**Problem #1: (34 points)**

(a) Given **two sorted lists**, **L1** and **L2**, complete a following **procedure in Java** to compute:

**L1 \ L2** = {x | x  L1 and x  L2} using **only the basic list operators** (next(), hasNext(), and compareTo()) **and one loop**.

public static <AnyType extends Comparable<? super AnyType>>

void difference(List<AnyType> L1, List<AnyType> L2, List<AnyType> Difference)

{

ListIterator<AnyType> iterL1 = L1.listIterator();

ListIterator<AnyType> iterL2 = L2.listIterator();

if ( iterL1.hasNext() && iterL2.hasNext() )

{

itemL1 = iterL1.next();

itemL2 = iterL2.next();

}

**// YOUR CODE GOES HERE**

**while iterL1 has next and iterL2 has next**

**if itemL1 compared to itemL2 is 1**

**add itemL1 to the Diffrence List**

**add itemL2 to the Diffrence List**

**print contents of Difference**

}

*Important Notes:*

• For the problem 1(a) you don’t need to submit any implementation in **Java**. Only the

**pseudocode** of your algorithm is required.

• Pseudocode is a simple way of writing programming code in English. It uses short phrases to write code for programs before you actually create it in a specific language.

• Example of **pseudocode**: Set total to zero

Set grade counter to one

While grade counter is less than or equal to ten

Input the next grade

Add the grade into the total

Set the class average to the total divided by ten

Print the class average.

• More information about pseudocode in:

<https://computersciencewiki.org/index.php/Pseudocode>

(b) **MyList** is a data structure consisting of a list of items, on which the following operations are possible:

**myPush(x)**: Insert item x on the front end of the MyList. **myPop()**: Remove the front item from the MyList and return it. **myInject(x)**: Insert item x on the rear end of the MyList.

Using the **LinkedList** class, write a class in Java to implement the **MyList** data structure and that take **O(1)** time per operation.

Note: The **MyList** class signature is:

public class MyList <AnyType> {

. . .

MyList() {. . .}

void myPush(AnyType x) {. . .} AnyType myPop() {. . .}

void myInject(AnyType x) {. . .}

}

*Important Notes:*

• For this problem, the implementation of the methods **myPush(x)**, **myPop()**, and

**myInject(x)**.

• It is not required that you implement the main method.

**Problem #2: (36 points)**

In this problem, you will write some **Java code** for simple operations on **binary search trees** where keys are integers. Assume you already have the following code and assume that the method bodies, even though not shown, are correct and implement the operations as we defined them in class.

public class BinarySearchTreeNode

{

public int key;

public BinarySearchTreeNode left;

public BinarySearchTreeNode right;

}

public class BinarySearchTree

{

private BinarySearchTreeNode root; public void insert(int key) { ... } public void delete(int key) { ... } public boolean find(int key) { ... }

}

(a) Add a method **public int positiveKeySum()** to the BinarySearchTree class that returns the sum of **all non-negative keys** in the tree. You will need an assistant/helper method.

(b) Add method **public void deleteMax()** to the BinarySearchTree class that deletes the maximum element in the tree (or does nothing if the tree has no elements).

(c) Add method **public void printTree()** to the BinarySearchTree class that iterates over the nodes to print then in increasing order. So the tree...

4

/ \

2 5

/ \

1 3

Produces the output "1 3 2 5 4".

Note: You will need an assistant/helper method.

*Important Notes:*

• For this problem, you only need to submit the implementation of four methods in Java

(**positiveKeySum**, **deleteMax**, **printTree**).

• It is not required that you implement the main method.

**Problem #3: (30 points)**

(a) Write a method in **Java** to **sort** a stack of n integer numbers, s, in ascending order.

**static Stack<Integer> sort(Stack<Integer> s)**

To implement this method, you must use **one more auxiliary stack** and you should not make any assumptions about how the stack is implemented. The following are the only functions that should be used to write this program: **push**, **pop**, **peek**, and **isEmpty**.

Example:

**Input:**

**. . . .**

Stack<Integer> s = **new** Stack<>();

s.push(9); s.push(6); s.push(8); s.push(5); s.push(4); s.push(11); s.push(4);

**Output:** (after sorting)

Stack (from top): 4,4,5,6,8,9,11

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

*Important Notes:*

• You must add the main method in your program to test your implementation.

• There are no data errors that need to be checked as all the data will be assumed correct.

• Your program MUST be submitted only in source code form (.java file).

• A program that does not compile or does not run loses all correctness points.