Please read the following guidelines carefully!

- Please write your name on the front and back of the exam.
- This examination has 3 questions. There are a total of 10 pages, DOUBLE-SIDED.
- You may always write helper functions/methods unless explicitly asked not to.
- Docstrings are not required unless explicitly asked for.

Take a deep breath.

This is your chance to show us

How much you’ve learned.

We WANT to give you the credit

That you’ve earned.

A number does not define you.

Good luck!
1. [12 marks] Point-form responses are acceptable here. You do not need to write a lot for full marks.

(a) [3 marks] Fill in the two missing pieces of code in this method from class LinkedList:

```python
def remove_last(self):
    """Remove the last item from this list. Do nothing if this list is empty."
    
    @type self: LinkedList
    @rtype: None
    
    # TODO: Initialize variables as needed.

    while current is not None and current.next is not None:
        previous = current
        current = current.next
    # TODO: Modify the list as needed.
```

(b) [2 marks] Draw the binary search tree that results if we insert these values, in this order:

10  31  25  6  18  40  8

If we were to delete the root, which of the leaf values in the tree could replace the root value without changing the position of any other nodes? If there is more than one, name them all.
(c) [2 marks] Recall that we defined the height of a tree in terms of nodes, so that the height of a tree with just a root is 1. What is the greatest possible height of a binary search tree with 8 nodes?

Draw a binary search tree of ints that has 8 nodes and the least possible height. Include the int in each node.

(d) [3 marks] Consider the following method for the Tree class.

```python
def longest_path(self):
    """Return a list of the items on the path from the root to the deepest leaf in the tree. If there is a tie, pick any such leaf."
    """
    if self.is_empty():
        return []
    else:
        max_path = []
        for subtree in self._subtrees:
            new_path = self.longest_path()
            if len(new_path) > len(max_path):
                max_path = new_path
        return [self._root] + max_path
```

Draw an example of a tree where this method will fail to do its documented behaviour.

What happens when you run `longest_path` on this tree? Be specific.

How do you fix this problem?
(e) [2 marks] Here is a `BinarySearchTree` method that returns the maximum item in a binary search tree.

```python
def find_max(self):
    """Return the maximum item in this BST, or None if this BST is empty."""
    if self._right.is_empty():
        return self._root
    else:
        return self._right.find_max()
```

Describe a tree on which this method would fail.

What happens when you run `find_max` on this tree? Be specific.
2. **[7 marks]** In the space below, implement the given method, to be added to class `LinkedList`. You may not use any `LinkedList` methods in your solution. Note the given precondition: this limits the cases you need to handle.

```python
def slice_out(self, j, k):
    """Remove the items in this list from positions <j> to <k-1> inclusive.

    Precondition: 0 < j <= k <= len(self)
    @type self: LinkedList
    @rtype: None
    >>> linky = LinkedList([0, 1, 2, 3, 4, 5, 6, 7])
    >>> str(linky)
    '[0 -> 1 -> 2 -> 3 -> 4 -> 5 -> 6 -> 7]'
    >>> linky.slice_out(2, 5)
    >>> str(linky)
    '[0 -> 1 -> 5 -> 6 -> 7]'  
    """
```
3. [9 marks] Consider the Tree method `partition_leaves`, which returns a tuple of two lists, where the first list contains the leaves of the tree that are negative, and the second list contains the leaves that are greater than or equal to 0. Assume that all values stored in the tree are integers.

(a) [1 mark] Suppose we have a variable `t` that is a Tree instance representing the following tree:

What should be the output of `t.partition_leaves()`?

(b) [2 marks] For each subtree of `t`, draw it, and write down below it what `partition_leaves` should return for it.

(c) [2 marks] Explain in English how to compute `t.partition_leaves()` from the recursive calls on its subtrees.
(d) [4 marks] In the space below, implement the `partition_leaves` method using recursion. You may not use any `Tree` methods other than `is_empty`.

```python
def partition_leaves(self):
    """Return a tuple of the negative and non-negative leaves in this tree.

    The first element of the tuple is a list of the values of the leaves that are less than 0. The second element of the tuple is a list of the values of the leaves that are greater than or equal to 0.

    Order does not matter within the two lists.

    Precondition: all items in this tree are integers.
    
    @type self: Tree
    @rtype: (list, list)
    """
    # HINT. Divide your code into the following three cases:
    # 1. When the tree is empty.
    # 2. When the tree has just a single item (a leaf).
    # 3. When the tree has at least one subtree.
```
Use this page for rough work. If you want work on this page to be marked, please indicate this clearly at the location of the original question.
Use this page for rough work. If you want work on this page to be marked, please indicate this clearly at the location of the original question.
Name:

<table>
<thead>
<tr>
<th>Grade</th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Out Of</td>
<td>12</td>
<td>7</td>
<td>9</td>
<td>28</td>
</tr>
</tbody>
</table>