Question 1:

What is the result of applying the following operations on a `Stack`? Provide the output of the code fragment.

```
stack = Stack(5)
stack.push('cat')
stack.push('dog')
print(stack.pop())
stack.push('fish')
stack.push('moose')
print(stack.pop())
print(stack.pop())
stack.push('bear')
print(stack.pop())
print(stack.pop())
```

Question 2:

What is the result of applying the following operations on a `Queue`? Provide the output of the code fragment.

```
queue = Queue(5)
queue.enqueue('cat')
queue.enqueue('dog')
print(queue.dequeue())
queue.enqueue('fish')
queue.enqueue('moose')
print(queue.dequeue())
print(queue.dequeue())
queue.enqueue('bear')
print(queue.dequeue())
print(queue.dequeue())
```

Question 3:

A common exercise is to implement a `Stack` using two `Queue` data structures. The following code implements this concept by using a list as a `Queue`, appending to the end and extracting only from the beginning.

```
def __init__(self):
     self.q1 = []
     self.q2 = []

 def push(self, x):
     self.q1.append(x)

 def pop(self):
     while len(self.q1) > 1:
        self.q2.append(self.q1.pop(0))
     item = self.q1.pop(0)
     while len(self.q2) > 0:
        self.q1.append(self.q2.pop(0))
     return item
```

(a) What is the big-oh running time of `push` in this implementation?

(b) What is the big-oh running time of `pop` in this implementation?

Question 4:

Given the `Queue` implementation in question 3, implement a method `empty` that returns True if the queue is empty and False otherwise.

Question 5:
Consider the `Message Board` application you worked with for Lab 10. Describe how you would add the ability to associate tags with each message and view all messages with a given tag. Your answer should include (1) how the web interface would change; (2) how the data structure(s) storing the data would change; and (3) how the logic for displaying all messages for a given tag would operate. It is *not* expected that you provide a complete implementation of this change. Instead, clearly describe the design of your revised solution.

Question 6:

What is the big-oh running time of the check_rows function you implemented for Project 3? Make sure to provide your answer in terms of n ***and*** to define what n represents.

Question 7:
The following code implements `bubble_sort`.

```
def bubble_sort(things):
    '''
    Function: bubble_sort -- sorting the elements of a list by
swapping
                             two consecutive elements that are out of
order
    Parameters:
       things -- the elements to be sorted
    Returns a new list with all of the elements in sorted order
    '''
    swapped = True
    while swapped is True:
        swapped = False
        for i in range(0, len(things) - 1):
            if things[i] > things[i + 1]:
                things[i], things[i + 1] = things[i+1], things[i]
                swapped = True
```

The code `things[i], things[i + 1] = things[i+1], things[i]` performs a single *swap*.

Given the following input, *how many swaps* will occur in the ***first*** iteration of the `while` loop?

```
things = [5, 1, 3, 2, 7, 6, 4]
```

For full credit, explain your answer.

Question 8:

Write a ***recursive*** function called all_stars that computes a new string where all the adjacent characters are now separate by a “*”.

```
all_star("hello") → "h*e*l*l*o"
all_star("abc") → "a*b*c"
all_star("ab") → "a*b"
```