HW

9.26

```
int Search_Bin_Recursive(SSTable ST, int key, int low, int high){ // 二分查找递归算法
    int mid;
    if(low > high) return -1; // 查找失败
    mid = (low + high) / 2; // 取中间位置
    if(key == ST.elem[mid].key) return mid; // 查找成功
    else if(key < ST.elem[mid].key) return Search_Bin_Recursive(ST, key, low, mid - 1); // 在前半区继续查找
    else return Search_Bin_Recursive(ST, key, mid + 1, high); // 在后半区继续查找
}
```

9.31

```
int Is_BSTree(Bitree T){ //判断二叉树 T 是否二叉排序树,是则返回 1,否则返回 0
    if(T==NULL) return 1;
    if(T->lchild!=NULL && T->lchild->data>T->data) return 0;
    if(T->rchild!=NULL && T->rchild->data<T->data) return 0;
    return Is_BSTree(T->lchild) && Is_BSTree(T->rchild);
}
```

9.34

```
void Delete_NLT(BiTree &T,int x){ //删除二叉排序树 T 中所有不小于 x 元素结点,并释放空间
    if(T==NULL) return;
    if(T->data<x){
        Delete_NLT(T->rchild,x);
    }
    else{
        BiTree p=T;
        T=T->lchild;
        free(p);
        Delete_NLT(T,x);
    }
}
```

9.38

```
void BSTree_Merge(BiTree &T,BiTree &S){ //把二叉排序树 S 合并到 T 中
    if(T==NULL) T=S;
    else if(S!=NULL){
        BSTree_Merge(T->lchild,S);
        BSTree_Merge(T->rchild,S);
        S=NULL;
    }
}
```

```
void Insert_Node(Bitree &T,BTNode *S){//把树结点 S 插入到 T 的合适位置上
    if(T==NULL) T=S;
    else if(S->data<T->data) Insert_Node(T->lchild,S);
    else Insert_Node(T->rchild,S);
}
```

9.40

```
typedef struct{
    int data;
    int bf;
    int lsize;
    BlcNode *lchild,*rchild;
}BlcNode,*BlcTree;

BTNode *Locate_BlcTree(BlcTree T,int k){ //在含 lsize 域的平衡二叉排序树 T 中确定第 k 小的结点指针
    if(T==NULL) return NULL;
    if(k==T->lsize+1) return T;
    if(k<=T->lsize) return Locate_BlcTree(T->lchild,k);
    else return Locate_BlcTree(T->rchild,k-T->lsize-1);
}
```