2429. Minimize XOR

Given two positive integers `num1` and `num2`, find the positive integer `x` such that:

- `x` has the same number of set bits as `num2`, and
- The value `x XOR num1` is **minimal**.

Note that `XOR` is the bitwise XOR operation.

Return *the integer* `x`. The test cases are generated such that `x` is **uniquely determined**.

The number of **set bits** of an integer is the number of `1`'s in its binary representation.

Example 1:

```
Input: num1 = 3, num2 = 5
Output: 3
Explanation:
The binary representations of num1 and num2 are 0011 and 0101, respectively.
The integer 3 has the same number of set bits as num2, and the value 3 XOR 3 = 0 is minimal.
```

Example 2:

```
Input: num1 = 1, num2 = 12
Output: 3
Explanation:
The binary representations of num1 and num2 are 0001 and 1100, respectively.
The integer 3 has the same number of set bits as num2, and the value 3 XOR 1 = 2 is minimal.
```

Constraints:

- `1 <= num1, num2 <=` 10^9

2429. Minimize XOR

```
/*
   Bit manipulation
   Time compelxity: O(log2 max(a,b))
   Space compelxity: O(1)
*/
class Solution {
   public:
      // If k>=n:
      // initialize `ans` to `a`
      // set all m=(k-n) unset bits, from right to left
      int case1(int a,int b,int n,int k){
         int ans=a;
         int i=0; // To track the rightmost bit
         int m=k-n;
         // While the rightmost (k-n) bits are still unset
         while(m){
            // If rightmost bit in not set
            if(!((ans>>i)&1)){
               // Set it
               int x=(ans>>i)|1;
               // Append `i` 0s to the right and build the answer
               ans|=(x<<i);
               m--; // one bit is set
            }
            i++; // Next rightmost bit
         }
         return ans;
      }
```

```
    // If k<n:
    // initialize `ans` to `a`
    // Unset all m=(n-k) set bits, from right to left
    int case2(int a,int b,int n, int k){
        int ans=a;
        int m=n-k;
        int i=0;
        // Keep shifting `a` to the right until reaching the (n-k)-th set bit
        // This helps us to find the number of 0s to add afer finding the (n-k)-th set bit
        while(m){
            if(ans&1) m--;
            ans>>=1;
            i++;
        }
        // Shift `a` by `i` positions the left to add `i` 0s at the left
        ans<<=i;
        return ans;
    }

    int minimizeXor(int num1, int num2){
        // n=#setbits in nums1
        int n=__builtin_popcount(num1);
        // k=#setbits in nums2
        int k=__builtin_popcount(num2);

        // Manage cases
        if(k>=n) return case1(num1,num2,n,k);
        return case2(num1,num2,n,k);
    }
};
```