190. Reverse Bits

Input: n = 00000010100101000001111010011100
Output: 964176192 (00111001011110000010100101000000)

```
class Solution:
    def reverseBits(self, n: int) -> int:
        res = 0 # 32 bit integer
        for i in range(32):
            bit = n & 1 # bit = n % 2
            res = res | (bit << 31 - i)
            n = n >> 1
        return res

        # res = 0 # 32 bit integer
        # for i in range(32):
        #     bit = (n >> i) & 1
        #     res = res | (bit << 31 - i)

        # return res

        # res = 0 # 32 bit integer
        # for i in range(32):
        #     bit = n & (1 << i)
        #     res = res | (bit << 31 - i)
        #     n = n >> 1
        # return res
        # res, j = 0, 0
        # while n > 0:
        #     bit = n & 1
        #     res = res | (bit << 31 - j)
        #     n >>= 1
        #     j += 1
        # return res
```

268. Missing Number

Example 1:

Input: nums = [3,0,1]
Output: 2
Explanation: n = 3 since there are 3 numbers, so all numbers are in the range [0,3]. 2 is the missing number in the range since it does not appear in nums.

```
class Solution:
    def missingNumber(self, nums: List[int]) -> int:
        # for i in range(len(nums) + 1):
```

```
        #     if i not in nums:
        #         return i

        #XOR

        # res = len(nums)

        # for i in range(len(nums)):
        #     res = res ^ i
        #     res = res ^ nums[i]
        # return res

        #SUM

        res = len(nums)
        for i in range(len(nums)):
            res += (i - nums[i])
        return res
```

191. Number of 1 Bits

Example 1:

Input: n = 00000000000000000000000000001011
Output: 3
Explanation: The input binary string **00000000000000000000000000001011** has a total of three '1' bits.

```
class Solution:
    def hammingWeight(self, n: int) -> int:
        # print(n)
        # x = 0
        # k = str(n)
        # for i in range(len(k)):
        #     if k[i] == '1':
        #         x += 1
        # return x

        res = 0
        while n > 0:
            if n % 2 == 1:
                res += 1
            n = n >> 1
        return res

        # res = 0
        # while n > 0:
```

```
        #     if 1 & n == 1:
        #         res += 1
        #     #res += n & 1
        #     n = n >> 1
        # return res

        # res = 0
        # while n > 0:
        #     n = n & (n -1)
        #     res += 1
        # return res
```

136. Single Number

Example 2:

Input: nums = [4,1,2,1,2]
Output: 4

```
class Solution:
    def singleNumber(self, nums: List[int]) -> int:
        if len(nums) == 1:
            return nums[0]
        res = 0
        for n in nums:
            res = n ^ res
        return res


        # of course if we could use extra memory we would have used dictionary / set
/ hash set / map
        # s = set()
        # for n in nums:
        #     if n not in s:
        #         s.add(n)
        #     else:
        #         s.remove(n)
        # for i in s:
        #     return i
```