Group members:

- Titas Geryba tg1404
- Shunya Watanabe sw3212
- Christine Dah-In Chung cic266

Problem 2

Here is the Binary Search Tree that was reconstructed:

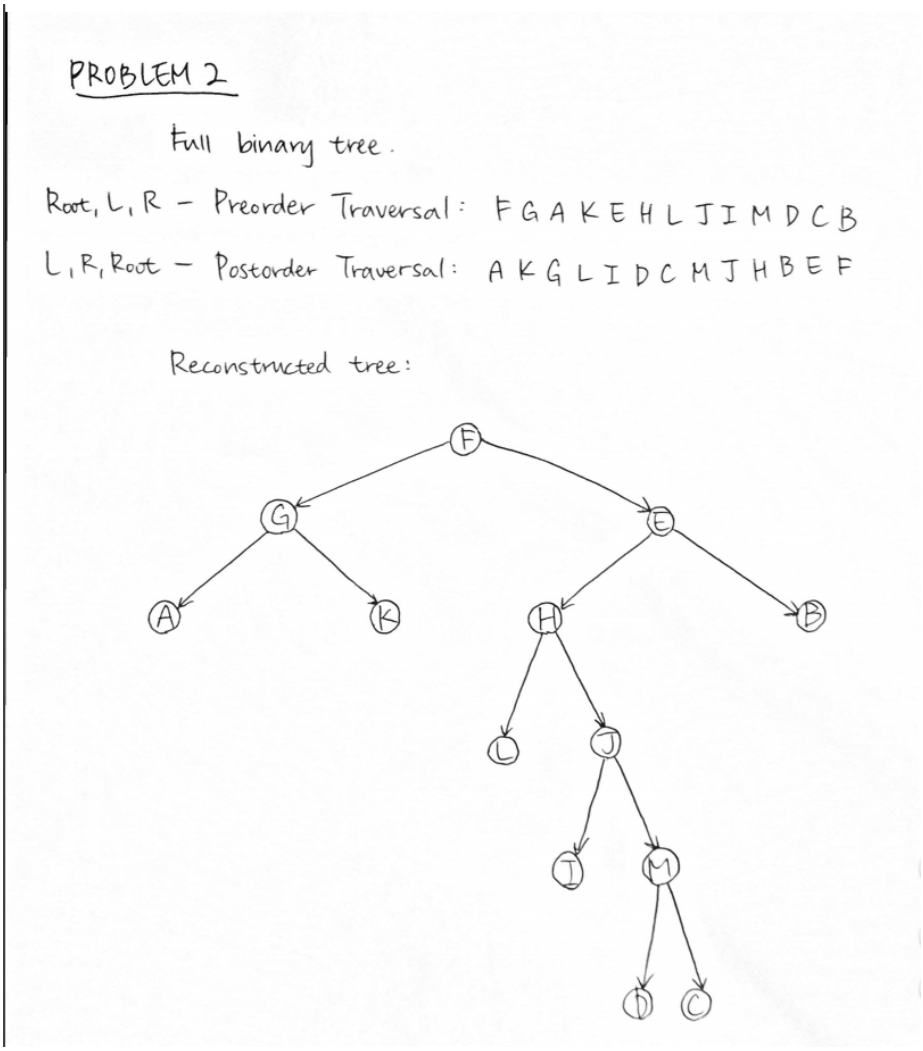

Figure 1: Reconstructed Binary Search Tree

Explanation of the reconstruction

The preorder traversal moves from the root to the left subtree and then to the right subtree. The postorder traversal moves from the left subtree to the right subtree and then to the root.

1. Since the first letter in the preorder traversal and the last letter in the postorder traversal is 'F', 'F' must be the root of the tree.
2. In the preorder traversal, 'G' follows 'F', so 'G' is the left child of 'F'.
3. In the postorder traversal, 'A' is the first letter, so 'A' is the leftmost node of the tree. This means that 'A' is the left child of 'G', and it is also a leaf.
4. In the postorder traversal, 'K' is in between 'A' and 'G'. Since it is a full binary tree, 'K' is a sibling of 'A' and the right child of 'G'.
5. In the preorder traversal, 'E' comes right after 'K'. This means that 'E' is either the left child of 'K', or the right child of 'F'. Since 'E' comes right before 'F' in the postorder traversal, it must be the right child of 'F'.
6. In the preorder traversal, 'H' comes after 'E', so it is the left child of 'E'.
7. In the postorder traversal, 'B' is in between 'H' and 'E', and 'B' is the last letter in the preorder traversal. This means that 'B' is the right child of 'E', and it is a leaf.
8. In the preorder traversal, 'L' is the next letter after 'H', so it is the left child of 'H'. It is also the leftmost child of the subtree - and thus a leaf - because it is the next letter after 'G' in the postorder traversal.
9. 'H' must have a right child because we are reconstructing a full binary tree, and since 'J' follows right after the leaf, 'L', 'J' is the right child of 'H'.
10. At this point, 'J' is the only node that can have any more children. In the preorder traversal, 'I' follows 'J', so 'I' is the left child of 'J'. Also, 'I' follows 'L' in the postorder traversal, so 'I' must be a leaf.
11. 'I' must have a sibling, and 'M' follows 'I' in the preorder traversal, so 'M' is the right child of 'J'.
12. In the preorder traversal, 'D' comes after 'M', and 'C' follows. In the postorder traversal, 'C' comes before 'M', and 'D' comes before 'C'. Thus, 'D' is the left child of 'M', and 'C' is the right child of 'M'.

Problem 3

To do this problem we will use a recursive function that takes a pointer and two values. The function declaration and the way it is called is shown below:

```
//declaration
bool isBST(Node *node, int max, int min) {...}
//call assuming that root is not null:
isBST(root, root->value - 1, root->value + 1);
```

The algorithm

1. First we check if the node pointer passed to the function is not null. If it is we return true.
2. If the node is not Null, then we check whether the node value is between the min and max values that were passed to the function.
   - If no return false.
   - Otherwise call the recursive function with the two children of the node.
3. Call the function with the left and right child of the node.
   - for the left child min stays the same, but the max now turns into the value of the parent node.
   - for the right child max stays the same, but the min now turns into the value of the parent node.
4. If all of the nodes below the root return true, then we have a binary search tree, otherwise we know that the tree is not a binary search tree.

Possible implementation in c++:

```
bool isBST(Node *node, int min, int max) {
    if (node == null) return true;
    if  (  node->value < max
        && node->value > min
        && isBST(node->left, min, node->value)
        && isBST(node->right, node->value, max)
        )  return true;
    else
           return false;
    }
}
```