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);
    }
}

public void insert(int item, int position) {
    if (first == null || position == 0) {
        addFirst(item);
        return;
    }
    IntNode currentNode = first;
    while (position > 1 && currentNode.next != null) {
        position--;
        currentNode = currentNode.next;
    }
    IntNode newNode = new IntNode(item, currentNode.next);
    currentNode.next = newNode;
Add another method to the SLList class that reverses the elements. Do this using the existing IntNodes (you should not use new).

```java
public void reverse() {
    IntNode frontOfReversed = null;
    IntNode nextNodeToAdd = first;
    while (nextNodeToAdd != null) {
        IntNode remainderOfOriginal = nextNodeToAdd.next;
        nextNodeToAdd.next = frontOfReversed;
        frontOfReversed = nextNodeToAdd;
        nextNodeToAdd = remainderOfOriginal;
    }
    first = frontOfReversed;
}
```

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.

```java
public void reverse() {
    first = reverseRecursiveHelper(first);
}
private IntNode reverseRecursiveHelper(IntNode front) {
    if (front == null || front.next == null) {
        return front;
    } else {
        IntNode reversed = reverseRecursiveHelper(front.next);
        front.next.next = front;
        front.next = null;
        return reversed;
    }
}
```
2 Arrays

2.1 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 \([5, 9, 6, 14, 15]\).

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

No, because arrays have a fixed size, so to add an element, you need to create a new array.

Extra: Write the described method:

```java
public static int[] insert(int[] arr, int item, int position) {
    int[] result = new int[arr.length + 1];
    position = Math.min(arr.length, position);
    for (int i = 0; i < position; i++) {
        result[i] = arr[i];
    }
    result[position] = item;
    for (int i = position; i < arr.length; i++) {
        result[i + 1] = arr[i];
    }
    return result;
}
```

2.2 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?

Half the length of the array. You can swap the two paired indices at the same step. One additional variable as a temporary buffer during the swap; one index for the iteration. More may make your code neater.

Extra: Write the method:

```java
public static void reverse(int[] arr) {
    for (int i = 0; i < arr.length / 2; i++) {
        int j = arr.length - i - 1;
        int temp = arr[i];
        arr[i] = arr[j];
        arr[j] = temp;
    }
}
```
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) {
    int total = 0;
    for (int item : arr) {
        total += item;
    }
    int[] result = new int[total];
    int i = 0;
    for (int item : arr) {
        for (int counter = 0; counter < item; counter++) {
            result[i] = item;
            i++;
        }
    }
    return result;
}
```