<https://leetcode.com/problems/mirror-reflection/description/>

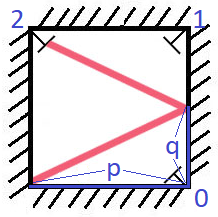
There is a special square room with mirrors on each of the four walls. Except for the southwest corner, there are receptors on each of the remaining corners, numbered 0, 1, and 2.

The square room has walls of length p and a laser ray from the southwest corner first meets the east wall at a distance q from the 0th receptor.

Given the two integers p and q, return *the number of the receptor that the ray meets first*.

The test cases are guaranteed so that the ray will meet a receptor eventually.

**Example 1:**



**Input:** p = 2, q = 1

**Output:** 2

**Explanation:** The ray meets receptor 2 the first time it gets reflected back to the left wall.

**Example 2:**

**Input:** p = 3, q = 1

**Output:** 1

**Constraints:**

1 <= q <= p <= 1000

**Attempt 1: 2025-05-03**

**Solution 1: Math (60 min)**

class Solution {

    public int mirrorReflection(int p, int q) {

        // Number of room extensions in p direction

        int m = 1;

        // Number of room extensions in q direction

        int n = 1;

        // Find the smallest m where m\*p = n\*q

        while(m \* p != n \* q) {

            n++;

            if(m \* p < n \* q) {

                m++;

            }

        }

        // If horizontal room extensions is odd and

        // vertical is even, we need to return 2

        if(m % 2 == 1 && n % 2 == 0) {

            return 2;

        }

        // If both number of extensions of our room are

        // odd, like in our image, we need to return 1

        if(m % 2 == 1 && n % 2 == 1) {

            return 1;

        }

        // If horizontal room extensions is even

        // and vertical is odd, we need to return 2

        if(m % 2 == 0 && n % 2 == 1) {

            return 0;

        }

        // Note, that it can not happen that both number

        // are even, than we have one angle before, if we

        // divide these two numbers by 2

        return -1;

    }

}

Time Complexity: O(log(min(p, q)))

Space Complexity: O(1)

**Refer to**

<https://leetcode.com/problems/mirror-reflection/solutions/146336/java-solution-with-an-easy-to-understand-explanation/>

The idea comes from this post: <https://leetcode.com/problems/mirror-reflection/discuss/141773>, and here I add some explaination.

public int mirrorReflection(int p, int q) {

int m = 1, n = 1;

while(m \* p != n \* q){

n++;

m = n \* q / p;

}

if (m % 2 == 0 && n % 2 == 1) return 0;

if (m % 2 == 1 && n % 2 == 1) return 1;

if (m % 2 == 1 && n % 2 == 0) return 2;

return -1;

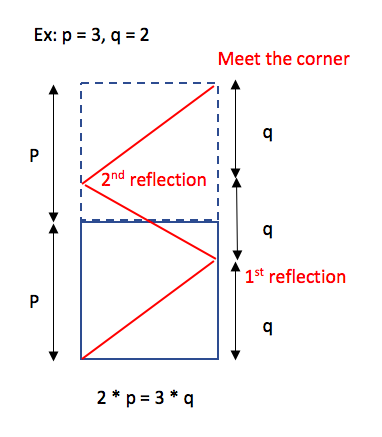
}

First, think about the case p = 3 & q = 2.

So, this problem can be transformed into finding m \* p = n \* q, where

m = the number of room extension + 1.

n = the number of light reflection + 1.



If the number of light reflection is odd (which means n is even), it means the corner is on the left-hand side. The possible corner is 2.Otherwise, the corner is on the right-hand side. The possible corners are 0 and 1.

Given the corner is on the right-hand side.If the number of room extension is even (which means m is odd), it means the corner is 1. Otherwise, the corner is 0.

So, we can conclude:

m is even & n is odd => return 0.

m is odd & n is odd => return 1.

m is odd & n is even => return 2.

Note: The case m is even & n is even is impossible. Because in the equation m \* q = n \* p, if m and n are even, we can divide both m and n by 2. Then, m or n must be odd.

--

Because we want to find m \* p = n \* q, where either m or n is odd, we can do it this way.

public int mirrorReflection(int p, int q) {

int m = q, n = p;

while(m % 2 == 0 && n % 2 == 0){

m /= 2;

n /= 2;

}

if (m % 2 == 0 && n % 2 == 1) return 0;

if (m % 2 == 1 && n % 2 == 1) return 1;

if (m % 2 == 1 && n % 2 == 0) return 2;

return -1;

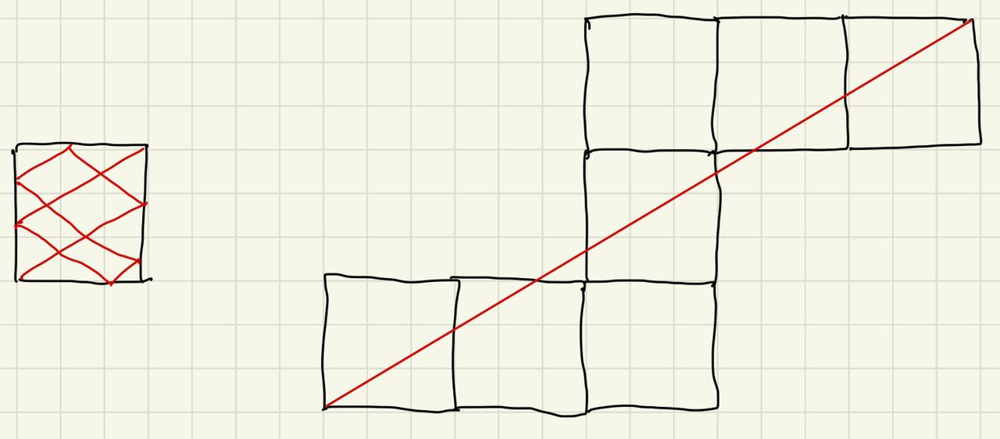
}

**A easier understanding way**

**Refer to**

<https://leetcode.com/problems/mirror-reflection/solutions/938792/python-oneliner-solution-with-diagram-explained/>

The idea of this problem, instead of reflecting our ray, we will look at it from the point of view of our ray, which is straight line: see the following image for better understanding.



How we can find what we need to return: 0, 1 or 2:

**If both number of reflections of our room are odd, like in our image, we need to return 1.**

**If horizontal line of reflections is odd and vertical is even, we need to return 0.**

**If horizontal line of reflections is even and vertical is odd, we need to return 2.**

**Note, that it can not happen that both number are even, than we have one angle before, if we divide these two numbers by 2.**

Finally, all this can be written in oneliner as follow:

**Complexity**: time and space complexity is O(1) if we assume that we can find gcd in O(1) time (which is true for int32 numbers). If you want to be more mathematically correct, it is O(log n).

class Solution:

def mirrorReflection(self, p, q):

return (q//gcd(p,q)%2 - p//gcd(p,q)%2 + 1)

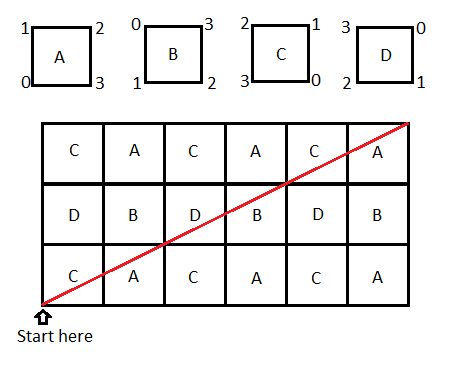
**Best picture to explain**

**Refer to**

<https://leetcode.com/problems/mirror-reflection/solutions/938821/python-pure-geometry-illustrated/>

Consider four reflections of the room. Based on the corners these are encoded as A, B, C, D. This is displayed in the figure below (first row).

As you can see we start from a room of type C. An easy way to incorporate reflections is to propagate them thru a stacked set of rooms based on the reflections. We hit a corner at an integer k such that mod(k\*q, p) = 0. In the solution below we first calculate k. Depending on the even/odd values of k and k\*q//p we find the answer.



In this case the answer will be 2, as we intersect at k=2. I drew a larger diagram for illustrative purposes. (You don't need the rest of it for this one.)

class Solution:

def mirrorReflection(self, p, q):

k = 1

while k\*q%p: k += 1

if k%2==1 and (k\*q//p)%2==0: return 0

if k%2==1 and (k\*q//p)%2==1: return 1

if k%2==0 and (k\*q//p)%2==1: return 2

**Reduction**

The if-conditionals can be compactified as

class Solution:

def mirrorReflection(self, p, q):

k = 1

while k\*q%p: k += 1

return 2 if k%2==0 else (k\*q//p)%2

**Refer to Deepseek**

This problem involves finding which receptor a laser beam will meet first in a square room with mirrors on all walls. Here's a Java solution that uses a mathematical approach to solve the problem efficiently.

**Approach**

The key observation is that we can think of the room as being reflected (mirrored) every time the laser hits a wall, instead of calculating the actual reflections. This way, the problem reduces to finding the least common multiple (LCM) of the room size (p) and the distance from the starting point (q), and then determining which receptor the beam would hit based on how many times the room was reflected in each direction.

**Solution Code**

class Solution {

public int mirrorReflection(int p, int q) {

int m = 1; // number of room extensions in p direction

int n = 1; // number of room extensions in q direction

// Find the smallest m where m\*p = n\*q

while (m \* p != n \* q) {

n++;

if (m \* p < n \* q) {

m++;

}

}

if (m % 2 == 0 && n % 2 == 1) return 0;

if (m % 2 == 1 && n % 2 == 1) return 1;

if (m % 2 == 1 && n % 2 == 0) return 2;

return -1; // should never reach here for valid inputs

}

}

**Explanation**

Finding the Meeting Point: The loop continues until we find integers m and n such that m\*p = n\*q, which represents the point where the laser beam hits a corner after traveling through reflected rooms.

Determining the Receptor:

If m is even and n is odd, the beam hits receptor 0 (left wall).

If both m and n are odd, the beam hits receptor 1 (top-right corner).

If m is odd and n is even, the beam hits receptor 2 (right wall).

This approach efficiently calculates the first receptor the laser will hit by treating the reflections as extensions of the original room and using simple parity checks to determine the correct receptor. The time complexity is O(p/q) in the worst case, but in practice it's much faster due to the use of LCM properties.

**Time and Space Complexity Analysis**

**Time Complexity: O(log(min(p, q)))**

The loop in the solution runs until m \* p = n \* q. This is equivalent to finding the \*\*Least Common Multiple (LCM)\*\* of p and q, which can be expressed as:

[

\text{LCM}(p, q) = \frac{p \cdot q}{\text{GCD}(p, q)}

]

The loop increments n and adjusts m until m \* p = n \* q, which happens after at most \text{LCM}(p, q)/p iterations.

The number of iterations is proportional to \frac{q}{\text{GCD}(p, q)} (since m = \frac{n \cdot q}{p} must be an integer).

The \*\*Euclidean algorithm\*\* for GCD takes O(log(min(p, q))) time, and since the loop is bounded similarly, the \*\*time complexity is O(log(min(p, q)))\*\*.

**Space Complexity: O(1)**

The algorithm uses a constant amount of extra space (m, n, and temporary variables), so the \*\*space complexity is O(1)\*\*.

**Why This Complexity?**

\*\*Loop Termination\*\*: The loop stops when m \* p = n \* q, which happens after at most \text{LCM}(p, q)/p steps.

\*\*GCD Relation\*\*: Since \text{LCM}(p, q) = \frac{p \cdot q}{\text{GCD}(p, q)}, the number of iterations is O\left(\frac{q}{\text{GCD}(p, q)}\right) = O\left(\frac{\text{LCM}(p, q)}{p}\right).

\*\*Euclidean GCD\*\*: The worst-case number of iterations is logarithmic in min(p, q) due to the properties of GCD.

**Comparison with Alternative Approaches**

|  |  |  |  |
| --- | --- | --- | --- |
| Approach | Time Complexity | Space Complexity | Notes |
| **LCM + Parity Check (This Solution)** | O(log(min(p, q))) | O(1) | Most efficient |
| **Simulation (Bouncing Ray)** | O(p / \text{GCD}(p, q)) | O(1) | Slower for large  p, q |
| **Mathematical (Using GCD Directly)** | O(log(min(p, q))) | O(1) | Equivalent to this solution |

**Final Answer**

\*\*Time Complexity\*\*: O(log(min(p, q))) (due to GCD-like behavior in the loop).

\*\*Space Complexity\*\*: O(1) (constant extra space).

This makes the solution \*\*optimal\*\* for large inputs. 🚀