Assignment/Homework #2

COP-3530, Summer 2021

Purpose:

This assignment has **2 main problems**. These problems relate to some of the important topics we studied in **Modules 2 and 3 (AVL-trees not included)**.

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 **ZIP** file and ***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. Inside **ZIP** folder, there should be a separate folder for each question (i.e. Problem 1, Problem 2, Problem 3, etc.)
  4. **For questions that require Java implementation:**
     - The **.java** file must be inside the corresponding problem folder. DO NOT MIX ANSWERS.
     - ONLY SUBMIT **.JAVA** FILES. DO NOT SUBMIT YOUR WHOLE PROJETC'S FOLDER!!!
     - If is required, each .java file should contain ITS OWN main method at the bottom of the file. Please add only one main method for EACH .java file.
  5. **For written questions:**
     - Submit these files INSIDE the specific problem folder.
     - Each answer MUST be identified. It should be easy to tell which question and subsection you are answering!
     - Written questions must be only in **PDF** format.
  6. 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.
 *************************************************************/
```

  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 #2: (35 pts)

Given an array A of n integers, a leader element of the array A is the element that appears more than half of the time in A.

Using **one stack**, implement in **Java** an **O(n)** running time complexity method

static int leader(int[] A)

to find a leader element and return the index (any) of the leader in A. The program must returns -1 if no leader element exists.

Examples:

```
int[] a = {23, 23, 67, 23, 67, 23, 45}; ====> leader(a) = 5

int[] a = {23, 24, 67, 23, 67, 23, 45}; ====> leader(a) = -1
```

Important Notes:

- You must add the main method in your program in order to test your implementation.
- There are no data errors that need to be checked as all the data will be assumed correct.
- You can use the array of the previous example to test your program, however, I suggest that you also use other input arrays to validate the correctness and efficiency of your solution.
- 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.

Problem #2: (65 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 deleteMin()** to the BinarySearchTree class that deletes the minimum 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 2 3 4 5".

Note: You will need an assistant/helper method.

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

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

Produces the output "1 3 2 5 4".

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**, **deleteMin**, **printTree**, and **printPostorder**).
- In the part e), you don't need to submit any implementation in Java and just you must answer the question.