

Bit Manipulation

Number System

- ① Decimal : Base 10 {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}
- ② Binary : Base 2 {0, 1, 10, 11, 100, 101, 110, 111, 1000}
- ③ Octal : Base 8 {0, 1, 2, 3, 4, 5, 6, 7, 10, 11, ..., 17, 20, ..., 2⁷, ...}
- ④ Hexadecimal : Base 16 {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F}

$$n = (60)_{10}$$

$$\text{base} = (?)_2$$

Decimal to Any Base

2	60	
2	30	0×10^0
2	15	0×10^1
2	7	1×10^2
2	3	1×10^3
2	1	1×10^4
0		1×10^5

Trick
destination base (d)
source base (s)

\Rightarrow Division & modulus
by d

\Rightarrow multiplier by s

$$\begin{aligned} &= 0 \times 1^0 + 0 \times 10^1 \\ &\quad + 1 \times 10^2 + 1 \times 10^3 \\ &\quad + 1 \times 10^4 + 1 \times 10^5 \end{aligned}$$

$$= (11100)_2$$

8	$(355)_{10}$	$48('0')$
8	44	$3 * 10^0 \rightarrow '3'$
8	5	$4 * 10^1 \rightarrow '4'$
0		$5 * 10^2 \rightarrow '5'$
		$= (543)_8$

destination base = 8 (octal)
given base = 10 (decimal)

16	$(743)_{10}$	
16	46	$7 \rightarrow '7'$
16	2	$14 \rightarrow 'E'$
0		$2 \rightarrow '2'$

given base = 10 (decimal)
required base = 16
(hexadecimal)

```

class Solution{
    static char get(int n){
        if(n < 10) return (char)('0' + n);
        if(n == 10) return 'A';
        if(n == 11) return 'B';
        if(n == 12) return 'C';
        if(n == 13) return 'D';
        if(n == 14) return 'E';
        return 'F';
    }

    static String getNumber(int B, int N){
        StringBuilder res = new StringBuilder();
        while(N != 0){
            char remainder = get(N % B);
            res.append(remainder);
            N = N / B;
        }
        return res.reverse().toString();
    }
}

```

logarithmic
 $O(\log_B N)$

$\approx O(1)$

Space $\approx O(1)$
 Constant extra space

Sum of digits in Base K

```
class Solution {  
    public int sumBase(int n, int b) {  
        int sum = 0;  
  
        while(n != 0){  
            sum += (n % b);  
            n = n / b;  
        }  
  
        return sum;  
    }  
}
```

Time $\Rightarrow O(\log_B n)$

Space $\Rightarrow O(1)$

Leetcode 1837

Base 7 (S04)

```
class Solution {
    public String convertToBase7(int N) {
        long res = 0, power = 1;

        while(N != 0){
            int remainder = N % 7;
            res = res + power * remainder;
            N = N / 7;
            power = power * 10;
        }

        return String.valueOf(res);
    }
}
```

~~Time~~ $O(\log_7 N)$

~~Space~~ $O(1)$

$$\begin{array}{r} (-239)_{10} \\ \hline 7 | -34 & -1 \times 10^0 \\ \hline 7 | -4 & -6 \times 10^1 \\ \hline 0 & -4 \times 10^2 \\ \hline -1 + (-60) \\ + (-400) \\ \hline = -\underline{\underline{461}} \end{array}$$

AND ($\Sigma \wedge \Sigma$)

$$0 \wedge 0 = 0$$

$$0 \wedge 1 = 0$$

$$1 \wedge 0 = 0$$

$$1 \wedge 1 = 1$$

binary operators
↓
OR ($\Sigma \Sigma$)

$$0 \mid 0 = 0$$

$$1 \mid 0 = 1$$

$$0 \mid 1 = 1$$

$$1 \mid 1 = 1$$

XOR $\begin{cases} \text{if diff bit} \\ \Rightarrow \text{res=1} \end{cases}$
 \Downarrow Same bits \Rightarrow res=0

$$0 \wedge 0 = 0$$

$$0 \wedge 1 = 1$$

$$1 \wedge 0 = 1$$

$$1 \wedge 1 = 0$$

NOT {Tilde}
Unary

$$\sim 0 = 1$$

$$\sim 1 = 0$$

logical \rightarrow boolean
operators $\{\&, \mid, \text{!}\}$

bitwise \rightarrow bits
operators $\{\wedge, \mid, \sim\}$

Convert Any Base to Decimal

10	$(10.01101)_2$	power $\rightarrow 2$
10	100110	division/modulus $\rightarrow 10$
10	10011	$1 * 2^0 = 1$
10	1001	$0 * 2^1$
10	100	$1 * 2^2 = 4$
10	100	$1 * 2^3 = 8$
10	10	$0 * 2^4$
10	1	$0 * 2^5$
10	0	$1 * 2^6 = 64$

"30F4A"

$(200522)_{10}$

'A' $\longrightarrow 10 \times 16^0 = 10$

'4' $\longrightarrow 4 \times 16^1 = 64$

'F' $\longrightarrow 15 \times 16^2 = 3840$

'0' $\longrightarrow 0 \times 16^3$

'3' $\longrightarrow 3 \times 16^4 = 196608$

65 'A' → 10 (0+10)

66 'B' → 11 (1+10)

67 'C' → 12 (2+10)

68 'D' → 13 (3+10)

69 'E' → 14 (4+10)

70 'F' → 15 (5+10)

0⁴⁸ '0' $\xrightarrow{-'0'} 0$

1⁴⁸ '1' $\xrightarrow{-'0'} 1$

2⁴⁸ '2' $\xrightarrow{-'0'} 2$

3⁴⁸ '3' $\xrightarrow{-'0'} 3$

8⁴⁸ '8' $\xrightarrow{-'0'} 8$

9⁴⁸ '9' $\xrightarrow{-'0'} 9$

```
static int get(char val){  
    if(val >= '0' && val <= '9')  
        return val - '0';  
    return val - 'A' + 10;  
}
```

```
static int decimalEquivalent(String N, int b) {  
    int res = 0, power = 1;  
  
    for(int idx = N.length() - 1; idx >= 0; idx--){  
        char ch = N.charAt(idx);  
        int val = get(ch);  
        if(val >= b) return -1; // invalid input  
  
        res = res + power * val;  
        power = power * b;  
    }  
  
    return res;  
}
```

Bit Manipulation

byte
↓
size 1 byte
= 8 bits

range -2^7 to
 $2^7 - 1$

short
↓
2 byte
= 16 bits

-2^{15} to
 $2^{15} - 1$

int
↓
4 byte
= 32 bits

-2^{31} to
 $2^{31} - 1$

long
↓
8 byte
= 64 bits

-2^{63} to
 $2^{63} - 1$

4 bit number system

$$0000 \rightarrow 0$$

$$0001 \rightarrow 1$$

$$0010 \rightarrow 2$$

$$0011 \rightarrow 3$$

$$0100 \rightarrow 4$$

$$0101 \rightarrow 5$$

$$0110 \rightarrow 6$$

$$0111 \rightarrow 7$$

$$1000 \rightarrow 8$$

$$1001 \rightarrow 9$$

$$1010 \rightarrow 10$$

$$1011 \rightarrow 11$$

$$1100 \rightarrow 12$$

$$1101 \rightarrow 13$$

$$1110 \rightarrow 14$$

$$1111 \rightarrow 15$$

unsigned
number system (ASCII)

Approach 1

$$\Rightarrow -2^3 \text{ to } +2^3 - 1$$

$$-8 \quad \text{to} \quad +7$$

$$\begin{array}{l} 0000 \rightarrow 0 \\ 0001 \rightarrow 1 \\ 0010 \rightarrow 2 \\ 0011 \rightarrow 3 \\ 0100 \rightarrow 4 \\ 0101 \rightarrow 5 \\ 0110 \rightarrow 6 \\ 0111 \rightarrow 7 \end{array} \quad \begin{array}{l} \downarrow \\ \downarrow \end{array} \quad \begin{array}{l} \times 1 \\ \times 1 \end{array}$$

MSB

$$\begin{array}{l} 1000 \rightarrow -0 \\ 1001 \rightarrow -1 \\ 1010 \rightarrow -2 \\ 1011 \rightarrow -3 \\ 1100 \rightarrow -4 \\ 1101 \rightarrow -5 \\ 1110 \rightarrow -6 \\ 1111 \rightarrow -7 \end{array} \quad \begin{array}{l} \downarrow \\ \downarrow \end{array} \quad \begin{array}{l} \times -1 \\ \times -2 \\ \times -3 \\ \times -4 \\ \times -5 \\ \times -6 \\ \times -7 \end{array}$$

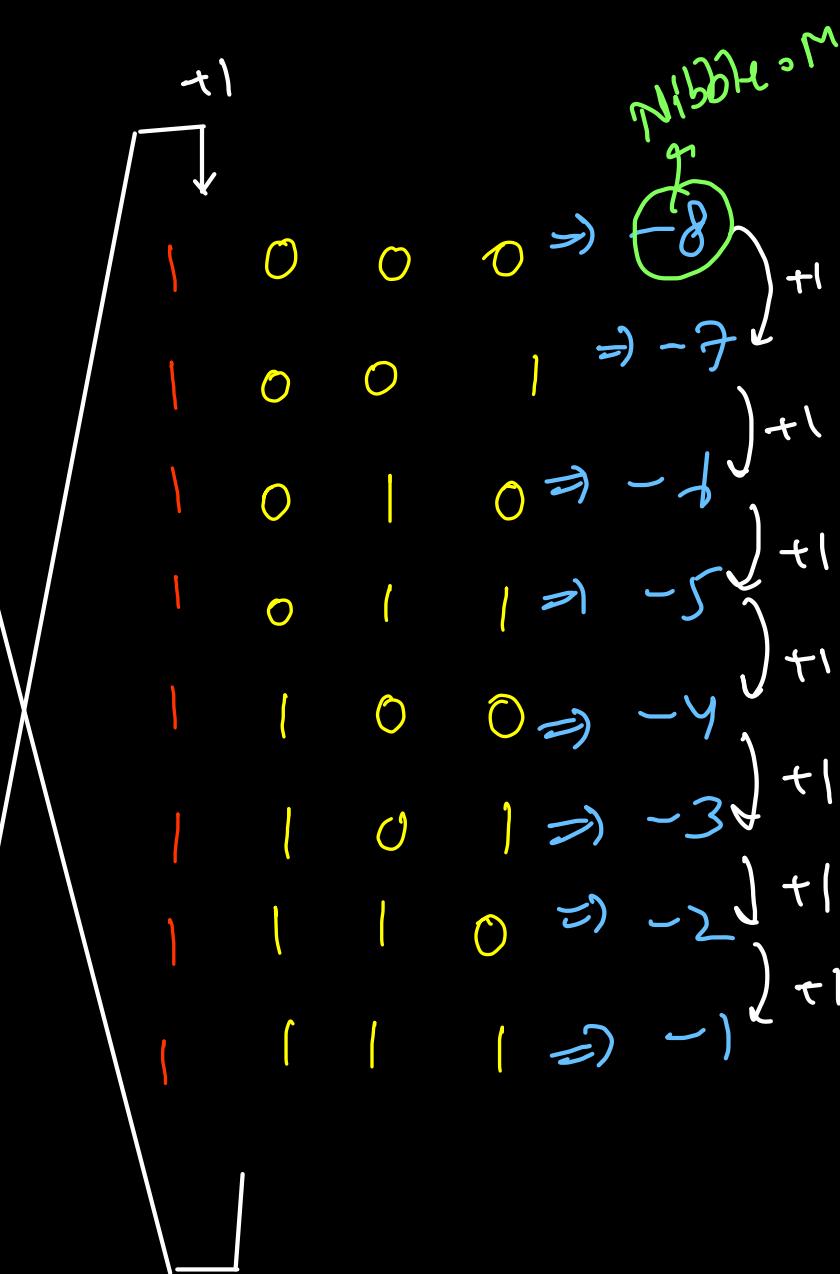
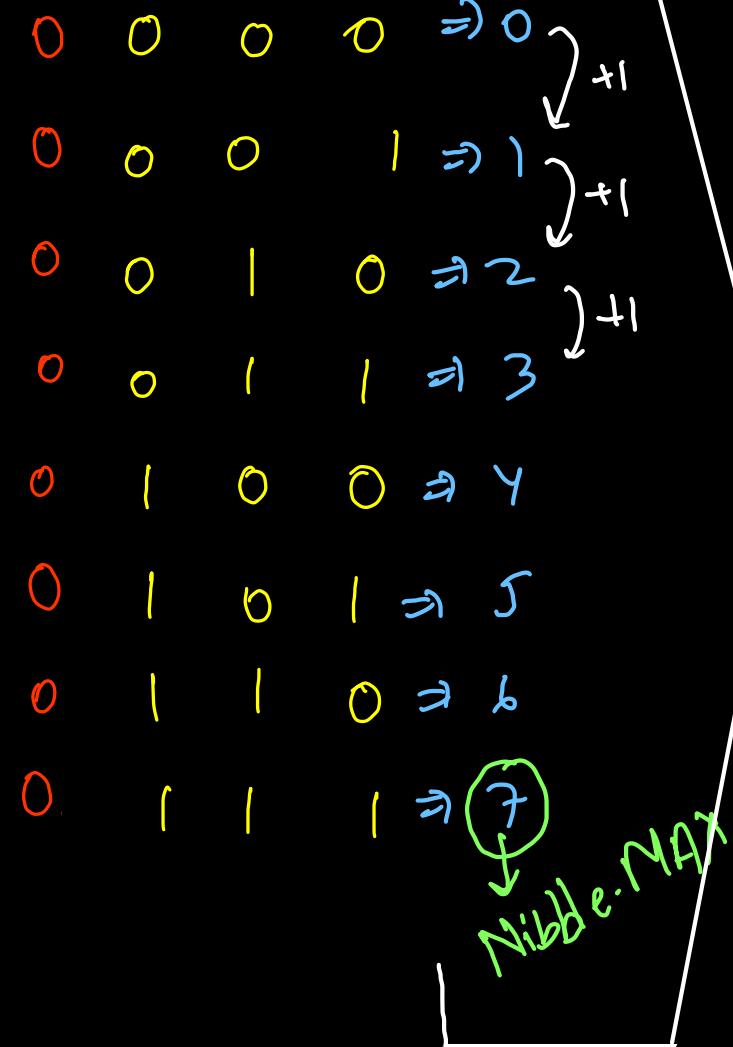
MSB

wrong

number system

Approach 2

Approach ③



Nibble.MIN
MSB (leftmost bit)
= 0
 \Rightarrow Number is +ve
 \Rightarrow directly convert
MSB (leftmost bit)
= 1
 \Rightarrow Number is -ve
decimal
= 2's complement
+
Convert into decimal

x
 $mSB = -1 \Rightarrow -ve$

| 0 0 0

| 0 0 1

| 0 1 0

| 0 1 1

| 1 0 0

| 1 0 1

| 1 1 0

| 1 1 1

Nx (toggle bits)
To complement

0 1 1 1

0 1 1 0

0 1 0 1

0 1 0 0

0 0 1 1

0 0 1 0

0 0 0 1

0 0 0 0

\Rightarrow

1 0 0 0 $\Rightarrow -8$

0 1 1 1 $\Rightarrow -7$

0 1 1 0 $\Rightarrow -6$

0 1 0 1 $\Rightarrow -5$

0 1 0 0 $\Rightarrow -4$

0 0 1 1 $\Rightarrow -3$

0 0 1 0 $\Rightarrow -2$

0 0 0 1 $\Rightarrow -1$

(add 1)
2's complement
 $Nx + 1$
 $= -x$

$$\left. \begin{array}{l}
 \text{Overflow} \\
 \text{Integer.MAX-VALUE} + 1 = -\infty \\
 \text{Integer.MIN-VALUE} - 1 = \text{Integer.MAX-VALUE} \\
 -\infty - 1 = +\infty
 \end{array} \right\}$$

Number System is circular

Decimal to Binary (4 bit No System)

Decimal \rightarrow +ve

$$0 \ 0 \ 0 \ 0 \Rightarrow 0$$

$$0 \ 0 \ 0 \ 1 \Rightarrow 1$$

$$0 \ 0 \ 1 \ 0 \Rightarrow 2$$

$$0 \ 0 \ 1 \ 1 \Rightarrow 3$$

$$0 \ 1 \ 0 \ 0 \Rightarrow 4$$

$$0 \ 1 \ 0 \ 1 \Rightarrow 5$$

$$0 \ 1 \ 1 \ 0 \Rightarrow 6$$

$$0 \ 1 \ 1 \ 1 \Rightarrow 7$$

$$1 \ 0 \ 0 \ 0 \Rightarrow -8$$

$$1 \ 0 \ 0 \ 1 \Rightarrow -7$$

$$1 \ 0 \ 1 \ 0 \Rightarrow -6$$

$$1 \ 0 \ 1 \ 1 \Rightarrow -5$$

$$1 \ 1 \ 0 \ 0 \Rightarrow -4$$

$$1 \ 1 \ 0 \ 1 \Rightarrow -3$$

$$1 \ 1 \ 1 \ 0 \Rightarrow -2$$

$$1 \ 1 \ 1 \ 1 \Rightarrow -1$$

① Convert to Binary

② Fit into Bits.

$$\text{nibble } aN=2^0 \\ (20)_{10} = (?)_2$$

$$(4)_{10} = (?)_2$$

$$0100$$

$$\text{X}0101 \\ \text{sys}0(\text{ans}) \Rightarrow 5$$

$$(1)_{10} = (?)_2$$

$$(15)_{10} = (?)_2$$

$$0001$$

leading zeros

$$| | | | \\ \Rightarrow -1$$

drop leading
bit

Decimal \Rightarrow -ve

Binary \Rightarrow ?

① Drop -ve sign

② Convert into binary (4 bits)

③ Find its' 2's complement

-1
toggle $\overset{0}{\underset{\text{same}}{\equiv}}$ toggle
+1

0 1 1 1
 \downarrow toggle
1 0 0 0
 \downarrow +1
1 0 0 1
 \equiv

Bitwise operators

① AND

$$0 \& 0 = 0$$

$$0 \& 1 = 0$$

$$1 \& 0 = 0$$

$$1 \& 1 = 1$$

$$5 \& 6 = ④$$

$$\begin{array}{r} 0101 \\ + 0110 \\ \hline 0100 \end{array}$$

② OR

$$0 | 0 = 0$$

$$1 | 0 = 1$$

$$0 | 1 = 1$$

$$1 | 1 = 1$$

$$5 | 6 = ⑦$$

$$\begin{array}{r} 0101 \\ + 0110 \\ \hline 0111 \end{array}$$

③ XOR

$$0 \wedge 0 = 0$$

$$0 \wedge 1 = 1$$

$$1 \wedge 0 = 1$$

$$1 \wedge 1 = 0$$

$$5 \wedge 6 = ③$$

$$\begin{array}{r} 0101 \\ + 0110 \\ \hline 0011 \end{array}$$

④ NOT

$$\sim 0 = 1$$

$$\sim 1 = 0$$

$$\sim 5$$

$$\begin{array}{r} 2^3 \text{ comp} \\ 1 \\ 0101 \\ + 1 \\ \hline 0110 \end{array}$$

$$\begin{array}{r} 0101 \\ + 1010 \\ \hline 1010 \end{array}$$

4bit system: -6

```
int a = 5; // 0101
int b = 6; // 0110

System.out.println(a & b); // 4 = 0100
System.out.println(a | b); // 7 = 0111
System.out.println(a ^ b); // 3 = 0011

System.out.println(~a); // ! 0101 = 1010 = -0110 = -6
System.out.println(~b); // ! 0110 = 1001 = -0111 = -7
```

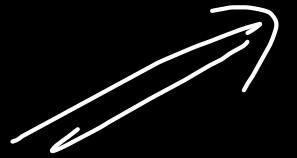
⑤ \ll
left shift operator

$$5 \ll 1 == 5 * 2^1$$

$$0101 \ll 1 = 1010
= +10$$

$$5 \ll 2 = 5 * 2^2$$

$$00000101 \ll 2 = 00010100
= +20$$



$$\cancel{0}101 \ll 1 = 10\cancel{10}$$

Binary representation of 5: 0101

Binary representation of 10: 1010

Shifted bits: 2³, 2², 2¹, 2⁰
Values: 4, 2, 1, 0



$$\cancel{0}00101 \ll 2 = 010100$$

Binary representation of 20: 00010100

Binary representation of 40: 010100

Shifted bits: 2⁴, 2³, 2²
Values: 16, 8, 4

$$a \ll \frac{n}{\cancel{J}} = a * 2^{\cancel{n}}$$

$O(\log n)$

⑥ $>>$ (signed)
right shift operators

0 1 01 $>> 1$

0 010

$$5 >> 1 = 2$$

0 0 1 1 0 $>> 2$

$$26 >> 2 = 6$$

$$a >> n = a / 2^n$$

8 bits
1 0 1 1 0 0 1 0 $>> 3$
empty spaces by 1
1 1 1 1 0 1 1 0

-78
div by 2^3
-10

$$\frac{-78}{2^3} = \frac{-78}{8} = -10$$

$$\left\{ \left[\frac{-78}{8} \right] = \left[\frac{-39}{4} \right] = \left[\frac{-20}{2} \right] = -10 \right\}$$

⑦ Unsigned right shift ($>>$) → Empty spaces will be always 0's

~~Eg 1~~ $0101 >> 1 = 5/2 = 2$

~~Eg 2~~ $\underbrace{10110010}_{\text{ }} >> 3$

$\underline{0} \underline{0} \underline{0} \underline{1} 0110 \Rightarrow 22$

Integer Class Bitwise functions

`bitCount(int i)`

number of set bits

$O(32)$

Returns the number of one-bits in the two's complement binary representation of the specified int value.

`highestOneBit(int i)`

leftmost set bit (LSB)

$O(32)$

Returns an int value with at most a single one-bit, in the position of the highest-order ("leftmost") one-bit in the specified int value.

`lowestOneBit(int i)`

rightmost set bit (RSB)

$O(32)$

Returns an int value with at most a single one-bit, in the position of the lowest-order ("rightmost") one-bit in the specified int value.

64 5 4 3 2 1 0
32 16 8 4 2 1
? 0 1 1 0 0 0 ⇒ 88
•

`System.out.println(Integer.toBinaryString(88));`

1011000

`System.out.println(Integer.bitCount(88));`

3

`System.out.println(Integer.highestOneBit(88));`

64 place value

`System.out.println(Integer.lowestOneBit(88));`

8 place value

toBinaryString(int i)

Returns a string representation of the integer argument as an unsigned integer in base 2.

toHexString(int i)

Returns a string representation of the integer argument as an unsigned integer in base 16.

toOctalString(int i)

Returns a string representation of the integer argument as an unsigned integer in base 8.

toString()

Returns a `String` object representing this `Integer`'s value.

2	$(88)_{10}$	
2	44	0
2	22	0
2	11	0
2	5	1
2	2	1
2	1	0
0	1	

$\begin{array}{r} 1011000 \\ \times 0000001 \\ \hline 1011000 \end{array} \quad \& 1 = 0$

$\begin{array}{r} 101100 \\ \times 00001 \\ \hline 101100 \end{array} \quad \& 1 = 0$

$\begin{array}{r} 10110 \\ \times 0001 \\ \hline 1011 \end{array} \quad \& 1 = 1$

101 101
 $\cancel{10}$ $\cancel{1001}$
 1 $\cancel{001}$
 0 110
 $\cancel{1}$ $\cancel{000}$

odd $5 \% . 2 = 1$ famous
 even odd $6 \% . 2 = 0$

odd $101 \% . 2 = 1$ even $110 \% . 2 = 0$

$a \% . 2 = a \& 1$

$a / 2 = a \gg 1$

```
static String getBinaryRep(int N){
    // given base = decimal = 10
    // required base = binary = 2

    StringBuilder res = new StringBuilder();
    for(int i = 0; i < 30; i++){
        res.append(N & 1);  $\Rightarrow N \% . 2$ 
        N = N >> 1;  $\Rightarrow N = n / 2$ 
    }

    return res.reverse().toString();
}
```

$$a \% 2$$

$$6 \% 2 == 0 \quad (\text{even})$$

$$5 \% 2 == 1 \quad (\text{odd})$$

$$\begin{array}{r} 2=4 \\ 1 \\ 1 \\ 1 \\ 0 \\ \hline 110 \end{array} = 1 \times 2^2 + 1 \times 2^1 = 4 + 2 = 6$$

$$\begin{array}{r} 1=4 \\ 0 \\ 1 \\ \hline 101 \end{array} = 1 \times 2^2 + 1 \times 2^0 = 4 + 1 = 5$$

if last bit is set, number is odd

else number is even

$$\begin{array}{r} \text{even} \\ 110 \\ \hline \underline{2\ 001} \\ 000 \\ \hline \end{array} \quad \begin{array}{r} \text{odd} \\ 101 \\ \hline \underline{2\ 001} \\ 001 \\ \hline \end{array}$$

$$a \% 2 == a \& 1$$

Bitmasking

$$(88)_{10} = (10110000)_2$$

6 5 4 3 2 1 0

Q1) i th bit \Rightarrow check $\xrightarrow{\text{set}(1)} \text{unset}(0)$

3rd bit $\rightarrow \checkmark$
2nd bit $\rightarrow \times$

Q2) i th bit \Rightarrow set

$$\begin{matrix} 1 & \rightarrow & 1 \\ 0 & \rightarrow & 1 \end{matrix}$$

3rd bit $\rightarrow 101\dot{1}000$
2nd bit $\rightarrow 10111.00$

Q3) i th bit \Rightarrow unset

$$\begin{matrix} 0 & \rightarrow & 0 \\ 1 & \rightarrow & 0 \end{matrix}$$

3rd bit $\rightarrow 1010000$
2nd bit $\rightarrow 10110.00$

Q4) i th bit \Rightarrow toggle

$$\begin{matrix} 0 & \rightarrow & 1 \\ 1 & \rightarrow & 0 \end{matrix}$$

3rd bit $\rightarrow 1010000$
2nd bit $\rightarrow 1011100$

AND \rightarrow powerful bit $\rightarrow 0$

$$0 \otimes 0 = 0$$

$$1 \otimes 0 = 0$$

$$? \otimes 0 = 0$$

unset

XOR

same bits $\rightarrow 0$

diff bits $\rightarrow 1$

toggle

OR \rightarrow powerful bit $\rightarrow 1$

$$0 \mid 1 = 1$$

$$1 \mid 1 = 1$$

$$? \mid 1 = 1$$

set

Q1) i th bit \Rightarrow check $\xrightarrow{\text{set}(1)}$
 $\xrightarrow{\text{unset}(0)}$

$88, 3^{\text{rd}}$
num & $1 \ll i = 0$
 $88, 2^{\text{nd}}$
num & $1 \ll i = 0$

6 5 4 3 2 1 0
num = 1011000
mask = 0001000
and = 0001000 $\neq 0$

num = 1011000
mask = 0000100
and = 0000000 = 0

1 = $2^0 = 1 \ll 0$
10 = $2^1 = 1 \ll 1$
100 = $2^2 = 1 \ll 2$
1000 = $2^3 = 1 \ll 3$

mask (only 3rd bit is set)

$$= 1 \ll 3$$

i th bit is set mask
 $= 1 \ll i$

```
int mask = 1 << k;  
if((n & mask) == 0) return false;  
return true;
```

check i th bit using OR

10 11000

or 11 10111

$$\begin{array}{r} \\ \hline \sim(1111111) = 0 \end{array}$$

10 10000

or 11 10111

$$\begin{array}{r} \\ \hline \sim(1110111) \neq 0 \end{array}$$

Q2) i th bit \Rightarrow set

$$\begin{matrix} 1 \rightarrow 1 \\ 0 \rightarrow 1 \end{matrix}$$

ans = $(1 \ll i)$ num

1 0 1 1 0 0 0
^{3rd}

1 0 1 0 0 0 0
^{3rd}

mask 0 0 0 1 0 0 0

or 1 0 1 1 0 0 0

0 0 0 1 0 0 0

1 0 1 1 0 0 0

```
static int setKthBit(int N, int K){  
    return (N | (1 << K));  
}
```

Q3) i th bit \Rightarrow unset

$$\begin{array}{l} 0 \rightarrow 0 \\ 1 \rightarrow 0 \end{array}$$

$$\text{mask} = \sim(1 \ll i);$$

return $N \& \text{mask};$

10 11 0 0 0

mask 1 1 1 0 1 1 1

and 10 1 0 0 0 0

10 1 0 0 0 0

mask 1 1 1 0 1 1 1

and 10 1 0 0 0 0

```
public static int resetAllBitsInRange(int N, int L, int R){  
    for(int i = R - 1; i < L; i++){  
        int mask = ~(1 << i);  
        N = N & mask;  
    }  
  
    return N;  
}
```

→ unset all bits from
R to L
(1-based indexing)

Q4) i^{th} bit \Rightarrow toggle

$0 \rightarrow 1$
 $1 \rightarrow 0$

mask = $(1 << i);$
return $N \wedge \text{mask};$

Eg¹ 1011000
mask 0001000
↓ same

Xor 1010000

Eg² 1010000
mask 0001000
↓ diff

Xor 1011000

```
static int toggleBits(int N , int L , int R) {  
    for(int i = L - 1; i <= R - 1; i++){  
        int mask = (1 << i);  
        N = N ^ mask;  
    }  
    return N;  
}
```

→ toggling All bits from L to R.-
(l-based indexing)

Tip #1 Check odd/even using bitwise operators

(1)

```
class Solution{
    static String oddEven(int N){
        if((N & 1) == 0) return "even";
        else return "odd";
    }
}
```

Tip #2 check if number is power of 2

input = 32 ✓

input = 1 ✓

input = 20 ✗

1 10 100 1000 10000 100000

```

class Solution {
    public boolean isPowerOfTwo(int n) {
        if(n <= 0) return false;
        return (Integer.bitCount(n) == 1);
    }
}.

```

$O(32)$ solution

```

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

```

$O(1)$

$$\begin{matrix} 0 & 0 & 0 & 0 \\ \uparrow & & & \\ 2^0 & 0 & 0 & 0 & 1 \\ \uparrow & & & & \\ 2^1 & 0 & 0 & 1 & 0 \end{matrix}$$

$$\begin{matrix} 0 & 0 & 1 & 1 \\ \uparrow & & & \\ 2^2 & 0 & 1 & 0 & 0 \\ \uparrow & & & \\ 0 & 1 & 0 & 1 \end{matrix}$$

$$0 \ 1 \ 1 \ 0$$

$$0 \ 1 \ 1 \ 1$$

$$2^3 \quad 1000$$

$$1001$$

$$1010$$

$$1011$$

$$1100$$

$$1101$$

$$1110$$

$$1111$$

$$100000$$

$$\begin{matrix} n-1 & 0 & 1 & 1 & 1 & n & 0 & 1 & 1 & 1 \\ \downarrow & n & 1 & 0 & 0 & \downarrow & n & 1 & 0 & 0 & 0 \\ \hline & & & & & & & & & 0 \end{matrix}$$

$$\begin{matrix} n-1 & 0 & 1 & 1 & 1 & n & 0 & 1 & 1 & 1 \\ \downarrow & n & 1 & 0 & 0 & \downarrow & n & 1 & 0 & 0 & 0 \\ \hline & & & & & & & & & 0 \end{matrix}$$

$$n-1 \quad 011111$$

$$\begin{matrix} n-1 & 0 & 1 & 1 & 1 & 1 & n & 0 & 0 & 0 & 0 \\ \downarrow & n & 1 & 0 & 0 & 0 & \downarrow & n & 1 & 0 & 0 & 0 \\ \hline & & & & & & & & & 0 \end{matrix}$$

$$n-1 \quad 011111$$

Tip 3: Swap 2 No's w/o using temporary variable
and arithmetic operators

$$a = 5$$

$$0101$$

$$b = 6$$

$$0110$$

```
a = a ^ b;  
b = a ^ b;  
a = a ^ b;
```

$$\textcircled{1} \quad a \wedge b = 0101 \wedge 0110 = 0011 = \textcircled{3}$$

$$a = a \wedge b$$

$$\boxed{a=3}$$

$$\textcircled{2} \quad b = a \wedge b = 3 \wedge 6 = 0011 \wedge 0110 = 0101 = \textcircled{5}$$

$$\boxed{b=5}$$

$$\begin{aligned} \textcircled{3} \quad a &= a \wedge b = 3 \wedge 5 \\ &= 0011 \wedge 0101 = 0110 = \textcircled{6} \end{aligned}$$

$$\boxed{a=6}$$

Tip #4 XOR all numbers from 1 to N.

0	1
1	$1 \wedge 2$
11	$1 \wedge 2 \wedge 3$
0	$1 \wedge 2 \wedge 3 \wedge 4$
100	$\dots \wedge n$
1	
111	
0	
1000	
1	
1011	
0	
1100	
1	
1111	
0	
10000	
1	
10011	
0	

$$1 \wedge \dots \wedge 19 = 19 \wedge 18 \wedge 17$$

$$1 \wedge 2 \wedge 3 \wedge 4 = 0$$

$$1 \wedge 2 \wedge 3 \wedge 4 \wedge 5 \wedge 6 \wedge 7 \wedge 8 = 0$$

$$1 \wedge 2 \wedge \dots \text{ multiple of } 4 = 0$$

```
int computeXOR(int n)
{
    // If n is a multiple of 4
    if (n % 4 == 0)
        return n;

    // If n%4 gives remainder 1
    if (n % 4 == 1)
        return 1;

    // If n%4 gives remainder 2
    if (n % 4 == 2)
        return n + 1;

    // If n%4 gives remainder 3
    return 0;
}
```

Rightmost Set Bit (RSB)

Integer. lowestOneBit

$$n \& (1 \ll i) = 0$$

$$(88)_{10} = (1011000)_2 \Rightarrow \text{LSB} = (1000)_2$$

$\begin{array}{r} 65\ 4\ 3\ 2\ 1\ 0 \\ | \quad | \quad | \quad | \downarrow \\ 1011000 \\ 64\ 32\ 16\ 8\ 4\ 2\ 1 \end{array}$

$$\begin{array}{r} 1011000 \\ \& 1000 \\ \hline 0001000 \end{array} \neq 0$$

Burte force \rightarrow Loop from 0th bit to 3rd bit

if $(n \& (1 \ll i) \neq 0)$
return i;

```

public static int log2(int x){
    return (int) (Math.log(x) / Math.log(2)) + 1;
}
// Approach 1:
public static int getFirstSetBit(int n){
    if(n == 0) return 0;
    return log2(Integer.lowestOneBit(n));
}

```

$$O(32) \approx O(\log_2 N) \approx O(1)$$

```

// Approach 2:
for(int bit = 0; bit < 32; bit++){
    if((n & (1 << bit)) != 0){
        return bit + 1;
    }
}
return 0;

```

$$88 = 001011000$$

$\begin{matrix} 6 \\ 2^6 \\ 64 \end{matrix}$
 $\begin{matrix} 5 \\ 2^5 \\ 32 \end{matrix}$
 $\begin{matrix} 4 \\ 2^4 \\ 16 \end{matrix}$
 $\begin{matrix} 3 \\ 2^3 \\ 8 \end{matrix}$
 $\begin{matrix} 2 \\ 2^2 \\ 4 \end{matrix}$
 $\begin{matrix} 1 \\ 2^1 \\ 2 \end{matrix}$
 $\begin{matrix} 0 \\ 2^0 \\ 1 \end{matrix}$

$$\begin{array}{r} 1011000 \\ 0001000 \\ \hline 0001000 \end{array}$$

*rightmost
set
bit*

$$\begin{array}{r}
 n = 1011000 \\
 -n = 0101000 \\
 \hline
 \text{and} \quad 0001000
 \end{array}$$

$$RSB = n \& -n;$$

$$\begin{aligned}
 m &= 88 \\
 1s \text{ complement} &= nn \\
 &= 0100111
 \end{aligned}$$

$$\begin{array}{r}
 2s \text{ complement} \\
 = nn + 1 \\
 0100111 \\
 + 1 \\
 \hline
 0101000 \\
 \underline{+ \quad \quad \quad} \\
 32 \quad 8
 \end{array}$$

```

public static int log2(int x){
    return (int) (Math.log(x) / Math.log(2)) + 1;
}

public static int approach1(int n){
    int rsb = Integer.lowestOneBit(n);
    return log2(rsb);
}

public static int approach2(int n){
    for(int bit = 0; bit < 32; bit++){
        if((n & (1 << bit)) != 0){
            return bit + 1;
        }
    }
    return 0;
}

public static int getFirstSetBit(int n){
    if(n == 0) return 0;
    int rsb = n & -n;
    return log2(rsb);
}

```

$$O(\log_2 N) = O(32)$$

$$O(1)$$

$$\begin{array}{r}
 1011000 \\
 - 0001000 \\
 \hline
 1010000
 \end{array}$$

$$\begin{array}{r}
 R_{SB} = \\
 \hline
 n^{k-n}
 \end{array}$$

$$\begin{array}{r}
 1011000 \xrightarrow{\quad} 1010000 \xrightarrow{\quad} 1000000 \\
 - n^{k-n} \quad 0010000 \quad n^{k-n} 1000000 \\
 \hline
 1000000
 \end{array}$$

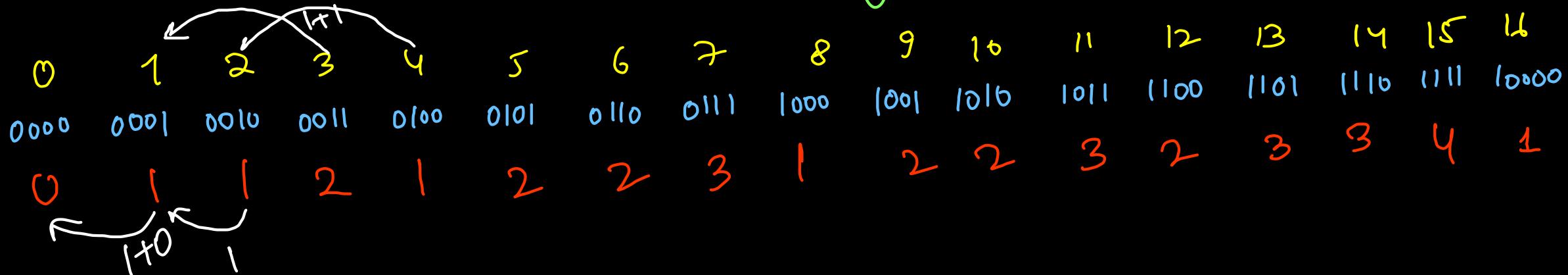
```
public int hammingWeight(int n) {  
    // Approach 1:  
    return Integer.bitCount(n);  
  
    // Approach 2:  
    int count = 0;  
    for(int bit = 0; bit < 32; bit++){  
        if((n & (1 << bit)) != 0)  
            count++;  
    }  
    return count;  
  
    // Approach 3:  
    int count = 0;  
    while(n != 0){  
        int rsb = n & -n;  
        n = n - rsb;  
        count++;  
    }  
    return count;  
}
```

Online Assessment

= Coding round

- ① resume shortlisting
 - ② coding round
 - ③ interview
- coding
HR/behavioral

LeetCode 338 Counting Bits



Input: n = 5

Output: [0,1,1,2,1,2]

Explanation:

0 --> 0

1 --> 1

2 --> 10

3 --> 11

4 --> 100

5 --> 101

$$\begin{array}{r} 32 \\ 16 \\ 8 \\ 4 \\ 2 \\ 1 \end{array}$$

$$10100\textcircled{1} = 41$$

$$10100 = 20$$

$$10100\textcircled{0} = 40$$

$$10100 = 20$$

$$\text{setBit}(41) = 1 + \text{setBit}(20)$$

$$\text{setBit}(40) = \text{setBit}(20)$$

```
// Approach 2: O(N): DP With Bitmasking
for(int idx = 1; idx <= n; idx++){
    // res[idx] = res[idx / 2] + idx % 2;
    res[idx] = res[(idx >> 1)] + (idx & 1);
}

return res;
```

Hamming weight

1011000 (88)

0110010 (50)

XOR
1101010 \Rightarrow count set bits \Rightarrow Answer \Rightarrow 4

XOR gives set bits whenever $a \oplus b$
are different

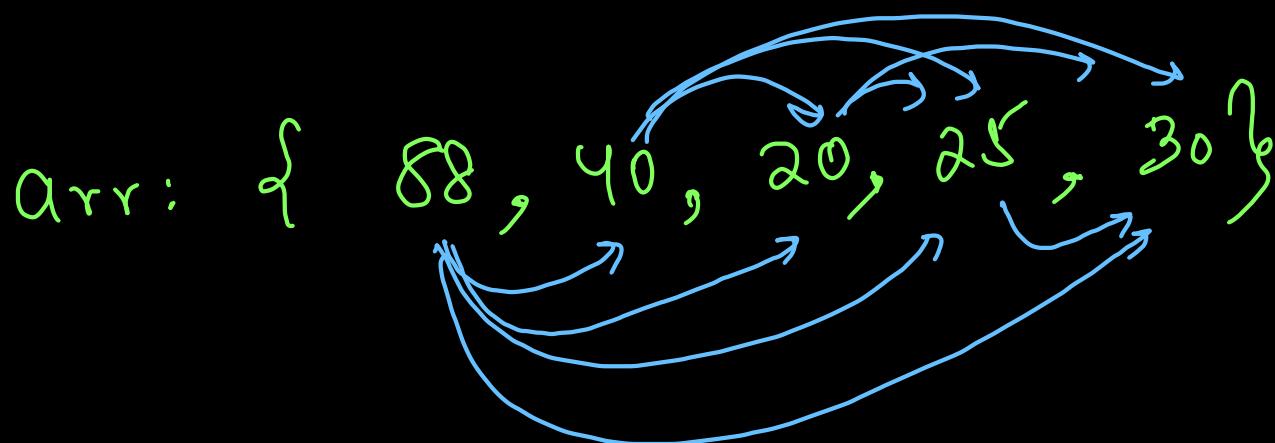
```
class Solution {
    public int hammingDistance(int x, int y) {
        return Integer.bitCount(x ^ y);
    }
}
```

$\hookrightarrow O(\log n)$ worst case

477) Total Hamming distance

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`*.



```
// Approach 1: Brute Force
int sum = 0;
for(int a: nums){
    for(int b: nums){
        sum += Integer.bitCount(a ^ b);
    }
}
return sum / 2;
```

1001001
 1010101
 1001000
 0111111
 1000000

0th bit
1st bit
 :
 :
31st bit

Time $\Rightarrow O(n \log n)$
 Space $\Rightarrow O(1)$

3 (1s)	2 (0s)	$3 * 2$
1 (1s)	4 (0s)	$1 * 4$

```

public int totalHammingDistance(int[] nums) {
    int sum = 0;

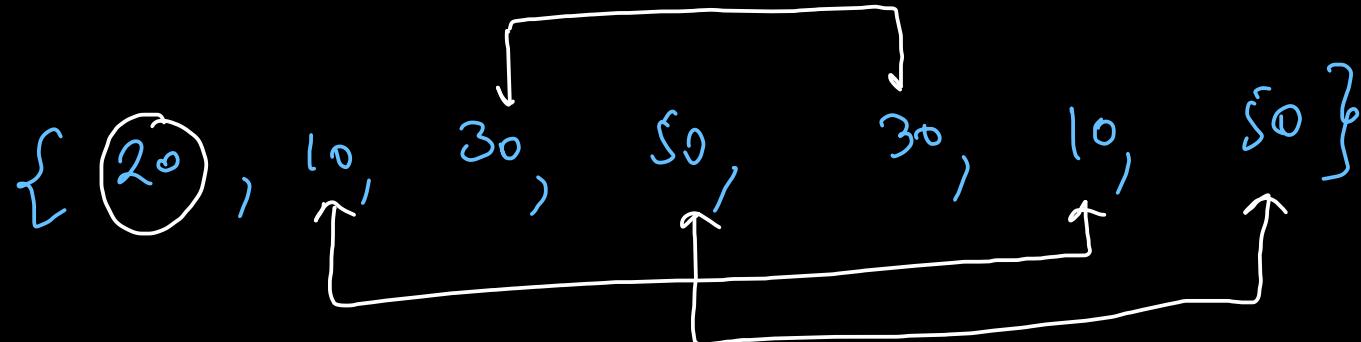
    for(int bit = 0; bit < 32; bit++){
        int zeros = 0, ones = 0;
        for(int val: nums){
            if((val & (1 << bit)) == 0)
                zeros++;
            else ones++;
        }

        sum = sum + zeros * ones;
    }

    return sum;
}
  
```

Single Number - I

Leetcode 136



XOR properties

- ① $a \Delta b = b \Delta a$ {commutative}
- ② $a \Delta 0 = a$ { $0 \Rightarrow$ identitive property}
- ③ $a \Delta a = 0$
- ④ $a \Delta b = c$
 $\Rightarrow a \Delta c = b$
 $\Rightarrow b \Delta c = a$

```
public int singleNumber(int[] nums) {  
    int xor = 0;  
    for(int val: nums) xor = xor ^ val;  
    return xor;  
}
```

Time = $O(n)$ linear
Space = $O(1)$ constant extra

leetcode
2226
start goal

0 1 0 0 1 1 0 → 1 0 0 1 1 0 0

$$xor = 1101010$$

↳ set bits ≥ 4

Copy Set Bits in Range

Given two numbers x and y, and a range $[l, r]$ where $1 \leq l, r \leq 32$.

Find the set bits of y in range $[l, r]$ and set these bits in x also.

$x = 0100\cancel{1}1101$ ($1 \ll \text{bit}$)

$y = 00111000$

The diagram illustrates the bit manipulation process. Bit 1 of y is circled in red. An arrow points from this circled bit to bit 2 of x. A red box highlights the bits from index 2 to 5 of y, corresponding to the range [l, r].

$$l=2, r=5$$

```

int setSetBit(int x, int y, int l, int r){
    for(int bit = l - 1; bit <= r - 1; bit++){ // looping on bits [l, R]
        int mask = (1 << bit);
        if((y & mask) != 0){ // check if y has current set bit
            x = (x | mask); // set the current bit in x
        }
    }
    return x;
}

```

Time $\Rightarrow O(32)$
 $= O(\log N)$

$x = \begin{array}{cccc} 3 & 2 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 1 \end{array} \text{ or } l = 1, R = 4$
 $mask = \begin{array}{ccccc} 0 & 0 & 1 & 0 & 0 \end{array} \text{ and } 0 \quad 3$
 $y = \begin{array}{ccccc} 0 & 0 & 1 & 1 & 0 & 1 \end{array}$

left
right
10100110

↓↓ Reverse Bits (Leetcode 190)

01100101

$O(\log n)$ ✓

```
public int reverseBits(int n) {
    int left = 0, right = 31;
    while(left <= right){
        n = swap(n, left, right);
        left++; right--;
    }
    return n;
}
```

$\rightarrow \text{leftmost} = 1 << \text{left}$
 $\rightarrow \text{rightmost} = 1 << \text{right}$

→ O(1)

```
public int swap(int n, int left, int right){
    int leftmask = (1 << left);
    int rightmask = (1 << right);

    boolean isLeftSet = ((n & leftmask) != 0);
    boolean isRightSet = ((n & rightmask) != 0);

    if(isLeftSet != isRightSet){
        n = n ^ leftmask;
        n = n ^ rightmask;
    }
    return n;
}
```

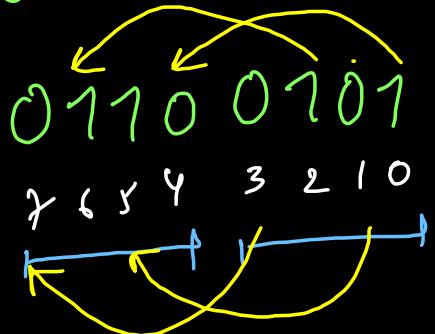
If diff bits
(0,1) or (1,0)
then toggle both
of them

25 14 3 2 1 0
10100110
↙ ↘ ↙ ↘ ↗

Swap Odd &
Even Bits
(GFG)

```
public static int swap(int n, int left, int right){  
    int leftmask = (1 << left);  
    int rightmask = (1 << right);  
  
    boolean isLeftSet = ((n & leftmask) != 0);  
    boolean isRightSet = ((n & rightmask) != 0);  
  
    if(isLeftSet != isRightSet){  
        n = n ^ leftmask;  
        n = n ^ rightmask;  
    }  
  
    return n;  
}  
  
//Function to swap odd and even bits.  
public static int swapBits(int n)  
{  
    for(int bit = 0; bit < 32; bit = bit + 2){  
        n = swap(n, bit, bit + 1);  
    }  
    return n;  
}
```

byte



Swap 2 nibbles in byte

0101 0110
7 6 5 4 3 2 1 0

0, 4

1, 5

2, 6

3, 7

```
static int swap(int n, int left, int right){  
    int leftmask = (1 << left);  
    int rightmask = (1 << right);  
  
    boolean isLeftSet = ((n & leftmask) != 0);  
    boolean isRightSet = ((n & rightmask) != 0);  
  
    if(isLeftSet != isRightSet){  
        n = n ^ leftmask;  
        n = n ^ rightmask;  
    }  
  
    return n;  
}  
  
static int swapNibbles(int N) {  
    N = swap(N, 0, 4);  
    N = swap(N, 1, 5);  
    N = swap(N, 2, 6);  
    N = swap(N, 3, 7);  
    return N;  
}
```

Gray code → Two adjacent numbers
will have 1 different bit

1 bit	2 bit	3 bit
0 0	00 00	000 0
1 1	01 11	001 1
{0, 1}	11 3	011 3
4!	10 2	010 2
		110 6
		111 7
		101 5
		100 4
		{0, 1, 3, 2, 6, 7, 5, 4}

```
public List<Integer> grayCode(int n) {  
    if(n == 1) {  
        List<Integer> res = new ArrayList<>();  
        res.add(0); res.add(1);  
        return res;  
    }  
  
    List<Integer> res = grayCode(n-1);  
  
    for(int idx = res.size() - 1; idx >= 0; idx--) {  
        int ans = (res.get(idx)) | (1 << n);  
        res.add(ans);  
    }  
  
    return res;  
}
```

Time Complexity
 $O(2^n)$
Space Complexity
 $O(n)$

```
class Solution
{
    ArrayList <String> generateCode(int n) {
        if(n == 1){
            ArrayList<String> res = new ArrayList<>();
            res.add("0"); res.add("1");
            return res;
        }

        ArrayList<String> res = generateCode(--n);
        for(int idx = res.size() - 1; idx >= 0; idx--){
            res.add("1" + res.get(idx));
            res.set(idx, "0" + res.get(idx));
        }

        return res;
    }
}
```

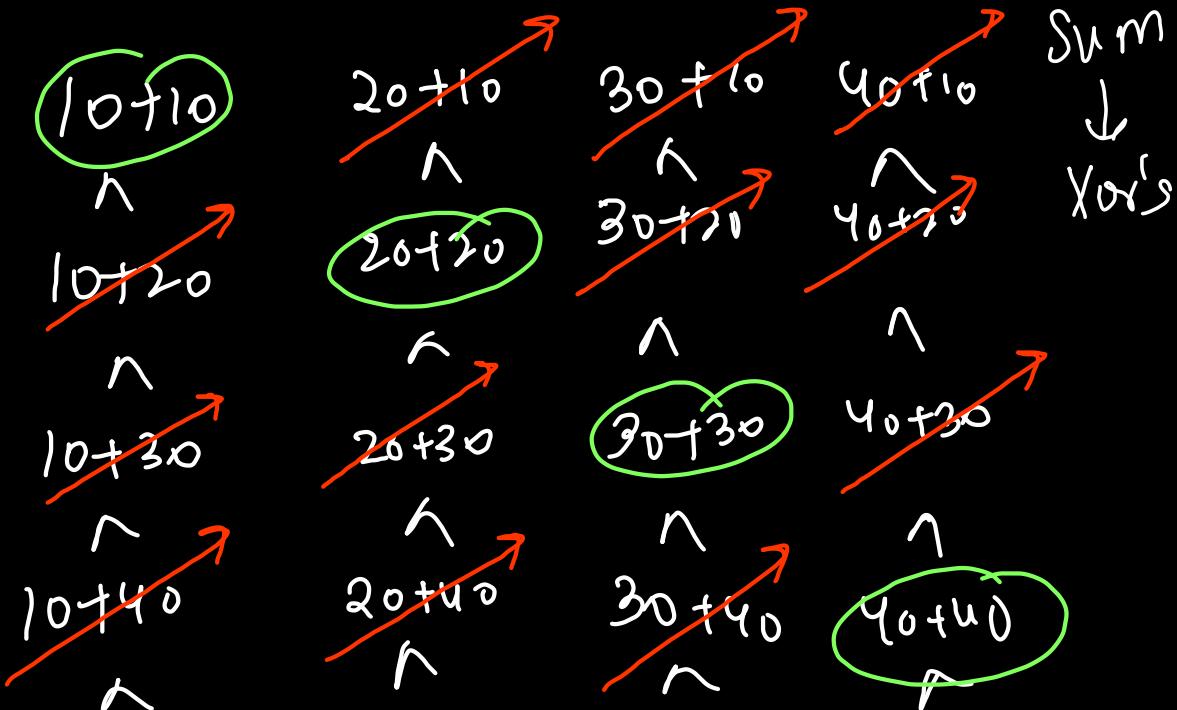
N bit
Gray code
in String
~~=f~~

XOR Sum Problem

$$i, j \in [0, N-1]$$

$\{10, 20, 30, 40\}$

constraint
 $N = 10^5$



$$\text{Sum} \Rightarrow A[i] + A[j]$$

All pairs



Sum



XOR

Time $\Rightarrow O(N)$
 Space $\Rightarrow O(1)$

$$ans = (20 \wedge 40 \wedge 60 \wedge 80)$$

Single Number - II

Leetcode 137

{ 2, 6, 5, 4, 4, 6, 6, 4, 2, 2, 3, 3, 3 }

010

110

010

110

010

110

700

700

700

011

011

011

101

↓

unique n°

$$\frac{1}{3^{p_2+1}}$$

$$\frac{0}{3^{p_3}}$$

$$\frac{1}{3^{p_1}}$$

$\{ 010, 110, 101, 100, 100, 110, 110, 100, 010, 010, 011, 011, 011 \}$
 $\{ 2, 6, 5, 4, 4, 6, 6, 4, 2, 2, 3, 3, 3 \}$

```

public int singleNumber(int[] nums) {
    int res = 0;

    for(int bit = 0; bit < 32; bit++){
        int mask = (1 << bit);
        int counter = 0;

        for(int val: nums){
            if((val & mask) != 0){
                counter++;
            }
        }

        if(counter % 3 != 0)
            res = (res | mask);
    }

    return res;
}

```

$$\begin{aligned}
\text{count} &\rightarrow 1+3 \\
&= 4
\end{aligned}$$

$$\begin{aligned}
\text{count} &\rightarrow 1 \\
&\quad 2+7 \\
&= 9
\end{aligned}$$

Count ~7

yes ~~1~~ 0 ~~1~~

2nd 1st 0th

hc 260 Single Number \rightarrow III

5	2	3	4	2	4	5	6
.
101	010	011	100	010	100	101	110
0	0	0	0	0	0	0	0

$$\text{nor} = \overbrace{5 \wedge 2 \wedge 3 \wedge 4}^{\text{for } 3 \text{ unique}} \wedge \overbrace{1 \wedge 2 \wedge 4}^5 \wedge \overbrace{1 \wedge 5}^6 = \boxed{3 \wedge 6}$$
$$= 010 \wedge 110$$
$$= 101 = 5$$



second
 $= (3 \wedge 6) \wedge 10$
 $= 6$

```

class Solution {
    public int[] singleNumber(int[] nums) {
        int xor = 0;
        for(int val: nums) xor = xor ^ val; → xor = 3 ⊕ 6
O(n) ←
        int rsb = Integer.lowestOneBit(xor);
        int first = 0; ↳ n & -n
O(n) ←
        for(int val: nums){
            if((val & rsb) != 0) → first = 3 ⊕ 3 ⊕ 6 = 3
                first = first ^ val;
        }
        int second = xor ^ first; → second = (3 ⊕ 6) ⊕ 3 = 6
        return new int[]{first, second};
    }
}

```

$\{ \begin{matrix} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11 \end{matrix} \}$
 ↓
 (i, j, k)

Given an array of integers arr .

We want to select three indices i , j and k where $(0 \leq i < j \leq k < \text{arr.length})$.

Let's define a and b as follows:

- $a = \text{arr}[i] \wedge \text{arr}[i + 1] \wedge \dots \wedge \text{arr}[j - 1]$
- $b = \text{arr}[j] \wedge \text{arr}[j + 1] \wedge \dots \wedge \text{arr}[k]$

Note that \wedge denotes the **bitwise-xor** operation.

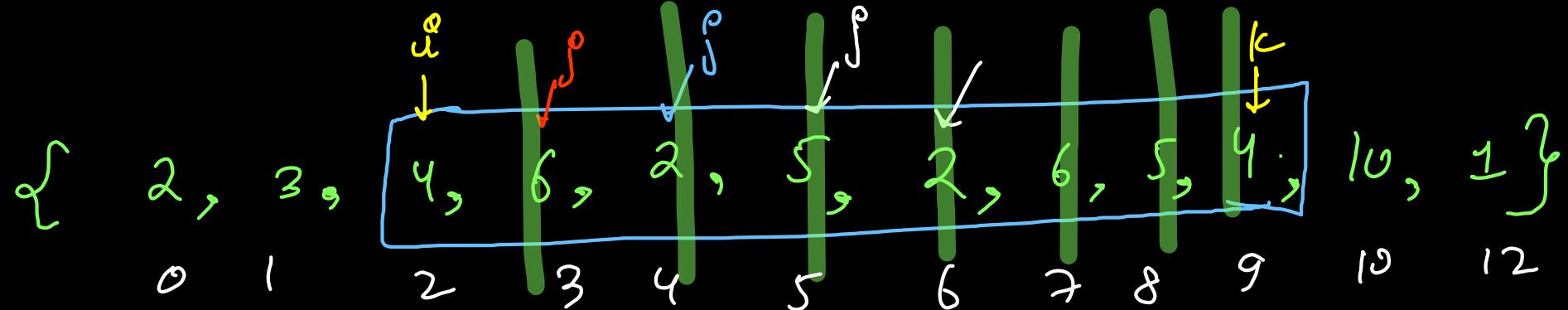
Return *the number of triplets* (i , j and k) Where $a == b$.

$$i < j \leq k$$

$$[i \dots j-1] = [j \dots k]$$

$$4 \wedge 6 = 2 \wedge 8 \wedge 10 \wedge 12 \wedge 14$$

$$(2, 4, 9)$$



$$i < j < k$$

$$\cancel{4 \wedge 6 \wedge 2 \wedge 5 \wedge 2 \wedge 6 \wedge 1 \wedge 8 \wedge 1 \wedge 4} = 0$$

$$[i \dots j \dots l] = [j \dots k]$$

$$y = \cancel{6 \wedge 2 \wedge 5 \wedge 2 \wedge 8 \wedge 1 \wedge 4}$$

$$4 \wedge 6 = \cancel{2 \wedge 5 \wedge 2 \wedge 6 \wedge 5 \wedge 4}$$

$$4 \wedge 6 \wedge 2 = \cancel{5 \wedge 2 \wedge 6 \wedge 5 \wedge 4}$$

$$4 \wedge 6 \wedge 2 \wedge 5 \wedge 2 = 6 \wedge 5 \wedge 4$$

```

public int countTriplets(int[] arr) {
    int ans = 0;
    for(int l = 0; l < arr.length; l++){
        int xor = 0;
        for(int r = l; r < arr.length; r++){
            xor = xor ^ arr[r];
            if(xor == 0) ans += (r - l);
        }
    }
    return ans;
}

```

$\{10, 15\}$
 including both
 $15-10+1 = 6$
 excluding 10 or 15
 $15-10 = 5$
 excluding both
 $15-10-1 = 4$

Time $\Rightarrow O(n^2)$, Space $\Rightarrow O(1)$

Property :

If subarray has $xor = 0$
 \Rightarrow we can partition at
 any index

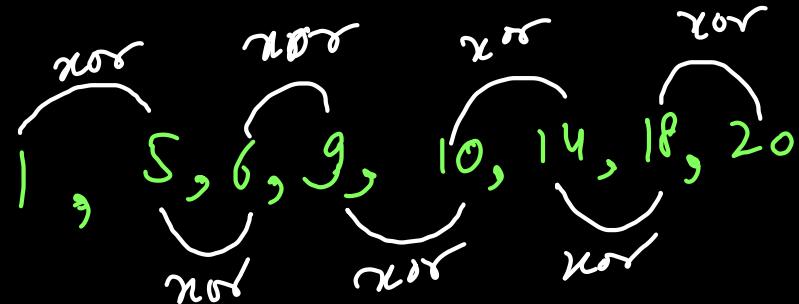
To get $a = b$
 $[i \dots j-1] = [j \dots k]$

6 20 5 1 14 18 9

min xor pair

10

Sort



Brute force

$\rightarrow O(n^2)$ nested loops

TLE

0 000
0 001

0 010
0 011

010 0

010 1

011 0

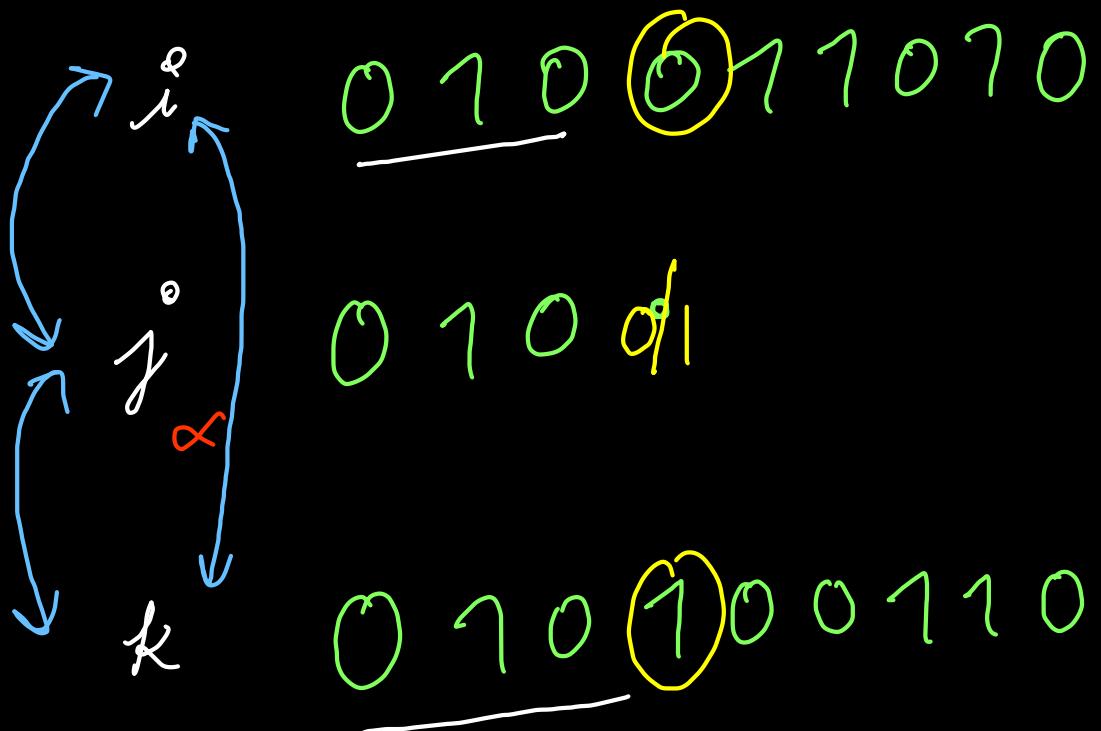
011 1

100 0

```
public int findMinXor(int[] A) {  
    Arrays.sort(A);  
  
    int ans = Integer.MAX_VALUE;  
    for(int idx = 0; idx < A.length - 1; idx++){  
        ans = Math.min(ans, A[idx] ^ A[idx + 1]);  
    }  
    return ans;  
}
```

↑
adjacent

$i^o < j^o < k$



$i^o \wedge k^o$ will never
be min^m

minⁿ will be
either $i^o \wedge j^o$ or $j^o \wedge k^o$

$$i^o \wedge k^o = 0001 \underline{\quad}$$

if $i^o \wedge j^o \Rightarrow 0 \Rightarrow i^o \wedge j^o : 0000 \dots$
 $\Rightarrow 1 \Rightarrow j^o \wedge k^o : 0000 \dots$

Max xor pair

Sample Input 0

10

15

Sample Output 0

7

$$10 \oplus 10 = 0$$

$$10 \oplus 11 = 1$$

$$10 \oplus 12 = 6$$

$$10 \oplus 13 = 7$$

$$10 \oplus 14 = 4$$

$$10 \oplus 15 = 5$$

$$11 \oplus 11 = 0$$

$$11 \oplus 12 = 7$$

$$11 \oplus 13 = 6$$

$$11 \oplus 14 = 5$$

$$11 \oplus 15 = 4$$

$$12 \oplus 12 = 0$$

$$12 \oplus 13 = 1$$

$$12 \oplus 14 = 2$$

$$12 \oplus 15 = 3$$

$$13 \oplus 13 = 0$$

$$13 \oplus 14 = 3$$

$$13 \oplus 15 = 2$$

$$14 \oplus 14 = 0$$

$$14 \oplus 15 = 1$$

$$15 \oplus 15 = 0$$

$$a \wedge b = \max$$

$$l \leq a, b \leq r$$



$$\textcircled{10} = \underline{1}010$$

$$11 = 1011$$

$$12 = 1100$$

$$13 = 1101$$

$$14 = 1110$$

$$\textcircled{15} = \underline{1}111$$



All
bits
after
lsb
in $a \wedge b$
will be 1

$l = \boxed{1001} \underline{0010}$

1001 0011

1001 0100 → 1011

;

;

1001

$r = \boxed{1001} \underline{1011}$

$lsb = 1011$

$res = \emptyset \emptyset \emptyset \emptyset$
1111

```
public static int maximizingXor(int l, int r) {  
    int lsb = Integer.highestOneBit(l ^ r);  
    int res = 0;  
    while(lsb != 0){  
        res = (res | lsb);  
        lsb = lsb >> 1;  
    }  
    return res;  
}
```

} set all bits
from lsb till 0th bit

10

~~x~~
11

~~x~~
12

~~x~~
13

~~x~~
14

15

1010

1011

1100

1101

1110

1111

1000

$\lambda = \overline{1001} \quad \overline{0010}$

1001 0011

1001 0100 → 1011

⋮

1001

$\gamma = \overline{1001} \quad \overline{1011}$

$\gamma\bar{\gamma} = 10010000$

```
public int rangeBitwiseAnd(int left, int right) {  
    int lsb = Integer.highestOneBit(left ^ right);  
  
    int res = left;  
    while(lsb != 0){  
        res = res & ~lsb; → set all bits as 0  
        lsb = lsb >> 1; ← from lsb till  
    } ← 0m bit/  
  
    return res;  
}
```

100]0000

100]1010

$$\begin{array}{r} \text{l n r} = \underline{\underline{00001----}} \\ \text{lsb} = 000010000 \end{array}$$

multiples of 3

3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, ...
decimal = sum of digits will be multiple of 3.

0011, 0110, 1001, 1100, 1111, 10010.

binary no of 3 = 11
 \Rightarrow sum of odd bits
= sum of even bits

m decimal multiples of 11 \Rightarrow $(\text{sum of odd digits} - \text{sum of even digits}) = \text{multiple of } 11$

??, ??, --- ??

??, ??, ??, ??, ??, --- ??

```
int isDivisible(String s) {  
    int odd = 0, even = 0;  
    for(int idx = 0; idx < s.length(); idx++){  
        if(s.charAt(idx) == '1'){  
            if(idx % 2 == 1) odd++;  
            else even++;  
        }  
    }  
  
    // System.out.println(odd + " " + even);  
    if(Math.abs(odd - even) % 3 == 0) return 1;  
    return 0;  
}
```

$$\begin{aligned}
 4^0 &= 1 &= & \text{1} \\
 4^1 &= 4 &= & \text{1000} \\
 4^2 &= 16 &= & \text{100000} \\
 4^3 &= 64 &= & \text{1000000} \\
 4^4 &= 256 &= & \text{1000000000}
 \end{aligned}$$

Power of 4

$\text{setbits} = 1$ $O(\log n)$

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

→ It is power of 2

→ set bit should be at even index.

```

public boolean isPowerOfFour(int n) {
    int setbits = 0;
    for(int bit = 0; bit < 32; bit++){
        if((n & (1 << bit)) != 0){
            setbits++;
            if((bit & 1) == 1) return false;
            // set bit should not be at odd index
        }
    }
    return (setbits == 1);
    // power of 4 should be power of 2
}
  
```

1	2	4	8	16	32	64	128	256	512
✓	✗	✓	✗	✓	✗	✓	✗	✓	✗
✓/3	1	2	1	2	1	2	1	2	1

```
public boolean isPowerOfFour(int n) {
    return ((n & (n - 1)) == 0) && (n % 3 == 1);
}
```

↳ Time $\Rightarrow O(1)$

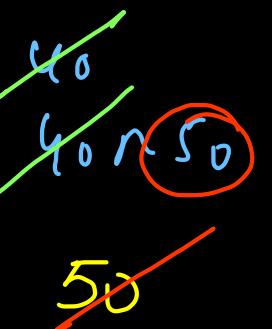
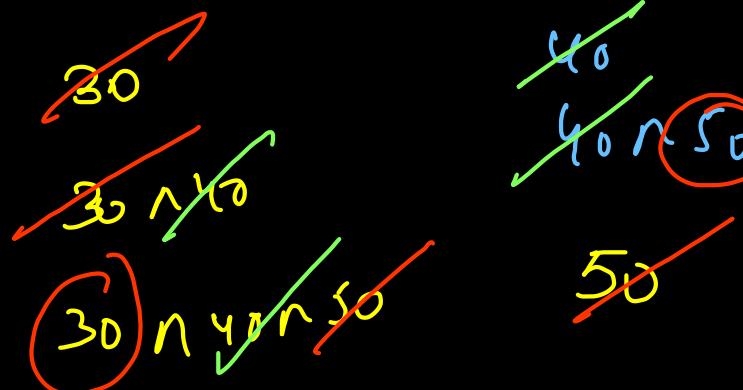
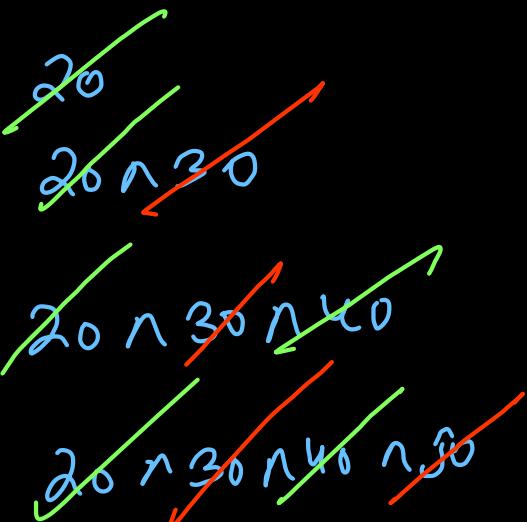
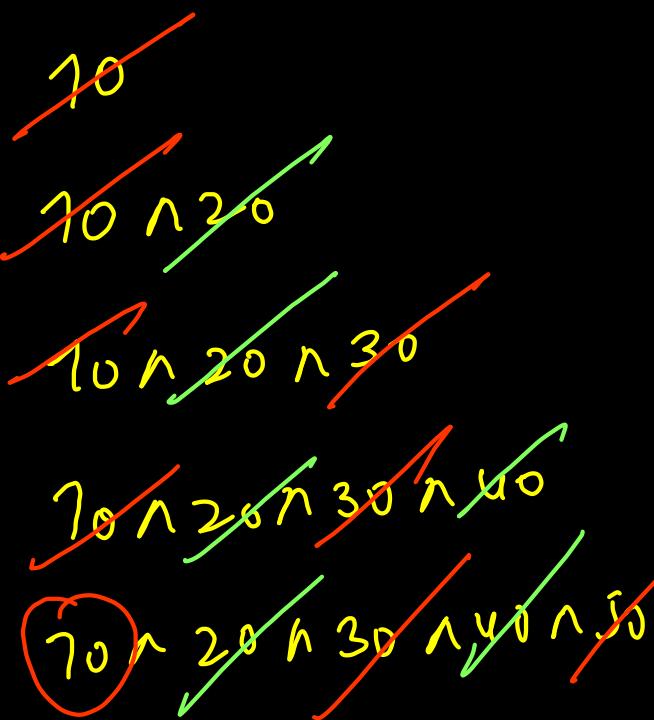
Space $\Rightarrow O(1)$

Game of XOR

{ 0, 1, 2, 3, 4
10, 20, 30, 40, 50 }
 \downarrow \downarrow \downarrow \downarrow \downarrow
1 * 5 2 * 4 3 * 3 4 * 2 5 * 1

$$\text{res} = 10 \wedge 30 \wedge 50$$

occurrences(i) = $\frac{(i+1) * (n-i)}{2}$
Odd $\Rightarrow \text{arr}[i]$ once
Even $\Rightarrow \text{arr}[i]$ don't



Solution 1

```
static int gameOfXor(int N , int[] A) {  
    int res = 0;  
  
    for(int idx = 0; idx < N; idx++){  
        long occurrences = (idx + 1l) * (N - idx);  
        if((occurrences & 1) == 1){  
            res = res ^ A[idx];  
        }  
    }  
  
    return res;  
}
```

Time $\Rightarrow \mathcal{O}(n)$
Space $\Rightarrow \mathcal{O}(1)$

↓ small optimization

Solution 2

```
static int gameOfXor(int N , int[] A) {  
    if(N % 2 == 0) return 0;  
    int res = 0;  
    for(int idx = 0; idx < N; idx += 2)  
        res = res ^ A[idx];  
    return res;  
}
```

Example Input

Input 1:

$$A = [5, 4, 10, 15, 7, 6] \quad \text{*s* & *t* = 0}$$

B = 5

Input 2:

A = [3, 6, 8, 10, 15, 50]

B = 5

Example Explanation

Explanation 1:

$$(10 \ ^ \ 15) = 5$$

Explanation 2:

$$(3 \ ^ \ 6) = 5 \text{ and } (10 \ ^ \ 15) = 5$$

Count Pairs with given XOR

$$a[i] \wedge a[j] = B \Rightarrow a[i^0] \wedge B = a[j^0]$$

i, i⁰ *i ≤ j⁰*

↑
search
(hashset)

Approach 1) Brute force

nested loop

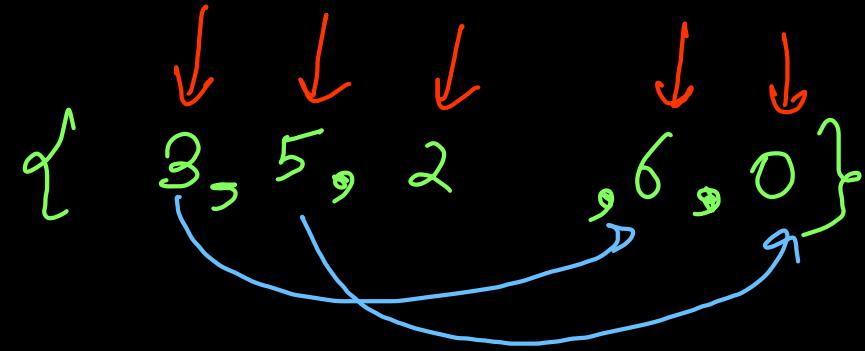
⇒ $O(n^2)$ time

$$\text{Property: } \rightarrow a \wedge b = c \Rightarrow \begin{aligned} a \wedge c &= b \\ b \wedge c &= a \end{aligned}$$

```
public int solve(int[] A, int B) {
    HashSet<Integer> vis = new HashSet<>();
    for(int val: A) vis.add(val); O(n)
    int pairs = 0;
    for(int val: A){
        if(vis.contains(val ^ B) == true){
            pairs++;
        }
    }
    return pairs / 2;
}
```

$$\text{Time} = O(n)$$

$$\text{Space} = O(n)$$



$$B = 5$$

$$\text{pairs} = \begin{matrix} \emptyset \\ \times \\ \times \\ \checkmark \\ 4 \end{matrix}$$

$$3 \wedge 5 = 6$$

$$3 \wedge 6 = 5$$

$$5 \wedge 5 = 0$$

$$5 \wedge 0 = 5$$

$$2 \wedge 5 = 7$$

$$6 \wedge 5 = 3$$

$$0 \wedge 5 = 5$$

Leetcode 1310 XOR Subarray Queries

$\{2, 6, 4, 5, 0, 1\}$

0 1 2 3 4 5

$0 \leq l \leq r \leq n-1$

Brute force

$O(n * q)$ loop from l to r for each query

$$n * q = 3 * 10^4 * 3 * 10^4 = 9 * 10^8$$

9sec \Rightarrow TLE

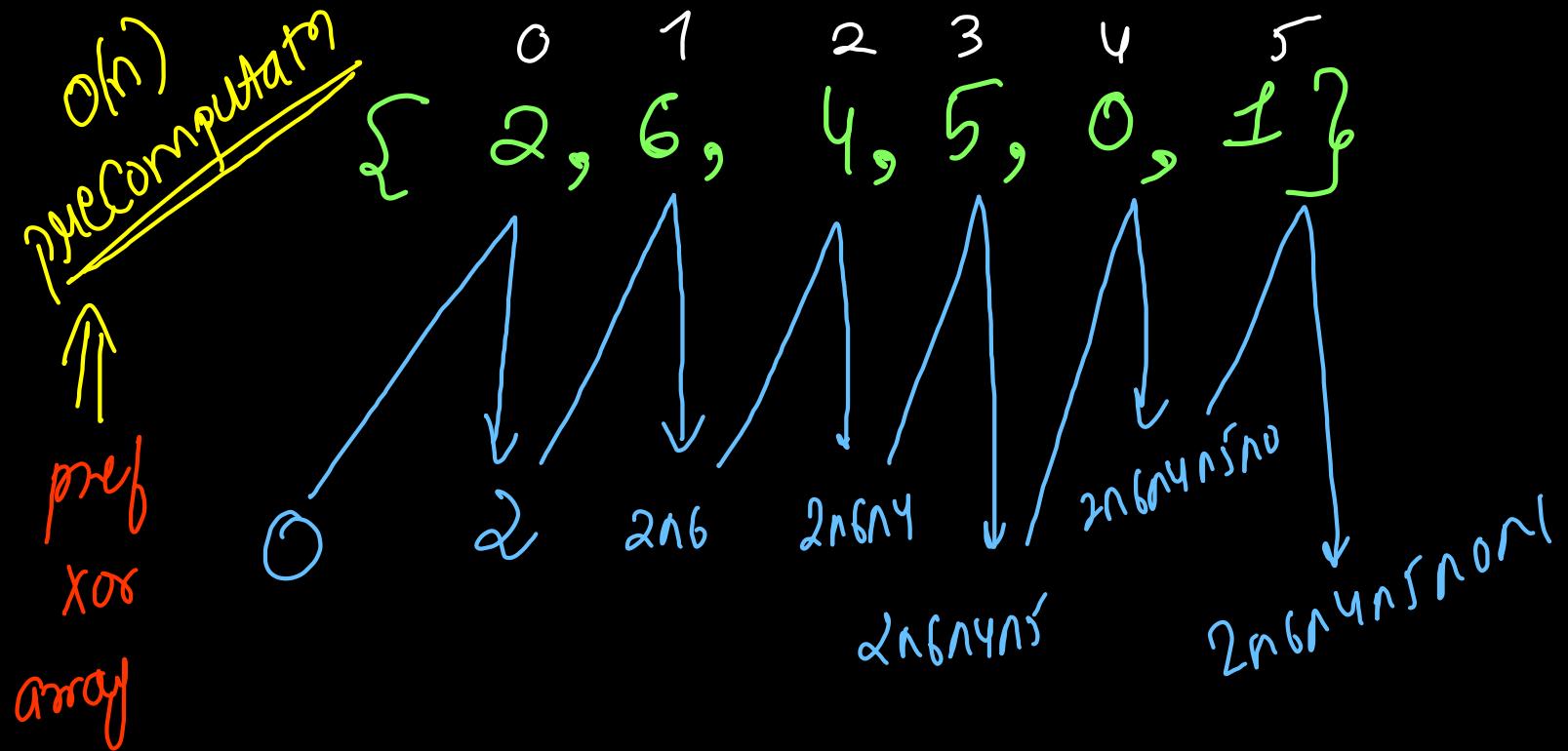
$$\{2, 6, 4, 5, 0, 1\}$$

0 1 2 3 4 5

$$[0, 2] \Rightarrow 2 \wedge 6 \wedge 4$$
$$[3, 5] \Rightarrow 5 \wedge 0 \wedge 1$$
$$[1, 4] \Rightarrow 6 \wedge 4 \wedge 5 \wedge 0$$
$$[0, 5] \Rightarrow 2 \wedge 6 \wedge 4 \wedge 5 \wedge 0 \wedge 1$$
$$\{2, 6, \boxed{4, 5, 0}, 1\}$$

0 1 2 3 4 5

$$[\ell, r] = 4 \wedge 5 \wedge 0$$
$$= (4 \wedge 5 \wedge 0 \wedge 2 \wedge 6) \\ \wedge (2 \wedge 6)$$
$$= \text{pref}[4] \wedge \text{pref}[2-1]$$
$$[\ell, r] = \text{pref}[r] \wedge \text{pref}[\ell-1]$$



Total time
 $= O(n) + O(q)$
 ↓
 precomp query
Accepted!

each query can be solved
 in constant time

$O(q)$

$$\text{query}[l, r] = \text{pref}[r] \wedge \text{pref}[l-1]$$

0 1 2 3 4 5
{ 2, 6, 4, 5, 0, 1 }

pref
xor
[0,0] [0,1] [0,2] [0,3] [0,4] [0,5]

[0,2] \Rightarrow prefix[2]

[3,5] \Rightarrow prefix[5] \wedge prefix[2]
~~prefix[5] \wedge prefix[2]~~ \wedge ~~prefix[5] \wedge prefix[2]~~

Approach 1)

```
public int[] xorQueries(int[] arr, int[][] queries) {  
    int n = arr.length, q = queries.length;  
  
    int[] pref = new int[n];  
    pref[0] = arr[0];  
    for(int i = 1; i < n; i++){  
        pref[i] = pref[i - 1] ^ arr[i];  
    }  
  
    int[] res = new int[q];  
    for(int i = 0; i < q; i++){  
        int l = queries[i][0];  
        int r = queries[i][1];  
  
        res[i] = pref[r];  
        if(l > 0) res[i] = res[i] ^ pref[l - 1];  
    }  
    return res;  
}
```

$\uparrow O(n)$
precomp

$\uparrow O(q)$
work

Total time $\Rightarrow O(n+q)$
linear

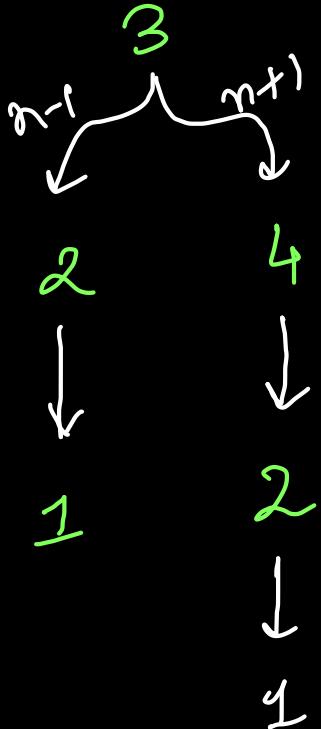
Approach 2)

```
public int[] xorQueries(int[] arr, int[][] queries) {  
    int n = arr.length, q = queries.length;  
  
    int[] pref = new int[n + 1];  
    for(int i = 1; i <= n; i++){  
        pref[i] = pref[i - 1] ^ arr[i - 1];  
    }  
  
    int[] res = new int[q];  
    for(int i = 0; i < q; i++){  
        int l = queries[i][0];  
        int r = queries[i][1];  
        res[i] = pref[r + 1] ^ pref[l];  
    }  
    return res;  
}
```

Integer Replacement

0 ops

1 op

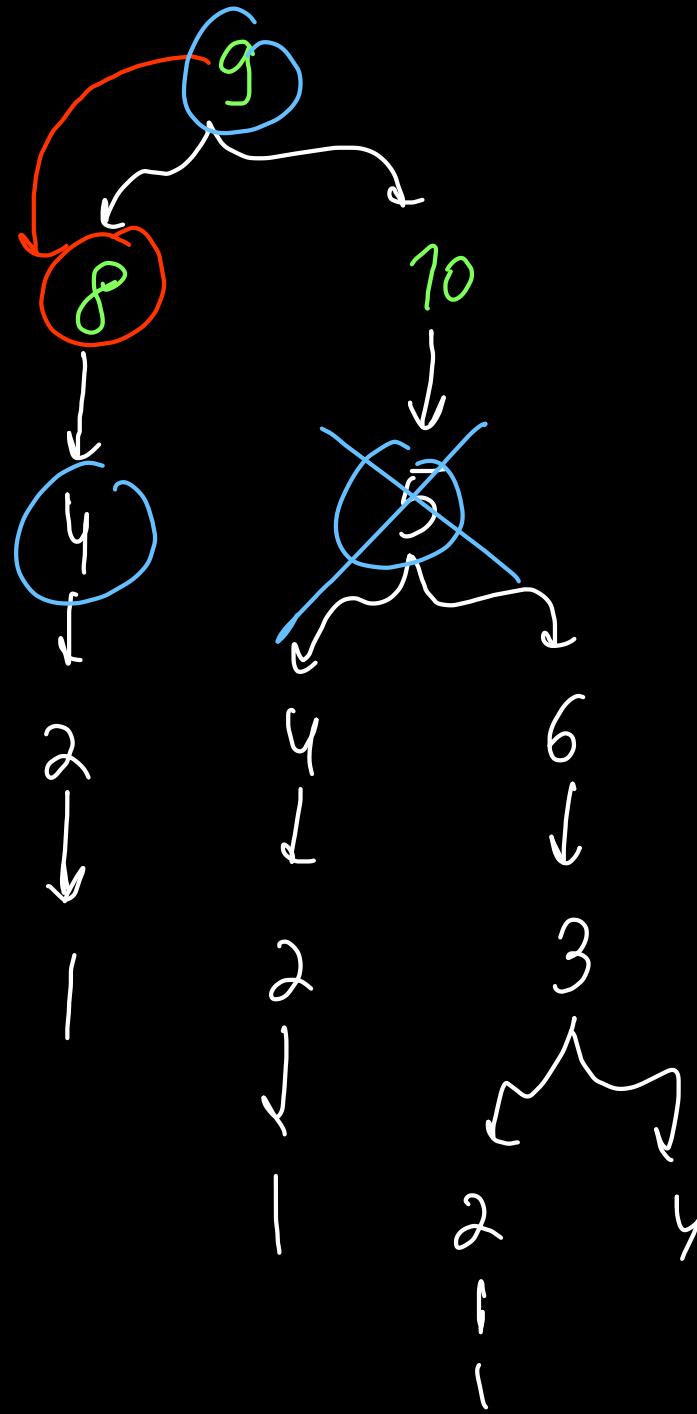
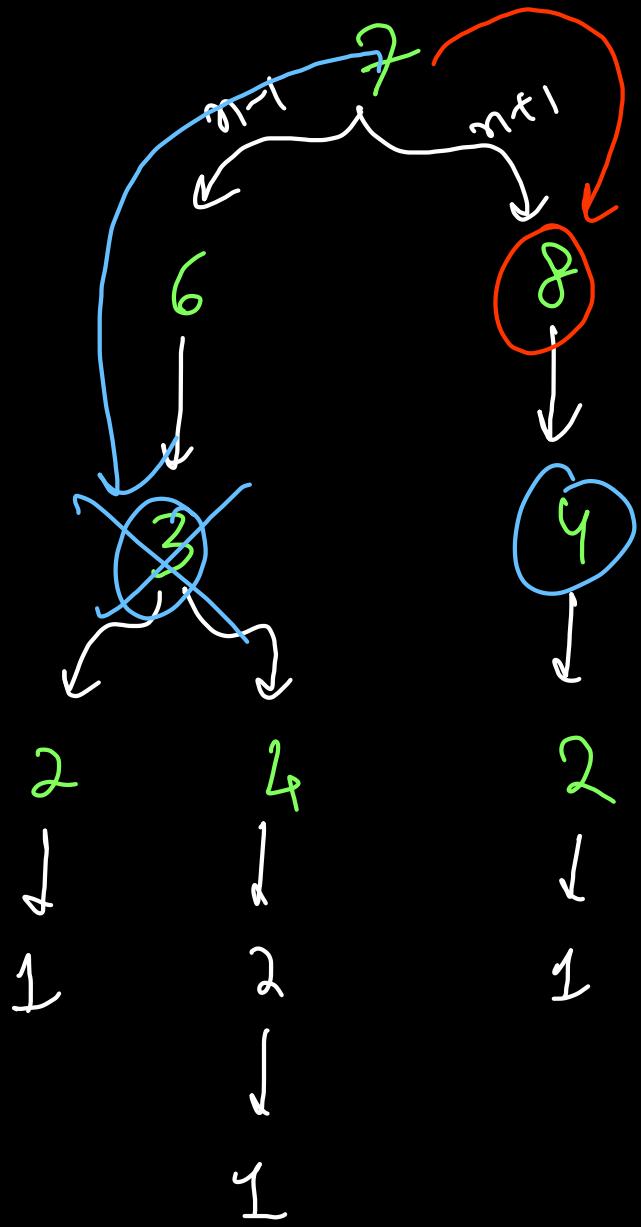


2 ops

2 ops



3 ops



```
public int integerReplacement(int n) {  
    if(n == Integer.MAX_VALUE) return 32;  
    if(n <= 3) return n - 1;  
    if(n % 2 == 0) return 1 + integerReplacement(n / 2);  
    if(((n - 1) / 2) % 2 == 0) → greedy choice  
        return 1 + integerReplacement(n - 1);  
    else return 1 + integerReplacement(n + 1);  
}
```

$O(\log N)$
 $\sqrt{2}$
time

	2	1	0	
	c	b	a	
decimal				binaries
0	0	0	0	
1	0	0	1	
2	0	1	0	
3	0	1	1	
4	1	0	0	
5	1	0	1	
6	1	1	0	
7	1	1	1	

subset

" "

"a"

"b"

"ab"

"c"

"ac"

"bc"

"abc"

Power Set

{ ~~"0"~~, "a", "b", "c", "d",
 "ab", "ac", "ad", "bc", "bd",
 "abc", "abd", "acd", "bcd", "abcd" }

- Empty subset ignore
- Lexicographically sorted

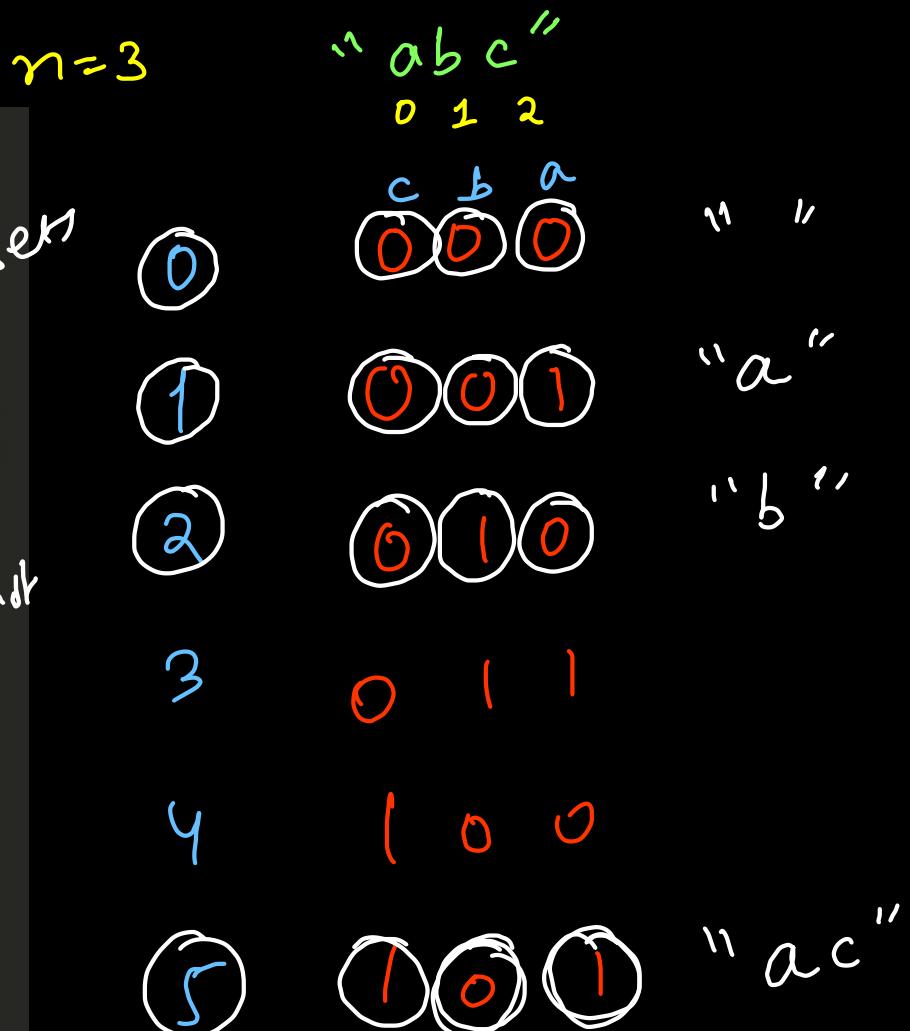
$$\delta = 1 \ll 3$$

```

public List<String> AllPossibleStrings(String s)
{
    List<String> res = new ArrayList<>();
    int n = s.length();
    for(int decimal = 1; decimal < (1 << n); decimal++){
        String subset = "";
        for(int bit = 0; bit < n; bit++){
            if((decimal & (1 << bit)) != 0) → Check set
                subset += s.charAt(bit); → Set or not
        }
        res.add(subset);
    }

    Collections.sort(res);
    return res;
}

```



$$1 \ll 0 = 1$$

$$\begin{aligned}1 \ll 1 &= 2 \\1 \ll 2 &= 4\end{aligned}$$

$$6 \quad | \quad | \quad 0$$

$$7 \quad | \quad | \quad |$$

Exponentiation

$$3^{10} = 3 \times 3$$

$$3^{10} = 3^8 \cdot 3^2$$

$$a^{x+y} = a^x \cdot a^y$$

$$5^{15} = 5^8 \cdot 5^4 \cdot 5^2 \cdot 5^1$$

power = n = sum terms of power of 2.

$$3^{11} = 3^1 * 3^2 * 3^8$$

✓ ✓ ✓

$$11 = \begin{matrix} 8 & 2 & 2 & 1 \\ 2 & 2 & 2 & 0 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 \end{matrix}$$

multiplier
(n) = $\frac{(3 * 3) * (3 * 3)}{(3 * 3 * 3 * 3)}$

ans = $1 * 3 * 3^2 * 3^8$

$$2^{-5} : \left(\frac{1}{2}\right)^5 : (0.5)^5 = 6^{-5} * 0.5 * 0.5 * 0.5 * 0.5$$

$$x^n = \left(\frac{1}{n}\right)^n$$

```

public double myPow(double base, int power) {
    double ans = 1.0;
    long n = power;

    if(n < 0){
        base = 1.0 / base;
        n = -n;
    }

    for(int bit = 0; bit < 64; bit++){
        if((n & (1l << bit)) != 0){
            ans = ans * base;
        }
        base = base * base;
    }

    return ans;
}

```

$$13 = 8 + 4 + 1$$

$$= 2^3 + 2^2 + 2^0$$

$$6^{13} = \cancel{6}^1 \times \cancel{6}^4 \times \cancel{6}^8$$

$$\text{base} = (6^2)^2 = (6^4)^2 = 6^8$$

$$\text{power} = 13$$

$$\text{ans} = 1 \times 6^1 \times 6^4 \times 6^8 = 6^{13}$$

Time $O(\log N)$

$$13 = \begin{matrix} 1 & 1 & 0 & 1 \\ \dots & \dots & \dots & \dots \end{matrix}$$

Logarithmic

Space $O(1)$

$$6^{13} = \checkmark 6^1 * 6^4 * 6^8$$

power
13 = 1101₂

base = 6

ans = 1

$$6^1$$

$$1 * 6 = 6^1$$

$$6^4$$

$$1 * 6 * 6^4 = 6^5$$

$$6^8$$

$$6^5 * 6^8 = 6^{13}$$

$$3 = ?$$

$$1 = ?$$

Approach 1)

```
public double myPow(double base, int power) {  
    double ans = 1.0;  
    long n = power;  
  
    if(n < 0){  
        base = 1.0 / base;  
        n = -n;  
    }  
  
    for(int bit = 0; bit < 64; bit++){  
        if((n & (1l << bit)) != 0){  
            ans = ans * base;  
        }  
        base = base * base;  
    }  
  
    return ans;  
}
```

Approach 2)

```
public double myPow(double base, int power) {  
    long n = power;  
  
    if(n < 0){  
        base = 1.0 / base;  
        n = -n;  
    }  
  
    double ans = 1.0;  
  
    while(n != 0){  
        if((n & 1) == 1){  
            ans = ans * base;  
        }  
        base = base * base;  
        n = n >> 1;  
    }  
  
    return ans;  
}
```

