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: _____

6 questions, 9 pages (including this cover page and 3 blank pages at the end.)
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	14	
2	8	
3	4	
4	8	
5	8	
6	8	
Total:	50	

1. (14 points) **HEAP**

(a) In a max-heap with $n > 2$ distinct elements, the index of the minimum element is always $> \lfloor n/2 \rfloor$? (the first index is one) √ **True** ◯ False
Briefly justify your answer.

A minimum can only occur in one of the leaf nodes and the number of nodes which are not leaves is always $\lfloor n/2 \rfloor$.

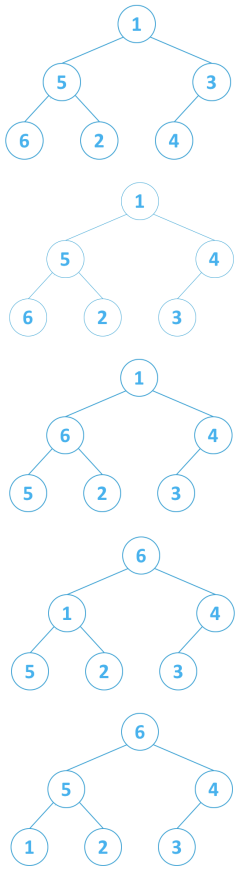
(b) In a max-heap with n distinct elements, the index of the median element is always $> \lfloor n/2 \rfloor$?
◯ True √ **False**
Briefly justify your answer.

A counter example is [5 3 4 2 1] the median is 3 and its index is 2.

(c) The efficient algorithm BUILD-MAX-HEAP provided in this course takes an array as an input and modifies the array such that the final result becomes a binary max heap. What is the complexity of BUILD-MAX-HEAP? Give your answer in terms of Θ notation. Do not justify your answer. $\underline{\Theta(n)}$

(d) Run the algorithm of BUILT-MAX-HEAP on the given array [1, 5, 3, 6, 2, 4] and show the result of the array step by step by filling the array. (If you need more number of steps, draw new arrays, if you need less, leave the last arrays empty.)

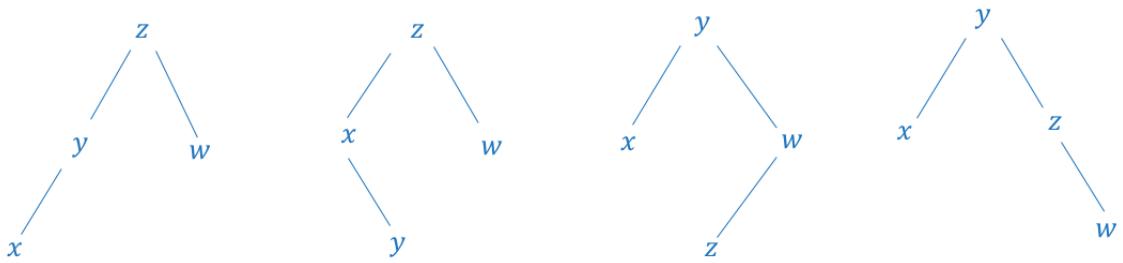
Step 0:	1	5	3	6	2	4
Step 1:	1	5	4	6	2	3
Step 2:	1	6	4	5	2	3
Step 3:	6	1	4	5	2	3
Step 4:	6	5	4	1	2	3



2. (8 points) **AVL-Tree**

Assume that 4 values $x < y < z < w$ are stored in an AVL tree.

(a) What are the possible different structures of AVL trees that store these values? Show all the possible structures by tree diagrams with values x, y, z, w inside the nodes.



(b) Assume that we pick one of the AVL trees in part (a) uniformly at random, and then we call the AVL-search algorithm to find the value x. What would be the **average case** for the number of look-ups of nodes required to find x (including the node that stores x) ? Justify your answer.

Remark: Note that this question asks about the average number of look-ups for this given input stored in an AVL tree with size 4 not for the general AVL algorithm.

Let x_i be a random variable that shows the number of lookups for tree i where $0 < i \leq 4$.
$E(X) = \sum_{i=1}^{4} x_i P(x_i) = 3 * 1/4 + 2 * 1/4 + 2 * 1/4 + 2 * 1/4 = 9/4$

3. (4 points) **HASH TABLES**

(a) The load factor α of an open addressing hash table must satisfy $\alpha \leq 1$.
√ **True** ◯ False Justify your answer in a short sentence.
α = number of elements / number of slots
In open addressing the number of elements cannot be more than the number of slots.

(b) In a hash table that uses linear probing and we might both insert and delete, insertion and search require the same number of look-ups. ◯ True √ **False**
Briefly justify your answer.
Counter example: for a given input x which does not exist in the list, insert(x) would look up the slots until it finds an empty or deleted slot, but search(x) would look up the slots until it finds an empty slot.

4. (8 points) **RANDOMIZED QUICKSORT**

When RANDOMIZED-QUICKSORT runs on a set of element with size n,

(a) how many calls are made to the random number generator RANDOM in the worst case to choose the random pivots?(in terms of Θ notation) $\underline{\Theta(n)}$

(b) How about in the best case? (in terms of Θ notation) $\underline{\Theta(n)}$
$\Theta(1)$ answers would also get a full mark if one assumed the improved version of quicksort for the case that all the elements are equal.
You do not need to justify your answer.

(c) When the input is the array $[1, 5, 6, 3, 9, 10]$, what is the probability that randomized quicksort compares 5 and 10 directly to each other? Explain your reasoning.

Randomized quick sort compares 5 and 10 if and only if among the values 5, 6, 9, 10 either 5 or 10 are chosen as a pivot for the first time. So the probability would be $2/4 = 1/2$.

5. (8 points) **ANAGRAMS**

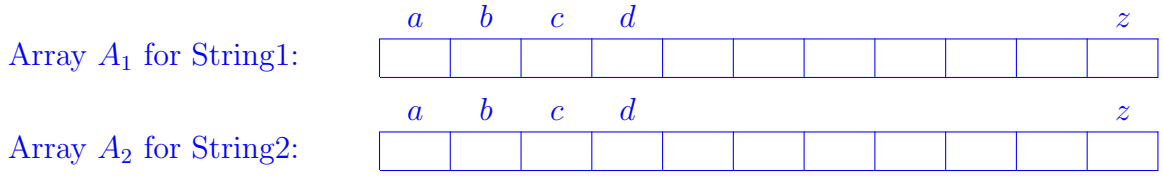
An *anagram* of a string is another string that contains the same characters, only the order of characters can be different. For example, "*abcd*" and "*dabc*" are anagrams of each other, but "*abcd*" and "*abdcb*" are not anagrams.

We are looking for an algorithm that checks whether two given strings with maximum n characters are anagram of each other. Assume that each string only could contain lower-case letters i.e. $\{a, ..., z\}$. The algorithm returns "YES" or "NO". It needs to be $O(n)$ time in the worst case.

Which data structure that you've learned in this course would be the best choice to implement the algorithm?Direct access table

Briefly explain the idea of your algorithm and explain the algorithm in a very high level description. You do not need to write the detailed pseudo-code and you do not need to explain the time complexity of your algorithm.

The idea is to use two arrays of size 26 (number of possible letters) as two direct access tables and then for the two given strings we compute the number of occurrence of each letter in $O(n)$. Then, we need to compare the two arrays to see if all the elements of the array have the same values.



The arrays store the occurrence of each letter for the two strings. Then, the two array should be compared.

6. (8 points) **BST**

Let T be a binary search tree and x and y be a pair of numbers such that $x \leq y$. Given T, x and y, the problem is to find the set of elements in T with key values in the range of $[x, y]$, if there is no element in the range "no element" should be printed. For example, if a BST stores values $\{2, 4, 5, 7, 9\}$ and $x = 3, y = 7$. then the algorithm prints $4, 5, 7$.

(a) Write a detailed algorithm (either in pseudo-code or in a form of step by step algorithm by plain English). You are allowed to call any procedures that you have learned in the lectures. The time complexity of your algorithm should be $O(h+k)$ where h is the height of the tree and k is the number of elements within the range. You are **not** allowed to use $O(n)$ or more space for your algorithm.

The idea is

- to find the smallest element a in the tree with key at least x, using a procedure similar to TREE-SEARCH.
- Then we can successfully call TREE-SUCCESSOR starting with a until the returned element has $key > y$ (or returns NIL)

```
RANGED-SEARCH(root, x, y):
1:  a ←FINDFIRST(root, x):
2:      if a == NIL :
3:          print ("no element")
4:      else:
5:          while a ≠NIL and a.key ≤ y
6:              print(a)
7:              a ← TREE-SUCCESSOR(a)

FINDFIRST(root, x):
1:  current = root, parent = NIL
2:  while (current ≠ NIL) and (current.key ≠ x):
3:      parent ← current
4:      if (current.key < x)
5:          current ← current.right
6:      else if (current.key > x)
7:          current ← current.left
8:  if (current == NIL)
9:      if (x > parent.key)
10:         a ← TREE-SUCCESSOR(parent)
11:     else
12:         a ← parent
13: else
14:     a ← current
15: return a
```

(b) Explain why the time complexity of your algorithm is the $O(h+k)$.

The algorithm clearly works because it keeps collecting elements within the range $[x, y]$ until there is no more left. The algorithm takes $O(h)$ time to find the element a and makes successive calls of TREE-SUCCESSOR to find the next k elements. It can be shown that k successive calls of procedure TREE-SUCCESSOR take a total of $O(h+k)$ time (exercise 12.2-8 from the CLRS) . Therefore, the total running time for this algorithm is $O(h+k)$.