CSC 263 Fall 2015
Midterm Solutions

Note: These are just solution sketches to help you learn. (They may contain errors! If so, please let me know!) For marking purposes, you were often required to give more detail that I am giving below.

1. (10 marks) We consider several data structures to store a set of n elements with distinct integer keys, and several operations that we want to perform.

   (a) For each data structure and each operation, give the **worst-case** runtime by filling in the table entries below with one of these options: $\Theta(1)$, $\Theta(\log n)$, $\Theta(n)$, $\Theta(n \log n)$.

   In the table below, MinHeap is a heap sorted in nondecreasing order, k is a key and x is a pointer to an element in the data structure. $SEARCH(k)$ returns the element with key k (without removing it); $FINDMIN$ returns the element with minimum key (without removing it); $DELETE(x)$ removes the element in the data structure pointed to by x; $SUCCESSOR(x)$ returns the element with the smallest key that is greater than the key pointed to by x. (and returns "not found" if no such element exists).

	MinHeap	AVL Trees
SEARCH(k)	$\log n$	$\log n$
FINDMIN	O(1)	$\log n$
DELETE(x)	$\log n$	$\log n$
SUCCESSOR(x)	n	$\log n$

   (b) Now the keys are stored in an array using hashing into an array of size M with chaining, where the hash function is chosen uniformly at random from an efficient universal family of hash functions from a universe $U = \{0, \ldots, N-1\}$ to $\{0, \ldots, M-1\}$. In the following table, list the *expected* runtime, assuming that selecting the hash function, and computing the hashed value of an element, are constant-time. Your answer should be one of these: $O(1)$, $O(N)$, $O(M)$, $O(m)$, where N is the size of U, M is the size of the array, and m is the total number of items stored.

	Hashing
SEARCH(k)	O(1)
FINDMIN	O(M)

2. (8 marks, 2 each) Let $\mathcal{A}$ be an algorithm, $T(n)$ be the worst-case time complexity of $\mathcal{A}$ on inputs of length n, and $g(n)$ a function, $g : N \to N$. Fill in the missing parts of each of the following statements with $\forall$, $\exists$, "at least" or "at most" so that they hold:

(a) $T(n)$ is $O(g(n))$ if and only if: $\exists c > 0$, $\exists n_0 > 0$, $\forall n \geq n_0$, **for all** x of size n, $\mathcal{A}$ takes **at most** $c \cdot g(n)$ steps when it is executed on input x.

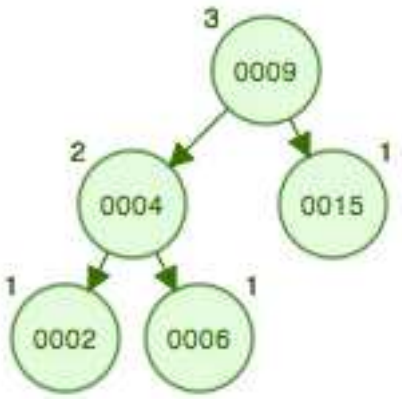
(b) $T(n)$ is $\Omega(g(n))$ if and only if: $\exists c > 0$, $\exists n_0 > 0$, $\forall n \geq n_0$, **there exists** x of size n, $\mathcal{A}$ takes **at least** $c \cdot g(n)$ steps when it is executed on input x.

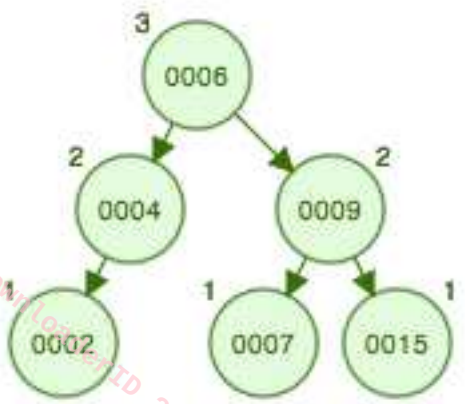
(c) True or False: For all functions g, $g(n)$ is $O(g(n))$. True

(d) True or False: For all functions g, $g(n)$ is not $O(g(n)^3)$. False

3. (14 marks, 7 each)

(a) Starting with the following AVL tree, already containing the items $2, 4, 6, 9, 15$. Show the tree that results from inserting 7. (Note: the labels indicate the height, with the convention that a tree with one vertex has height 1.)





(b) You are given the following array A of 8 integers: $[8, 3, 5, 4, 15, 12, 28, 14]$. Apply the algorithm Build-Max-Heap(), which was described in class to the array A and *show the resulting array*. Do *not* show any intermediate result.

The array order should be: 28, 15, 12, 14, 3, 8, 5, 4

4. (8 marks) Consider the following gambling game. Alice flips a fair coin 5 times, each time with probability 1/2 it is heads and with probability 1/2 it is tails. For every head, Alice earns 1 dollar, and for every tail she loses 25 cents. Assume that Alice starts with 5 dollars.

   (a) What is the smallest amount of money that Alice could end up with when the game is over, and what is the probability that this happens? Just give your answer; no justification necessary.

   If Alice loses every time, she will lose 1.25, and will be left with 3.75. This happens with probability 1/32.

   b) What is the expected amount of money that Alice will end up with? Show your work.

   We will calculate the expected amount of money that she wins/loses and then the expected amount that she will be left with will be the five dollars that she started with, minus this amount (by linearity of expectation). Let X be a random variable that is the amount of money that she wins/loses after playing the gambling game. There are 6 scenarios: she can win all 5 times, win 4 times and lose once, win 3 times and lose twice, win 2 times and lost 3 times, win 1 time and lose 4 times, and lose each time. If she wins i times and loses $5-i$ times, she earns $i-(5-i)/4$. The probability that she wins i times and loses $5-i$ times is: $\binom{5}{i}/32$. So we have: $E[X] = \sum_{i=0}^{5}(i-(5-i)/4)\binom{5}{i}/32$. Thus we have $E[X] = (20+75+150-5)/(32\times 4) = 240/128 = 15/8 = 1.875$. Thus the expected amount of money that she will have left is 6.875.

5. (15 marks) Consider an abstract data type that consists of a totally ordered set of keys V. Each key has an associated *count*, which is a non-negative integer. It supports two operations,

   - INC-COUNT(v), which increments the count associated with $v \in V$, and
   - DEC-BIG-COUNT(), which finds some $v \in V$ whose associated count has the largest positive value, decrements its count, and returns v. If all values in V have count 0, return *false*.

   Initially, the count of every value is 0. To implement this, we will use an AVL tree with one node for each $v \in V$ that has positive count. Both operations should take **worst-case** time $O(\log n)$, where n is the number of keys in V that have a positive count.

   (a) (5 marks) What fields would you add to each node of the tree?

   Use an AVL tree, where each node has a value, which is used as the key, and a positive count. Augment each node with an additional field, *biggest*, which contains the maximum value of count in all the nodes in the subtree rooted at this node.

   (b) (10 marks) Explain how to use AVL tree operations to perform INC-COUNT(v) in $O(\log n)$ time, and also how to perform DEC-BIG-COUNT() also in $O(\log n)$ time.

   We perform INC-COUNT(v) as follows. Search the AVL tree for v. If a node is found whose value is v, increment its count. Let c be the new value of its count. Go up the tree from this node incrementing every biggest field to c, until a node is found whose biggest field was already at least c or the biggest field of the root is set to c.

   If no such node is found, insert a new node with value v and count 1 into the AVL tree. Update the biggest fields as necessary, using the fact that the biggest field of a node is the maximum of its count field and the biggest fields of its children.

   This maintains the invariant that, at every node, *biggest* contains the maximum value of *count* in all the nodes in the subtree rooted at this node.

We perform DEC-BIG-COUNT() as follows. If the tree is empty, return false. Otherwise, let c be the value of the biggest field of the root. Go down the tree to find a node whose count is c: If the biggest field of its left subtree has value c, go left, else, if the biggest field of its right subtree has value c, go right. Otherwise, its count field has value c. Let v be its value field.

If $c > 1$, then decrement the count and biggest fields to $c - 1$. Go up the tree from this node decrementing every biggest field from c to $c - 1$, until the biggest field of the root has been decremented or a node is found whose sibling has a biggest field with value c or whose parent has a count field with value c.

If $c = 1$, then delete this node. Update the biggest fields as necessary, using the fact that the biggest field of a node is the maximum of its count field and the biggest fields of its children.

Return the value v.