UNIVERSITY OF TORONTO
Faculty of Arts & Science

Winter 2025 Test 1-v2

CSC 111 H1S

Duration: 90 minutes

Aids Allowed: None

*Do **not** turn this page until you have received the signal to start.* In the meantime, fill out your information below (please do this now!) and *carefully* read *all* the information on the rest of this page.

First name (please write as legibly as possible within the boxes)

Last name

Student ID Number (DIGITS only)

- This test consists of 4 questions on 6 pages (including this one), printed on both sides of the paper. *When you receive the signal to start, please make sure that your copy of the test is complete.*
- Answer each question directly on the test paper, in the space provided.

Marking Guide

Nº 1: _____/ 5

Nº 2: _____/ 5

Nº 3: _____/ 5

Nº 4: _____/ 5

TOTAL: _____/20

Question 1. Linked Lists [5 marks]

Based on the provided code for the `LinkedList` method below: (1) fill in any necessary preconditions, (2) fill in the correct, expected values for the two doctest examples, and (3) write a good docstring description.

```
class LinkedList:
    def mystery(self) -> None:
        """









        Preconditions:
        -



        >>> lst = LinkedList([1])
        >>> lst.mystery()
        >>> lst.to_list()




        >>> lst = LinkedList([2, 1, 4])
        >>> lst.mystery()
        >>> lst.to_list()



        """

        new_node = _Node(self._first.item - 1)
        new_node.next = self._first
        self._first = new_node

        curr = self._first.next
        while curr.next is not None:
            new_node = _Node(curr.next.item - 1)
            new_node.next = curr.next
            curr.next = new_node
            curr = new_node.next
```

Question 2. Recursion [5 MARKS]

Complete the following recursive function according to its docstring:

```
def nested_list_equal(nested_list1: int | list, nested_list2: int | list) -> bool:
    """Return whether two nested lists are equal, i.e., have the same value.

    Note: order matters.

    >>> nested_list_equal(17, [1, 2, 3])
    False
    >>> nested_list_equal([1, 2, [1, 2], 4], [1, 2, [1, 2], 4])
    True
    >>> nested_list_equal([1, 2, [1, 2], 4], [4, 2, [2, 1], 3])
    False
    """

    # COMPLETE THE FUNCTION BODY BELOW; YOUR CODE MUST BE RECURSIVE
```

Question 3. Trees [5 marks]

Fill in the blanks below to complete the following `Tree` method according to the docstring.

Your method **must be recursive**. **Do NOT use any other Tree methods** (except this method itself, i.e. `insert_child`) within the code you write below.

```
class Tree:
    def insert_child(self, item: Any, parent: Any) -> bool:
        """Insert <item> as a new value in this tree, as a child of <parent>.

        If successful, return True. If <parent> is not in this tree,
        return False.

        If <parent> appears more than once in this tree, <item> should only
        be inserted once (you can pick where to insert it).
        """

        # COMPLETE THE METHOD BODY BELOW; YOUR CODE MUST BE RECURSIVE

        if self.is_empty():




        elif self._root == parent:




        else:
```

Question 4. Binary Search Trees [5 MARKS]

Consider the following method added to our `BinarySearchTree` class:

```
class BinarySearchTree:
    def sum_in_range(self, low: int, high: int) -> int:
        """Return the sum of the values in this binary search tree that are
        between low and high, inclusive.

        Preconditions:
            - high >= low
            - every value in this binary search tree is an integer.
        """
        if self.is_empty():
            return 0
        elif self._root < low:
            return self._right.sum_in_range(low, high)
        elif self._root > high:
            return self._left.sum_in_range(low, high)
        else:
            left_sum = self._left.sum_in_range(low, high)
            right_sum = self._right.sum_in_range(low, high)
            return left_sum + right_sum
```

Part (a) [2 MARKS]

Which of the following binary search trees, and the associated method call, does the above code **NOT** work correctly for? **Select ALL that apply, by clearly circling your choices.**

1.
```
>>> bst = BinarySearchTree(None) # empty tree
>>> bst.sum_in_range(0, 4)
```

2.
```
>>> bst = BinarySearchTree(5)
>>> bst.sum_in_range(2, 6)
```

3.
```
>>> bst = BinarySearchTree(5, BinarySearchTree(2), BinarySearchTree(8))
>>> bst.sum_in_range(0, 3)
```

4.
```
>>> bst = BinarySearchTree(7, \
      BinarySearchTree(4, BinarySearchTree(1), BinarySearchTree(6)), \
      BinarySearchTree(10))
>>> bst.sum_in_range(42, 100)
```

Part (b) [1 MARK]

Briefly explain in words, what the issue with the code is.

Part (c) [2 MARKS]

Annotate the code above (that is, either add some code to, and/or cross out lines and re-write them if needed) to fix the implementation (**keep your changes minimal**).

[Use the space below for rough work. This page will not be marked unless you clearly indicate the part of your work that you want us to mark.]