# 题目

给你一个整数 n ，以二进制字符串的形式返回该整数的 负二进制（base -2）表示。

注意，除非字符串就是 "0"，否则返回的字符串中不能含有前导零。

示例 1：

输入：n = 2

输出："110"

解释：(-2)2 + (-2)1 = 2

示例 2：

输入：n = 3

输出："111"

解释：(-2)2 + (-2)1 + (-2)0 = 3

示例 3：

输入：n = 4

输出："100"

解释：(-2)2 = 4

提示：

0 <= n <= 109

# 分析

## 方法一：进制转换

这个算法用于将整数 n 转换为负二进制（base -2）表示，核心思路是通过逐步提取最低位、调整数值并除以 -2，最终得到负二进制字符串。其关键在于利用位运算简化余数计算，并通过调整确保数值始终能正确收敛到 0。

算法思路详解：

1、特殊情况处理：

当 n 为 0 或 1 时，其负二进制表示就是自身（"0" 或 "1"），直接返回即可。

2、核心迭代过程：

对于 n > 1 或 n < 0 的情况，通过循环逐步计算负二进制的每一位：

- 提取最低位（余数）：

用 n & 1 计算 n 除以 2 的余数（结果为 0 或 1），这也是负二进制当前位的值（负二进制每位只能是 0 或 1）。

（原理：n & 1 等价于 n % 2，但位运算更高效，且结果一定是非负的 0 或 1。）

- 调整数值：

减去当前余数（n -= remainder），使剩余的 n 能被 -2 整除（确保下一步除法的正确性）。

- 除以 -2：

将调整后的 n 除以 -2，进入下一轮迭代，计算更高位的数值。

3、结果构建：

循环中收集的余数是负二进制的低位到高位（如先得到第 0 位，再得到第 1 位等），因此最后需要反转字符串，得到“高位到低位”的正确表示。

代码：

class Solution {

public:

string baseNeg2(int n) {

if (n == 0 || n == 1) {

return to\_string(n);

}

string res;

while (n != 0) {

int remainder = n & 1;

res.push\_back('0' + remainder);

n -= remainder;

n /= -2;

}

reverse(res.begin(), res.end());

return res;

}

};

复杂度分析

- 时间复杂度：O(log |n|)，循环次数与 n 的二进制位数成正比，每次迭代为常数操作。

- 空间复杂度：O(log |n|)，存储结果的字符串长度与循环次数一致。