认识O(NlogN)的排序

剖析递归行为和递归行为时间复杂度的估算

用递归方法找一个数组中的最大值，系统上到底是怎么做的?

masetr公式的使用：T(N)=a*T(N/b)+O(N^d)

- log(b,a)>d ---> 复杂度为O(N^log(b,a))
- log(b,a)=d ---> 复杂度为O(N^d*logN)
- log(b,a)<d ---> 复杂度为O(N^d)

T(N)中的N指母问题的数据量

T(N/b)中的N/b指子问题的规模都是N/b

a指子问题被调用了多少次，调用次数

O(N^d)，除去调用的子问题之外，剩余过程的时间复杂度

此时不符合master公式，因为子问题规模不一样，N/3、2N/3

归并排序

1. 整体就是一个简单递归，左边排好序、右边排好序、让其整体有序
2. 让其整体有序的过程里用了排外序方法
3. 利用master公式来求解时间复杂度
4. 归并排序的实质

时间复杂度O(N*logN)，额外空间复杂度O(N)

困扰了很长时间的bug：int mid=left+(right-left)>>1，错误，因为+比>>的优先级高，所以应该写成：

int mid=left+((right-left)>>1)

归并排序时间复杂度分许

为什么merge sort可以做到O(NlogN)这个时间复杂度?

选择、冒泡、插入排序效果差O(N^2)的原因是：浪费了大量的比较行为。选择排序比较了N次，才搞定了一个数，才确定一个数的位置，

选择排序的每轮比较都是独立的，一轮排序只确定了一个数的位置，丢弃了大量相关信息；冒泡、插入排序也是同样道理，浪费了大量比较行为。

归并排序没有浪费比较行为，

左侧有序，右侧有序，两边比较，比较没有浪费，变成了一个更大的有序部分，跟下一个更大的有序部分，进行比较，变成了一个更大的有序部分。

归并排序的扩展

1. 小和问题：在一个数组中，每一个数左边比当前数小的数累加起来，叫做这个数组的小和。求一个数组的小和。eg：[1，3，4，2，5]，1左边比1小的数，没有；3左边比3小的数，1；4左边比4小的数，1、3；2左边比2小的数，1；5左边比5小的数，1、3、4、2；所以小和为1+1+3+1+1+3+4+2=16

相等的情况，必须先拷贝左侧，否则会出现遗漏的情况，这一点与单纯的MergeSort不同。

按照上述情况，如果相等的时候(1==1)，先拷贝左侧，则当右边指向2时，左边也指向2。右边的2会遗漏左边的4个1

排序不能省：因为我们通过下表计算的方式：(right-j+1)知道比左边的数大的个数，用了O(1)，所以不能省

左程云版本：

左侧小和、右侧小和、merge产生的小和，相加

1. 逆序对问题：在一个数组中，左边的数如果比右边的数大，则这两个数构成一个逆序对，请打印所有逆序对。

与上一题等效

总结：一般题目中出现左边、右边，要考虑归并排序、二分法

快速排序

荷兰国旗问题

问题1

给定一个数组arr，和一个数num，请把小于等于num的数放在数组的左边，大于num的数放在数组的右边。要求额外空间复杂度为O(1)，时间复杂度O(N)。这里使用双指针

```java
//给定一个数组arr，和一个数num，请把小于等于num的数放在数组的左边，大于num的数放在数组的右边。要求额外空间复杂度O(1)。时间复杂度O(N)
public static void netherlandFlag1(int[] arr,int num){
    int len=arr.length;
    if(len<2)
        return;
    int index=0;
    for(int i=0;i<len;i++){
        if(arr[i]<=num){
            /* index和i可能时同一个位置，所以不保险
            arr[index]=arr[index]^arr[i];
            arr[i]=arr[index]^arr[i];
            arr[index]=arr[index]^arr[i];
            */
            int tmp=arr[i];
            arr[i]=arr[index];
            arr[index]=tmp;
            index++;
        }
    }
}
```

双指针、快慢指针

i在遍历过程中，小于等于区域推着大于区域往前走，大于区域推到整个数组时就做到了这一点：小于等于都在左边，大于都在右边

问题2（荷兰国旗问题）

给定一个数组arr，和一个数num，请把小于num的数放在数组的左边，等于num的数放在数组的中间，大于num的数放在数组的右边。要求额外空间复杂度O(1)，时间复杂度O(N)。这里使用三指针

```java
//荷兰国旗问题
//给定一个数组arr，和一个数组num，请把小于num的数放在数组的左边，等于num的数放在数组的中间，大于num的数放在数组的右边。要求额外空间复杂度
//O(1)，时间复杂度O(N)
public static void netherlandFlag2(int[] arr,int num){
    int len=arr.length;
    if(len<2)
        return;
    int index=0;
    for(int i=0;i<len;i++){
        if(arr[i]<=num){
            int tmp=arr[i];
            arr[i]=arr[index];
            arr[index]=tmp;
            index++;
        }
    }

    len=index;
    index=0;
    for(int i=0;i<len;i++){
        if(arr[i]<num){
            int tmp=arr[i];
            arr[i]=arr[index];
            arr[index]=tmp;
            index++;
        }
    }
}
```

分三种情况：

1. [i]<num，[i]和小于区域下一个交换，小于区间右扩，i++
2. [i]==num，i++
3. [i]>num，[i]和大于区域前一个交换，大于区域左扩，i不变，右边的数是新过来的，还不知道是否<num，所以i原地不动

自己理解：把大于num的数放在右边，小于num的数放在左边，等于num的数被交换来、交换去，等于num的数自然就到了中间区域。把大于num的数放在右边，小于num的数放在左边，等于num的数就只能放在中间了

```java
//荷兰国旗问题，版本2
public static void netherland(int[] arr,int num){
    int len=arr.length;
    if(len<2)
        return;

    int left=0;
    int right=len-1;
    int index=0;
    for(int i=0;i<=right;){
        if(arr[i]<num){//  <num
            int tmp=arr[i];
            arr[i]=arr[index];
            arr[index]=tmp;
            index++;
            i++;
        } else if(arr[i]>num){//  >num
            int tmp=arr[i];
            arr[i]=arr[right];
            arr[right]=tmp;
            right--;
        } else {//  ==num
            i++;
        }
    }
}
```

左程云解释：i往右走，压缩待定区域，让小于区域推着等于区域往右走，奔向大于区域；要么[i]与大于区域交换，让大于区域往左扩，压缩待定区域；当小于区域推着等于区域与大于区域撞上的时候，搞定了。

快速排序

快速排序1.0

```
/*
public static void quickSort1(int[] arr,int left,int right){
    if(left>=right)
        return;

    //从小到大
    int l=left;
    int r=right-1;
    while(l<=r){
        while((l<=r)&&(arr[l]<=arr[right])){
            l++;
        }
        while((l<=r)&&(arr[r]>arr[right])){
            r--;
        }
        if(l<r){
            swap(arr,l,r);
        }
    }//肯定不会出现l==r，因为数组中的数字要么<=arr[right]、要么>arr[right]
    swap(arr,l,right);
    quickSort1(arr,left,l-1);
    quickSort1(arr,l+1,right);
}
*/
public static void quickSort1(int[] arr,int left,int right){
    if(left>=right)
        return;
    int l=left;
    int r=right-1;
    int rand=left+(int)((right-left+1)*Math.random());
    swap(arr,rand,right);
    while(l<=r){
        while((l<=r)&&(arr[l]<=arr[right]))
            l++;
        while((l<=r)&&(arr[r]>arr[right]))
            r--;
        if(l<r)
            swap(arr,l,r);
    }
    swap(arr,l,right);
    quickSort1(arr,left,l-1);
    quickSort1(arr,l+1,right);
}

public static void quickSort1_1(int[] arr,int left,int right){
    if(left>=right)
        return;
    int index=left;
    for(int i=left;i<right;i++){
        if(arr[i]<=arr[right]){
            swap(arr,index++,i);
        }
    }
    swap(arr,index,right);
    quickSort1_1(arr,left,index-1);
    quickSort1_1(arr,index+1,right);
}
```

1. 利用最后一个数字，作为基准
2. 将数组分为<=num的区域，和>num的区域；
3. 然后将>num区域的第一个数字与num（数组最后一个数字）互换位置
4. 中间就变成=num。将数组在这个位置，分为左边<=num区域、右边>num区域

然后对<=num的区域和>num的区域分别进行重述上述步骤，之后区间只剩1或0个数字

快速排序2.0

利用荷兰国旗问题，中间全部变成=num，左边<num，右边>num。相比快速排序1.0，中间成了一批=num的数字。然后对左边<num区域、右边>num区域分别重做荷兰国旗问题，直到<num区间、>num区间只剩1或0个数字。

```
//利用荷兰国旗问题
public static void quickSort2(int[] arr,int left,int right){
    if(left>=right)
        return;

    int l=left;
    int r=right-1;
    for(int i=left;i<=r;){
        if(arr[i]<arr[right]){
            swap(arr,i,l);
            l++;
            i++;
        } else if(arr[i]>arr[right]){
            swap(arr,i,r);
            r--;
        } else {//arr[i]==arr[right]的情况
            i++;
        }
    }
    swap(arr,r+1,right);
    quickSort2(arr,left,l-1);
    quickSort2(arr,r+2,right);
}
```

等于5的区域不再划分，搞定了一批等于5的数字。

6和基准5做交换：

最后<5区域在左边、=5区域在中间、>5区域在右边。

快速排序最差情况都是O(N^2)，每次划分只搞定一个数：

快速排序3.0

最差情况是由于划分值选取不合理，导致出现斜树。

划分值的好情况：两边数字个数几乎都是相等，都一半

最好的情况是左右两边递归的规模一样，应用master公式，此种情况时间复杂度为O(NlogN)

最差情况变成时间复杂度为O(N^2)的算法：

快速排序3.0：在数组中随机选取一个数字与数组最后一个数字交换，然后以最后一个数字为基准进行快速排序。此时，最差情况和最好情况出现概率是相等的。

打到N/5：T(N)=T(N/5)+T(4N/5)+O(N)

打到N/3：T(N)=T(N/3)+T(2N/3)+O(N)

打到N/2：T(N)=2*T(N/2)+O(N)

打到2N/3：T(N)=T(2N/3)+T(N/3)+O(N)

打到4N/5：T(N)=T(4N/5)+T(N/5)+O(N)

.....

每一种情况出现的概率都是相等的。每一种位置都是等概率事件，概率都是1/N

所有的情况都是1/N概率，然后求数学期望，得到时间复杂度（数学长期期望）为O(NlogN)

左程云版本快速排序：

```java
public static void quickSort3(int[] arr,int left,int right){
    if(left>=right)
        return;
    int r=left+(int)((right-left+1)*Math.random());
    swap(arr,r,right);
    int less=left-1;
    int more=right;
    int i=left;
    while(i<more){
        if(arr[i]<arr[right])
            swap(arr,++less,i++);
        else if(arr[i]>arr[right])
            swap(arr,--more,i);
        else
            ++i;
    }
    swap(arr,more,right);
    quickSort3(arr,left,less);
    quickSort3(arr,more+1,right);
}
```

//数组p的长度一定为2，表示<num的右边界、>num的左边界，开区间[...)、(...]

返回12、13两个数，返回划分值等于5的左边界和右边界，[....]

不改进的快速排序

1. 把数组范围中的最后一个数作为划分值，然后把数组通过荷兰国旗问题分成三个部分：

   左侧<划分值、中间==划分值、右侧>划分值

2. 对左侧范围和右侧范围，递归执行

分析：

- 划分值越靠近两侧，复杂度越高；划分值越靠近中间，复杂度越低
- 可以轻而易举的举出最差的例子，所以不改进的快速排序时间复杂度为O(N^2)

随机快速排序（改进的快速排序）

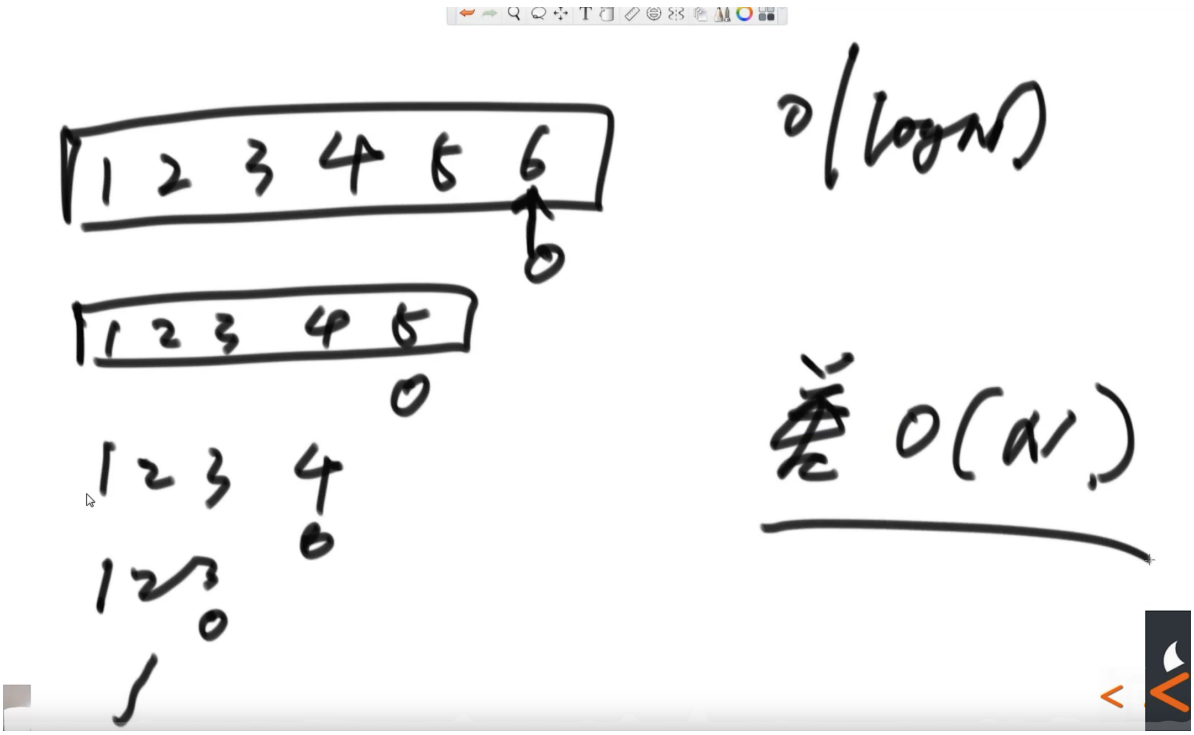
1. 在数组范围中，等概率随机选一个数作为划分值，然后把数组通过荷兰国旗问题分成三个部分：

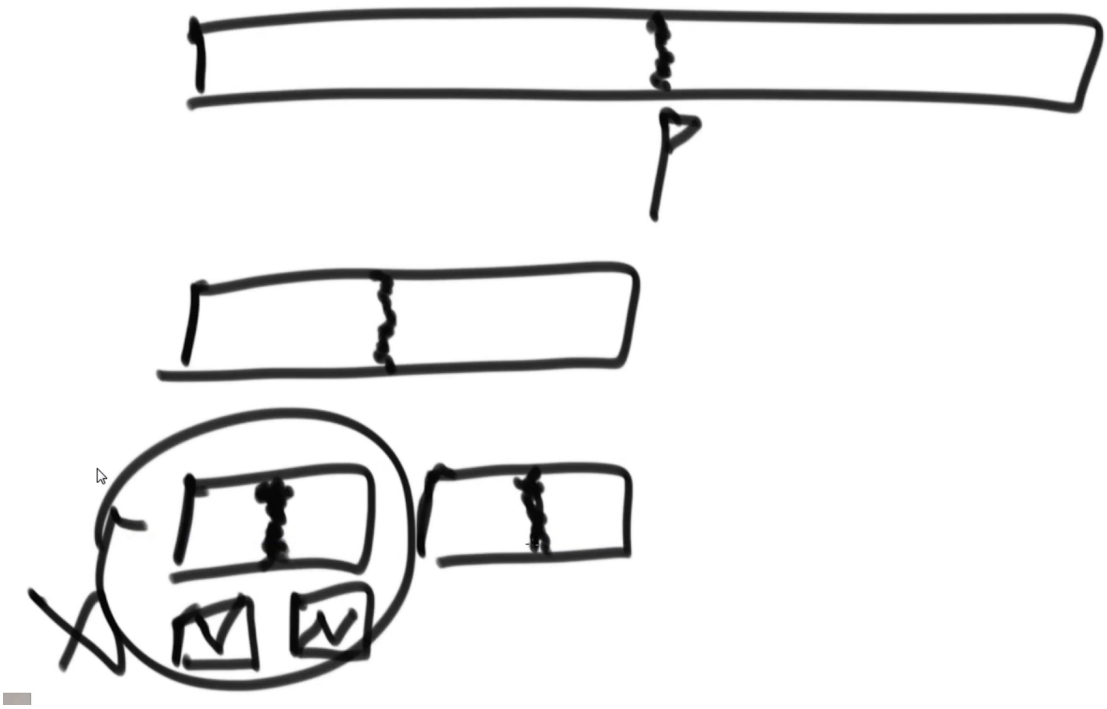
   左侧<划分值、中间==划分值、右侧>划分值

2. 对左侧范围和右侧范围，递归执行

3. 时间复杂度为O(NlogN)

快速排序空间复杂度最好、平均O(logN)，最差O(N)





需要开logN个栈空间，记录中间位置。记录了中间位置，才知道左边、右边的位置