

Lecture 31: 交换排序

31.1 冒泡排序

31.1.1 基础概念

概念

- 思想:
 - 从后往前 (或从前往后) 两两比较相邻元素的值, 若为逆序 (即 $A[i-1] > A[i]$) 则交换它们, 直到序列比较完。称这样过程为“一趟”冒泡排序;
 - 从最后一个元素开始第一趟冒泡, 每次待比较的位置往左边移动一个, 和前面元素比一下大小, 小的往前去, 大的往后;
- 实现:
 - 第一趟排序使关键字值最小的一个元素“冒”到最前面;
 - 第二趟排序, 一直排序到第一趟结束的元素前一个位置 -> 前边已经确定最终位置的元素不用再对比;
- 稳定性:
 - 只有 $A[j-1] > A[j]$ 时才交换, 因此算法是稳定的;
- 性能:
 - 空间复杂度:
 - $O(1)$
 - 时间复杂度:
 - 有序时: $O(n-1) = O(n)$
 - 逆序时: $O(\frac{n(n-1)}{2}) = O(n^2)$
- 适用性:
 - 可以适用于链表;
 - 可从前往后“冒泡”, 每一趟将更大的元素“冒”到链尾;

31.1.2 算法实现

代码实现

- 概念:
 - i 所指位置之前的元素都已“有序”;
- 交换:

```
//交换(每次交换都要移动3次)
void swap(int &a, int &b){
    int temp = a;
    a = b;
    b = temp;
}
```

- 冒泡:

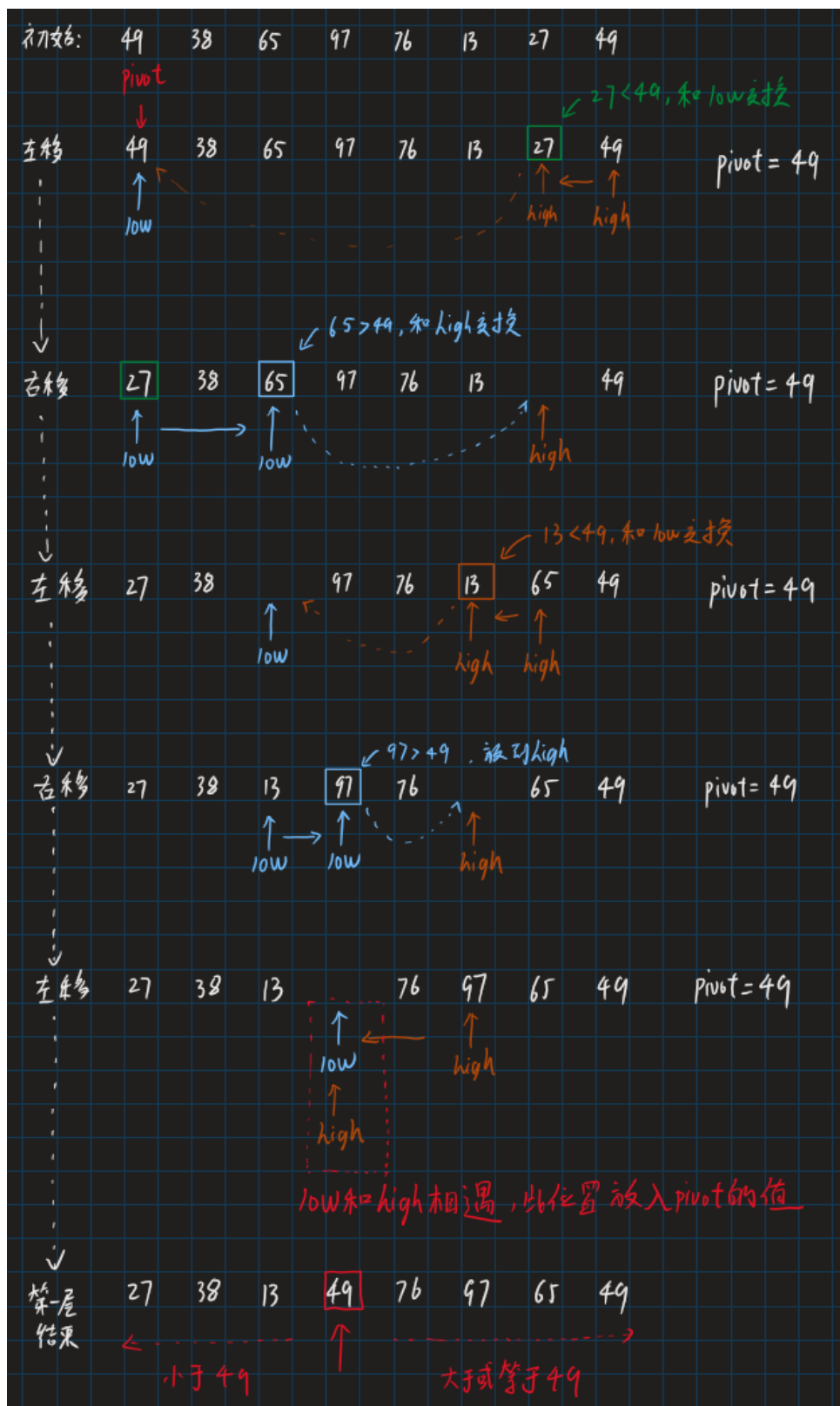
```
//冒泡排序
void BubbleSort(int A[],int n){
    for(int i=0;i<n-1;i++){
        bool flag=false;           //表示本趟冒泡是否发生交换的标志
        for(int j=n-1;j>i;j--)      //一趟冒泡过程
            if(A[j-1]>A[j]){        //若为逆序
                swap(A[j-1],A[j]); //交换
                flag=true;
            }
        if(flag==false)
            return;                //本趟遍历后没有发生交换,说明表已经有序
    }
}
```

31.2 快速排序

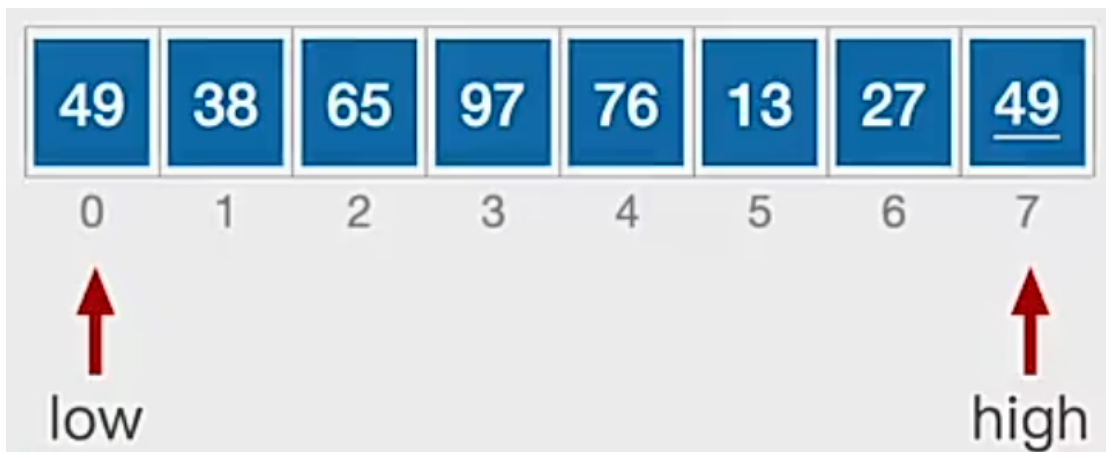
31.2.1 基础概念

概念

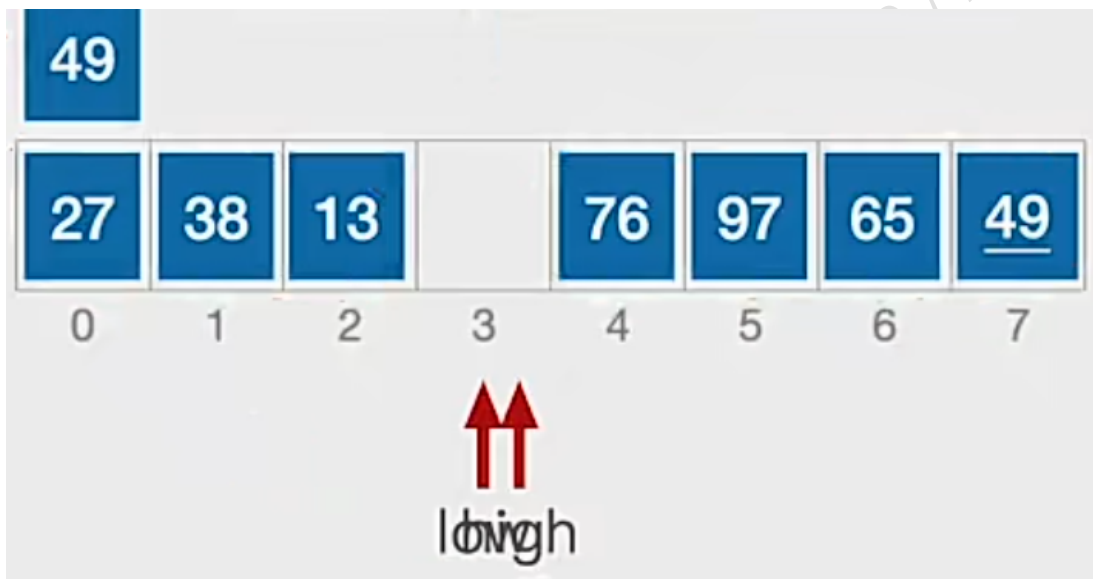
- 思想:
 - 在待排序表 $L[1.. N]$ 中任取一个元素 $pivot$ 作为枢轴 (或基准, 通常取首元素), 通过一趟排序将待排序表划分为独立的两部分: $L[1... K-1]$ 和 $L[k+1... N]$;
 - 使得
 - $L[1.. K-1]$ 中的所有元素小于 $pivot$;
 - $L[k+1... N]$ 中的所有元素大于等于 $pivot$;
 - 则 $pivot$ 放在了其最终位置 $L(k)$ 上, 这个过程称为一次“划分”;
 - 然后分别递归地对两个子表重复上述过程, 直至每部分内只有一个元素或空为止, 即所有元素放在了其最终位置上;
- 两个指针: low 和 $high$
 - $high$: 往左移动, 每次移动前需要判断当前 $high$ 指向的位置是否大于 $pivot$, 如果大于、则将 $high$ 往左移动, 如果 $high$ 小于 $pivot$, 则将 $high$ 位置的值得放到 low 的位置上;
 - low : 往右移动, 每次每次移动前需要判断当前 low 指向的位置是否小于 $pivot$, 如果小于、则将 low 往右移动, 如果 low 大于 $pivot$, 则将 low 位置的值得放到 $high$ 的位置上;
- 分析: 一趟快排过程



- 举例: 完整递归过程
 - 分成 low 和 high



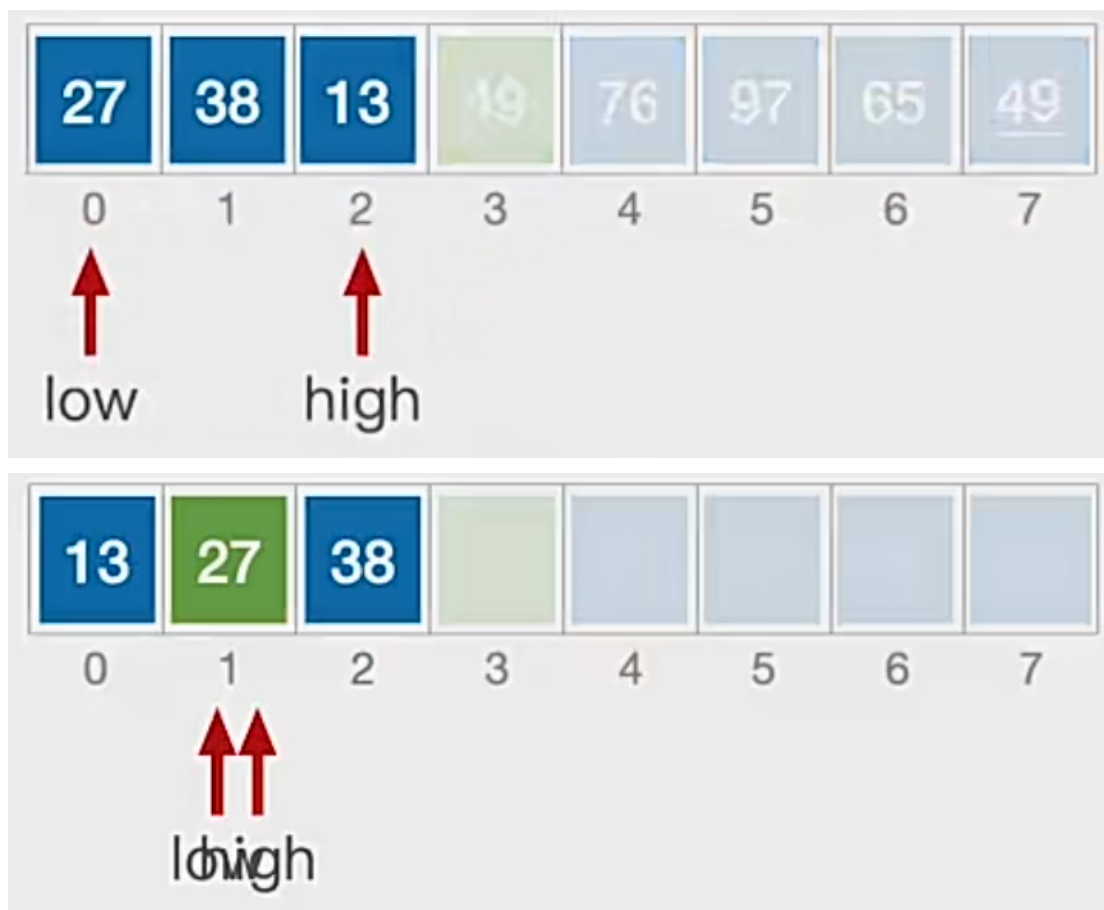
- 选择 low 作为基准元素：更小的元素都交换到左边，更大的元素都交换到右边；
 - 即：用第一个元素把待排序序列“划分”为两个部分。左边更小，右边更大。该元素的最终位置已确定；



- 基准元素位置：



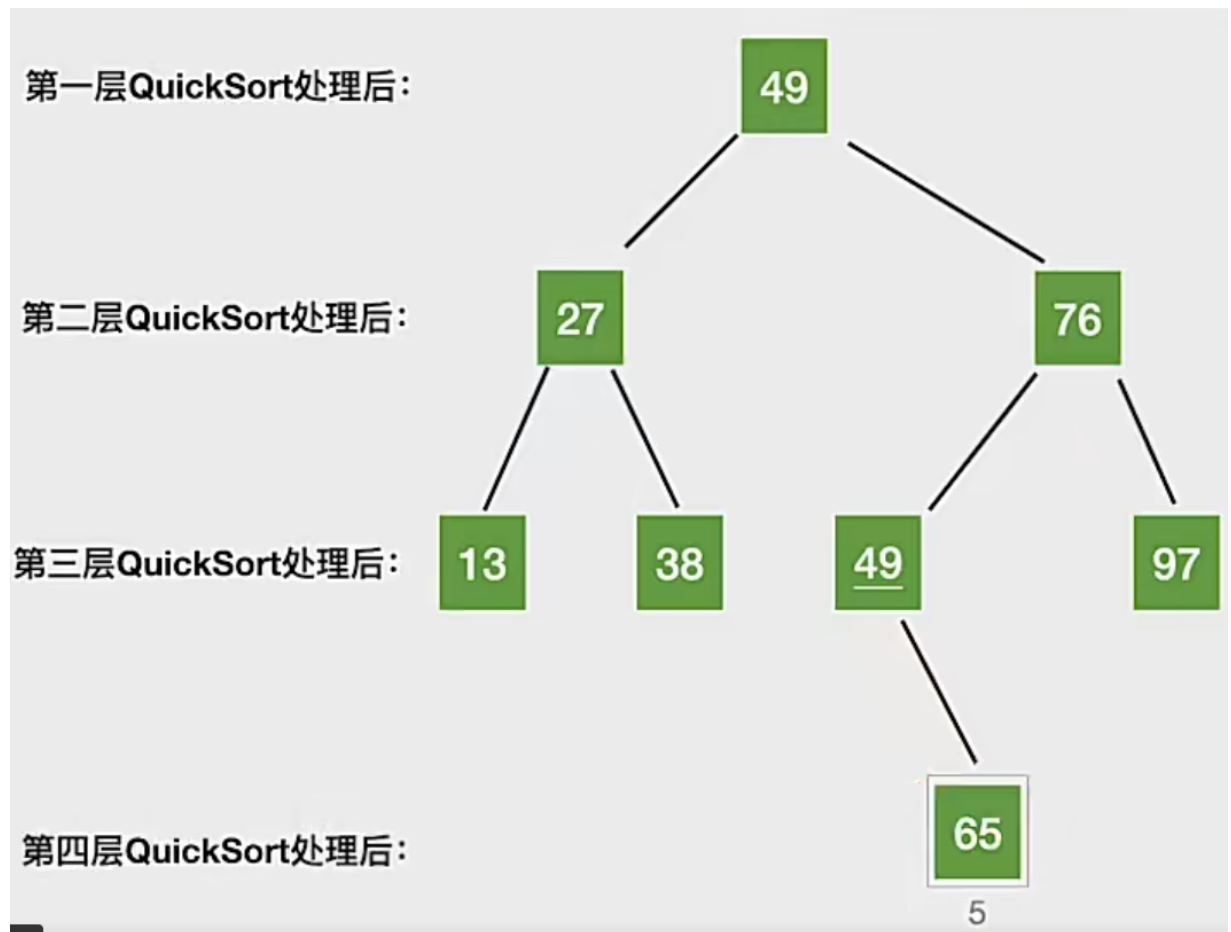
- 此时：49 元素的位置肯定正确的
- 接下来可以分别对左右子表进行递归，指向上述过程：



算法效率分析

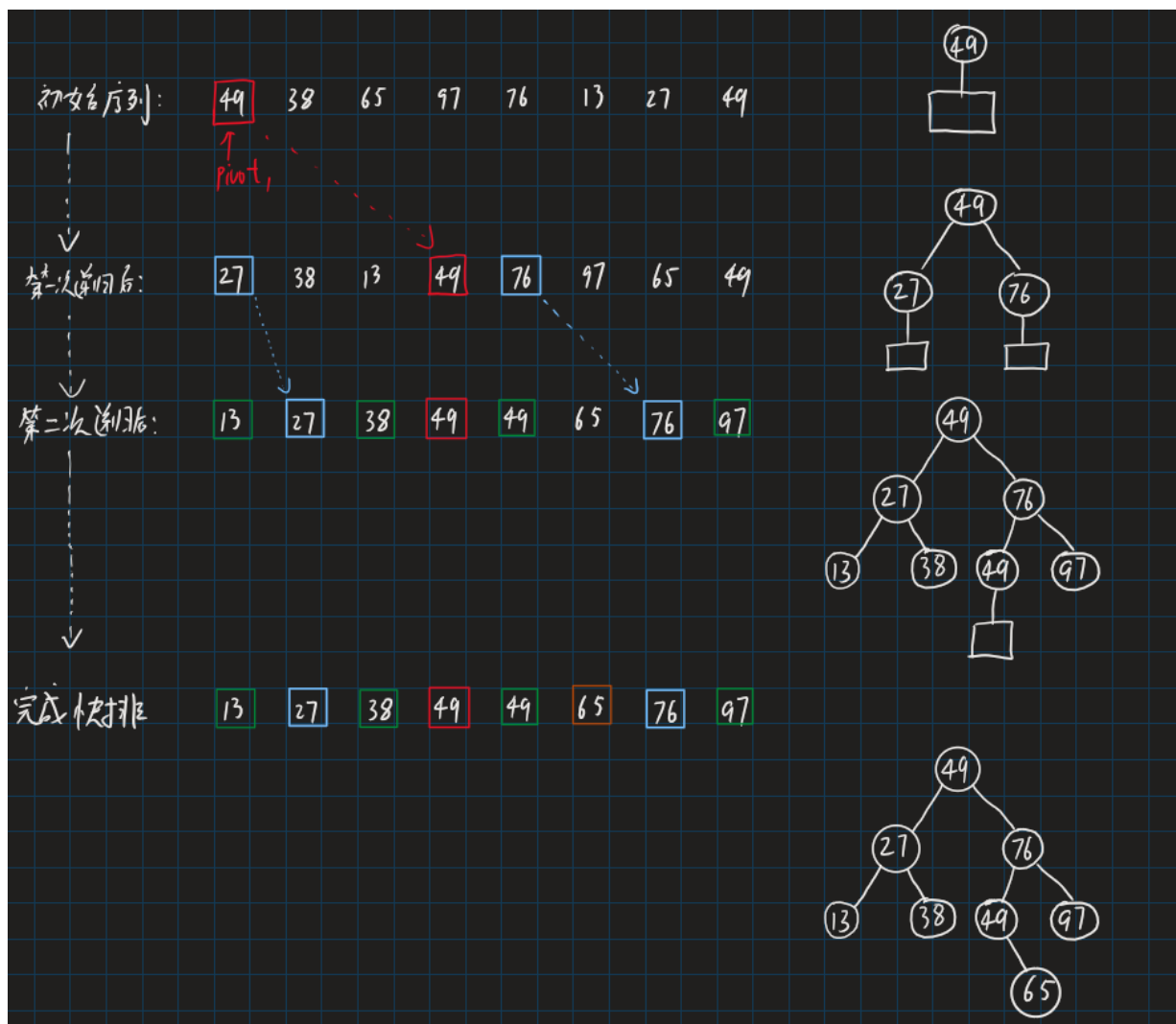
- 时间复杂度：
 - 时间复杂度为 $O(n * \text{递归层数}) = O()$
 - n 个结点的二叉树：
 - 最小高度 $= \log_2 n + 1$
 - 最大高度 $= n$
 - 最好时间复杂度： $O(n \log_2 n)$
 - 最差时间复杂度： $(O(n^2))$
- 空间复杂度：
 - 空间复杂度也和递归层数有关： $O(\text{递归层数})$
 - 最好空间复杂度： $O(n \log_2 n)$
 - 最差空间复杂度： $(O(n^2))$
- 稳定性：
 - **不稳定**；
- 分析：
 - 若每一次选中的“枢轴”将待排序序列划分为均匀的两个部分，则递归深度最小，算法效率最高；
 - 快速排序算法优化思路：尽量选择可以把数据中分的枢轴元素；
 - 举例：1. 选头、中、尾三个位置的元素，取中间值作为枢轴元素；2. 随机选一个元素作为枢轴元素；
- 结论：
 - 快速排序是所有内部排序算法中**平均性能最优的排序算法**；

- 图示：
 - QS 的递归树



- 总结:

获取笔记持续更新中



31.2.2 代码实现

代码实现

- QuickSort

```
//快速排序
void QuickSort(int A[],int low,int high){
    if(low<high){ //递归跳出的条件
        int pivotpos = Partition(A,low,high); //划分
        QuickSort(A,low,pivotpos-1); //划分左子表
        QuickSort(A,pivotpos+1,high); //划分右子表
    }
}
```

- Partition

```
//用第一个元素将待排序序列划分成左右两个部分
int Partition(int A[],int low,int high){
    int pivot=A[low]; //第一个元素作为枢纽
    while(low<high){ //用low, high搜索枢纽的最终位置
        while(low<high&&A[high]>=pivot)
            --high;
    }
}
```

```

A[low]=A[high];    //比枢轴小的元素移动到右端
while(low<high&&A[low]<=pivot)
    ++low;
A[high]=A[low];    //比枢轴元素大的移动到右端
}
A[low]=pivot;      //枢轴元素存放在最终位置
return low;        //返回存放枢轴的最终位置
}

```

31.2.3 补充：分析某趟排序后的可能枢纽

补充：分析某趟排序后的可能枢纽

• 题目：

【2019 统考真题】排序过程中，对尚未确定最终位置的所有元素进行一遍处理称为一“趟”。下列序列中，不可能是快速排序第二趟结果的是（ ）。

- A. 5,2,16,12,28,60,32,72
- B. 2,16,5,28,12,60,32,72
- C. 2,12,16,5,28,32,72,60
- D. 5,2,12,28,16,32,72,60

• 总结：

第一步：两趟后排序

三个枢纽时

第二步：找可能枢纽

A: 5 2 16 12 28 60 32 72
X X X X ✓ X X ✓
2 16 5 12
但 2 16 5 12 在 28 的右边

D: 5 2 12 28 16 32 72 60
X X ✓ X X ✓ X X

第三步：

★ 分析可能枢纽的方法

- 第一步：确定递归树
根据当前执行的趟数，分析抽象递归树的可能情况
- 第二步：找可能枢纽
根据当前可能的递归情况，在序列中对每个元素进行判断，判断其是否左边的元素都比其小、右边的元素都比其大，如果满足的话，这个数字就是一个可能的枢纽
- 第三步：分析枢纽是否符合递归树
得到所有可能的枢纽后，看起是否符合可能的抽象递归树；