Slot **1** – Question **3 [BST - Rotation]** Marks 30

Gist: Create a binary search tree from the given inputs and rotate certain elements.

Tasks:

- Helper functions for insertnode, leftrotate, rightrotate, a function rotatenode that calls the other functions for rotation 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 rotatenode that recursively traverses the tree and calls the two functions above and acts accordingly. Also checks if rotation is possible or not.

[2 Marks]: Correct implementation of preorder traversal

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

```
#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 rotateNode(node root, int value, char rot){
    if(root==NULL)
        return root;
    if(value < root->val)
        root->left = rotateNode(root->left, value, rot);
    else if(value > root->val)
        root->right = rotateNode(root->right, value, rot);
    else{
        if(rot=='R'){
            if(root->left == NULL)
                printf("Rotation not possible around %d\n", value);
            else
                root = rightRotate(root);
        }
        else if(rot == 'L'){
            if(root->right == NULL)
                printf("Rotation not possible around %d\n", value);
            else
                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;
        char rot;
        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 %c", &val, &rot);
                root = rotateNode(root, val, rot);
                preOrder(root);
        }
}
```