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

   ```python
   class UnorderedList:
       def __init__(self):
           self._size = 0
           self._tail = None

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

    while self._current is not None and current.getData() == targetItem:
        self._previous = self._current
        self._current = self._current.getNext()
        if self._current is None:
            return False
    return True

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 in list`

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

   ```python
   def append(self, item):
       if self.search(item):
           raise ValueError("Cannot append duplicate items.")

       temp = Node(item)
       if self._size == 0:
           self._head = temp
           self._tail = temp
       else:
           self._tail.setNext(temp)
           self._size += 1
   ```

d) Why do you suppose I added a `_tail` attribute?

   `So we can find tail node in O(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
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?

```python
def remove(self, item):
    if self.search(item) == False:
        raise ValueError("Cannot remove item not in list.")
    temp = self._current
    if self._current == self._tail:
        if self._previous == None:
            self._head = self._current.get_next()
        else:
            self._previous.set_next(self._current.get_next())
    self._size -= 1
    self._current = None
    return temp.get_data()
```

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. (Note: added `self._currentIndex` counter to search method)

```python
def index(self, item):
    if self.search(item) == False:
        raise ValueError("Cannot find item in list.")
    return self._currentIndex
```

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

```python
def remove(self, item):
    if self.search(item) == False:
        raise ValueError("Cannot remove item not in list.")
    temp = self._current
    if self._current == self._tail:
        if self._previous == None:
            self._head = self._current.get_next()
        else:
            self._previous.set_next(self._current.get_next())
    self._size -= 1
    self._current = None
    return temp.get_data()
```