1. The textbook's ordered list ADT uses a singly-linked list implementation. I added the `size`, `tail`, `current`, `previous`, and `currentIndex` attributes:

The `search(targetItem)` method searches for `targetItem` in the list. It returns `True` if `targetItem` is in the list; otherwise it returns `False`. Additionally, it has the side-effects of setting `current`, `previous`, and `currentIndex`. The complete `search(targetItem)` method code for the `OrderedList` is:

```python
class OrderedList:

    def search(self, targetItem):
        if self._current != None and self._current.getData() == targetItem:
            return True

        self._previous = None
        self._current = self._head
        self._currentIndex = 0
        while self._current != None:
            if self._current.getData() == targetItem:
                return True
            elif self._current.getData() > targetItem:
                return False
            else: # inch-worm down list
                self._previous = self._current
                self._current = self._current.getNext()
                self._currentIndex += 1
            return False

a) What's the purpose of the "elif self._current.getData() > targetItem:" check?

Since the list is ordered, we can stop because we would have found
the targetItem already if it was in the list.

Consider the `add(item)` method with the precondition: `item` is not in the list.

b) Write the precondition check at the start of the `add(item)` method.

```python
if self.search(item) == True:
    raise Exception("cannot add duplicate items")
```

c) Suppose you are adding the item value of 's'. Update the above picture for this "normal" case, and number the steps in the drawing.

d) What special cases need to be considered for the `add` method?

- before first Node
- after last Node
- first in list
2. A recursive function is one that calls itself. Complete the recursive code for the `countDown` function that is passed a starting value and proceeds to count down to zero and prints "Blast Off!!!".

Hint: The `countDown` function, like most recursive functions, solves a problem by splitting the problem into one or more simpler problems of the same type. For example, `countDown(10)` prints the first value (i.e., 10) and then solves the simpler problem of counting down from 9. To prevent "infinite recursion", if-statement(s) are used to check for trivial base case(s) of the problem that can be solved without recursion. Here, when we reach a `countDown(0)` problem we can just print "Blast Off!!!".

```python
# File: countDown.py

def main():
    start = eval(input("Enter count down start: "))
    print("\nCount Down:")
    countDown(start)

def countDown(count):
    if count == 0:
        print("Blast Off!!!")
    else:
        print(count)
        countDown(count-1)
```

Program Output:
Enter count down start: 10
Count Down:
10
9
8
7
6
5
4
3
2
1
Blast Off!!!

(a) Trace the function call `countDown(5)` on paper by drawing the run-time stack and showing the output.

(b) What do you think will happen if your call `countDown(-1)`?

(c) Why is there a limit on the depth of recursion?

Memory is finite (default 1000)
3. The non-recursive `__str__` method for `OrderedList` object below would return: "(head)(a m)(tail)"

```python
def __str__(self):
    resultStr = "(head) "
    current = self._head
    while current != None:
        resultStr += str(current.getData()) + " "
        current = current.getNext()
    return resultStr + "(tail)"
```)

We can thing of building the string for the list as "a" + (string for the rest of the list)
a) Complete the recursive `strHelper` function in the `__str__` method for our `OrderedList` class.

```python
def __str__(self):
    """ Returns a string representation of the list with a space between each item. """

def strHelper(current):
    if current is None:
        return ""
    else:
        return str(current.getData()) + " " + strHelper(current.getNext())

# Start of __str__ method execution
return "(head) " + strHelper(self._head) + "(tail)"
```

4. Some mathematical concepts are defined by recursive definitions. One example is the Fibonacci series:

\[0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89\]

After the second number, each number in the series is the sum of the two previous numbers. The Fibonacci series can be defined recursively as:

\[Fib_0 = 0, \quad Fib_1 = 1, \quad Fib_n = Fib_{n-1} + Fib_{n-2} \text{ for } n \geq 2.\]

a) Complete the recursive function:

```python
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)
```

b) Draw the call tree for `fib(5)`.