Practice Problems on BST that you can code and verify:

Simple Problems:
https://leetcode.com/problems/minimum-distance-between-bst-nodes/description/
https://www.hackerrank.com/challenges/is-binary-search-tree/problem
https://leetcode.com/problems/kth-smallest-element-in-a-bst/

Slightly harder:
https://leetcode.com/problems/serialize-and-deserialize-bst/
https://leetcode.com/problems/merge-bsts-to-create-single-bst/

Balanced BSTs: Some problems from an earlier tutorial

Problems on binary trees:

Q1. Write a function that, given a pointer to the root of a binary tree, reverses the left and right subtrees of every node in the tree. This means that for every node u in the tree, the left subtree of u will be the reverse of the right subtree of u in the old tree, and conversely. Your function should return a pointer to the root of the reversed tree.

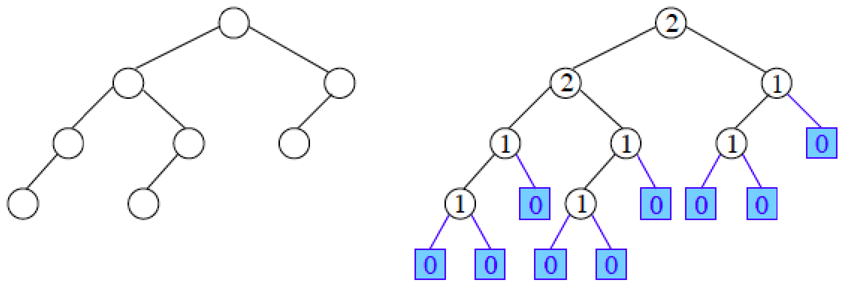
Solution:

```
typedef struct _tnode {
        int key;
        struct _tnode *L;
        struct _tnode *R;
} tnode;
typedef tnode *tree;


tree reverse ( tree T )
{
        tree L, R;
        if (T == NULL) return T;
        L = reverse(T -> R); R = reverse(T -> L);
        T -> L = L; T -> R = R;
        return T;
}
```

Q2.

. [*Extended binary tree*] Let T be a binary tree. We add special nodes to the tree so that each original node in T has exactly two children (and the special nodes have no children). A binary tree and its extended tree are shown in the following figure. The special nodes are shown as squares.



(a) A binary tree (b) The extended tree with s values

(a) Write a C function that takes a binary tree as input, and outputs its extended tree.

The s value of a node in the extended tree is defined as the shortest distance from a node to a special node. That is, $s(x) = 0$ if x is a special node. Otherwise, $s(x) = 1+\min(s(\text{left}(x)), s(\text{right}(x)))$. The s values of all nodes in the above example are shown in Part (b) of the above figure.

(b) Add a provision to store the s value in each node in a binary tree. Write a C function to compute (and store) the s values of all the nodes in an extended binary tree.

(c) Write a C function that computes and stores the s values of all the nodes in the original binary tree *without* creating the extended binary tree.

Solution:
Part (a).

```
tree extend ( tree T )
{
        treenode *p;
        if (T == NULL) return T;
        if (T -> L != NULL) extend(T -> L);
        else {
                p = malloc(sizeof(treenode));
                p -> L = p -> R = NULL;
                T -> L = p;
        }
        if (T -> R != NULL) extend(T -> R);
        else {
                p = malloc(sizeof(treenode));
                p -> L = p -> R = NULL;
                T -> R = p;
        }
        return T;
}
```

Part (c).

```
void computeSVals ( tree T )
{
        if (T == NULL) return;
        if (T -> L != NULL) computeSVals(T -> L);
        if (T -> R != NULL) computeSVals(T -> R);
        if ((T -> L == NULL) || (T -> R == NULL)) T -> s = 1;
        else T -> s = 1 + min(T -> L -> s, T -> R -> s);
}
```

More problems on balanced BSTs

Problem 1

You are given an array a containing n distinct integers. You are also given multiple integers x_i. For each x_i, you need to find number of indices i such that $a[i] < x$. You are allowed to do pre-computation of $O(n \log n)$, but for each i, you should be able to answer in $O(\log n)$. You are not allowed to use sorting.

Problem 2

Suppose that $\{(a_1, b_1), (a_2, b_2) \dots (a_n, b_n)\}$ is a set of n pairs of integers. Assume that all a_i and b_i values are distinct. Formally, for any i, j such that $1 \leq i, j \leq n$ and $i \neq j$, all a_i, b_i, a_j and b_j are distinct. You need to create a data structure to support search, insert and deletion in $O(\log n)$ time. Each search or deletion can be with respect the first component (a_i) or second component (b_i). i.e. Given an integer x, you need to find whether there exists a pair (a_i, b_i) with $a_i = x$ or $b_i = x$. For deletion, you delete the pair (a_i, b_i) with $a_i = x$ or $b_i = x$. Likewise for the second component.

Problem 3

Suppose that $\{(a_1, b_1), (a_2, b_2) \dots (a_n, b_n)\}$ is a set of n pairs of integers. Assume that all a_i values are distinct and so are all the b_i values. For each (a_i, b_i), you need to find number of indices j such that $a_j < a_i$ and $b_j < b_i$. Your algorithm should run in $O(n \log n)$ overall.

Solutions

Problem 1

We create a balanced BST from the given integers. At each node, we also maintain the size of the sub-tree rooted at this node in the BST. Now, given any given x, we start from the root and search for x. We maintain a variable *count* initialized to 0. Whenever, we follow the right child of a node, we add the size of sub-tree of the left child of the node to *count*. We also add 1 more for the current node itself. After the search is finished, *count* has the answer.

Problem 2

We maintain two balanced BSTs. Both the BSTs will contain the n pairs given initially. One of them uses a_i as the key while the other one uses b_i. For search with x, we can search in both the BSTs for key x. For delete we do the same except once we find a pair (x, y) or (y, x). We delete the pair from both the BSTs.

Problem 3

First we sort the given pairs with respect to a_i. Now for any i, for all j such that $j < i, a[j] < a[i]$. So, for each i, we now need to find number of j such that $j < i$ and $b[j] < b[i]$.

We iterate through the array from index 1 to n. We also maintain a balanced BST. Whenever we are at index i, the BST contains all b_j such that $j < i$. So, at index i, we just need to know how many elements in the BST are less than b_j. This can be done using the solution for Problem 1.