CPSC 4100 HW #3 – Programming Assignment, Fall 2022

27 points + (bonus: 3 points)

Date assigned: Friday, 10/07/2022

Due: 11:59 pm, Friday, 10/14/2022

1. Problem

Rabbit Jerry is preparing for a C++ software development job interview. He was told that Binary Search Trees (BST) are one of the data structures which many interview problems are landed on. He wants to master BST and he starts to work on some BST problems.

He plans to create a balanced BST from a sorted list of *unique* integers, and then work on the created BST with a few problems.

A node on the BST is defined as follows:

```
struct Node {
  int data_;
  Node* left_;
  Node* right_;
  int size_;
  void* extra_; //only useful for the bonus problem. Ignore it for other problems.
};
```

In addition to three common data members including data/key, left child pointer and right child pointer, each BST node maintains an additional data member: size (the number of nodes on each subtree). When Jerry creates the balanced BST, the size on each node must be populated properly at the same time. You cannot use another tree traversal (e.g., DFS) to populate this data field on each node.

Jerry wants to work on the following problems, and the solutions to the problems will be implemented as a static public member function for a C++ class Prog2.

Node* createBalancedBst(int* A, int n);	Given a list of n *unique* integers stored in array A, sorted in increasing order, create a balanced BST. Return a Node pointer pointing to the root node of the resulting BST. (hint: use Divide-and-Conquer)
void zigzagLevelTraversal(Node* root);	Perform a zigzag level traversal on the given BST, starting from level 0 (left to right), 1 (right to left), 2 (left to right), 3 (right to left), … Print out the integers (aka, data_) in the order of being visited during this zigzag level traversal to stdout. (hint: you can use STL class templates)

	E.g., 10 / \ 5 18 / \ / \ 3 7 14 23 The order of integers is: 10, 18, 5, 3, 7, 14, 23 For the output to stdout, use exactly one space to separate each integer, no space after the last integer, and no leading space before the first integer.
int gap(Node* root, Node* p, Node* q);	Two pointers *p* and *q* each pointing to a node in a BST whose root node is specified by *root*. You can always assume that *p* and *q* are valid. Return the gap between the two nodes pointed by *p* and *q* respectively. The gap is measured by the number of edges between the two nodes on the BST. E.g., 10 / \ 5 18 / \ / \ 3 7 14 23 If p points to node (5) and q points to node (14), then the function returns 3.
long long pathSum(Node* root, Node* p, Node* q);	In a tree, a unique path exists between any pair of nodes. Given the BST specified by *root*, return the sum of integers on the nodes of the path from *p* to *q*, including the integers on node *p* and *q*. You can always assume that *p* and *q* are valid. E.g., 10 / \ 5 18 / \ / \ 3 7 14 23

	If p points to node (5) and q points to node (14), then the function returns 5+10+18+14 = 47.
int kthLargest(Node* root, int k);	Returns the k-th largest integer on the BST. E.g., 10 / \ 5 18 / \ / \ 3 7 14 23 If k = 5, then the function returns 7.
int bonus(Node* root);	This is a bonus problem, worth 3 points. All-or-nothing policy is applied here. This function counts and returns the total number of paths whose sum of integers on the path is a multiple of 3. A path must consist of at least one edge. The number can be too large (causing integer overflow) and thus the function instead returns the remainder of the number divided by 1,000,000, 007. E.g., 10 / \ 5 18 / \ / \ 3 7 14 23 Some example paths are: 5-7, 3-5-7, 10-18-14, 7-5-10-18-14, etc. The total number of such paths = 10

The header file hw3.h is provided below.

```
/*
 *  *File: hw3.h
 *
 */

#ifndef _HW3_H
#define _HW3_H
//#define _BONUS   //uncomment this line if you have solved the bonus problem
using namespace std;

struct Node {
  int data_;
  Node* left_;
  Node* right_;
  int size_;
  void* extra_; //only useful for the bonus problem. Ignore it for other problems.
};

class Prog3 {
 public:
  static Node* createBalancedBst(int* A, int n);
  static void zigzagLevelTraversal(Node* root);
  static int gap(Node* root, Node* p, Node* q);
  static long long pathSum(Node* root, Node* p, Node* q);
  static int kthLargest(Node* root, int k);
#ifdef _BONUS
  static int bonus(Node* root);
#endif
};

#endif
```

Do not modify C++ struct Node and the public static member functions in class Prog3. The only line in hw3.h you can change is:
#define _BONUS

You need to implement all the public static member functions in hw3.cpp.

2. Time Efficiency

For each problem, you need to analyze time efficiency of your algorithm. You need to provide: (1) recurrent relation if it's a recursive algorithm; and (2) Big-O notation.

Include time efficiency in README file.

3. Testing

You can write your own driver/client program to test your code. However, to maximize your concentration on the algorithm design, a list of files is provided to ease your testing:

- hw3.h
- client.cpp