

本节内容

快速排序

王道考研/CSKAOYAN.COM

知识总览



基于“交换”的排序：根据序列中两个元素关键字的比较结果来对换这两个记录在序列中的位置

王道考研/CSKAOYAN.COM

49	38	65	97	76	13	27	<u>49</u>
0	1	2	3	4	5	6	7

算法思想：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。

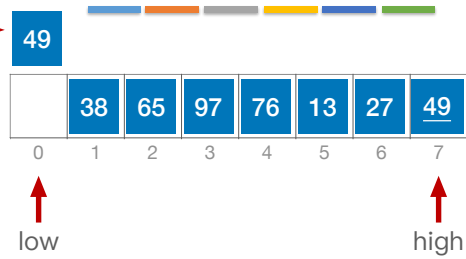
王道考研/CSKAOYAN.COM

Diagram illustrating an array with 8 elements: 49, 38, 65, 97, 76, 13, 27, 49. The first and last elements are underlined. Below the array, a red arrow points up to the first element (index 0) labeled 'low', and another red arrow points up to the last element (index 7) labeled 'high'.

王道考研/CSKAOYAN.COM

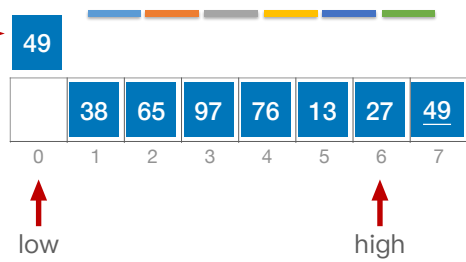
快速排序

更小的元素都交换到左边
更大的元素都交换到右边



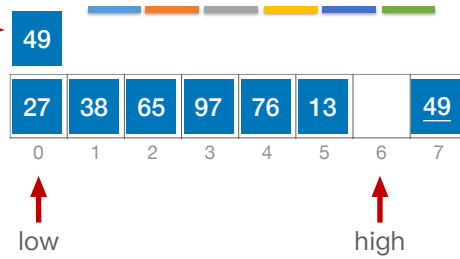
快速排序

更小的元素都交换到左边
更大的元素都交换到右边



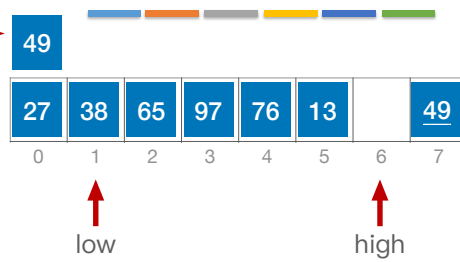
快速排序

更小的元素都交换到左边
更大的元素都交换到右边



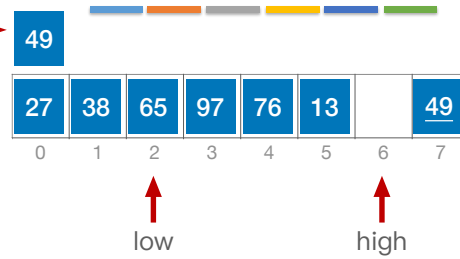
快速排序

更小的元素都交换到左边
更大的元素都交换到右边



快速排序

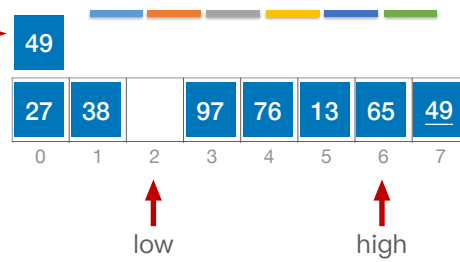
更小的元素都交换到左边
更大的元素都交换到右边



王道考研/CSKAOYAN.COM

快速排序

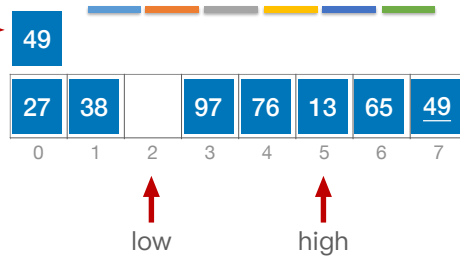
更小的元素都交换到左边
更大的元素都交换到右边



王道考研/CSKAOYAN.COM

快速排序

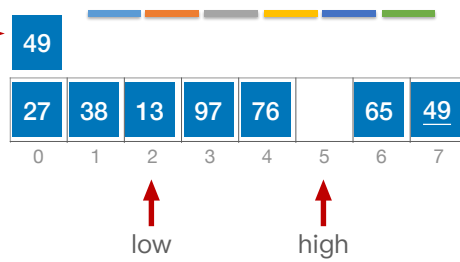
更小的元素都交换到左边
更大的元素都交换到右边



王道考研/CSKAOYAN.COM

快速排序

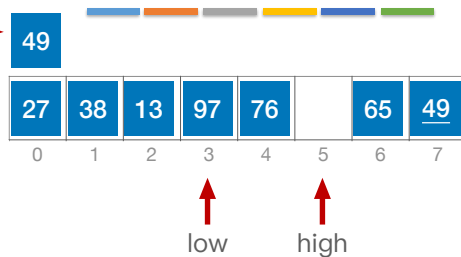
更小的元素都交换到左边
更大的元素都交换到右边



王道考研/CSKAOYAN.COM

快速排序

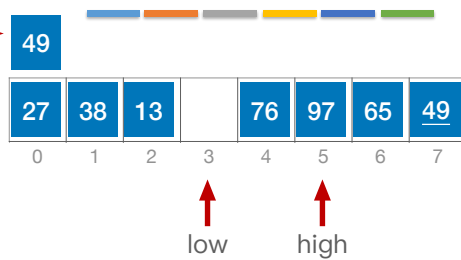
更小的元素都交换到左边
更大的元素都交换到右边



王道考研/CSKAOYAN.COM

快速排序

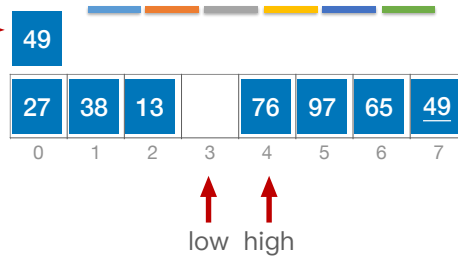
更小的元素都交换到左边
更大的元素都交换到右边



王道考研/CSKAOYAN.COM

快速排序

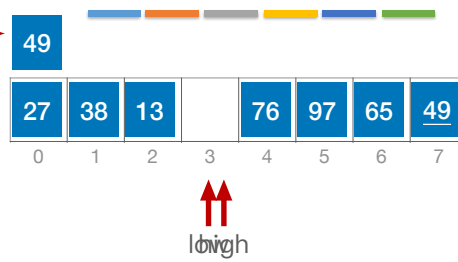
更小的元素都交换到左边
更大的元素都交换到右边



王道考研/CSKAOYAN.COM

快速排序

更小的元素都交换到左边
更大的元素都交换到右边



王道考研/CSKAOYAN.COM

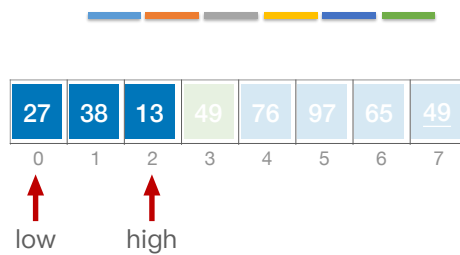
快速排序



算法思想：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。

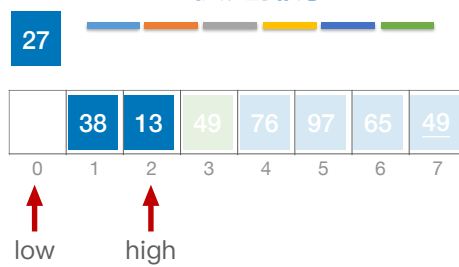
王道考研/CSKAOYAN.COM

快速排序



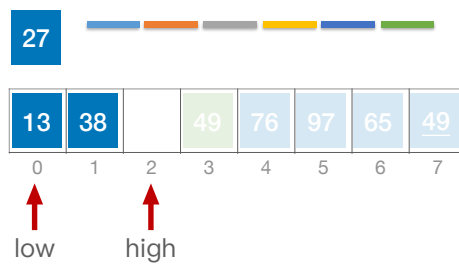
王道考研/CSKAOYAN.COM

快速排序



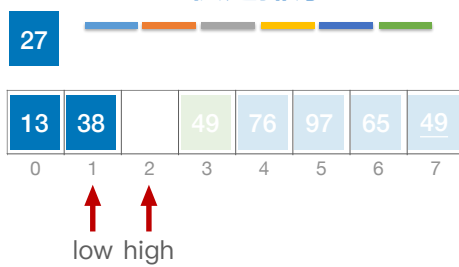
王道考研/CSKAOYAN.COM

快速排序



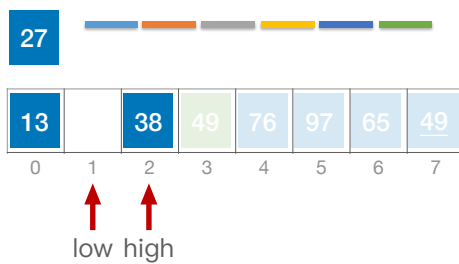
王道考研/CSKAOYAN.COM

快速排序



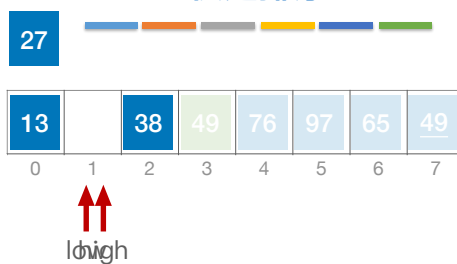
王道考研/CSKAOYAN.COM

快速排序



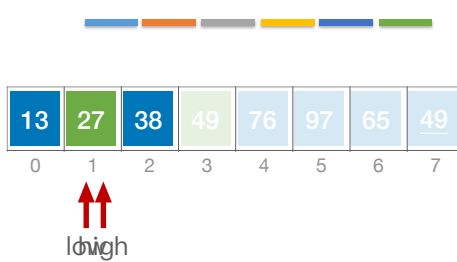
王道考研/CSKAOYAN.COM

快速排序



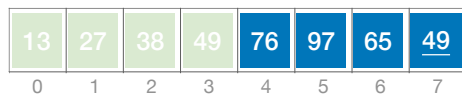
王道考研/CSKAOYAN.COM

快速排序



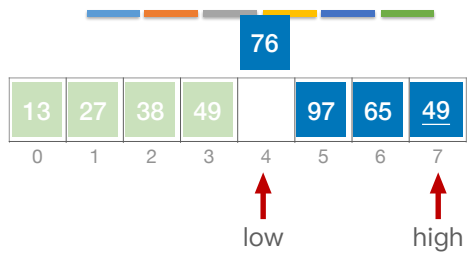
王道考研/CSKAOYAN.COM

快速排序



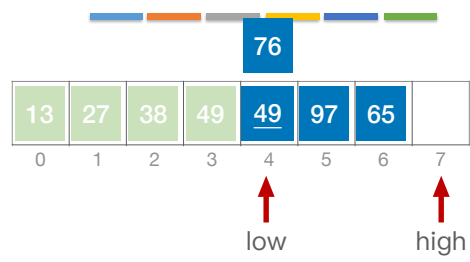
王道考研/CSKAOYAN.COM

快速排序



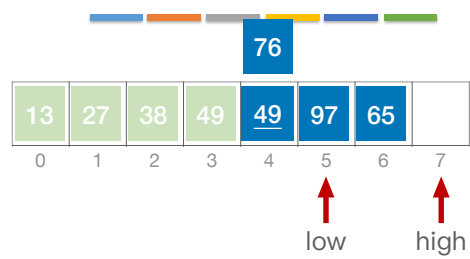
王道考研/CSKAOYAN.COM

快速排序



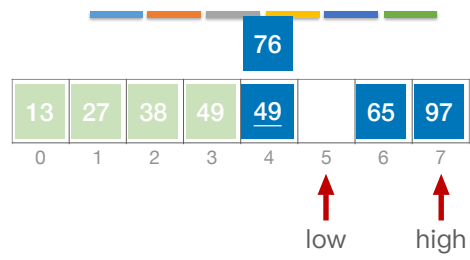
王道考研/CSKAOYAN.COM

快速排序



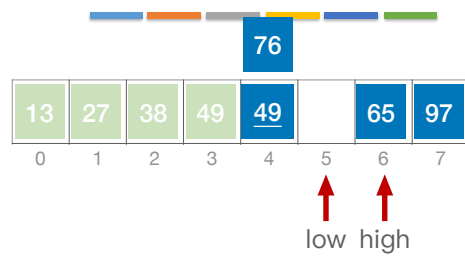
王道考研/CSKAOYAN.COM

快速排序



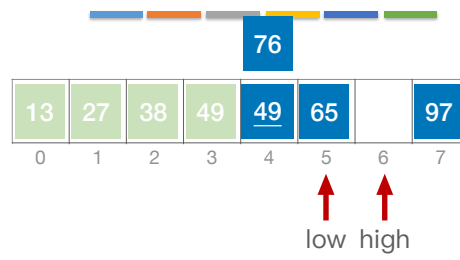
王道考研/CSKAOYAN.COM

快速排序



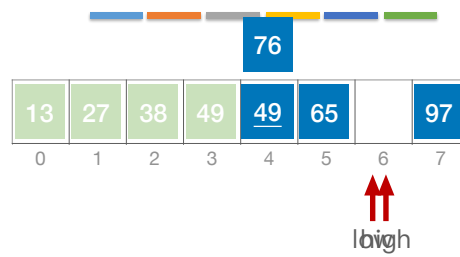
王道考研/CSKAOYAN.COM

快速排序



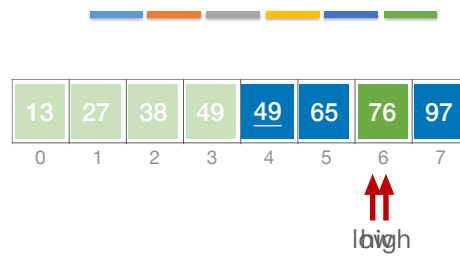
王道考研/CSKAOYAN.COM

快速排序



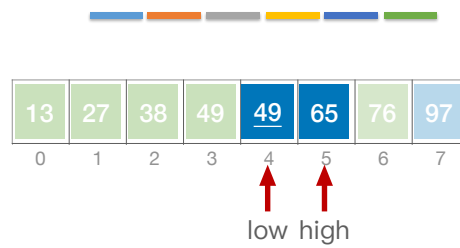
王道考研/CSKAOYAN.COM

快速排序



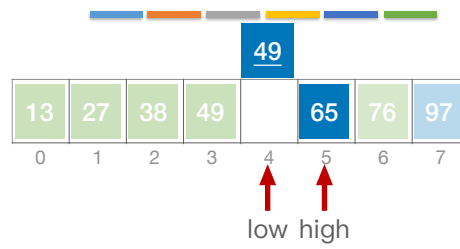
王道考研/CSKAOYAN.COM

快速排序



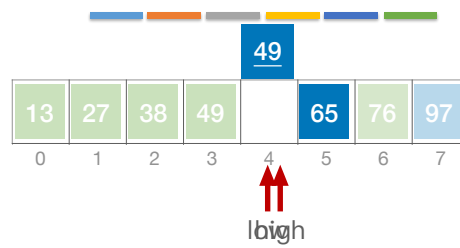
王道考研/CSKAOYAN.COM

快速排序



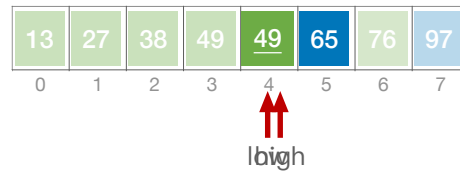
王道考研/CSKAOYAN.COM

快速排序



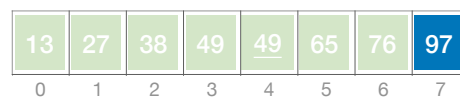
王道考研/CSKAOYAN.COM

快速排序



王道考研/CSKAOYAN.COM

快速排序



王道考研/CSKAOYAN.COM

快速排序

13	27	38	49	49	65	76	97
0	1	2	3	4	5	6	7

算法思想：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。

王道考研/CSKAOYAN.COM

快速排序

49	38	65	97	76	13	27	49
0	1	2	3	4	5	6	7

QuickSort
函数

$l=0, h=7$

递归工作栈

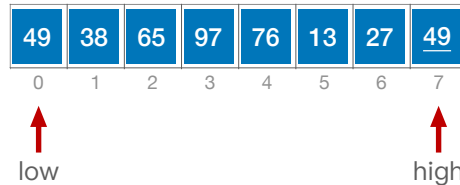
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];          //第一个元素作为枢轴
    while(low<high){           //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];        //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];        //比枢轴大的元素移动到右端
    }
    A[low]=pivot;              //枢轴元素存放到最后位置
    return low;                //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){           //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);       //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



QuickSort
函数

Partition 函数
(处理0~7)
#96, l=0, h=7
递归工作栈

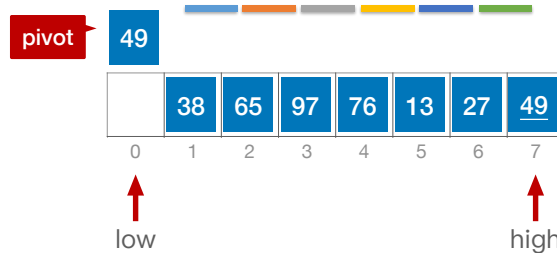
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



Partition 函数
(处理0~7)
#96, l=0, h=7
递归工作栈

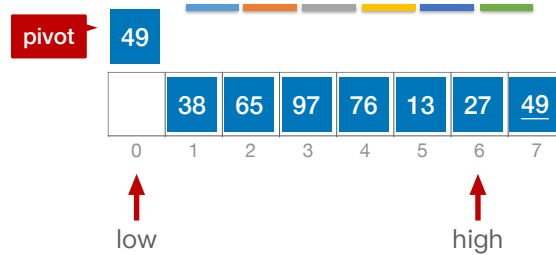
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



Partition 函数
(处理0~7)

#96, l=0, h=7

递归工作栈

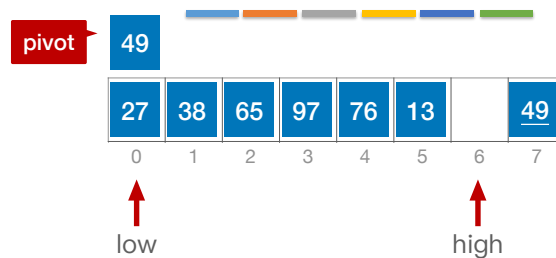
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



Partition 函数
(处理0~7)

#96, l=0, h=7

递归工作栈

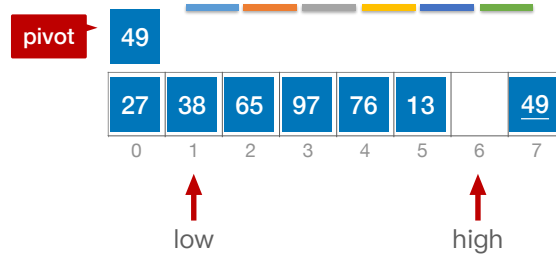
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



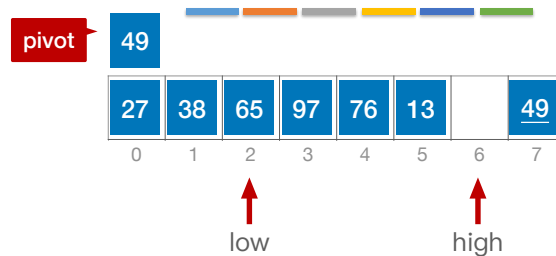
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



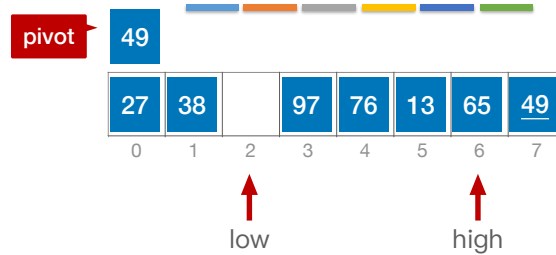
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



递归工作栈

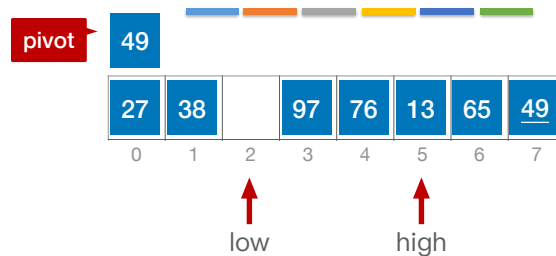
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



递归工作栈

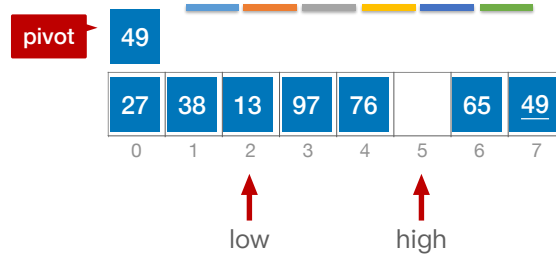
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



Partition 函数
(处理0~7)

#96, l=0, h=7

递归工作栈

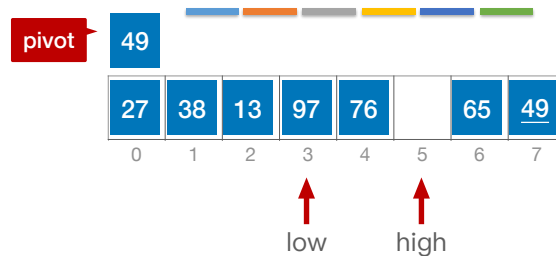
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);       //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



Partition 函数
(处理0~7)

#96, l=0, h=7

递归工作栈

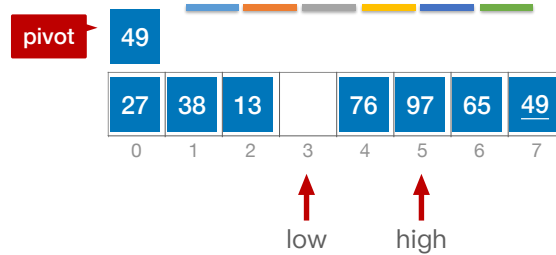
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);       //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



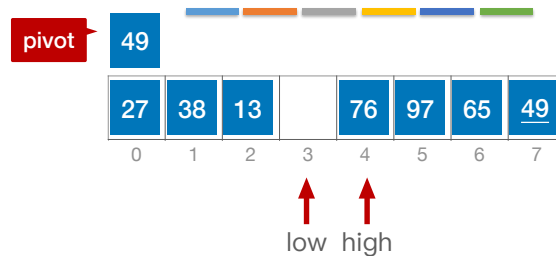
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



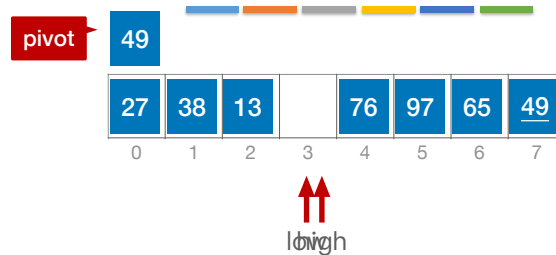
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);        //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



递归工作栈

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

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);        //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



#96, l=0, h=7
p=3

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



#96, l=0, h=7
p=3

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序

27	38	13	49	76	97	65	49
0	1	2	3	4	5	6	7

第二层
QuickSort

l=0, h=2

#97, l=0, h=7
p=3

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序

27	38	13	49	76	97	65	49
0	1	2	3	4	5	6	7

第二层
QuickSort

#96, l=0, h=2

#97, l=0, h=7
p=3

递归工作栈

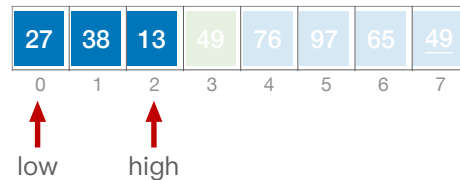
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第二层
QuickSort



递归工作栈

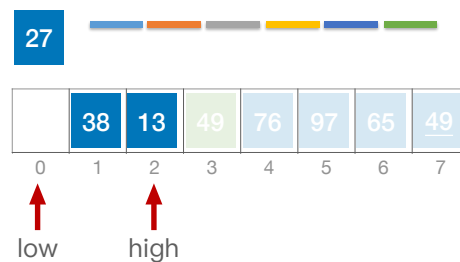
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];     //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第二层
QuickSort



递归工作栈

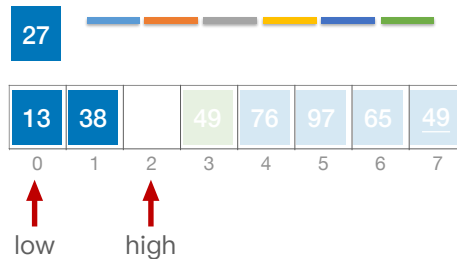
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];     //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第二层
QuickSort



递归工作栈

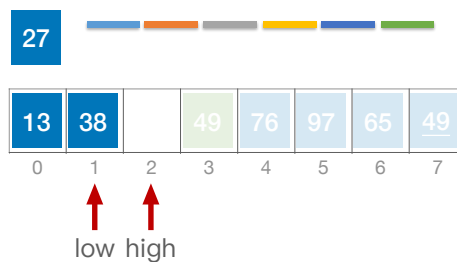
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);        //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第二层
QuickSort



递归工作栈

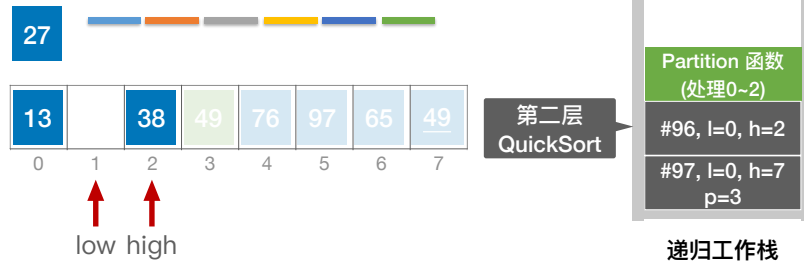
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);        //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



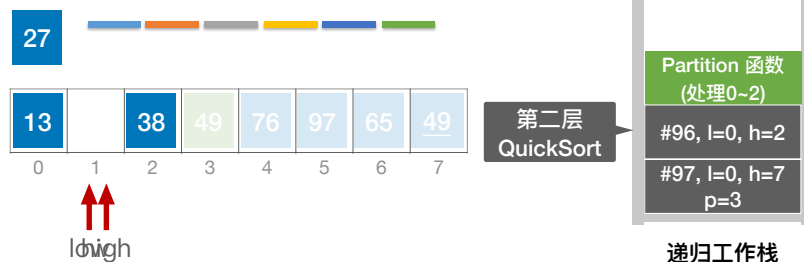
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);        //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



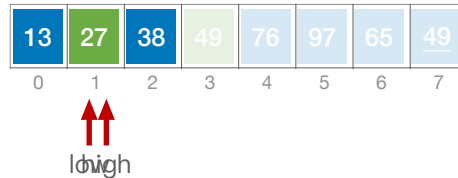
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);        //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第二层
QuickSort



递归工作栈

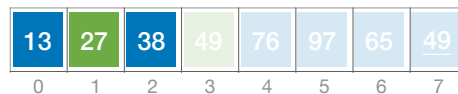
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

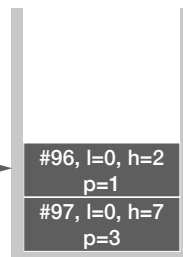
```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第二层
QuickSort



递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序

13	27	38	49	76	97	65	49
0	1	2	3	4	5	6	7

第二层
QuickSort

#96, l=0, h=2
p=1
#97, l=0, h=7
p=3

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序

13	27	38	49	76	97	65	49
0	1	2	3	4	5	6	7

第三层
QuickSort

l=0, h=0
#97, l=0, h=2
p=1
#97, l=0, h=7
p=3

递归工作栈

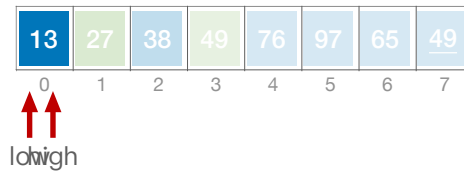
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

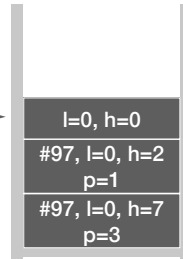
```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第三层
QuickSort



递归工作栈

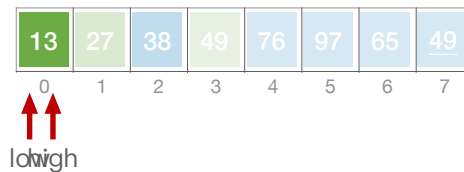
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

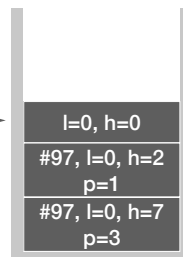
```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){        //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);        //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第三层
QuickSort



递归工作栈

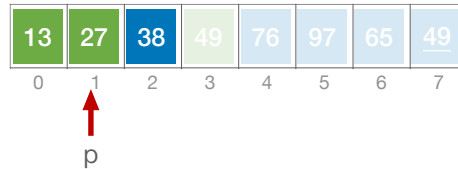
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){        //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);        //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第二层
QuickSort

#98, l=0, h=2
p=1
#97, l=0, h=7
p=3

递归工作栈

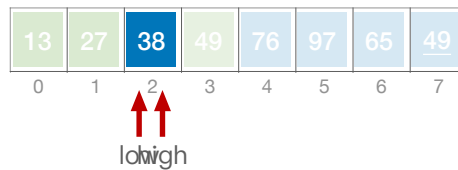
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第三层
QuickSort

l=2, h=2
#98, l=0, h=2
p=1
#97, l=0, h=7
p=3

递归工作栈

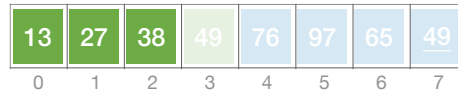
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第二层
QuickSort

#98, l=0, h=2
p=1
#97, l=0, h=7
p=3

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第一层
QuickSort

#97, l=0, h=7
p=3

递归工作栈

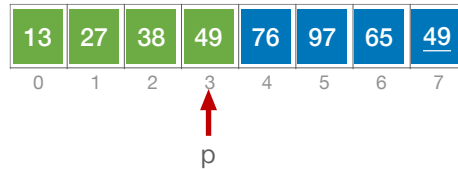
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第一层
QuickSort

#98, l=0, h=7
p=3

递归工作栈

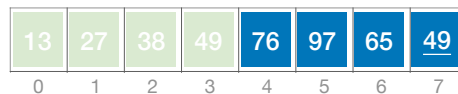
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最终位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1); //划分左子表
98         QuickSort(A,pivotpos+1,high); //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第二层
QuickSort

l=4, h=7
#98, l=0, h=7
p=3

递归工作栈

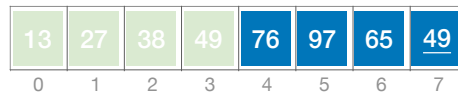
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最终位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1); //划分左子表
98         QuickSort(A,pivotpos+1,high); //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第二层
QuickSort



递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第二层
QuickSort



递归工作栈

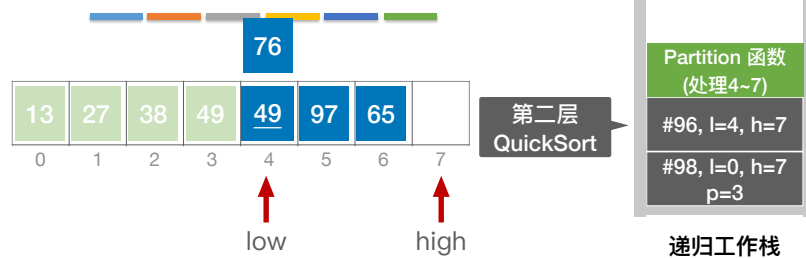
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



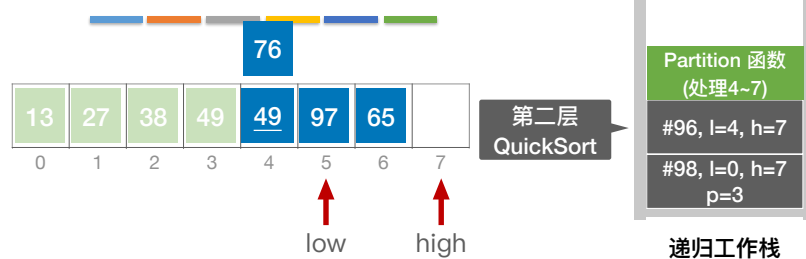
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);        //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



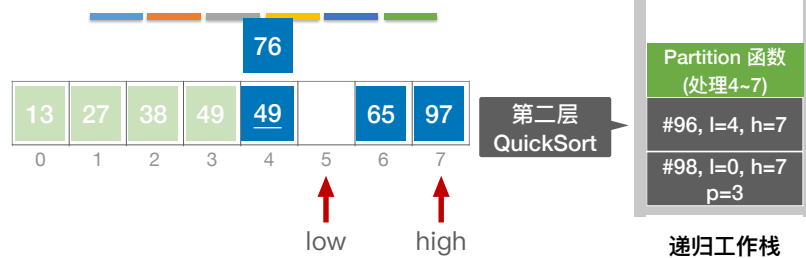
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);        //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



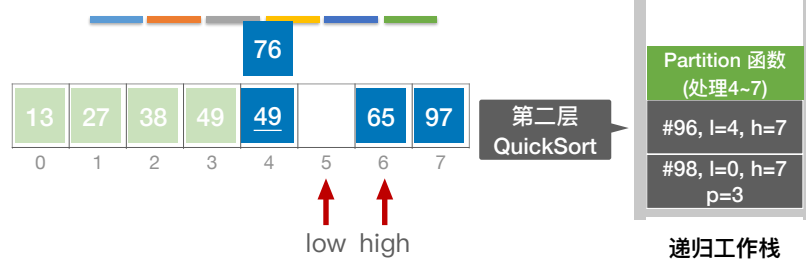
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];     //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);       //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



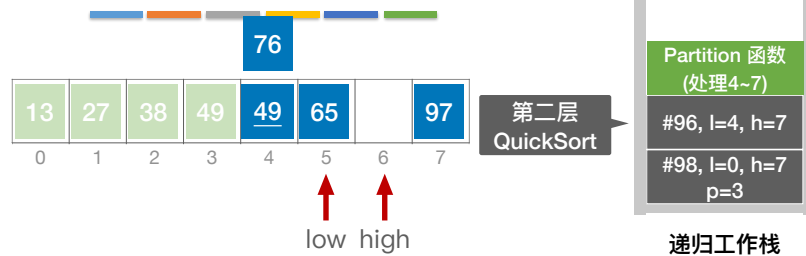
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];     //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);       //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



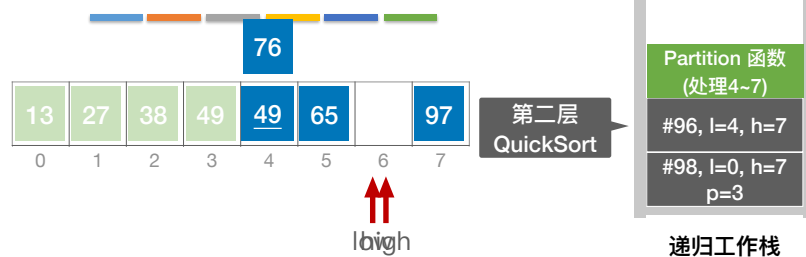
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);        //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);        //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序

13	27	38	49	49	65	76	97
0	1	2	3	4	5	6	7

↑↑
low high

第二层
QuickSort



递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

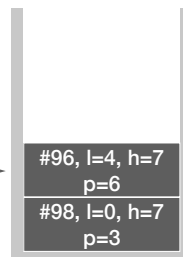
```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);       //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序

13	27	38	49	49	65	76	97
0	1	2	3	4	5	6	7

第二层
QuickSort



递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);       //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序

13	27	38	49	49	65	76	97
0	1	2	3	4	5	6	7

第二层
QuickSort

#96, l=4, h=7
p=6
#98, l=0, h=7
p=3

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序

13	27	38	49	49	65	76	97
0	1	2	3	4	5	6	7

第三层
QuickSort

l=4, h=5
#97, l=4, h=7
p=6
#98, l=0, h=7
p=3

递归工作栈

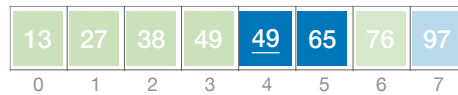
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第三层
QuickSort

Partition 函数 (处理4~5)	
#96, l=4, h=5	
#97, l=4, h=7 p=6	
#98, l=0, h=7 p=3	

递归工作栈

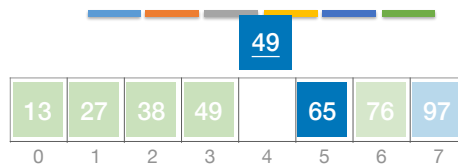
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第三层
QuickSort

Partition 函数 (处理4~5)	
#96, l=4, h=5	
#97, l=4, h=7 p=6	
#98, l=0, h=7 p=3	

递归工作栈

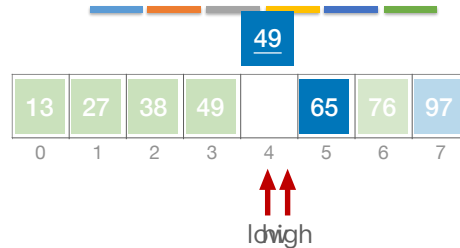
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第三层
QuickSort

Partition 函数 (处理4~5)	
#96, l=4, h=5	
#97, l=4, h=7 p=6	
#98, l=0, h=7 p=3	

递归工作栈

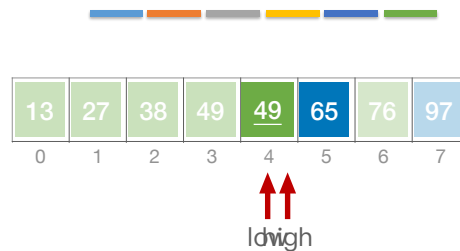
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);        //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第三层
QuickSort

Partition 函数 (处理4~5)	
#96, l=4, h=5	
#97, l=4, h=7 p=6	
#98, l=0, h=7 p=3	

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);        //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第三层
QuickSort

#96, l=4, h=5
p=4
#97, l=4, h=7
p=6
#98, l=0, h=7
p=3

递归工作栈

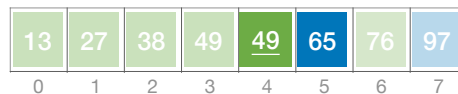
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第三层
QuickSort

#96, l=4, h=5
p=4
#97, l=4, h=7
p=6
#98, l=0, h=7
p=3

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){         //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];      //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];      //比枢轴大的元素移动到右端
    }
    A[low]=pivot;            //枢轴元素存放到最后位置
    return low;              //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){         //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);      //划分左子表
98         QuickSort(A,pivotpos+1,high);     //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第四层
QuickSort

l=4, h=3
#97, l=4, h=5 p=4
#97, l=4, h=7 p=6
#98, l=0, h=7 p=3

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);       //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第三层
QuickSort

#97, l=4, h=5 p=4
#97, l=4, h=7 p=6
#98, l=0, h=7 p=3

递归工作栈

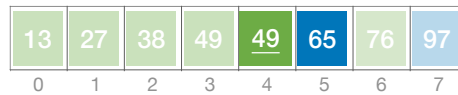
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);       //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第三层
QuickSort

#97, l=4, h=5
p=4
#97, l=4, h=7
p=6
#98, l=0, h=7
p=3

递归工作栈

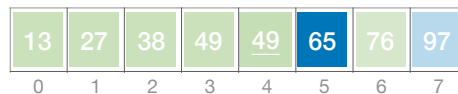
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第四层
QuickSort

l=5, h=5
#98, l=4, h=5
p=4
#97, l=4, h=7
p=6
#98, l=0, h=7
p=3

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第四层
QuickSort

l=5, h=5
#98, l=4, h=5 p=4
#97, l=4, h=7 p=6
#98, l=0, h=7 p=3

递归工作栈

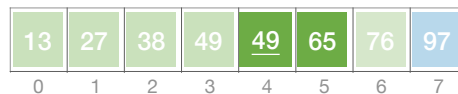
//用第一个元素将待排序序列划分成左右两个部分

```
int Partition(int A[],int low,int high){
    int pivot=A[low];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最终位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序



第三层
QuickSort

#98, l=4, h=5 p=4
#97, l=4, h=7 p=6
#98, l=0, h=7 p=3

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最终位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序

13	27	38	49	49	65	76	97
0	1	2	3	4	5	6	7

第二层
QuickSort

#97, l=4, h=7
p=6
#98, l=0, h=7
p=3

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序

13	27	38	49	49	65	76	97
0	1	2	3	4	5	6	7

第二层
QuickSort

#97, l=4, h=7
p=6
#98, l=0, h=7
p=3

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序

13	27	38	49	49	65	76	97
0	1	2	3	4	5	6	7

第三层
QuickSort

l=7, h=7
#98, l=4, h=7 p=6
#98, l=0, h=7 p=3

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];        //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];        //比枢轴大的元素移动到右端
    }
    A[low]=pivot;              //枢轴元素存放到最后位置
    return low;                //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){           //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);        //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序

13	27	38	49	49	65	76	97
0	1	2	3	4	5	6	7

第三层
QuickSort

l=7, h=7
#98, l=4, h=7 p=6
#98, l=0, h=7 p=3

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];        //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];        //比枢轴大的元素移动到右端
    }
    A[low]=pivot;              //枢轴元素存放到最后位置
    return low;                //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){           //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);        //划分左子表
98         QuickSort(A,pivotpos+1,high);        //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序

13	27	38	49	49	65	76	97
0	1	2	3	4	5	6	7

#98, l=4, h=7
p=6
#98, l=0, h=7
p=3

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序

13	27	38	49	49	65	76	97
0	1	2	3	4	5	6	7

#98, l=0, h=7
p=3

递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

快速排序

13	27	38	49	49	65	76	97
0	1	2	3	4	5	6	7



递归工作栈

//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];        //第一个元素作为枢轴
    while(low<high){          //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];       //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        A[high]=A[low];       //比枢轴大的元素移动到右端
    }
    A[low]=pivot;             //枢轴元素存放到最后位置
    return low;               //返回存放枢轴的最终位置
}
```

```
93 //快速排序
94 void QuickSort(int A[],int low,int high){
95     if(low<high){          //递归跳出的条件
96         int pivotpos=Partition(A,low,high); //划分
97         QuickSort(A,low,pivotpos-1);       //划分左子表
98         QuickSort(A,pivotpos+1,high);      //划分右子表
99     }
100 }
```

王道考研/CSKAOYAN.COM

算法效率分析

初始序列

49	38	65	97	76	13	27	49
0	1	2	3	4	5	6	7

(待排序元素: 0~7)

第一层QuickSort处理后:

27	38	13	49	76	97	65	49
0	1	2	3	4	5	6	7

(待排序元素: 0~2, 4~7)

第二层QuickSort处理后:

13	27	38	49	49	65	76	97
0	1	2	3	4	5	6	7

(待排序元素: 0~0, 2~2, 4~5, 7~7)

第三层QuickSort处理后:

13	27	38	49	49	65	76	97
0	1	2	3	4	5	6	7

(待排序元素: 5~5)

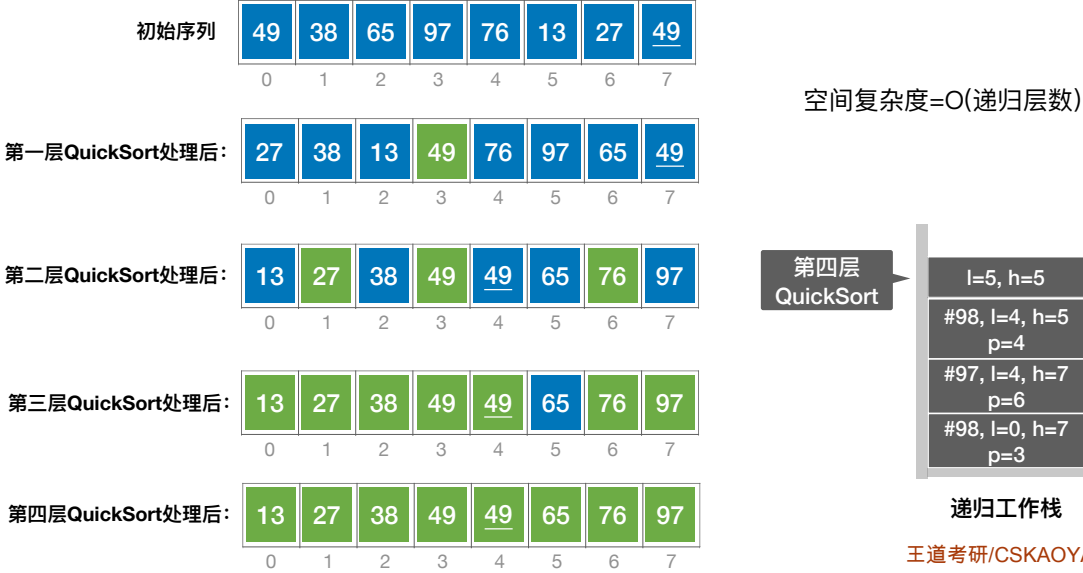
第四层QuickSort处理后:

13	27	38	49	49	65	76	97
0	1	2	3	4	5	6	7

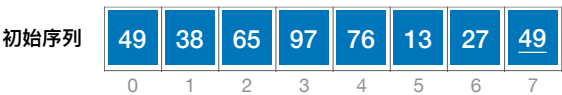
每一层的QuickSort 只需要处理剩余的待排序元素, 时间复杂度不超过 $O(n)$
时间复杂度= $O(n \times \text{递归层数})$

王道考研/CSKAOYAN.COM

算法效率分析



算法效率分析



算法效率分析

第一层QuickSort处理后:



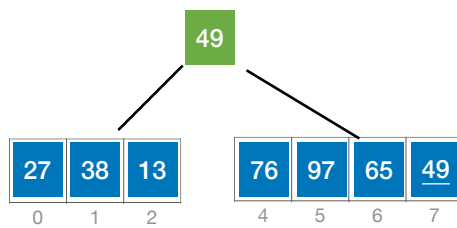
27	38	13	49	76	97	65	49
0	1	2	3	4	5	6	7

王道考研/CSKAOYAN.COM

算法效率分析

第一层QuickSort处理后:

第二层QuickSort
要处理的部分:

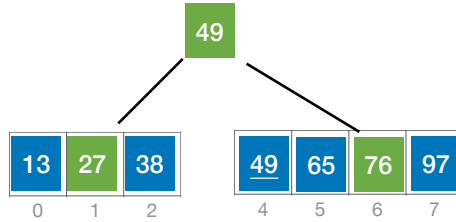


王道考研/CSKAOYAN.COM

算法效率分析

第一层QuickSort处理后:

第二层QuickSort处理后:



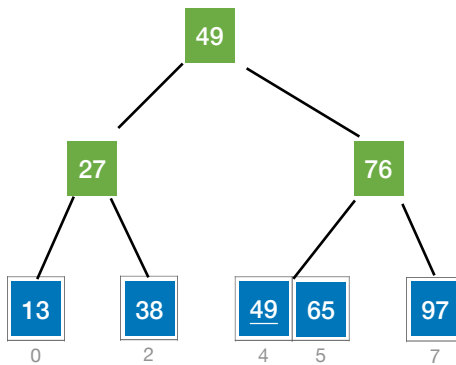
王道考研/CSKAOYAN.COM

算法效率分析

第一层QuickSort处理后:

第二层QuickSort处理后:

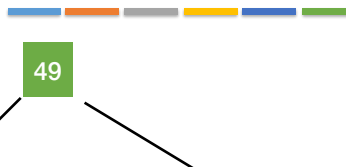
第三层QuickSort
要处理的部分:



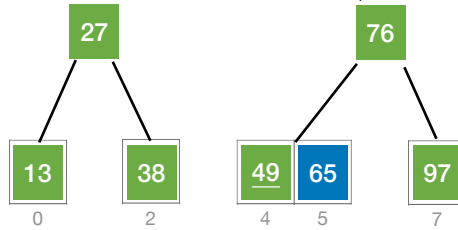
王道考研/CSKAOYAN.COM

算法效率分析

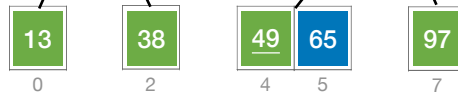
第一层QuickSort处理后:



第二层QuickSort处理后:



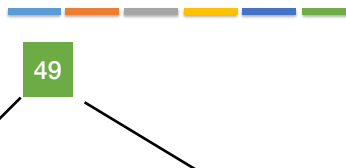
第三层QuickSort处理后:



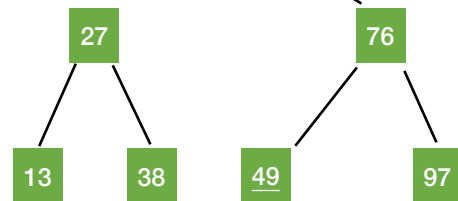
王道考研/CSKAOYAN.COM

算法效率分析

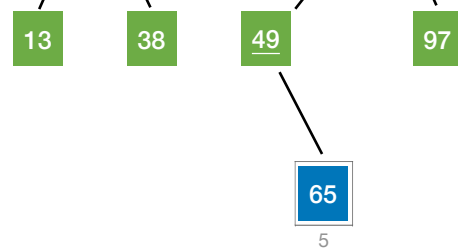
第一层QuickSort处理后:



第二层QuickSort处理后:



第三层QuickSort处理后:



第四层QuickSort
要处理的部分:



王道考研/CSKAOYAN.COM

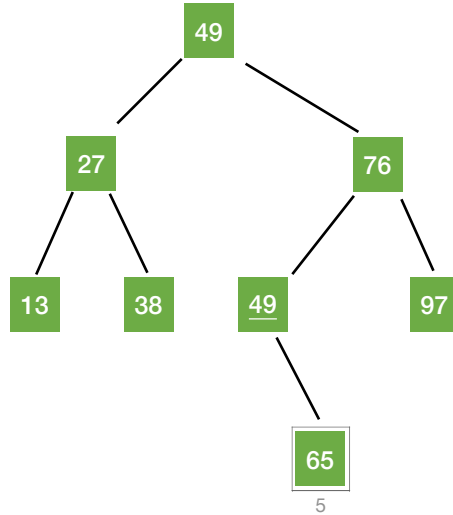
算法效率分析

第一层QuickSort处理后:

第二层QuickSort处理后:

第三层QuickSort处理后:

第四层QuickSort处理后:



王道考研/CSKAOYAN.COM

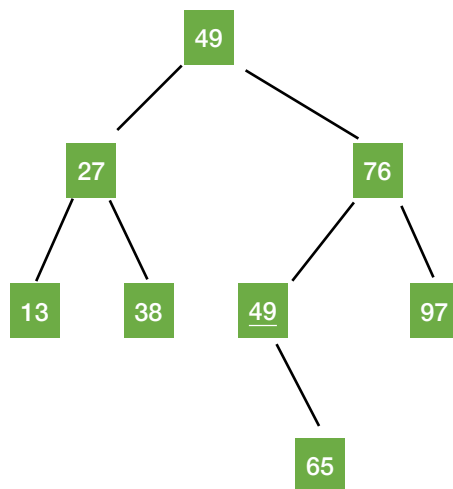
算法效率分析

第一层QuickSort处理后:

第二层QuickSort处理后:

第三层QuickSort处理后:

第四层QuickSort处理后:



把n个元素组织
成二叉树，二叉
树的层数就是递
归调用的层数

n个结点的二叉树
最小高度 = $\lfloor \log_2 n \rfloor + 1$
最大高度 = n

王道考研/CSKAOYAN.COM

n个结点的二叉树

$$\text{最小高度} = \lfloor \log_2 n \rfloor + 1$$

最大高度 = n

时间复杂度= $O(n \times \text{递归层数})$

空间复杂度=O(递归层数)

最好时间复杂度= $O(n\log_2 n)$

最好空间复杂度= $O(\log_2 n)$

最坏时间复杂度= $O(n^2)$

最坏空间复杂度= $O(n)$

王道考研/CSKAOYAN.COM

初始序列



0

1

2

3

4

5

6

7

↑
high

王道考研/CSKAOYAN.COM

最坏的情况



第一层QuickSort处理后:



王道考研/CSKAOYAN.COM

最坏的情况



第一层QuickSort处理后:



若每一次选中的“枢轴”将待排序序列划分为很不均匀的两个部分，则会导致递归深度增加，算法效率变低

第二层QuickSort处理后:



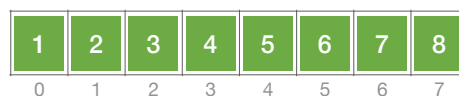
若初始序列有序或逆序，则快速排序的性能最差（因为每次选择的都是最靠边的元素）

第三层QuickSort处理后:



⋮

第8层QuickSort处理后:



王道考研/CSKAOYAN.COM

比较好的情况

初始序列	49	38	65	97	76	13	27	49
	0	1	2	3	4	5	6	7
第一层QuickSort处理后:	27	38	13	49	76	97	65	49
	0	1	2	3	4	5	6	7
第二层QuickSort处理后:	13	27	38	49	49	65	76	97
	0	1	2	3	4	5	6	7
第三层QuickSort处理后:	13	27	38	49	49	65	76	97
	0	1	2	3	4	5	6	7
第四层QuickSort处理后:	13	27	38	49	49	65	76	97
	0	1	2	3	4	5	6	7

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。

eg：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

王道考研/CSKAOYAN.COM

算法效率分析

时间复杂度= $O(n \times \text{递归层数})$

空间复杂度= $O(\text{递归层数})$

最好时间复杂度= $O(n \log_2 n)$

最坏时间复杂度= $O(n^2)$

每次选的枢轴元素都能将序列划分成均匀的两部分

最好空间复杂度= $O(\log_2 n)$

最坏空间复杂度= $O(n)$

若序列原本就有序或逆序，则时、空复杂度最高（可优化，尽量选择可以把数据中分的枢轴元素。）

厉害厉害



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

平均时间复杂度= $O(n \log_2 n)$

王道考研/CSKAOYAN.COM

稳定性



2	2	1
---	---	---

0

1

2



low

high

稳定性



2

	2	1
--	---	---

0

1

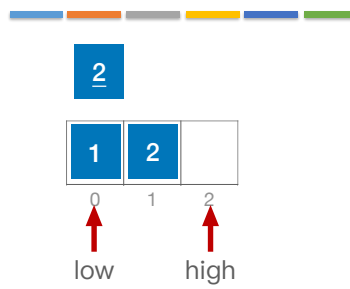
2



low

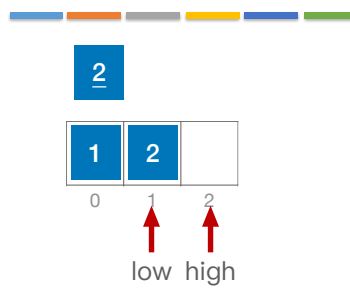
high

稳定性



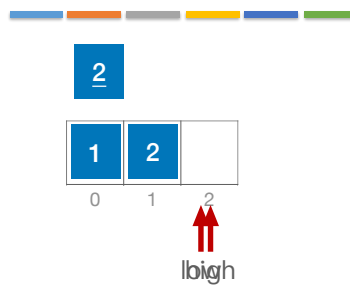
王道考研/CSKAOYAN.COM

稳定性



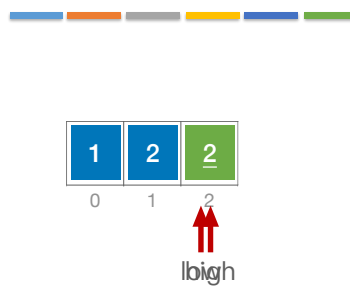
王道考研/CSKAOYAN.COM

稳定性



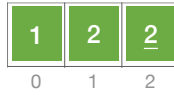
王道考研/CSKAOYAN.COM

稳定性



王道考研/CSKAOYAN.COM

稳定性



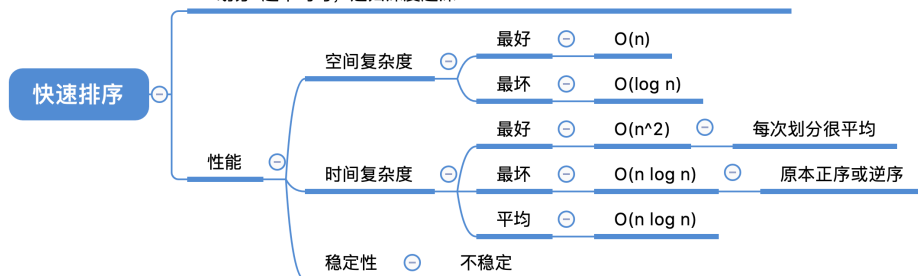
不稳定!

王道考研/CSKAOYAN.COM

知识回顾与重要考点

算法思想：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。

算法表现主要取决于递归深度，若每次“划分”越均匀，则递归深度越低。
“划分”越不均匀，递归深度越深



注：408原题中说，对所有尚未确定最终位置的所有元素进行一遍处理称为“一趟”排序，因此一次“划分”≠一趟排序。一次划分可以确定一个元素的最终位置，而一趟排序也许可以确定多个元素的最终位置。

王道考研/CSKAOYAN.COM