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

```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 they are ordered items, we would have found it already.

b) Complete the `add(item)` method including a check of its precondition: `newItem` is not in the list.

```python
def add(self, newItem):
    if self.search(newItem):
        raise ValueError("Cannot add duplicate to ordered list.")

    temp = Node(newItem)
    temp.setNext(self.current)
    if self.previous == None:
        self.head = temp
    else:
        self.previous.setNext(temp)
    self.size += 1
    self.current = temp
```

Lecture 9 - Page 1
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!!!”.

```
""" 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)

main()
```

---

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)`?  

```
while count >= 0:
    print(count)
    count = count - 1
print("Blast Off!!!")
```

---

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

---

```
return to (execute)  
return to (execute)
return to (execute)
return to (execute)
return to (execute)
return to (execute)
return to (execute)
```

---

Memory is limited so run-time stack is too.
Function call - cause call-frame to be pushed on the run-time stack

- call-frame has three things
  - return address - where to return after function ends
  - parameters
  - local variables - new variables defined in the function

When function ends or "returns", we pop the call-frame and return execution at the return address. If no value is returned, the default is the None value.
3. 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 == None:
            return ""
        else:
            return str(current.getData()) + " " + strHelper(current.getNext())
    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, ...

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:

   $Fib_0 = 0$

   $Fib_1 = 1$

   $Fib_n = Fib_{n-1} + Fib_{n-2}$ for $N \geq 2$.

a) Complete the recursive function:

   ```python
def fib (n):
```

b) Draw the call tree for `fib(5)`. 
Non-recursive scan for ordered list

```python
    temp = self._head
    result Str = ""
    while temp != None:
        result Str += str(temp.getData()) + " ",
        temp = temp.getNext()
    return result Str
```