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;
      }
      if(first.val % 2 == 1) {
            System.out.println(first.val);
      }
      printOddsRecursive(first.next);
}
```

1-2)

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

Problem 2: Comparing two algorithms

2-1) *time efficiency of algorithm A:* O(n)
Explanation: The for loop crosses each position in the list exactly once, copying it from one list to the other. Therefore, however many items are in the list is the number of times it will take to complete the algorithm.

2-2) *time efficiency of algorithm B:* O(n^2)
Explanation: The for loop crosses each position in the list, thus it loops n times. Meanwhile, for each element, we execute the addItem function which is another n occurrences per item. Therefore, the algorithm efficiency is n*n which is n^2.

2-3)
Algorithm A is more efficient as it has a time efficiency of O(n), whereas the other algorithm requires O(n^2).

Problem 3: Choosing an appropriate representation

3-1) *ArrayList or LLList?*
explanation:

3-2) *ArrayList or LLList?*
explanation:

3-3) *ArrayList or LLList?*
explanation:

Problem 4: Improving the efficiency of an algorithm

4-1)

The for loop runs n times as it goes over each element in the list1. Within that for loop, there is another for loop which runs m times as it goes over the elements in list2. Therefore the run time is O(m*n) for this algorithm.

4-2)

```
public static LLList intersect(LLList list1, LLList list2) {
	LLList a = new LLList();
	list1.sort();
	list2.sort();
	Object item1 = list1.getItem();
	Object item2 = list2.getItem();
	while (list1.hasItem() && list2.hasItem()) {
		if (item1.equals(item2)) {
			a.addItem(item2);
		} else if (item1.isLesser(item2)) {
			item1 = list1.getItem();
		} else {
			item2 = list2.getItem();
		}
	}
	return a;
}
```

4-3)

We can implement a more efficient algorithm by utilizing a method similar to quicksort. This would create a run time of mlogm for list1 and nlogn for list 2. The while loop which iterates over n times creates the resulting run time to be mlogm + nlogn + n. Finally, this leaves us with an overall algorithm efficiency of O(mlogm + nlogn).