05. 写出折半查找的递归算法。初始调用时，low 为 1，high 为 ST.length。

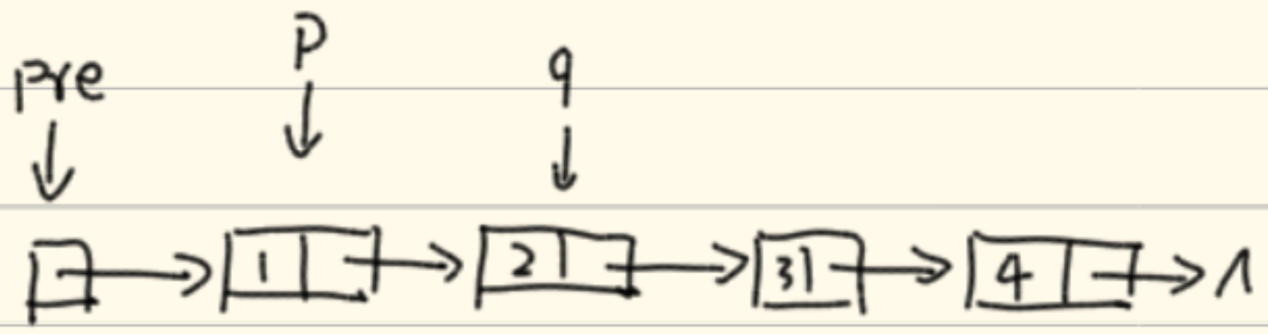
```
typedef struct SSTable{
        int *elem;
        int length;
} SSTable;
int BiSearch(SSTable T. int x. int low. int high){
    int mid;
    if(low > high) return 0;
    mid = low + (high-low)/2 ;
    if(T.elem[mid] == x) return mid;
    if(T.elem[mid] > x ) return BiSearch(T.x, low. mid-1);
    return BiSearch(T. x. mid+1, high);
}.
```

06. 线性表中各结点的检索概率不等时，可用如下策略提高顺序检索的效率：若找到指定的结点，则将该结点和其前驱结点（若存在）交换，使得经常被检索的结点尽量位于表的前端。试设计在顺序结构和链式结构的线性表上实现上述策略的顺序检索算法。

1	3	2	4	7	6	5	8

key = 7.

```
int SeqSearch(int arr[], int key){
    int i=0;
    while(i<n && arr[i]!=key) i++;
    if(i>0 && i<n){   //找到了且不在第1个
      int temp = arr[i-1];
      arr[i-1] = arr[i];
      arr[i] = temp;
      return --i;
    }.
    return -1;
}.
```



key = 2.

```
LNode * ListSearch(LinkList L, int key){.
    LNode *pre=NULL, *p=L., *q;
    while(p->next && p->next->data != key){
      pre = p;
      p = p->next;
    }.
    if(Pre == NULL) return p;      //找第一个
    if(P->next == NULL) return NULL;  //没找到.
    q = p->next;
    Pre->next=q; P->next = q->next; q->next =P;
    return q; }
```

06. 试编写一个算法，判断给定的二叉树是否是二叉排序树。

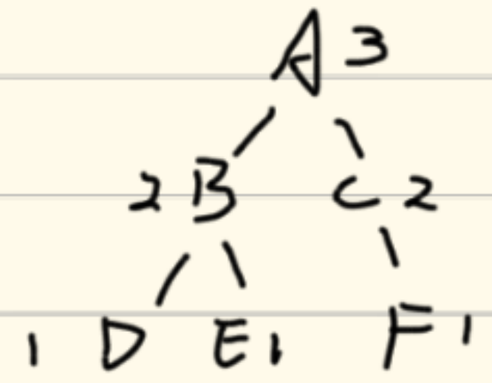
二叉排序树对的中序遍历为一个递增序列.

```
int pre = -INF;
bool judge(BiTree t){
    if(!t) return true;
    bool L,r;
    L = judge(t->left);
    if( !L || pre >= t->data) return false;
    pre = t->data;
    r= judge(t->right);
    return r;
}.
```

07. 设计一个算法，求出指定结点在给定二叉排序树中的层次。

```
int level(BiTree t, TNode *x){
    int ans = 0;
    TNode *P = t;
    while(P){
      ans++;
      if(P==x) return ans;
      if(x->data > P->data) P = P->rchild;
        else P = P->lchild;
    }.
    return -1;    //没找到.
}.
```

08. 利用二叉树遍历的思想编写一个判断二叉树是否是平衡二叉树的算法。



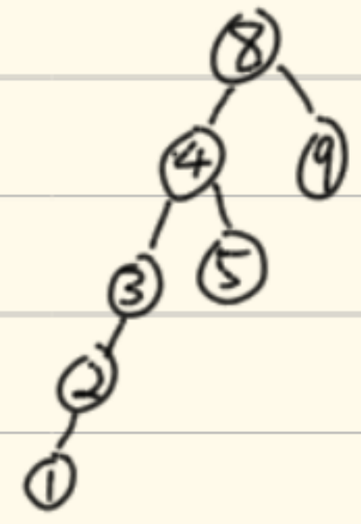

```
bool judge(BiTree t, int& h){
    if(!t){
        h=0;
        return true;
    }.
    int h1=0, h2=0;
    if(judge(t->lchild, h1) && judge(t->rchild, h2)){
        int diff = h1-h2;
        if(abs(diff)<=1){
            h= 1+(h1>h2? h1:h2);
            return true;
        }.
    }.
    return false;
}.
```

09. 设计一个算法，求出给定二叉排序树中最小和最大的关键字。

```
int findMax(BiTree t){
    while(t->rchild) t=t->rchild;
    return t->data;
}
int findMin(BiTree t){
    while(t->lchild) t=t->lchild;
    return t->data;
}
```

10. 设计一个算法，从大到小输出二叉排序树中所有值不小于 k 的关键字。

```
void OutPut(BiTree t, int k){          //右根左
    if(!t) return;
    OutPut(t->right, k);
    if(t->data >= k)  cout << t->data ;
    OutPut(t->left, k);
}
```



11. 编写一个递归算法，在一棵有 n 个结点的、随机建立起来的二叉排序树上查找第 k（$1 \le k \le n$）小的元素，并返回指向该结点的指针。要求算法的平均时间复杂度为 $O(\log_2 n)$。二叉排序树的每个结点中除 data, lchild, rchild 等数据成员外，增加一个 count 成员，保存以该结点为根的子树上的结点个数。

分析：若 t->lchild 为空

① 若 t->rchild 非空且 k=1，那 t 就是第 1 小.

② 若 t->rchild 非空且 k>1，那进右子树找 k-1 小.

若 t->lchild 不空.

① 若 t->lchild->count+1 = k，那 t 就是第 k 小.

② 若 t->lchild->count+1 > k，那进入左子树找第 k 小.

③ 若 t->lchild->count+1 < k，那进入右子树找第 k-(t->lchild->count+1) 小

count=3.

```
TNode* SearchKmin(BiTree t, int k){
    if(k<1 || !t) return NULL;
    if(!t->lchild){
        if(t->rchild && k==1) return t;
        if(t->rchild && k>1) return SearchKmin(t->rchild, k-1);
    }.
    else {
        if(t->lchild->count+1==k) return t;
        if(t->lchild->count+1>k) return SearchKmin(t->lchild, k);
        if(t->lchild->count+1<k)
            return SearchKmin(t->rchild, k-t->lchild->count+1);
    }.
}.
```