EASY:
绿色

Medium
黄色

Hard:
红色

LC 136 : Single number

PS: run O(n) time. Better without extra memory.

① brute force : data : 36% time, 18% memory.

allocate a boolean array. length : $\frac{n+1}{2}$

initialize : all → False.

first meet : set to ¬ False

second meet : set to ¬ True

find out the number with 'True'

② slight optimization : use a bunch a 64-bit integers.

$64 * m > \frac{n+1}{2}$ should be enough !

这想法貌似不好实现：你得查询这个数是否出现过，哪个bit是这个数的"桩"

Might not work

③ use XOR : bit manipulation.

why XOR works ?

性质：XOR 的 communicative : $a \wedge b = b \wedge a$

∴ 给 a : [2, 1, 4, 5, 2, 4, 1] :

$\Rightarrow 2 \wedge 1 \wedge 4 \wedge 5 \wedge 2 \wedge 4 \wedge 1 = 1 \wedge 1 \wedge 2 \wedge 2 \wedge 4 \wedge 4 \wedge 5$

$= 0 \wedge 5 = 5$

code:

```
def singleNumber(self, nums):
    res = 0
    for num in nums:
        res ^= num
    return res
```

其它解法：3种方法，7种实现.

1. 用 Counter： 或 dict.

```
def singleNumber(self, nums):
    counter = collections.Counter(nums)
    for key, val in counter.items():
        if val == 1: return key
```

2、用 set：

```
def singleNumber(self, nums):
    return 2 * sum(set(nums)) - sum(nums)
```

3. 直接异或：

```
def singleNumber(self, nums):
    res = 0
    for num in nums:
        res ^= num
```

```
        return nums
```

4. 直接修改 nums[0]，比前3种内存和时间都更优：

```
def singleNumber(self, nums):
    for i in range(1, len(nums)):
        nums[0] ^= nums[i]
    return nums[0]
```

5. 使用 reduce + lambda 表达式

```
from functools import reduce  # "functool.reduce"不可以
def singleNumber(self, nums):
    return reduce(lambda x, y: x^y, nums)
```

6. 使用 reduce + operator.xor：

```
def singleNumber(self, nums):
    return reduce(operator.xor, nums)
```