1. The textbook's unordered list ADT uses a singly-linked list implementation. I added the _size and _tail attributes:

```
class UnorderedList:
    def search(self, targetItem):
        if self._current != None and self._current.getData() == targetItem:
            return True
        self._current = self._next
        self._previous = None
        self._current = self._head
        while self._current != None:
            if self._current.getData() == targetItem:
                return True
            else:
                self._previous = self._current
                self._current = self._current.getNext()
        return False
```

a) The search(targetItem) method searches for targetItem in the list. It returns True if targetItem is in the list; otherwise, it returns False. Complete the search(targetItem) method code:

b) The textbook's unordered list ADT does not allow duplicate items, so operations add(item), append(item), and insert(pos, item) would have what precondition?

```
item is not already in list
```

c) Complete the append(item) method including a check of it's precondition(s):

```
def append(self, item):
    if self.search(item):
        raise Exception("cannot append duplicate items")
    temp = Node(item)
    if self._size == 0:
        self._head = temp
    else:
        self._tail.setNext(temp)
```

d) Why do you suppose I added a _tail attribute? to make append O(1)

```
self._tail = temp
self._size += 1
```
e) The textbook's `remove(item)` and `index(item)` operations "Assume the item is present in the list." Thus, they would have a precondition like "Item is in the list." When writing a program using an UnorderedList object (say `myGroceryList = UnorderedList()`), how would the programmer check if the precondition is satisfied?

```python
itemToRemove = input("Enter the item to remove from the Grocery list: ")
if myGroceryList.search(itemToRemove):
    myGroceryList.remove(itemToRemove)
```

f) The `remove(item)` and `index(item)` methods both need to look for the item. What is inefficient in this whole process? Code using the list will call `search` to make sure item is in the list, remove will start with precondition check by calling `search`, and remove code must "search" down linked list looking for item to remove. Three traversals down list needed.

g) Modify the `search(targetItem)` method code in (a) to set additional data attributes to aid the implementation of the `remove(item)` and `index(item)` methods.

h) Write the `index(item)` method including a check of its precondition(s).

```python
def index(self, item):
    if self.search(item) == False:
        raise Exception("")
    return self.currentIndex
```

i) Write the `remove(item)` method including a check of its precondition(s).

```python
def remove(self, item):
```
Remove normal case - remove from middle of list

Normal case picture redrawn:

```
self
- head
- size: 4
- previous
- current: 3
- currentIndex
```

1. 2. 3. 4.

6: temp

4:

return y