Problem Set 5

This problem set is due in lecture on Monday, October 30, 2006. The homework lab for this problem set will be held 2–4 P.M. on Sunday, October 29, 2006, in room 32-124.

Reading: Chapter 14 and Skip List Lecture Notes.

Both exercises and problems should be solved, but only the problems should be turned in. Exercises are intended to help you master the course material. Even though you should not turn in the exercise solutions, you are responsible for material covered in the exercises.

Mark the top of each sheet with your name, the course number, the problem number, your recitation section, the date and the names of any students with whom you collaborated. Please staple and turn in your solutions on 3-hole punched paper.

You will often be called upon to “give an algorithm” to solve a certain problem. Your write-up should take the form of a short essay. A topic paragraph should summarize the problem you are solving and what your results are. The body of the essay should provide the following:

1. A description of the algorithm in English and, if helpful, pseudo-code.
2. At least one worked example or diagram to show more precisely how your algorithm works.
3. A proof (or indication) of the correctness of the algorithm.
4. An analysis of the running time of the algorithm.

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

Exercise 5-1. Do Exercise 14.1-5 on page 307 of CLRS.

Exercise 5-2. Do Exercise 14.2-1 on page 310 of CLRS.

Exercise 5-3. Do Exercise 14.3-4 on page 317 of CLRS.
Problem 5-1. The Inconvenient Defense

You are a Ph.D. student trying to organize your thesis defense. Your thesis committee has a very flexible timetable, so you get to choose to hold your defense at any time $t$ within the interval $[T_{\text{min}}, T_{\text{max}}]$ based on when your friends and acquaintances can attend. (Here $[a, b)$ denotes the interval of all times $t$ such that $a \leq t < b$. For the purposes of this problem, your defense is a single point in time; you can think of this as when your defense starts.) To this end, you create a web form where your friends can post the time intervals when they can attend. Your server should automatically generate (and update) the “best” time for your defense. However, in order to minimize the number of awkward questions from the audience, your definition of “best” is the time that minimizes the number of attendees. Your goal is to create an efficient dynamic data structure that supports the following two operations:

1. **INSERT**($[a, b)$), which inserts the interval $[a, b)$ (where $T_{\text{min}} \leq a < b \leq T_{\text{max}}$) when a friend can attend. Each friend may post multiple intervals, but they are guaranteed to be disjoint. Different friends may post overlapping intervals.

2. **PERFECT-TIME()**, which returns a time in $[T_{\text{min}}, T_{\text{max}})$ when the minimum number of people can attend your defense.

You decide to create your data structure by augmenting a balanced binary search tree such as red-black trees or AVL trees.

(a) Describe the data contained in each node of your augmented BST. (Hint: Read the rest of the problem first.)

(b) Describe your implementation of **INSERT**, disregarding for the moment rotations necessary to rebalance the tree. Prove its correctness and analyze its running time as a function of the current height $h$ of the tree. (Hint: Implement **INSERT**($[a, b)$) as a pair of subroutine calls, **ADD**($a$) and **SUB**($b$). **ADD**($a$) adds one attendee to all times at or beyond $a$; **SUB**($b$) subtracts one attendee from all times at or beyond $b$.)

(c) Describe your implementation of **INSERT**, taking into account possible rotations to rebalance the tree. Prove its correctness and analyze its running time as a function of the total number $n$ of intervals inserted so far.

(d) Describe your implementation of **PERFECT-TIME**, prove its correctness, and analyze its running time in terms of number $n$ of intervals inserted so far.
Problem 5-2. Random BSTs have logarithmic height with high probability

Armed with the idea of “with high probability” bounds from Lecture 12, and the knowledge that such bounds are usually much stronger than expectation bounds, you decide to revisit the bound on the height of a randomly built binary search tree from Lecture 9. Your goal is to prove that a randomly built binary search tree of size \( n \geq 1 \) (generated by inserting \( n \) keys in random order) has height \( O(\lg n) \) with high probability. Assume that the \( n \) keys are distinct.

You can view a randomly built binary search tree of size \( n \) as constructed by recursively applying the following construction step: randomly select one of the \( n \) elements as the root of the tree, then recursively construct the left subtree from the remaining elements smaller than the root (if any), and then recursively construct the right subtree from the remaining elements larger than the root (if any).

(a) Prove that there exists a constant \( \alpha < 1 \) such that, when constructing a random BST of size \( n \), any particular element \( x \) has probability at least \( \frac{1}{2} \) of either becoming the root or being placed into a subtree with at most \( \alpha n \) elements.

(b) Using the result from part (a), prove that the construction of a random BST of size \( n \) places any one particular element \( x \) at a depth \( O(\lg n) \) with high probability.

(c) Using the result from part (b), prove that a randomly built BST of size \( n \) has height \( O(\lg n) \) with high probability.