Name and ID: **Solution** Section: **BCS-3G** Marks: 15

Question#01 [6 Marks]

a. Write a recursive function that takes a Binary Search Trees (BST) root pointer and return height of the given tree. [4+1]

   i. int HeightOfTheTree(const BTNode *tree)

   Height of a Binary Tree: The height for a null tree is 0, which the height of a binary tree is the distance from root to the farthest leaf node.

```
int HeightOfTheTree(BTNode* root) {
// Base case: empty tree has height 0
if (root == 0)
return 0;
// recur for left and right subtree and consider farthest
//each call add one to the height
return 1 + max(HeightOfTheTree(root->left), HeightOfTheTree(root->right));
}
```

   ii. What will be the time complexity of this operation? O(n)

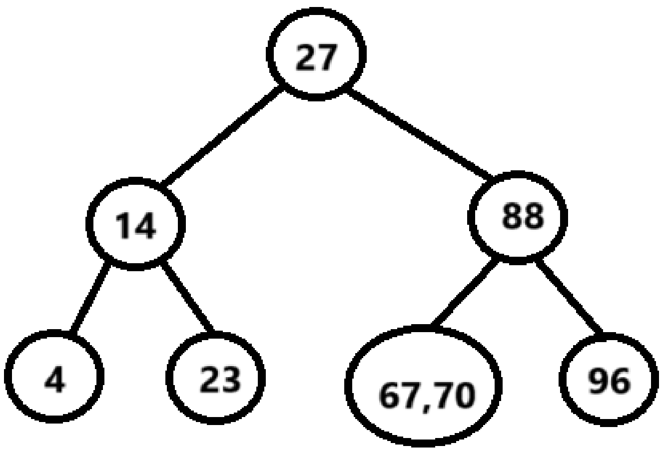
b. What are the postfix forms of the expression? [1]

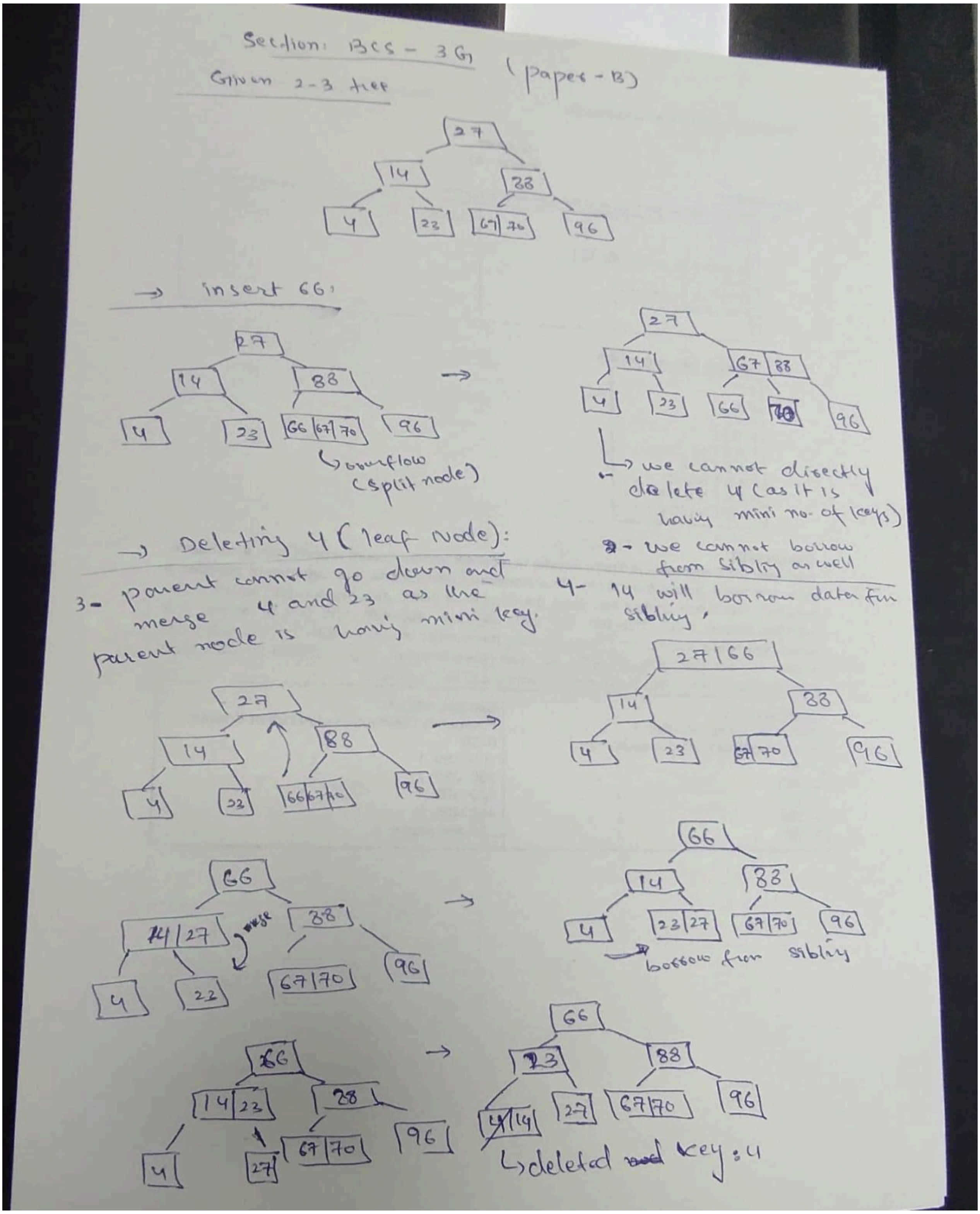
   A+B*(C-D)/(P-R)

   Postfix form: ABCD-*PR-/+

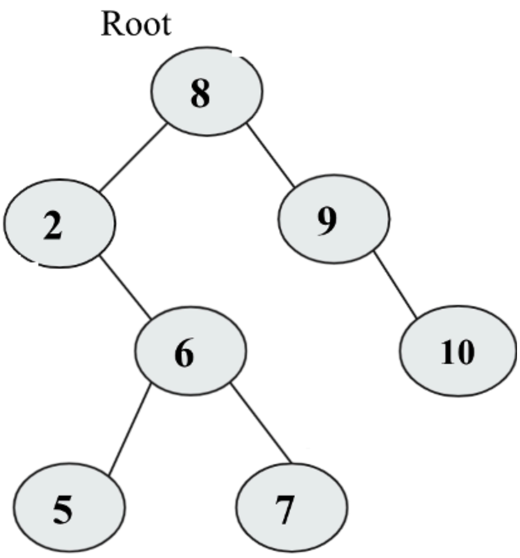
Question#02 [4+5 Marks]

a. Given the 2-3 tree below, insert 66 and delete 4. Show each step of the process clearly and the final resulting tree by drawing each step clearly.

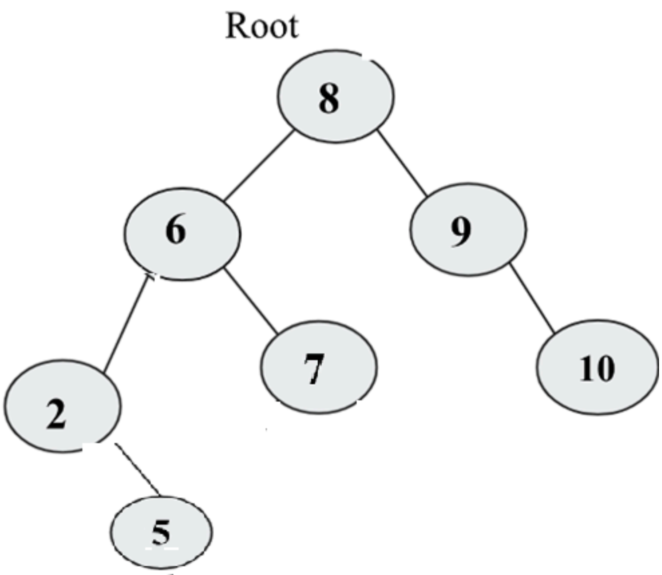




a. Find which node is imbalance in the following tree and which AVL rotation is used to balance the node. Show rotation dry run and write C++ function for that rotation case.



Rotation case: RR imbalance (left rotate)



left rotation

```
Node* RR_rotation(Node* node) {
    Node* child = node->right_node;
    node->right_node = child->left_node;
    child->left_node = node;

    node->height = max(get_height(node->left_node), get_height(node->right_node)) + 1;
    child->height = max(get_height(child->left_node), get_height(child->right_node)) + 1;

    return child;
}
```