Section #6

Ashley Taylor and Shreya Shankar (Based on handouts by previous CS106B instructors and TAs)

Trees

1. Size

Write a function named `size` that accepts a pointer to the root of a binary tree of integers. Your function should return the number of nodes in a tree. The size of an empty (nullptr) tree is defined to be 0.

Constraints: You must implement your function recursively and without using loops. Do not construct any new `BinaryTreeNode` objects in solving this problem (though you may create as many `BinaryTreeNode*` pointer variables as you like). Do not use any auxiliary data structures to solve this problem (no array, vector, stack, queue, string, etc). Your function should not modify the tree's state; the state of the tree should remain constant with respect to your function.

Assume that you are using the `BinaryTreeNode` structure as defined below:

```
struct BinaryTreeNode {
    int data;
    BinaryTreeNode* left;
    BinaryTreeNode* right;
}
```

2. Is Balanced

Write a function named `isBalanced` that accepts a pointer to the root of a binary tree of integers. Your function should return whether or not a binary tree is balanced. A tree is balanced if its left and right subtrees are also balanced trees whose heights differ by at most 1. The empty (nullptr) tree is balanced by definition.

Constraints: You must implement your function recursively and without using loops. Do not construct any new `BinaryTreeNode` objects in solving this problem (though you may create as many `BinaryTreeNode*` pointer variables as you like). Do not use any auxiliary data structures to solve this problem (no array, vector, stack, queue, string, etc). Your function should not modify the tree's state; the state of the tree should remain constant with respect to your function.

Assume that you are using the `BinaryTreeNode` structure as defined below:

```
struct BinaryTreeNode {
    int data;
    BinaryTreeNode* left;
    BinaryTreeNode* right;
}
```

3. Is Binary Search Tree?

Write a function named `isBST` that accepts a pointer to a BinaryTreeNode representing the root of a binary tree of integers. Your function should return whether or not a binary tree is arranged in valid binary search tree (BST) order. A BST is a tree in which every node N's left subtree is a BST that contains only values less than N's data, and its right subtree is a BST that contains only values greater than N's data. The empty (nullptr) tree is a BST by definition.

Constraints: Do not construct any new `BinaryTreeNode` objects in solving this problem (though you may create as many `BinaryTreeNode*` pointer variables as you like). Do not use any auxiliary data structures to solve this problem (no array, vector, stack, queue, string, etc). Your function should not modify the tree's state; the state of the tree should remain constant with respect to your function.

Assume that you are using the `BinaryTreeNode` structure as defined below:

```
struct BinaryTreeNode {
    int data;
    BinaryTreeNode* left;
    BinaryTreeNode* right;
}
```

4. Follows Min Property

Recall that in a min heap, each parent is smaller than both of its children. Write a function named `followsMinProperty` that checks whether each parent in a tree has a value less than its children's values.

Assume that you are using the `TreeNode` structure as defined below:

```
struct TreeNode {
    int data;
    TreeNode* left;
    TreeNode* right;
}
```

5. Heap Add/Remove

Recall the implementation of a priority queue using a vertically-ordered tree called a heap. Recall that the heap structure "bubbles" elements up and down as they are added and removed to maintain its vertical ordering.

Given the following string/priority pairs:

`a:68, b:77, c:40, d:70, e:9, f:34, g:94, h:16, i:47, j:22, l:77`

a) Write the final array representation of the binary heap that results when all of the above elements are enqueued (added in the given order) with the given priorities to an initially empty heap. This is a "min-heap", that is, priorities with lesser integer values are higher in the tree. Write your answer in the following format:

{`a:17, b:63, c:40`}

b) After adding all the elements, perform 2 dequeue operations (remove-min operations) on the heap. Write the final array representation of the heap that results after the two elements are removed, in the same format.