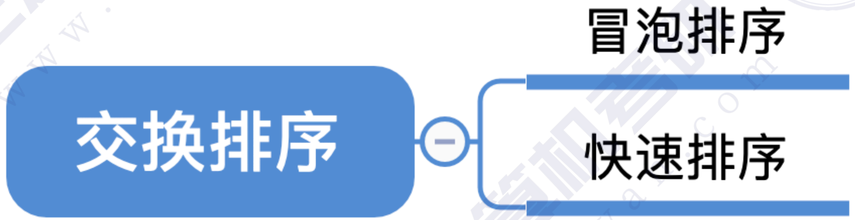
本节内容

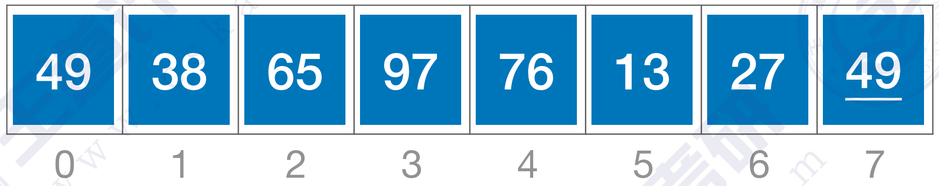
快速排序

知识总览



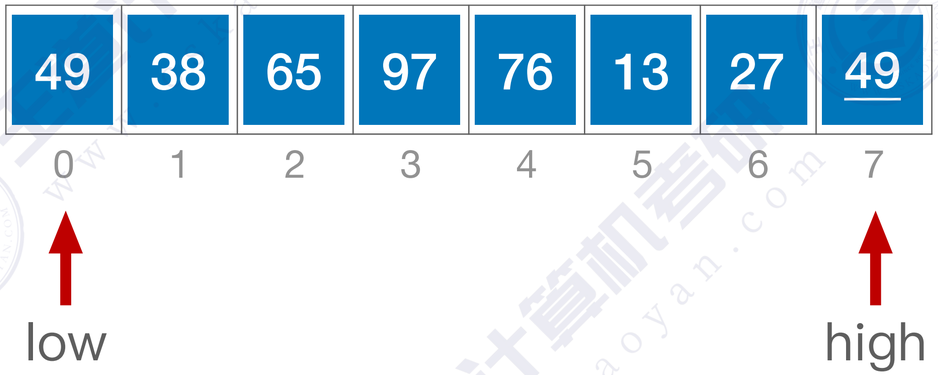
基于“交换”的排序：根据序列中两个元素关键字的比较结果来对换这两个记录在序列中的位置

快速排序



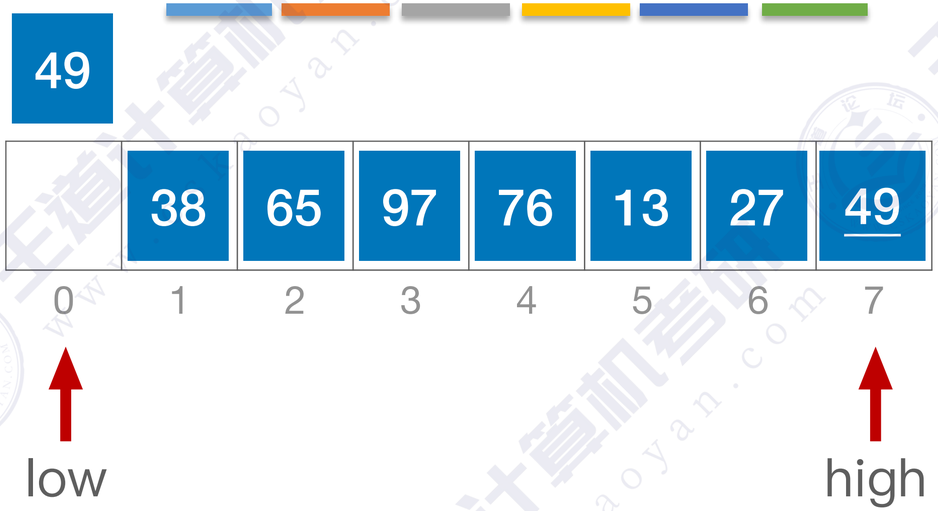
算法思想：在待排序表L[1...n]中任取一个元素pivot作为枢轴（或基准，通常取首元素），通过一趟排序将待排序表划分为独立的两部分L[1...k−1]和L[k+1...n]，使得L[1...k−1]中的所有元素小于pivot，L[k+1...n]中的所有元素大于等于pivot，则pivot放在了其最终位置L(k)上，这个过程称为一次“划分”。然后分别递归地对两个子表重复上述过程，直至每部分内只有一个元素或空为止，即所有元素放在了其最终位置上。

快速排序



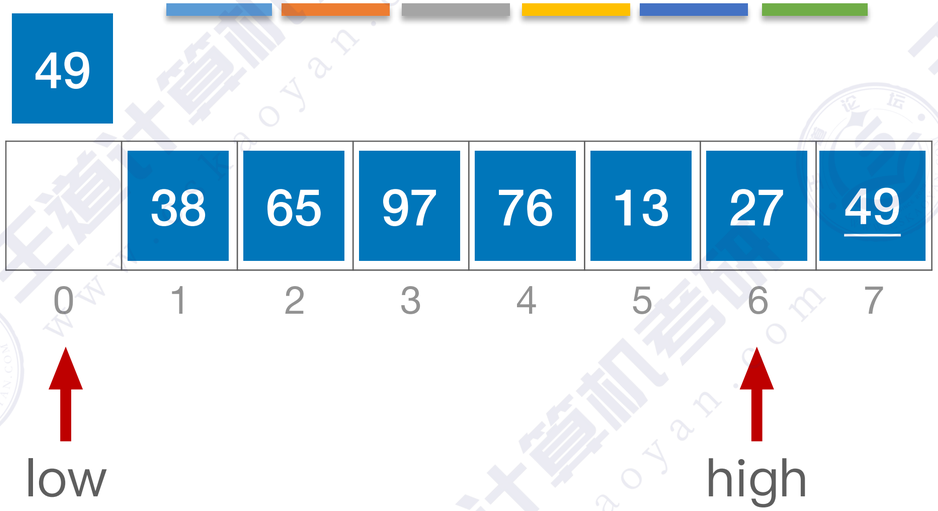
快速排序

更小的元素都交换到左边
更大的元素都交换到右边



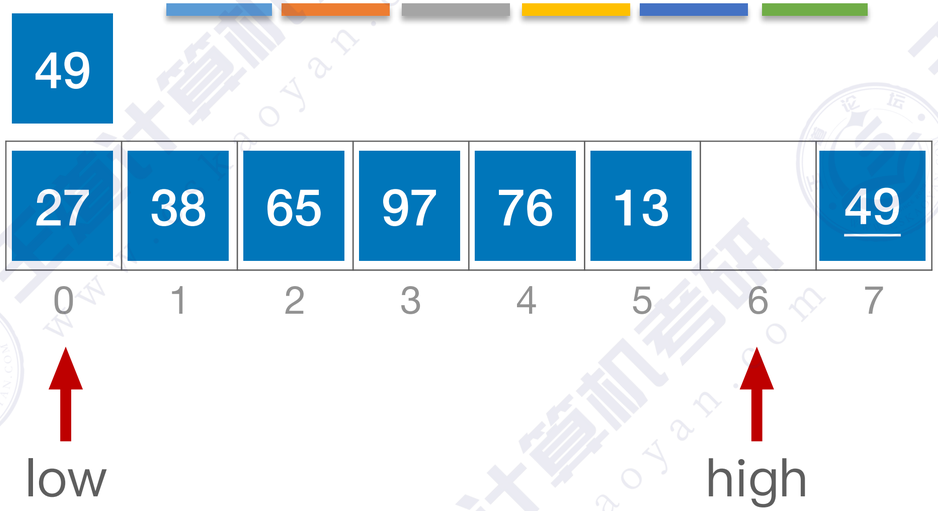
快速排序

更小的元素都交换到左边
更大的元素都交换到右边



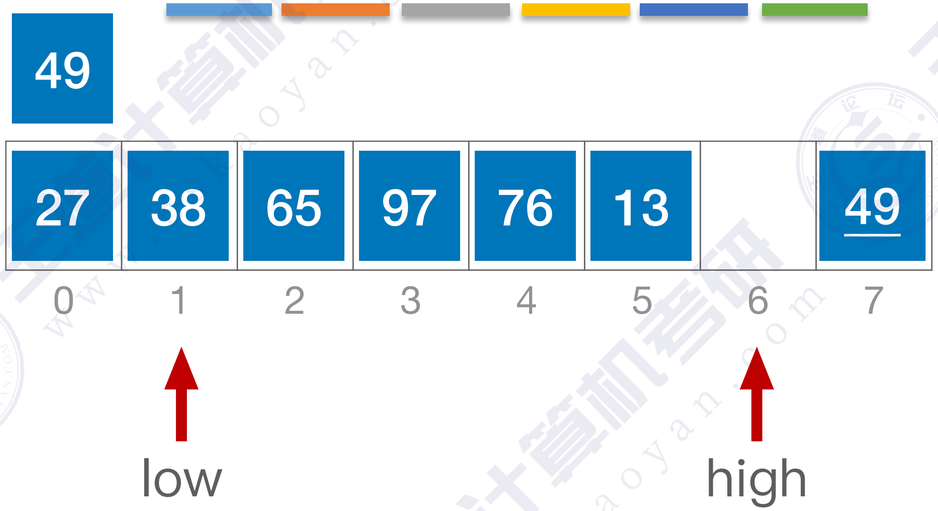
快速排序

更小的元素都交换到左边
更大的元素都交换到右边



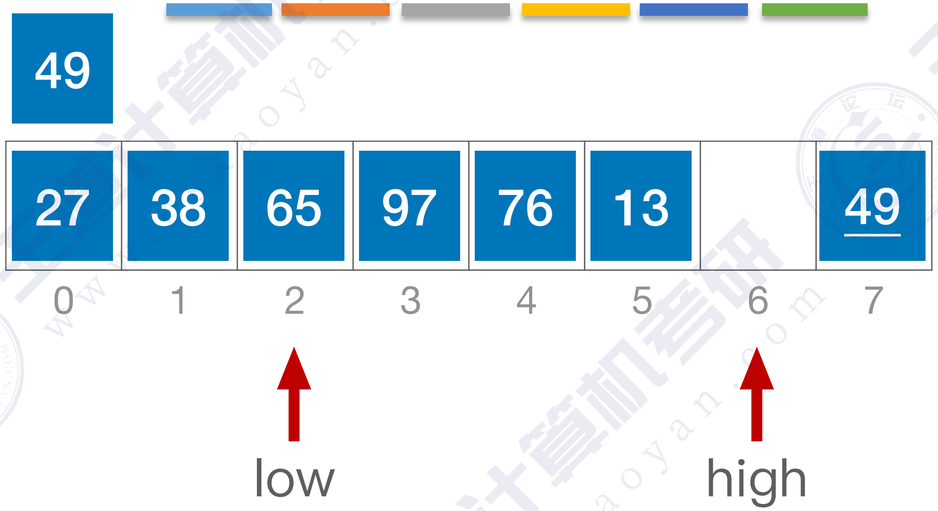
快速排序

更小的元素都交换到左边
更大的元素都交换到右边



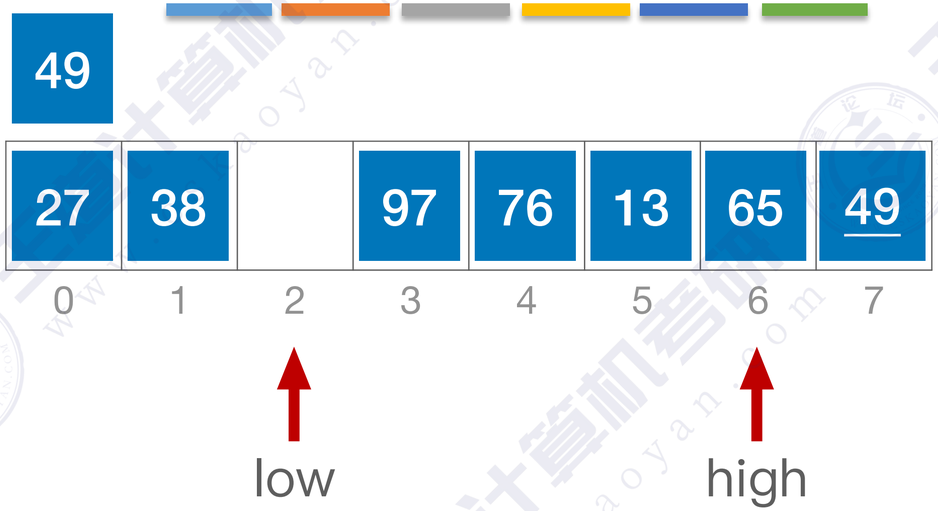
快速排序

更小的元素都交换到左边
更大的元素都交换到右边



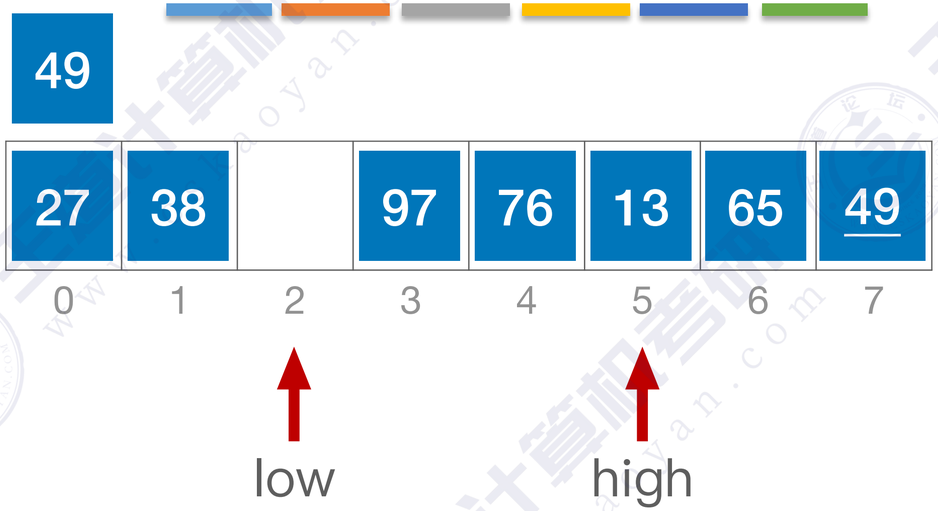
快速排序

更小的元素都交换到左边
更大的元素都交换到右边



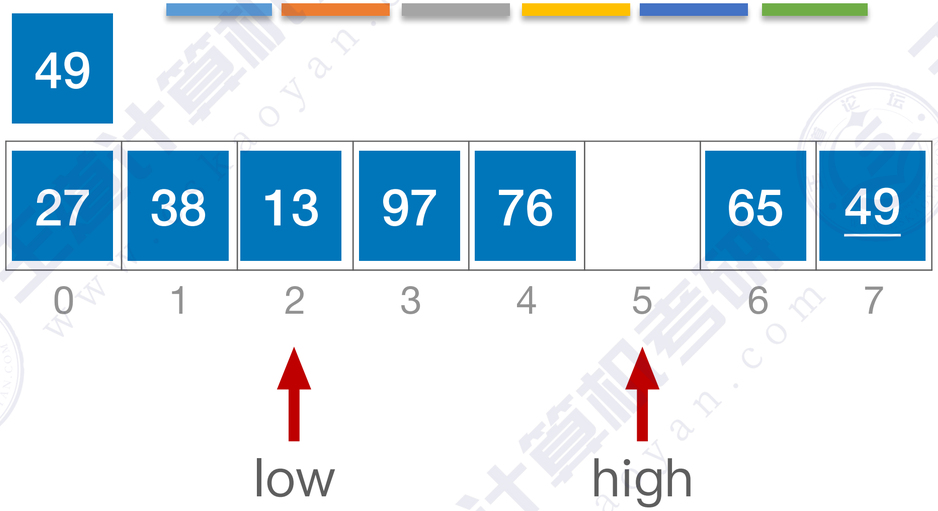
快速排序

更小的元素都交换到左边
更大的元素都交换到右边



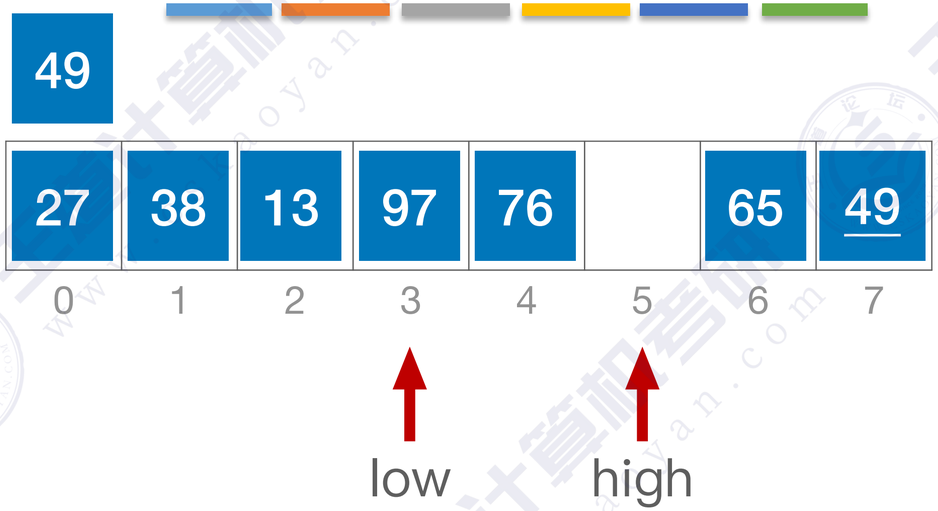
快速排序

更小的元素都交换到左边
更大的元素都交换到右边



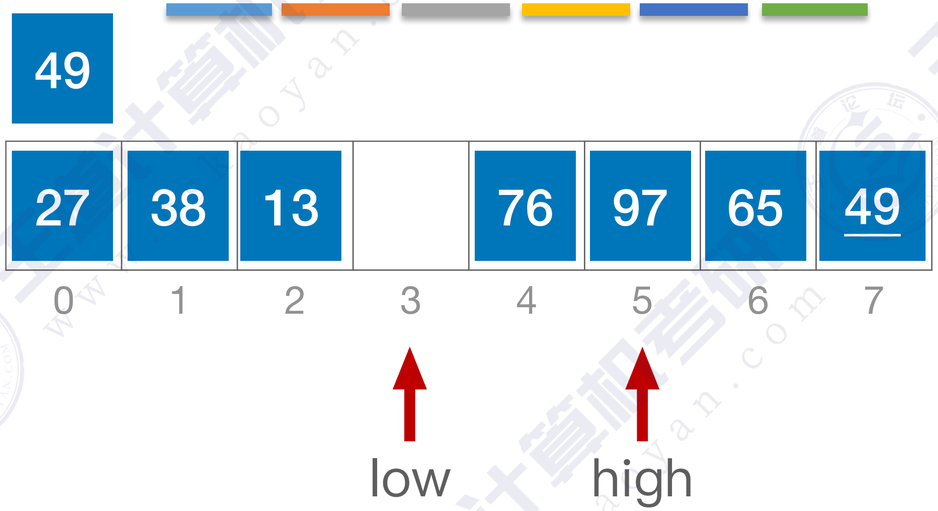
快速排序

更小的元素都交换到左边
更大的元素都交换到右边



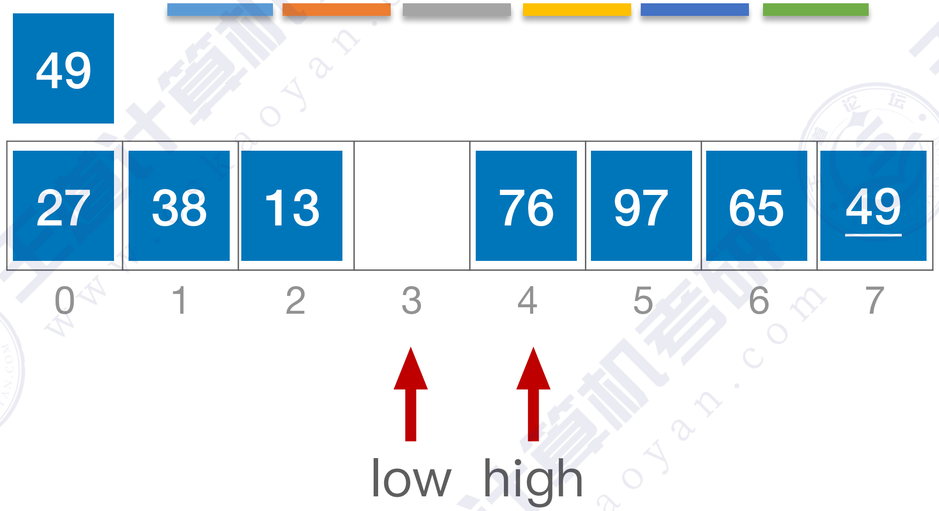
快速排序

更小的元素都交换到左边
更大的元素都交换到右边



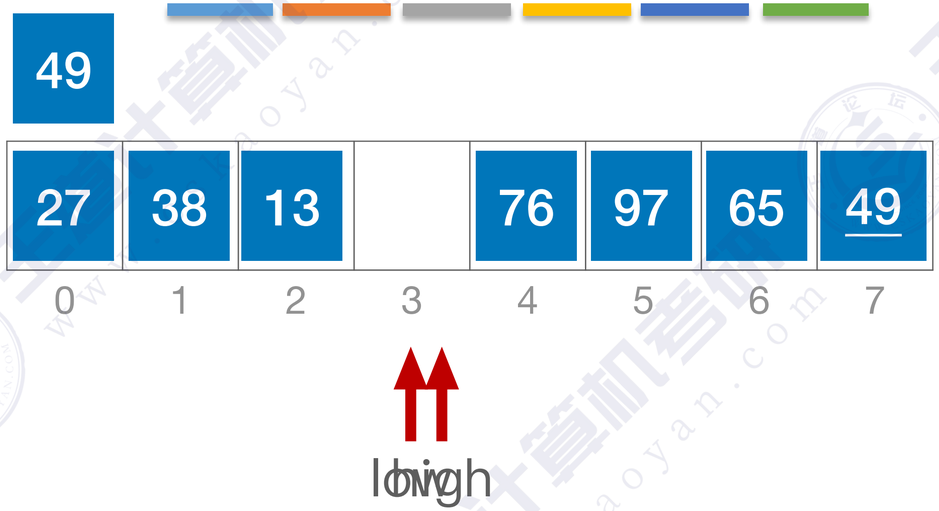
快速排序

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

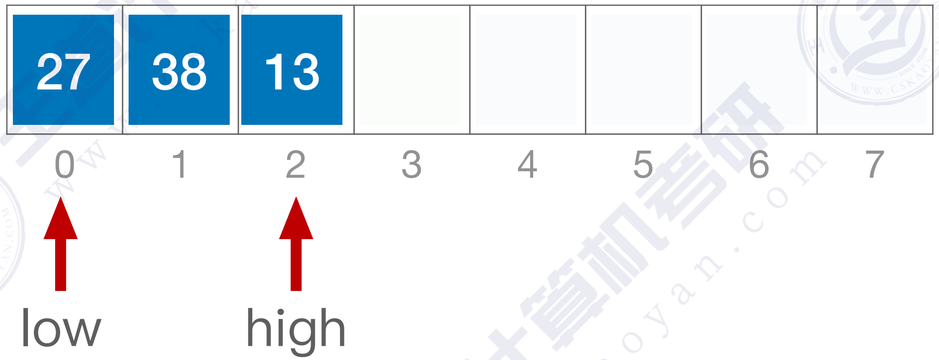
更小的元素都交换到左边
更大的元素都交换到右边



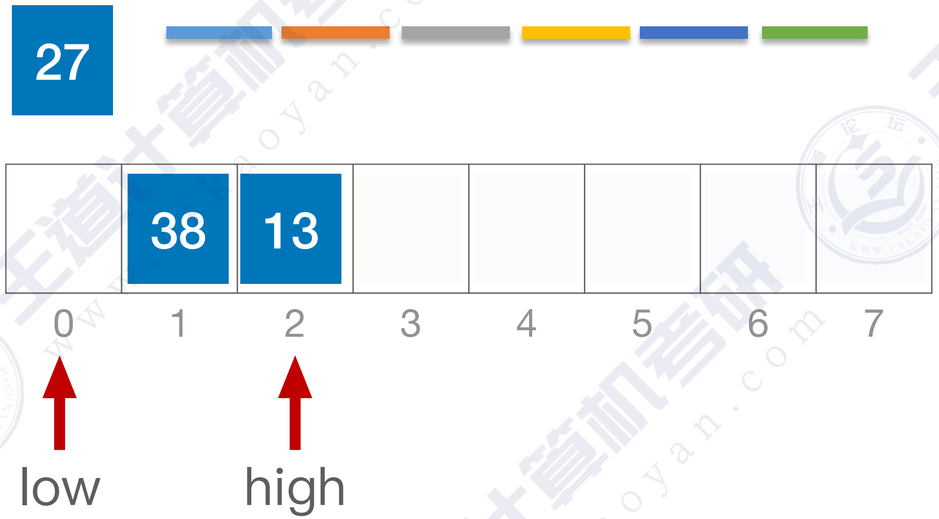
快速排序

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。

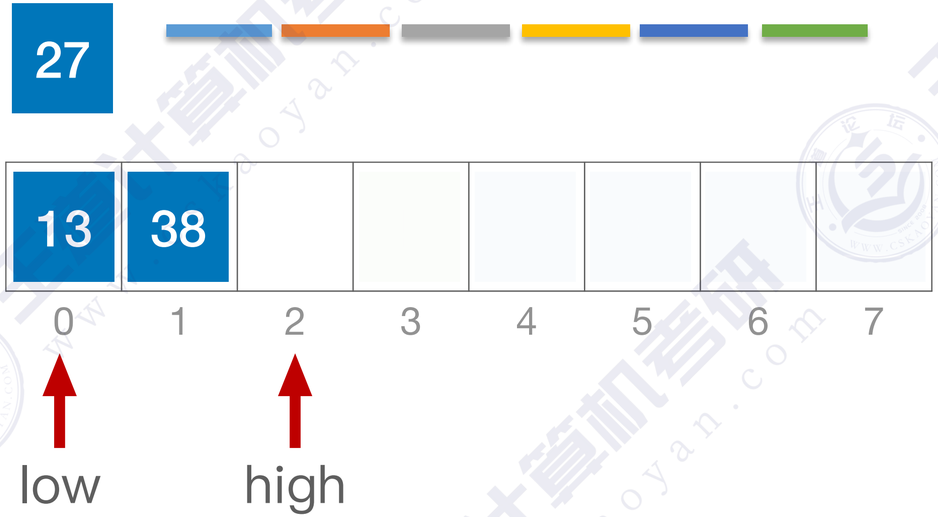
快速排序



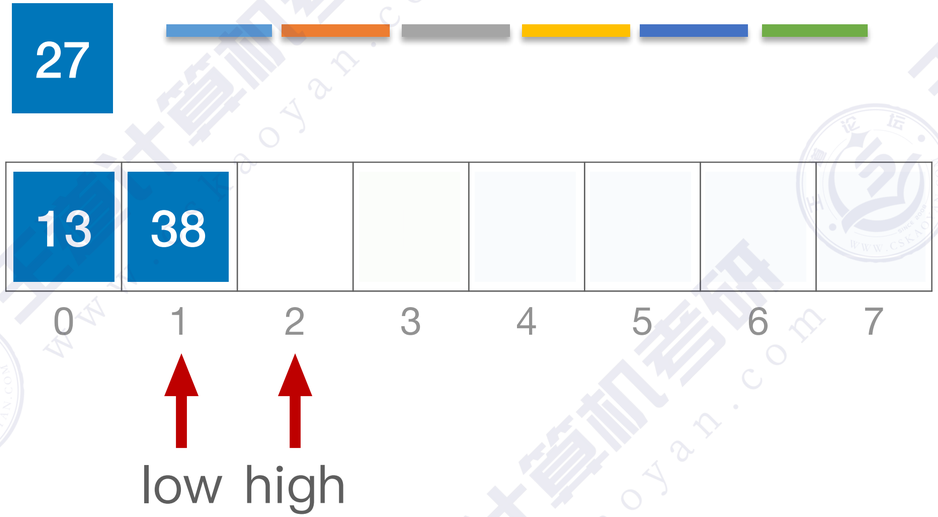
快速排序



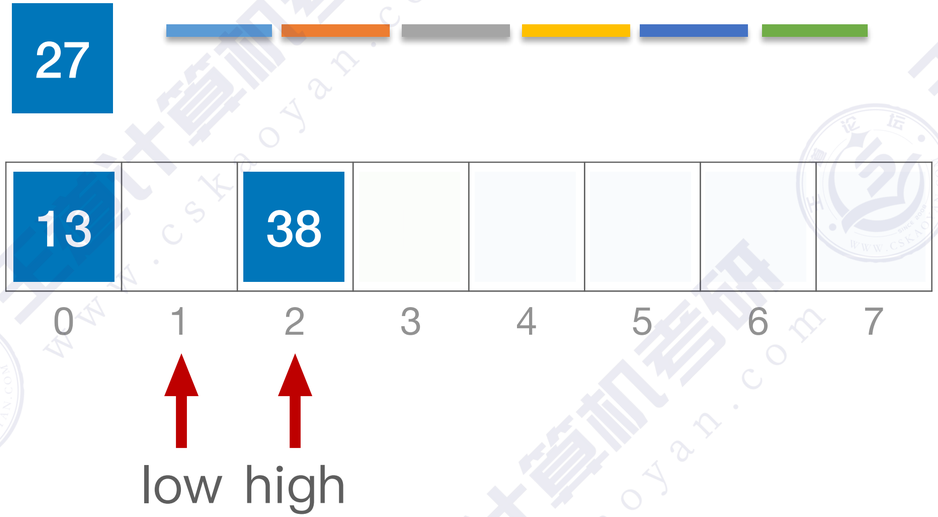
快速排序



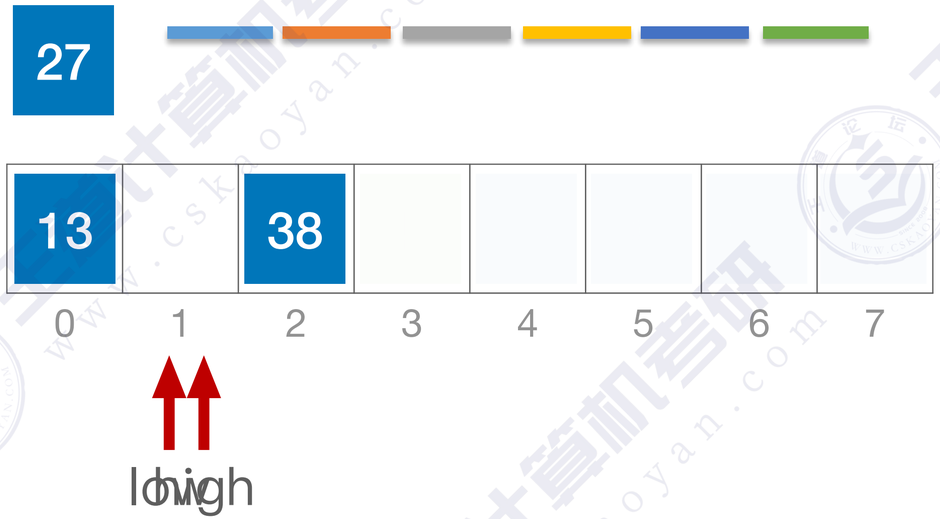
快速排序



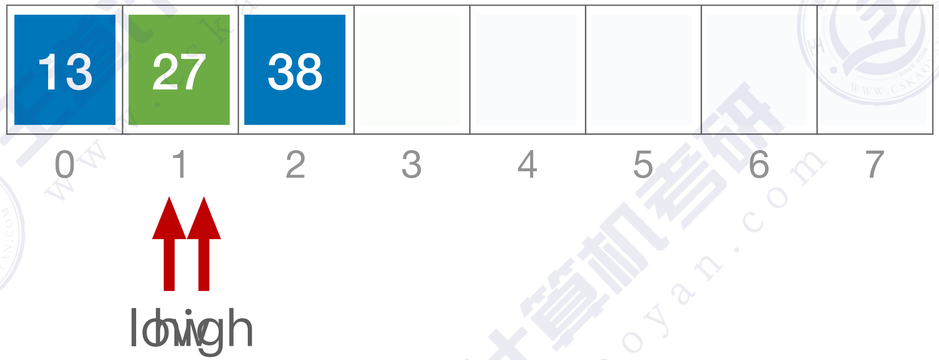
快速排序



快速排序

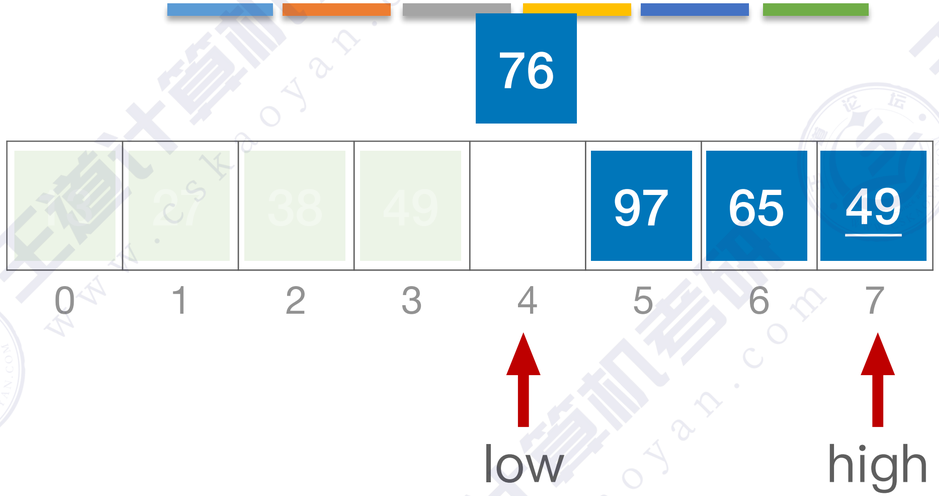


快速排序

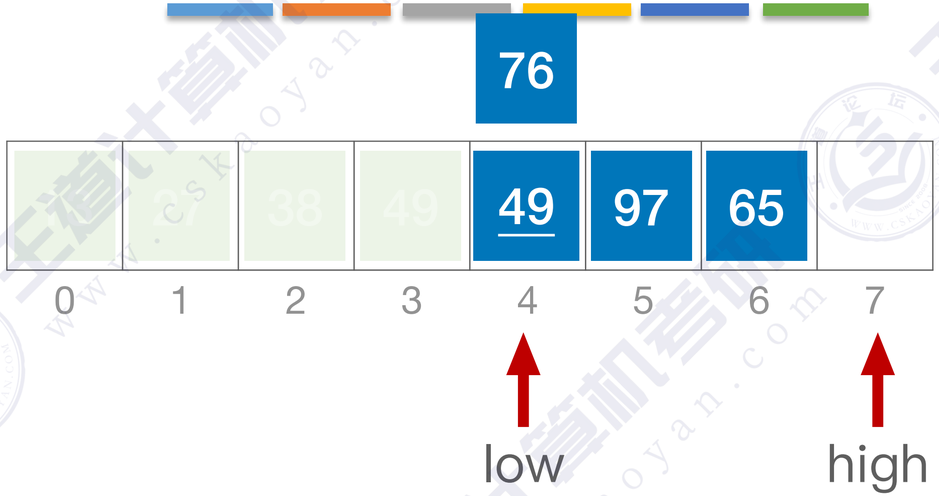


快速排序

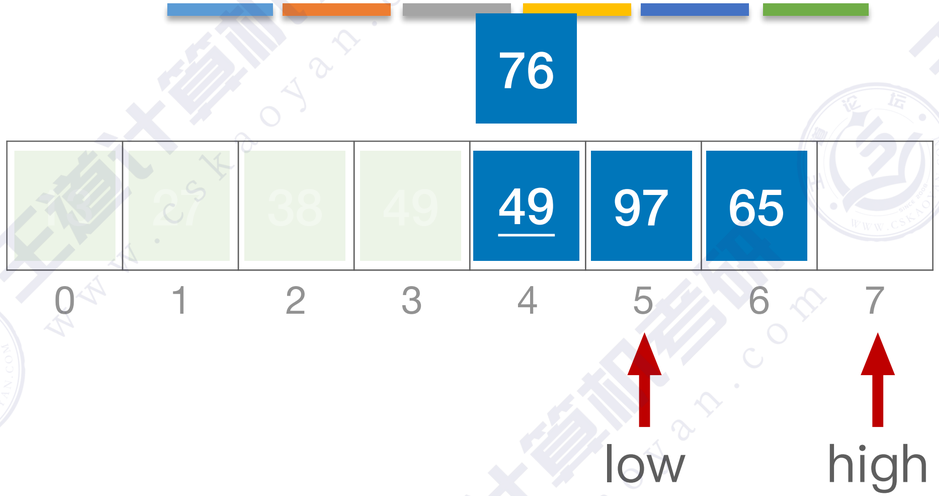
快速排序



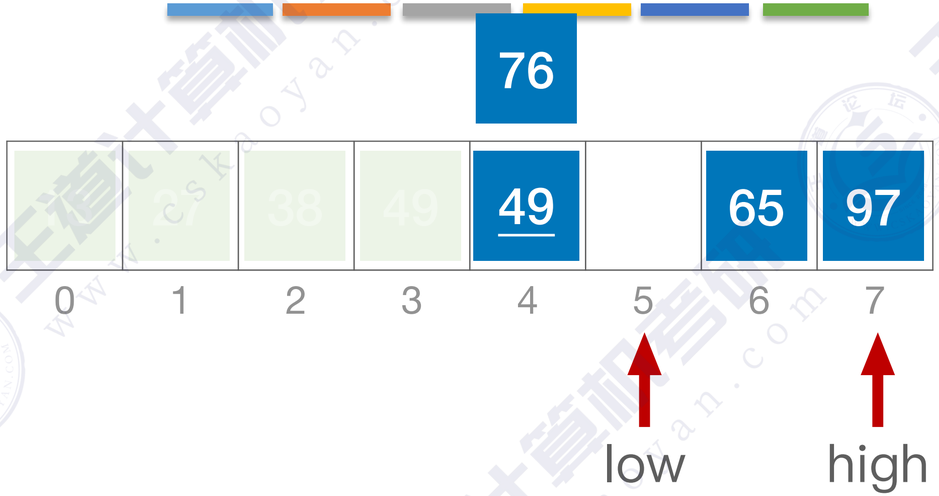
快速排序



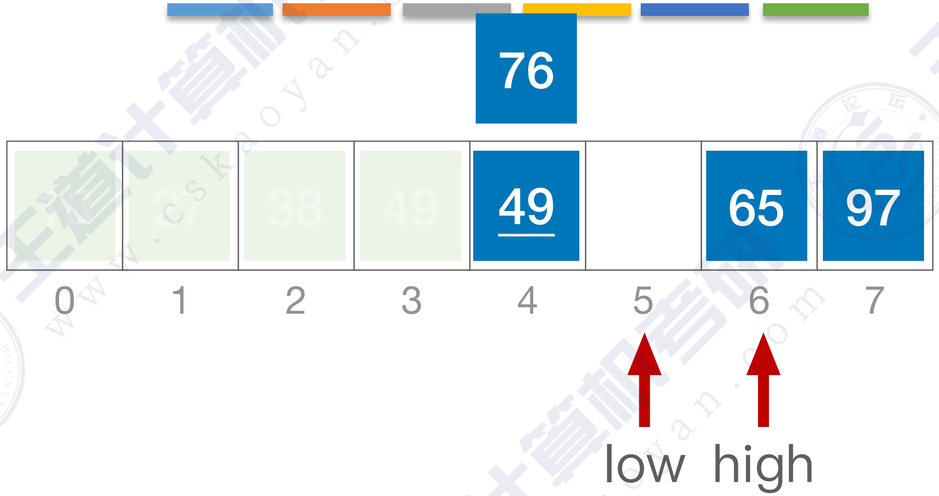
快速排序



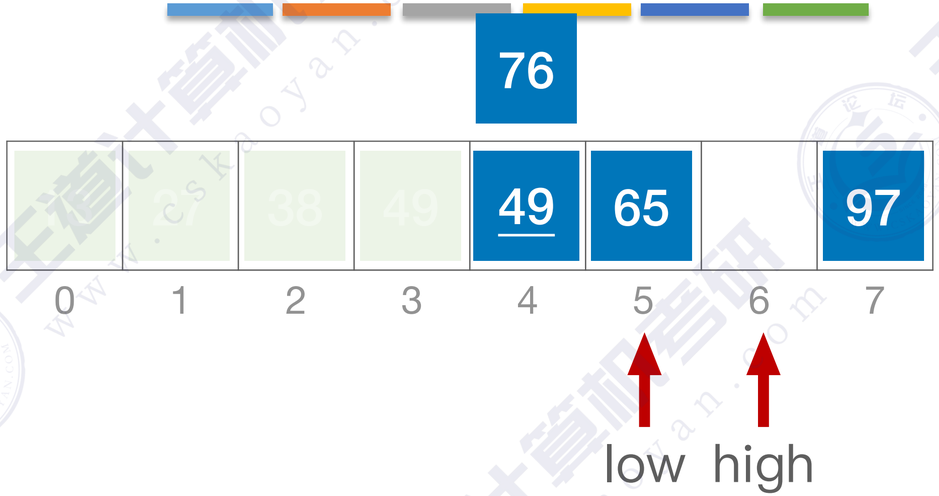
快速排序



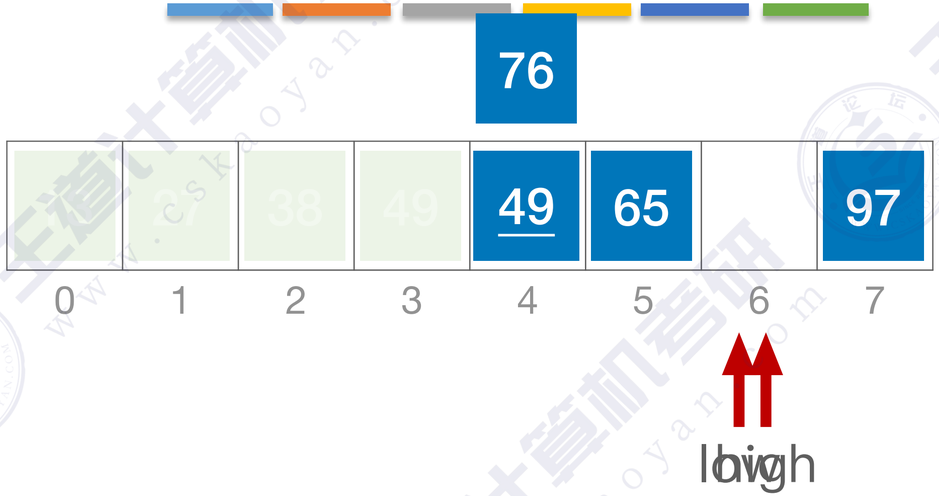
快速排序



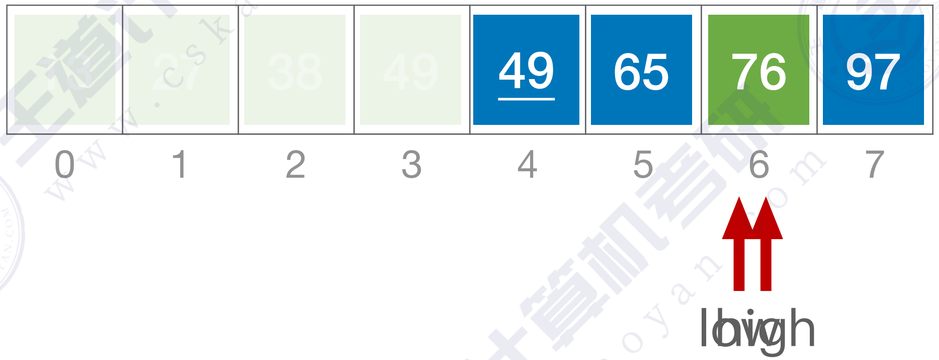
快速排序



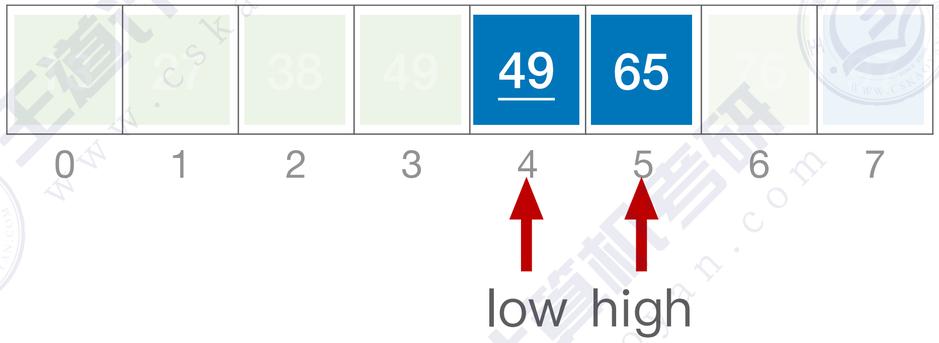
快速排序



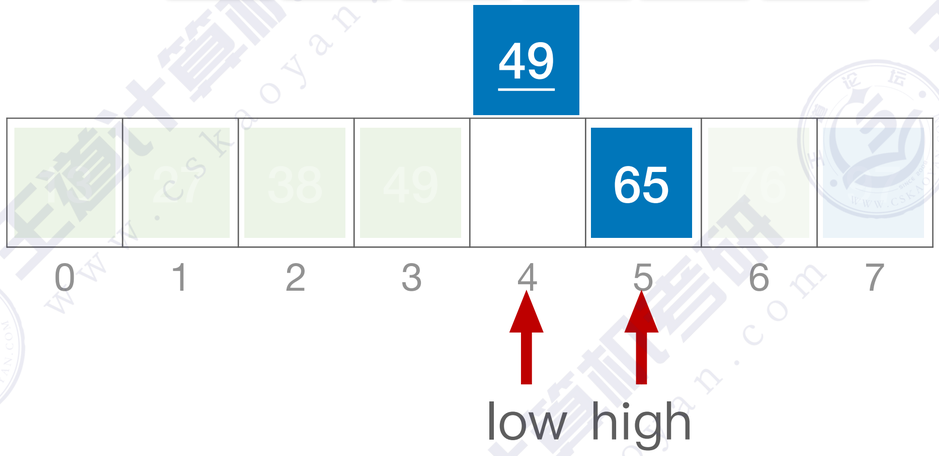
快速排序



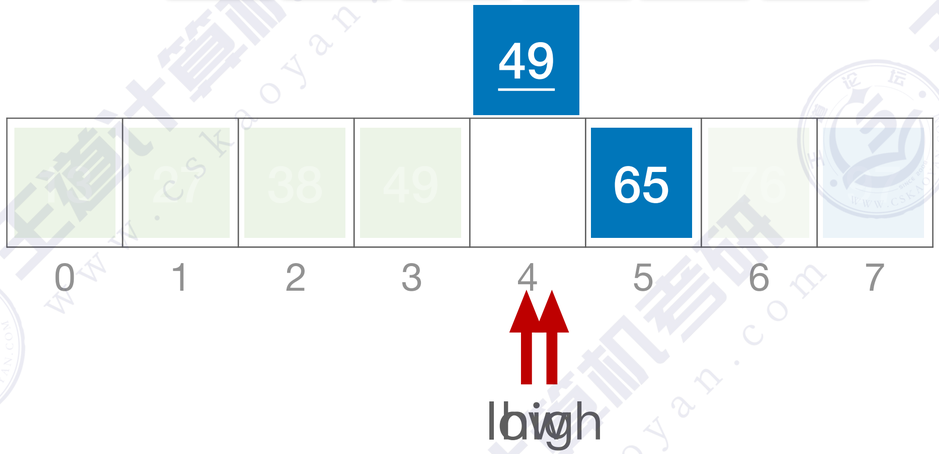
快速排序



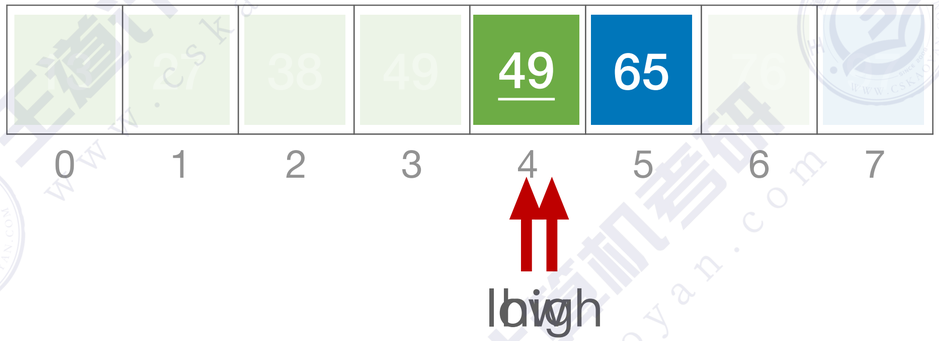
快速排序



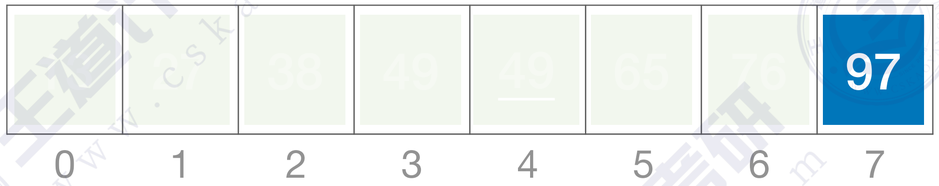
快速排序



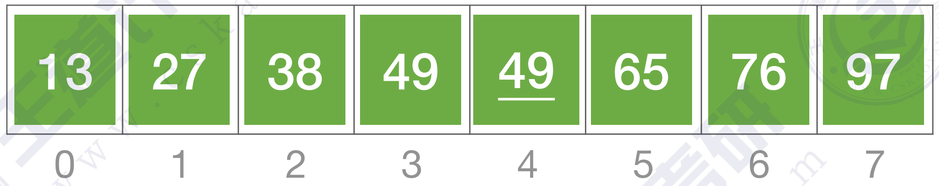
快速排序



快速排序

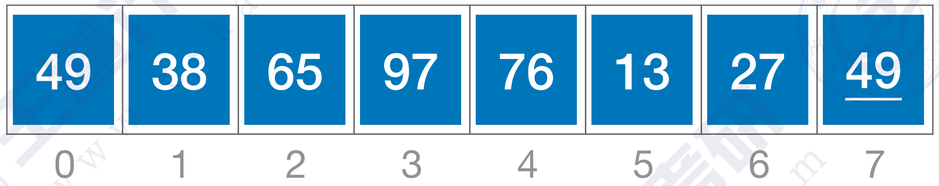


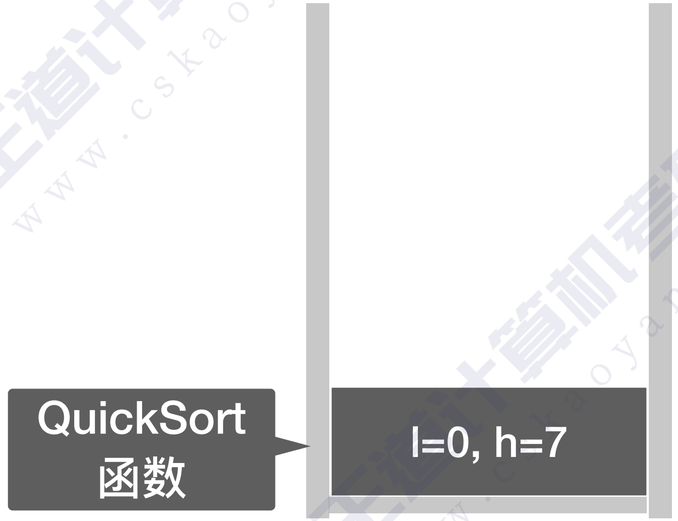
快速排序



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。

快速排序



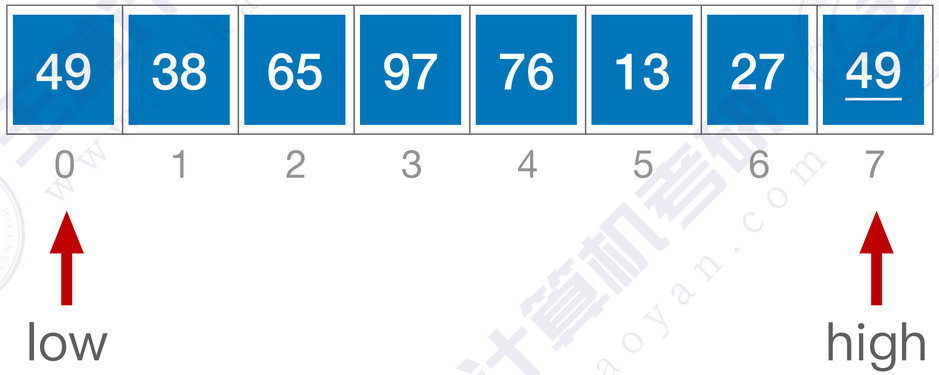


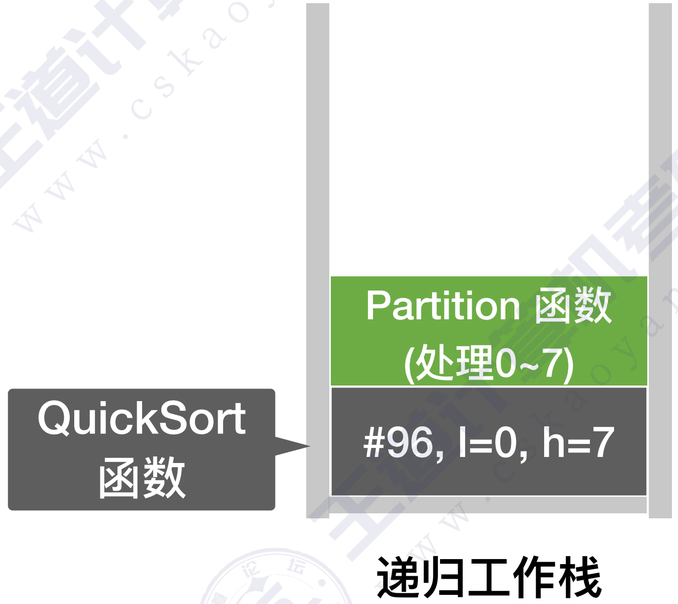
递归工作栈

```
//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 }
```

快速排序

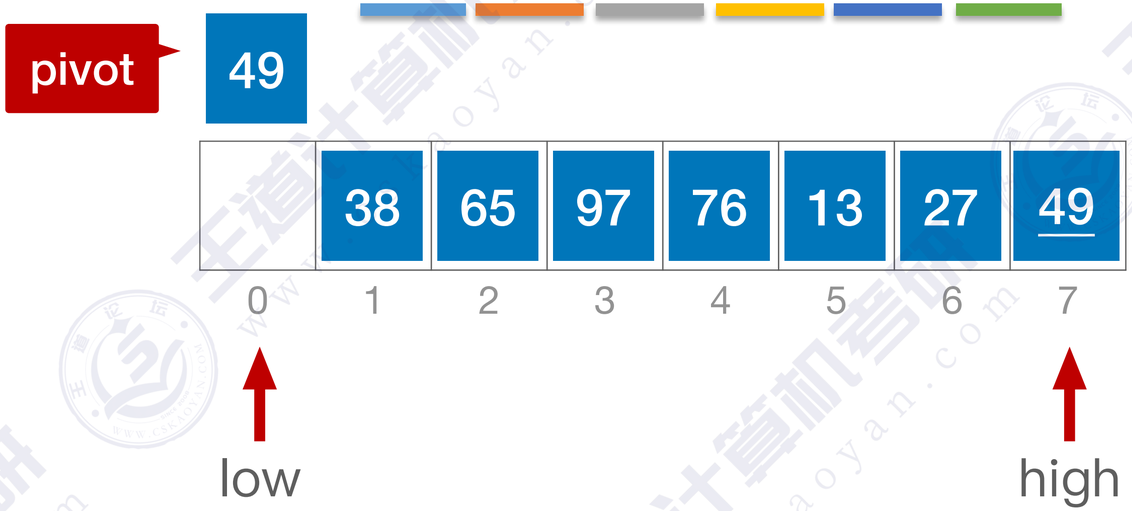





```
//用第一个元素将待排序序列划分成左右两个部分
int Partition(int A[],int low,int high){
    int pivot=A[low];          //第一个元素作为枢轴
    while(low<high){           //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];        //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        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 }
```

快速排序

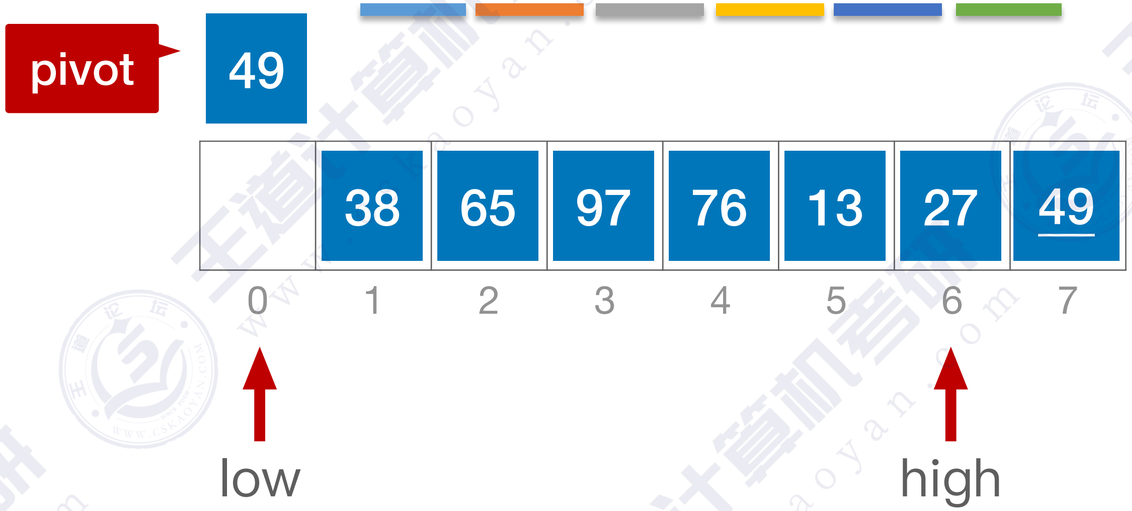


递归工作栈

```
//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 }
```

快速排序

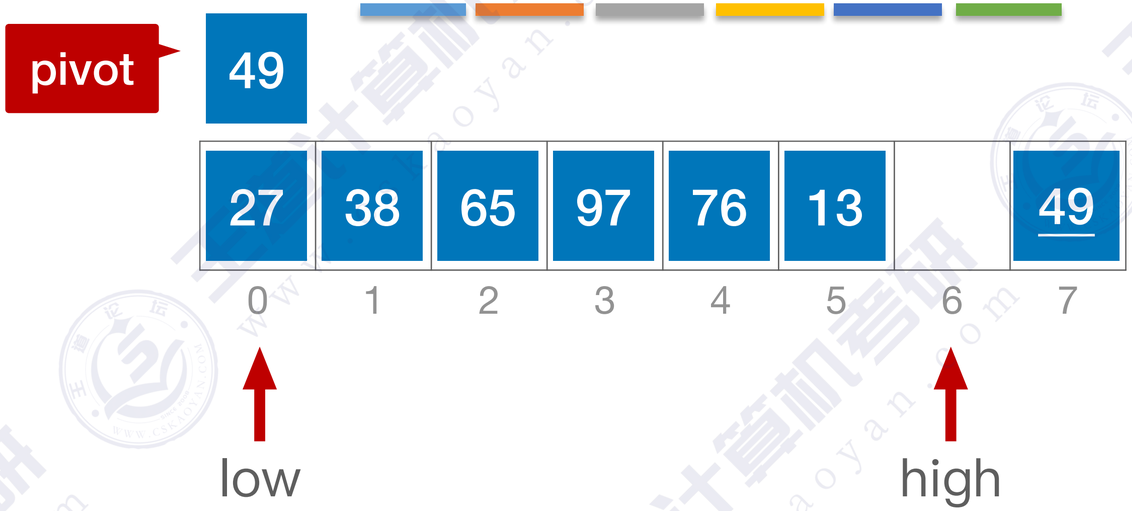


递归工作栈

```
//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 }
```

快速排序

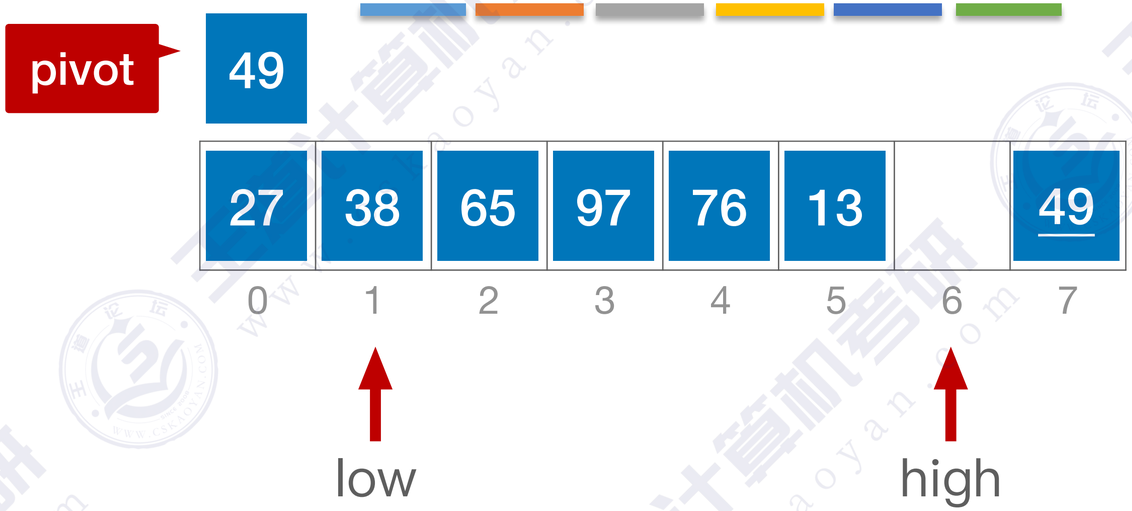


递归工作栈

```
//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 }
```

快速排序

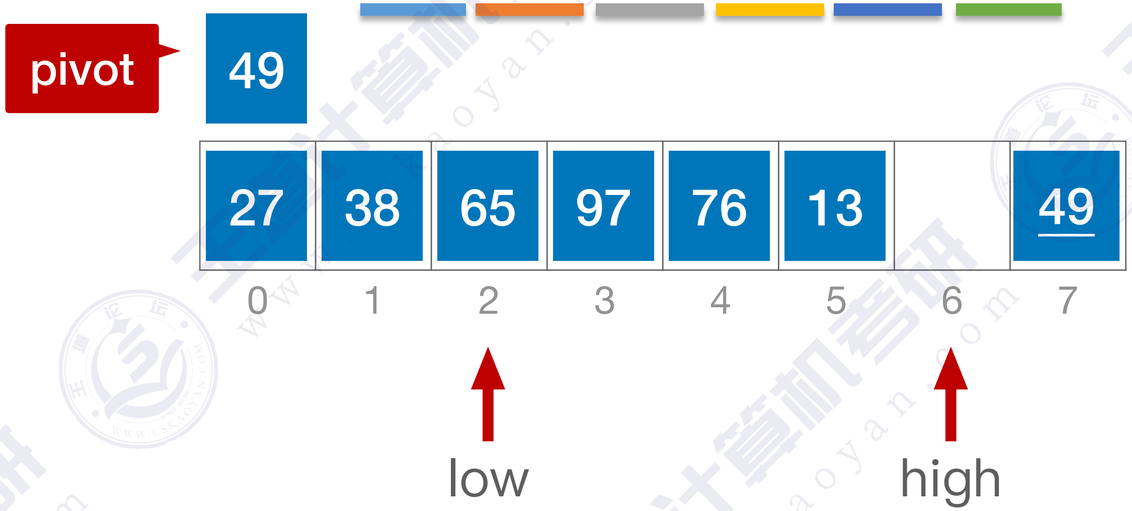


递归工作栈

```
//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 }
```

快速排序

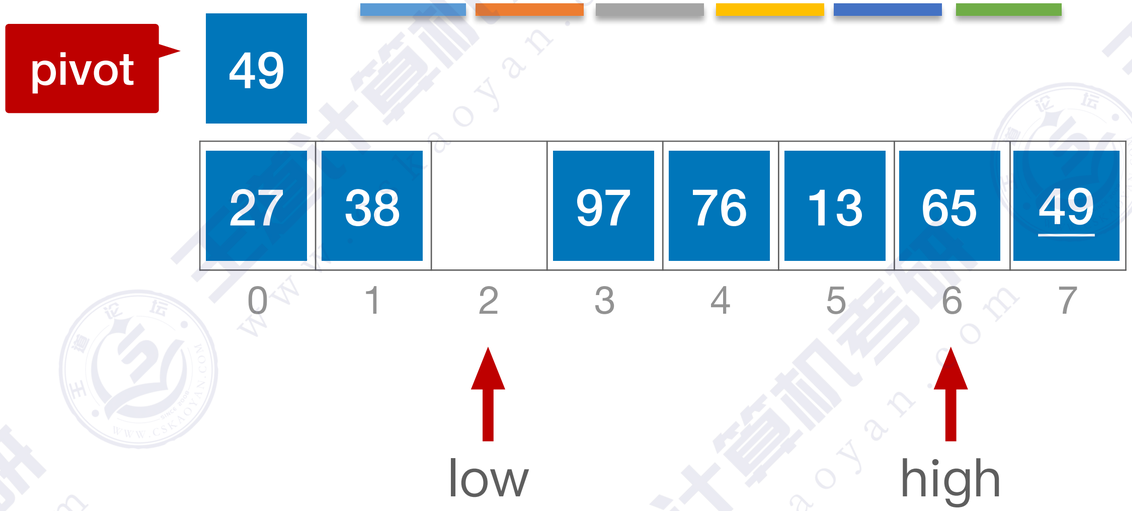


递归工作栈

```
//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 }
```

快速排序

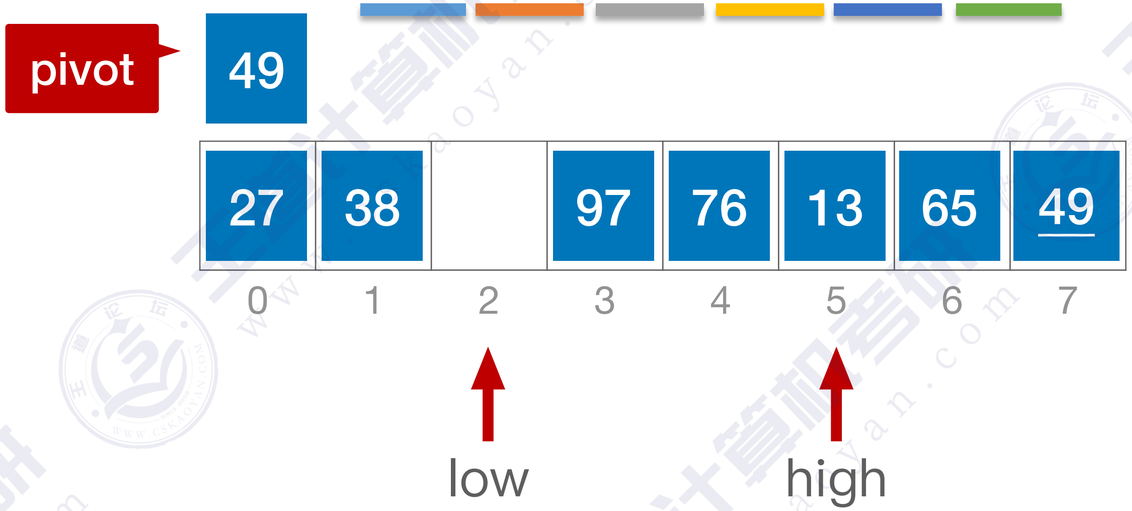


递归工作栈

```
//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 }
```

快速排序

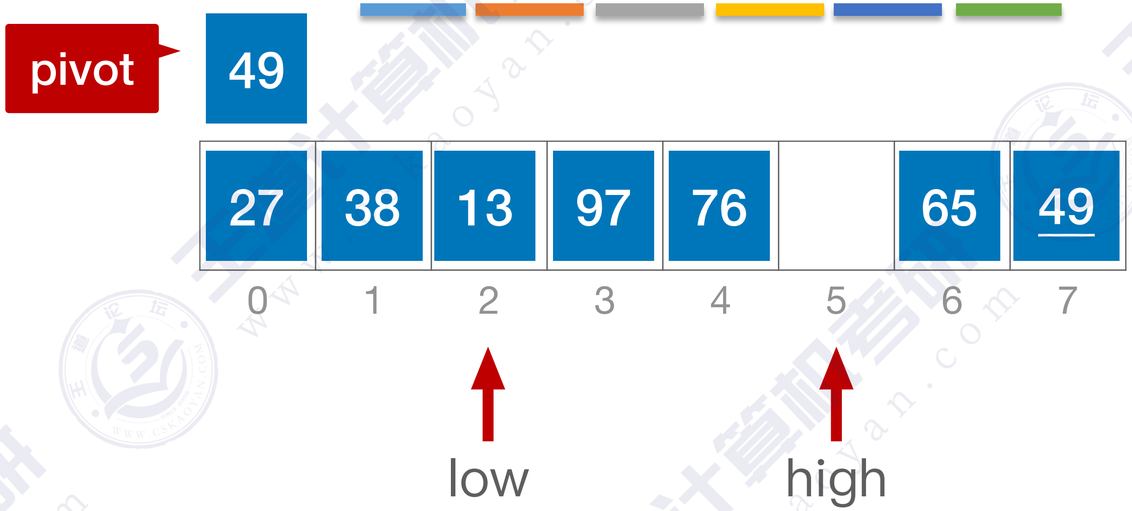


递归工作栈

```
//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 }
```

快速排序

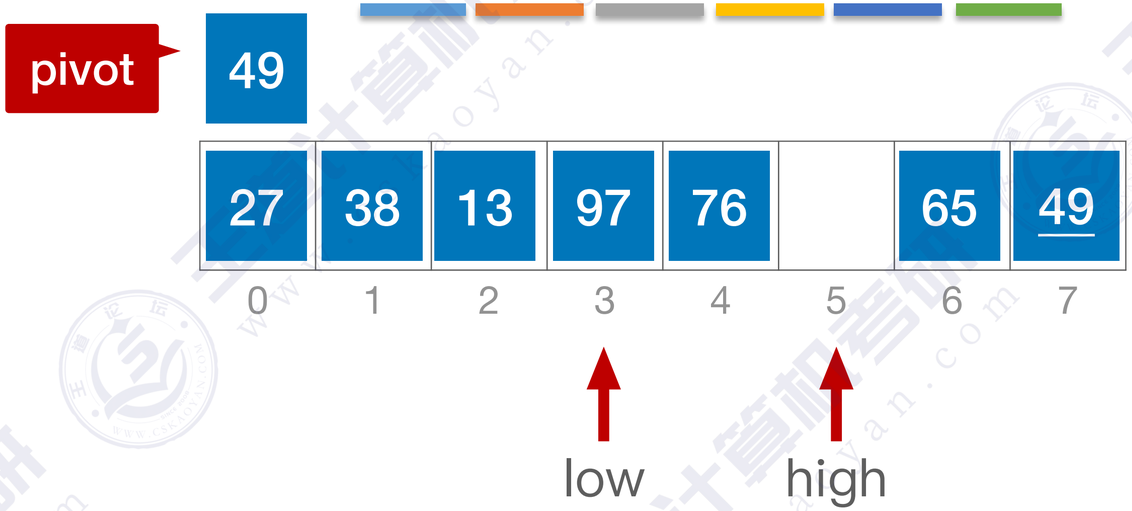


递归工作栈

```
//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 }
```

快速排序

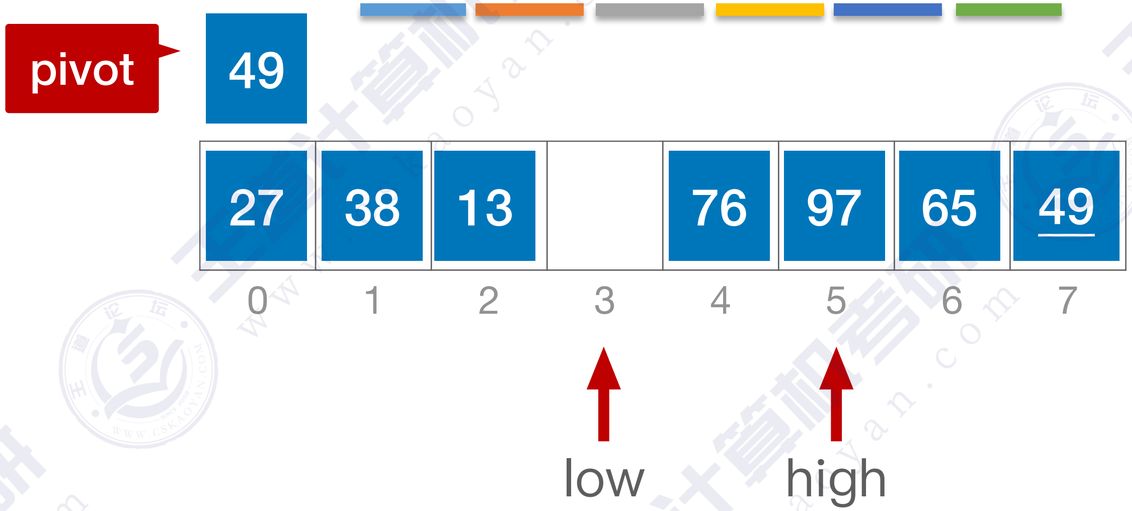


递归工作栈

```
//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 }
```

快速排序

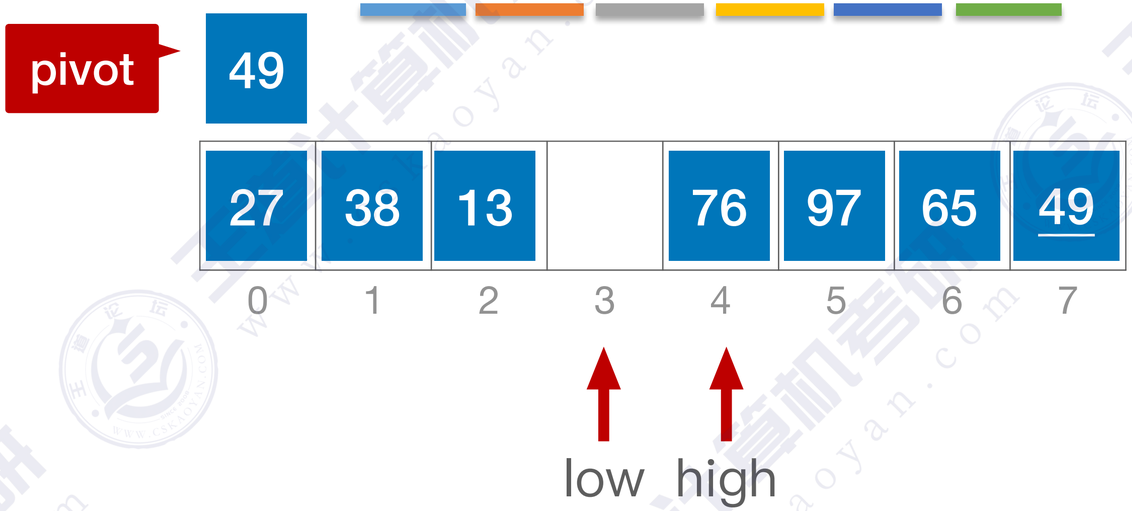


递归工作栈

```
//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 }
```

快速排序

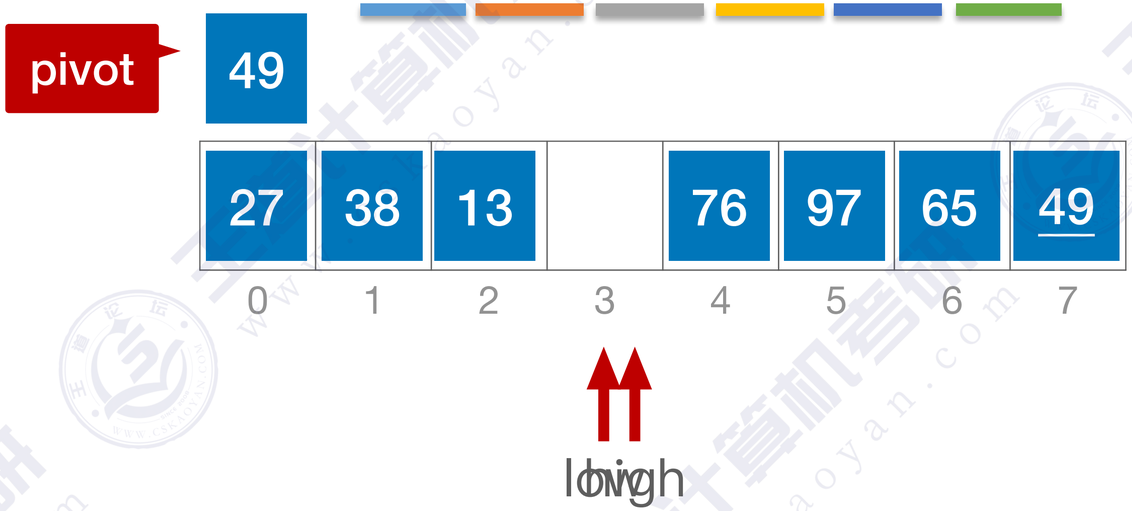


递归工作栈

```
//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 }
```

快速排序

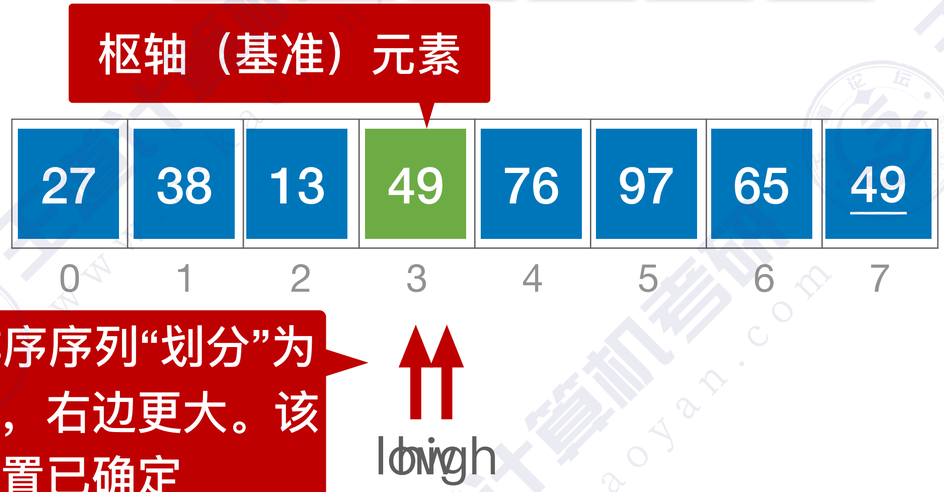


递归工作栈

```
//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 }
```

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 }
```

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 }
```

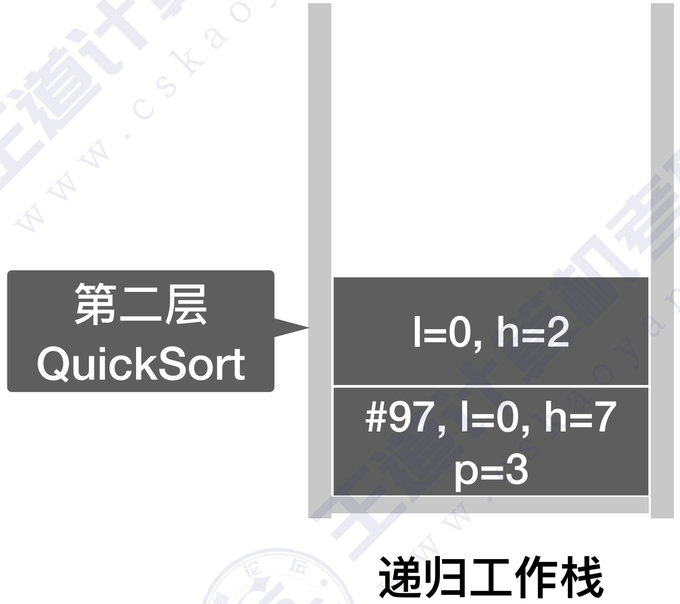
快速排序

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 }
```

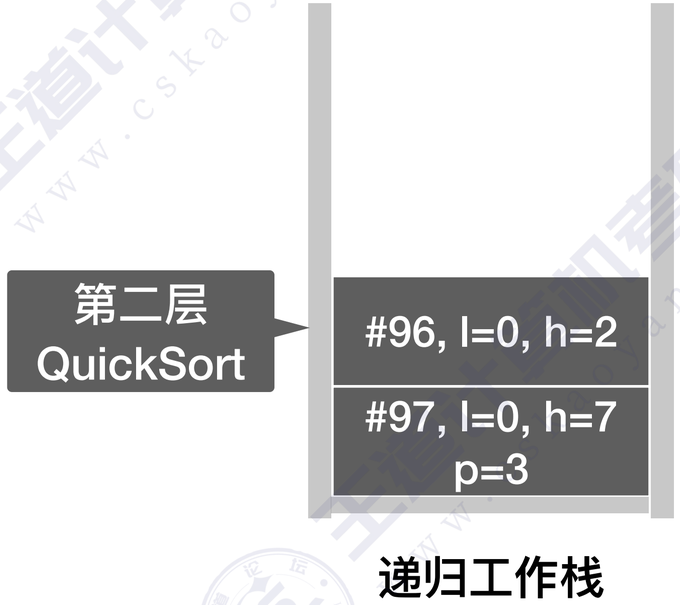
快速排序




```
//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 }
```

快速排序

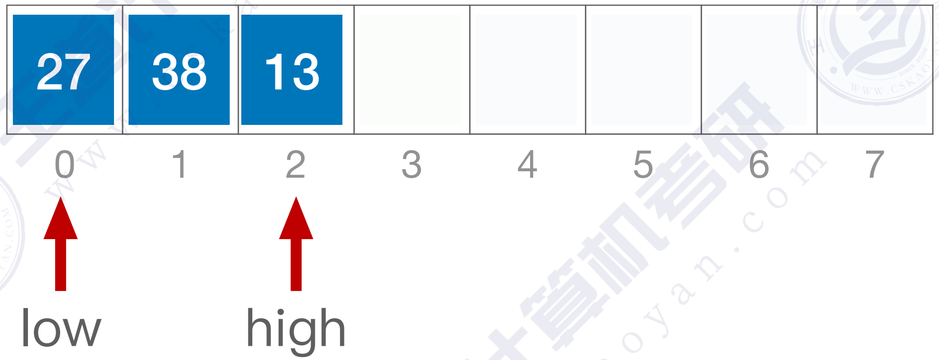


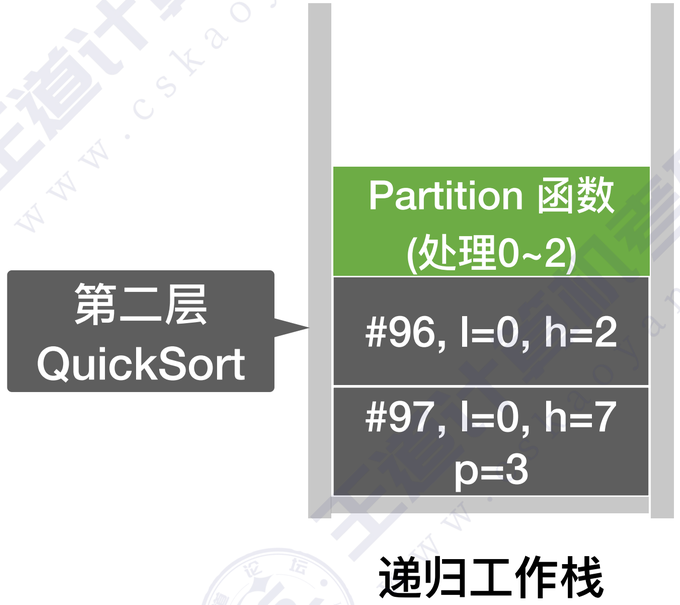
递归工作栈

```
//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 }
```

快速排序

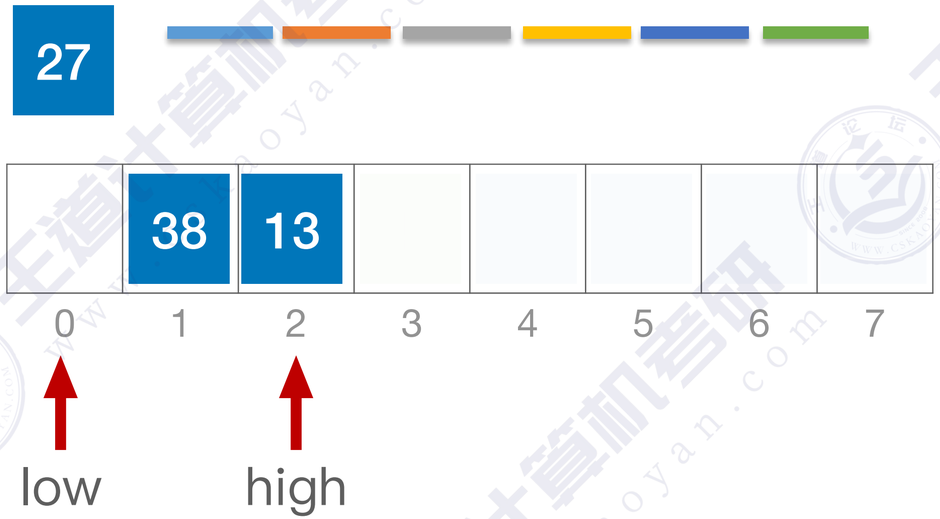


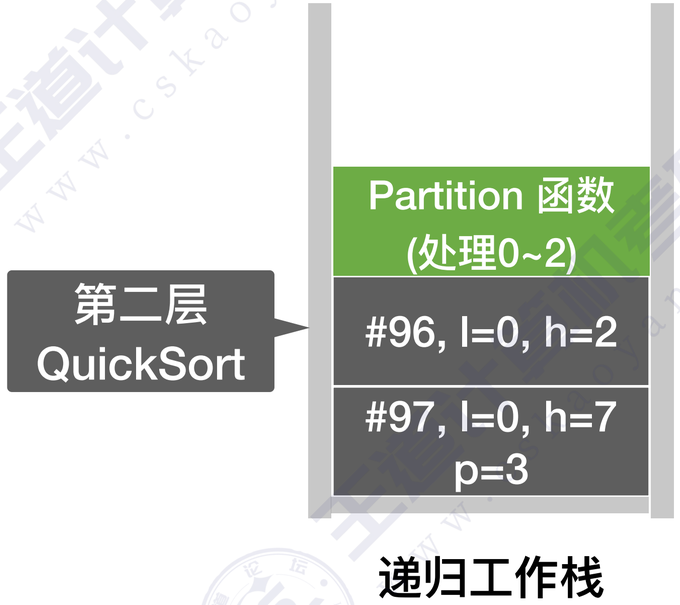



```c
//用第一个元素将待排序序列划分成左右两个部分
int Partition(int A[],int low,int high){
    int pivot=A[low];          //第一个元素作为枢轴
    while(low<high){           //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];        //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        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
}
```

```c
//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
    }
}
```

快速排序



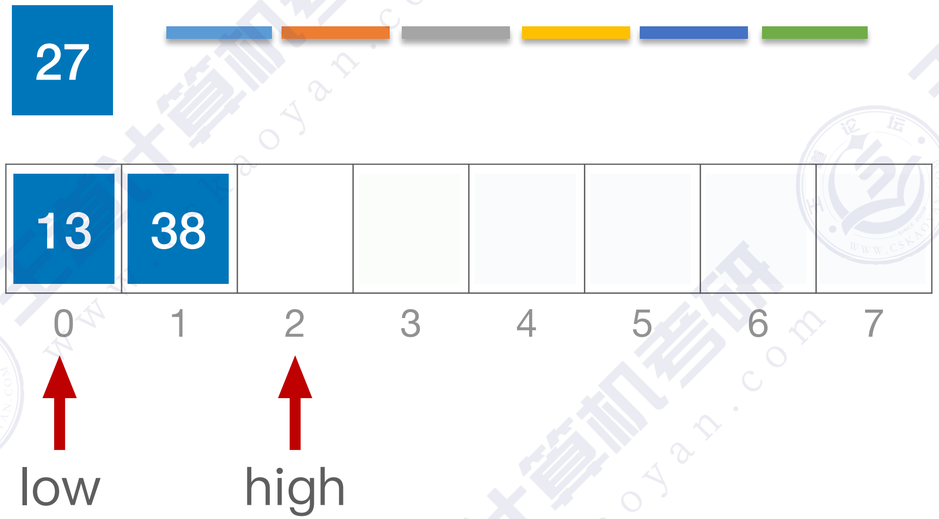


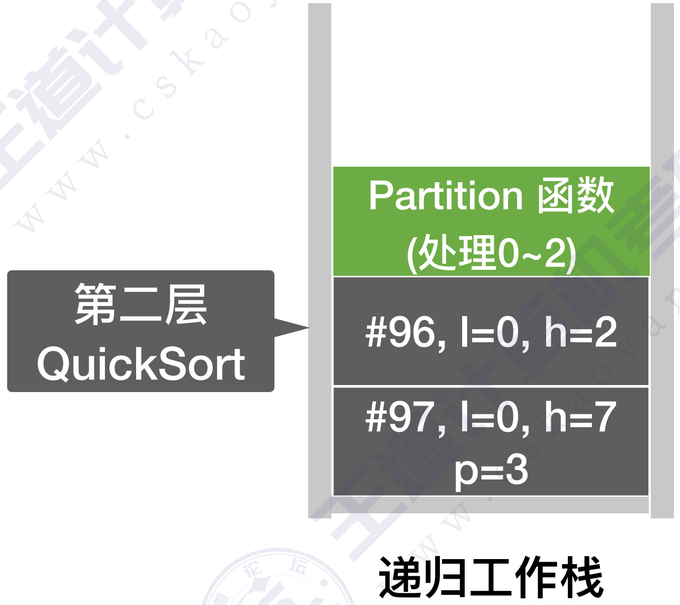
递归工作栈

```
//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 }
```

快速排序



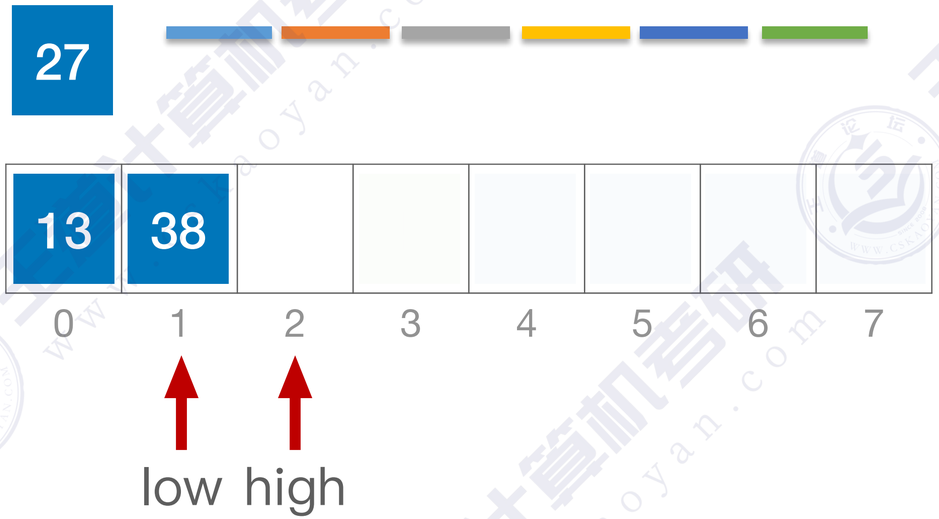


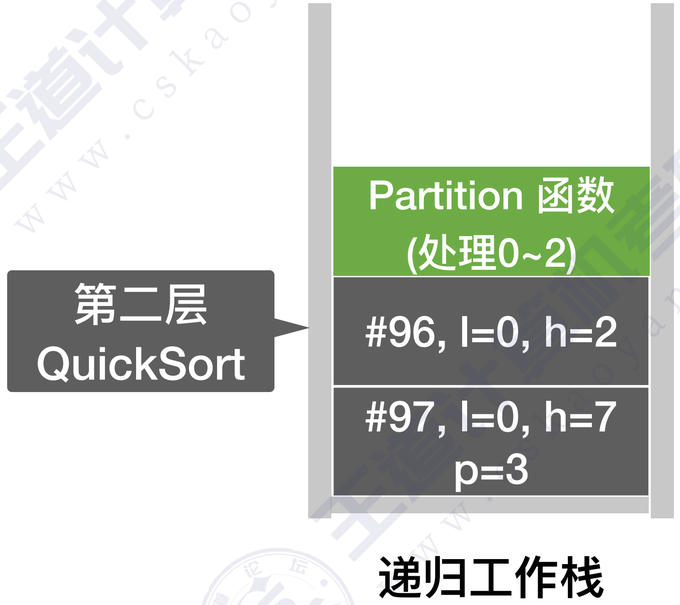
递归工作栈

```
//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 }
```

快速排序



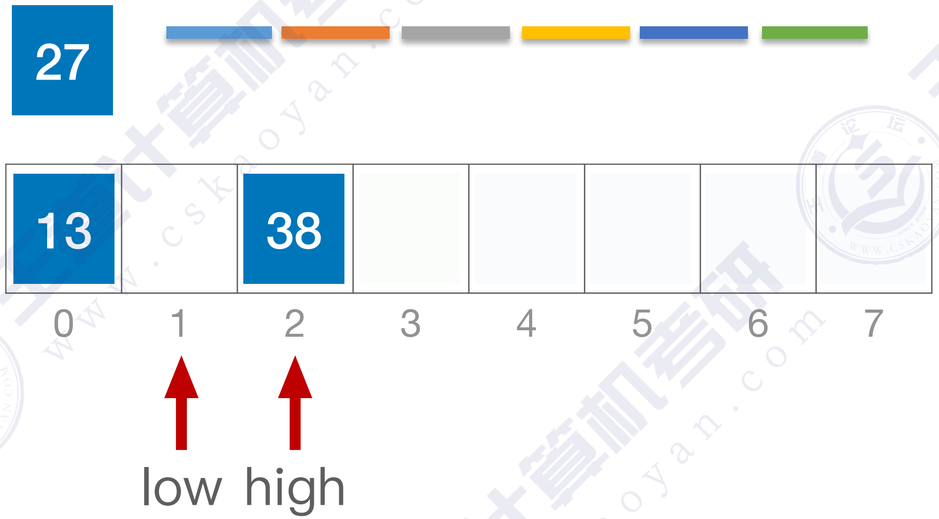


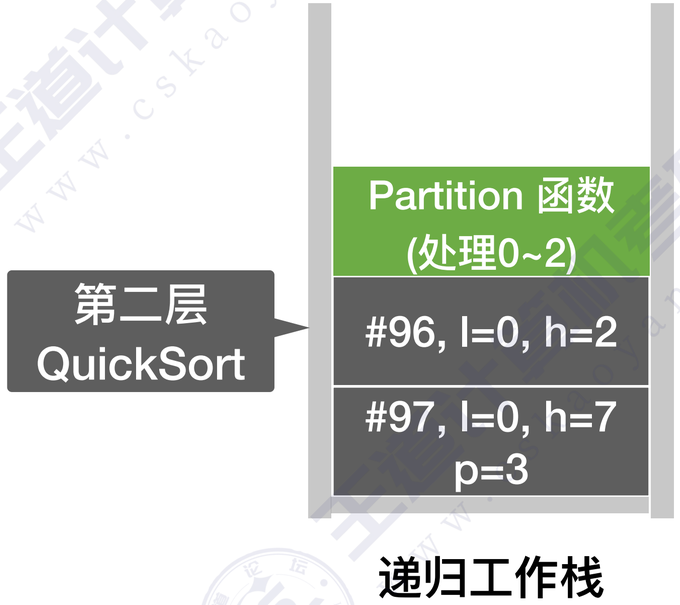
递归工作栈

```
//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 }
```

快速排序



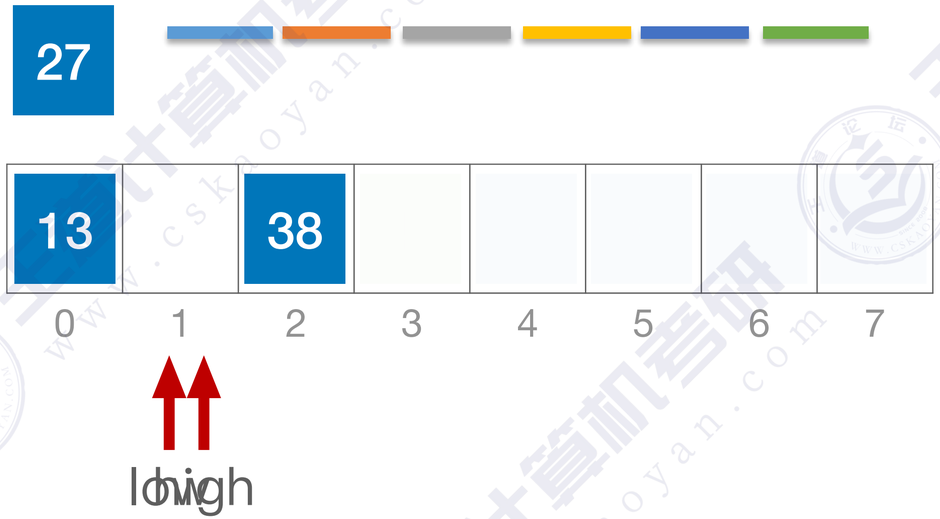


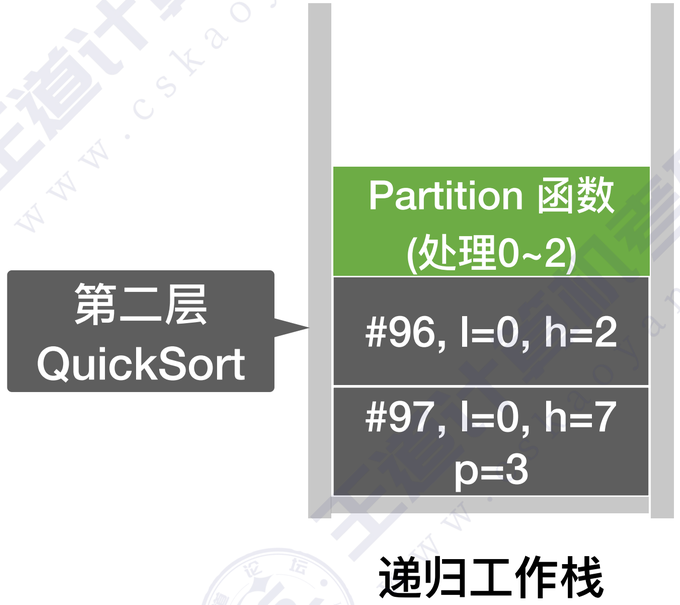
递归工作栈

```
//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 }
```

快速排序



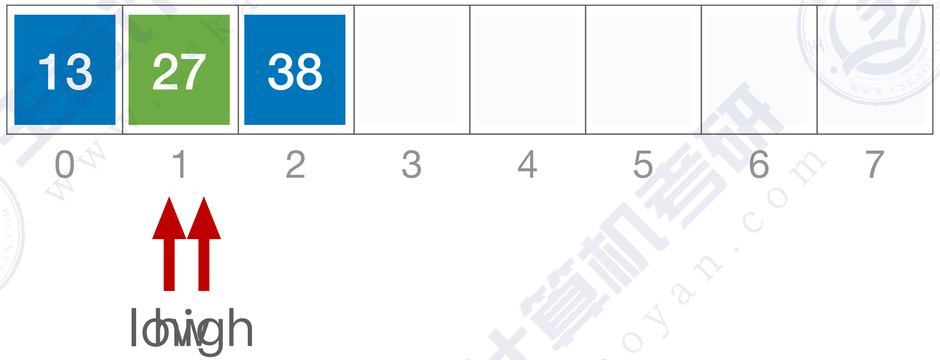


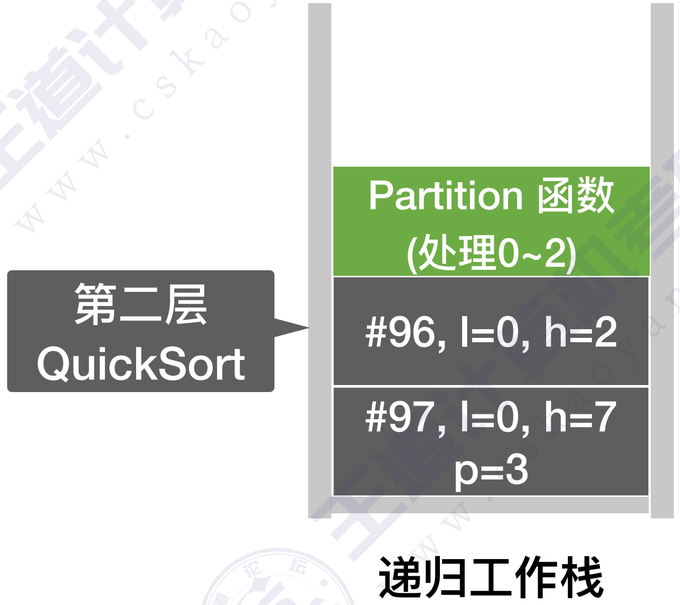
递归工作栈

```
//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 }
```

快速排序



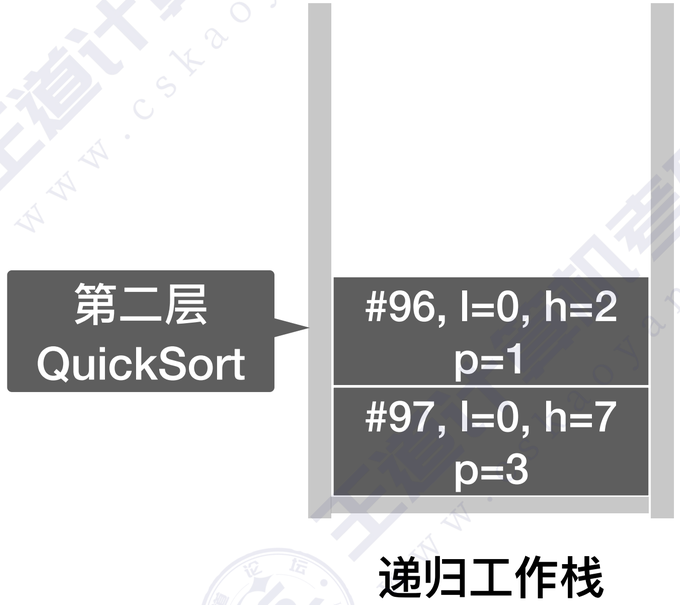


递归工作栈

```
//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 }
```

快速排序

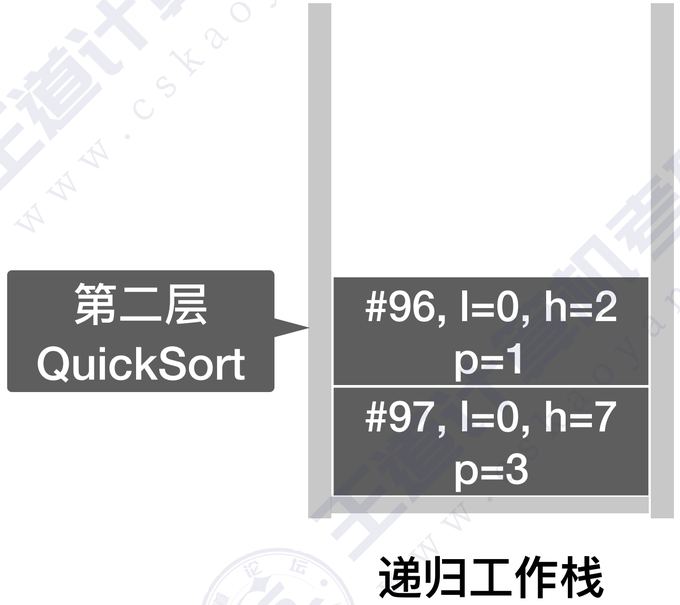


递归工作栈

```
//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 }
```

快速排序

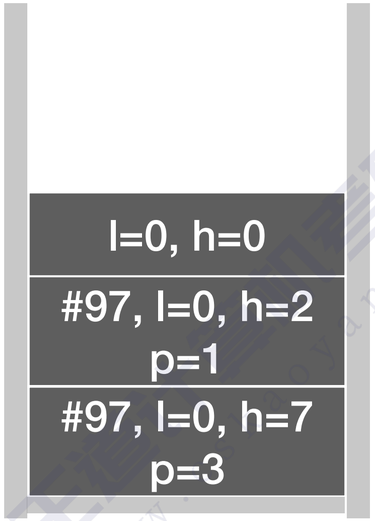



```
//用第一个元素将待排序序列划分成左右两个部分
int Partition(int A[],int low,int high){
    int pivot=A[low];          //第一个元素作为枢轴
    while(low<high){           //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];        //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        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 }
```

快速排序

第三层
QuickSort

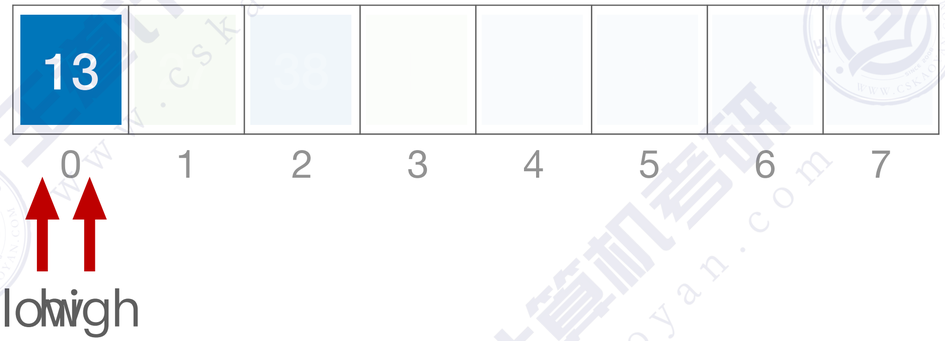


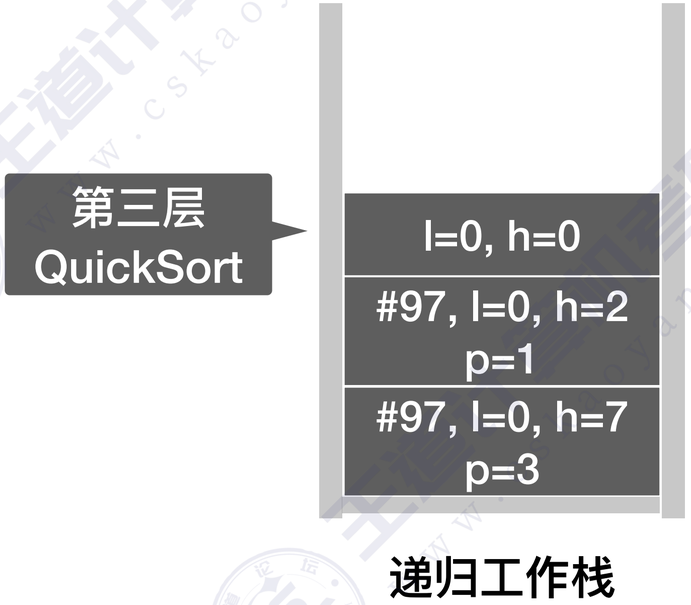
递归工作栈

```
//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 }
```

快速排序



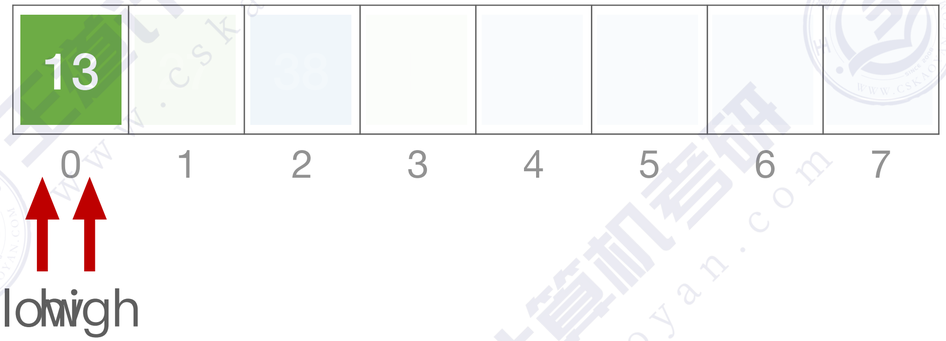


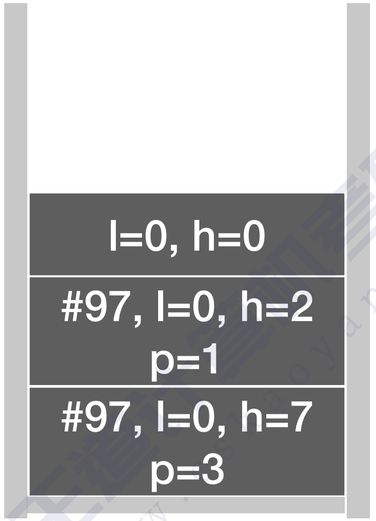
递归工作栈

```
//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 }
```

快速排序



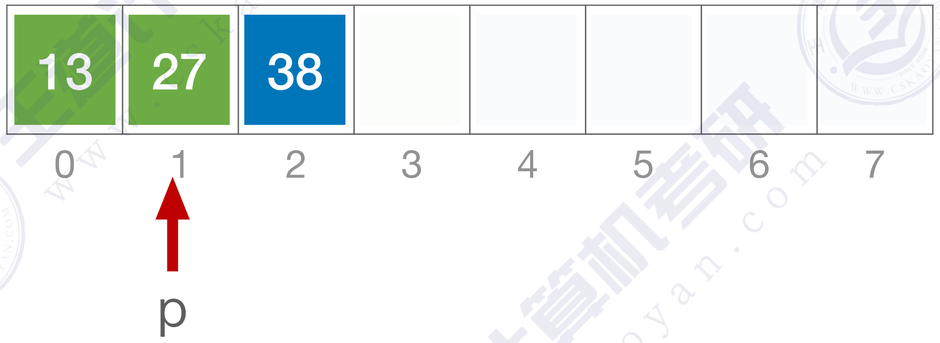


递归工作栈

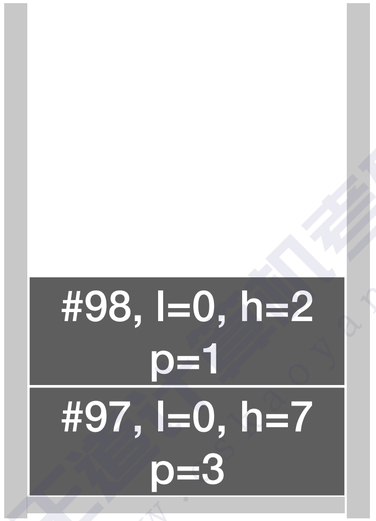
```
//用第一个元素将待排序序列划分成左右两个部分
int Partition(int A[],int low,int high){
    int pivot=A[low];          //第一个元素作为枢轴
    while(low<high){           //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];        //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        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 }
```

快速排序



第二层 QuickSort

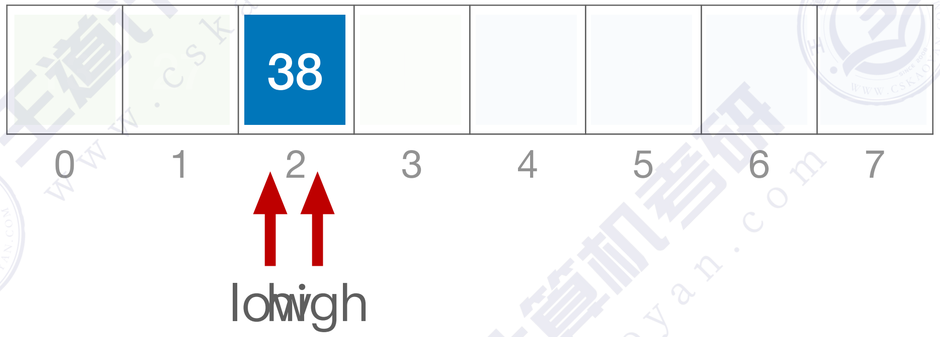


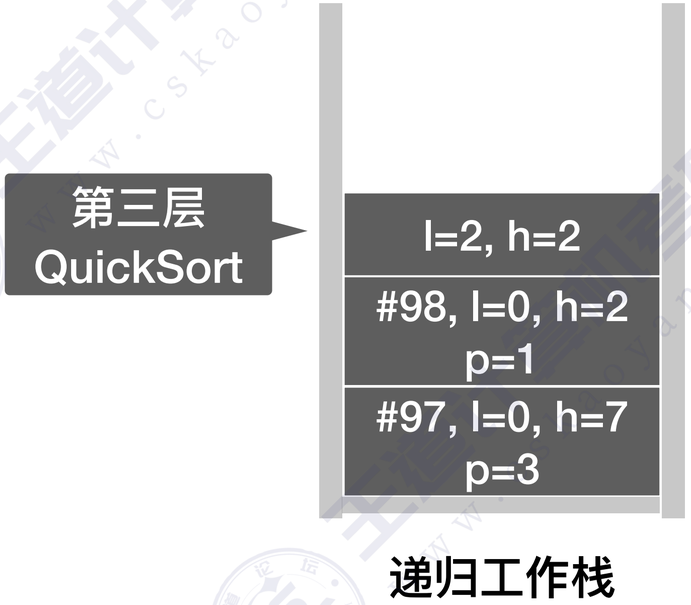
递归工作栈

```
//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 }
```

快速排序





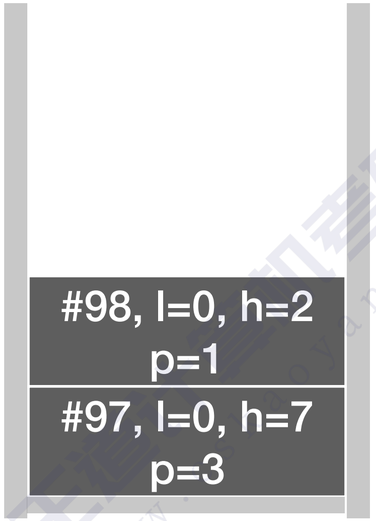
递归工作栈

```
//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 }
```

快速排序

第二层 QuickSort

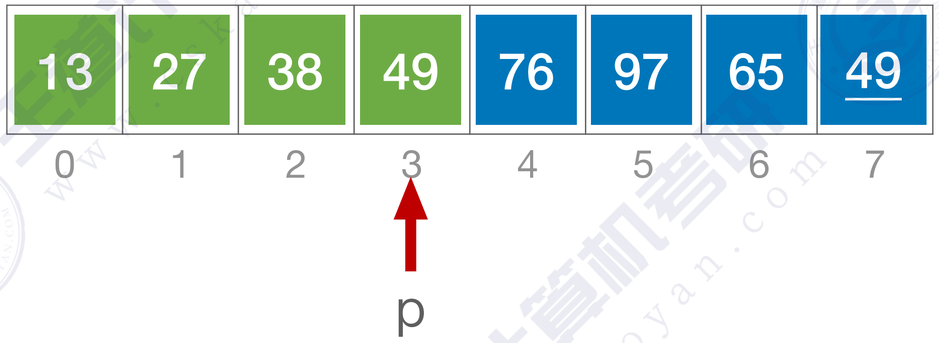


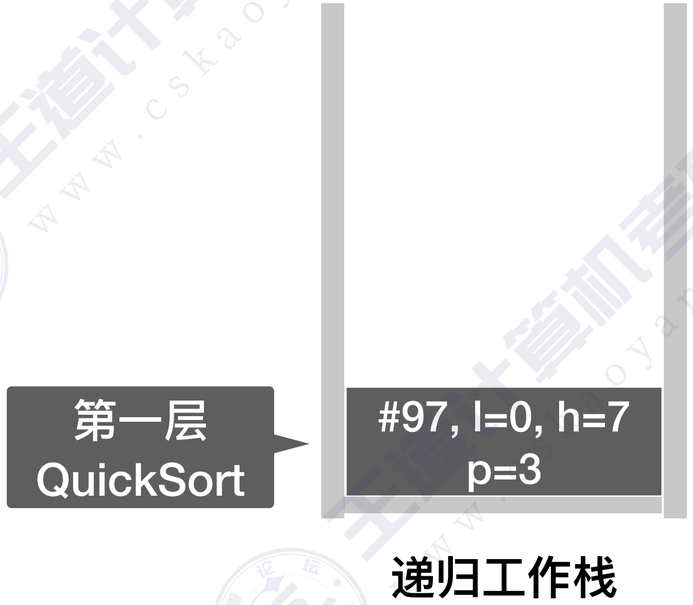
递归工作栈

```
//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 }
```

快速排序

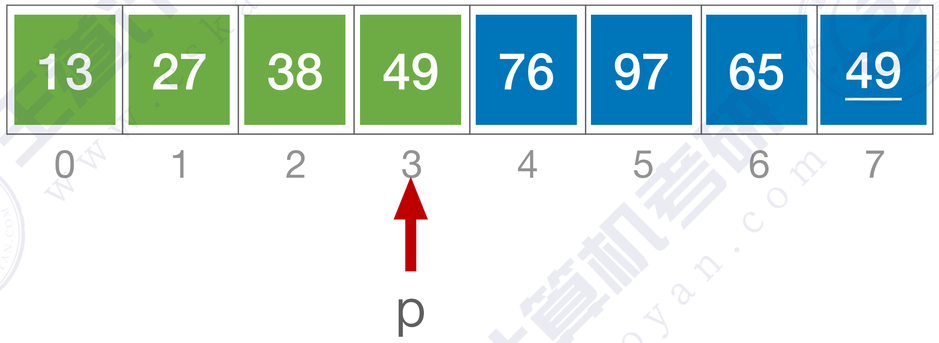


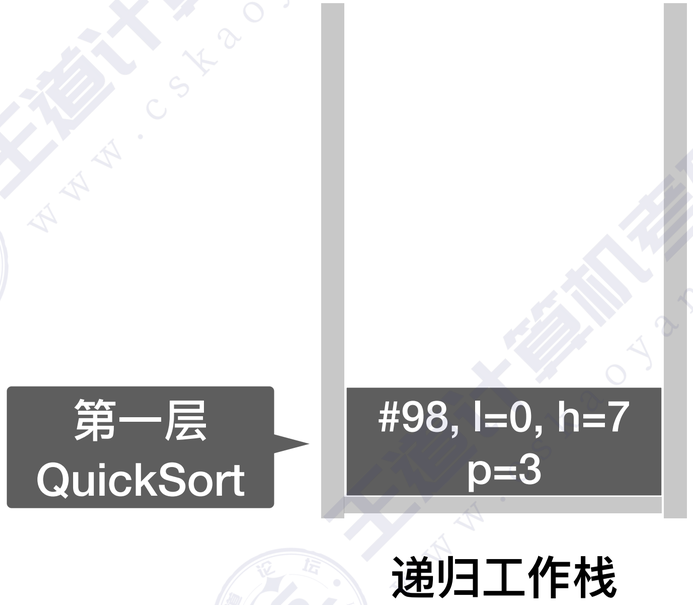



```
//用第一个元素将待排序序列划分成左右两个部分
int Partition(int A[],int low,int high){
    int pivot=A[low];          //第一个元素作为枢轴
    while(low<high){           //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];        //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        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 }
```

快速排序



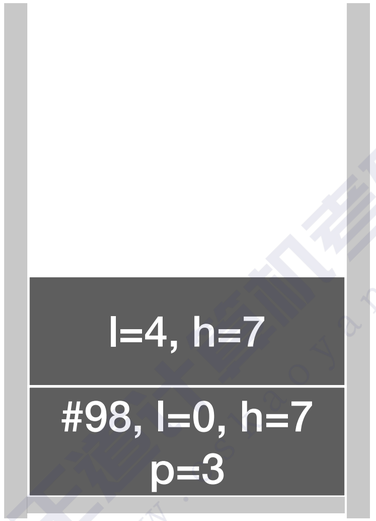



```
//用第一个元素将待排序序列划分成左右两个部分
int Partition(int A[],int low,int high){
    int pivot=A[low];          //第一个元素作为枢轴
    while(low<high){           //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];        //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        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 }
```

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 }
```

快速排序

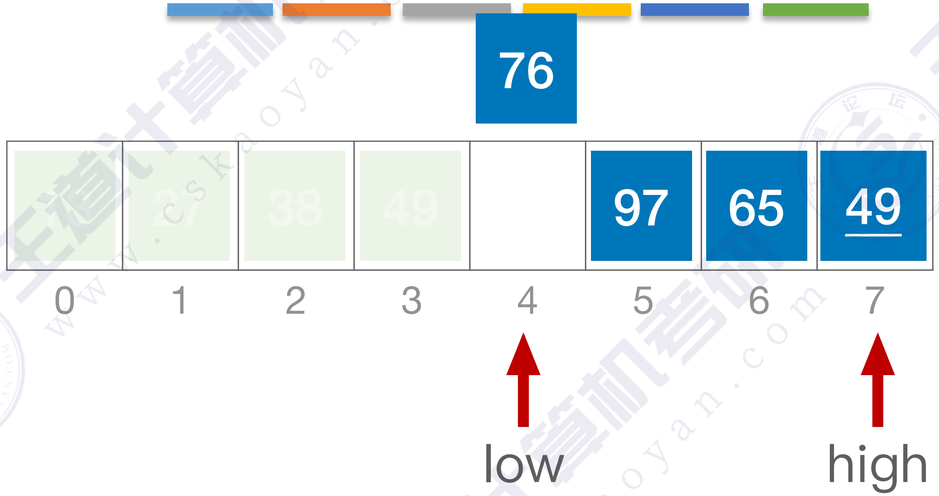
第二层 QuickSort

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 }
```

快速排序



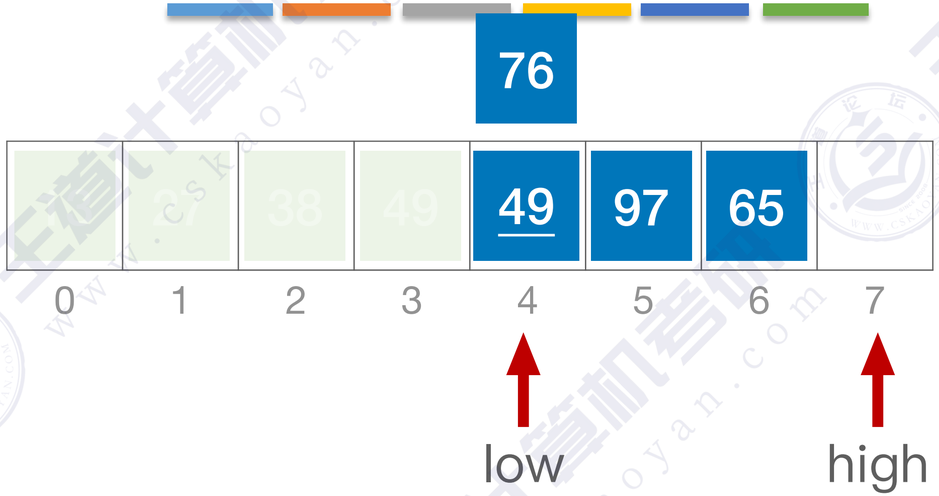
第二层 QuickSort

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 }
```

快速排序



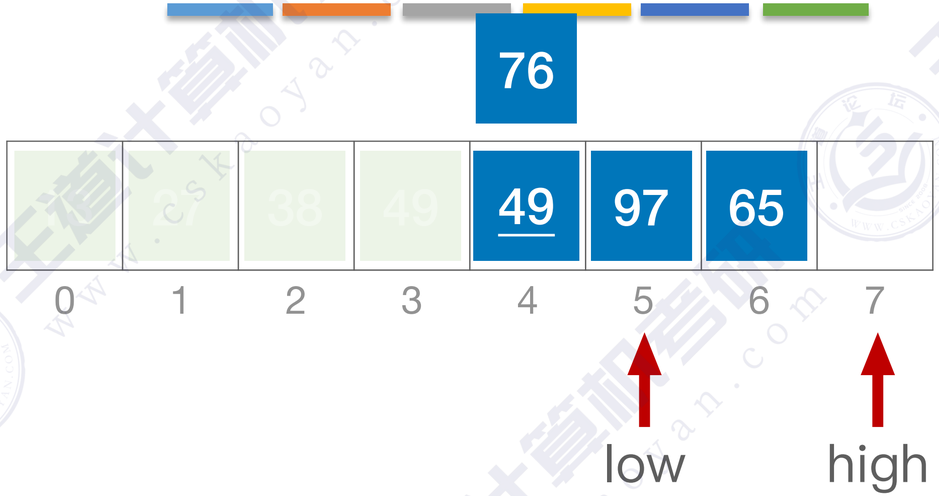
第二层 QuickSort

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 }
```

快速排序



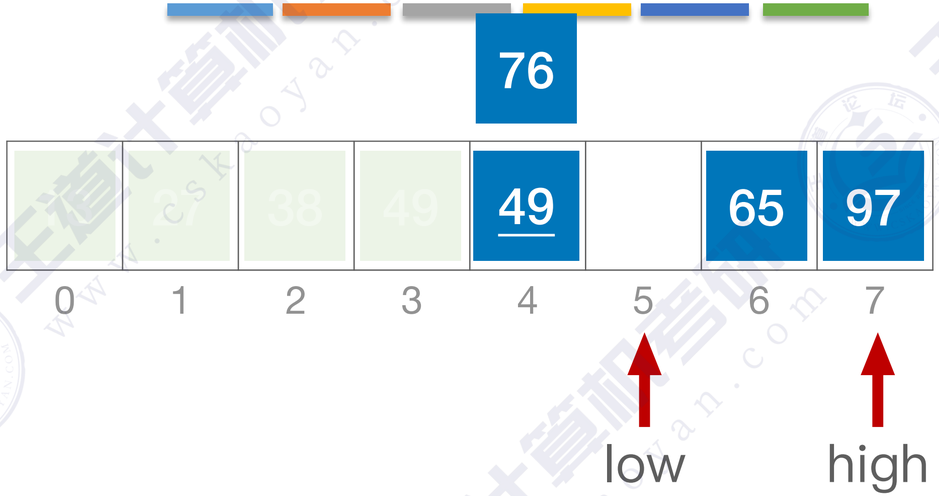
第二层 QuickSort

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 }
```

快速排序



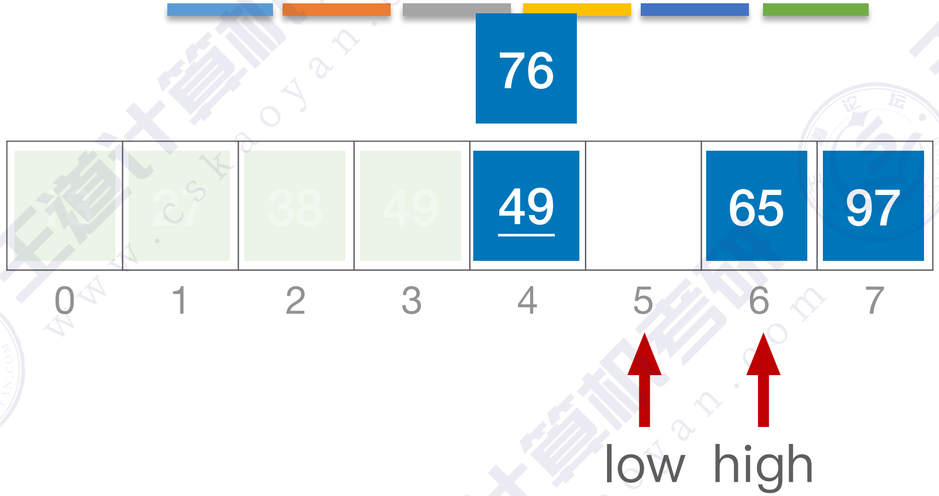
第二层 QuickSort

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 }
```

快速排序



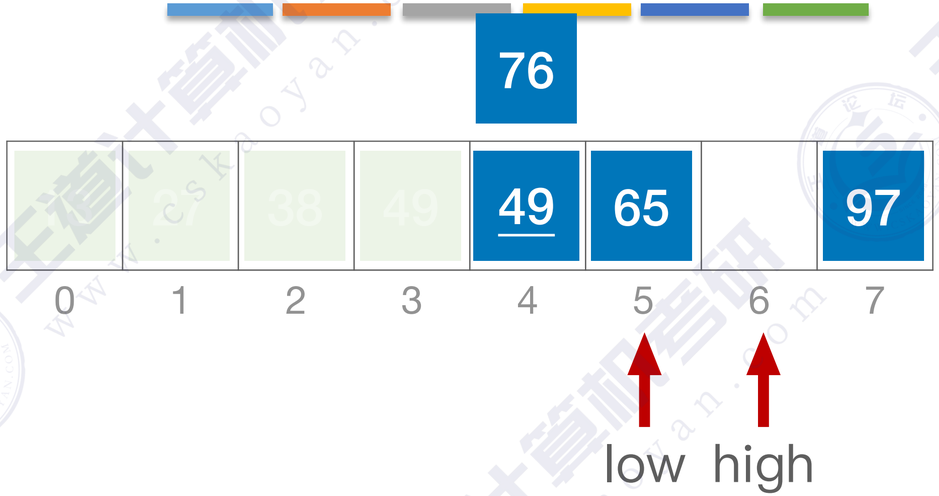
第二层 QuickSort

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 }
```

快速排序



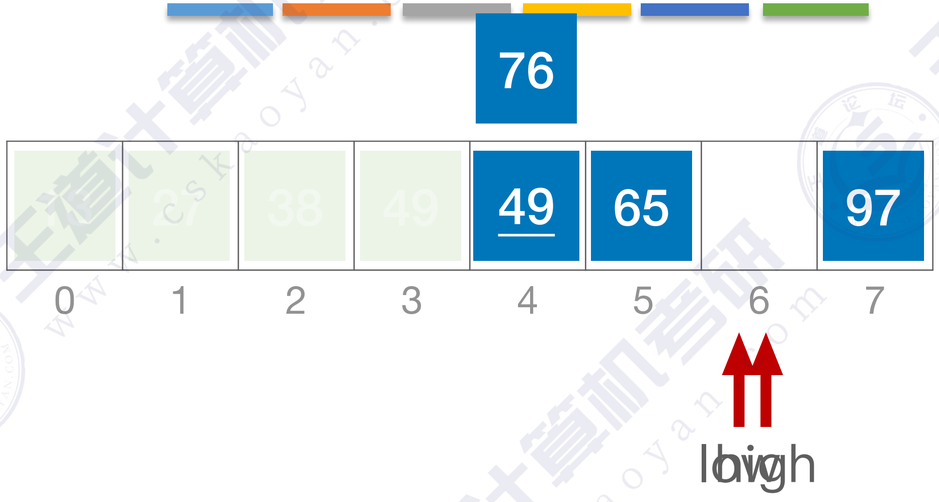
第二层 QuickSort

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 }
```

快速排序



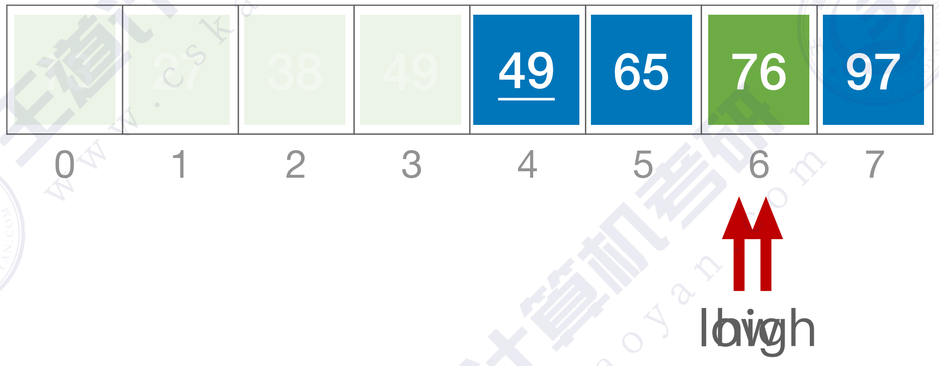
第二层 QuickSort

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 }
```

快速排序



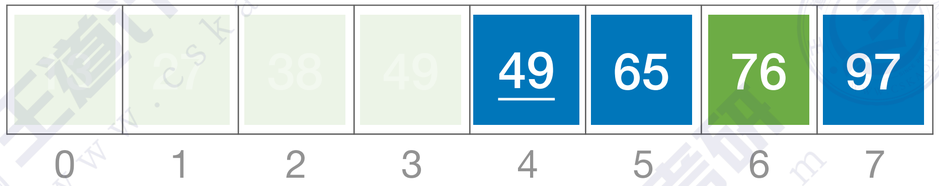
第二层 QuickSort

递归工作栈

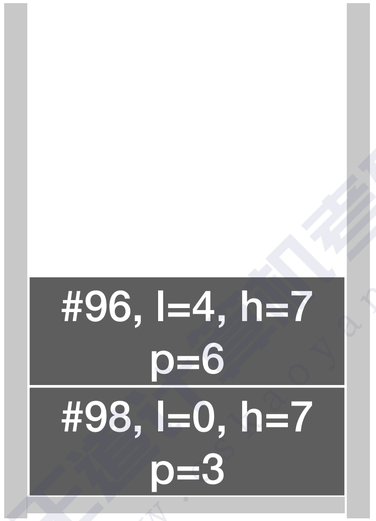
```
//用第一个元素将待排序序列划分成左右两个部分
int Partition(int A[],int low,int high){
    int pivot=A[low];          //第一个元素作为枢轴
    while(low<high){           //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];        //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        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 }
```

快速排序



第二层 QuickSort

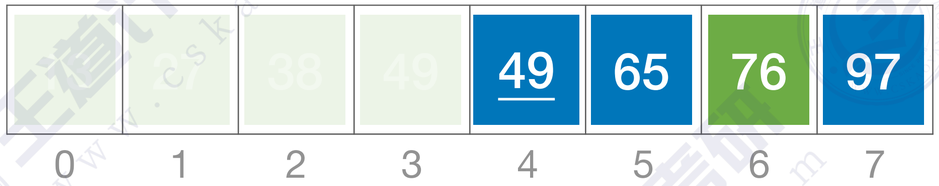


递归工作栈

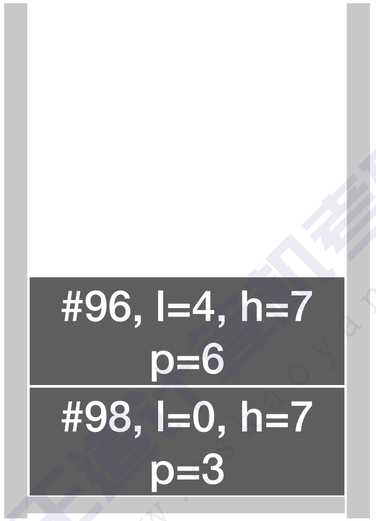
```
//用第一个元素将待排序序列划分成左右两个部分
int Partition(int A[],int low,int high){
    int pivot=A[low];          //第一个元素作为枢轴
    while(low<high){           //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];        //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        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 }
```

快速排序



第二层 QuickSort

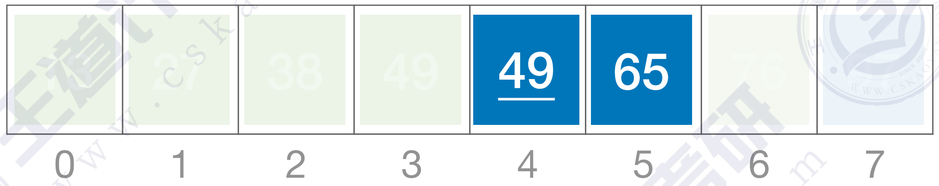


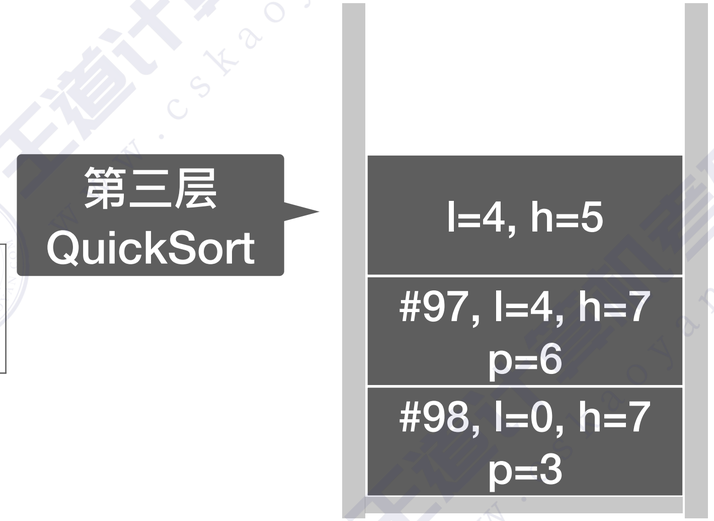
递归工作栈

```
//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 }
```

快速排序



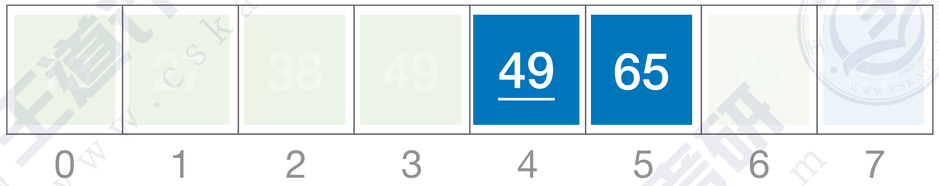


递归工作栈

```
//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 }
```

快速排序



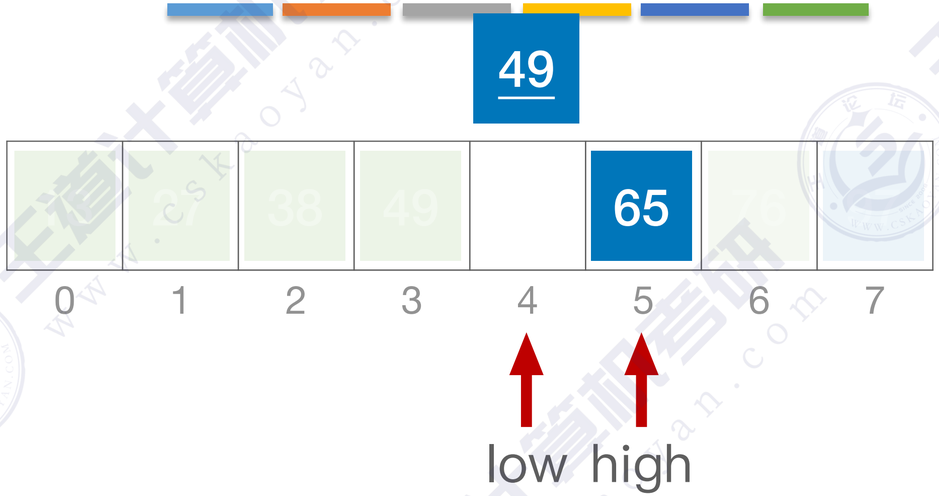
第三层 QuickSort

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 }
```

快速排序



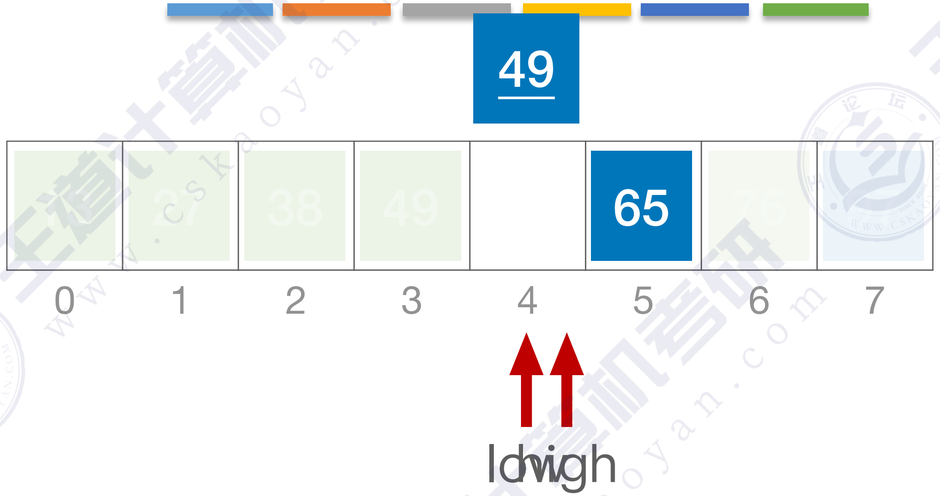
第三层 QuickSort

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 }
```

快速排序



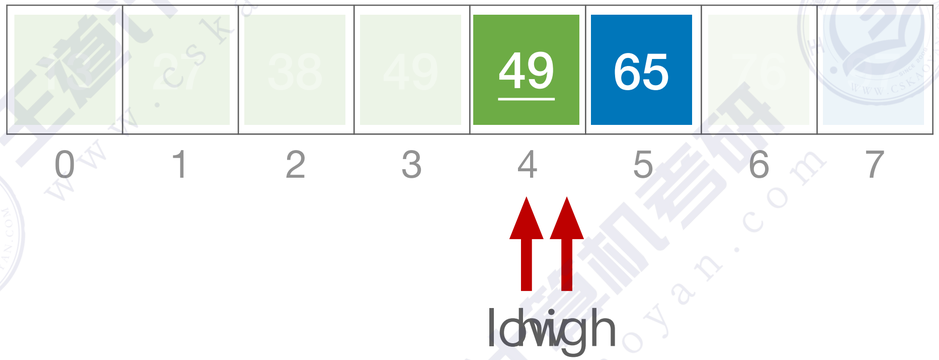
第三层 QuickSort

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 }
```

快速排序



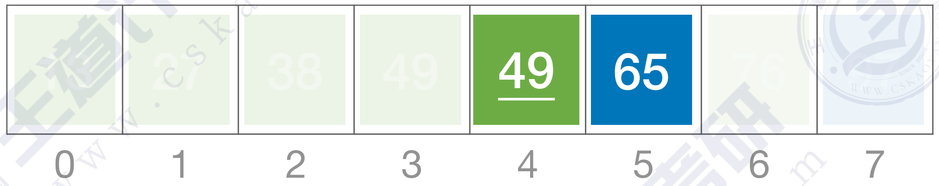
第三层 QuickSort

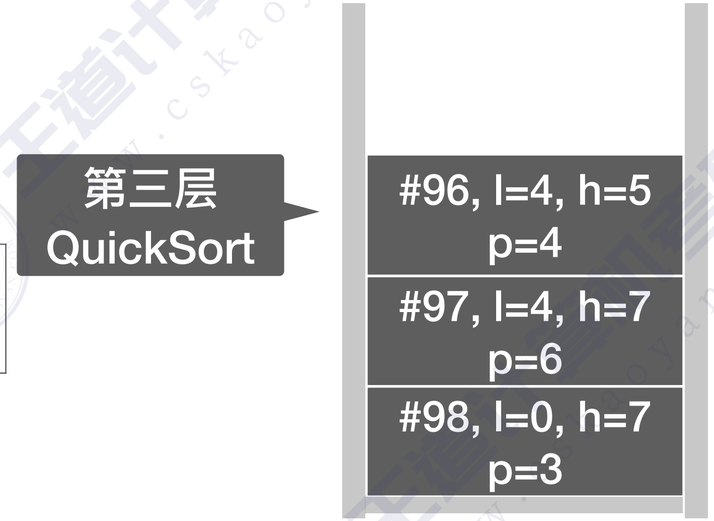
递归工作栈

```
//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 }
```

快速排序



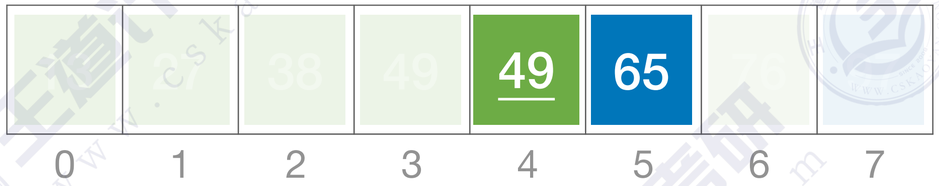


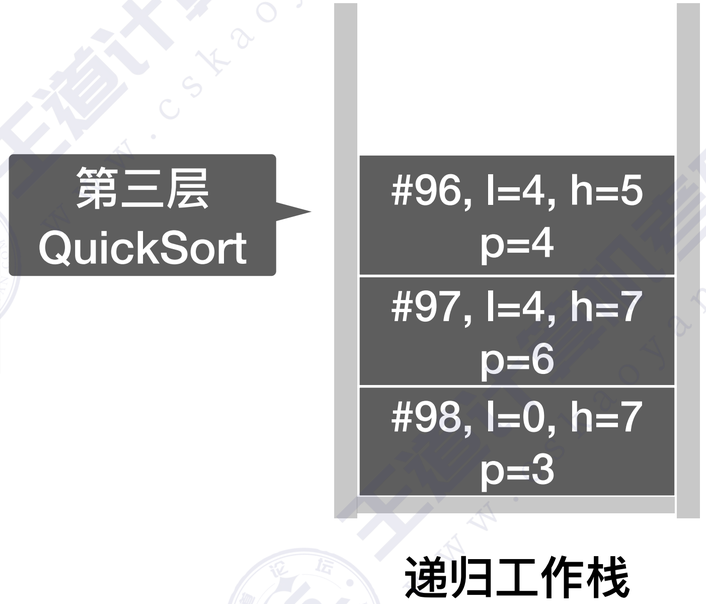
递归工作栈

```
//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 }
```

快速排序

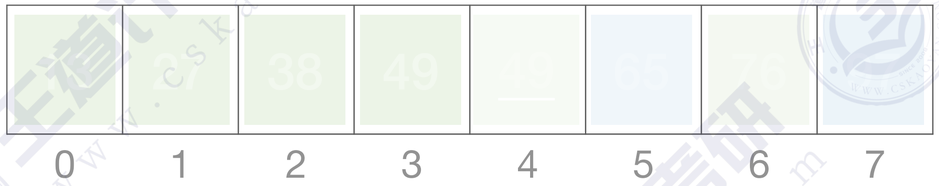


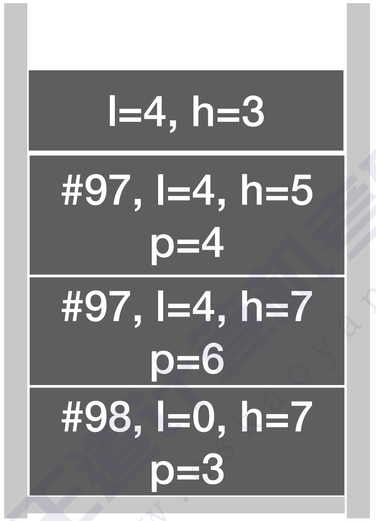



```
//用第一个元素将待排序序列划分成左右两个部分
int Partition(int A[],int low,int high){
    int pivot=A[low];          //第一个元素作为枢轴
    while(low<high){           //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];        //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        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 }
```

快速排序



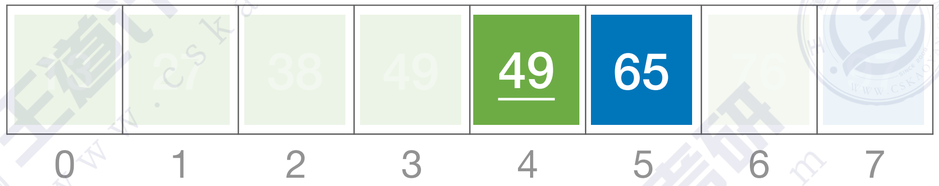


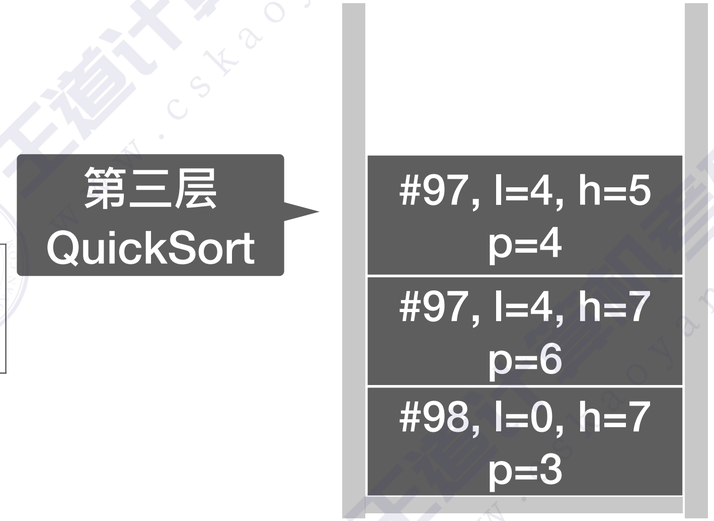
递归工作栈

```
//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 }
```

快速排序



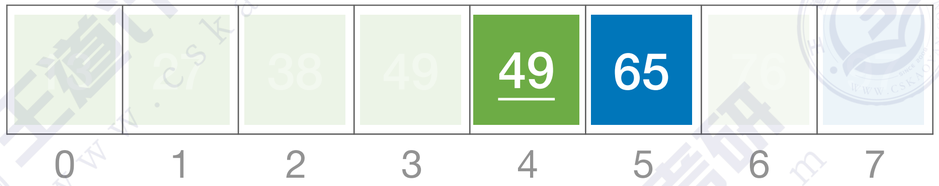


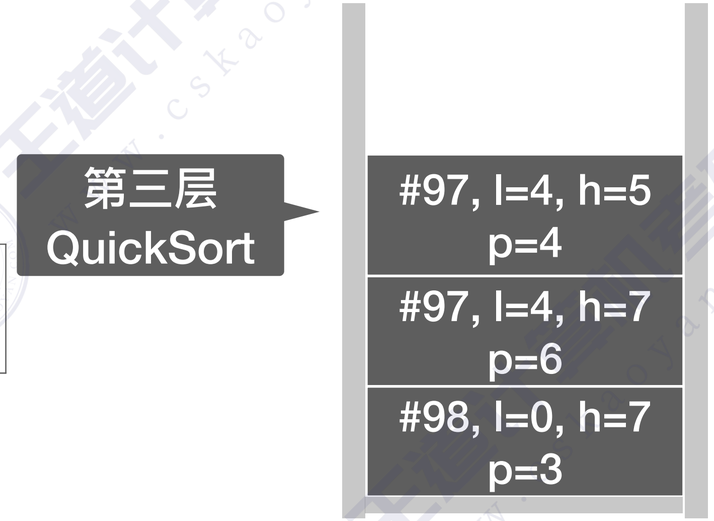
递归工作栈

```
//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 }
```

快速排序



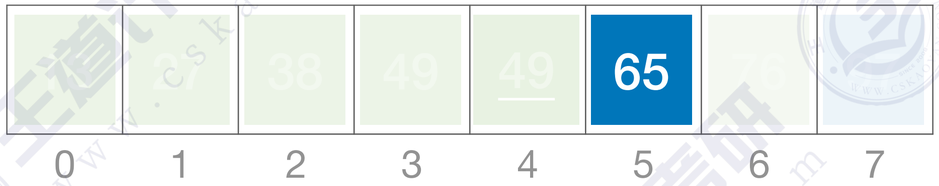


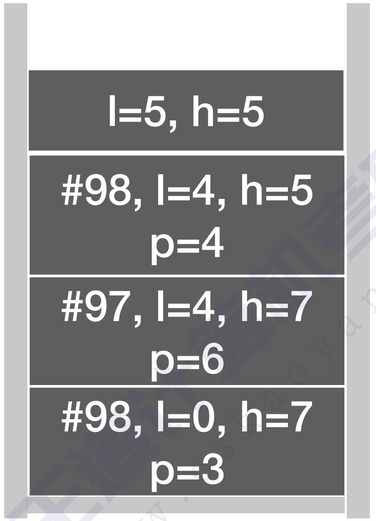
递归工作栈

```
//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 }
```

快速排序



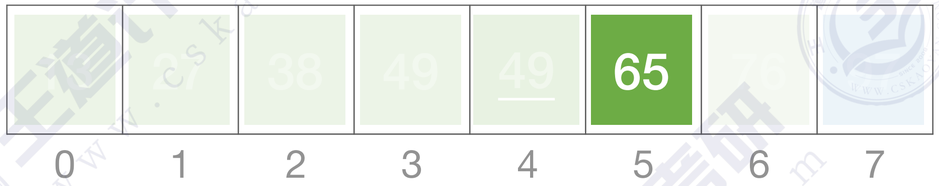


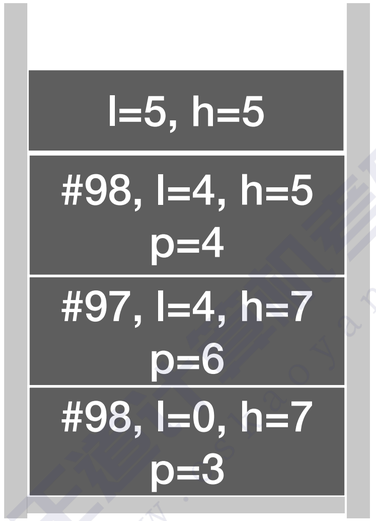
递归工作栈

```
//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 }
```

快速排序



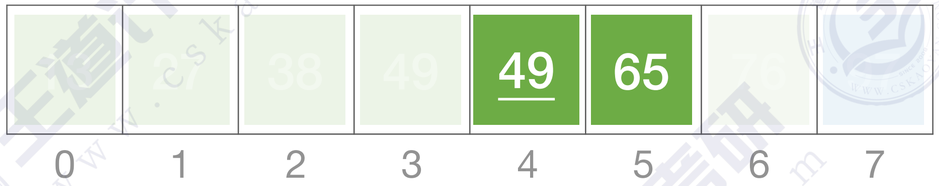


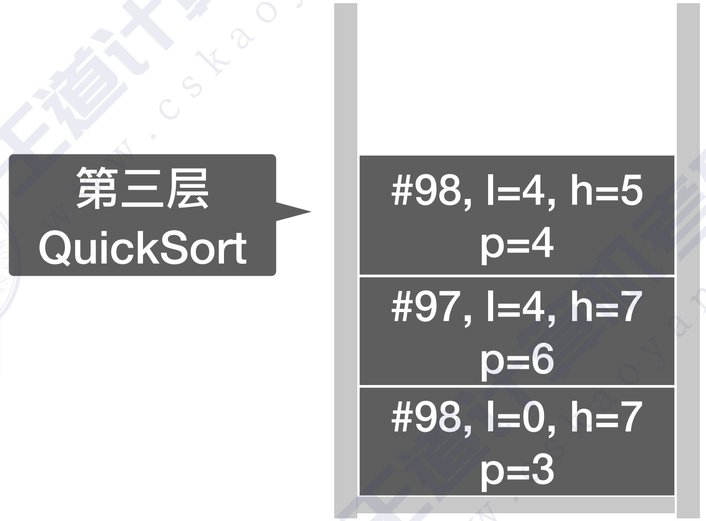
递归工作栈

```
//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 }
```

快速排序



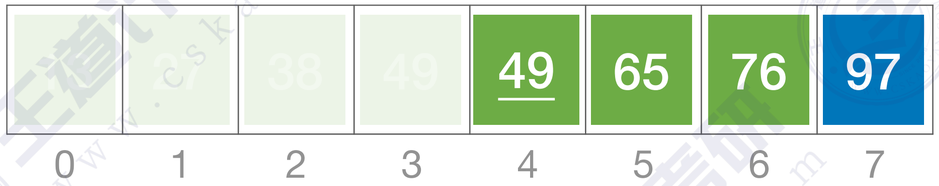


递归工作栈

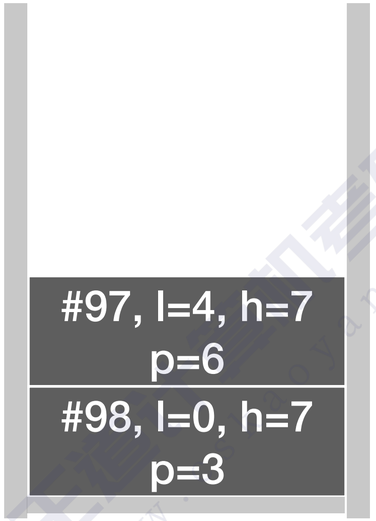
```
//用第一个元素将待排序序列划分成左右两个部分
int Partition(int A[],int low,int high){
    int pivot=A[low];          //第一个元素作为枢轴
    while(low<high){           //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];        //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        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 }
```

快速排序



第二层 QuickSort

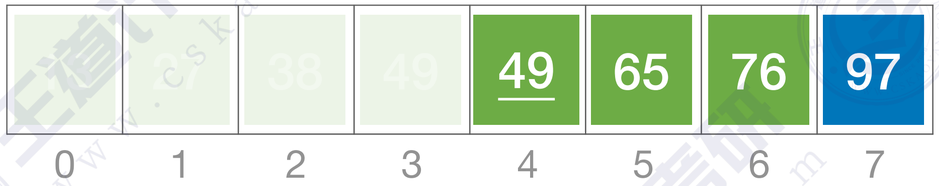


递归工作栈

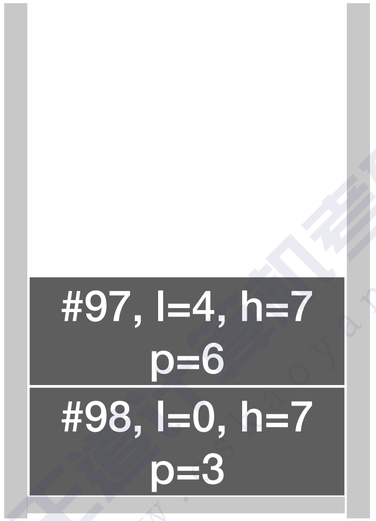
```
//用第一个元素将待排序序列划分成左右两个部分
int Partition(int A[],int low,int high){
    int pivot=A[low];          //第一个元素作为枢轴
    while(low<high){           //用low、high搜索枢轴的最终位置
        while(low<high&&A[high]>=pivot) --high;
        A[low]=A[high];        //比枢轴小的元素移动到左端
        while(low<high&&A[low]<=pivot) ++low;
        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 }
```

快速排序



第二层 QuickSort

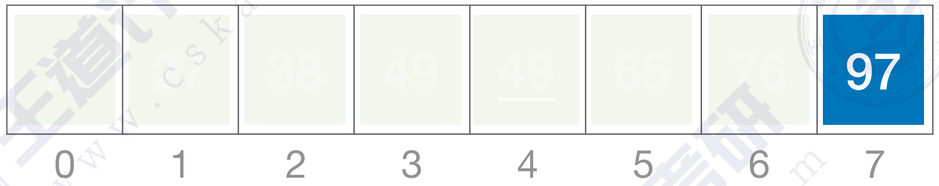


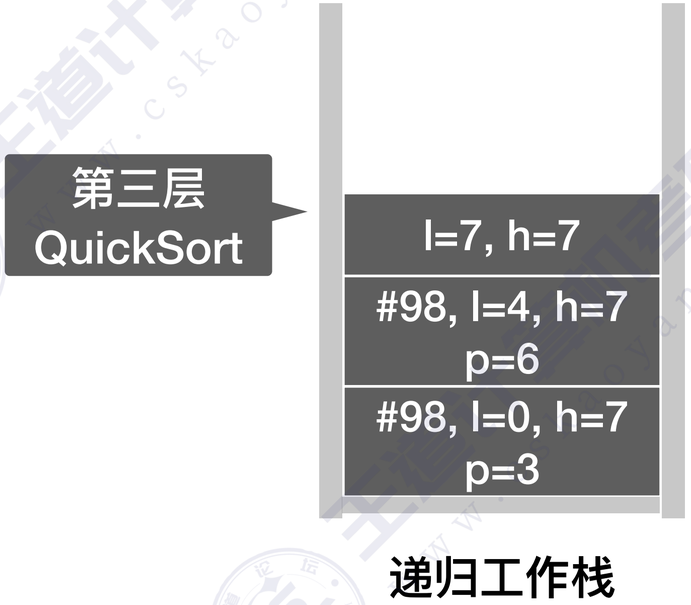
递归工作栈

```
//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 }
```

快速排序



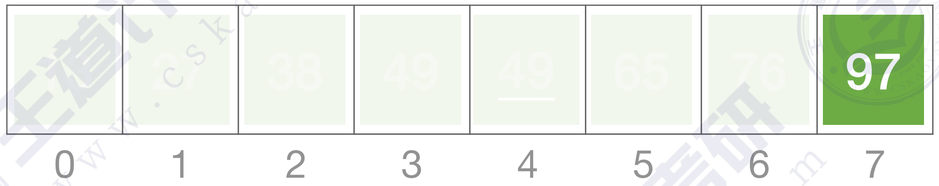


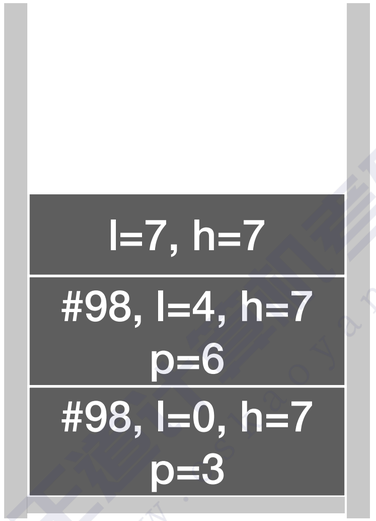
递归工作栈

```
//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 }
```

快速排序



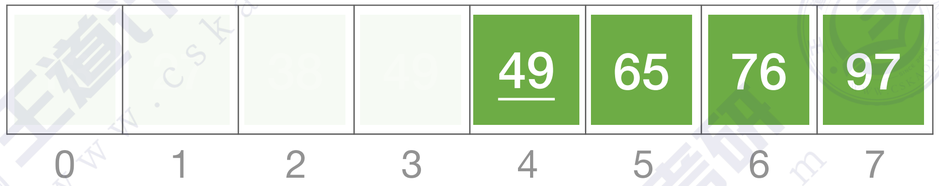


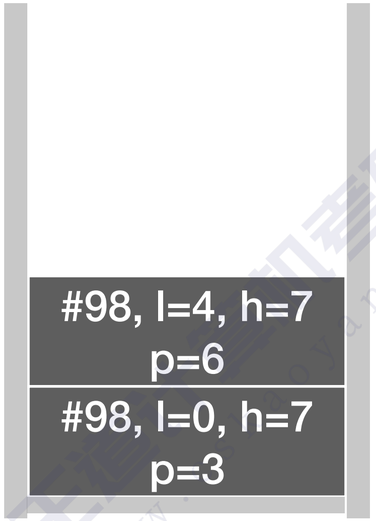
递归工作栈

```
//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 }
```

快速排序



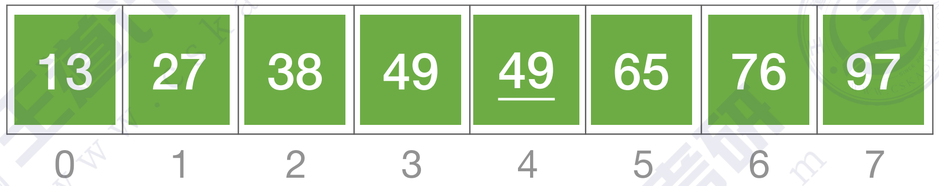


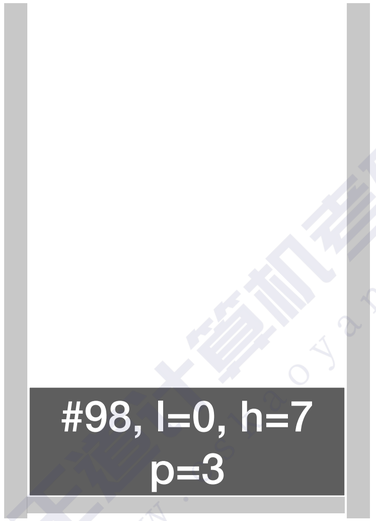
递归工作栈

```
//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 }
```

快速排序



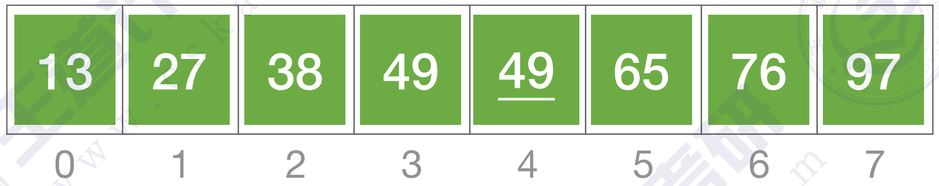


递归工作栈

```
//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 }
```

快速排序



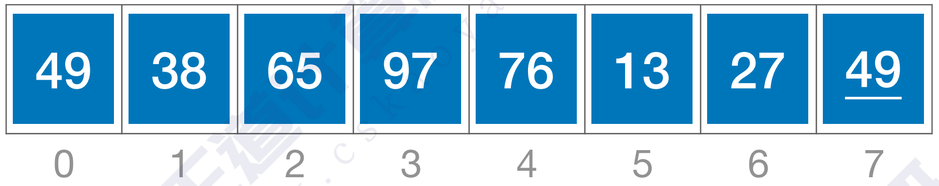
递归工作栈

```
//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 }
```

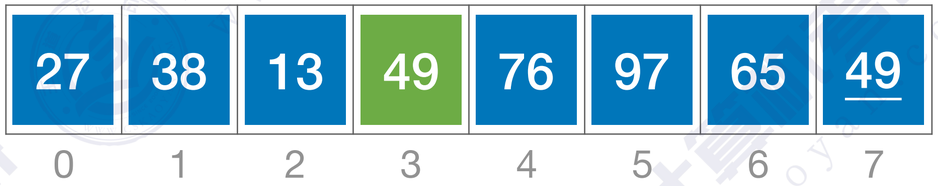
算法效率分析

初始序列



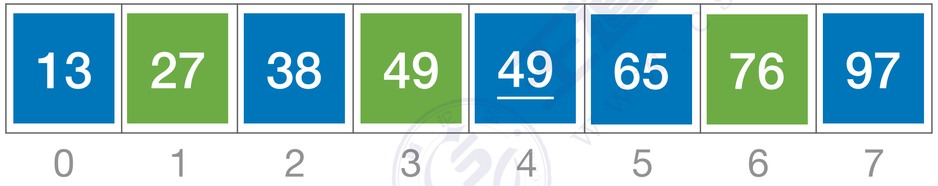
（待排序元素：0~7）

第一层QuickSort处理后：



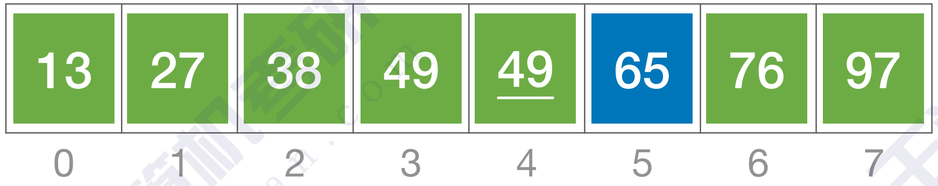
（待排序元素：0~2, 4~7）

第二层QuickSort处理后：



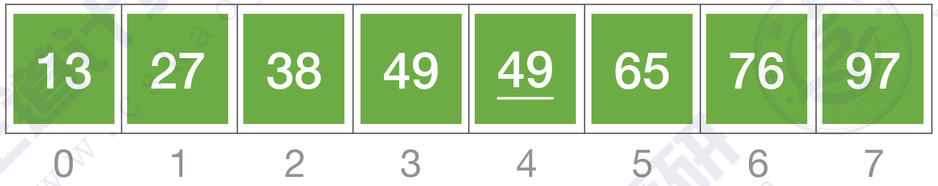
（待排序元素：0~0, 2~2, 4~5, 7~7）

第三层QuickSort处理后：



（待排序元素：5~5）

第四层QuickSort处理后：



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*递归层数)

算法效率分析

初始序列

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

第一层QuickSort处理后：

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

第二层QuickSort处理后：

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

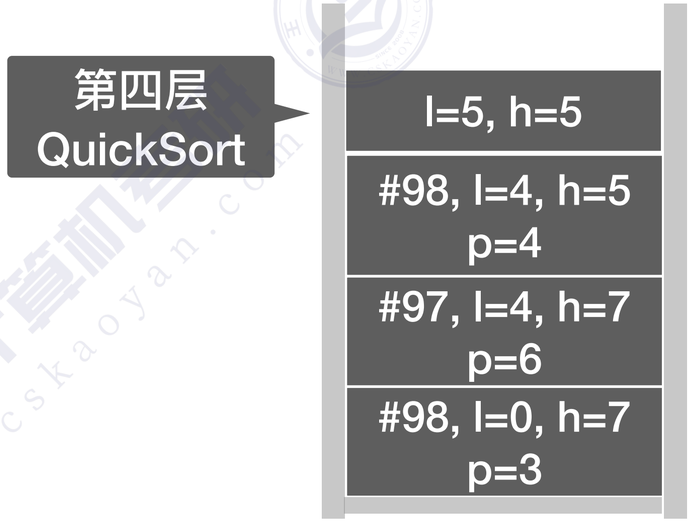
第三层QuickSort处理后：

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

第四层QuickSort处理后：

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

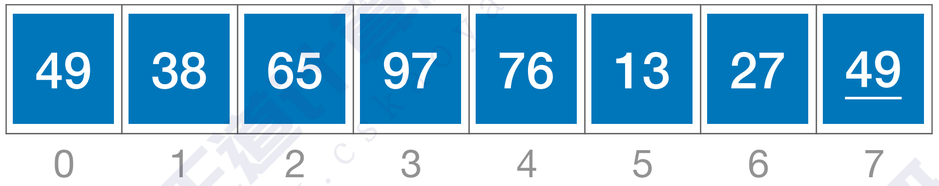
空间复杂度=O(递归层数)



递归工作栈

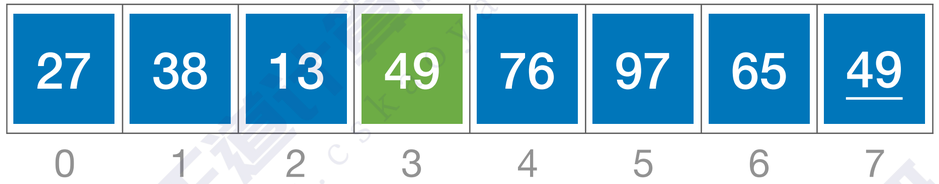
算法效率分析

初始序列



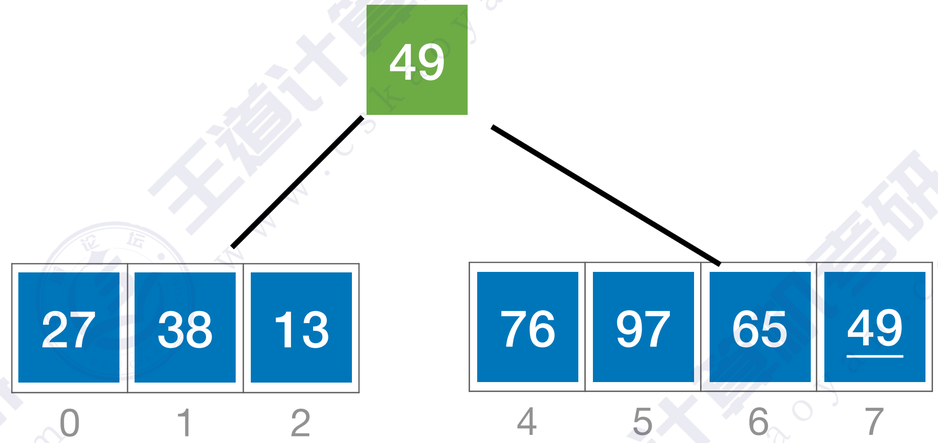
算法效率分析

第一层QuickSort处理后：



算法效率分析

第一层QuickSort处理后：



第二层QuickSort
要处理的部分：

算法效率分析

第一层QuickSort处理后：

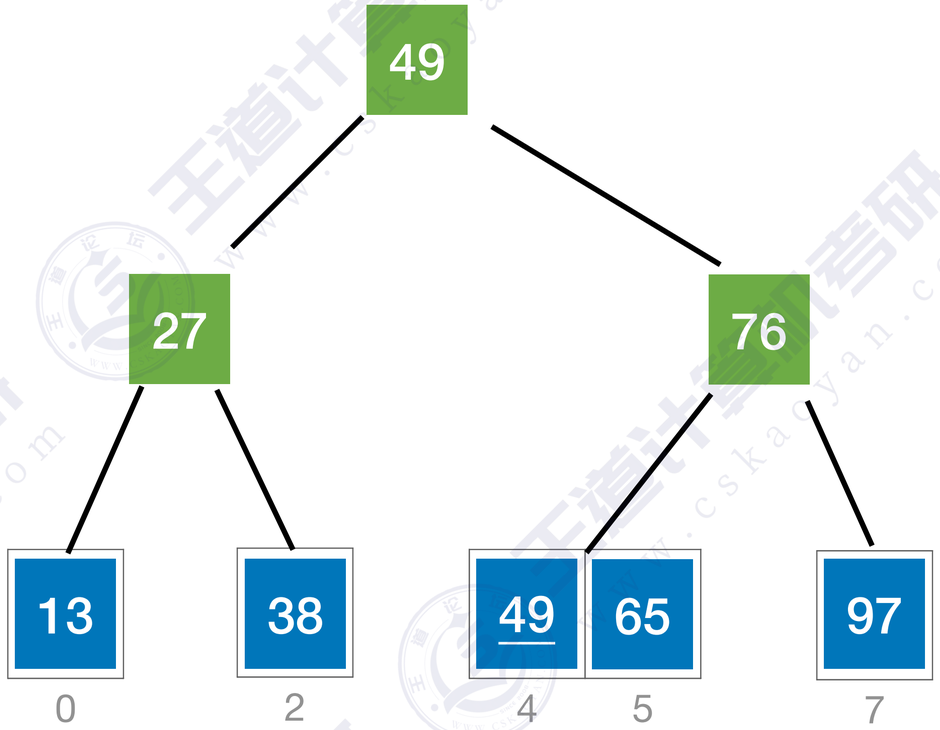
第二层QuickSort处理后：

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：

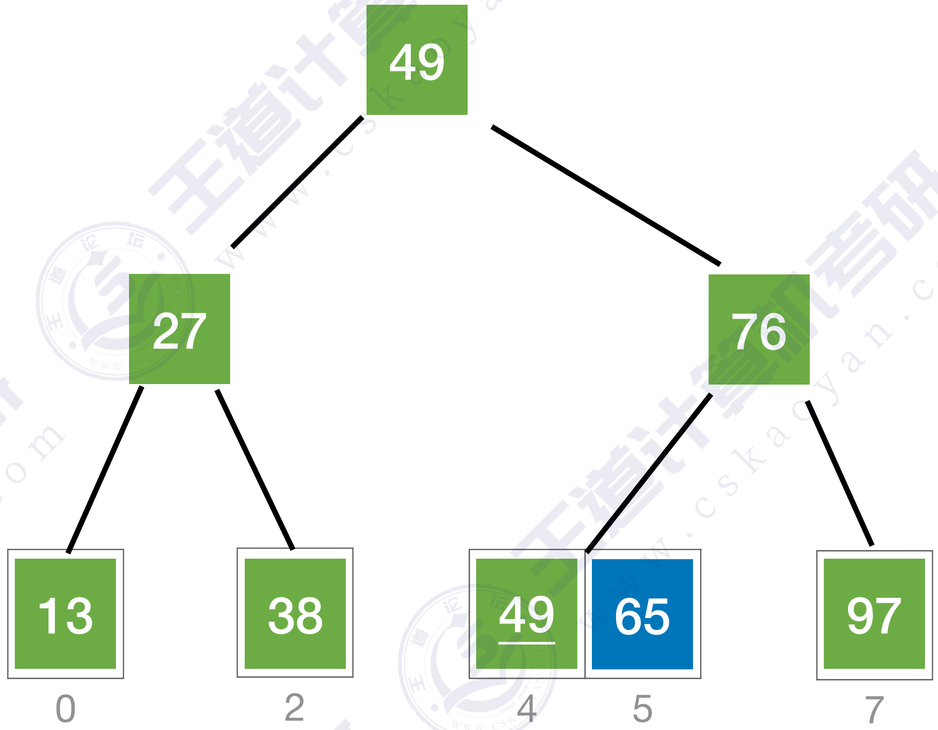


算法效率分析

第一层QuickSort处理后：

第二层QuickSort处理后：

第三层QuickSort处理后：



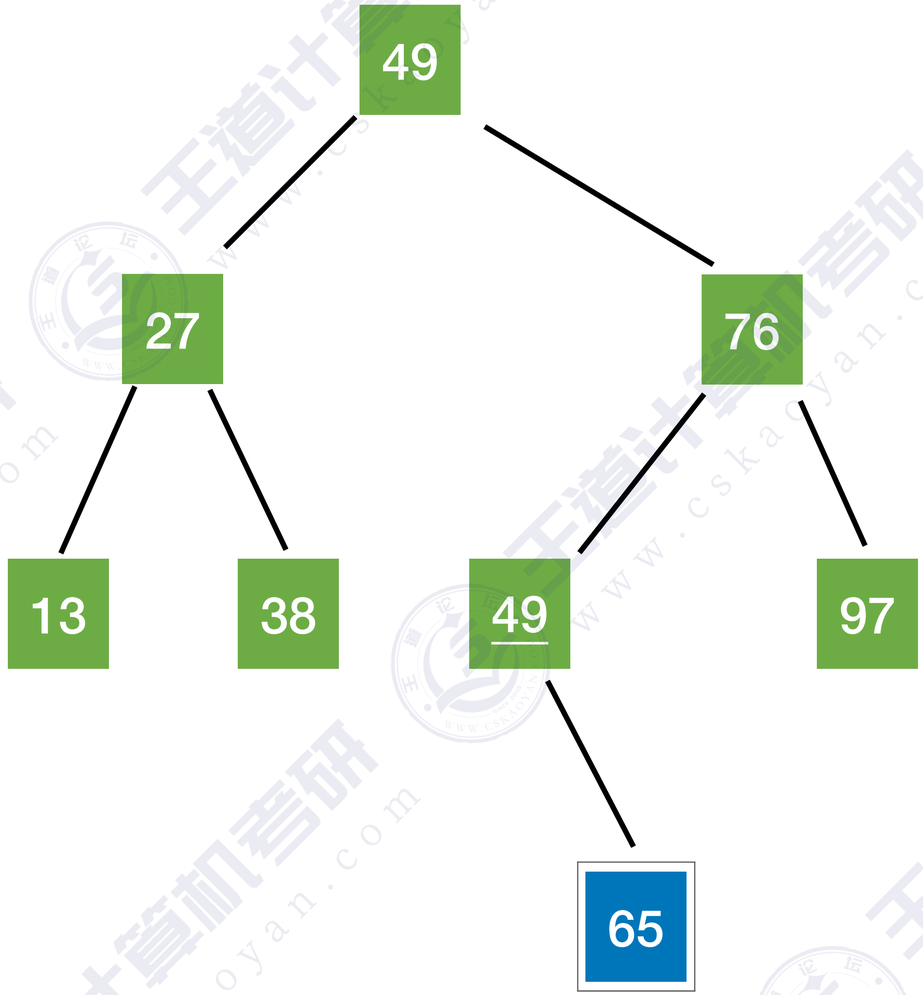
算法效率分析

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：



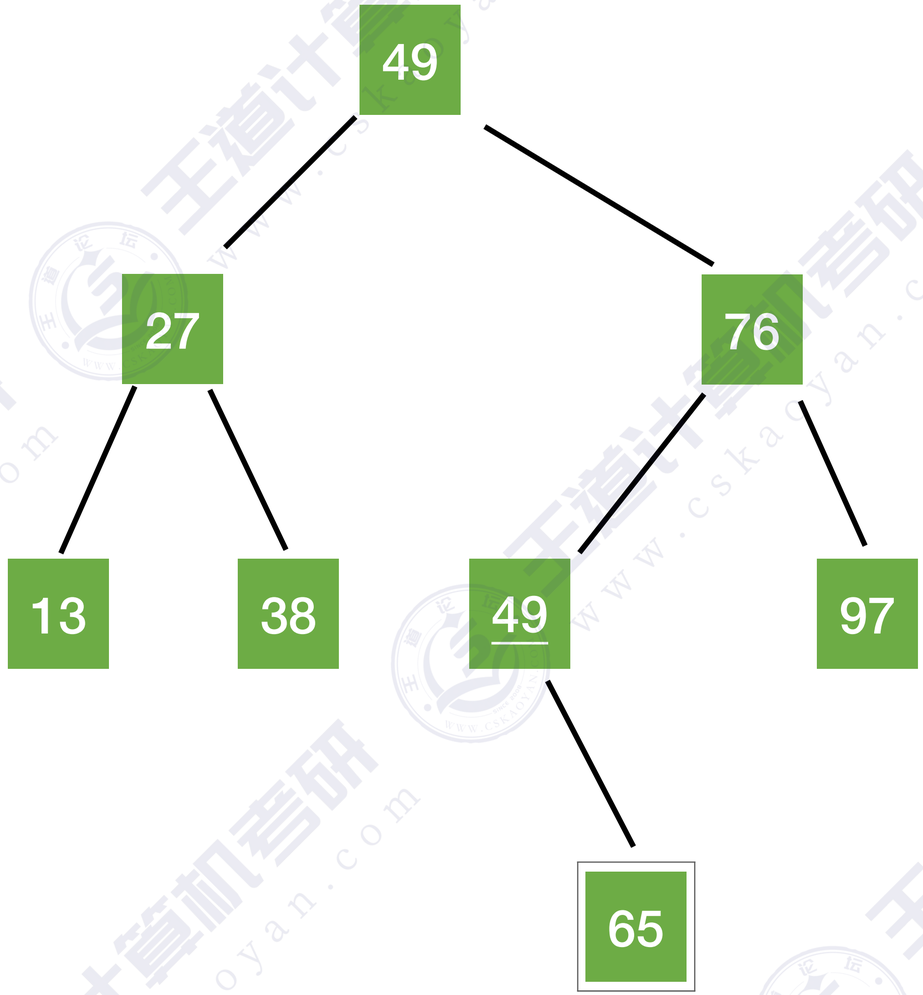
算法效率分析

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：



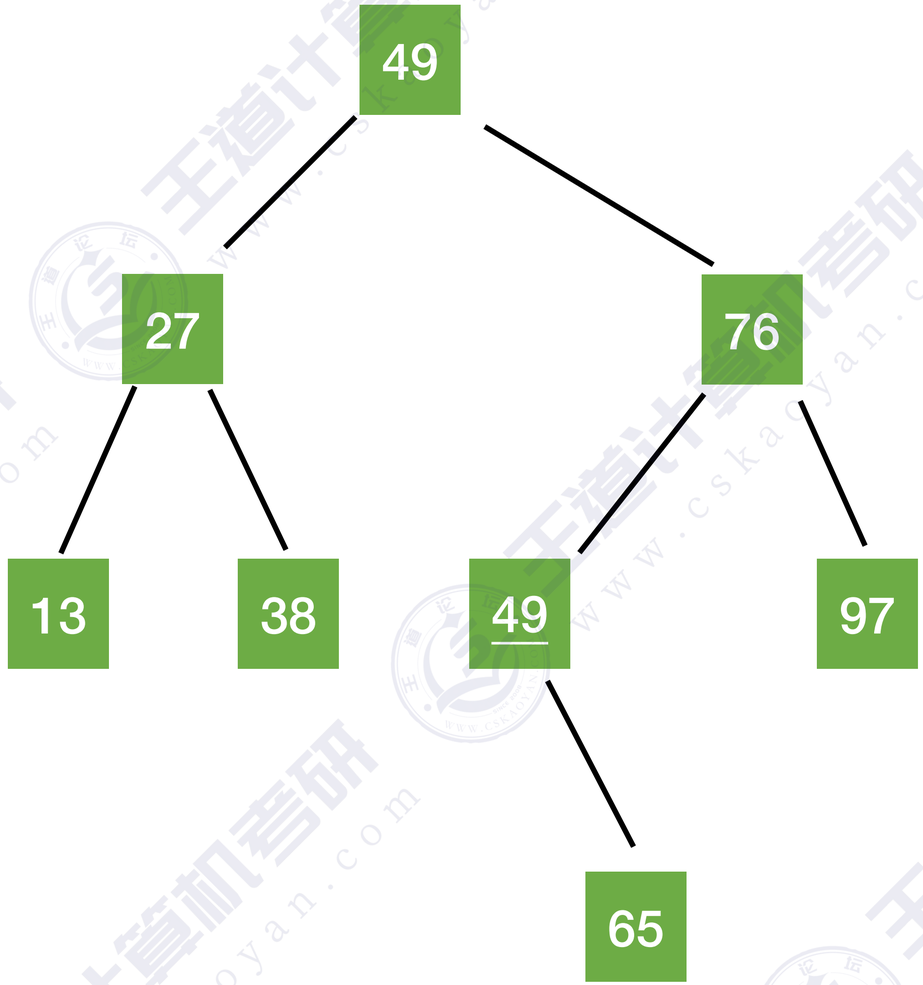
算法效率分析

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

算法效率分析

n个结点的二叉树

最小高度 = $\lfloor log_2 n \rfloor + 1$

最大高度 = n

时间复杂度=O(n*递归层数)

空间复杂度=O(递归层数)

最好时间复杂度=O($nlog_2n$)

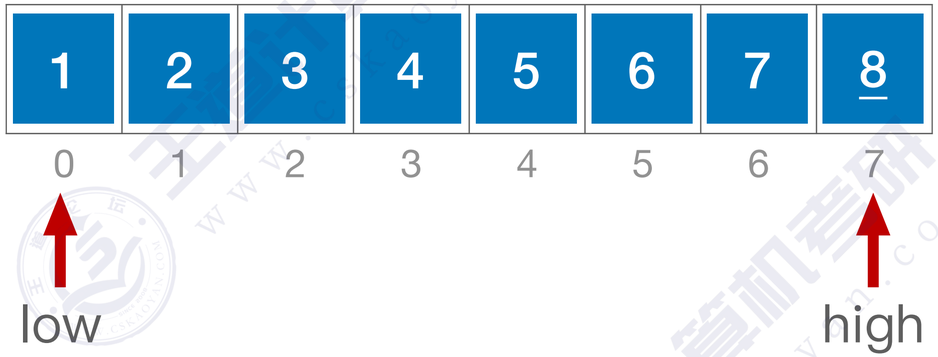
最坏时间复杂度=O(n^2)

最好空间复杂度=O(log_2n)

最坏空间复杂度=O(n)

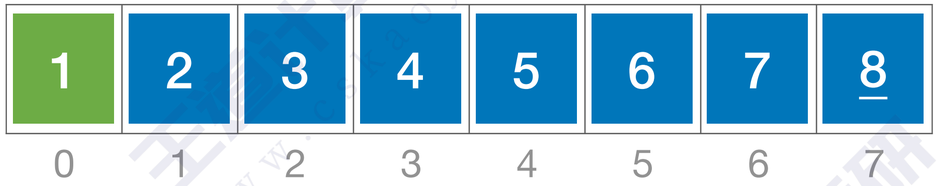
最坏的情况

初始序列



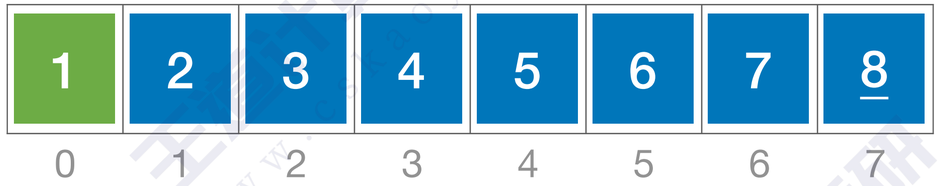
最坏的情况

第一层QuickSort处理后：

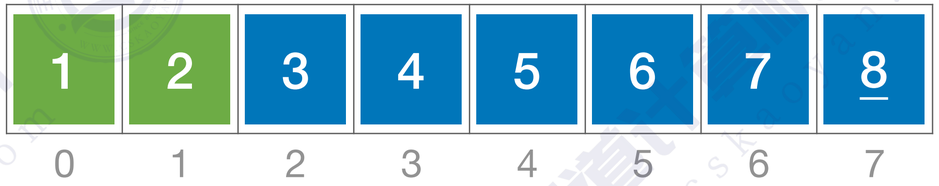


最坏的情况

第一层QuickSort处理后：

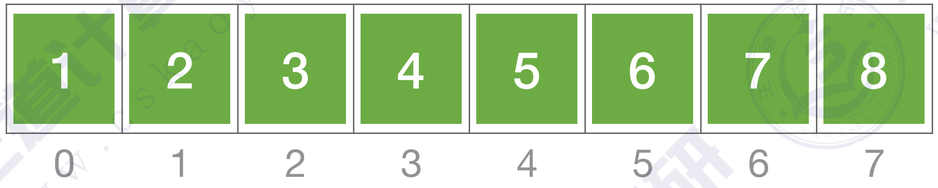


第二层QuickSort处理后：



第三层QuickSort处理后：

第8层QuickSort处理后：

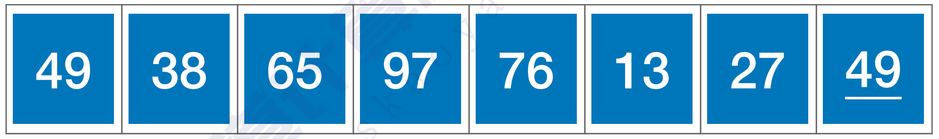


若每一次选中的“枢轴”将待排序序列划分为很不均匀的两个部分，则会导致递归深度增加，算法效率变低

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）

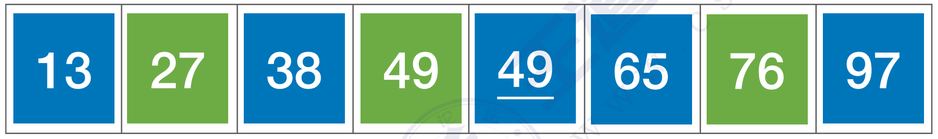
比较好的情况

初始序列



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：

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

算法效率分析

时间复杂度=O(n*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)

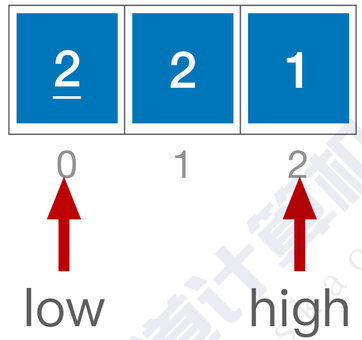
若序列原本就有序或逆序，则时、空复杂度最高（可优化，尽量选择可以把数据中分的枢轴元素。）

平均时间复杂度=O($n\log_2 n$)

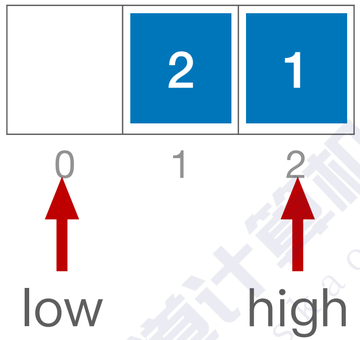
厉害厉害

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

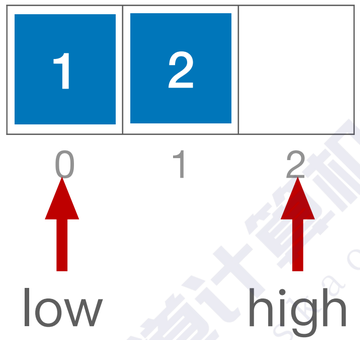
稳定性



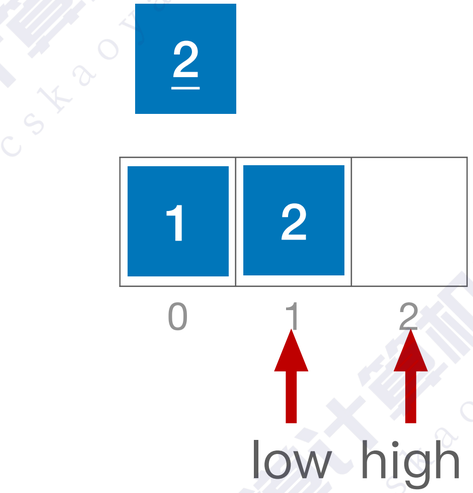
稳定性



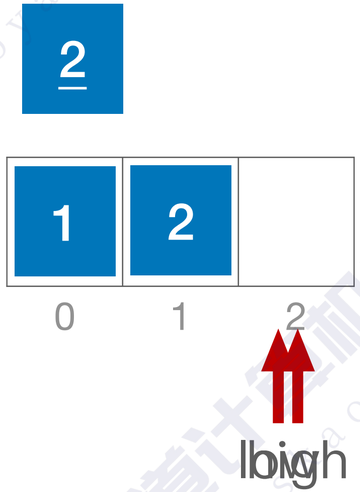
稳定性



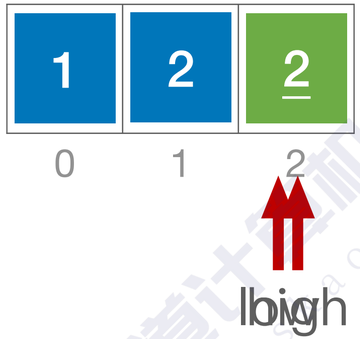
稳定性



稳定性



稳定性



稳定性

不稳定！

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。

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。