异或运算性质

i 和 j 不能相同，如果相同就是同一块区域，就会变成0；解释如下：

因为i和j在同一块内存，相当于跟自己异或，最后会变成0，所以

arr[i]=arr[j]=arr[i]^arr[j]^arr[j]=arr[i]

arr[i]=arr[i]^arr[j]=arr[i]^arr[i]=0

异或运算性质：异或可以理解为无进位相加

异或运算性质：

1. 0^N=N

   N^N=0

2. 异或运算满足交换率和结合率

   a^b=b^a

   a^b^c=a^(b^c)=(a^b)^c

3. 一堆数异或，与异或顺序无关，结果都是一样的；同一批数异或起来，结果都是一样的

   结果上的某一位是否为1，与异或数据对应位上1的个数有关；偶数个1，则结果对应位为0；奇数个1，则结果对应位上为1；所以异或运算与顺序无关

   异或与顺序无关

这样写进行交换的前提是：a和b在内存中是两块独立的区域

第二题解析：异或完，eor=a^b，因为a不等于b，所以eor某一位必定为1

假设eor的第8位为1，则a、b的第八位必定不一样。根据第8位，将a、b分开，将整个数组分类(第8位是1、不是1)，

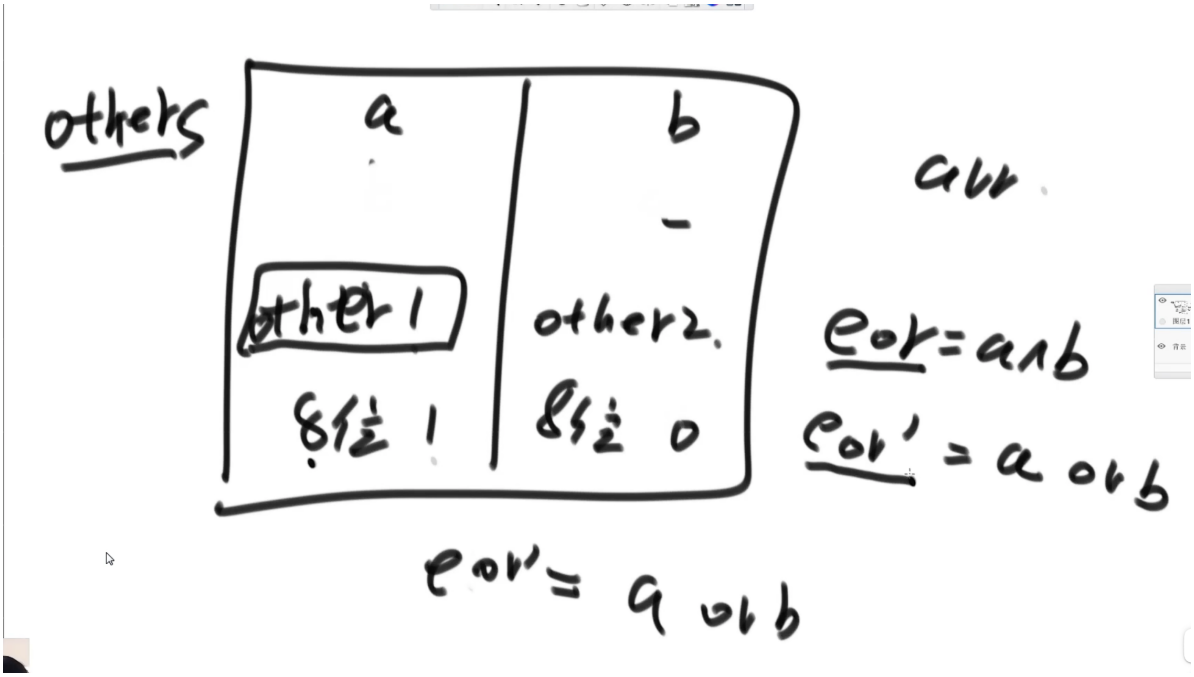
我们再开辟一个变量eor'，让eor' 异或第8位是1的那些数；只有这个数字第8位是1，才让eor' 与这个数字异或，那么eor' 最终会得到a or b，

eor & (~eor+1)提取出最右侧的1，其他位都变成了0：

```java
//左程云版本
//一种数出现奇数次，其他数都出现偶数次
public static void printOddTimesNum1(int[] arr){
    int eor=0;
    for(int cur:arr){
        eor^=arr;
    }
    System.out.println(eor);
}

//两种数出现奇数次，其他都出现偶数次
public static void printOddTimesNum2(int[] arr){
    int eor=0;
    int onlyOne=0;
    for(int curNum:arr){
        eor^=curNum;
    }

    //eor=a^b;
    //eor!=0
    //eor必然有一个位置上是1
    int rightOne=eor&(~eor+1);//提取出最右的1
    for(int cur:arr){
        //if((cur&rightOne)==1)，这样也可以
        if((cur & rihgtOne)==0){//根据最右侧的1将数组分类，之后只有那一位为0，才将其与 eor' 异或，得到a or b
            onlyOne^=cur;
        }
    }
    System.out.println(onlyOne+"    "+(eor ^ onlyOne));
}
```



二分法详解与扩展

二分法的详解与扩展

1）在一个有序数组中，找某个数是否存在

2）在一个有序数组中，找>=某个数最左侧的位置

3）局部最小值问题

```java
//在一个有序数组中，找某个数是否存在
public static boolean binarySearch(int[] arr,int num){
    int len=arr.length;
    int left=0,right=len-1;
    int mid=0;
    boolean isFind=false;
    while(left<=right){
        mid=(left+right)/2;
        if(arr[mid]==num){
            isFind=true;
            break;
        } else if(arr[mid]>num){
            right=mid-1;
        } else {
            left=mid+1;
        }
    }
    return isFind;
}
```

```java
//在一个有序数组中，找>=某个数最左侧的位置：二分到结束
public static int findIndex(int[] arr,int num){
    int len=arr.length;
    int left=0;
    int right=len-1;
    int mid=0;
    int index=-1;
    while(left<=right){
        mid=(left+right)/2;
        if(arr[mid]>=num){
            index=mid;
            right=mid-1;
        } else {
            left=mid+1;
        }
    }
    return index;
}
```

```java
//局部最小值问题：在一个无序数组中，但是任何相邻的两个数不相等，求一个局部最小的位置/索引就可以
//局部最小：arr[0]<arr[1]，则arr[0]为局部最小；arr[N-2]>arr[N-1]，则arr[N-2]为局部最小；arr[i-1]>arr[i] && arr[i]<arr[i+1]，则arr[i]局部最小
//二分法不一定有序
public static int findLocalMin(int[] arr){
    int len=arr.length;
    if(arr==null||len<2){
        return -1;
    }
    int left=0,right=len-1;
    int mid=0;
    int index=-1;

    if(arr[0]<arr[1])//判断开头
        return 0;

    if(arr[len-2]>arr[len-1])//判断结尾
        return len-1;

    //根据罗尔定理、费马定理、极值定理，肯定存在局部最小值
    while(left<=right){
        mid=(left+right)/2;
        if((arr[mid]<arr[mid-1])&&(arr[mid]<arr[mid+1])){
            index=mid;
            break;
        }
        else if(arr[mid]>=arr[mid-1])
            right=mid-1;
        else
            left=mid+1;
    }
    return index;
}
```

使用二分法，不一定需要数组有序；根据罗尔定理、费马定理、零点定理，一定存在零点

流程优化方向

- 数据状况；数据特殊，可以优化
- 求解问题标准；问题比较特殊，可以优化

使用二分法求解局部最小，将上述两个优化方向结合起来

求解问题，可以甩掉一边，就可以使用二分法，二分法小拓展

总结：一般看到有序就要考虑到二分法

对数器的概念和使用

1. 有一个你想要测的方法a
2. 实现复杂度不好但是容易实现的方法b
3. 实现一个随机样本产生器
4. 把方法a和方法b跑相同的随机样本，看看得到的结果是否一样
5. 如果有一个随机样本使得比对结果不一致，打印样本进行人工干预，改对方法a或者方法b
6. 当样本数量很多时比对测试依然正确，可以确定方法a已经正确

比对测试，对比测试，看看方法a是否正确