Written Problems

Problem Set 3
Graham Schmidt
schmidtg@gmail.com
Harvard ID: 30825489

1. Printing the odd values in a list of integers:

a.

```
public static void printOddsRecursive(IntNode in) {
        if (in == null) {
            return;
        }

        // Print only odd nodes
        if (in.val % 2 == 1) {
            System.out.println(in.val);
        }

        printOddsRecursive(in.next);
    }
```

b.

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

  IntNode trav = in;

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

      trav = trav.next;
  }

  return;
}
```

2.

a. Worst case is you have two exact lists and the intersection is essentially a copy of them together. The worst-case is approximately O(n^3) since you'll work in the outer loop O(m) times, where m is the length of the first list since you'll need to traverse the list each time getItem(i) is called. The inner loop will be worked in O(n^2) times, where n is the length of the second list, since you'll need to traverse the entire length of list2 each time getItem(j) is called as well as addItem(). Thus making a terrible time efficiency of O(n^3).

b.

```
public static LLList intersect (LLList list1, LLList list2) {
     LLList inters = new LLList();

     ListIterator i = list1.iterator();

     While (i.hasNext()) {
          Object item1 = i.next();

          ListIterator j = list2.iterator();

          while (j.hasNext()) {
               Object item2 = j.next();

               if (item2.equals(item1)) {
                    inters.addItem(item2, inters.length());
                    break;
               }
          }
     }
     return inters;
}
```

c. The improved algorithm has a worst-case running time of O(n^2). It only has an outer loop that will run a maximum O(m) times, where m is the length of list1, and an inner while loop that is O(n) times where n is the length of list2. The inner loop is O(n) because in the case where the lists are identical, the inner loop will have to traverse the length of list2 when addItem() is called because it has to add item2 to the end of list2. The inner loop in does not have to start its traversal for the next item though, because it is using an iterator as the point where it last left off.

3.

```
public static DNode removeAllOccurences(DNode first, char ch) {
        DNode trav = first;

        if (first == null)
            return null;

        // Loop through first character until fully removed
        while (first.ch == ch) {
            first = first.next;
            first.prev = null;
            trav = first;
        }

        // Continue until end of list removing 'ch'
        while (trav != null) {

            if (trav.ch == ch) {
                if (trav.prev != null)
                    trav.prev.next = trav.next;
                if (trav.next != null)
                    trav.next.prev = trav.prev;
            }
            trav = trav.next;
        }

        return first;
    }
```

4. Testing for palindromes using a stack

```
public static boolean isPalindrome(String str) {
        // Sanity check
        if (str.length() == 0)
            return false;

        // Create a stack
        Stack<Character> stack = new
ArrayStack<Character>(str.length() - 1);
        char[] str_arr = str.toCharArray();
        for (int i = 0; i < str_arr.length; i++) {
            // Add items to stack
            stack.push(str_arr[i]);
        }

        for (int i = 0; i < str_arr.length; i++) {
            if (stack.pop() == (Character)str_arr[i]) {
                return false;
            }
        }

        return true;
    }
```