Problem set 3

1. Printing the odd values in a list of integers (10pts)

1.1. Recursion

```
public static void printOddsRecursive(IntNode iFirstNode) {

   // base case -> Node is null
   if( iFirstNode == null)
      return;

   // if odd value, print it
   if( iFirstNode.val%2 == 1)
      System.out.println( iFirstNode.val );

   // recursive call
   printOddsRecursive(iFirstNode.next);
}
```

1.2. Iteration

```
public static void printOddsIterative(IntNode iFirstNode) {

   // init IntNode we will use to iterate
   IntNode control = iFirstNode;

   while(control != null){
      if(control.val%2 == 1)
         System.out.println(control.val );

      control = control.next;
   }
}
```

2. Improving the efficiency of an algorithm (15pts)

2.1. Worst case

Illustrate the worst case:
L1 = {1,1,1,1,1,1,… } - (1 everywhere)
L2 = {……..,1} - (no 1 before the last)

Assuming:

- the first list has a length of n.
- the second list has a length of m.

$$WorstCase(n,m) = n * (\sum_{i=1}^{m-1} i + m) + \sum_{i=1}^{n-1} i = \frac{n * (m(m+1) + (n-1))}{2}$$

red: n calls to second loop
green: getItem in second loop
blue: addItem in second loop
orange: get item in first loop

Therefore the worst case is **O(n³).**

2.2. Improved method

private static void intersect(LLList iList1**, LLList** iList2**){**

```
LLList inters = new LLList();

ListIterator it1 = iList1.iterator();
ListIterator it2 = iList2.iterator();

while( it1.hasNext() ){
   Object item1 = it1.next();
   while( it2.hasNext() ){
      Object item2 = it2.next();
      if(item2.equals(item1)){
         inters.addItem(item2, inters.length());
         break;
      }
   }
}
```

}

2.3. Improved worst case

We iterate over the first list and compare to all of the elements in the second list.
addItem() is only reached on time "per second" loop.
Therefore, the worst case is **O(n²)**

3. Removing a value from a doubly linked list (10pts)

public static DNode removeAllOccurences(DNode iFirstNode**, char** iChar**){**

```
   // base case
   if( iFirstNode == null)
   return null;

DNode head = removeAllOcurrences(iFirstNode.next, iChar);

if(iFirstNode.value == iChar){
   return head;
}
else{
   // update pointers and return new head
   if(head != null)
      head.prev = iFirstNode;
   iFirstNode.next = head;
   return iFirstNode;
}

}
```

4. Testing for palindromes using a stack (10 pts)

```
Public static bool isPalidrome(String iWord){

If(iWord == null)
   throw new IllegalArgumentException("input is null");

// length 1 or empty strings
if(iWords.length() <= 1)
   return true;

LLStack<Character> leftStack = new LLStack< Character >();

for(int i = 0; i <= iWord.length()/2; i++)
   leftStack.push( iWord.charAt(i) );

for(int i = iWord.length()/2 + iWord.length()%2; i< iWord.length(); i++){
   if( iWords.charAt(i) != leftStack.pop() )
      return false;
}

return true;
}
```