Easy4.0 CSC148

Mock Final Exam III

This Final Examination paper consists of 6questions with a sum of 61 points in total, and you must earn at least 40% to pass the exam. Comments and docstrings are not required except where indicated, although they may help us mark your answers.

- You do not need to put import statements in your answers.
- No error checking is required: assume all user input and all argument values are valid.
- If you use any space for rough work, indicate clearly what you want marked.

Name:_____

.

Marking Guide

Q1	**/8**
Q2	**/10**
Q3	**/9**
Q4	**/14**
Q5	**/10**
Q6	**/10**
Total	**/61**

Question 1 [8 Marks]

Recall the Tree data structure we've defined in class.

```
class Tree:
    """
    A bare-bones Tree ADT that identifies the root with the entire tree.
    === Attributes ===
    @param object value: value of root node
    @param list[Tree|None] children: child nodes
    """
    def __init__(self, value=None, children=None):
        """
        Create Tree self with content value and 0 or more children
        @param Tree self: this tree
        @param object value: value contained in this tree
        @param list[Tree|None] children: possibly-empty list of children
        @rtype: None
        """
        self._value = value
        # copy children if not None
        # NEVER have a mutable default parameter...
        self._children = children[:] if children is not None else []
    # make self.value and self.children read-only by setting
    # only the get field of their property
    def _get_value(self):
        return self._value
    value = property(_get_value)
    def _get_children(self):
        return self._children
    children = property(_get_children)
```

Implement the method "gather_by_level" defined below according to its docstring. Assume root level starts at 1.

```
def gather_by_level (t: Tree) -> Dict[int:List[Object]]:
  """ Return a dictionary that the key is the levels and the value is all
     the node on that level.

  >>> t = Tree(5, [Tree(3), Tree(2)])
  >>> gather_by_level(t) == {1: [5], 2: [3, 2]}
  True

  """
```

Question 2 [10 Marks]
Suppose we have the DoubleLinkedList defined as follow:

```
class DoubleLinkedList:

    def __init__(self):
        self.front, self.back, self.size = None, None, 0

class Node:

    def __init__(self, value, next=None, prev=None):
        self.value, self.next, self.prev = value, next, prev
```

Implement the insert function for DoubleLinkedlist [8 marks]

```
def insert(self, value: object):
```

Question 3 [9 Marks]
Please indicate the Big-O expression of the following problems with a brief explanation.

Question a)

Function call	Runtime
f(5)	3.75s
f(6)	5.4s
f(7)	7.3s
f(8)	9.6s
f(9)	1.215

Question b)

```
def mystery2(n):
    for i in range(n * 2):
        if i % 2 == 1:
            for j in range(n):
                print(j)
        elif n % 2 == 1:
            for j in range(n**2):
                print(j)
```

Question c)

Function call	Runtime
f(5)	1s
f(6)	1s
f(7)	70s
f(8)	80s
f(9)	90s
f(10)	100s

Question 4 [14 Marks]

In assignments, we have designed classes for different games, now please implement the 9x9 sudoku based on the puzzle class provided as follow. The init method of Sudoku should take a board parameter that represents the current board of the Sudoku.

```
class Puzzle:
    """Abstract class for a generic one-player puzzle.

    Note that this class really represents a puzzle *state* and not just a generic type of n
    puzzle. In other words, an instance of this class could represent one particular
    puzzle, with a partially filled-in board.
    """

    def is_solved(self) -> bool:
        """Return whether this puzzle is in a solved state.
        """
        raise NotImplementedError()

    def all_possible_moves(self) -> list[Puzzle]:
        """ Return all possible moves that can be applied to this puzzle.
        """
        raise NotImplementedError()

    def make_move(self, move: Any) -> Puzzle:
        """
        Return the Puzzle that results from applying move to this Puzzle.
        """
        raise NotImplementedError()
```

Question 5 [10 Marks]

Recall the Binary Search Tree data structure we've defined in class.

```
class BinaryTree:
    """
    A Binary Tree, i.e. arity 2.
    """
    def __init__(self, value, left=None, right=None):
        """
        Create BinaryTree self with value and children left and right.
        @param BinaryTree self: this binary tree
        @param object value: value of this node
        @param BinaryTree|None left: left child
        @param BinaryTree|None right: right child
        @rtype: None
        """
        self.value, self.left, self.right = value, left, right
```

a) Implement the following question

```
def get_largest_odd(t) -> int:
    """Return the largest odd value in the Tree, return None if there
    is no odd value.

    Assume all values in the tree is int.
     """


    "
```

b) Briefly explain both the best and worst case runtime for Part a) using Big-O notation.

Question 6 [6 Marks]

1. Please provide a scenario that you would use hashtable instead of list, why?

2. Please trace the merge function call as calling merge sort on list
   [8, 1, 2, 4, 7, 5, 13, 6,]
   In the order they should happens in the execution.
   For example, merging list [1] and [2] can be expressed as merge([1], [2])