Question 1. [15 MARKS]

This question is about **linked lists**.

You are given a linked list class that maintains a reference to the first (head) and last (tail) nodes in the list (as you saw in lab). Answer the following multiple-choice questions by circling one of the answers.

Part (a) [2 MARKS]

[True / False] Adding to the head of the linked list is faster (in general) than adding to the tail of the list.

False, both take the same amount of time (since we have a tail pointer).

Part (b) [2 MARKS]

[True / False] Removing from the tail of the list is faster (in general) than removing from the middle of the list.

False, removing from the tail is not made faster by the tail pointer.

Part (c) [3 MARKS]

If we wanted to use such a linked list to implement a Queue, which end should you make the front of the list (as a reminder, dequeue removes from the front)? [the head / the tail / doesn't matter]

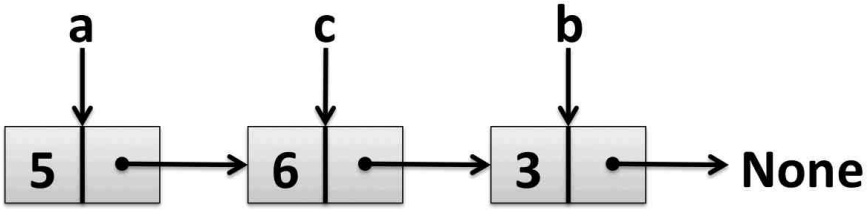
This question was not worded reasonably, and won't be marked.

Part (d) [8 MARKS]

Draw the linked list structure(s) that you are left with and what the `a`, `b`, and `c` variables point to at the end of the following program. You should follow the form we used in lecture, with each `Node` represented as a rectangle with two cells, one for each instance variable (the left `data` and the right for `next`).

```
class Node(object):
    def __init__(self, data, next=None):
        self.data = data
        self.next = next

a = Node(5)
b = Node(3, a)
c = Node(6, b)
a.next = c
c.next.next = None
# Draw the linked list structure(s) and what a, b, and c point to at this point in the code
```



Question 2. [10 MARKS]

This question is about **object oriented programming**.

You are given the following `Point` class (no need to import it):

```
class Point(object):
    """A 2-D Cartesian coordinate"""
    def __init__(self, x, y):
        """(Point, int, int) -> NoneType
        A Point at (x, y)
        """
        self.x = x
        self.y = y
```

Part (a) [5 MARKS]

Write a class called `Line` that represents a line in a 2-D Cartesian plane. `Line` objects should have no public instance variables (but may have private ones). Line objects are created from two `Point` objects that represent the end points of the line, `p1`: `(x1, y1)` and `p2`: `(x2, y2)`, with the additional constraint that `x2` is at least `x1`. If `x1` is greater than `x2`, the constructor should raise a `ValueError` (you do not need to supply an error message). `Line` objects should show up in print statements as:
`(<x1 value>, <y1 value>, <x2 value>, <y2 value>)`

Your `Line` class should work with the following testing code:

```
v = Line(Point(0, 1), Point(0, -1))
assert str(v) == "(0, 1, 0, -1)"
try:
    x = Line(Point(5, 5), Point(0, 0))
    assert False
except ValueError:
    pass
```

Write your `Line` class here:

```
class Line(object):
    def __init__(self, p1, p2):
        """(Line, Point, Point) -> NoneType"""
        if p1.x > p2.x:
            raise ValueError()

        self._p1 = p1
        self._p2 = p2

    def __str__(self):
        """Line -> str"""
        return "(%d, %d, %d, %d)" % (self._p1.x, self._p1.y,
                                     self._p2.x, self._p2.y)
```

(Question continued on next page.)

Part (b) [5 MARKS]

Write a method called `midpoint` (to go inside your `Line` class) that returns a new `Point` at the midpoint of the `Line` (the point halfway between the two end points). Make sure to call the `Point` constructor with values of the appropriate type. Integer division should be used, so the midpoint of (0, 0) and (2, 5) is (1, 2). Your method should work with the following code:

```
mid = Line(Point(0, 0), Point(2, 5)).midpoint()
assert mid.x == 1
assert mid.y == 2
```

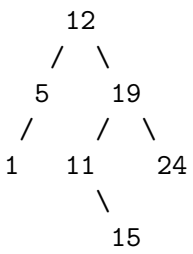
Write your `midpoint` method here:

```
def midpoint(self):
    """Line -> Point"""
    x = int((self._p1.x + self._p2.x) / 2)
    y = int((self._p1.y + self._p2.y) / 2)
    return Point(x, y)
```

Question 3. [20 MARKS]

This question is about **binary trees**.

The questions on this page concern the following binary tree:



Part (a) [1 MARK] What is the height of this tree? (hint: length in nodes)

4

Part (b) [1 MARK] What is the tree's branching factor?

2

Part (c) [1 MARK] What is the depth of the 24? (hint: depth in edges)

2

Part (d) [1 MARK] What is the height of the subtree rooted at 19?

3 (was incorrectly posted as 2)

Part (e) [2 MARKS] Is this tree a BST? [Yes / No]

No

Part (f) [2 MARKS] What is the pre-order traversal of this tree? (numbers, comma-separated)

12, 5, 1, 19, 11, 15, 24

Part (g) [2 MARKS] What is the post-order traversal of this tree?

1, 5, 15, 11, 24, 19, 12

Part (h) [2 MARKS] What is the in-order traversal of this tree?

1, 5, 12, 11, 15, 19, 24

(Question continued on next page.)

Part (i) [8 MARKS]

This part uses the following `Node` class:

```
class Node(object):
    def __init__(self, data, left=None, right=None):
        self.data = data
        self.left = left
        self.right = right
```

Using recursion, complete the following function. You are not allowed to define any helper functions. For example, calling `has_leaf_at_depth` on the 12 node of the tree on the previous page will return `True` for depths 2 and 3. An empty tree (root is `None`) is not a leaf node. *Hint: an easy base case is when you are at a leaf node.*

```
def has_leaf_at_depth(root, depth):
    """(Node, int) -> bool
    Return whether or not the binary tree rooted at root has a leaf
    node at the given depth.
    """

    if root is None:
        return False
    elif root.left is None and root.right is None:
        return depth == 0
    else:
        return has_leaf_at_depth(root.left, depth - 1) or \
            has_leaf_at_depth(root.right, depth - 1)
```

Question 4. [10 MARKS]

This question is about **assignment 1**.

You have a functional solution to assignment 1 in a file called `a1_solution.py` in the current directory. On the next page, write a `LazyCustomer` class (like a `SimpleCustomer`, but lazier!) to go in a different file in the same directory. As in the assignment, this class should have just one method, `get_drink`, that takes a `Refrigerator` object as a parameter (in addition to `self`) and either returns a `Drink` object or `None` as follows:

```
A LazyCustomer will only ever look at the first Shelf in the Refrigerator.
If the first shelf is empty, get_drink will return None.
Otherwise, get_drink removes and returns the first Drink on the first Shelf.
```

As a reminder, here are the classes and relevant methods you might interact with:

```
Refrigerator:
    __iter__(self): Refrigerator -> Iterator(Shelf)
        Allows iteration over the Shelf objects in the refrigerator with a for loop
    __len__(self): Refrigerator -> int
        Allows len(r) to return number of Shelfs in Refrigerator r

Shelf:
    put_drink(self, drink): (Shelf, Drink) -> NoneType
        Add the Drink to the front of the Shelf
    take_drink(self): Shelf -> Drink
        Remove and return Drink at the front of the Shelf
    stock_drink(self, drink): (Shelf, Drink) -> NoneType
        Add the Drink to the back of the Shelf
    is_empty(self): Shelf -> bool
        Return whether or not the Shelf is empty
    is_full(self): Shelf -> bool
        Return whether or not the Shelf is full
    __len__(self): Shelf -> int
        Allows len(s) to return number of drinks in Shelf s

Drink:
    days_until_expiry(self): Drink -> int
        Return the number of days until the Drink expires
    is_expired(self): Drink -> bool
        Return whether or not the Drink is expired
```

(Question continued on next page.)

Part (a) [2 MARKS]

The `Refrigerator`, `Shelf`, and `Drink` class definitions are in `a1_solution.py`, and not in the file you are writing. If your `LazyCustomer` class requires any imports, write them in the space below. If no imports are necessary, draw a line through the space below.

Part (b) [8 MARKS] Write your `LazyCustomer` class in the space below:

```
class LazyCustomer(object):
    def get_drink(self, refrigerator):
        """(LazyCustomer, Refrigerator) -> Drink"""
        for shelf in refrigerator:
            if shelf.is_empty():
                return None
            else:
                return shelf.take_drink()
```

Question 5. [20 MARKS]

This question is about **stacks**, **queues**, and **unittest**.

Part (a) [12 MARKS]

Write a function called `abs_sorted_to_sorted(q)` that takes a `Queue` containing numbers sorted by their absolute value as its argument (the first number dequeued is the one with the lowest absolute value). The function should *modify* `q` so that it contains the same numbers sorted normally (the first number dequeued is the one with the lowest value). You are only allowed to use `Queue`s and `Stack`s as temporary data structures. The only `Queue` methods you are allowed to use are `enqueue(o)`, `dequeue()`, and `is_empty()`. The only `Stack` methods you are allowed to use are `push(o)`, `pop()`, and `is_empty()`.

For example:

A `Queue`'s contents before calling `abs_sorted_to_sorted`: `[1, -2, 3, -4]`

The same `Queue`'s contents after calling the function on it: `[-4, -2, 1, 3]`

Write your `abs_sorted_to_sorted` method in the space below:

```
def abs_sorted_to_sorted(q):
    """Queue -> NoneType"""
    pos = Queue()
    neg = Stack()
    while not q.is_empty():
        e = q.dequeue()
        if e >= 0:
            pos.enqueue(e)
        else:
            neg.push(e)

    while not neg.is_empty():
        q.enqueue(neg.pop())

    while not pos.is_empty():
        q.enqueue(pos.dequeue())
```

(Question continued on next page.)

The next parts concern the following unit test code (assume that `abs_sorted_to_sorted` is defined in the same file):

```
import unittest

class TestQueueSorting(unittest.TestCase):
    def setUp(self):
        print "set up Q"
        self.queue = Queue()

    def tearDown(self):
        print "tear down Q"
        self.queue = None

    def test_one_pos(self):
        print "test with one positive number"
        self.queue.enqueue(1.5)
        abs_sorted_to_sorted(self.queue)
        self.assertEqual(self.queue.dequeue(), -2530)  # this will fail

if __name__ == '__main__':
    unittest.main()
```

Part (b) [4 MARKS]

What is printed by the program? (Write the output of just the `print` statements.)

```
set up Q
test with one positive number
tear down Q
```

Part (c) [4 MARKS]

Write a test method for class `TestQueueSorting` that tests `abs_sorted_to_sorted` on a more interesting `Queue` of numbers. You should test the general correctness of the function, so think about what sorts of numbers you should enqueue (*hint: more than just positive integers*). You do not need to provide a docstring. Make sure the `Queue` you give to `abs_sorted_to_sorted` adheres to the function's specification.

```
def test_thorough(self):
    values = [0, -0.1, 1, 5, -5, 5]
    for i in values:
        self.queue.enqueue(i)
    values.sort()
    abs_sorted_to_sorted(self.queue)
    for i in values:
        self.assertEqual(self.queue.dequeue(), i)
```