# 题目

实现 pow(x, n) ，即计算 x 的 n 次幂函数。

示例 1:

输入: 2.00000, 10

输出: 1024.00000

示例 2:

输入: 2.10000, 3

输出: 9.26100

示例 3:

输入: 2.00000, -2

输出: 0.25000

解释: 2-2 = 1/22 = 1/4 = 0.25

说明:

-100.0 < x < 100.0

n 是 32 位有符号整数，其数值范围是 [−231, 231 − 1] 。

**类似题目：**剑指offer 16

# 分析

## 方法一:递归

class Solution {

public:

double myPow(double x, int n) {

long long N = n;

//入参是int类型，需转换为long long，否则运行报错，或修改入参类型long long

if(N<0) return 1.0/quickMulti(x,-N);

else return quickMulti(x,N);

}

double quickMulti(double x,int n)

{

if(n==0) return 1.0;

double y = quickMulti(x,n/2);

if(n%2==0) return y\*y;

else return y\*y\*x;

}

};

**复杂度分析**

时间复杂度：O(logn)，即为递归的层数。

空间复杂度：O(logn)，即为递归的层数。这是由于递归的函数调用会使用栈空间。

## 方法二：迭代

**思路：**

**代码：**

class Solution {

public:

double quickMul(double x, long long N) {

double ans = 1.0;

// 贡献的初始值为 x

double x\_contribute = x;

// 在对 N 进行二进制拆分的同时计算答案

while (N > 0) {

if (N % 2 == 1) {

// 如果 N 二进制表示的最低位为 1，那么需要计入贡献

ans \*= x\_contribute;

}

// 将贡献不断地平方

x\_contribute \*= x\_contribute;

// 舍弃 N 二进制表示的最低位，这样我们每次只要判断最低位即可

N /= 2;

}

return ans;

}

double myPow(double x, int n) {

long long N = n;

return N >= 0 ? quickMul(x, N) : 1.0 / quickMul(x, -N);

}

};

**复杂度分析**

时间复杂度：O(logn)，即为对n进行二进制拆分的时间复杂度。

空间复杂度：O(1)