CSC 263H5S 2017 Midterm Test
Duration — 50 minutes
Aids allowed: one double-sided 8.5"x11" sheet

Student Number: |_|_|_|_|_|_|_|_|_|_|_|_|_|

Last Name: _____ **First Name:** _____

☐ Registered Lecture Section: L0101 (Dan Zingaro)
☐ Registered Lecture Section: L0102 (Larry Zhang)

---

*Do **not** turn this page until you have received the signal to start.*
(Please fill out the identification section above, **write your name on the back of the test**, and read the instructions below.)

This test consists of 3 questions on 12 pages (including this page). When you receive the signal to start, please make sure that your copy is complete. If you write your answer in the space for rough work, indicate clearly what you want marked.

1: _____/24
2: _____/10
3: _____/10

TOTAL: _____/44

(L)

Question 1. [24 MARKS]

[15 minutes] Each of the following questions expects a short answer.

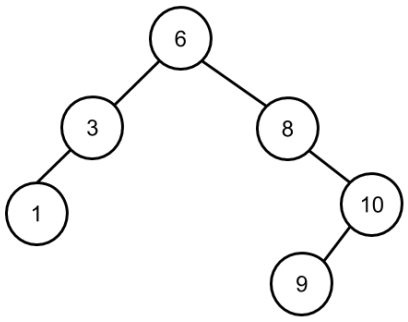
Part (a) [2 MARKS]

Consider an array $A[0 \ldots 9]$ (the index starts at 0 and the size is 10) that stores a binary max-heap. What are the indices that could hold the **second largest** element of the heap? Write all possible indices in the space below.

Part (b) [2 MARKS]

Consider the following BST T created by inserting a sequence of keys into an initially empty BST. How many different insert sequences are there that would result in exactly the same tree as T? Write your answer in the space below; do NOT show the steps of calculation.

Answer: _____



Part (c) [2 MARKS]

Consider inserting the sequence $63, 62, 61, \ldots, 1$ into an initially empty AVL-tree, what is the height of the resulting AVL-tree?

Answer: _____

Part (d) [4 MARKS]

You are given a binary tree T that satisfies the following conditions: (1) the left and right subtrees of T's root are both valid BSTs, and (2) T's root is larger than its left child and is smaller than its right child. Assume all nodes are unique. Is T guaranteed to be a valid BST? First, circle Yes or No, then briefly justify your answer.

Circle one: Yes / No

Justification:

Part (e) [4 MARKS]

Consider the following claim on open-addressing hash tables: After calling `Insert(1)`, `Insert(2)`, ... , `Insert(m)` on an open-addressing hash table of size `m`, the load factor of the table is guaranteed to be 1.0. Is this claim true or false? First, circle True or False, then briefly justify your answer.

Circle one: True / False

Justification:

Part (f) [4 MARKS]

Recall the lecture example of expanding dynamic arrays during a sequence of `APPEND` operations. Our analysis showed that, by doubling the size of the array when it is full (i.e., expansion factor = 2), the amortized cost per operation is tightly upper-bounded by 3. Now we want to change the expansion factor so that the amortized cost per operation becomes tightly upper-bounded by 7. What should the new expansion factor be? Write your answer in the space below and briefly justify.

New expansion factor: _____

Justification:

Part (g) [6 MARKS]

Suppose that we have an undirected graph $G = (V, E)$ with $V = \{1, 2, \ldots, |V|\}$, and we want to implement an operation `GET-NEIGHBOURS`(G, i) that returns the list of neighbours of vertex i in G. We are considering the following three data structures for implementing the graph.

- Adjacency matrix, as discussed in the lecture.
- Adjacency list, as discussed in the lecture.
- Edge list: simply use an unordered list of pairs (i, j) to store all edges in the graph.

What are the worst-case runtimes of `GET-NEIGHBOURS` for each of the implementations? Your answer for each space below should be one of the following: $\Theta(1)$, $\Theta(|V|)$, $\Theta(|E|)$, $\Theta(|V| + |E|)$, $\Theta(|V|^2)$, and $\Theta(|E|^2)$. No justification is needed.

Adjacency matrix: _____

Adjacency list: _____

Edge list: _____

[This page will be marked only if you clearly indicate the part that should be marked.]

Question 2. [10 MARKS]

[10 minutes] You'll get 2 out of 10 marks if you leave this question completely blank.

Consider the following operation called `FIND-LARRY` that takes a list of strings as the input and outputs the first index that holds the string "Larry". We want to analyze the runtime of this operation by counting how many times **Line #3** is executed, i.e., the number of comparisons made. Let n denote the length of `L`.

```
1 def FIND-LARRY(L):
2     for i from 0 to n-1:     # list index starts at 0
3         if A[i] == "Larry":  # count only this line
4             return i
5     return -1
```

Input distribution: The sample space consists of all permutations of a list of distinct strings `["Arnold", "Dan", "Larry", ..., "Zelda"]`, and the length of the list is at least 2. These permutated lists are distributed in such a way that the string "Larry" is **equally likely** to appear at any index from 0 to $n-2$, and "Larry" never appears at index $n-1$.

Now, perform runtime analysis for `FIND-LARRY` by answering the following questions. All your answers must be in exact forms rather than in asymptotic notations.

Part (a) [1 MARK]

In the best case, what is the number of comparisons made by `FIND-LARRY`?

Part (b) [1 MARK]

What is the probability of the best case?

Part (c) [1 MARK]

In the worst case, what is the number of comparisons made by `FIND-LARRY`?

Part (d) [1 MARK]

What is the probability of the worst case?

Part (e) [6 MARKS]

In the average case, what is the expected number of comparisons made by `FIND-LARRY`? Show detailed steps of your calculation.

Question 3. [10 MARKS]

[12 minutes] You'll get 2 out of 10 marks if you leave this question completely blank.

Imagine you are the owner of a bank and have access to the dataset of a large collection of transactions. The amount of each transaction in the collection is a positive integer that can be arbitrarily large, and there can be duplicate amount values, i.e., there may be multiple transactions having the same amount. Your job is to design a data structure to implement an ADT called `BANK-SET` that stores the collection of the transaction amounts and supports the following operations. Let S denote a given instance of `BANK-SET`.

- `INSERT-TRANSACTION`(S, x): Insert a transaction of amount x into S.
- `RANGE-QUERY`(S, x_1, x_2): Return the number of transactions whose amounts are between x_1 and x_2, inclusive, i.e., the number of transactions whose amounts x satisfy $x_1 \leq x \leq x_2$.

Requirements: Let n denote the total number of transactions stored in S.

- `INSERT-TRANSACTION` must have worst-case runtime $\mathcal{O}(\log n)$.
- `RANGE-QUERY` must also have worst-case runtime $\mathcal{O}(\log n)$.

You will design this data structure using an **augmented AVL-tree**. Describe this design by answering the following questions. You may use operations and results from lectures without repeating their details.

(a) What attributes are stored in each node of the AVL-tree? No need to mention standard AVL-tree attributes including `x.left`, `x.right`, `x.p`, and `x.balance_factor`.

(b) Which node attribute is the sorting key of the AVL-tree?

(c) Which node attribute is the augmenting attribute, and why can it be maintained efficiently upon modifications to the AVL-tree?

(d) Explain in clear and concise English how `INSERT-TRANSACTION`(S, x) works, and justify why its worst-case runtime is $\mathcal{O}(\log n)$.

(e) Explain in clear and concise English how `RANGE-QUERY`(S, x_1, x_2) works; write pseudocode if necessary. Justify why it works correctly and why its worst-case runtime is $\mathcal{O}(\log n)$. In particular, explain why it works correctly when there are duplicate amount values.

[This page will be marked only if you clearly indicate the part that should be marked.]

[This page will be marked only if you clearly indicate the part that should be marked.]

Last Name: _____ **First Name:** _____