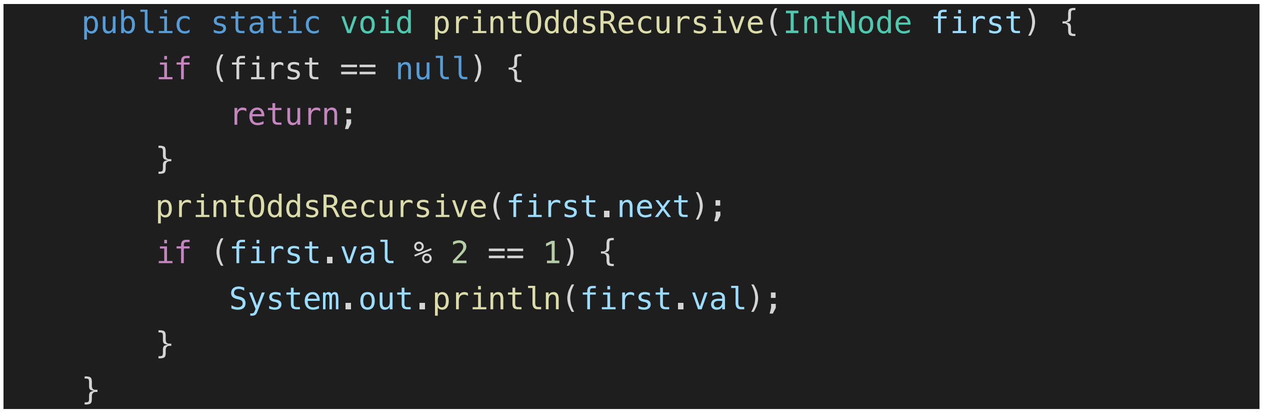
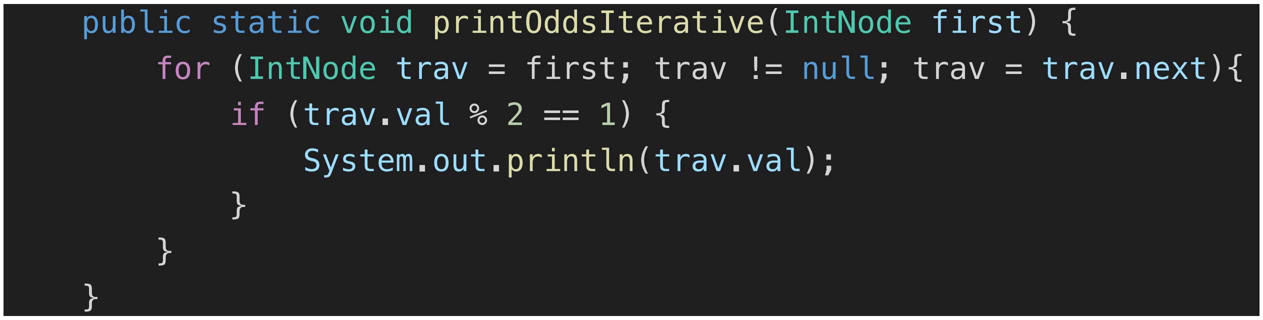
Problem Set 6, Part I

Problem 1: Printing the odd values in a linked list of integers
1-1)



```
    public static void printOddsRecursive(IntNode first) {
        if (first == null) {
            return;
        }
        printOddsRecursive(first.next);
        if (first.val % 2 == 1) {
            System.out.println(first.val);
        }
    }
```

1-2)



```
    public static void printOddsIterative(IntNode first) {
        for (IntNode trav = first; trav != null; trav = trav.next){
            if (trav.val % 2 == 1) {
                System.out.println(trav.val);
            }
        }
    }
```

Problem 2: Comparing two algorithms
2-1) *time efficiency of algorithm A:* O(n)
explanation: Let n be the length of aList. The outer for-loop has to iterates n times. It has two inner method, aList.getItem() and addItem(). The aList.getItem()'s time efficiency is O(1), and the LLList.addItem()'s time efficiency is O(1) because each element is added to the front, which is at position 0. Thus, the overall time efficiency is O(n).

2-2) *time efficiency of algorithm B:* O(n^2)
explanation: Let n be the length of aList. The outer for-loop has to iterates n times. It has two inner method, aList.getItem() and addItem(). The aList.getItem()'s time efficiency is O(1), and the LLList.addItem()'s time efficiency is O(n) because each element is added to the position i. Thus, the overall time efficiency is O(n^2).

2-3) Algorithm A is more efficient than Algorithm B because O(n^2) takes longer time than O(n)

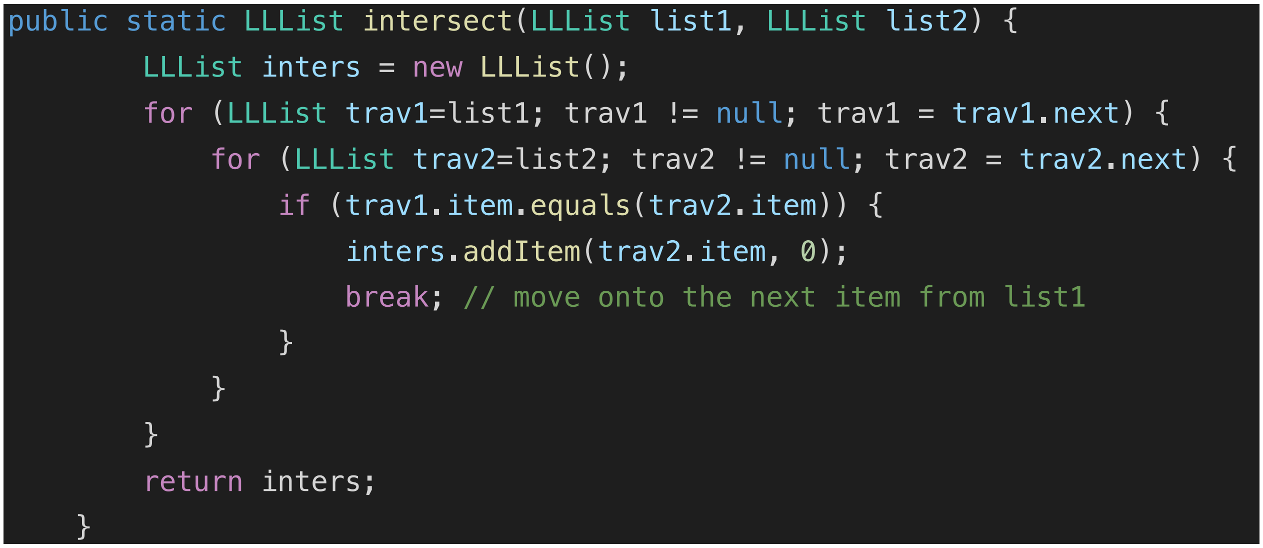
Problem 3: Choosing an appropriate representation
3-1) *ArrayList*
explanation: Since I need random access to the items in the list, I prefer ArrayList.

3-2) *LLList*
explanation: Since the number of tweets can vary widely, I prefer LLList which can grow to an arbitrary length. I can add the most recent tweets at the front to keep the time efficient for add method be O(1), which is also convenient for display the tweets in reverse chronological order: just print from front to back.

3-3) *ArrayList*
explanation: Considering the number of events is fairly consistent from month to month, I prefer the ArrayList because no extra memory needed for links.

Problem 4: Improving the efficiency of an algorithm
4-1) O(m*n^2)
The i-loop iterates m times, and list1.getItem()'s time efficiency is O(m).
The j-loop iterates n times, and list2.getItem()'s time efficiency is O(n).
The time efficiency of addItem is O(inters.length()).
Therefore, the overall time efficiency is
O(m*(m+n*(n + inters.length()))) = O(m*(m+n^2)) = O(m^2 + m*n^2) = O(m*n^2).

4-2)



```
public static LLList intersect(LLList list1, LLList list2) {
        LLList inters = new LLList();
        for (LLList trav1=list1; trav1 != null; trav1 = trav1.next) {
            for (LLList trav2=list2; trav2 != null; trav2 = trav2.next) {
                if (trav1.item.equals(trav2.item)) {
                    inters.addItem(trav2.item, 0);
                    break; // move onto the next item from list1
                }
            }
        }
        return inters;
    }
```

4-3) O(nm)
The outer for-loop iterates m times. The inner for-loop iterates n times. The addItem()'s time efficiency is O(1) because items are added to the front (afterall the sequence of items in inters is not required). Therefore, the overall time efficiency is O(nm)