Advanced Data Structures
Homework 1

For this homework, you are welcome to use the internet and discuss the problems with others in the class - in fact, I highly encourage this! However, you must write up your own solutions, with NO verbatim copying of other's work. As a general rule, I'd suggest re-writing your solution without looking at any notes from your meetings or webpages, after you understand the solution. If you significantly used any online source, you must also include a reference to that website. Verbatim copying of any webpage will result in a 0 on the homework - use the internet to understand the solution, but ALWAYS rewrite in your own works while NOT looking at the page. (Note that I've googled all of these, so I have a pretty good idea of what's out there...)

I highly recommend using latex for typsetting your entire homework, but I'll accept the first two problems handwritten. (Please type problem 3 and submit via pdf, since submitting handwritten references never goes well.)

Required Problems

1. Suppose we want to maintain a Union-Find structure, but allowing items to be deleted. The standard way to do this is lazy deletes: mark items as deleted and have your "find" skip over them, but still use them in the tree connectivity structure. Give modified pseudocode for both the union and find functions with this addition, as well as (short) pseudocode for the delete function. What does this do to the worst case runtime analysis of union and find, and what is the worst case runtime for delete? (Justify your answer, but you shouldn't have to re-do all the analysis.) Will it change the amortized analysis, and why? (Don't re-do the entire analysis - just tell me some lemma from our prior analysis fails and how, or if it still works unchanged.)

2. Assume you have a Binary Search Tree data structure, specified with a node class or struct where each node contains some data, plus parent, left and right pointers. (The tree itself is specified by a pointer to the root node.) You are guaranteed that it is a BST, so for any node v, every node in the subtree rooted at left(v) has a value $\leq$ v's value, and any node in the subtree rooted at right(v) has a value $>$ v's value. You are NOT given any guarantee as far as height or balance factors, so the height can be linear in the size of the subtree in the worst case.

   Give an algorithm (in pseudocode) which, given a node v as input, rebalances the subtree rooted at v to be perfectly balanced in $O(size(v))$ time. So for a tree of size $O(n)$, your algorithm must rebalance in $O(n)$ time and use exactly $\lceil log_2 n \rceil$ levels. Justify correctness and runtime - I don't require full proofs, but do require an explanation beyond simple pseudocode, as in the various lecture not explanations.

3. Short essay/lit search question: Do a bit of internet searching on your own about skiplists. When are they useful in general, as opposed to binary search trees, and why? Find at least one or two specific examples, as well as links to your reference articles or books you find them in, and write a few paragraphs in your own words summarizing your findings. Please keep your answer to about a page, or 500(ish) words of text (excluding references and such).