Name:

Student Number:

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) [2 marks] Here is a function that takes in a list of LinkedListRec objects, and counts the total number of occurrences of a given item in these lists by calling the count method from the LinkedListRec class. Assume count has been implemented properly.

```python
def count_all(linked_lists, item):
    """Count the number of occurrences of <item> in the list of LinkedListRec objects."""
    total = 0
    for linky in linked_lists:
        total += linky.count(item)
    return total
```

Recall that our recursive linked list class, LinkedListRec, included this representation invariant:

    _first is None if and only if _rest is None. This represents an empty list.

This ensures that count_all works properly even when linked_lists contains empty lists. Suppose we represent an empty linked list using None instead. Describe a call to count_all that wouldn’t work in this case, and state exactly what would go wrong.

Revise the body of the function so that it would work if we represented the empty list using None.

(b) [3 marks] Consider 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
    """
    previous = None
    current = self._first
    while current is not None and current.next is not None:
        previous = current
        current = current.next
    if current is not None:
        previous.next = current.next
```

Describe a linked list on which this method would fail, and state exactly what would go wrong.

Make a small change to the code above that will fix it.
(c) [2 marks] Draw a binary tree whose post-order traversal is this: 5 9 4 8 1 2 3 7

(d) [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. Draw a binary search tree of height 3 containing the numbers 1 through 7 inclusive.

Could such a tree have the value 5 at the root? Circle one answer: Yes No

Explain.
(e) [3 marks] Here is a recursive function that counts the number of positive integers in a nested list:

```python
def count_pos(obj):
    """Return the number of integers greater than 0 in the given nested list.
    @type obj: int | list
    @rtype: int
    """
    count = 0
    if isinstance(obj, int):
        if obj > 0:
            return 1 + count
        else:
            return count
    else:
        for lst_i in obj:
            count += count_pos(lst_i)
```

Give a concrete example of a nested list where this method will fail to do its documented behaviour.

What happens when you run `count_pos` on this nested list? Be specific.

On the code above, make one small change that will fix it.
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.

```python
def merge_pairs(self):
    """Merge each consecutive pair of elements in this list, replacing the two values with their sum.

    If there is an odd number of elements in this list, leave the final element unchanged.

    Precondition: each item in this list is an int.
    """

    @type self: LinkedList
    @rtype: None

>>> linky = LinkedList([1, 2, 30, 40, 5, 6, 7])
>>> str(linky)
'1 -> 2 -> 30 -> 40 -> 5 -> 6 -> 7'
>>> linky.merge_pairs()
>>> str(linky)
'3 -> 70 -> 11 -> 7'
```
```
3. [9 marks] Consider the Tree method `longest_path`, which returns a list of items on the longest possible path between the root of the tree and one of its leaves. The list is ordered by increasing depth, so the tree’s root is always the first element.

If there is more than one longest possible path, this method returns the one that ends at the leaf that is furthest to the left. Assume all subtrees are stored in left-to-right order in the `_subtrees` attribute.

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

```
    17
   / \   \   
  -2  3   4
 /   \   \   
5     6   -7   8
       /   \   
      -8   13   9
```

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

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

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

```python
def longest_path(self):
    """Return a list of items on the longest possible path between the root of this tree and one of its leaves.

    If there is more than one path with the maximum length, return the one that ends at the leaf that is furthest to the left.

    If this tree is empty, return an empty list.

@type self: Tree
@rtype: list
"""
```
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></th>
<th>Q1</th>
<th>Q2</th>
<th>Q3</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Grade</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Out Of</td>
<td>12</td>
<td>7</td>
<td>9</td>
<td>28</td>
</tr>
</tbody>
</table>