University of Waterloo
CS240 Spring 2016
Assignment 3

Due Date: Wednesday, June 15, at 5:00pm

Please read `http://www.student.cs.uwaterloo.ca/~cs240/s16/guidelines.pdf` for guidelines on submission. This assignment contains only written problems. Submit your written solutions electronically as a PDF with file name a03wp.pdf using MarkUs. We will also accept individual question files named a03q1w.pdf, a03q2w.pdf, ... , a03q4w.pdf if you wish to submit questions as you complete them.

Problem 1 [6 marks]

We define the relation $A < B$ to mean that all of the keys in AVL tree A are smaller than all the keys in AVL tree B.

Design an algorithm to concatenate/merge any two AVL trees for which $A < B$. The result should be a single valid AVL tree. The worst-case running time of your algorithm should be $O(h)$ where h is the maximal height of the two trees. Explain how your algorithm satisfies these requirements.

Problem 2 [3+3+3+5=14 marks]

Suppose we have BST T. For any node $u \in T$, we define the left and right subtrees as $left(u)$ and $right(u)$. We also define $H(u)$ to be the height of u, and $N(u)$ to be the number of nodes in the subtree rooted at u ($N(u) = 1$ if u is a leaf and $N(u) = 0$ if u is the empty subtree).

a) [3 + 3 + 3 marks] Which of the following trees must be of height $O(logn)$? Prove or disprove.

1. There is a constant $c > 0$ such that for all nodes $u \in T$
   $H(left(u)) \leq H(right(u)) + c$.
2. There is a constant $c > 0$ such that for all nodes $u \in T$
   $H(left(u)) - c \leq H(right(u)) \leq H(left(u)) + c$.
3. Every internal node $u \in T$ has exactly two children.

b) [5 marks] Assume that there exists a constant $0 < c \leq 1$ such that for every node $u \in T$:

- $N(left(u)) \geq c \times N(right(u)) - 1$, and
- $N(right(u)) \geq c \times N(left(u)) - 1$.

Show that the tree has height $O(logn)$. (Hint: The constant hidden in big-O should depend on c.)

Problem 3 [4+4=8 marks]

We wish to improve upon binary search for the sorted array A with n distinct values. Instead of just blindly selecting the median index $i = \frac{l+r}{2}$ from the range of indices $A[l, r]$, *interpolation search* guesses the index of key k by estimating "how far away it should be from l". The guess is calculated by interpolating between the left (l) and right (r) boundary values.

InterpSearch($A[l, r]$, k)
A: an array
l: index of the left boundary
r: index of the right boundary
k: key to search for

if $A[l] > k \;\; || \;\; A[r] < k$
    return false
$i \leftarrow l + \lfloor (r - l)\frac{k - A[l]}{A[r] - A[l]} \rfloor$
if $A[i] = k$
    return true
else if $A[i] < k$
    return $InterpSearch(A[i + 1, r], k)$
else
    return $InterpSearch(A[l, i - 1], k)$

For example, let $A = (51, 58, 81, 87, 99)$ and $k = 87$. Starting the search in the range $l = 0$ and $r = 4$, then $i = 3$, and $A[i] = 87 = k$ so k is in A.

a) [4 marks] Which array with n distinct values results in the best case search time? Be as general as possible. Show that search always terminates in $O(1)$ time for this array, regardless of whether the key is stored in the array or not.

b) [4 marks] Which array values and search key yield the worst case search time? Give an array A with n values (i.e., by defining $A[i] = f(i)$ for some function f) that demonstrates the worst case search.

Problem 4 [5+6+4+4+5+(4)+(4)=24 marks]

Consider a procedure called $skipify$ which consumes an array of n items in ascending order, and then deterministically, produces a skip list with $log(n)$ levels: starting with the leftmost element $-\infty$, promote every other element from the previous level to the level above; as an exception, always promote $+\infty$ along with $-\infty$ to keep them the same height.

a) [5 marks] Write pseudocode for $skipify$ following the conventions and notation from the notes (Module 5, Slide 9/16). If you're latex-ing your solution, consider using an $algorithme$ object (see $LBSLInsert$ in A3.tex for an example). Your code should run in $\Theta(n)$ time–explain how this requirement is satisfied.

b) [6 marks] Define a *balanced skip list* (BSL) as a skip list built by $skipify$ but with exactly $n = 2^k - 1$ elements for some $k > 0$. Derive the best and worst case search times for a BSL (with $n = 2^k - 1$ elements). Assume that BSL search works exactly the same way as the randomly-generated skip lists from the lectures.

Suppose that we want to preserve the balanced structure of the BSL to guarantee the search times we derived in **part b**. This prevents us from modifying the BSL after building it with $skipify$, so there's no direct way to insert new elements.

Problem: design a data structure that provides search complexity similar to the BSL, but also allows insertion in a reasonable amount of time. Solution: since we can create as many BSLs as we want, we define a List of BSLs (LBSL) as follows.

An LBSL contains a set of BSLs, each containing $2^i - 1$ elements for some i values. For example, if $n = 11$, the LBSL contains a BSL with $2^3 - 1 = 7$ nodes, a BSL with $2^2 - 1 = 3$ nodes, and a BSL with $2^1 - 1 = 1$ nodes (Figure 1).

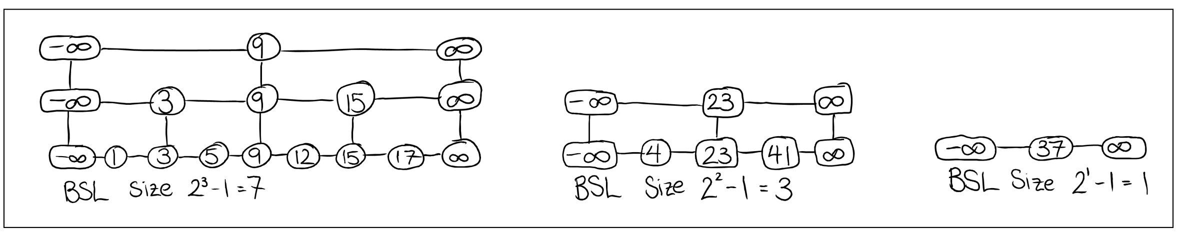

Figure 1: The LBSL produced after inserting elements (5, 17, 9, 12, 3, 1, 15, 41, 23, 4, 37).

To insert new elements, we merge smaller BSLs to make bigger ones:

LBSLInsert(e, L)
e: item to be inserted
L: an LBSL with $2^{k-1} < n \leq 2^k - 1$ *elements*

    for $i \leftarrow k$ **to** 1 **do**
        if L contains two BSLs A_1, A_2 of size $2^i - 1$
            using $skipify$, replace A_1, A_2 with a BSL of size $2^{i+1} - 1$ containing A_1, A_2 and e
            return
    create a new BSL E of size 1 that contains only e
    add E to L

c) [4 marks] Create an empty LBSL and insert values 1 through 7 using $LBSLInsert$. Draw the LBSL after every insertion.

d) [4 marks] Consider the worst case insertion time for an LBSL with n elements (not necessarily $n = 2^k - 1$). Describe the structure of the LBSL just before the insertion, and then derive the worst case insertion time.

e) [5 marks] Give pseudocode for searching the LBSL and derive the worst case runtime. Also, discuss whether theres an advantage to searching the BSLs from largest to smallest, or vice versa.

f) [Bonus: 4 marks] Insert calls $skipify$ every time it replaces two BSLs size $2^i - 1$ with a new BSL size $2^{i+1} - 1$. Consider the total cost of all $skipify$ calls after inserting values 1 through $n = 2^k - 1$. Show that this total cost is $\Theta(nlogn)$.

g) [Bonus: 4 marks] Show that the LBSL insertion cost, amortized over n elements, is $\Theta(logn)$. Note: we probably won't cover amortized analysis before this assignment is due. This bonus question is intended as an opportunity to do some independent investigation; in that regard, please do not ask questions about the bonus problems on Piazza.