Slot **2** – Question **3 [BST - Makeroot]** Marks 30

Gist: Create a binary search tree from the given inputs and make a certain element the root.

Tasks:

- Helper functions for insertnode, leftrotate, rightrotate, a function makeroot that calls the other functions to make a certain node the root are given
- Students have to flesh out the helper functions
- If they write completely different set of functions (or write one function inside another or main()), then they can be given marks based on the functionalities above

---

Grading Scheme: MANUAL

ANY FORM OF HARD-CODING ATTRACTS FULL PENALTY.

[1 Mark] : Creating the node properly

[2 Marks] : Inserting the node properly to create the BST

[3 Marks] : Correct implementation of the rotateleft

[3 Marks]: Correct implementation of the rotateright

[4 Marks]: Correct implementation of the makeroot that recursively traverses the tree and calls the two functions above and acts accordingly. The makeroot also recursively acts on all the nodes from the node to the root to make the node the root.

[2 Marks]: Correct implementation of preorder traversal

========= GOLD CODE ==========

```
// To makeroot a node in a BST
// Assume given value is guaranteed to be in the

#include <stdio.h>
#include <stdlib.h>

struct tree_node{
   struct tree_node* left;
   struct tree_node* right;
   int val;
```

```
};

typedef struct tree_node* node;

node makeNode(int value){
    node root;
    root = (node)malloc(sizeof(struct tree_node));
    root->left = NULL;
    root->right = NULL;
    root->val = value;
    return root;
}

/* A function to insert a new node with given value in BST */
node insertNode(node nde, int value)
{
    if (nde == NULL) return makeNode(value);
    if (value < nde->val)
        nde->left  = insertNode(nde->left, value);
    else
        nde->right = insertNode(nde->right, value);
    return nde;
}


// A utility function to right
// rotate subtree rooted with y
node rightRotate(node y)
{
    node x = y->left;
    node T2 = x->right;

    // Perform rotation
    x->right = y;
    y->left = T2;

    // Return new root
    return x;
```

```
}


// A utility function to left
// rotate subtree rooted with x
node leftRotate(node x)
{
    node y = x->right;
    node T2 = y->left;

    // Perform rotation
    y->left = x;
    x->right = T2;

    // Return new root
    return y;
}


node makeRoot(node root, int value){
    if(root==NULL)
        return root;
    if(value < root->val){
        root->left = makeRoot(root->left, value);
        root = rightRotate(root);
    }
    else if(value > root->val){
        root->right = makeRoot(root->right, value);
        root = leftRotate(root);
    }
    return root;
}

void preOrder(node root){
    if (root == NULL)
        return;
    printf("%d ", root->val);
    preOrder(root->left);
```

```
    preOrder(root->right);
}



int main(){
    int i, n, m, val;
    scanf("%d %d", &n, &m);
    node root = NULL;
    for (i=0; i<n; i++){
        scanf("%d", &val);
        root = insertNode(root, val);
    }
    preOrder(root);
    for (i=0; i<m; i++){
        printf("\n");
        scanf("%d", &val);
        root = makeRoot(root, val);
        preOrder(root);
    }
}
```