Assignment/Homework #2

COP-3530, Fall 2016

Rules & Instructions:

- Due date: Friday, September 30, 2016 at 5 p.m. (Eastern Time)
- This assignment has **3 problems**.
- The assignment/homework will be submitted **by email** to abajuelo@fiu.edu
- Your submission must be a ZIP file (not RAR format). **Please name your submission as 2_xxxxxxx.zip, where xxxxxxx is your seven digit Panther ID number)**.
- Please include the following header for each **Java source program**:

```
/*******************************************************************
  Purpose/Description: <a brief description of the program>
  Author's Panther ID: <your Panther ID number>
  Certification:
    I hereby certify that this work is my own and none of it is the work of
    any other person.
 ******************************************************************/
```

- Please indicate in the **subject of your email message** the following information:

COP-3530, SECTION U05, ASSIGNMENT #2

- Please make sure that you do not include any other personal information in your submission (besides the **Panther ID** in the name of the ZIP file and in the headers of your Java files as explained above). For example, no date of birth or name should be found in the document(s) you submit.
- Submissions turned in after the due date and/or which don't meet the established formatting rules will not be accepted.

Problem #1

Given **two sorted lists of comparable items**, L1 and L2. You can assume that in L1 and L2 all elements are different (no duplicates) but the interception between L1 and L2 may be non-empty.

(a) Implement an efficient method in **Java** to determinate if the interception of L1 and L2 is empty.

(b) Implement an efficient method in **Java** to compute the **symmetric difference** (∆) between L1 and L2, L1 ∆ L2. Please remember that in **set theory**, the symmetric difference of two sets A and B is the set of elements either in A or in B but not in both. Example: Suppose A = {1,3,5,7,9} and B = {1,2,3,4,5}, A ∆ B = {2,4,7,9}.

(c) What is the running time complexity of your methods? Justify.

Note: For this problem you can use only the **Java Collections API** basic methods (next(), hasNext(), compareTo(), and add()). The signatures of the methods are:

```
public static <AnyType extends Comparable<? super AnyType>>
        boolean disjoint(List<AnyType> L1, List<AnyType> L2)

public static <AnyType extends Comparable<? super AnyType>>
        void symDifference(List<AnyType> L1, List<AnyType> L2,
                                           List<AnyType> Difference)
```

Problem #2:

There are n items, numbered from 1 to n in a given array and given a positive integer number k (k < n). We start from item number 1 and delete from the array the item number k and so on until one item remains. The task is to determinate the initial position of this remaining item (the last survivor).

For example:

(1) if we have an array = [1, 2, 3, 4, 5, 6, 7], n = 7, and **k = 3**, then the numbers 3, 6, 2, 7, 5, 1 are deleted in order, and the number 4 survives.

(2) if we have an array = [1, 2, 3, 4, 5, 6, 7], n = 7, and **k = 4**, then the numbers 3, 6, 2, 7, 5, 1 are deleted in order, and the number 2 survives.

(a) Implement a **recursive method** in **Java**

void printLastSurvivor(int arr[], int n, int k)

that simulates this process of determination of the last survivor in a given array and prints "The last survivor is in the left-half of the array" if the position of the last survivor (in a given array) is less or equal than the integer part of the division of the length of the array by 2, or "The last survivor is in the right-half of the array", otherwise.

In our previous example (1), the method must prints "The last survivor is in the right-half of the array" and for the example (2), the method must prints "The last survivor is in the left-half of the array".

(b) What is the running time complexity of your function? Justify

Problem #3:

Given a **single linked list** of unknown length, L, and a nonnegative integer number k.

(a) Propose an **efficient algorithm** (**only pseudo-code**, not a Java program) that returns the value of the node at the position k from the end of L. For example, if the single linked list L is:

9 →11 → 6 → 4 → 3 →2 → 6 → 5 →11 →11;

and k = 7, the node at the position 7 from the end of L is the node with the value 4.

(b) What is the running time complexity of your algorithm? Justify.