Duration: **50 minutes**
Aids Allowed: ***None***

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

Last (Family) Name(s): ______________________________

First (Given) Name(s): ______________________________

---

*Do **not** turn this page until you have received the signal to start.*
In the meantime, please read the instructions below *carefully*.

This term test consists of 4 questions on 10 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, fill in the identification section above, and write your name on the back of the last page.*

Answer each question directly on the test paper, in the space provided, and use one of the "blank" pages for rough work. If you need more space for one of your solutions, use a "blank" page and *indicate clearly the part of your work that should be marked.*

In your answers, you may use any of the Python builtin functions and standard modules listed on the accompanying Python reference sheet. You must write your own code for everything else.

Comments and docstrings are *not required*, except where indicated, although they may help us grade your answers. Also, they may be worth part marks if you cannot write a complete answer.

If you are unable to answer a question (or part of a question), remember that you will get 10% of the marks for any solution that you leave *entirely blank* (or where you cross off everything you wrote to make it clear that it should not be marked).

Marking Guide

# 1:	______/12
# 2:	______/10
# 3:	______/10
# 4:	______/ 8
TOTAL:	______/40

Good Luck!

Question 1. [12 MARKS]

Implement each method in class `Queue` below. Use instances of class `LLNode` to store the items in a linked list. *Do not use any builtin Python container in your solution!* (No `list`, `set`, `dict`, `file`, or even `str`.) **Your code will be marked on its style and design in addition to its correctness.**

```
class LLNode:
    """A node in a linked list."""
    def __init__(self, item, link=None):
        self.item = item
        self.link = link


class EmptyQueueError(Exception):
    pass


class Queue:
    """Implementation of the Queue ADT."""

    def __init__(self):
        """Initialize this queue to be empty."""







    def enqueue(self, item):
        """Add item to the back of this queue."""
```

[...continued on the next page...]

Question 1. (continued)

```
# ...class Queue, continued...

    def dequeue(self):
        """Remove the front item from this queue and return that item.
        Raise EmptyQueueError if this queue is empty."""
```

```
    def is_empty(self):
        """Return True iff this queue is empty."""
```

Question 2. [10 MARKS]

Consider the following node class (for linked lists).

```
class LLNode:
    """A node in a linked list."""
    def __init__(self, item, link=None):
        self.item = item
        self.link = link
```

Write the body of the function below to satisfy its docstring. This will require a bit of thinking, so it would be a good idea to provide some internal comments to describe what you are trying to do.
Your code will be marked on its style and design in addition to its correctness.

```
def merge_odd(head):
    """(LLNode) -> NoneType
    Modify the linked list starting at node head so that every two consecutive
    odd integers are replaced by their sum (this will require removing nodes
    from the list). Precondition: every item in the linked list is an integer.
    For example, if head refers to a linked list as follows:
            head: 1 -> 3 -> 5 -> 7 -> 9
    then after the call merge_odd(head), the linked list should look like this
    (1 and 3 have been added together, 5 and 7 have been added together):
            head: 4 -> 12 -> 9
    (Note: this function does NOT return anything.)
    """
```

Use the space on this "blank" page for scratch work, or for any solution that did not fit elsewhere.
Clearly label each such solution with the appropriate question and part number.

Question 3. [10 MARKS]

Consider the following node class (for binary trees).

```
class BTNode:
    """A node in a binary tree."""
    def __init__(self, item, left=None, right=None):
        self.item = item
        self.left = left
        self.right = right
```

Part (a) [4 MARKS]

Write the body of the method below to satisfy its docstring. (*Note*: This is a **method** *inside* class `BTNode`.)
Your code will be marked on its style and design in addition to its correctness.

```
    def invert(self):
        """(BTNode) -> NoneType
        Replace every item in the subtree rooted at this node with its inverse
        (1 / item). Precondition: every item in the subtree rooted at this node
        is a number. For example, if self is a reference to the root of the
        tree pictured on the left below, then after this method finishes
        executing, the tree's items have changed as pictured on the right.

            before invert()                after invert()
                  2                              0.5
                 / \                            /   \
                1   4                         1.0   0.25
                   /                                 /
                  5                                0.2
        """
```

Question 3. (continued)

Part (b) [6 marks]

Write the body of the function below to satisfy its docstring. (*Note*: This is a **function** *outside* class BTNode.) **Your code will be marked on its style and design in addition to its correctness.**

```
def singles_list(root):
    """(BTNode) -> [object]                                              A
    Return a list of every item stored in a node with only one child,     \
    in the binary tree rooted at root. For example, if root refers to      C
    the root of the binary tree pictured on the right, then the return    /
    value should be [A, C] (or [C, A] -- order does not matter).         D
    """
```

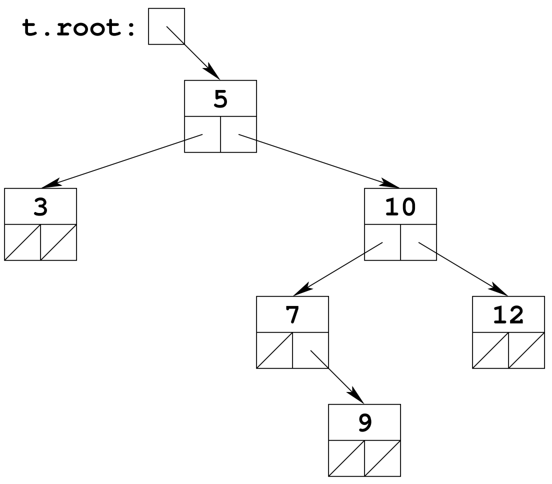
Question 4. [8 MARKS]

Part (a) [4 MARKS]

Draw **two different** Binary Search Trees that each store the three values `1`, `2`, `3`.

Part (b) [4 MARKS]

Let `t` be the binary search tree pictured below. Use the rest of the space on this page to draw a picture of the tree after performing the operations `t.insert(4)` and `t.insert(8)`, in that order. (For your reference, we provide parts of the BST code from class on the next page.)



Question 4. (continued)

Part (b) (continued)

```
class BST:
    # ...other methods omitted...
    def insert(self, item):
        self.root = _insert(self.root, item)

def _insert(root, item):
    if root is None:
        root = _BSTNode(item)
    elif item < root.item:
        root.left = _insert(root.left, item)
    elif item > root.item:
        root.right = _insert(root.right, item)
    else:  # item == root.item
        pass
    return root
```

On this page, please write nothing except your name.

Last (Family) Name(s):	
First (Given) Name(s):	

Total Marks = 40