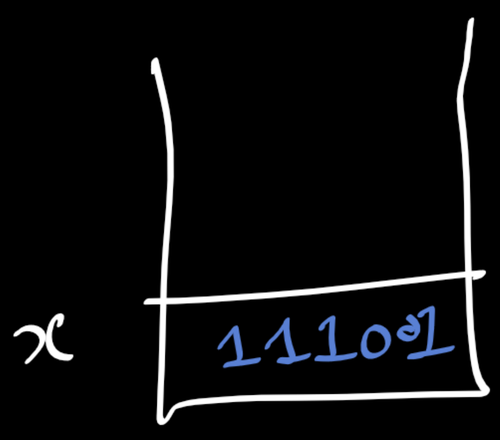
Basics of Bit Manipulation

Let us say we have a variable $x = 57$.

In machine, it will be stored as a binary number



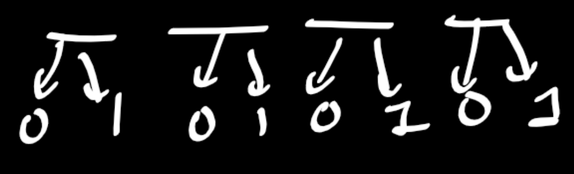
2	57	
2	28	1
2	14	0
2	7	0
2	3	1
2	1	1
	0	1

$(111001)_2$

* Store in binary
* return / print in decimal

Sysout(x) → 57

byte	short	int	long
8 bits (2^8)	16 bits (2^{16})	32 bits (2^{32})	64 bits (2^{64})



if we have 4 bits, 2^4 distinct nos are possible.

So, if we have N bits, 2^N distinct nos are possible

Let us assume that there is a datatype called nibble whose size is 4bits.

1000

No is -ve

For calculating value

2's comp of 1000 is

1000

↓

value is 8

So ans is -8

0	0 00	→ 0
0	0 0 1	→ 1
0	0 1 0	→ 2
0	0 1 1	→ 3
0	1 0 0	→ 4
0	1 0 1	→ 5
0	1 1 0	→ 6
0	1 1 1	→ 7
-8	1 0 00	→ 8 -0
-7	1 0 0 1	→ 9 -1
-6	1 0 1 0	→ 10 -2
-5	1 0 1 1	→ 11 -3
-4	1 1 0 0	→ 12 -4
-3	1 1 0 1	→ 13 -5
-2	1 1 1 0	→ 14 -6
-1	1 1 1 1	→ 15 -7

* App → 1 take nos from 0 to $2^N - 1$ (all +ve)
* App → 2 use MSB for sign & all other bits for value.
  - Two 0s, +0 & -0
  - After 0111, 1000 has +1 val in Bin however -0 & -1 are -1 val diff.

App → 3 use MSB for sign & complete no's 2's comp as value.

Range: -2^{N-1} to $2^{N-1}-1$

byte (1byte = 8bits) = -2^7 to 2^7-1

Short (2bytes = 16 bits) = -2^{15} to $2^{15}-1$

int (4bytes = 32 bits) = -2^{31} to $2^{31}-1$

long (8bytes = 64 bits) = -2^{63} to $2^{63}-1$

What happens when we try to store a No out of Range?

0 000	0
0 001	1
0 010	2
0 011	3
0 100	4
0 101	5
0 110	6
0 111	7
1 000	-8
1 001	-7
1 010	-6
1 011	-5
1 100	-4
1 101	-3
1 110	-2
1 111	-1

Let us say that we want to store 12 in nibble

2	12	
2	6	0
2	3	0
2	1	1
2	0	1

$(1100)_2$

(1100) in nibble is −4. So, it will store 1100, but when asked to print, we get −4.

If we try to store 16 i.e 10000, only last 4 bits will be stored.

Convert Bin to Decimal

MSB = 0
→ convert to dec

MSB = 1
- 2's comp
- Convert to dec
- -ve Sign

Convert Dec to Bin

+ve
- Convert to Bin
- fit in bits

-ve
- Leave the sign
- Convert to Bin
- fit in bits
- Store 2s comp.

	&	^	<<	>>
or	AND	XOR	left shift	right shift

>>>	~	-x
trs	1's complint	2's complint

Left Shift

$x = 00\,1\,0\,1\,0\,1\,1$

$x << 3 = 0\,1\,0\,1\,1\,0\,0\,0$

Right Shift

$y = 10\,1\,0\,0\,110$

$y >> 3 \Rightarrow$ drops last 3 bits
$\Rightarrow$ Brings 1st 3 bits equal to current MSB & shifts

prev MSB

$y = 111\,1\,0\,1\,0\,0$

↳ 3 bits eq to MSB

Triple RS

$y = 1\,0\,1\,0\,0\,110$

$y >>> 3 \Rightarrow$ Drops 3 LSBs
$\Rightarrow$ Brings 3 0s at MSBs always.

$y = 000\,1\,0\,1\,0\,0$

$\Rightarrow$ ON, OFF, CHECK/GET & TOGGLE are already discussed in Tries Notes.

Basics Of Bit Manipulation

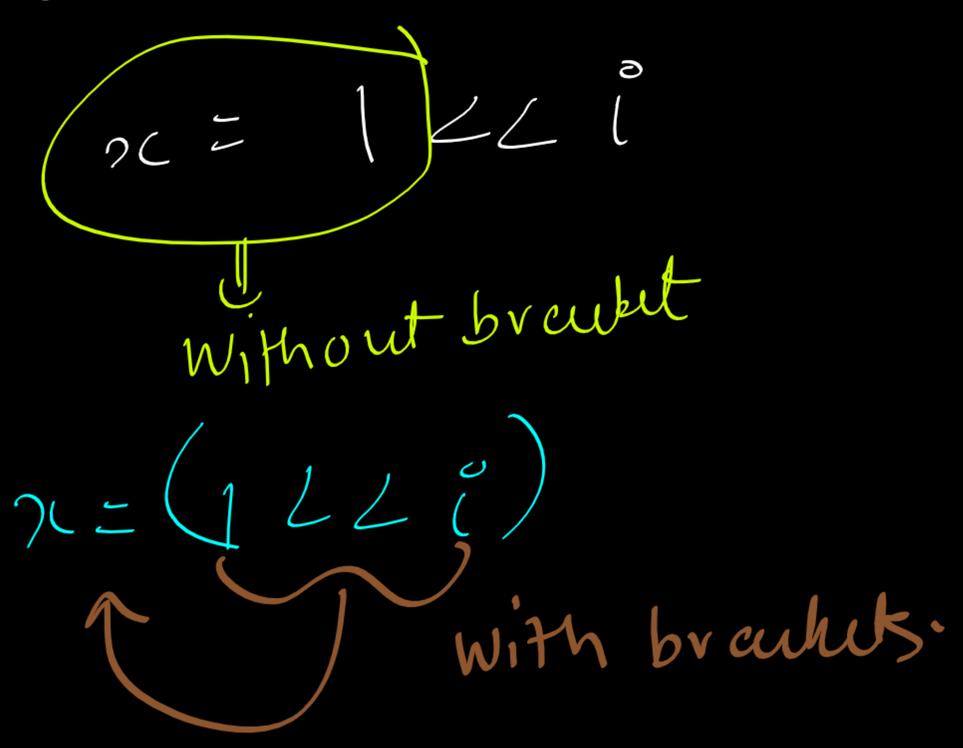
Easy

Prev Next

1. You are given a number n.
2. Print the number produced on setting its i-th bit.
3. Print the number produced on unsetting its j-th bit.
4. Print the number produced on toggling its k-th bit.
5. Also, Check if its m-th bit is on or off. Print 'true' if it is on, otherwise print 'false'.

```java
public static void main(String[] args){
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    int i = scn.nextInt();
    int j = scn.nextInt();
    int k = scn.nextInt();
    int m = scn.nextInt();

    //write your code here

    //setting the ith bit
    int maskForSetting = (1 << i); //or with the number
    int maskForOff = ~(1 << j); //and with the number
    int maskForToogle = (1 << k); //XOR with the number;
    int maskForChecking = (1 << m); //and with number

    int setBitNo = (n | maskForSetting);
    int unsetBitNo = (n & maskForOff);
    int toggledNo = (n ^ maskForToogle);

    System.out.println(setBitNo);
    System.out.println(unsetBitNo);
    System.out.println(toggledNo);

    if((n & maskForChecking) != 0) {
        System.out.println(true); //bit was set
    } else {
        System.out.println(false); //bit was off
    }
}
```

⇒ We have to use brackets ()
because the precedence of bitwise operators is even lower than assignment operator ("=").



Binary Representation - GfG

Binary representation

School Accuracy: 50.04% Submissions: 3382 Points: 0

Write a program to print Binary representation of a given number N.

Example 1:

Input:
N = 2
Output:
00000000000000000000000000000010

```java
class Solution{
    static String getBinaryRep(int N){
        // code here
        int temp = N;
        String res = "";
        while(temp > 0) {
            int rem = temp % 2;
            temp = temp / 2;
            res = "" + rem + res;
        }

        int len = res.length();
        int limit = 30-len;

        for(int i=1;i<=limit;i++) {
            res = "0" + res;
        }

        return res;
    }
```

Right Most Set Bit (& RMSB Mask)

$x = 76$

2	76	
2	38	0
2	19	0
2	9	1
2	4	1
2	2	0
2	1	0
	0	1

$76 = (1001100)_2$

↑ Right Most Set Bit

Right Most Set Bit Mask = 0 0 0 0 1 0 0

All 0s except RMSB

formula to calculate RMSB Mask of a no x in O(1) time is = $x \,\&\, x''$

i.e x & 2s complement of x.

Two's comp of x can be calculated as $\sim x + 1$

$x =$	A 1's and 0's	1	B 0's	①
$\sim x =$	A 0's and 1's	0	B 1's	
$x'' = \sim x + 1 =$	A 0's and 1's	1	B 0's	②

RMSB Mask = A 0's (1) B 0's ① & ②

So, $rmsb = x \,\&\, (\sim x + 1)$

2's comp can also be calculated as $-x$

So, RMSB Mask = $x \,\&\, -x$

```java
import java.io.*;
import java.util.*;

public class Main {

  public static void main(String[] args){
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();

    //write your code here
    int res = (n & -n);
    System.out.println(Integer.toBinaryString(res));
  }

}
```

} code for RMSB Mask

⇒ calculate RMSB Mask.

⇒ Convert it to Binary String

⇒ The length of the string is our answer.

Find first set bit

Easy Accuracy: 52.49% Submissions: 42460 Points: 2

Given an integer an **N.** The task is to return the position of **first set bit found from the right side** in the binary representation of the number.

Note: If there is no set bit in the integer N, then return 0 from the function.

Example 1:

```
Input: N = 18
Output: 2
Explanation: Binary representation of
18 is 010010,the first set bit from the
right side is at position 2.
```

```java
class Solution
{
    //Function to find position of first set bit in the given number.
    public static int getFirstSetBit(int n){

        // Your code here
        int rmsbm = (n & -n);
        String res = Integer.toBinaryString(rmsbm);

        if(res.equals("0") == true) return 0;

        return res.length();
    }
}
```

Find position of set bit

Basic Accuracy: 50.02% Submissions: 27212 Points: 1

Given a number **N** having only one '1' and all other '0's in its binary representation, find position of the only set bit. If there are 0 or more than 1 set bit the answer should be -1. Position of set bit '1' should be counted starting with 1 from LSB side in binary representation of the number.

1 1 0 1 0 1 0 0

⇒ There are multiple 1's

⇒ Hence the answer should be -1

Also, ans is -1 when there are more than one 1s.

Kernighan's Algorithm

This algorithm is used to count the no of set bits in a Number, without going through all the bits of that number.

Algorithm is as follows

```
While (N != 0) {
    Calculate RMSB Mask
    N = N - RMSB Mask
    count++;
}
```

x = 1 00 1 00 100

① x = 1 00 1 00 1 00

- rmsbm = 0 00 0 00 1 00

New x = 1 00 1 00 0 00 0

counter = 1

② x = 1 00 1 00 0 00

- rmsbm = 0 00 1 00 0 00

x = 1 00 0 00 0 00

counter = 2

③ x = 1 00 0 00 0 00

- rmsbm = 1 00 0 00 0 00

x = 0 00 0 00 0 00

counter = 3

```java
import java.io.*;
import java.util.*;

public class Main {

  public static void main(String[] args){
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();

    //write your code here
    int counter = 0;
    while(n != 0) {
        int rmsbm = (n & -n);
        n -= rmsbm;
        counter++;
    }

    System.out.println(counter);
  }

}
```

Find position of set bit

Basic Accuracy: 50.02% Submissions: 27212 Points: 1

Given a number **N** having only one '1' and all other '0's in its binary representation, find position of the only set bit. If there are 0 or more than 1 set bit the answer should be -1. Position of set bit '1' should be counted starting with 1 from LSB side in binary representation of the number.

```java
class Solution {
    static int findPosition(int N) {
        // code here
        int counter = 0;
        int temp = N;
        //kerninghan's algorithm
        while(temp != 0) {
            int rmsbm = (temp & -temp);
            temp -= rmsbm;
            counter++;
        }

        if(counter == 0 || counter > 1) return -1;

        int rmsbm = (N & -N);
        String res = Integer.toBinaryString(rmsbm);
        return res.length();
    }
};
```

① Use kerninghans' algo to count no of set bits.

② Use rmsb Mask to find 1st set bit pos.

Number of 1 Bits

Easy Accuracy: 64.8% Submissions: 38238 Points: 2

Given a positive integer N, print count of set bits in it.

```java
// User function Template for Java
class Solution {
    static int setBits(int N) {
        // code here
        int counter = 0;
        while(N != 0) {
            int rmsbm = (N & -N);
            N -= rmsbm;
            counter++;
        }

        return counter;
    }
}
```

Kerninghans' Algo. (same code)

No of set bits is also known as HammingWeight.

191. Number of 1 Bits

Easy 2998 792 Add to List Share

Write a function that takes an unsigned integer and returns the number of '1' bits it has (also known as the Hamming weight).

Note:

- Note that in some languages, such as Java, there is no unsigned integer type. In this case, the input will be given as a signed integer type. It should not affect your implementation, as the integer's internal binary representation is the same, whether it is signed or unsigned.
- In Java, the compiler represents the signed integers using 2's complement notation. Therefore, in **Example 3**, the input represents the signed integer. `-3`.

```java
public class Solution {
    // you need to treat n as an unsigned value
    public int hammingWeight(int n) {
        int counter = 0;
        while(n != 0) {
            int rmsbm = (n & -n);
            n -= rmsbm;
            counter++;
        }

        return counter;
    }
}
```

Same code

461. Hamming Distance

Easy 3032 197 Add to List Share

The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Given two integers `x` and `y`, return *the **Hamming distance** between them.*

⇒ Xor will give 1s at all those bits that are diff. Then count set bits in xor.

```java
class Solution {
    public int hammingDistance(int x, int y) {
        int xor = (x ^ y);

        int hDist = 0;
        while(xor != 0) {
            int rmsbm = (xor & -xor);
            xor -= rmsbm;
            hDist++;
        }

        return hDist;
    }
}
```

477. Total Hamming Distance

Medium 1626 80 Add to List Share

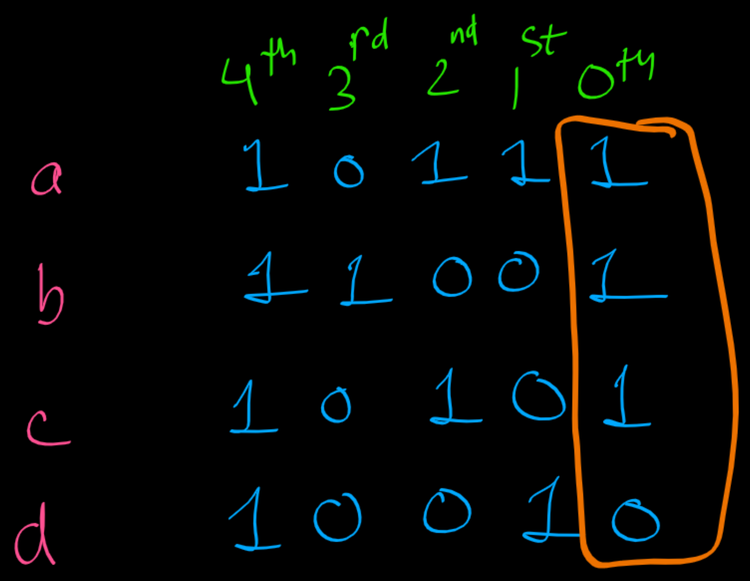
The Hamming distance between two integers is the number of positions at which the corresponding bits are different.

Given an integer array `nums`, return *the sum of* ***Hamming distances*** *between all the pairs of the integers in* `nums`.

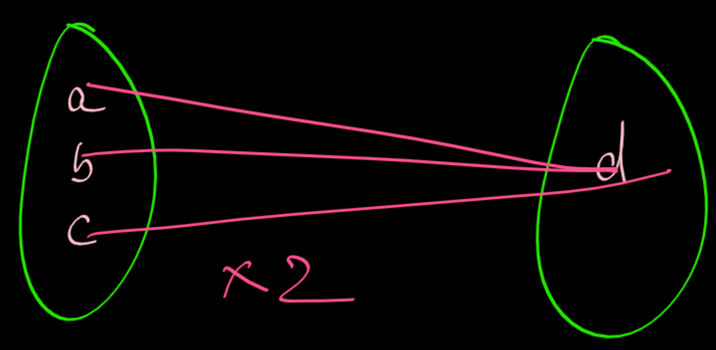
$O(N^2)$ approach is generating all pairs and finding hamming distance. ⇒ TLE

Optimized Approach

We will consider bit by bit all the pairs-



Consider 0th bit



ad da } all the pairs
bd db
cd dc }

Do this for all 32 bits. Hence the TC is $O(32*N)$

*2 is for Pepcoding ques. On leetcode, consider a|b & don't consider b,a.

```java
class Solution {
    public int totalHammingDistance(int[] nums) {
        long res = 0;

        for(int i=0;i<32;i++) {
            long counton = 0;

            for(int val : nums) {
                int bit = ((val & (1 << i)) == 0) ? 0 : 1;
                if(bit == 1) {
                    counton++;
                }
            }

            long countoff = nums.length - counton;
            long curr = counton * countoff;
            res += curr;
        }

        return (int)res;
    }
}
```

Code for Total Hamming Distance - Leetcode.

338. Counting Bits

Easy 6833 318 Add to List Share

Given an integer `n`, return *an array* `ans` *of length* `n + 1` *such that for each* `i` (`0 <= i <= n`), `ans[i]` *is the* ***number of*** `1` ***'s*** *in the binary representation of* `i`.

```java
class Solution {

    public int kerninghans(int n) {
        int counter = 0;
        while(n != 0) {
            int rmsbm = (n & -n);
            n -= rmsbm;
            counter++;
        }

        return counter;
    }

    public int[] countBits(int n) {
       int[] res = new int[n+1];

       for(int i=0;i<=n;i++) {
           res[i] = kerninghans(i);
       }

       return res;
    }
}
```

Bleak Numbers

Bleak Numbers

Medium Accuracy: 55.02% Submissions: 2886 Points: 4

Output: 0
Explanation: 3 is a Bleak number as 2 + countSetBit(2) = 3.

Your Task:
You don't need to read or print anything. Your task is to complete the function **is_bleak()** which takes n as input parameter and returns 1 if n is **not** a Bleak number otherwise returns 0.

Expected Time Complexity: O(log(n) * log(n))
Expected Space Complexity: O(1)

```java
class Solution
{
    public int kerninghans(int n) {
        int counter = 0;
        while(n != 0) {
            int rmsbm = (n & -n);
            n -= rmsbm;
            counter++;
        }

        return counter;
    }

    public int is_bleak(int n)
    {
        // Code here
        for(int i=1;i<=n;i++) {
            int setBitCount = kerninghans(i);
            int res = i + setBitCount;
            if(res == n) return 0;
        }

        return 1;
    }
}
```

↳ for every no less than or equal to N,
check if the sum of that No & its set bits is equal to N or Not.

Small optimization.

Auxiliary Space: O(1)
Method 2 (Efficient)
The idea is based on the fact that the largest count of set bits in any number smaller than n cannot exceed ceiling of $Log_2 n$. So we need to check only numbers from range n – ceilingLog2(n) to n.

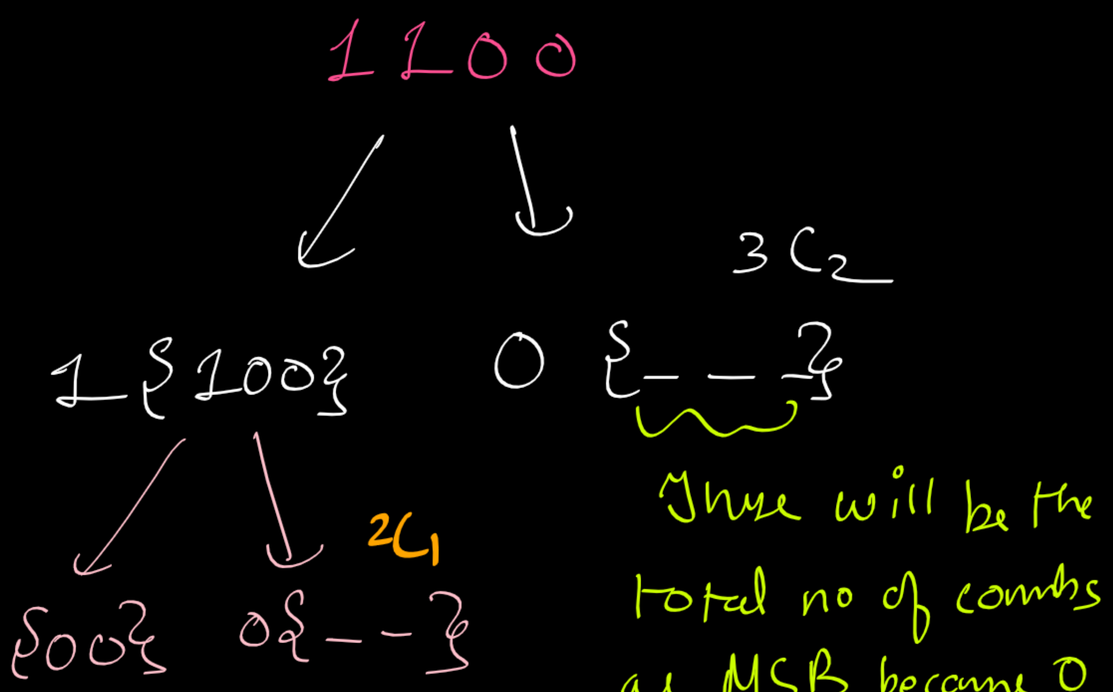
Same Bits (Shreyansh & his Bits - GfG)

Shreyansh and his bits

Medium Accuracy: 38.73% Submissions: 395 Points: 4

Shreyansh has an integer N. He is really curious about the binary representation of integers. He sees that any given integer has a number of set bits. Now he wants to find out that how many positive integers, strictly less than N, have the **same number of set bits as N**.
He is a little weak in maths. Help him find the number of integers.
Note : Since N takes large values, brute force won't work.

Say, input is 12 → 1100

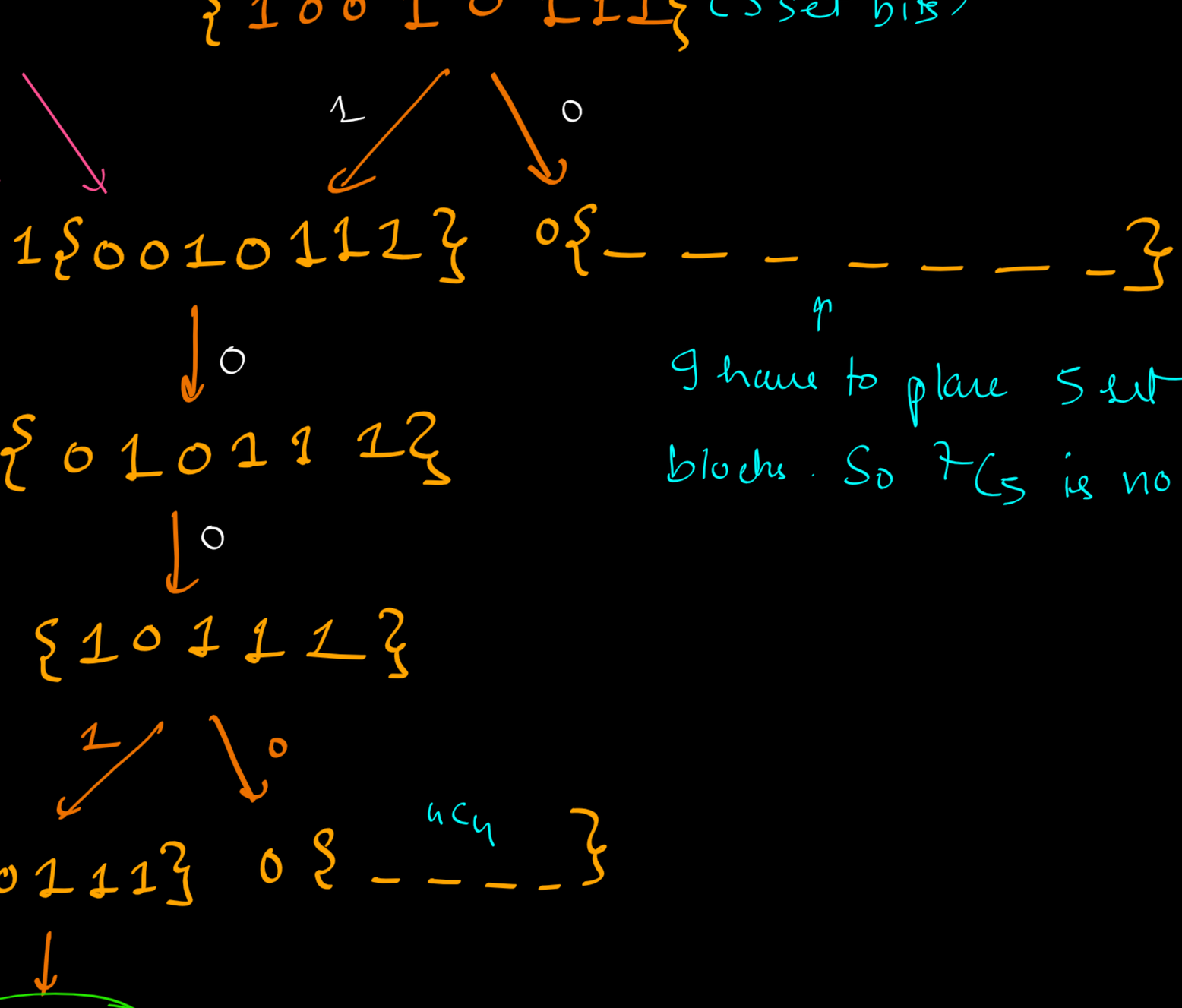


There will be the total no of combs as MSB became 0 i.e it has contributed in producing a small no.

f(10010111)

{10010111} (5 set bits)

find count of nos with 4 set bits recursively.



9 have to place 5 set bits in 7 blocks. So 7C5 is no of ways to do it.

111 ⇒ Base case as there is no smaller no than 111 with 3 set bits.

Power of Two

231. Power of Two

Easy 3178 288 Add to List Share

Given an integer `n`, return *`true`* *if it is a power of two. Otherwise, return* *`false`*.

An integer `n` is a power of two, if there exists an integer `x` such that `n == 2^x`.

if a no is a power of 2 then

$n \& (n-1) = 0$

```java
class Solution {
    public boolean isPowerOfTwo(int n) {
        if(n < 1) return false;
        if((n & (n-1)) == 0) return true;
        return false;
    }
}
```

Highest Power of 2 Less than or Equal to Given No.

```java
import java.util.*;

public class Main {

    Run | Debug
    public static void main(String[] args) throws Exception {
        Scanner scn = new Scanner(System.in);
        int n = scn.nextInt();
        System.out.println(highestPowerOf2(n));
        scn.close();
    }

    public static int highestPowerOf2(int n) {
        String bin = Integer.toBinaryString(n);

        int res = 0;
        for(int i=1;i<=bin.length();i++) {
            int curr = (1 << i);
            if(curr > n) break;
            res = curr;
        }

        return res;
    }
}
```

Keep on left shifting and generating the nos. Stop when you are at a no which is greater than N

Gray Code { 1 bit - diff }

1 bit

0

1

2 bit

00

01 } ∝ Not valid gray code.
10

00
01
11
10 } Valid 2 bit gray code.

3 bit

000
001
011
010 } Valid 3-bit gray code
110
111
101
100

3 bit Gray Code = 0 [2 bit gray code] + 1 [Reverse order of 2 bit gray code]

N bit Gray Code = 0 [N-1 bit gray code] + 1 [Reverse of N-1 bit gray code]

[000, 001, 011, 010, 110, 111, 101, 100]

gc(3) → 0[00, 01, 11, 10] = [000, 001, 011, 010] ⑥
→ 1[10, 11, 01, 00] = [110, 111, 101, 100] ②

gc(3) → [00, 01, 11, 10]
→ 0 [0, 1] = [00, 01] ①
→ 1 [1, 0] = [11, 10] ②

gc(2) → [0, 1]

gc(1)

This is when we have List<Integer> as Strings
Not that imp as we will have to deal with nos only

Unique Number - 1

136. Single Number

Easy 9687 348 Add to List Share

Given a **non-empty** array of integers `nums`, every element appears *twice* except for one. Find that single one.

You must implement a solution with a linear runtime complexity and use only constant extra space.

```java
class Solution {
    public int singleNumber(int[] nums) {

        int xor = nums[0];
        for(int i=1;i<nums.length;i++) {
            xor = (xor ^ nums[i]);
        }

        return xor;
    }
}
```

Unique Number - 2

137. Single Number II

Medium 4057 477 Add to List Share

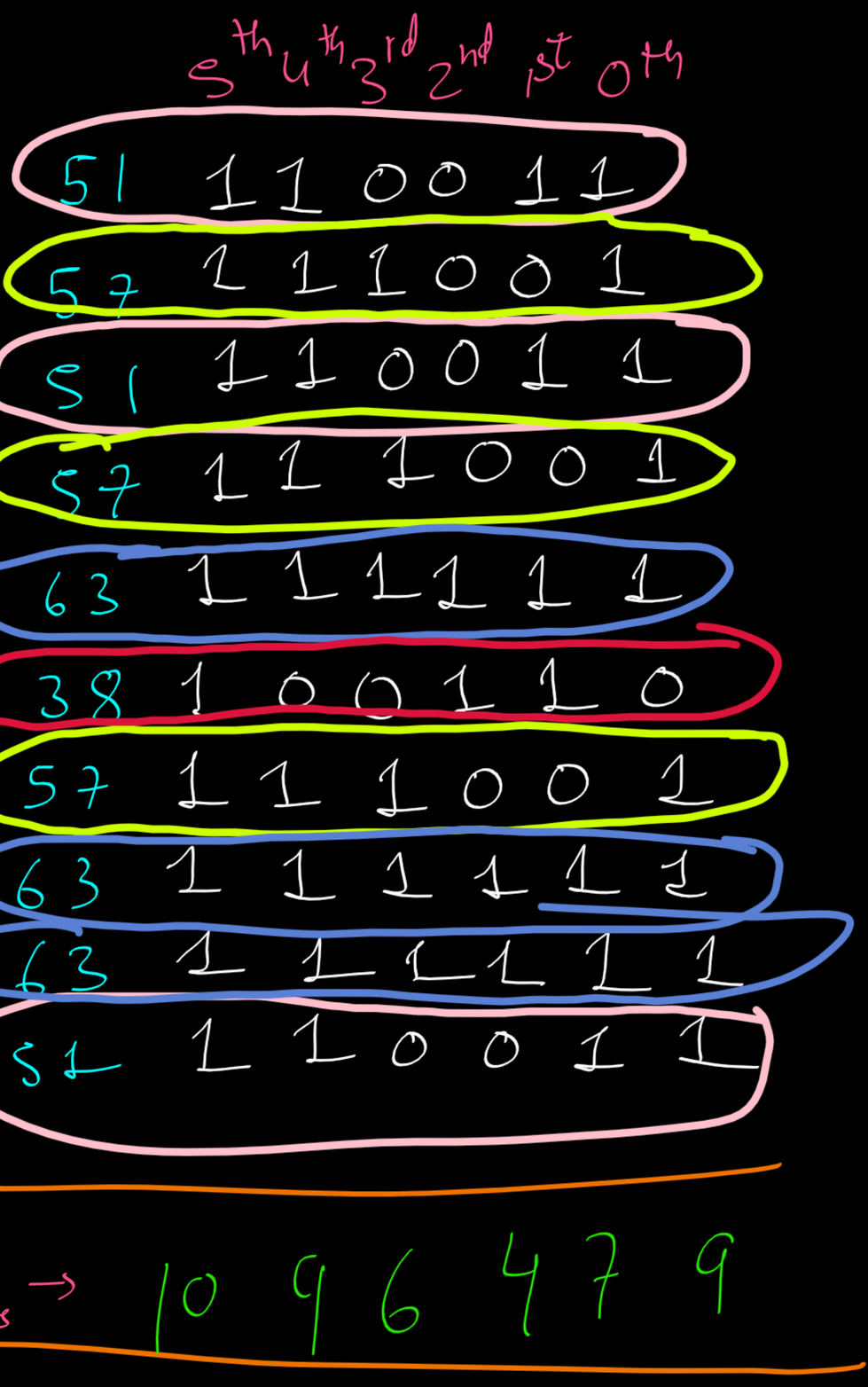
Given an integer array `nums` where every element appears **three times** except for one, which appears **exactly once**. *Find the single element and return it.*

You must implement a solution with a linear runtime complexity and use only constant extra space.

```java
class Solution {
    public int singleNumber(int[] nums) {
        int[] res = new int[32];

        for(int i=0;i<=31;i++) {

            int count1 = 0;
            for(int val : nums) {
                int mask  = (1 << i);
                int bit = (val & mask);
                if(bit != 0) count1++;
            }

            res[i] = count1;
        }

        int ans = 0;
        for(int i=0;i<32;i++) {
            if(res[i] % 3 != 0) {
                ans += (1 << i);
            }
        }

        return ans;
    }
}
```

TC :=

O(32*N)



5th 4th 3rd 2nd 1st 0th

10 - 9 - 6 - 4 - 7 - 9

(3n+1) 3n 3n (3n+1) (3n+1) 3n

where there are 3n bits, put 0
& where there are 3n+1 bits, put 1.

5th 4th 3rd 2nd 1st 0th

10 - 9 - 6 - 4 - 7 - 9

1 0 0 1 1 0 = 38

↳ This is our unique number.

This solution works because if there was no unique no & all nos repeated thrice, the set bits at all positions of all nos would have been 3n. But 3n+1 bit is contributed by one extra number.

	5	4	3	2	1	0th
51	1	1	0	0	1	1
57	1	1	1	0	0	1
51	1	1	0	0	1	1
	3	3	1	0	2	3
57	1	1	1	0	0	1
63	1	1	1	1	1	1
38	1	0	0	1	1	0
57	1	1	1	0	0	1
63	1	1	1	1	1	1
63	1	1	1	1	1	1
51	1	1	0	0	1	1

3n	3n+1	3n+2
110001	00100010	00001010

Basically, at any point of time, we have to try to maintain 3 terms 3n, 3np1 & 3np2 such that if a bit position is set 3n times, it is set in 3np1 & off in 3np2 and so is the case for 3np1 & 3np2 as well.

If we are able to do this somehow, at last the ans will be 3np1 term.

So, let us now try to achieve this is O(N) TC without using any extra space.

	5	4	3	2	1	0th
51	1	1	0	0	1	1
57	1	1	1	0	0	1
51	1	1	0	0	1	1
	3	3	1	0	2	3
57	1	1	1	0	0	1
63	1	1	1	1	1	1
38	1	0	0	1	1	0
57	1	1	1	0	0	1
63	1	1	1	1	1	1
63	1	1	1	1	1	1
51	1	1	0	0	1	1

3n	3n+1	3n+2
1 1 1 1 1 1	0 0 0 0 0 0	0 0 0 0 0 0
0 0 1 1 0 0	1 1 0 0 1 1	0 0 0 0 0 0
0 0 0 1 0 0	0 0 1 0 1 0	1 1 0 0 0 1
1 1 0 1 0 1	0 0 1 0 0 0	0 0 0 0 1 0

Initially 3n ⇒ 1 1 1 1 1 1 ⇒ We haven't checked anything. no of 1s at every place is 0. Hence 0 is 3n

3np1 ⇒ 0 0 0 0 0 0 0
3np2 ⇒ 0 0 0 0 0 0 0
} Since we haven't checked any bit so, all are 0s & 0s are all 3n not 3np1 & 3np2.

Let us take the 1st No

3n	3np1	3np2
1 1 0 0 1 1	1 1 0 0 1 1	1 1 0 0 1 1
3n 1 1 1 1 1 1	3np1 0 0 0 0 0 0	3np2 0 0 0 0 0 0
sbc3n 1 1 0 0 1 1	sbc3np1 0 0 0 0 0 0	sbc3np2 0 0 0 0 0 0

All those set bits that are common in 3n will nowbe off in 3n.
Because now they have become 3n+1. (See table)

Let us do it for second no.

3n

1 1 1 0 0 1

0 0 1 1 0 0

0 0 1 0 0 0

csb3n → Turn this bit ON for 3N+1 & off for 3N.

3n+1

1 1 1 0 0 1

1 1 0 0 1 1

1 1 0 0 0 1

csb3np2 → Turn this bits ON for 3N+2 & off for 3N+1.

3n+2

1 1 1 0 0 1

0 0 0 0 0 0

0 0 0 0 0 0

csb3np2 → No bits will be turned on for 3N. & off for 3N+2.

3n 0 0 0 1 0 0

3n+1 0 0 1 0 1 0

3n+2 1 1 0 0 0 1

New values

for 3rd value in Array (idx = 2)

51 1 1 0 0 1 1

3n 0 0 0 1 0 0

0 0 0 0 0 0

3n+1 1 1 0 0 1 1

0 0 1 0 1 0

0 0 0 0 1 0

3n+2 1 1 0 0 0 1

1 1 0 0 0 1

sbc3n → Turn ON 1s in 3np1 & off in 3n

sbc3np1 → turn ON 1s in 3np2 & off in 3np2.

sbc3np2 → turn ON 1s in 3n & off in 3np2

3n 1 1 0 1 0 1

3n+1 0 0 1 0 0 0

3n+2 0 0 0 0 1 0

New values.

How to do this? Say x is the current value.

sbc3n = (x & 3n)

sbc3np1 = (x & 3np1)

sbc3np2 = (x & 3np2)

Setting bits ON in next values.

3n = (3n & sbc3np2)

3np1 = (3np1 & sbc3np2)

3np2 = (3np2 & sbc3np2)

Turning off bits in current values.

3n = (3n & ~sbc3n)

3np1 = (3np1 & ~sbc3np1)

3np2 = (3np2 & ~sbc3np2)

```java
class Solution {
    public int singleNumber(int[] arr) {
        int tn = -1, tnp1 = 0, tnp2 = 0;

        for(int i=0;i<arr.length;i++) {
            int sbcw3n = (arr[i] & tn);
            int sbcw3np1 = (arr[i] & tnp1);
            int sbcw3np2 = (arr[i] & tnp2);


            tn = (tn & (~sbcw3n));
            tnp1 = (tnp1 | sbcw3n);

            tnp1 = (tnp1 & (~sbcw3np1));
            tnp2 = (tnp2 | sbcw3np1);

            tnp2 = (tnp2 & (~sbcw3np2));
            tn = (tn | sbcw3np2);
        }

        return tnp1;
    }
}
```

Optimized Approach

(Space Optimized)

TC = O(N)

Unique Number - III

260. Single Number III

Medium 3844 182 Add to List Share

Given an integer array `nums`, in which exactly two elements appear only once and all the other elements appear exactly twice. Find the two elements that appear only once. You can return the answer in **any order**.

You must write an algorithm that runs in linear runtime complexity and uses only constant extra space.

```java
class Solution {
    public int[] singleNumber(int[] nums) {

        int xor = 0;
        for(int val : nums) {
            xor = xor ^ val;
        }

        int rmsbm = (xor & -xor);
        int x = 0;
        int y = 0;

        for(int val : nums) {
            if((val & rmsbm) != 0) {
                x = x ^ val;
            } else {
                y = y ^ val;
            }
        }

        return new int[]{x,y};
    }
}
```

36- 1 0 0 1 0 0

50- 1 1 0 0 1 0

24- 0 1 1 0 0 0

56- 1 1 1 0 0 0

36- 1 0 0 1 0 0

24 - 0 1 1 0 0 0

42 - 1 0 1 0 1 0

50 - 1 1 0 0 1 0

0 1 0 0 1 0

36 15 0 12 4 15 0 1

36 1 2 4 1 4 2 1 5 0

= 56 1 42 =

18

0 1 0 0 1 0

↓

2 meanings for this 1

→ Bits of 2 nos whose this is a XOR are diff

→ With respect to array, odd no of elements have a set bit at this position.

One set will be of all those elements whose bit corresponding to the RMSB of XOR will be 1 & other set of those nos whose bit will be 0.

Set 1 will have 1 of the 2 nos & set 2 will have the other. Rest will be the repeating eles in both the sets. So, XOR(Set 1) is the first unique no & XOR(set 2) is the other unique no.

Calculate Square of Number W/O using *, -, / or pow()

```
square(n) = 0 if n == 0
if n is even
   square(n) = 4*square(n/2)
if n is odd
   square(n) = 4*square(floor(n/2)) + 4*floor(n/2) + 1
```

gf no is x

```
if (x%2 == 0) {
        return 4 * square(x/2);

else
   return 4* square(x/2) + 4*(x/2) + 1
```

Proof

If n is even

$n = 2 * x \longrightarrow x = n/2$

$n^2 = 4 * x^2$

$\Rightarrow n^2 = 4 * (\frac{n}{2})^2$

If n is odd

$n = 2 * x + 1$

$n^2 = (2 * x + 1)^2 = 4x^2 + 4x + 1$

```java
public static int square(int n) {

    if(n == 0) return 0;

    if(n < 0) n = -n;

    //check if the number is even i.e. divisible by 2
    //if a number is divisible by 2, its LSB is 0

    int lsb = (n & 1);
    int x = (n >> 1); //floor(n/2)
    if(lsb == 1) {
        //number is odd
        // 4*x2 + 4 * x + 1 here x = floor(n/2)
        int nby2square = square(x);
        return ((nby2square << 2) + (x << 2) + 1);
    }

    // if the number is even
    // 4 * x2

    int nby2square = square(x);
    return (nby2square << 2);

}
```

Calculate 7n/8 without using *, /, + operators.

```java
import java.io.*;
import java.util.*;

public class Main {

  public static void main(String[] args){
    Scanner scn = new Scanner(System.in);
    int n = scn.nextInt();
    //write your code here

    int en = (n << 3);
    int num = en - n;
    int res = (num >> 3);
    System.out.println(res);
  }

}
```

$$\frac{7n}{8} = \frac{(8n - n)}{8}$$

$8 * n \Leftrightarrow (n << 3)$

$n/8 \Leftrightarrow (n >> 3)$

Divide 2 Integers

29. Divide Two Integers

Medium 2702 9455 Add to List Share

Given two integers `dividend` and `divisor`, divide two integers **without** using multiplication, division, and mod operator.

The integer division should truncate toward zero, which means losing its fractional part. For example, `8.345` would be truncated to `8`, and `-2.7335` would be truncated to `-2`.

Return *the* ***quotient*** *after dividing* `dividend` *by* `divisor`.

Note: Assume we are dealing with an environment that could only store integers within the **32-bit** signed integer range: $[-2^{31}, 2^{31} - 1]$. For this problem, if the quotient is **strictly greater than** $2^{31} - 1$, then return $2^{31} - 1$, and if the quotient is **strictly less than** -2^{31}, then return -2^{31}.

```java
class Solution {
    public int divide(int dividend, int divisor) {

        if(dividend == 1 << 31 && divisor == -1) return Integer.MAX_VALUE;

        boolean sign = (dividend >= 0) == (divisor >= 0) ? true : false;
        dividend = Math.abs(dividend);
        divisor = Math.abs(divisor);

        int res = 0;
        while(dividend - divisor >= 0) {
            int count = 0;
            while(dividend - (divisor << 1 << count) >= 0) {
                count++;
            }
            res += 1 << count;
            dividend -= divisor << count;
        }

        return sign ? res : -res;

    }
}
```

dividend = 10

divisor = 3

dividend = divisor * quotient + remainder.

Naive Approach

Keep substracting divisor from dividend and calculate quotient by incrementing counter.

Q Dividend = 10
Divisor = 3
10 - 3 = 7 ✓
count = 1

② Dividend = 7
Divisor = 3
7 - 3 = 4 ✓
count = 2

③ Dividend = 4
Divisor = 3
4 - 3 = 1 ✓
count = 3

④ Dividend = 1
Divisor = 3
1 - 3 = -2 ✗

```java
class Solution {
    public int divide(int dividend, int divisor) {

        if(dividend == 1 << 31 && divisor == -1) return Integer.MAX_VALUE;

        if(divisor == 1) return dividend;

        boolean sign = (dividend >= 0) == (divisor >= 0) ? true : false;
        dividend = Math.abs(dividend);
        divisor = Math.abs(divisor);

        int res = 0;
        while(dividend - divisor >= 0) {
            res++;
            dividend -= divisor;
        }

        return sign ? res: -res;
    }
}
```

Basic Approach

Submission Detail

992 / 992 test cases passed, but took too long.

Status: **Time Limit Exceeded**

Submitted: **0 minutes ago**

=> On leetcode