<u>Assignment #2</u>

COP-3530 (Regular), Summer 2019

Release: June 4, 2019 **Due date:** June 12, 2019

Purpose:

This assignment has **4 problems**. These problems relate to some of the important topics we studied in the **Modules 2, 3** and **4**.

The purpose of this assignment is:

- To analyze problems and to implement efficient solutions to problems using the LinkedList class of Java utility library.
- To analyze problems and to implement efficient solutions to operations involving Binary Search Trees, AVL-trees and binary heaps. In some cases, you will be using the recursive methods to solve in efficient way the problem.
- To apply basic techniques of algorithm analysis.

Submitting Your Assignment:

- Assignments must be turned in via Canvas.
- Please follow these steps for every assignment:
  1. You are allowed to upload only a single file. Since programs will consist of multiple files, your upload must be a compressed "ZIP" file (with extension **.ZIP**).

     No other kinds of compressed files will be accepted!
  2. Please name your submission as **2_XXXXXXX.ZIP**, where **XXXXXXX** is your seven digit **Panther ID number**.
  3. **Before creating the zip file**, make sure your output is correct and your program and output meet all of the published specifications.
  4. **<u>Before uploading the zip file</u>**, extract the individual files and examine them to make sure you have included all that is required. In your .ZIP you must include **ONLY** Java files (.java) and PDF files (generated using text editors like MS Word, WordPerfect, TextMaker, LaTex, etc). Photos and scanned images, as well as solutions in a different format that the ones stated above will be not accepted.
  5. Please include the following header for each Java 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.
 *************************************************************/
```

  6. Please make sure that you <u>do not</u> 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, <u>no date of birth or name should be found in the document(s) you submit</u>.
  7. Submissions turned in after the due date and/or which don't meet the established formatting rules will not be accepted.

Failure to follow these simple directions may result in a loss of credit for the assignment.

Problem #1: (10 pts)

A **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: (30 pts)

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 keySum()** to the BinarySearchTree class that returns the sum of all the keys in the tree. You will need an assistant/helper method.

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

Continue on next page

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

```
    4
   / \
  2   5
 / \
1   3
```

Produces the output "5 4 3 2 1".

Note: You will need an assistant/helper method.

d) Add method **public void printPreorder()** to the BinarySearchTree class that prints out the nodes of the tree according to a "preorder" traversal. So the tree...

```
    4
   / \
  2   5
 / \
1   3
```

Produces the output "4 2 1 3 5".

Note: You will need an assistant/helper method.

e) You have a **binary search tree**. Consider a leave *l*. B is the set of keys in the path *p* of *l* including the leave *l* and the root of the tree. A is the set of keys to the left of the path p. C is the set of keys to the right of the path *p*. Is the following statement true or false? Given <u>any</u> element *a* in A; *b* in B; *c* in C; a ≤ b ≤ c. *Justify your answer*.

Important Notes:

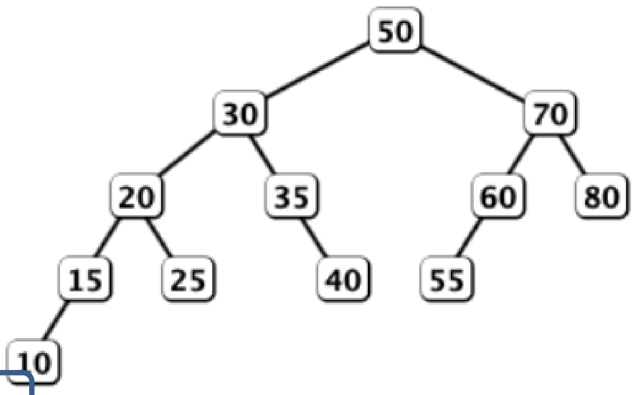
- For this problem, you only need to submit the implementation of four methods in Java (**keySum**, **deleteMax**, **printTree**, and **printPreorder**).
- In the part (e), you don't need to submit any implementation in **Java** and just you must answer the question.

Problem #3: (30 pts)

a) Show the **result of every step** of inserting the following sequence of elements into an initially empty **AVL-tree**: **14, 17, 11, 7, 53, 4, 13, 12, 8**

Continue on next page

b) Show the resulting **AVL-tree**, after physical deletion (NOT a lazy deletion) of the node with the key 80 from the following **AVL-tree**.



c) Show the AVL tree that results after each of the integer keys 9, 27, 50, 15, 2, 21, and 36 are inserted, in that order, into an initially empty AVL tree. Clearly show the tree that results after each insertion, and make clear any rotations that must be performed.

d) Draw an **AVL-tree** of height 4 that contains the minimum possible number of nodes.

e) The following items are inserted into an **AVL tree**: 1, 2, 3, 8, 6. How many and what type of rotations are performed? Justify.

Note: We assume that double rotations count as one rotation.

Important Notes:

- For problem #3, you don't need to submit any implementation in Java.
- For all parts of this problem, you must draw the AVL-trees using the appropriate graphics tools at your convenience. AVL-trees that were drawn by hand will be not accepted.

Problem #4: (30 pts)

(a) Show the result of inserting 10, 12, 1, 14, 6, 5, 8, 15, 3, 9, 7, 4, 11, 13, and 2, one at a time and in the giver order, into an initially empty **binary min heap**.

(b) Show the result of performing three **deleteMin** operations in the heap of the previous **binary min heap**.

Continue on next page

(c) Write a method **replaceKey** in the **MinHeap** class with the following signature:

public void replaceKey(Integer oldKey, Integer newKey)

The method will replace the first occurrence of **oldKey** with the **newKey**, and restore the **Min-Heap** property after the change. If the **oldKey** does not exist in the heap, the method prints an appropriate message and returns without changing the heap.

Example: Suppose our binary heap object (**bh**) has the following keys:

***	4	6	7	32	19	64	26	99	42	54	28

Then the method call: **bh.replaceKey (oldKey Integer(54), newKey Integer(2))** should change the keys to:

***	2	4	7	32	6	64	26	99	42	19	28

Note: You can assume that the methods **perlocateUp** and **perlocateDown** are already implemented in your **MinHeap** class.

Important Notes:

- For this problem, you have only one Java implementation in part (c). In this part you need to submit only the implementation of the **replaceKey** method.
- For parts (a), (b), and (e) of this problem, you must draw the min (or max) heaps using the appropriate graphics tools at your convenience. Heaps that were drawn by hand will be not accepted.

= = = The End = = =