University of Toronto
Faculty of Arts and Science

Midterm, Fall 2017

CSC263: Data Structure and Analysis

Duration: 50 minutes

First Name: _____ Last Name: _____ Student number: _____

5 questions, 9 pages (including this cover page and 3 blank pages at the end.) The last blank pages are for rough work only, they will not be marked.
Please bring any discrepancy to the attention of an invigilator.
Total Mark: 50
Answer all questions. WRITE LEGIBLY!
If you need to make any additional assumptions to answer a question, be sure to state those assumptions in your test booklet.
Answer the questions in the spaces provided on the question sheets. If you run out of room for an answer, use the last empty pages.

Question	Points	Score
1	10	
2	8	
3	12	
4	10	
5	10	
Total:	50	

1. (10 points) **HEAP**

   (a) A leaf in a tree is a node with no children. An internal node in a tree is a node with at least one child. So a node is an internal node if it is not a leaf. Recall that binary heaps learned in the course represent binary trees where the keys associated with nodes satisfy the heap property. Let H be a heap with $n = 263$ items. How many internal nodes does H have? Your answer should be an exact number. Do not justify your answer. 131

   (b) What is the height of a tree that represents a binary max heap with n elements. Your answer should be in the form of Θ. Explain in a short sentence why your bound is correct.
   $\Theta(\log n)$. Because, a binary max heap represents a nearly complete tree.

   (c) Briefly explain why the worst case time complexity for inserting an element into a binary heap is $O(\log n)$. Your answer needs a few sentences.

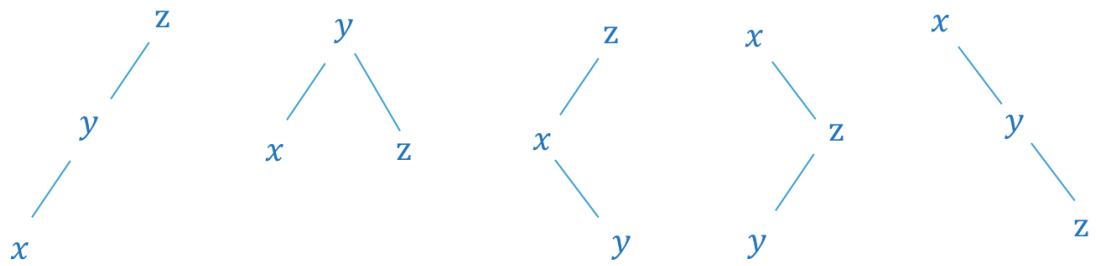
   For inserting an element to a max=heap, we increment the size of the array and insert the element into the last position of the array in a constant time and then we heapify or bubble up it, e.i. while the inserted element is larger than its parent, it is compared with its ancestors up to the root, and it is lager it is swapped with its parent. Therefore, the time complexity would be number of swaps. Since the number of swaps cannot be more than the hight of the tree, the upper bound for the procedure is $O(\log n)$

   (d) Show that the worst case time complexity for inserting in a binary heap is $\Omega(\log n)$.

   We need to show an input such that the number of swaps is $\Omega(\log n)$. Assume that the inserted element has a value larger than all the elements in the array. Then the procedure of swapping (bubble up or heapify) must continue from the leaf up to the root. So the lower bound is $\Omega(\log n)$.

2. (8 points) **BST**
Assume that 3 numbers $x < y < z$ are stored in a Binary search tree.

(a) What are the possible structures for the Binary search trees? Show all possible trees with values x, y, z in the nodes.



(b) Suppose that we pick one of these Binary search trees in part (a) uniformly at random and then call BST-Search algorithm to find the value x. What would be the average case for the number of nodes required to be looked-up to find x (including the node that stores x)?

Remark: Note that this question asks about the average number of look-ups for this specific given input stored in a BST with size 3 not for the general BST search algorithm.

Let x_i be a random variable that shows the number of lookups for tree i where $0 < i \leq 5$.
$E(X) = \sum_{i=1}^{5} x_i P(x_i) = 3 * 1/5 + 2 * 1/5 + 2 * 1/5 + 1 * 1/5 + 1 * 1/5 = 9/5$

3. (12 points) **HASH TABLES**

Consider inserting the keys $10, 22, 21, 4, 15, 59$ into a hash table of length $m = 11$ using open addressing with the primary hash function $h(k) = k \mod m$. Illustrate the result of inserting these keys in the table below and show the calculation of your hash keys. The initial table is empty in both parts.

(a) using quadratic probing with $c_1 = 1$ and $c_2 = 3$. (c_1 is the linear coefficient, while c_2 is the quadratic one.)

$h(k, i) = h(k) + c_i i + c_2 i^2 \mod m = (h(k) + i + 3i^2) \mod m$

0	1	2	3	4	5	6	7	8	9	10
22			21	4			59	15		10

$h(10, 0) = 10$
$h(22, 0) = 0$
$h(21, 0) = 10 \quad h(21, 1) = (10 + 1 + 3) \mod 11 = 3$
$h(4, 0) = 4$
$h(15, 0) = 4 \quad h(15, 1) = (4 + 1 + 3) \mod 11 = 8$
$h(59, 0) = 4 \quad h(59, 1) = (4 + 1 + 3) \mod 11 = 8$
. $\quad h(59, 2) = (4 + 2 + 12) \mod 11 = 7$

(b) using double hashing with $h_2(k) = 1 + (k \mod (m - 1))$.

$h(k, i) = h(k) + ih_2(k) \mod m$

0	1	2	3	4	5	6	7	8	9	10
22	21		59	4	15					10

$h(10, 0) = 10$
$h(22, 0) = 0$
$h(21, 0) = 10 \quad h(21, 1) = (10 + h_2(k)) \mod 11 = (10 + 2) \mod 11 = 1$
$h(4, 0) = 4$
$h(15, 0) = 4 \quad h(15, 1) = (4 + h_2(15)) \mod 11 = (4 + 6) \mod 11 = 10$
. $\quad h(15, 2) = (4 + 2h_2(15)) \mod 11 = (4 + 2 * 6) \mod 11 = 5$
$h(59, 0) = 4 \quad h(59, 1) = (4 + h_2(59)) \mod 11 = (4 + 10) \mod 11 = 3$

4. (10 points) **AUGMENTATION**

In this question, you will augment AVL trees to implement the following new operation:

- NumSmaller(k): return the number of elements with a key strictly smaller than k.

(a) What additional information will you store at each node of the AVL tree? Be specific.

$x.size$: The size of the subtree rooted at the node.

(b) Write an algorithm in pseudo-code for NumSmaller operation. What is the running time of your algorithm, briefly justify your answer. (You do not need to explain about Insert and Delete algorithms).

The idea is

- to find the largest element a in the tree with key $\leq k$, using a procedure similar to Tree-Search.
- Each time you descend a level and go to the right add the size of the subtrees you skip and finally return total calculated value.

```
NumSmaller(k)
1:  current = root, c = 0
2:  while (current ≠ NIL) and (current.key ≠ k):
3:      if (current.key < k)
4:          if current.left == NIL
5:              c = c + 1
6:          else
7:              c = c + current.left.size + 1
8:          current ← current.right
9:      else
10:         current ← current.left
11: return c
```

The running time is $O(\log n)$. Since the algorithm traverses the hight of the tree when updating the number of nodes, the running time cannot exceed $O(logn)$.

5. (10 points) **QUICKSORT**

(a) What are the best case and average case and worst case running time of the **quicksort** algorithm (not randomized version)? All your answers should be in terms of Θ.
Best case: $\Theta(n \log n)$ Average case: $\Theta(n \log n)$, ($\Theta(n^2)$ is also correct since the input distribution is not necessarily random Worst case $\Theta(n^2)$ Do not justify your answer.

(b) Randomized quicksort is a Las Vegas Algorithm. √ **True** ◯ False
Briefly justify your answer.

Randomized algorithm always has a correct answer, and most of the time fast e.i. $\Theta(n \log n)$.

(c) When the input is the array $[1, 5, 6, 3]$, what is the probability that **randomized quicksort** compares 1 and 5 directly to each other? Explain your reasoning.
Randomized quick sort compares 1 and 5 if and only if among the values 1, 3, 5 either 1 or 5 are chosen as a pivot for the first time. So the probability would be 2/3.