<https://leetcode.com/problems/minimum-operations-to-make-the-integer-zero/description/>

You are given two integers num1 and num2.

In one operation, you can choose integer i in the range [0, 60] and subtract 2i + num2 from num1.

Return the integer denoting the minimum number of operations needed to make num1 equal to 0.

If it is impossible to make num1 equal to 0, return -1.

**Example 1:**

**Input:** num1 = 3, num2 = -2

**Output:** 3

**Explanation:** We can make 3 equal to 0 with the following operations:

- We choose i = 2 and subtract 22 + (-2) from 3, 3 - (4 + (-2)) = 1.

- We choose i = 2 and subtract 22 + (-2) from 1, 1 - (4 + (-2)) = -1.

- We choose i = 0 and subtract 20 + (-2) from -1, (-1) - (1 + (-2)) = 0.

It can be proven, that 3 is the minimum number of operations that we need to perform.

**Example 2:**

**Input:** num1 = 5, num2 = 7

**Output:** -1

**Explanation:** It can be proven, that it is impossible to make 5 equal to 0 with the given operation.

**Constraints:**

1 <= num1 <= 10^9

-10^9 <= num2 <= 10^9

**Attempt 1: 2025-02-12**

**Solution 1: Bit Manipulation (180 min)**

class Solution {

    public int makeTheIntegerZero(int num1, int num2) {

        // Iterate over possible values of k from 1 to 60

        for (int k = 1; k <= 60; k++) {

            // Compute the target value

            long target = num1 - (long) k \* num2;

            // Check if target is non-negative and satisfies the conditions

                return k;

            }

        }

        // If no valid k is found, return -1

        return -1;

    }

}

Time Complexity: O(1)

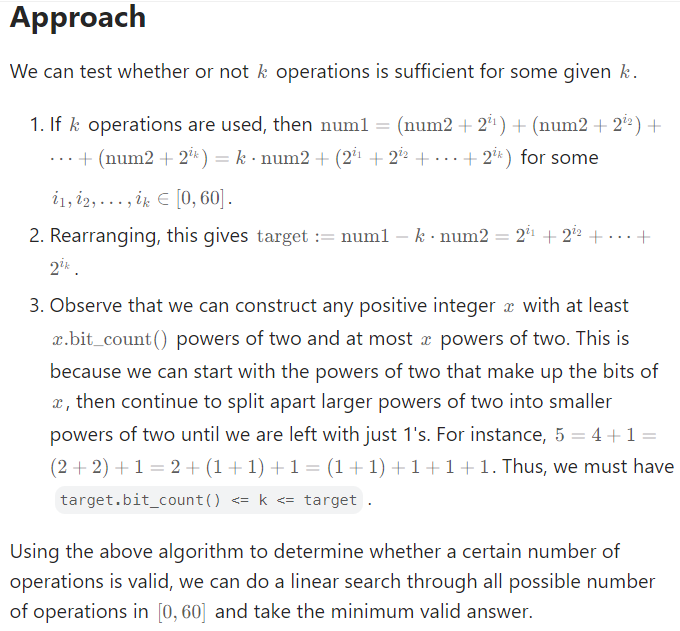
Space Complexity: O(1)

**Refer to**

<https://leetcode.com/problems/minimum-operations-to-make-the-integer-zero/solutions/3679094/linear-search-on-answer-ft-bit-count/>

**Intuition**

After a bit of playing around, it seems hard to find an explicit formula for the answer. As with many computer science problems, we can instead try to search over the answer space, trading runtime for ease of implementation.



class Solution:

def makeTheIntegerZero(self, num1: int, num2: int) -> int:

for k in range(61):

target = num1 - k \* num2

if target >= 0 and target.bit\_count() <= k <= target:

return k

return -1

**Why the answer is bounded from above by 60**

For those who are asking, here's an explanation of why the answer is bounded from above by 60. It can be seen intuitively but may require a bit of casework to prove rigorously.

Assume there is no valid answer for k <= 60. Then, at least one of the following hold for k = 60:

target < 0,

target.bit\_count() > k, or

k > target.

(2) is impossible since target := num1 - k \* num2 is bounded from above by the input constraints.

If (1) is true, then num1 - k \* num2 < 0 implies num2 > 0, meaning target is strictly decreasing while k is strictly increasing. If we have not found a solution by this point, this implies there is no solution past 60.

If (3) is true, k > num1 - k \* num2 implies k \* (num2 + 1) > num1, meaning either num2 = 0, in which case k = num1.bit\_count() will have previously been a valid answer, or num2 > 0. Again, target is strictly decreasing while k is strictly increasing; there is no solution.

In all cases, there is either a contradiction or no solution; k is bounded from above. As a matter of fact, I believe there can be a tighter bound on k as long as the argument for (2) holds.

**Refer to Deepseek**

**Key Observations:**

Operation Mechanics:

Each operation subtracts 2^i + num2 from num1.

The value of i can range from 0 to 60 (since 2^60 is a very large number, and num1 is bounded by 10^9).

Target Calculation:

After k operations, the total subtracted value is k \* num2 + sum of (2^i) for all chosen i.

For num1 to be reduced to 0, the following must hold:

num1 - k \* num2 = sum of (2^i) for all chosen i.

Let target = num1 - k \* num2. This target must be expressible as the sum of k powers of 2 (i.e., target.bit\_count() <= k).

Constraints:

target must be non-negative (target >= 0).

The number of set bits in target (i.e., target.bit\_count()) must be less than or equal to k.

k must be less than or equal to target (k <= target), because you cannot use more operations than the value of target.

Bounded Search:

Since i is bounded by 60, we can iterate over possible values of k from 1 to 60 and check if the conditions above are satisfied.

**Solution Approach:**

Iterate over possible values of k from 1 to 60.

For each k, compute target = num1 - k \* num2.

Check if target is non-negative, and if the number of set bits in target is less than or equal to k.

If both conditions are satisfied, return k.

If no valid k is found after the loop, return -1.

**Java Implementation:**

class Solution {

public int makeTheIntegerZero(int num1, int num2) {

// Iterate over possible values of k from 1 to 60

for (int k = 1; k <= 60; k++) {

// Compute the target value

long target = num1 - (long) k \* num2;

// Check if target is non-negative and satisfies the conditions

return k;

}

}

// If no valid k is found, return -1

return -1;

}

}

**Explanation of the Code:**

Loop Over k:

We iterate over k from 1 to 60 because the maximum value of i is 60.

Compute target:

For each k, we compute target = num1 - k \* num2.

We use long to avoid integer overflow, as k \* num2 could be very large.

Check Conditions:

target >= 0: Ensures that target is non-negative.

Long.bitCount(target) <= k: Ensures that the number of set bits in target is less than or equal to k.

k <= target: Ensures that k is not greater than target.

Return k:

If all conditions are satisfied, return k as the minimum number of operations.

Return -1:

If no valid k is found after the loop, return -1.

**Example Walkthrough:**

**Example 1:**

Input: num1 = 3, num2 = -2

Iteration:

For k = 1:

target = 3 - 1 \* (-2) = 5

Long.bitCount(5) = 2 (binary 101 has 2 set bits)

Condition 2 <= 1 fails.

For k = 2:

target = 3 - 2 \* (-2) = 7

Long.bitCount(7) = 3 (binary 111 has 3 set bits)

Condition 3 <= 2 fails.

For k = 3:

target = 3 - 3 \* (-2) = 9

Long.bitCount(9) = 2 (binary 1001 has 2 set bits)

Condition 2 <= 3 and 3 <= 9 holds.

Output: 3

**Example 2:**

Input: num1 = 5, num2 = 7

Iteration:

For all k from 1 to 60:

target = 5 - k \* 7 will always be negative or fail the conditions.

Output: -1

**Complexity Analysis:**

Time Complexity: O(1)

The loop runs at most 60 times, which is a constant.

Space Complexity: O(1)

Only a few variables are used, and no additional space is required.

This solution is efficient and works within the problem constraints.

**Approach Recap:**

Operation Breakdown:

Each operation subtracts num2 + 2^i from num1.

After k operations, the total subtracted value is:

k \* num2 + (2^{i\_1} + 2^{i\_2} + ... + 2^{i\_k})

Rearranging, we get:

num1 - k \* num2 = 2^{i\_1} + 2^{i\_2} + ... + 2^{i\_k}

Let target = num1 - k \* num2.

Conditions for Validity:

target must be non-negative (target >= 0).

The number of set bits in target (i.e., target.bit\_count()) must be less than or equal to k.

k must be less than or equal to target (k <= target).

Linear Search:

Iterate over possible values of k from 1 to 60.

For each k, compute target = num1 - k \* num2.

Check if target satisfies the conditions above.

If valid, return k as the minimum number of operations.

If no valid k is found, return -1.

**Verification of the Solution:**

The solution I provided earlier implements this approach exactly. Here's how it maps to the approach:

Iteration Over k:

The loop for (int k = 1; k <= 60; k++) iterates over all possible values of k from 1 to 60.

Compute target:

long target = num1 - (long) k \* num2; computes the value of target for each k.

**Check Conditions:**

**target >= 0: Ensures target is non-negative.**

**Long.bitCount(target) <= k: Ensures the number of set bits in target is less than or equal to k.**

**k <= target: Ensures k is not greater than target.**

Return Result:

If all conditions are satisfied, return k.

If no valid k is found after the loop, return -1.

**Example Walkthrough:**

**Example 1:**

Input: num1 = 3, num2 = -2

Iteration:

For k = 1:

target = 3 - 1 \* (-2) = 5

Long.bitCount(5) = 2 (binary 101 has 2 set bits)

Condition 2 <= 1 fails.

For k = 2:

target = 3 - 2 \* (-2) = 7

Long.bitCount(7) = 3 (binary 111 has 3 set bits)

Condition 3 <= 2 fails.

For k = 3:

target = 3 - 3 \* (-2) = 9

Long.bitCount(9) = 2 (binary 1001 has 2 set bits)

Condition 2 <= 3 and 3 <= 9 holds.

Output: 3

**Example 2:**

Input: num1 = 5, num2 = 7

Iteration:

For all k from 1 to 60:

target = 5 - k \* 7 will always be negative or fail the conditions.

Output: -1

**Conclusion:**

The solution I provided earlier is consistent with the approach described in the snapshot. It correctly implements the logic to determine the minimum number of operations required to reduce num1 to 0 using the given operation. If no valid solution exists, it returns -1. This approach is efficient and works within the problem constraints.

**Refer to**

[L650.2 Keys Keyboard (Ref.L991,L2749)](note://WEBacf225934747d5abde829b9fc81664a5)

[L991.Broken Calculator (Ref.L650,L2749)](note://WEBf7687f1214cd00462721a353fd513567)