0x55555555)的结果描述了每两个 bit 成一组 1 的数量分布。以 n=-1(11111111111111111111111111111)为例进行说明，n=(n & 0x55555555) + ((n >>> 1) & 0x55555555)为 10101010101010101010101010101010，可以看到每两个 bit 成一组 1 的数量状况为 10，也就是每组 2 个。

接下来，0x33333333 即 00110011001100110011001100110011，所以(n & 0x33333333) + ((n >>> 1) & 0x33333333)就描述了 4 个 bit 成一组 1 的数量分布。此时 n=(n & 0x33333333) + ((n >>> 1) & 0x33333333)为 01000100010001000100010001000100，它就代表 4 个 bit 位成一组的 1 数量状况为 0100，也就是每组 4 个。

接下来 n 依次为 00001000000010000000100000001000，代表 8 个 bit 位成一组 1 的数量状况为 00001000，也就是每组 8 个。00000000000100000000000000010000 代表 16 个 bit 成一组 1 的数量状况为 0000000000010000，也就是每组 16 个。00000000000000000000000000100000 代表 32 个 bit 成一组 1 的数量状况为 00000000000000000000000000100000，也就是每组 32 个。

类似并归的过程，组与组之间的数量合并成一个大组，进行下一步的并归。

除此之外，还有很多极为逆天的算法可以解决这个问题，比如 MIT hackmem 算法等。有兴趣的读者可以去网上查找，但对面试来说，那些方法实在太偏、难、怪，所以本书不再介绍。

在其他数都出现偶数次的数组中找到出现奇数次的数

【题目】

给定一个整型数组 arr，其中只有一个数出现了奇数次，其他的数都出现了偶数次，打印这个数。

【进阶】

有两个数出现了奇数次，其他的数都出现了偶数次，打印这两个数。

【要求】

时间复杂度为 $O(N)$，额外空间复杂度为 $O(1)$。

【难度】

尉 ★★☆☆

【解答】

整数 n 与 0 异或的结果是 n，整数 n 与整数 n 异或的结果是 0。所以，先申请一个整型变量，记为 eO。在遍历数组的过程中，把 eO 和每个数异或（eO=eO^当前数），最后 eO 的值就是出现了奇数次的那个数。这是什么原因呢？因为异或运算满足交换律与结合律。为了方便说明，我们假设 A，B，C 这三个数出现了偶数次，D 这个数出现了奇数次，并且出现的顺序为：C，B，D，A，A，B，C。因为异或运算满足交换律和结合律，所以任意调整异或的顺序也不会改变最终 eO 的值，那么按照原始顺序异或得到的 eO 结果与按照如下顺序异或出的 eO 结果是相同的：A，A，B，B，C，C，D。而按照这个顺序的异或最终结果就是 D。也就是说，先异或还是后异或某一个数，对最终的结果是没有任何影响的，最终结果等同于连续异或同一个出现偶数次的数之后，再连续异或下一个出现偶数次的数，等到所有出现偶数次的数异或完，异或结果肯定是 0，最后再去异或出现奇数次的数，最终结果自然是出现奇数次的树。所以对任何排列的数组，只要这个数组有一个数出现了奇数次，另外的数出现了偶数次，最终异或结果都是出现了奇数次的数。请参看 printOddTimesNum1 方法。

```
public void printOddTimesNum1(int[] arr) {
        int eO = 0;
        for (int cur : arr) {
                eO ^= cur;
        }
        System.out.println(eO);
}
```

如果只有 a 和 b 出现了奇数次，那么最后的异或结果 eO 就是 a^b。所以，如果数组中有两个出现了奇数次的数，最终的 eO 一定不等于 0。那么肯定能在 32 位整数 eO 上找到一个不等于 0 的 bit 位，假设是第 k 位不等于 0。eO 在第 k 位不等于 0，说明 a 和 b 的第 k 位肯定一个是 1 另一个是 0。接下来再设置一个变量记为 eOhasOne，然后再遍历一次数组。在这次遍历时，eOhasOne 只与第 k 位上是 1 的整数异或，其他的数忽略。那么在第二次遍历结束后，eOhasOne 就是 a 或者 b 中的一个，而 eO^eOhasOne 就是另外一个出现奇数次的数。请参看 printOddTimesNum2 方法。

```
public static void printOddTimesNum2(int[] arr) {
```

```
        int eO = 0, eOhasOne = 0;
        for (int curNum : arr) {
                eO ^= curNum;
        }
        int rightOne = eO & (~eO + 1);
        for (int cur : arr) {
                if ((cur & rightOne) != 0) {
                        eOhasOne ^= cur;
                }
        }
        System.out.println(eOhasOne + " " + (eO ^ eOhasOne));
}
```

在其他数都出现 *k* 次的数组中找到只出现一次的数

【题目】

给定一个整型数组 arr 和一个大于 1 的整数 k。已知 arr 中只有 1 个数出现了 1 次，其他的数都出现了 k 次，请返回只出现了 1 次的数。

【要求】

时间复杂度为 $O(N)$，额外空间复杂度为 $O(1)$。

【难度】

尉 ★★☆☆

【解答】

以下的例子是两个七进制数的无进位相加，即忽略进位的相加，比如：

七进制数 a:　　　6 4 3 2 6 0 1

七进制数 b:　　　3 4 5 0 1 1 1

无进位相加结果：2 1 1 2 0 1 2

可以看出，两个七进制的数 a 和 b，在 i 位上无进位相加的结果就是(a(i)+b(i))%7。同理，k 进制的两个数 c 和 d，在 i 位上无进位相加的结果就是(c(i)+d(i))%k。那么，如果 k 个相同的 k 进制数进行无进位相加，相加的结果一定是每一位上都是 0 的 k 进制数。

理解了上述过程之后，解这道题就变得简单了，首先设置一个变量 eO，它是一个 32 位的 k 进制数，且每个位置上都是 0。然后遍历 arr，把遍历到的每一个整数都转换为 k 进