1 More Practice with Linked Lists

```java
public class SLList {
    private class IntNode {
        public int item;
        public IntNode next;
        public IntNode(int item, IntNode next) {
            this.item = item;
            this.next = next;
        }
    }
    private IntNode first;
    public void addFirst(int x) {
        first = new IntNode(x, first);
    }
}
```

1.1 Implement SLList.insert which takes in an integer `x` and inserts it at the given position. If the position is after the end of the list, insert the new node at the end.

For example, if the SLList is `5 → 6 → 2`, insert(10, 1) results in `5 → 10 → 6 → 2`.

```java
public void insert(int item, int position) {
```
Add another method to the SLList class that reverses the elements. Do this using the existing IntNodes (you should not use new).

```
public void reverse() {
}
```

Extra: If you wrote reverse iteratively, write a second version that uses recursion (you may need a helper method). If you wrote it recursively, write it iteratively.

2 Arrays

Consider a method that inserts item into array arr at the given position. The method should return the resulting array. For example, if x = [5, 9, 14, 15], item = 6, and position = 2, then the method should return [5, 9, 6, 14, 15]. If position is past the end of the array, insert item at the end of the array.

Is it possible to write a version of this method that returns void and changes arr in place (i.e., destructively)?

Extra: Write the described method:

```
public static int[] insert(int[] arr, int item, int position) {
```
Consider a method that destructively reverses the items in `arr`. For example calling `reverse` on an array `[1, 2, 3]` should change the array to be `[3, 2, 1]`.

What is the fewest number of iteration steps you need? What is the fewest number of additional variables you need?

*Extra:* Write the method:

```java
public static void reverse(int[] arr) {
```

[2.3] Extra: Write a non-destructive method `replicate(int[] arr)` that replaces the number at index `i` with `arr[i]` copies of itself. For example, `replicate([3, 2, 1])` would return `[3, 3, 3, 2, 2, 1]`.

```java
public static int[] replicate(int[] arr) {
```