Introduction to Bit manipulation

Binary Number Convention (1's and 2's complement)

Ex:- $(7)_{10} \longrightarrow (1\ 1\ 1)_2$

2	7	1
2	3	1
2	1	

$\overset{2\,1\,0}{(1\ 1\ 1)_2} \longrightarrow (\quad)_{10}$

~~8 4 2 1~~

$1\times2^0 + 1\times2^1 + 1\times2^2$

$= 1 + 2 + 4 = 7$

$(13)_{10} \longrightarrow (1\ 1\ 0\ 1)_2$

2	13	1
2	6	0
2	3	1
2	1	

$\overset{3\,2\,1\,0}{(1\ 1\ 0\ 1)_2} \longrightarrow (\quad)_{10}$

$1\times2^0 + 0\times2^1 + 1\times2^2 + 1\times2^3$

$= 1 + 0 + 4 + 8$

$= 13$

Decimal to binary Code:

```
String count2Binary(int n)
{
    res = " ";
    while(n!=1)
    { if(n%2==1) res += '1';
      else{
          res += '0';
      }
      n = n/2
    }
    res = n + res;
    reverse(res);
    return res;
```

TC → $\log_2 n$

SC → $\log_2 n$

Binary to Decimal code:

```
int count2Binary(string x)
{ int len = x.length;
  P2 = 1;
  num = 0;
  for(int i = len-1 → 0)
  { if (n[i] == '1'){
        num = num + P2;
    }
    P2 = P2 x2;
  }
  return num;
}
```

int can store 32 bit
longlong " " 64 bit

int x = 13

$$\underbrace{0 \ \cdots \ 0}_{28bit} \ \underbrace{1 \ 1 \ 0 \ 1}_{4bit}$$

1's complement

$(13) \longrightarrow (1\ 1\ 0\ 1)_2$

flip

$(0\ 0\ 1\ 0)_2$

2's complement

1. 1's complement
2. then add 1

$(13)_{10} \longrightarrow$ 1's complement $(0\ 0\ 1\ 0)_2$

0	0	1	0
0	0	0	1
(0	0	1	$1)_2$

Operator → AND, OR, XOR, SHIFT, NOT

AND operator :-

all true → true
else 1 false → false

1	1	0	1
0	1	0	0
0	1	0	0

OR operator :-

1 true → true
all false → false

1	1	0	1
0	1	0	0
1	1	0	1

XOR operator :-
no of 1's → odd → 1
→ even → 0

1	1	0	1
0	1	0	0
1	0	0	1

Right Shift :-

$13 >> 1 \quad \left(\frac{13}{2^1}\right) = 6$

1 1 0 1 → 13

0 1 1 0 → 6

$13 >> 2 \quad \left(\frac{13}{4}\right) = 3$

1 1 0 1 → 13

0 0 1 1 → 3

$13 >> 3 \quad \left(\frac{13}{8}\right) = 1$

1 1 0 1 → 13

0 0 0 1 → 1

$$\boxed{n >> k = \frac{n}{2^k}}$$

$13 \longrightarrow (1101)_2 \longrightarrow 1\times2^3 + 1\times2^2 + 0\times2^1 + 1\times2^0$

$13 >> 1 \longrightarrow (0110)_2 \longrightarrow 0\times2^3 + 1\times2^2 + 1\times2^1 + 0\times2^0$

$= 1\times2^2 + 1\times2^1 + 0\times2^0 \times 0$

NOT OPERATOR:-

$n = 13$ | $n = -13$

$\boxed{0}$ 0 1 1 0 1

sign (Positive)

0 . . . . . 0 1 1 0 1 filp

1 . . . . . . 1 0 0 1 0

+ 1 add 1

$\boxed{1}$ 1 0 0 1 1

negative → $\boxed{-13}$

longest Int:-

$\boxed{0}$ 1 1 1 1 1

sign

$= 2^{30} + 2^{29} + 2^{28} + \dots + 2^0$

$= (2^{31} - 1)$ INT_MAX

Smallest :- -2^{31}

$\boxed{1}$ 0 0 0 ------- 0

↓

$$
\begin{array}{r}
0\ 1\ 1\ 1\ \dots\dots\ 1 \\
1 \\
\hline
\boxed{1}\ 0\ \dots\dots\ 0\ 0\ 0
\end{array}
$$

negative

$$\underbrace{\hspace{6em}}_{-2^{31}\ (\text{INT\_MIN})}$$

Left shift :-

13 << 1

$$
\begin{array}{ccccccl}
4 & 3 & 2 & 1 & 0 & & \\
0 & 1 & 1 & 0 & 1 & \rightarrow 13 : & 1\times2^{3} + 1\times2^{2} + 0\times2^{1} + 1\times2^{0} \\
1 & 1 & 0 & 1 & 0 & \rightarrow 26 : & 1\times2^{4} + 1\times2^{3} + 0\times2^{2} + 1\times2^{1} + 0\times2^{0}
\end{array}
$$

num << k $\longrightarrow$ num $\times 2^{k}$

$(2^{31} - 1) << 1$ means overflow

NOT OPERATOR(~) :

1. flip
2. check -ve
   - yes → 2's comp
   - no → STOP

5)

0 . . . . . . 0 0 1 0 1 → 5

$\boxed{1}$ 1 1 0 1 0

-ve

0 ~~. . . . 0 0 1 0~~ 0 2' comp

1 — — — 1 1 0 1 0

flip

0 — — — 0 0 1 0 1

+ 1

1

$\boxed{1}$ 0 . . . . . 0 0 1 1 0 → 6

sing

(-6)

Num = -6

x = ~(-6)

0 0 0 . . . . . 0 1 1 0 (+6)

1 1 1 . . . . . 1 0 0 1

+ 1

1 1 1 . . . . . 1 0 1 0 → (-6)

$\boxed{0}$ 0 0 . . . 0 1 0 1 → 5

Pos

Swap 2 number :-

a = 5
b = 6.

a = a ^ b (5 ^ 6)
b = a ^ b [(5 ^ 6) ^ 6]
= 5
a = a ^ b [(5 ^ 6) ^ 5]
= 6

swap successfull

Check if the ith bit set or not?

Ex:- N = 13   i = 2

1) (1 << 2) = 100

	1	1	0	1
&	0	1	0	0
	0	1	0	0

→ != 0 means set

if (N & (1 << i)) != 0 ; means set
else
not set ;

Ex: N = 13   i = 1

1) (1 << 1) = 10

	3	2	1	0
	1	1	0	1
&	0	0	1	0
	0	0	0	0

== 0 not set

Other method

N = 13   i = 2

1	1	0	1	right shift $(13 >> 2)$ $(N >> i)$
0	0	1	1	AND 1
& 0	0	0	1	
0	1	0	0	!= 0 (set)

N = 13   i = 1

1	1	0	1	Shift
0	1	1	0	AND 1
0	0	0	1	
0	0	0	0	→ == 0 (not set)

if $[(N >> i) \& 1 \; != 0]$ means set
else
  not set

Set ith bit

Ex:- N = 9   i = 2

$(1 << 2) = 100$

	1	0	0	1	
	1	0	1	0	0
OR	1	1	0	1	

→ $\boxed{\text{set 2 bit}}$

Condition:-

$N | (1 << i)$

N = 13   i = 2 $(1 << 2) = 100$

$$\begin{array}{r} 1\ 1\ 0\ 1 \\ |\ 0\ 1\ 0\ 0 \\ \hline 1\ 1\ 0\ 1 \rightarrow 13 \end{array}$$

$\boxed{N \mid (1 << i)}$

Clean ith bit

N = 13   i = 2 $\sim(1 << 2) = \sim(0100)$

$\therefore \sim(0100)$

(1011)

$$\begin{array}{r} 1\ 1\ 0\ 1 \\ \&\ 1\ 0\ 1\ 1 \\ \hline 1\ 0\ 0\ 1 \end{array}$$

$\rightarrow$ clean

Condition :- $\boxed{N \,\&\, [\sim(1 << i)]}$

Toggle ith bit :-

Ex: N = 13   i = 2 $(1 << 2) = 100$

$$\begin{array}{r} 1\ 1\ 0\ 1 \\ \wedge\ 0\ 1\ 0\ 0 \\ \hline 1\ 0\ 0\ 1 \end{array}$$

$\rightarrow$ Toggle

Ex:-

N = 13   i = 1

$(1 << i) = 10$

```
  1 1 0 1
^ 0 0 1 0
---------
  1 1 1 1  -> Toggle
```

$\boxed{\text{Condition :- } N \wedge (1 << i)}$

Remove the last set bit (Rightmost):

N = 16

```
  1 0 0 0 0
  0 1 1 1 1   } 15
& ---------
  0 0 0 0 0
```

N = 13

```
  0 1 1 0 1
  0 1 1 0 0   } 12
& ---------
  0 1 1 0 0
```

N = 40

```
  1 0 1 0 0 0
  1 0 0 1 1 1   } 39
& -----------
  1 0 0 0 0 0
```

Condition :- $[N \,\&\, (N-1)]$

If num is a power of 2 or not :-

N = 16	N = 13	N = 32
1 0 0 0 0	0 1 1 0 1	1 0 0 0 0 0
& 0 1 1 1 1 ← 15	& 0 1 1 0 0 ← 12	& 0 1 1 1 1 1 ← 31
0 0 0 0 0	0 1 1 0 0	0 0 0 0 0 0

Condition [N & (N-1) == 0] → power of 2
else
not power of 2

Count the number of set bit

```
int countsetbit(int n)
{ int cnt = 0;
  while(n>1)
  { if (n%2 == 1) cnt += 1;        <- [n&(1)==1]  or cnt+= n&1
    else
      n = n/2 ;  ------>  n = n>>1
  }
  if(n == 1) cnt += 1;
  return cnt;
```

~~another approach~~

$2^n = 1 << n$

last bit of Odd number always 1
last bit of even number always 0

Ex:- 13 → 1101 } odd
11 → 1011 }

12 → 1100 } even
8 → 1000 }

Check even or odd :-

13 → 1 1 0 1
& 0 0 0 1
0 0 0 1 → Ans 1

(N & 1) = = 1 → odd

11 → 1 0 1 1
& 0 0 0 1
0 0 0 1

12 → 1 1 0 0
& 0 0 0 1
0 0 0 0 = = 0

(N & 1) = = 0 → even

8 → 1 0 0 0
0 0 0 1
0 0 0 0 = = 0

Divided by 2 :

N = 13 | N = 12
i = 2 | i = 2
Ans = 6 (n/i) | Ans = 6 (n/i)

13 ← 1 1 0 1   13 ← 1 1 0 0
6 ← 0 1 1 0   6 ← 0 1 1 0

$\boxed{\text{Condition :- } N = (N>>1)}$

Count number of set bit

```
int cnt = 0;
While (N != 0)
{
    N = (N & (N-1));
    cnt++;
}
return cnt
```

N = 13

```
  1 1 0 1
& 1 1 0 0 → 12
  -------
  1 1 0 0   cnt = 1

  1 1 0 0
& 1 0 1 1 → 11
  -------
  1 0 0 0   cnt = 2

  1 0 0 0 → 8
& 0 1 1 1 → 7
  -------
  0 0 0 0   cnt = 3
```

Minimum Bit flips to convert a number

start = 10 goal = 7

1 0 1 0 0 1 1 1

1) first XOR start and goal.

```
  1 0 1 0
^ 0 1 1 1
---------
  1 1 0 1
```

② 1 1 0 1 → check how many bit set this is the min number flips to convert a number

Code :-

```
int minBitFlip(int start, int goal)
{ int num = start ^ goal;
  int count = 0;
  while(num != 0)
  { num = num & (num-1));
    cnt++;
  }
  return cnt;
```

TC → $O(\log_2(\text{start} \wedge \text{goal}))$

SC → $O(1)$

Power Set (prints all subsets)

nums = [1 2 3]   n = 3
(indices 0 1 2)

	2	1	0	
0 →	0	0	0	→ [ ]
1 →	0	0	1	→ [1]
2 →	0	1	0	→ [2]
3 →	0	1	1	→ [1, 2]
4 →	1	0	0	→ [3]
5 →	1	0	1	→ [1, 3]
6 →	1	1	0	→ [2, 3]
7 →	1	1	1	→ [1, 2, 3]

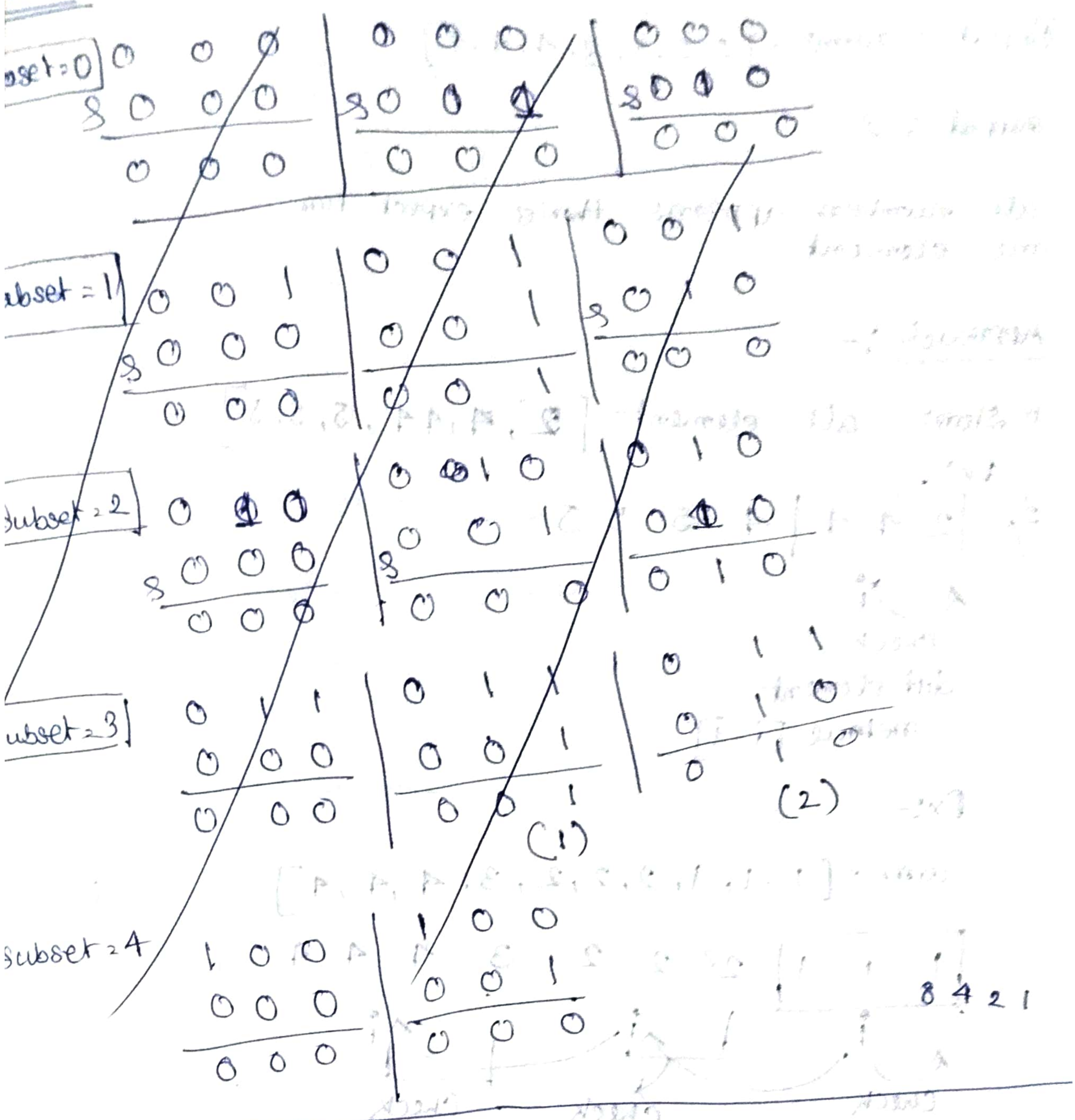
Subset

Code :-

```
Subset = (2^n) = 1 << n     arr = [ ]
for (num = 0 → subset - 1)
  {  list = [ ]
     for (i = 0 → n - 1)
       { if (num & (1 << i))
             list.add [nums[i]]
       }
     arr.add [list]
  }
return arr;
```

$2^0 = 1$
$2^1 = 2$
$2^2 = 4$

Ex:-



Example :- num = 6 (0110)

1 iteration: 0 (0000) | num & (1<<0)
(1<<0) = 0001 | = 0110 & 0001
= 0000

Ans[2, 3]

2 iteration: 1 (0001) | num & (1<<1)
(1<<1) = 0010 | = 0110 & 0010
= 0010 → 1 bit element [2]

3 iteration 2 (0010) | num & (1<<2)
(1<<2) = 0100 | = 0110 & 0100
= 0100 → 2 bit element [3]

Single number - II

Input = nums = [5, 5, 5, 2, 4, 4, 4]

output : 2

all number appears thrice expect for one element.

Approach :-

1) Sort all element [2, 4, 4 4, 5, 5, 5]

Ex: .

2) | 2 4 4 | 4 5 5 5

i

Check diff element return [i-1]

Ex:-

arr = [1, 1, 1, 2, 2, 2, 3, 4, 4, 4]

| 1 1 1 | 2 2 2 3 4 4 4

i   i   i

Check   check   Check diff element return [i-1]

দেখে নিচ্ছি পরপর দুটি element সমান কিনা
যদি দুটি সমান থাকে তবে সেই element thrice
আর আমি তাই তিনবার অন্তর i traverse হচ্ছে,

P.T.O —

Code:-

```
int findSingleNumber(vector<int>& nums)
{ sort(nums.begin(), nums.end());
    for (int i=1; i<nums.size(); i+=3)
      { if (nums[i] != nums[i-1]){
            return nums[i-1];
        }
      }

    return nums[nums.size()-1];
}
```

Single - 1

nums = [4, 1, 2, 1, 2]

which is onece Ans = 4

Approach = 4 ^ 1 ^ 2 ^ 1 ^ 2

= 4

Code :-

```
int XOR = 0
for(int i = 0 → n-1)
  { XOR = XOR ^ num[i];
  }
return XOR
```

TC → O(N)

SC → O(1)

→

~~Sigma~~ Single Number-III

Problem Statement = Given an array of integers in which exactly two elements appear only once and all ~~x~~ the other elements appear exactly twice. Return the two element the appears only once. You can return them in any order.

Approach :-

1. 1st XOR all element
2. Identify the right most bit
3. convert all numbers into two bucket according to rightmost bit
4. then XOR two bucket's all numbers and find two unique numbers.

Ex:- nums = [2, 4, 2, 14, 3, 7, 7, 3]

XOR = 2 ^ 4 ^ 2 ^ 14 ^ 3 ^ 7 ^ 7 ^ 3

= 14 ^ 4 = 1110 ^ 0100

= 1010

Identify rightmost set bit :- num ^ [num & (num-1)]

num = 1010

& num-1 = 1001

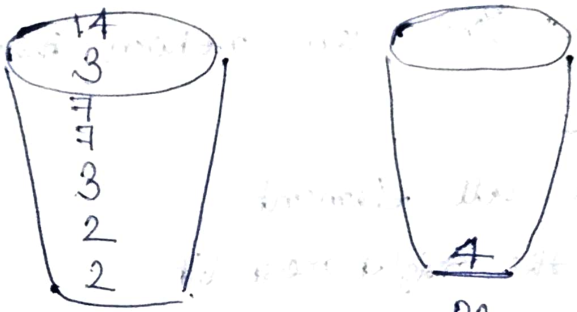
1000

^ 1010

0010 → isolate the rightmost bit.

Convert all bit number in binary and according to rightmost bit set, convert into two bucket of all number.

	8	4	2	1
2 →	0	0	1	0
4 →	0	1	0	0
2 →	0	0	1	0
14 →	1	1	1	0
3 →	0	0	1	1
7 →	0	1	1	1
7 →	0	1	1	1
3 →	0	0	1	1



B1
Store which rightmost bit is set

B2
Store which rightmost bit is not set bit

then XOR all which is in B1 and XOR which is in B2

$B1 = 2 \wedge 2 \wedge 3 \wedge 7 \wedge 7 \wedge 3 \wedge 14$

$= 14$

$B2 = 4$

Code :-

```
vector <int> singleNumber (vector <int> & nums){
    int xorr = 0;
    for(int i=0; i<nums.size(); i++){
        xorr = xorr ^ num[i];        // XOR all element
    }

    if (xorr == INT_MIN){
        rightmost = INT_MIN;
    }
    else{
        rightmost = (xorr &(xorr-1))^xorr;  // find right
                                              most bit
    }
    int bucket1 = 0;
    int bucket2 = 0;

    for (int i=0 ; i< nums.size(); i++)
        { if (num[i] & rightmost){
              bucket1 = nums[i] ^bucket1;   // divides into
                                              two bucket
          }
          else {
              bucket2 = num[i] ^ bucket2;
          }
    }

    return {bucket1, bucket2};
```

XOR of numbers in a given range

			Ans
N = 1	→ 1	(N%4 == 1)	1
N = 2	→ 3	(N%4 == 2)	N+1
N = 3	→ 0	(N%4 == 3)	0
N = 4	→ 4	(N%4 == 0)	N

N = 5 → 1
N = 6 → 7
N = 7 → 0
N = 8 → 8

N = 9 → 1

Approach :-

& start point = 3
end point = 5

XORtillN (3-1) = 1^2
XORtillN(5) = 1^2^3^4^5

Ans = XORtillN(5) ^ XORtillN(3-1)
= 1^2^3^4^5^1^2
= 3^4^5 = 2

Code :-

```
int XORTillN(int n){
    if (n%4 == 1) return 1;
    if (n%4 == 2) return n+1;
    if (n%4 == 3) return 0;
    return n;
}

int findRangeXOR(int l, int r){
    return XORTillN(l-1) ^ XORTillN(r);
}
```

Divide 2 numbers without multiplication and division

Dry Run :- dividend = 10
divisor = 3

n = 10
d = 3
ans = 0

1) $10 >= 3$ → (i) cnt = 0

$10 >= (3<<1)$

$\Rightarrow 10 >= 3\times2^1$

$10 >= 6$ ✓

$\boxed{cnt = 1}$

(ii) $10 \not>= (3<<2)$

$10 \not>= 3\times2^2$

$10 \not>= 3\times4 = 12$

$ans = 0 + (1<<1)$
$= 0 + 2^1 = 2$

$n = n - d*(1<<1)$
$= 10 - 3*2$
$= 10 - 6 = 4$

$4 >= 3$ $cnt = 0$

$4 >= (3 << 1)$

$= 3 \times 2$

$4 \cancel{>}= = 6$

$cont = 0$

$ans = 2 + (1 << 0)$

$= 2 + 2^0$

$= 2 + 1 = 3$

$n = n - d * (1 << 0)$

$= 4 - 3 * 1$

$= 1$

$ans = 3$

Code :-

```
int divide (int dividend , int divison) {
  if( dividend == divison) return 1;
  if (dividend == INT_MIN && divison == -1)
      return INT_MAX;
  if (divison == 1) return dividend;

  bool ispositive = true;
  if (dividend >= 0 && divison < 0)
      ispositive = false;
  else if (dividend <= 0 && divison > 0)
      ispositive = false;
  else
      ispositive = true;
```

```
long n = dividend;
~~tog~~
long d = divisor;
  n = abs(n);
  d = abs(d);
  int ans = 0;

  while (n >= d){
        int count = 0;
     while (n >= (d << (count +1)))
        { count++;
        }
        ans += (1 << count);
        n -= d * (1 << count);
   }  // overflow
  if (ans == (1 << 31) && ispositive)
     return INT_MAX;
  if (ans == (1 << 31) && !positive)
     return INT_MIN;

  return ispositive ? ans : -1 * ans;

  }
};
```