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 null;
      }else{
            if(first.val /2 != 0){
                  System.out.println(first.val);
            }printOddRecurisve(first.next);
      }
}
```

1-2)

```
public static void printOddsIterative(IntNode first) {
      if(first == null){
            return null;
      }
      while(first != null){
            if(first.val /2 != 0){
                  System.out.println(fist.val);
            }
            First = first.next;
      }
}
```

Problem 2: Comparing two algorithms

2-1) *time efficiency of algorithm A:* O(n)
explanation: The loop executes total n times, so it makes the big O of n

2-2) *time efficiency of algorithm B:* O(n^2)
explanation: the loop execute total n times, but when it adds up in between the linked list, it also executes n times. So total n * n = n^2 runtime.

2-3) The first algorithm A is more efficient than the algorithm B

Problem 3: Choosing an appropriate representation

3-1) *ArrayList or LLList?* ArrayList
explanation: The Scenario rarely have adding or deleting from the list, so we do not need to think LLList would be a better option. Also, Arraylist has the benefit of random access which it can use for modifying product ID's. it also does not need extra memory for links.

3-2) *ArrayList or LLList? LLList*

explanation: Because the size of the array keeps changing by people keep tweeting and hence we can extend the list to an arbitrary length. It also traverse from the last node to the first node, so it can display the tweet from the recent to the last.

3-3) *ArrayList or LLList? LLList*
Explanation: LLList display the events in chronological order. Hence it doesn;t need to use Arraylist for random access, which requires more memory.

Problem 4: Improving the efficiency of an algorithm
4-1) O(m * n)
There are two loops for list 1 and list2 that runs each length times. So it makes the runtime Big O(m * n)

4-2)
```
public static LLList intersect(LLList list1, LLList list2) {
     LLList inter = new LLList();

     LLList l1 = list1.sort();
     LLList l2 = list2.sort();

     for(int i = 0; i < minLen; i ++){
          Object i = list1.getItem(i);
          Object j = list2.getItem(i);
     while(l1 != null && l2 != null){
          if(i.equals(j){
               inter.addItem(i);
          }else if(i.isLesser(j)){
               inter.addItem(i);
          }else{
               inter.addItem(j);
          }
     }return inter;
}
```

4-3) O(m log m + n log n)
It considering list 1 and list2 's length as m and n because sort() method has big o of n log n.