Question 1. [8 marks]

Read the docstring below for method `remove_first_satisfier`. You may assume that classes `LinkedListNode` and `LinkedList` from the API have been imported. Implement `remove_first_satisfier`. Note: The only `LinkedList` and `LinkedListNode` methods provided are those in the API.

```python
def remove_first_satisfier(self, predicate):
    """
    Remove first node whose value satisfies (returns True for) predicate. If there is no such node, leave self as is.
    
    @param LinkedList self: this linked list
    @param (object)->bool predicate: boolean function
    @rtype: None
    
    >>> list_ = LinkedList()
    >>> list_.append(5)
    >>> list_.append(3)
    >>> print(list_.front)
    5 -> 3 ->|
    >>> def f(n): return n > 4
    >>> list_.remove_first_satisfier(f)
    >>> print(list_.front)
    3 ->|
    >>> list_.append(5)
    >>> list_.append(7)
    >>> list_.remove_first_satisfier(f)
    >>> print(list_.front)
    3 -> 7 ->|
    """

previous_node, current_node = None, self.front
while current_node is not None and not predicate(current_node.value):
    previous_node = current_node
    current_node = current_node.next_
if current_node is not None:
    # current_node.value satisfies predicate
    if previous_node is None:
        # current_node was front
        self.front = current_node.next_
    else:
        # previous_node is a LinkedListNode
        previous_node.next_ = current_node.next_
        if self.back is current_node:
            self.back = previous_node
```
Question 2. [8 MARKS]

Read the docstring for function `count_lists` below, and then implement it.

```python
def count_lists(list_):
    """
    Return the number of lists, including list_ itself, contained in list_.
    
    @param list list_: list to count lists in
    @rtype: int
    
    >>> count_lists([])
    1
    >>> count_lists([5, [1, [2, 3], 4], 6])
    3
    """
    return 1 + sum([count_lists(c)  
                    if isinstance(c, list) else 0
                    for c in list_])
```

Question 3. [8 MARKS]

Read the docstring below for function `list_even_below`, as well as the API for class `Tree`. You may assume that class `Tree` has been imported, as well as helper function `gather_lists`. Implement function `list_even_below`.

Hint: The depth of a node is 1 less than the depth of its children.

```python
def list_even_below(t, n):
    """
    Return a list of even values in t with depth greater than n.
    
    Assume any values in t are integers.
    
    @param Tree t: tree to list values from
    @param int n: depth below which to list values
    @rtype: list[int]
    
    >>> t1 = Tree(5)
    >>> t2 = Tree(4)
    >>> t3 = Tree(2, [t1, t2])
```
>>> list_even_below(t3, 0)
[4]
"

if n >= 0 or t.value % 2 == 1:
    list_ = []
else:
    list_ = [t.value]
return list_ + gather_lists([list_even_below(c, n-1) for c in t.children])

Question 4.  [6 MARKS]

Read the docstring below and the API for BinaryTree. Then implement height.

def height(t):
    ""
    Return the height of BinaryTree t, that is 1 more than the maximum of the height of its children, 1 if t has no children, or 0 if t is the empty tree.
    
    @param BinaryTree|None t: possibly empty BinaryTree
    @rtype: int

    >>> height(None)
    0
    >>> t1 = BinaryTree(5)
    >>> t2 = BinaryTree(4, t1, None)
    >>> height(t1)
    1
    >>> height(t2)
    2
    ""

    if t is None:
        return 0
    else:
        return 1 + max(height(t.left), height(t.right))