Count smallers elements on right side

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

struct node
{
   int data, height, size;
   struct node *left, *right;
};

int height(struct node *root)
{
   return !root ? 0: root->height;
}

int size(struct node *root)
{
   return !root ? 0: root->size;
}

int max(int a, int b)
{
   return (a > b)? a : b;
}

struct node* newNode(int data)
{
   struct node *temp = (struct node*)malloc(sizeof(struct node));
   temp->data = data;
   temp->left = temp->right = NULL;
   temp->height = temp->size = 1;
   return temp;
}

struct node *rightRotate(struct node *root)
{
   struct node *temp1 = root->left;
   struct node *temp2 = temp1->right;
   temp2 = root;
   root->left = temp2;

   root->height = max(height(root->left), height(root->right)) + 1;
   temp1->height = max(height(temp1->left), height(temp1->right))+1;

   root->size = size(root->left) + size(root->right) + 1;
   temp1->size = size(temp1->left) + size(temp1->right) + 1;
   return temp1;
}

struct node *leftRotate(struct node *root)
{
   struct node *temp1 = root->right;
   struct node *temp2 = temp1->left;

   temp1->left = root;
   root->right = temp2;

   root->height = max(height(root->left), height(root->right)) + 1;
   temp1->height = max(height(temp1->left), height(temp1->right)) + 1;

   root->size = size(root->left) + size(root->right) + 1;
   temp1->size = size(temp1->left) + size(temp1->right) + 1;

   return temp1;
}

int getBalance(struct node *root)
{
   return !root ? 0: height(root->left) - height(root->right);
}

struct node* insert(struct node *root, int data, int *count)
{
   if (!root)
       return newNode(data);

   if (data < root->data)
       root->left  = insert(root->left, data, count);
   else
   {
       root->right = insert(root->right, data, count);
       *count += size(root->left) + 1;
   }

   root->height = max(height(root->left), height(root->right)) + 1;
   root->size   = size(root->left) + size(root->right) + 1;
   int balance = getBalance(root);
   if(balance > 1 && data < root->left->data)
       return rightRotate(root);
   if(balance < -1 && data > root->right->data)
       return leftRotate(root);
   if(balance > 1 && data > root->left->data)
   {
       root->left =  leftRotate(root->left);
       return rightRotate(root);
   }
   if(balance < -1 && data < root->right->data)
   {
       root->right = rightRotate(root->right);
       return leftRotate(root);
   }
   return root;
}
```

```
void countSmallerArray (int *arr, int *smaller, int size)
{
    int index;
    struct node *root = NULL;
    for(index = 0; index < size; index++)
        smaller[index] = 0;
    for(index = size - 1; index >= 0; index--)
        root = insert(root, arr[index], &smaller[index]);
}

void printArray(int *arr, int size)
{
    for(int index = 0; index < size; index++)
        printf("%d\t", arr[index]);
}

int main()
{
    int *arr, size, *lower;
    printf("Enter size of the array\n");
    scanf("%d", &size);

    //allocate memory
    arr = (int *)malloc(sizeof(int) * size);
    lower = (int *)malloc(sizeof(int) * size);

    printf("Enter elements in array\n");
    for(int index = 0; index < size; index++)
        scanf("%d", &arr[index]);
    countSmallerArray(arr, lower, size);
    printArray(lower, size);
    return 0;
}
```

Time complexity: O(nlogn)
Space complexity: O(n)