Program 2: Write a program using AVL trees to count number of smaller elements on the right of each element in ~~the~~ an array. Given an unsorted array arr[] of distinct integers, construct another array count_smaller[i] ~~such~~ such that countsmaller[i] contains ~~number~~ count of smaller elements on right side of each element arr[i] in array. [LAB TEST-1 ADS].

Adith Pai
USN:-1BM18CS005

```
#include<stdio.h>
#include <stdlib.h>
struct node
{
  int key;
  struct node *left;
  struct node *right;
  int height;
  int size;
}

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

int size (struct node*N)
{
  if (N==NULL)
    return 0;
  return N→size;
}

~~int max(~~ int height (struct node*N)
{ if(N==NULL)
    return 0;
  return ~~height;~~ N→height;
}
```

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

int getbalance(struct node*N)
 {
   if(N == NULL).
      return0;
   return height(N -> left) - height (N -> right);
 }

struct node* insert(struct node*, int key, int *count)
{
  if(node == NULL)
      return(newNode(key));
  if(key < node -> key)
        node -> left = insert(node->left, key, count);
  else
  {
    node -> right = insert(node -> right, key, count);
    *count = *count + size(node -> left) + 1;
  }
    node -> height = max(height(node -> left), height(node -> right
                                                              +1;

   node -> size = size(node -> left) + size(node -> right) + 1;
```

```
             // left left case
int balance = getbalance (node);
if (balance > 1 && key < node → left → key)
 return ~~left~~ rightRotate (node);
}
      // right right case
  if (balance < -1 && key < node → right → key)
 {
   node → right = rightRotate (node → right);
   return ~~node~~; leftRotate (node);
}

    // left right ~~case~~ case
     if (balance > 1 && key > node → left ·)
    { ~~return right ta right Rotate~~ node → left - leftRotate (node → left);
           return rightRotate (node);
     }
  if (balance < -1 && key < node → right → kay)
 { node → right = rightRotate (node → right);
   return leftRotate (node);
 }

    ~~to~~ return node;

}
```

```
void constructLowerArray (int arr[], int count_smaller [], int n)
{
 int i, j;
 struct node *root = NULL;
  for (i=0; i<n; i++)
       countSmaller[i]=0;
   for (i=n-1; i>=0; i--)
   { root = insert (root, arr[i], &countSmaller[i]);
   }
}
```