Some Important Functions Related To Binary Search Tree

Dr. Rahul Das Gupta

```
typedef struct bs_tree
{
   int data;
   struct bs_tree *left, *right;
}BS_TREE;

int height (BS_TREE  *t)
{
    if(t = =NULL)
          return 0;
    else
          return (1+max(height(t->left), height(t->right)));
}

int max(int x, int y)
{
   return  ( (x>y)? x:y);
}

void mirror_image (BS_TREE **aar)
{
    BS_TREE *t;
     if(*aar !=NULL)
    {
        mirror_image(&(*aar)->left);
        mirror_image(&(*aar)->right);
```

```
        t=(*aar)->left;
        (*aar)->left=(*aar)->right;
        (*aar)->right=t;
    }
}

int total_nodes (BS_TREE *t)
{
    if(t = = NULL)
        return 0;
    else
        return (total_nodes(t->left)+total_nodes(t->right)+1);
}

int leaf_nodes (BS_TREE *t)
{
    if(t = = NULL)
        return 0;
    else if (t->left = = NULL && t->right = = NULL)
        return 1;
    else
        return (leaf_nodes(t->left) + leaf_nodes(t->right));
}

int internal_nodes (BS_TREE *t)
{
    if((t = = NULL) || (t->left==NULL && t->right==NULL))
        return 0;
    else
```

```
  return (internal_nodes(t->left)+internal_nodes(t->right)+1);
}


BS_TREE * find_smallest(BS_TREE *t)
{
  if ((t==NULL) || t->left==NULL)
     return t;
  else
     return find_smallest(t->left);
}

BS_TREE * find_largest(BS_TREE *t)
{
  if ((t = = NULL) || t->right = = NULL)
     return t;
  else
     return find_ largest (t->right);
}

void remove_tree_memory (BS_TREE **aar)
{
    if(*aar != NULL)
    {
       remove_tree_memory (&(*aar)->left);
       remove_tree_memory (&(*aar)->right);
       free (*aar);
    }
}
```