; (2) if the node has one child just splice it; (3) if the node has two children then replace the key with the leftmost node of its right subtree (i.e., its successor) and then splice out the successor (which will have at most one child).

You are given two keys  $x$  and  $y$  to delete from  $T$ . In general, does it matter which one you delete first? That is, will the resulting tree be always the same irrespective of whether you delete  $x$  first and then delete  $y$  or you delete  $y$  first and then delete  $x$ ? Justify your answer in either case. (Hint: If you think the resulting trees will always be the same, then give an argument to justify it. If you think they will not be the same, give an example tree  $T$  and deletions of nodes  $x$  and  $y$ .)

(10 points)

## Problem 2

Start with an empty red-black tree.

(a) Insert the first 10 numbers in the order  $1, 2, \dots, 10$ . Show the tree after adding every number. Note that you should show the colors of the nodes in the tree (red or black). You should also say how you obtain the tree after each addition (i.e., e.g., you can say things like "left rotate node  $x$  and then change color of node  $y$  etc".) (10 points)

(b) After you have inserted all the 10 numbers, delete 5 and then delete 6. Show the steps and the resulting tree after each deletion. (10 points)

### Problem 3

We are given a set of *distinct* keys such that each key  $k$  also has a priority value  $p(k)$ . Assume that the key values  $k$  (we abuse notation to denote both the key and its value by  $k$ ) and priority values  $p(k)$  are drawn from a totally ordered set. We would like to construct a *binary search tree*  $T$  on the set of keys such that the priority values satisfy the *heap property* i.e., a key of maximum priority in any subtree of  $T$  is found at the root of that subtree. Note that the key values are stored in an inorder fashion in  $T$ .

(1) Assume that all priority values are distinct. Show that there *exists* a *unique* binary search tree  $T$  on the keys such that the priorities satisfy the heap property. (Hint: One way to do is to build the tree starting from the root node and show that there is a unique choice in every step. Start with the root, argue why it should be unique.) (5 points)

(2) Describe how to insert an element in  $T$  (while still maintaining the binary search tree and priority properties). Note that the element that is inserted has a key value and a priority value (you can assume that these values are distinct from all the other elements.) (Hint: You will need to use rotations.) (10 points)

(3) Describe how to delete an element in  $T$  (while still maintaining the binary search tree and priority properties). (Hint: Same as part (2).) (10 points)

### Problem 4

In a binary tree, define the  $npl(x)$  of any node  $x$  to be the length of the shortest path from  $x$  to a node *without two children*.  $npl(x) = 0$  if  $x$  has no children.

A leftist tree is a binary tree in which for every node  $x$ ,  $npl(\text{left-child of } x) \geq npl(\text{right-child of } x)$ . Thus, leftist heaps preferred to be skewed to the left!

A leftist (min-)heap is a leftist tree which is (min-)heap-ordered.

While in a traditional binary heap, it is difficult to do union (i.e., merging two heaps of size  $\Theta(n)$  which can take  $\Theta(n)$  time), leftist heaps offer an elegant solution where all the basic heap operations (including union) can be done in  $O(\log n)$  time.

1. Give an example of a tree that is leftist and one that is not leftist. (10 points)

2. Show that a leftist tree with  $r$  nodes on the right path must have at least  $2^r - 1$  nodes. (Hint: use induction on  $r$ ).

Argue why this implies that on a leftist tree of  $n$  nodes has a right path containing at most  $O(\log n)$  nodes. (10 points)

3. Show how to union two leftist heaps (each of size  $n$ ) in  $O(\log n)$  time. Give a clear description of your algorithm. (Hint: It will be easier to describe your algorithm as a recursive one. If either of the two heaps are empty we return the other heap. Otherwise, to merge the two heaps we compare their roots. The main idea for union is to merge using the right path. Recursively merge the heap with the larger root with the *right* subheap of the heap with the smaller root (this is the recursive part, you don't have to worry about how it is done). However, when the recursion the resulting heap need not be leftist. You can easily fix this by doing a simple operation.)

Illustrate your algorithm by merging the two leftist heaps shown below. The heaps are given in the following recursive format:  $root(leftsubtree, rightsubtree)$ . ”-” means that there is no child.

Heap A:  $2(11(12, 17(18, -)), 5(8(15, -), -))$

Heap B:  $4(9(18(31, -), 10), 6(11(21, -), -))$ .

(15 points)

4. Show how to do extract-min and insert in  $O(\log n)$  time in a leftist heap. (Hint: Use union operation as a subroutine for both these operations.) ( 10 points)