Review Session Practice Midterm Problems:

For each call to the following recursive method, indicate what output is produced:

1. Recursive Tracing, 15 points. Consider the following method:

```
public void mystery(int n) {
   if (n < 0) {
      System.out.print("-");
      mystery(-n);
   } else if (n < 10) {
      System.out.println(n);
   } else {
      int two = n % 100;
      System.out.print(two / 10);
      System.out.print(two % 10);
      mystery(n / 100);
   }
 }
```

For each call below, indicate what output is produced:

Method Call	Output Produced
mystery(7);	________________________
mystery(825);	________________________
mystery(38947);	________________________
mystery(612305);	________________________
mystery(-12345678);	________________________

2. Recursive Programming, 15 points.

Write a recursive method called digitMatch that takes two nonnegative integers as parameters and that returns
the number of digits that match between them. Two digits match if they are equal and have the same relative
position starting from the end of the number (i.e., starting with the ones digit). In other words, the method
should compare the last digits of each number, the second-to-last digits of each number, the third-to-last digits
of each number, and so forth, counting how many pairs match.
For example, for the call digitMatch(1072503891, 62530841), the method would compare as follows:

```
1 0 7 2 5 0 3 8 9 1
    | | | | | | | |
    6 2 5 3 0 8 4 1
```

The method should return 4 in this case because 4 of these pairs match (2-2, 5-5, 8-8, and 1-1). Below are more examples:

Method Call	Value Returned
digitMatch(38, 34)	1
digitMatch(5, 5552)	0
digitMatch(892, 892)	3
digitMatch(298892, 7892)	3
digitMatch(380, 0)	1
digitMatch(123456, 654321)	0
digitMatch(1234567, 67)	2

Your method should throw an IllegalArgumentException if either of the two parameters is negative. You are not allowed to
construct any structured objects to solve this problem (no array, String, StringBuilder, ArrayList, etc)
and you may not use a while loop, for loop or do/while loop to solve this problem; you must use recursion.

3. Collections Programming, 15 points.
Write a method wordCounts that takes a List of lower case Strings, made up of only alphabetic characters,
and that returns a map that maps from every word in the provided list to the number of times it appears.
The keys of the map should appear in alphabetic order.

for exmaple, if a list called words stores these values:

```
["a", "long", "time", "ago", "in", "a", "galaxy", "far", "far", "away"]
```

then the call wordCounts(words) should return the following map:

```
{a=2, ago=1, away=1, far=2, galaxy=1, in=1, long=1, time=1}
```

4. Linked Lists, 15 points. Linked List Nodes 008
Fill in the "code" column in the following table providing a solution that will turn the "before" picture into the "after" picture by modifying links between the nodes shown. You are not allowed to change any existing node's data field value and you are not allowed to construct any new nodes, but you are allowed to declare and use variables of type ListNode (often called "temp" variables).

You are writing code for the ListNode class discussed in lecture:

```
public class ListNode {
    public int data;       // data stored in this node
    public ListNode next;  // link to next node in the list

    <constructors>
}
```

As in the lecture examples, all lists are terminated by null and the variables p and q have the value null when they do not point to anything.

before	after	code
p->[1]->[2] q->[3]	p->[2] q->[3]->[1]	
p->[1] q->[2]->[3]	p->[3] q->[1]->[2]	
p->[1]->[2] q->[3]->[4]	p->[3]->[2]->[1] q->[4]	
p->[1]->[2] q->[3]->[4]->[5]	p->[1]->[3]->[5] q->[2]->[4]	

```
-----------------------+-----------------------+----------------------------
```

5. Array Programming, 10 points. removeFront
Write a method removeFront that takes an integer n as a parameter and that removes the first n values from a list of integers. For example, if a variable called list stores this sequence of values:

```
[8, 17, 9, 24, 42, 3, 8]
```

and the following call is made:

```
list.removeFront(4);
```

then it should store the following values after the call:

```
[42, 3, 8]
```

Notice that the first four values in the list have been removed and the other values appear in the same order as in the original list.

You are writing a method for the ArrayIntList class discussed in lecture (handout 3):

```
public class ArrayIntList {
    private int[] elementData; // list of integers
    private int size;          // current # of elements in the list

    <methods>
}
```

You are not to call any other ArrayIntList methods to solve this problem.
Your method should throw an IllegalArgumentException if the parameter n is less than 0 or greater than the number of elements in the list.
Your solution must run in O(n) time.

6. Stacks/Queues, 25 points. rearrangeDuplicates
Write a method called rearrangeDuplicates that takes a sorted queue of integers as a parameter and
that moves any duplicate values to the end of the queue preserving the relative order of the values.
You should assume that the queue you are passed as a parameter has been sorted in nondecreasing order.
As a result, any duplicates will be grouped together. Suppose, for example, that a variable q stores these values:

```
front [-10, -5, -5, -5, 3, 3, 3, 18, 19, 23, 23, 23, 23, 405] back
```

This sequence has duplicates of -5, 3, and 23. If we make the call:

```
rearrangeDuplicates(q);
```

then the duplicates are moved to the end of the queue:

```
front [-10, -5, 3, 18, 19, 23, 405, -5, -5, 3, 3, 23, 23, 23] back
        ^                       ^    ^                    ^
        |                       |    |                    |
        +-----unique values-----+    +-----duplicates-----+
```

Notice that the new sequence contains the unique values from the original sequence followed by the duplicates.
Also notice that the two sublists are still in their original order.

You are to use one stack as auxiliary storage to solve this problem.
You may not use any other auxiliary data structures to solve this problem, although you can have as many simple variables as you like.
You also may not solve the problem recursively.
Your solution must run in O(n) time where n is the size of the queue.
Use the Stack and Queue structures described in the cheat sheet and obey the restrictions described there (recall that you can't use the peek method or a foreach loop or iterator).