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}

Decimal to Any Base

$n = (60)_{10}$

base = $(?)_2$

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)

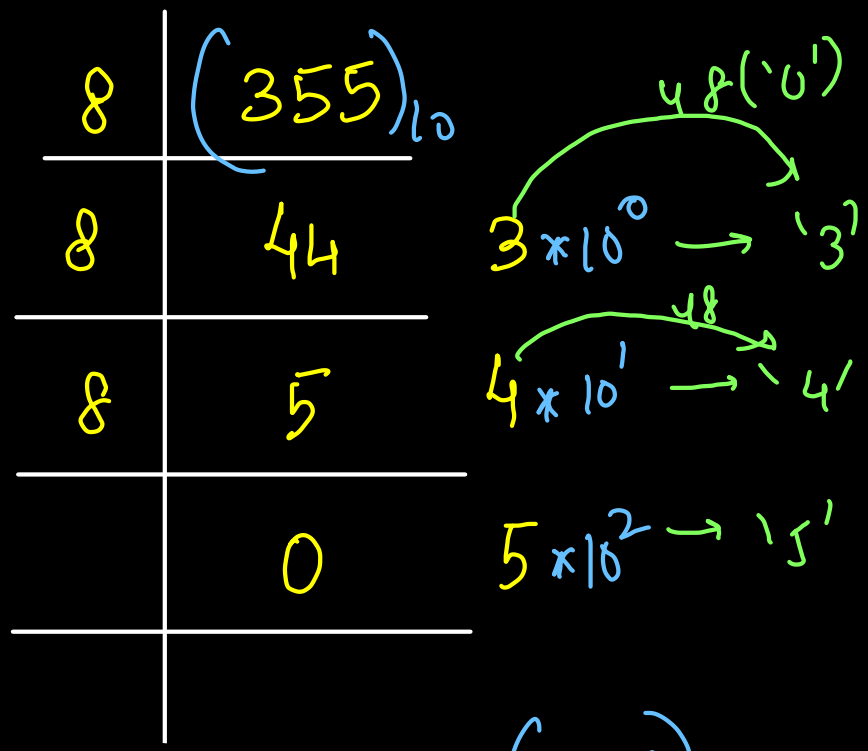
source base (s)

⇒ Division & modulus by d

⇒ multiplier by s

$= 0 * 10^0 + 0 * 10^1 + 1 * 10^2 + 1 * 10^3 + 1 * 10^4 + 1 * 10^5$

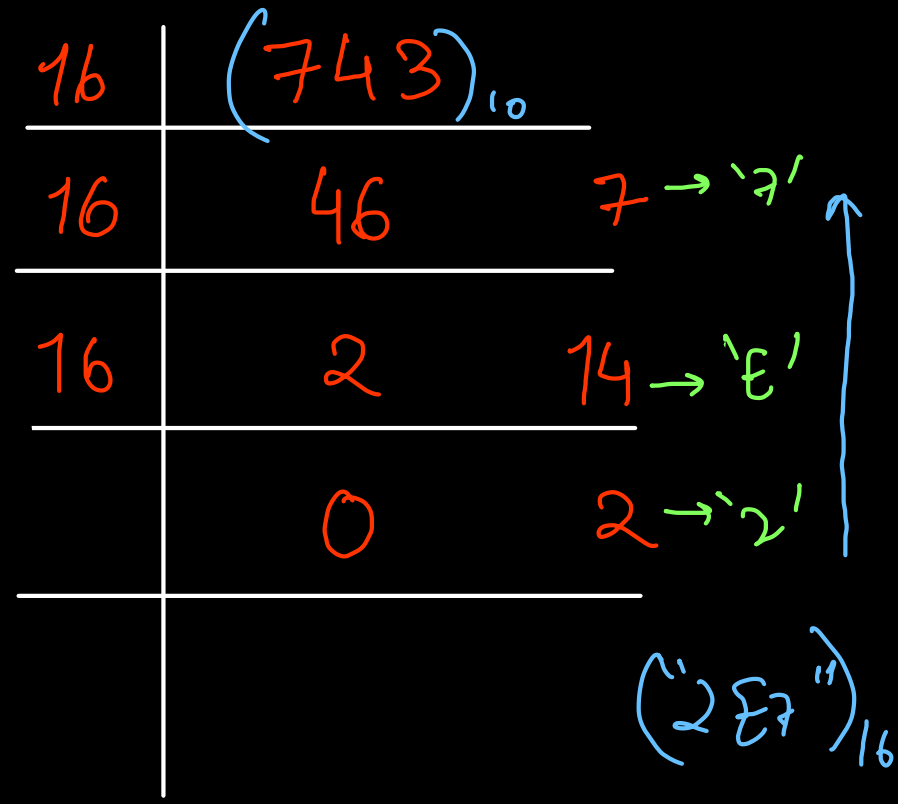
$= (111100)_2$



$= (543)_8$

destinate base = 8 (octal)

given base = 10 (decimal)



given base = 10 (decimal)

required base = 16 (hexadecimal)

```java
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();
    }
}
```

Time ~> logarithmic $O(\log_B N)$

$\approx O(1)$

Space ~> $O(1)$

constant extra space

Sum of digits in Base K

```java
class Solution {
    public int sumBase(int n, int b) {
        int sum = 0;

        while(n != 0){
            sum += (n % b);
            n = n / b;
        }

        return sum;
    }
}
```

Time ⇒ $O(\log_b N)$

Space ⇒ $O(1)$

Leetcode 1837

Base 7 (504)

```java
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 $O(1)$

7	$(-239)_{10}$	
7	-34	$-1 * 10^0$
7	-4	$-6 * 10^1$
	0	$-4 * 10^2$

$-1 + (-60) + (-400)$

$= -461$

binary operators

AND (और)

0 & 0 = 0

0 & 1 = 0

1 & 0 = 0

1 & 1 = 1

OR (या)

0 | 0 = 0

1 | 0 = 1

0 | 1 = 1

1 | 1 = 1

XOR

- if diff bit ⇒ res = 1
- same bits ⇒ res = 0

0 ^ 0 = 0

0 ^ 1 = 1

1 ^ 0 = 1

1 ^ 1 = 0

NOT {tilde} unary

~0 = 1

~1 = 0

logical operators → boolean {&&, ||, !}

bitwise operators → bits {&, |, ~}

Convert Any Base to Decimal

power → 2

division/modulus → 10

10	$(1001101)_2$	
10	100110	$1 * 2^0 = 1$
10	10011	$0 * 2^1$
10	1001	$1 * 2^2 = 4$
10	100	$1 * 2^3 = 8$
10	10	$0 * 2^4$
10	1	$0 * 2^5$
	0	$1 * 2^6 = 64$

$(77)_{10}$

"30F4A"

$(200522)_{10}$

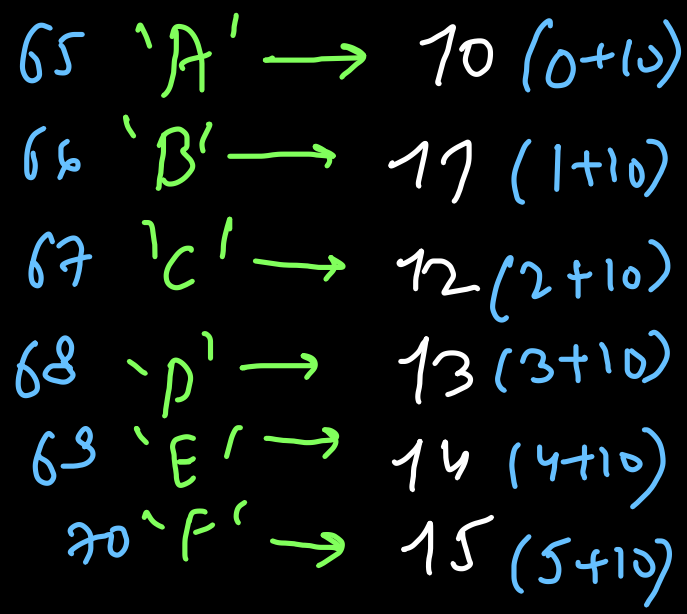
'A' ⟶ $10 * 16^0 = 10$

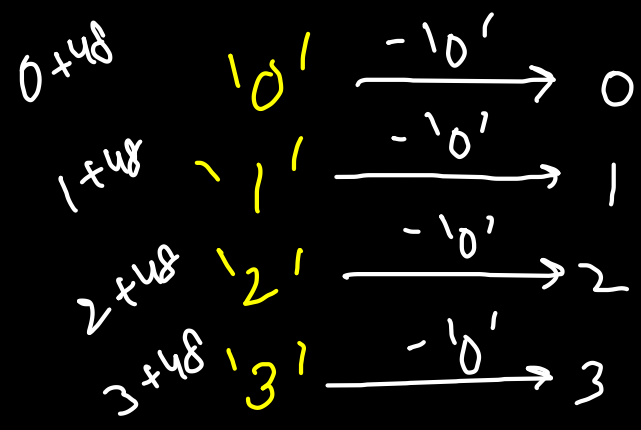
'4' ⟶ $4 * 16^1 = 64$

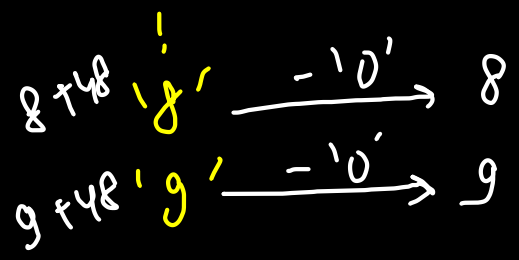
'F' ⟶ $15 * 16^2 = 3840$

'0' ⟶ $0 * 16^3$

'3' ⟶ $3 * 16^4 = 196608$








```java
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	Short	int	long
	↓	↓	↓	↓
size	1 byte = 8 bits	2 byte = 16 bits	4 byte = 32 bits	8 byte = 64 bits
range	-2^7 to 2^7-1	-2^{15} to $2^{15}-1$	-2^{31} to $2^{31}-1$	-2^{63} to $2^{63}-1$

int: -2^{31} to $2^{31}-1$ → -2×10^9 to 2×10^9

long: -2^{63} to $2^{63}-1$ → -10^{16} to 10^{16}

4 bit number system

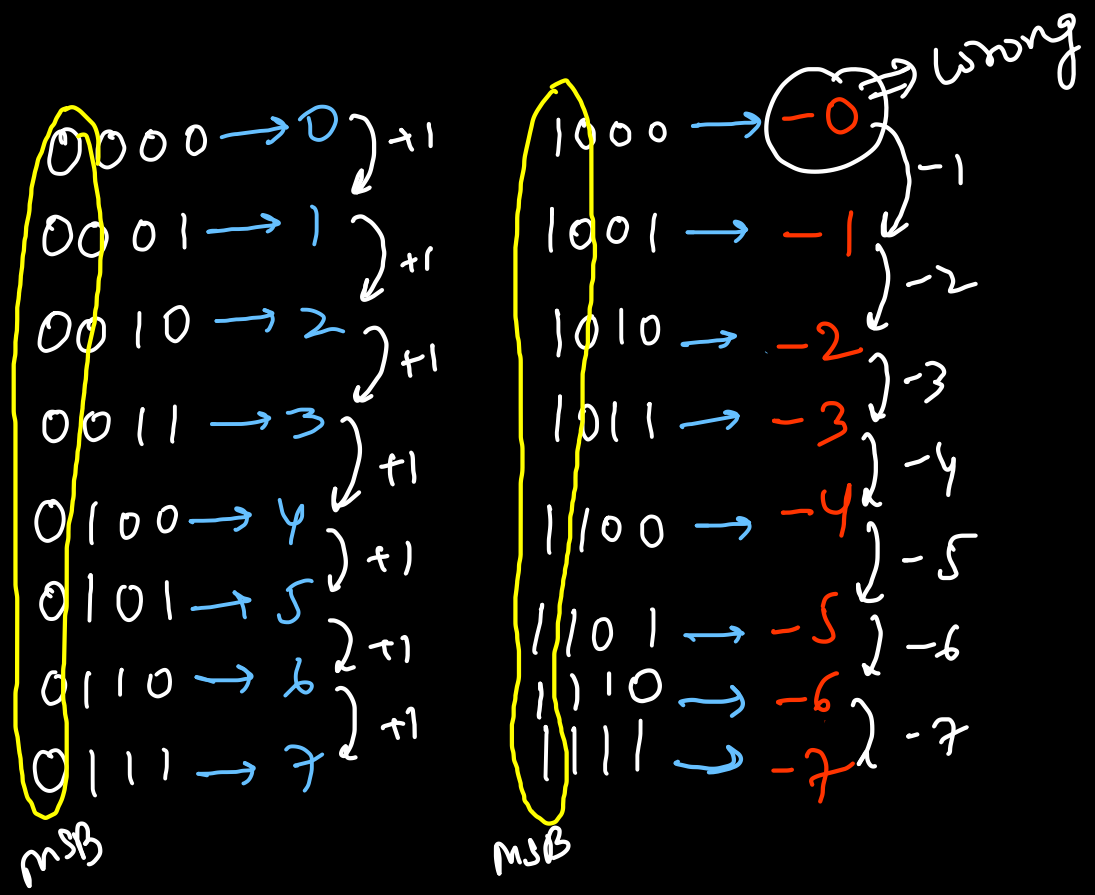
$\Rightarrow$ -2^3 to $+2^3 - 1$

-8 to $+7$

0000 → 0	1000 → 8
0001 → 1	1001 → 9
0010 → 2	1010 → 10
0011 → 3	1011 → 11
0100 → 4	1100 → 12
0101 → 5	1101 → 13
0110 → 6	1111 → 14
0111 → 7	

unsigned number system (ASCII)

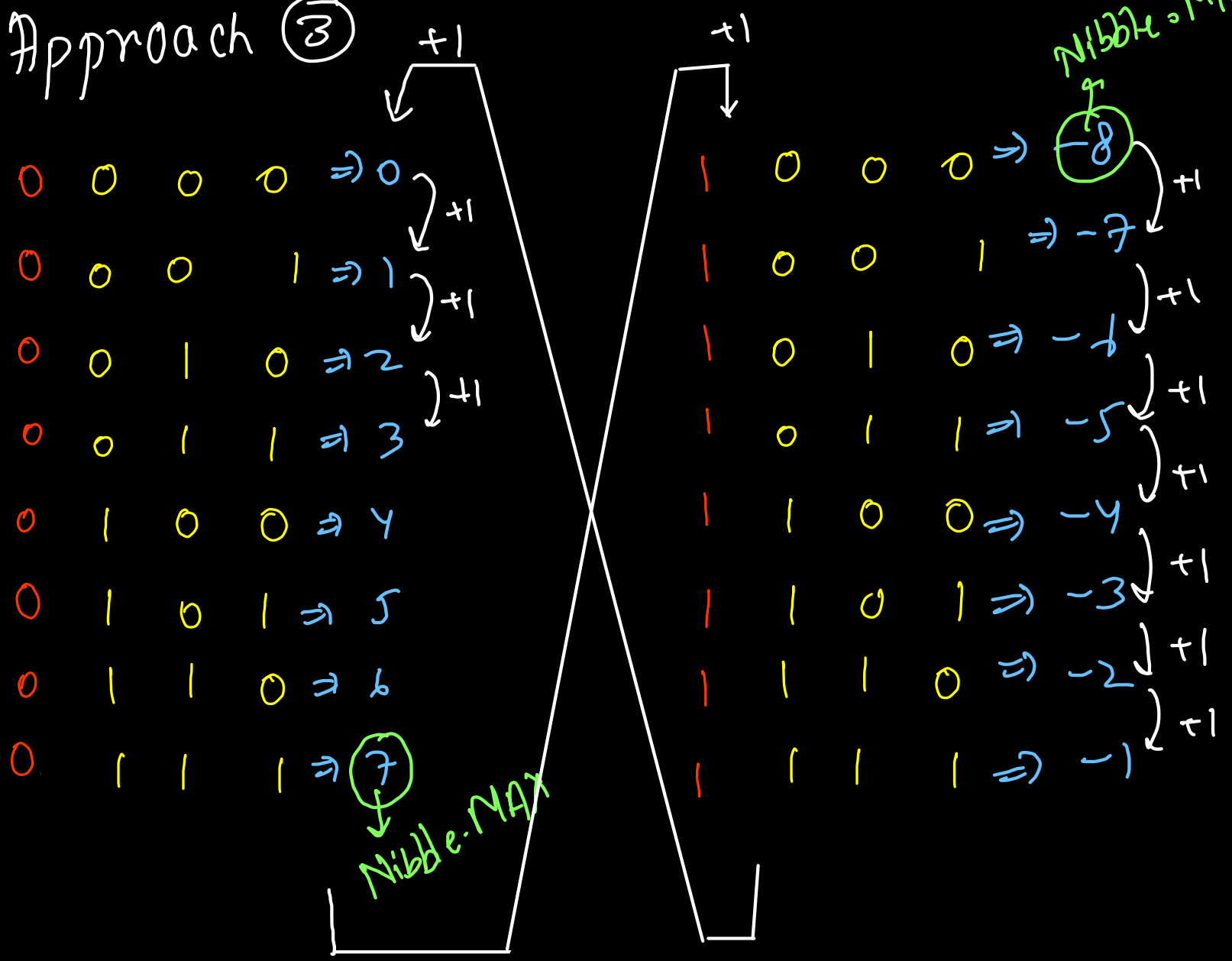
Approach 1



number system

Approach 2

Approach ③



MSB (leftmost bit) == 0

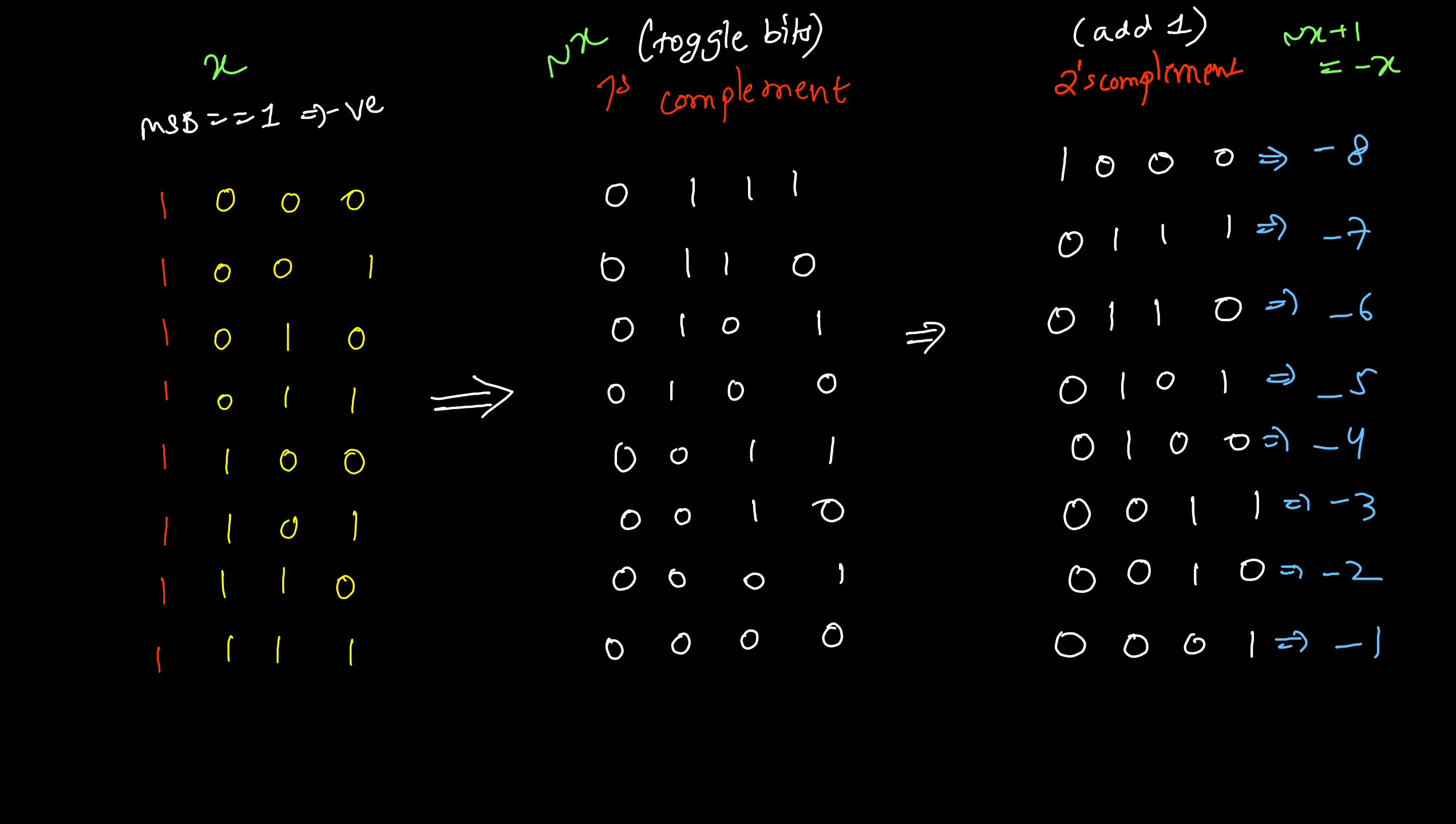
⇒ Number is +ve

⇒ directly Convert

MSB (leftmost bit) == 1

⇒ Number is -ve

decimal = 2's complement + Convert into decimal


x
MSB == 1 => -ve
1 0 0 0
1 0 0 1
1 0 1 0
1 0 1 1
1 1 0 0
1 1 0 1
1 1 1 0
1 1 1 1
~x (toggle bits)
1's 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
(add 1)
2's complement
~x+1 = -x
1 0 0 0 => -8
0 1 1 1 => -7
0 1 1 0 => -6
0 1 0 1 => -5
0 1 0 0 => -4
0 0 1 1 => -3
0 0 1 0 => -2
0 0 0 1 => -1

Overflow

$\infty + 1 = -\infty$

Integer.MAX_VALUE + 1 = Integer.MIN_VALUE

Integer.MIN_VALUE − 1 = Integer.MAX_VALUE

$-\infty - 1 = +\infty$

Number System is circular

Decimal to Binary (4 bit No System)

0 0 0 0	⇒ 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

1 0 0 0	⇒ -8
1 0 0 1	⇒ -7
1 0 1 0	⇒ -6
1 0 1 1	⇒ -5
1 1 0 0	⇒ -4
1 1 0 1	⇒ -3
1 1 1 0	⇒ -2
1 1 1 1	⇒ -1

Decimal ⇒ +ve

1. Convert to Binary
2. Fit into Bits.

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

0100

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

0001

leading zeros

nibble ans = 20

$(20)_{10} = (??)_2$

~~0~~ 1 0 1

sys(ans) ⇒ 5

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

1 1 1 1

⇒ -1

drop leading bits

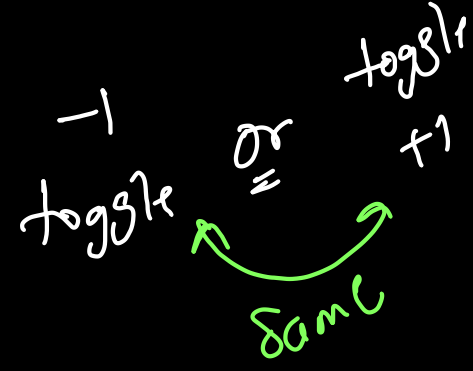
Decimal ⇒ -ve

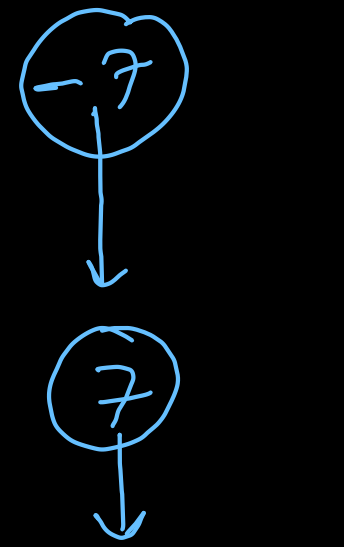
Binary ⇒ ?

① Drop -ve sign

② Convert into binary (4 bits)

③ Find its' 2's complement





0 1 1 1

↓ toggle

1 0 0 0

↓ +1

1 0 0 1

Bitwise operators

① AND

0 & 0 = 0

0 & 1 = 0

1 & 0 = 0

1 & 1 = 1

5 & 6 = ④

```
 0101
&0110
------
 0100
------
```

② OR

0 | 0 = 0

1 | 0 = 1

0 | 1 = 1

1 | 1 = 1

5 | 6 = ⑦

```
 0101
 0110
------
 0111
------
```

③ XOR

0 ^ 0 = 0

0 ^ 1 = 1

1 ^ 0 = 1

1 ^ 1 = 0

5 ^ 6 = ③

```
 0101
 0110
------
 0011
------
```

④ NOT

~0 = 1

~1 = 0

!5

```
 0101
------
 1010
------
```

2's comp

1

0101

+1

0110

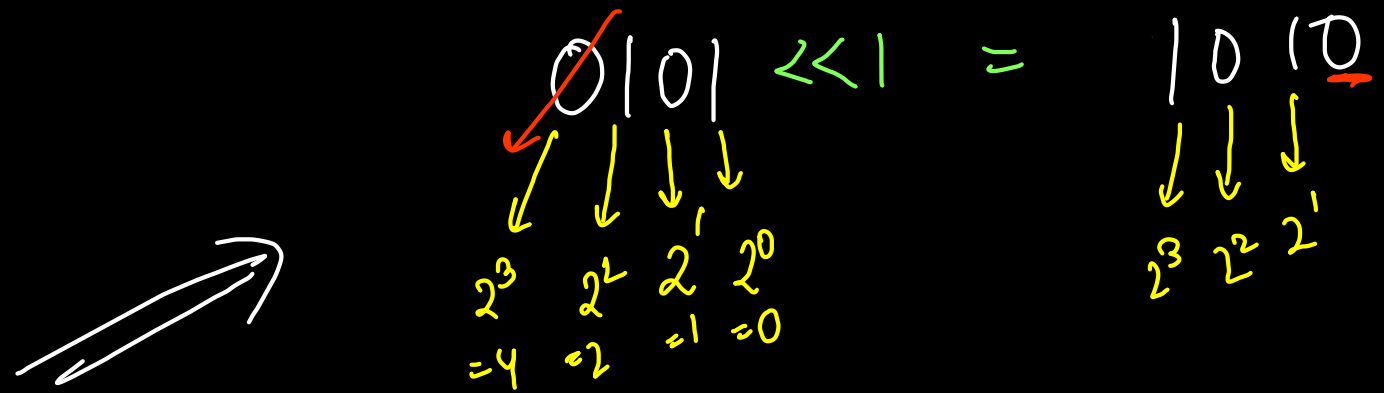
4bit system: -6

```java
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 <<
left shift operator

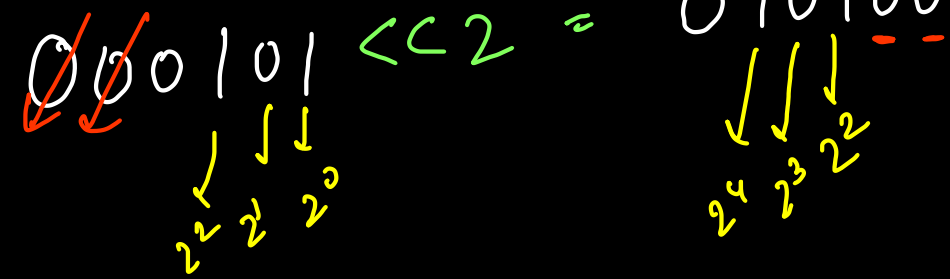
$5 << 1 == 5 * 2^1$

$0101 << 1 = 1010$
$= +10$



$5 << 2 = 5 * 2^2$

$00000101 << 2 = 00010100$
$= +20$



$a << x == a * 2^x$

$a << x$ → $O(1)$

$a * 2^x$ → $O(\log x)$

⑥ >> (Signed) right shift operators

$a >> x = a / 2^x$

0 1 0 1 >> 1

0 0 1 0

5 >> 1 == 2

0 0 1 1 0 1 0 >> 2

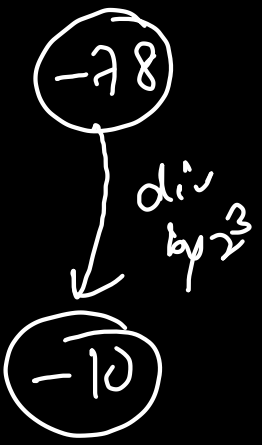
0 0 0 0 1 1 0

26 >> 2 = 6

8bit 1 0 1 1 0 0 1 0 >> 3

empty space by 1

1 1 1 1 0 1 1 0



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

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

③ Unsigned right shift (>>>) → Empty spaces will be always 0's

eg1 0101 >>> 1 = 5/2 = 2

eg2 10 1100 10 >>> 3

0 0 0 10110 ⇒ 22

Integer Class Bitwise functions

number of set bits

bitCount(int i)
Returns the number of one-bits in the two's complement binary representation of the specified `int` value.

O(32)

leftmost set bit (LSB)

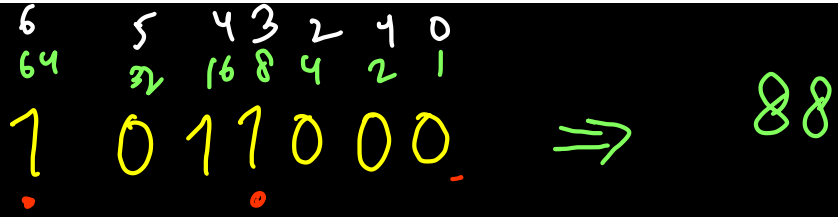
highestOneBit(int i)
Returns an `int` value with at most a single one-bit, in the position of the highest-order ("leftmost") one-bit in the specified `int` value.

O(32)

Rightmost set bit (RSB)

lowestOneBit(int i)
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.

O(32)




```
System.out.println(Integer.toBinaryString(88));
System.out.println(Integer.bitCount(88));
System.out.println(Integer.highestOneBit(88));
System.out.println(Integer.lowestOneBit(88));
```

1011000

3

64 placevalue

8 placevalue

`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

1011000 & 1 = 0
00000001

101100 & 1 = 0
000001

10110 & 1 = 0
00001

1011 & 1 = 1
0001

101

10

1

0

101
& 001
001

110
& 1
000

odd
5 % 2 = 1
101 & 1 = 1

famous
odd
even

even
6 % 2 = 0
110 & 1 = 0

a % 2 == a & 1

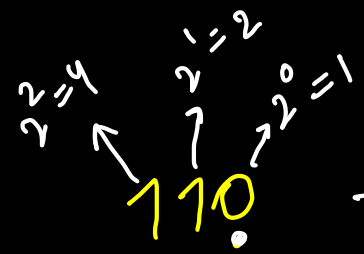
a / 2 == a >> 1

```java
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); ⇒ n%2
        N = N >> 1; ⇒ n = n/2
    }

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

a % 2

6 % 2 == 0 (even)

5 % 2 == 1 (odd)



$110_2 = 1 \times 2^2 + 1 \times 2^1 = 4+2 = 6$

$101_2 = 1 \times 2^2 + 1 \times 2^0 = 4+1 = 5$

if last bit is set, number is odd

else number is even

even	odd
110	101
& 001	& 001
000	001

a % 2 == a & 1

Bitmasking

$(88)_{10} = (1011000)_2$ (bits 6 5 4 3 2 1 0)

Q1) ith bit ⇒ check → set (1) / unset (0)

3rd bit → ✓

2nd bit → ✗

Q2) ith bit ⇒ set

1 → 1

0 → 1

3rd bit → 1011000

2nd bit → 1011100

Q3) ith bit ⇒ unset

0 → 0

1 → 0

3rd bit → 1010000

2nd bit → 1011000

Q4) ith bit ⇒ toggle

0 → 1

1 → 0

3rd bit → 1010000

2nd bit → 1011100

AND → powerful bit → 0

0 & 0 = 0

1 & 0 = 0

? & 0 = 0

unset

XOR

same bits → 0

diff bits → 1

toggle

OR → powerful bit → 1

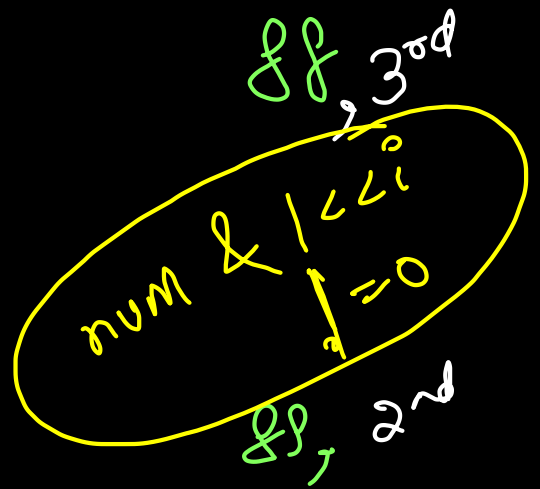
0 | 1 = 1

1 | 1 = 1

? | 1 = 1

set

Q1) ith bit ⇒ check → set (1) / unset (0)



6 5 4 3 2 1 0

num : 1 0 1 1 0 0 0

mask : 0 0 0 1 0 0 0

and : 0 0 0 1 0 0 0 ≠0

num : 1 0 1 1 0 0 0

mask : 0 0 0 0 1 0 0

and : 0 0 0 0 0 0 0 =0

$1 = 2^0 = 1<<0$

$10 = 2^1 = 1<<1$

$100 = 2^2 = 1<<2$

$1000 = 2^3 = 1<<3$

mask (only 3rd bit is set)

= 1<<3

ith bit is set mask

= 1<<i

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

check i^{th} bit using OR

$10\ 1\overset{\cdot}{1}000$

or $1111\ 0111$

$\sim(1111\ 111) = 0$

$10\ 10\ 000$

or $1111\ 0111$

$\sim(1110\ 111) \neq 0$

Q2) ith bit ⇒ set

1 → 1
0 → 1

ans = (1<<i) | num

3rd

1 0 1 1 1 0 0 0

mask 0 0 0 0 1 0 0 0

or 1 0 1 1 1 0 0 0

3rd

1 0 1 0 0 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) ith bit ⇒ unset

0 → 0
1 → 0

10 11000
mask 11 10111
and 10 10000

mask = ~(1<<i);
return N & mask;

10 10000
mask 11 10111
and 10 10000

```
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) ith bit ⇒ toggle 0 → 1, 1 → 0

mask = (1 << i);

return N ^ mask;

Eg1

```
  1 0 1 1 1 0 0 0
      ↕ same
  0 0 0 0 1 0 0 0   mask
-----------------
  1 0 1 0 0 0 0 0   xor
-----------------
```

Eg2

```
  1 0 1 0 0 0 0 0
      ↕ diff
  0 0 0 0 1 0 0 0   mask
-----------------
  1 0 1 1 0 0 0 0   xor
-----------------
```

```java
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..
(1-based indexing)

Tip#1 Check odd/even using bitwise operators

(1)

```java
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 ✗

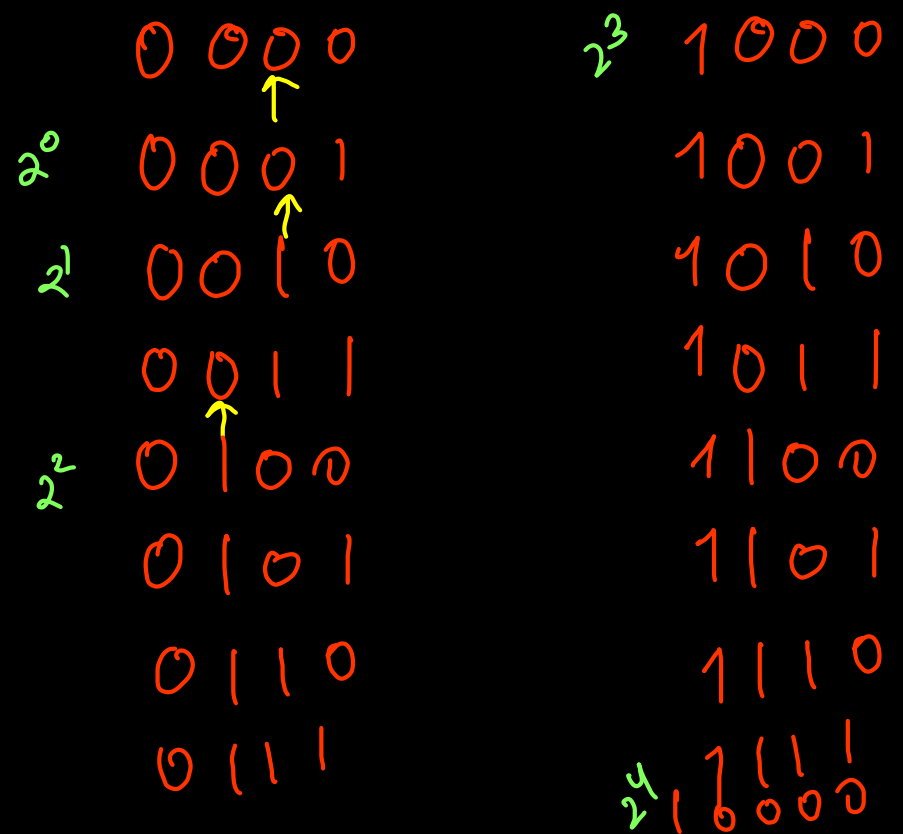
1 10 100 1000 1000 10000

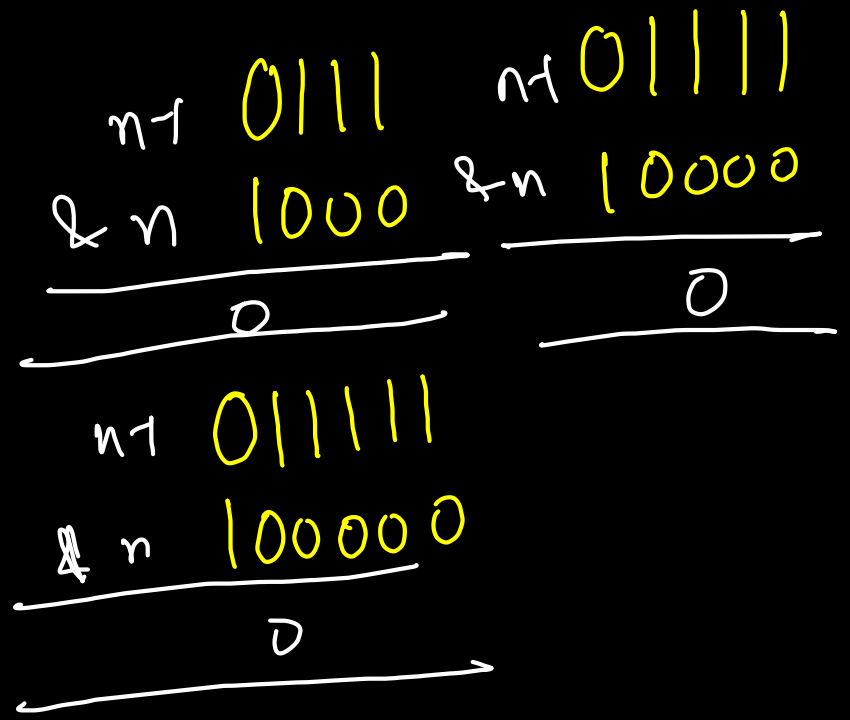
```java
class Solution {
    public boolean isPowerOfTwo(int n) {
        if(n <= 0) return false;
        return (Integer.bitCount(n) == 1);
    }
}
```

O(32) solution

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

O(1)





Tip 8 :→ Swap 2 No's w/o using temporary variable and arithmetic operators

$a = 5$ $\quad$ $b = 6$

0101 $\quad \rightarrow \quad$ 0110

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

① $a \wedge b = 0101 \wedge 0110 = 0011 = 3$

$a = a \wedge b$ $\quad$ $\boxed{a = 3}$

② $b = a \wedge b = 3 \wedge 6 = 0011 \wedge 0110 = 0101 = 5$

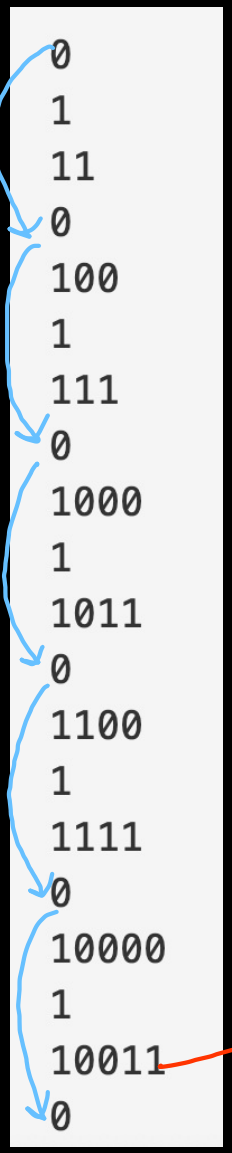
$\boxed{b = 5}$

③ $a = a \wedge b = 3 \wedge 5$

$= 0011 \wedge 0101 = 0110 = 6$

$\boxed{a = 6}$

Tip # 4 Xor all numbers from 1 to N.



1

$1 \wedge 2$

$1 \wedge 2 \wedge 3$

$1 \wedge 2 \wedge 3 \wedge 4$

$- - - - - \wedge 5$

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

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

$1 \wedge 2 \wedge - - - -$ multiple of 4 $= 0$

$1 \wedge - - \wedge 19 = \boxed{19 \wedge 18 \wedge 17}$

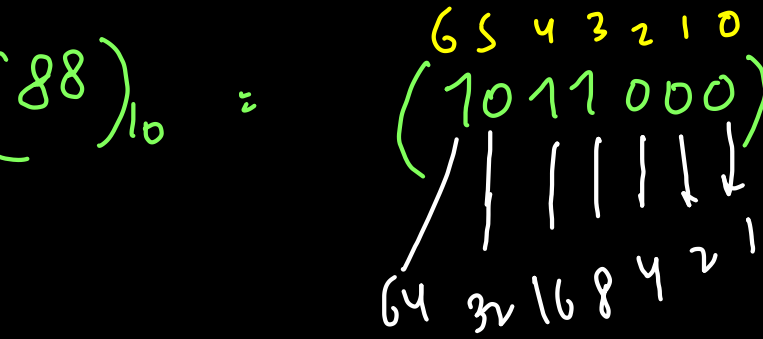
```
int computeXOR(int n)
{

  // 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

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

$(88)_{10}$ = (1011000) ⇒ LSB = $(1000)_2$



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

Brute force :→ Loop from 0th bit to 31st bit

```
if(n & (1<<i) ≠0)
    return i ;
```

```java
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));
}
```

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

$\approx O(1)$

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

88 = 0 0 1 0 1 1 0 0 0

2^6 2^5 2^4 2^3 2^2 2^1 2^0

64 32 16 8 4 2 1

```
  1011000
& 0001000
---------
  0001000
```

rightmost set bit

$n = 1\ 0\ 1\ 1\ 0\ 0\ 0$

$-n = 0\ 1\ 0\ 1\ 0\ 0\ 0$

and $0\ 0\ 0\ 1\ 0\ 0\ 0$

RSB = n & -n;

$n = 88$

1s complement = ~n

= 0100111

2scomplement

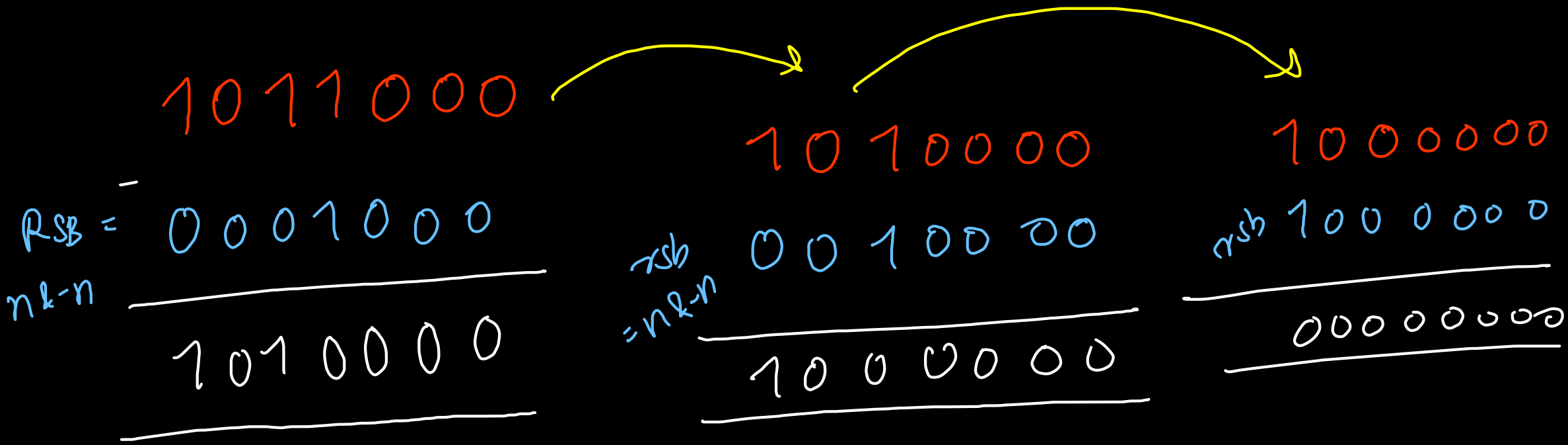
= ~n + 1

```
   111
0100111
    + 1
-------
0101000
 ↓ ↓
 32 8
```

```java
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);
}
```

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

$\rightarrow O(1)$


10111000
RSB = 00010000
nk-n
10110000
10110000
rsb 00100000
=nk-n
10000000
10000000
rsb 10000000
00000000

```java
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. interviews → 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
```

32 16 8 4 2 1

101001 = 41

10100 = 20

setBits(41) = 1 + setBits(20)

101000 = 40

setBits(40) = setBits(20)

10100 = 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

1 0 1 1 1 0 0 0 (88)

0 1 1 0 0 1 0 (50)

Xor 1 1 0 1 0 1 0 ⇒ countSetBits ⇒ Answer ⇒ 4

xor gives set bits wherever a & b are different

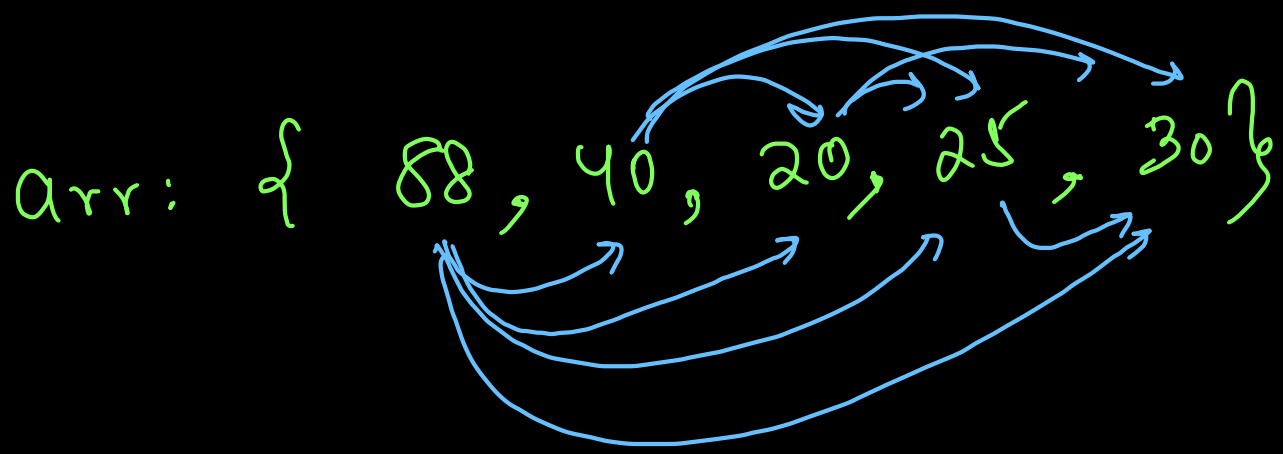
```
class Solution {
    public int hammingDistance(int x, int y) {
        return Integer.bitCount(x ^ y);
    }
}
```

↳ O(logN) worst case

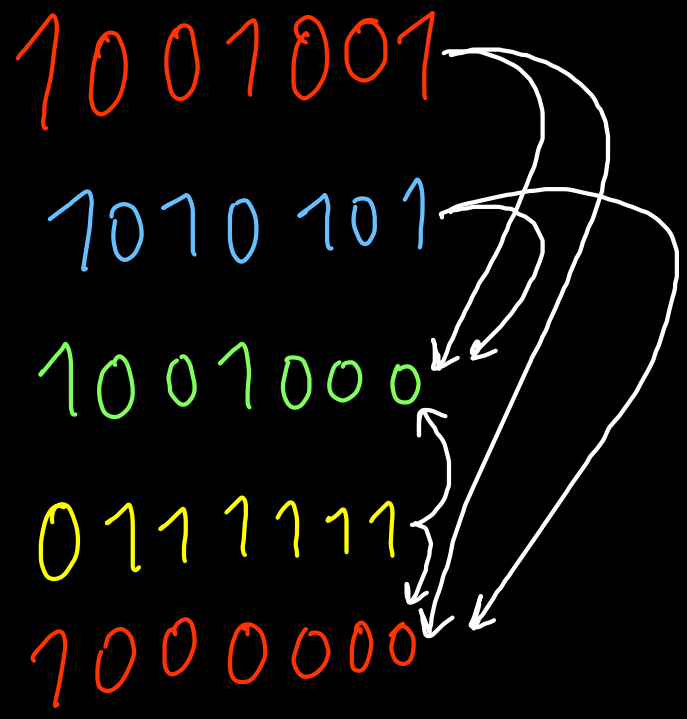
477) Total Hamming distance

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`.




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

return sum / 2;
```



0th bit 3 (1s) 2 (0s) 3*2

1st bit 1 (1s) 4 (0s) 1*4

31st bit

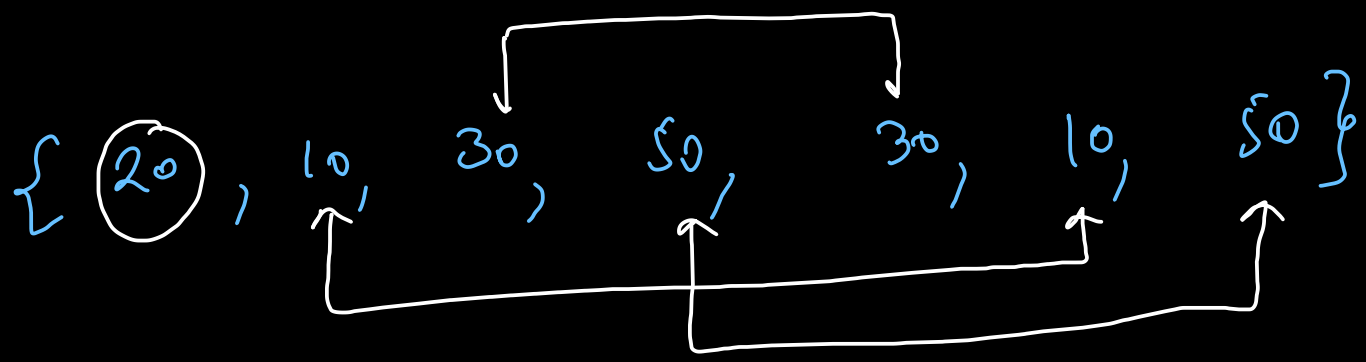
Time ⇒ O(nlogn)

Space ⇒ O(1)

```java
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



$20 \wedge 10 \wedge 30 \wedge 50$

$\wedge 30 \wedge 10 \wedge 50$

xor properties

① $a \wedge b = b \wedge a$ {commutative}

② $a \wedge 0 = a$ {0 ⇒ identitive property}

③ $a \wedge a = 0$

④ $a \wedge b = c$

$\Rightarrow a \wedge c = b$

$\Rightarrow b \wedge c = a$

```java
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: 0 1 0 0 1 1 0

goal: 1 0 0 1 1 0 0

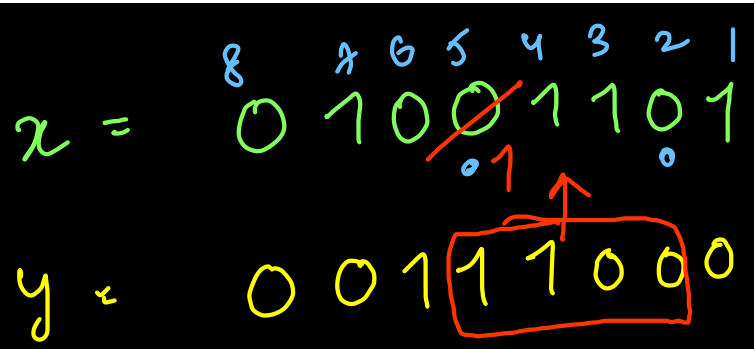
xor = 1 1 0 1 0 7 0

↳ set bits ⇒ 4

Copy Set Bits in Range

Given two numbers x and y, and a range [l, r] where 1 <= l, r <= 32.

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



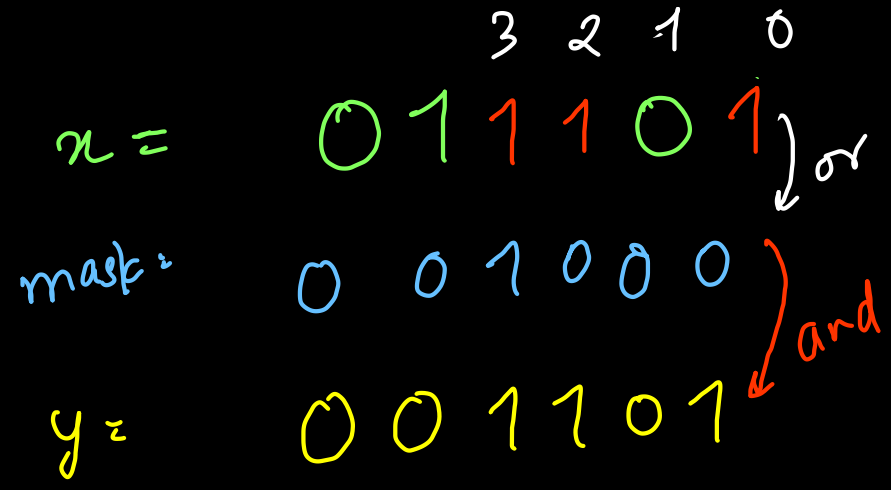
(1 << bit)

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;
}
```

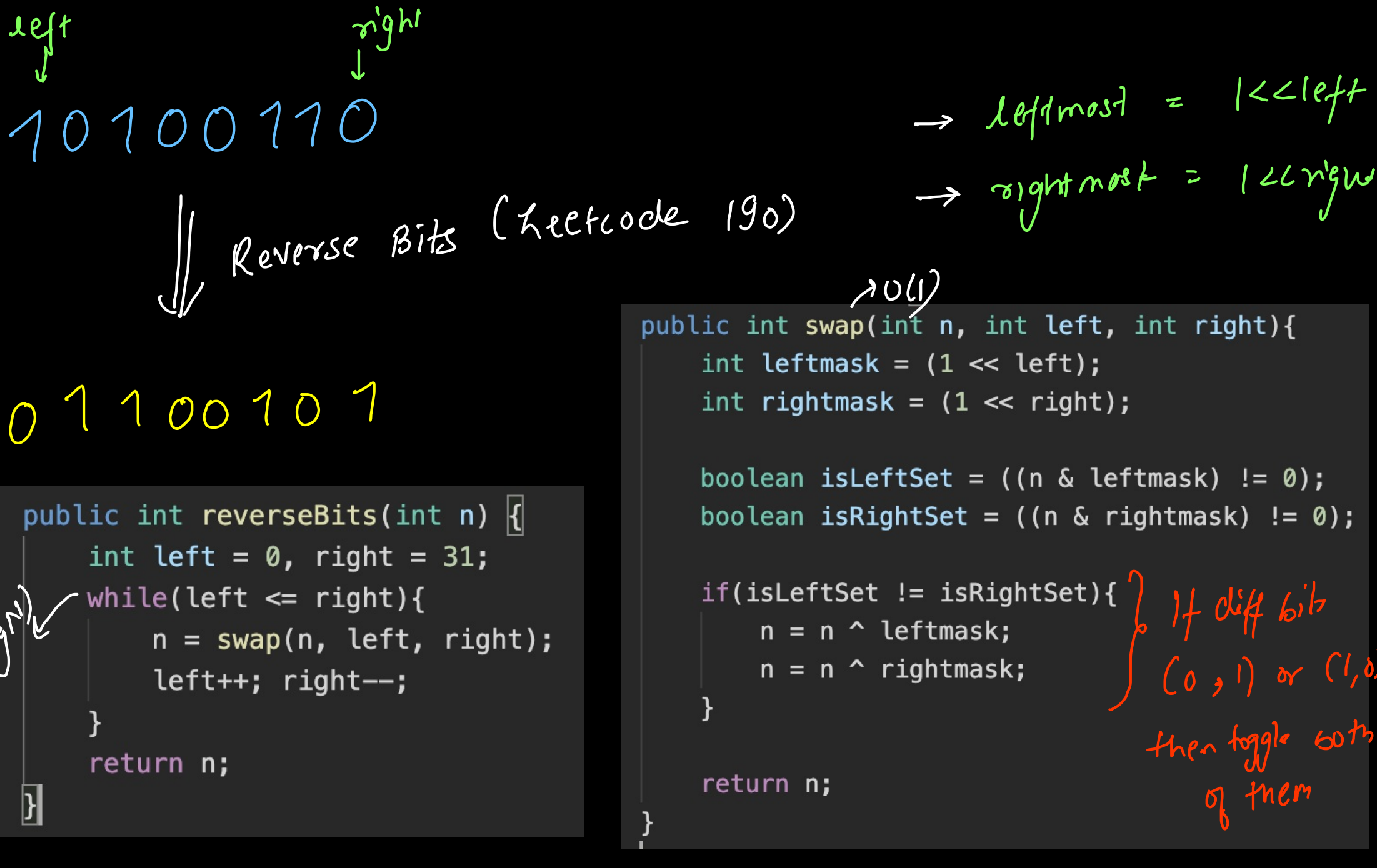
Time $\Rightarrow O(32)$

$= O(\log N)$



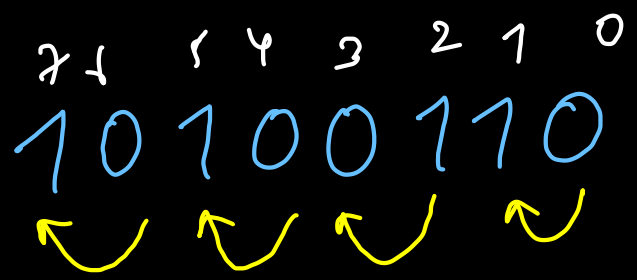
L = 1, R = 4

0 3




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

```java
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;
}
```

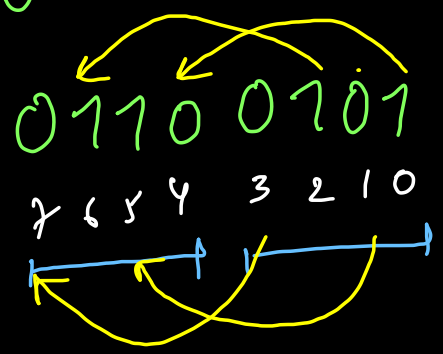


Swap Odd & Even Bits (GFG)

```java
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

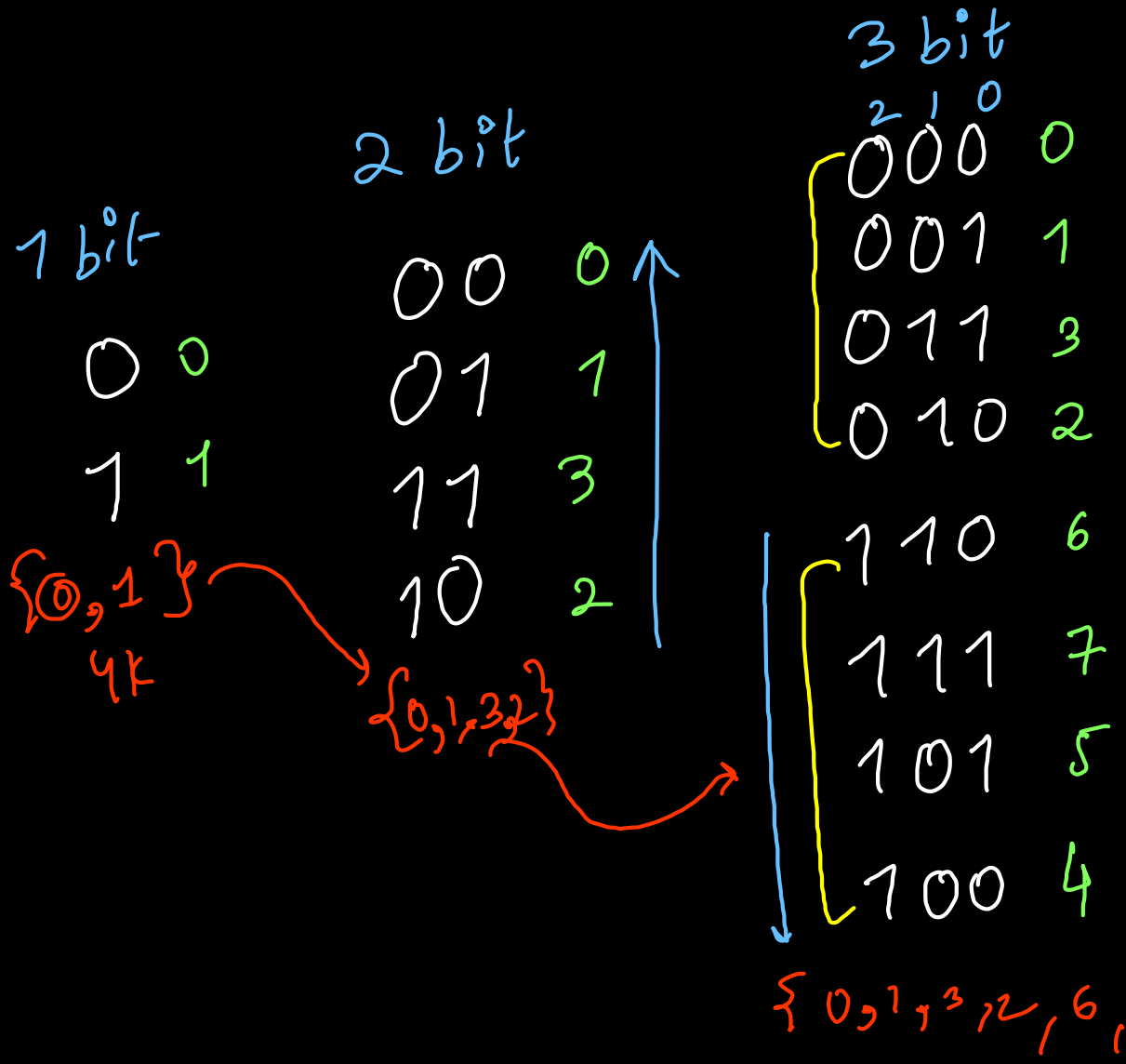
1, 5

2, 6

3, 7

```java
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




```java
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);

    for(int idx = res.size() - 1; idx >= 0; idx--){
        int ans = (res.get(idx)) | (1 << n);
        res.add(ans);
    }

    return res;
}
```

List ans = graycode(n-1);

Time Complexity
↳ $O(2^n)$

Space Complexity
↳ $O(n)$

```java
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

Xor Sum Problem

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

{ 10, 20, 30, 40 }

All pairs

↓

Sum ⇒ A[i] + A[j]

↓

Xor's

Constraint

$N = 10^5$

10+10	20+10	30+10	40+10
10+20	20+20	30+20	40+20
10+30	20+30	30+30	40+30
10+40	20+40	30+40	40+40

Time ⇒ O(N)

Space ⇒ O(1)

ans = (20 ∧ 40 ∧ 60 ∧ 80)

Single Number - II

Leetcode 137

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

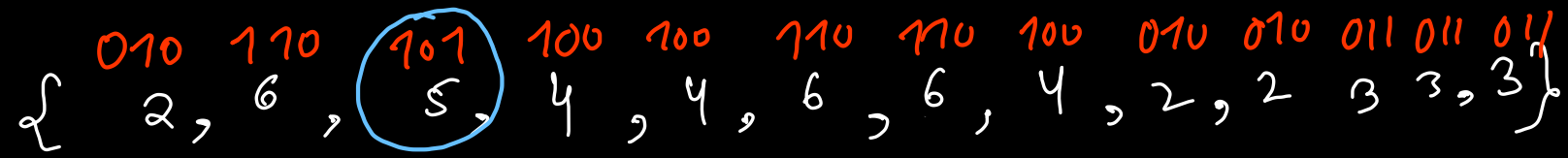
010	110	100	011
010	110	100	011
010	110	100	011

101

↓

unique no

1	0	1
7	9	4
3*2+1	3*3	3+1




```java
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;
}
```

count = 0 → 1+3 = 4

count = 0 → 1 → 2+7 = 9

Count = 7

res 1 0 1

0th 1st 0th

lc 260 Single Number → III

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

xor = 5 ^ 2 ^ 3 ^ 4 ^ 2 ^ 4 ^ 5 ^ 6 = 3 ^ 6

= 011 ^ 110

= 101 = 5

first unique

101 ^ 011 ^ 101 =

5 ^ 3 ^ 5

second

= (3 ^ 6) ^ (3)

= 6

```java
class Solution {
    public int[] singleNumber(int[] nums) {
        int xor = 0;
        for(int val: nums) xor = xor ^ val;

        int rsb = Integer.lowestOneBit(xor);
        int first = 0;

        for(int val: nums){
            if((val & rsb) != 0)
                first = first ^ val;
        }

        int second = xor ^ first;
        return new int[]{first, second};
    }
}
```

O(n)

O(n)

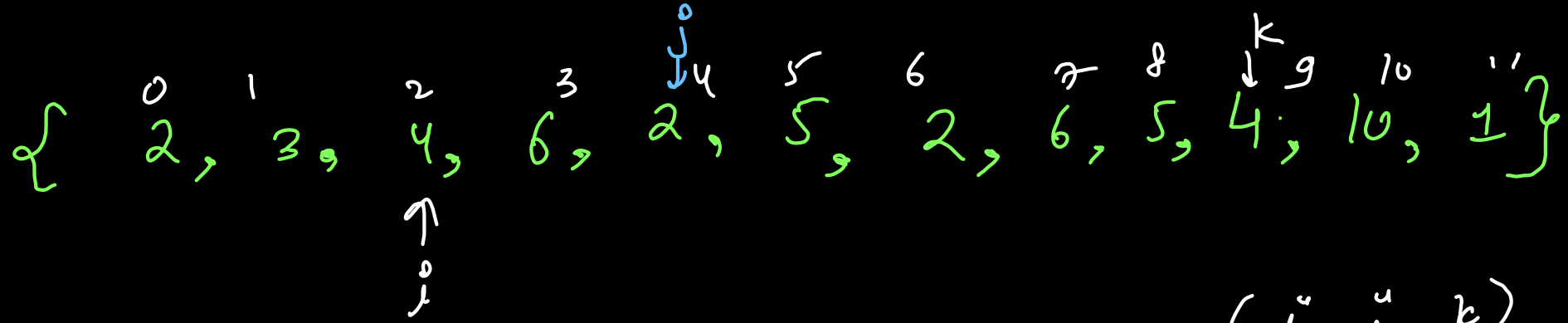
↳ n & -n

xor = 3 ^ 6

first = ~~5 ^ 3 ^ 5~~ = 3

second = ~~(5 ^ 6) ^ 5~~ = 6

3, 6



Given an array of integers `arr`.

We want to select three indices `i`, `j` and `k` where `(0 <= i < j <= k < arr.length)`.

Let's define `a` and `b` as follows:

- `a = arr[i] ^ arr[i + 1] ^ ... ^ arr[j - 1]`
- `b = arr[j] ^ arr[j + 1] ^ ... ^ arr[k]`

Note that ^ denotes the **bitwise-xor** operation.

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

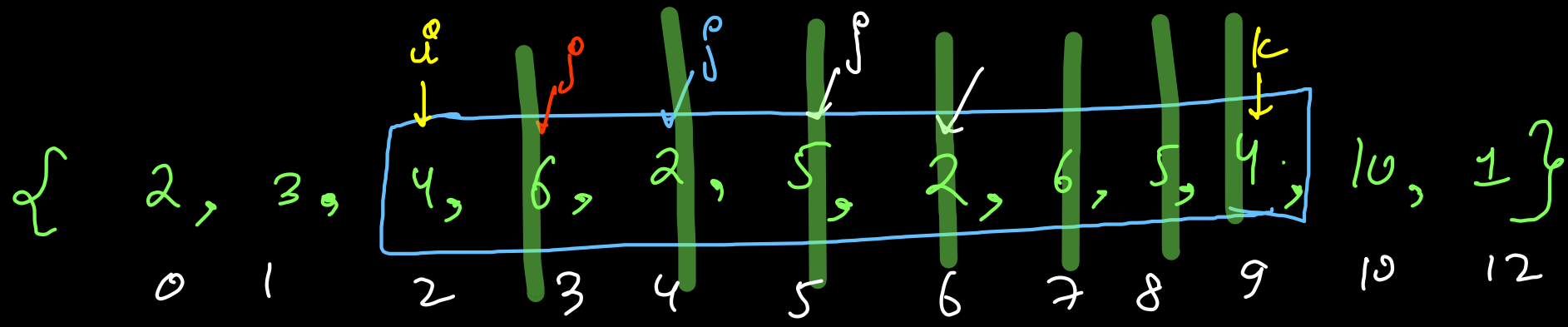
(i, j, k)

$i < j \leq k$

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

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

$(2, 4, 9)$



$i \le j \le k$

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

$4 \wedge 6 \wedge 2 \wedge 5 \wedge 2 \wedge 6 \wedge 5 \wedge 4 = 0$

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

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

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

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

Time ⇒ $O(n^2)$, Space ⇒ $O(1)$

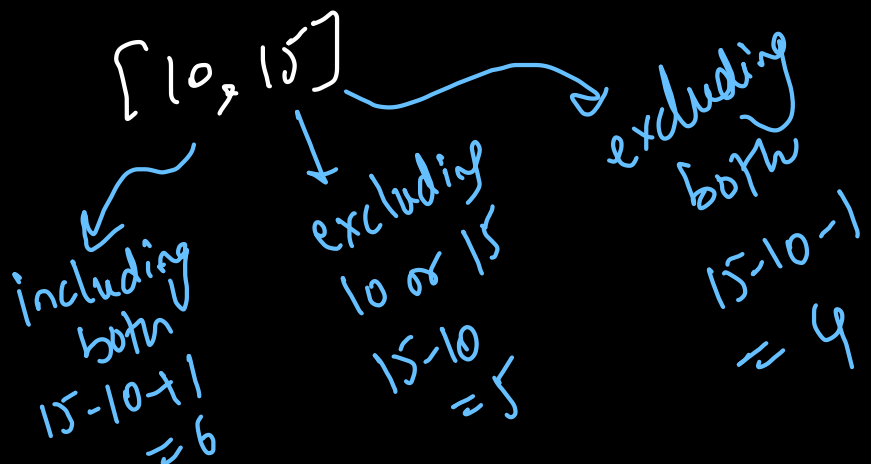
```java
public int countTriplets(int[] arr) {
    int ans = 0;
    for(int l = 0; l < arr.length; l++){
        int xor = 0;
        for(int r = l; r < arr.length; r++){
            xor = xor ^ arr[r];
            if(xor == 0) ans += (r - l);
        }
    }
    return ans;
}
```

Property:

If subarray has xor = 0

⇒ we can partition at any index to get $a = b$

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



Min Xor pair

6 20 5 1 14 18 9

0 000
0 001
0010
0011
0100
0101
0110
0111
1000

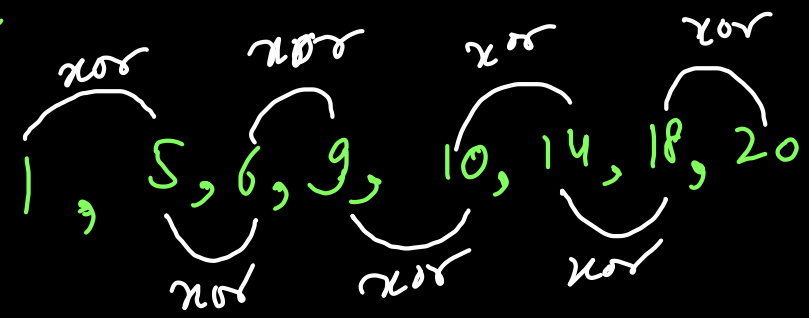
Brute force

→ $O(n^2)$ nested loops

TLE

10 ⇒ 1, 5, 6, 9, 10, 14, 18, 20

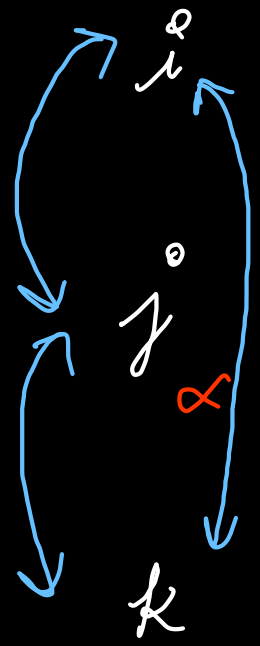
Sort




```java
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

$i < j < k$



0 1 0 0 1 1 0 1 0

0 1 0 0 1

0 1 0 1 0 0 1 1 0

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

min^n will be either $i \wedge j$ or $j \wedge k$

$i \wedge k = 0001$ _____

if in j ymn bit $\Rightarrow 0 \Rightarrow i \wedge j = 0000\ldots$

$\Rightarrow 1 \Rightarrow j \wedge k = 0000\ldots$

Max xor pair

Sample Input 0

```
10
15
```

Sample Output 0

```
7
```

$10 \oplus 10 = 0$
$10 \oplus 11 = 1$
$10 \oplus 12 = 6$
$10 \oplus 13 = 7$
$10 \oplus 14 = 4$
$10 \oplus 15 = 5$
$11 \oplus 11 = 0$
$11 \oplus 12 = 7$
$11 \oplus 13 = 6$
$11 \oplus 14 = 5$
$11 \oplus 15 = 4$
$12 \oplus 12 = 0$
$12 \oplus 13 = 1$
$12 \oplus 14 = 2$
$12 \oplus 15 = 3$
$13 \oplus 13 = 0$
$13 \oplus 14 = 3$
$13 \oplus 15 = 2$
$14 \oplus 14 = 0$
$14 \oplus 15 = 1$
$15 \oplus 15 = 0$

$a \wedge b = max$

$l \leq a, b \leq r$

10 = 1010

11 = 1011

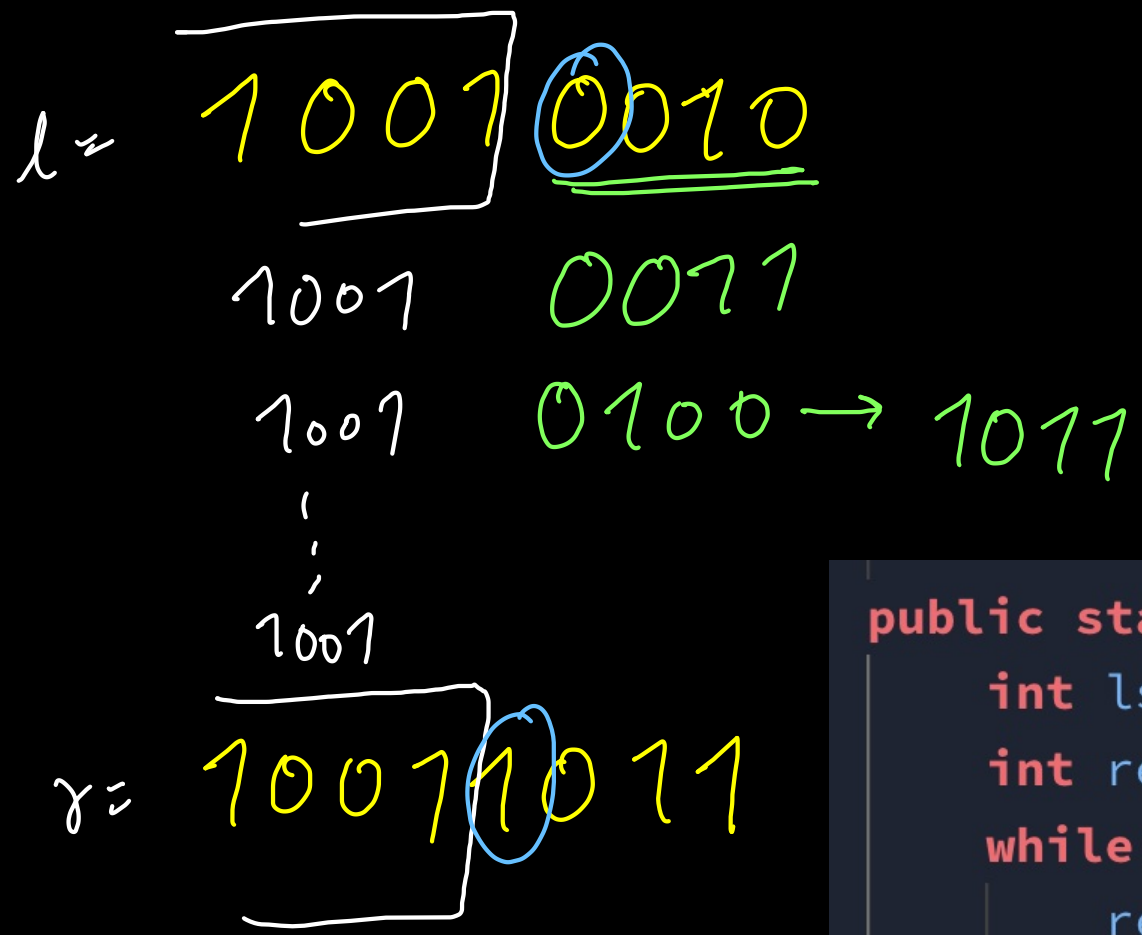
12 = 1100

13 = 1101

14 = 1110

15 = 1111

All bit after msb in a^b will be1

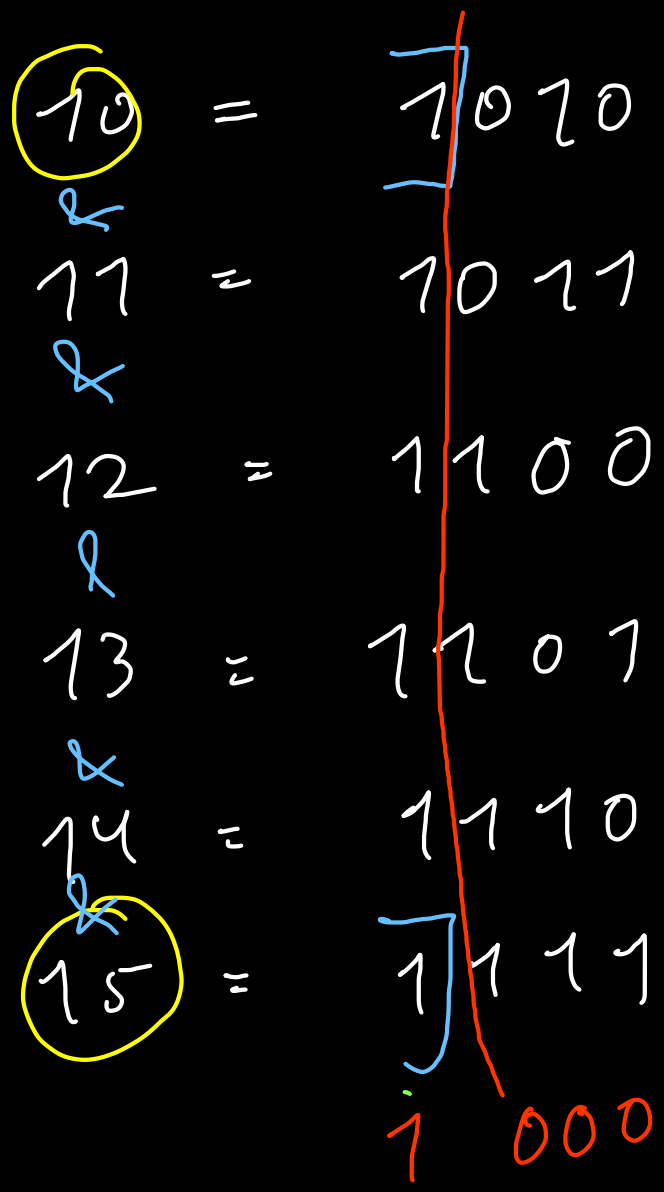


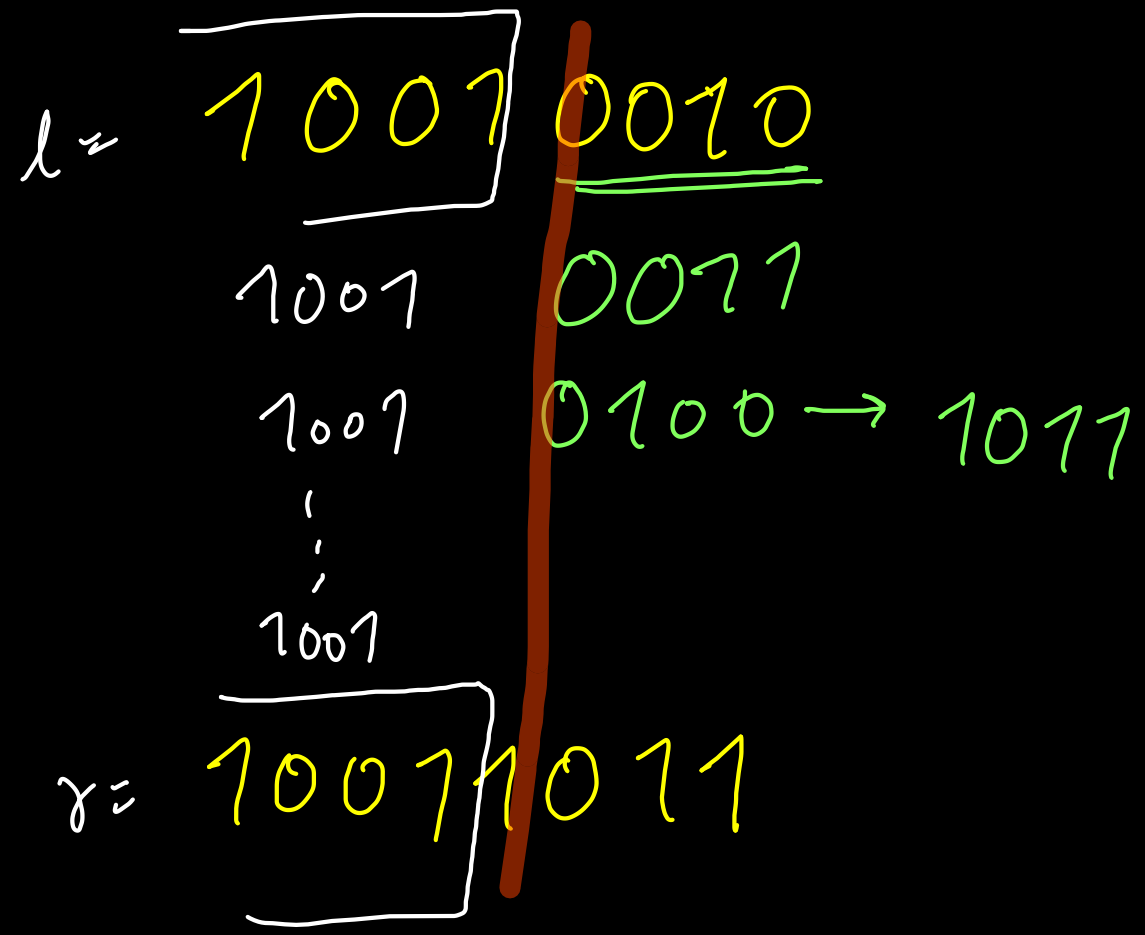
lsb = 1000

res = 0000
1111

```java
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 = 1010
& 11 = 1011
& 12 = 1100
& 13 = 1101
& 14 = 1110
& 15 = 1111
1 000


l = 1001 0010
1001 0011
1001 0100 → 1011
1001
r = 1001 1011

xy = 10010000

```java
public int rangeBitwiseAnd(int left, int right) {
    int lsb = Integer.highestOneBit(left ^ right);

    int res = left;
    while(lsb != 0){
        res = res & ~lsb;
        lsb = lsb >> 1;
    }

    return res;
}
```

Set all bits as 0 from lsb till 0th bit

1001]0000

1001]1010

l n r = 00001---

lsb = 0000010000

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

⇒ sum of odd bits
= = sum of even bits

In decimal multiples of 11 $\Leftrightarrow$ (Sum of odd digits − Sum of even digits) = multiple of 11

11, 22, ---- 99

110, 121, 132, 143, ... 209

```
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;
}
```

Power of 4

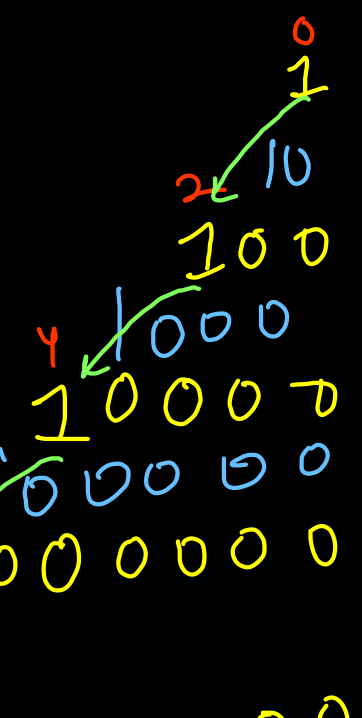
$4^0 = 1 = 1$

$4^1 = 4 = 100$

$4^2 = 16 = 10000$

$4^3 = 64 = 1000000$

$4^4 = 256 = 100000000$



setBits == 1 O(log n)

n & (n-1) == 0 O(1)

→ It is power of 2

→ set bit should be at even index.

```java
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	2

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

Time ⇒ O(1)

Space ⇒ O(1)

Game of XOR

0	1	2	3	4
10	20	30	40	50
↓	↓	↓	↓	↓
1*5	2*4	3*3	4*2	5*1

$occurences(i) = (i+1) * (n-i)$

- odd ⇒ arr[i] once
- even ⇒ arr[i] donot

res = 10 ^ 30 ^ 50

10

10 ^ 20

10 ^ 20 ^ 30

10 ^ 20 ^ 30 ^ 40

10 ^ 20 ^ 30 ^ 40 ^ 50

20

20 ^ 30

20 ^ 30 ^ 40

20 ^ 30 ^ 40 ^ 50

30

30 ^ 40

30 ^ 40 ^ 50

40

40 ^ 50

50

```java
class Solution {
    static int gameOfXor(int N , int[] A) {
        int res = 0;

        for(int idx = 0; idx < N; idx++){
            long occurences = (idx + 1l) * (N - idx);
            if((occurences & 1) == 1){
                res = res ^ A[idx];
            }
        }

        return res;
    }
}
```

Time ⇒ O(n)

Space ⇒ O(1)

↓ small optimizam

```java
class Solution {
    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]
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 and (10 ^ 15) = 5
```

Count Pairs with given Xor

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

$i, j \quad i < j$

search (hashset)

Approach 1) Brute force

nested loop

$\Rightarrow O(n^2)$ time

Property :→ $a \wedge b = c \Rightarrow a \wedge c = b$, $b \wedge c = a$

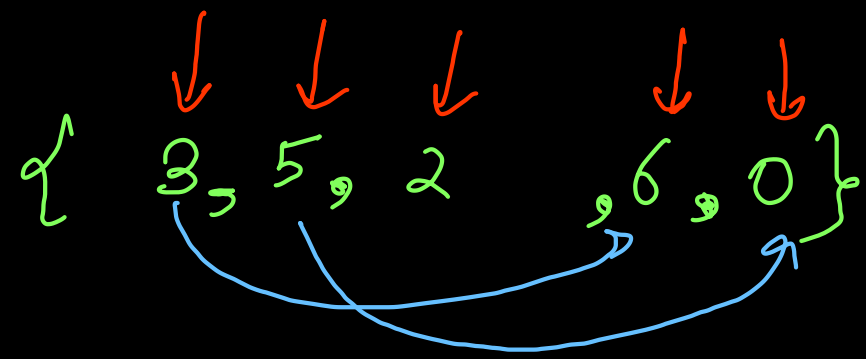
```java
public int solve(int[] A, int B) {
    HashSet<Integer> vis = new HashSet<>();
    for(int val: A) vis.add(val);

    int pairs = 0;
    for(int val: A){
        if(vis.contains(val ^ B) == true){
            pairs++;
        }
    }

    return pairs / 2;
}
```

O(n)

O(n)

Time = O(n)

Space = O(n)



B = 5

3 ∧ 5 = 6

3 ∧ 6 = 5

5 ∧ 5 = 0

5 ∧ 0 = 5

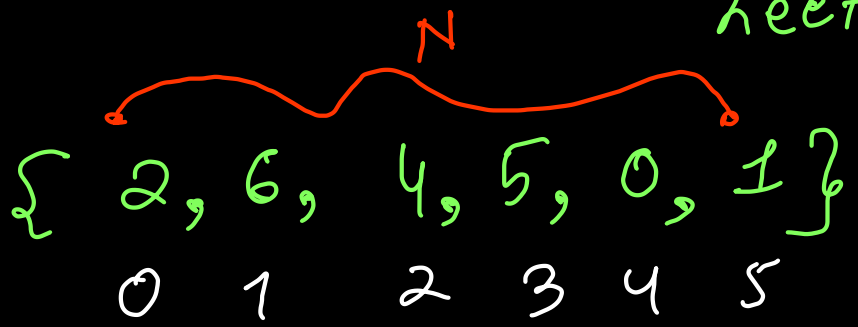
2 ∧ 5 = 7

6 ∧ 5 = 3

0 ∧ 5 = 5

pairs = 0 ~~1~~ ~~2~~ ~~3~~ 4

Leetcode 1310 Xor Subarray Queries



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

q

- [0,2] ⇒ 2 ^ 6 ^ 4
- [3,5] ⇒ 5 ^ 0 ^ 1
- [1,4] ⇒ 6 ^ 4 ^ 5 ^ 0
- [0,5] ⇒ 2 ^ 6 ^ 4 ^ 5 ^ 0 ^ 1

Brute force

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

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

9sec ⇒ TLE

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

$0\ \ 1\ \ 2\ \ 3\ \ 4\ \ 5$

$[0,2] \Rightarrow 2 \wedge 6 \wedge 4$

$[3,5] \Rightarrow 5 \wedge 0 \wedge 1$

$[1,4] \Rightarrow 6 \wedge 4 \wedge 5 \wedge 0$

$[0,5] \Rightarrow 2 \wedge 6 \wedge 4 \wedge 5 \wedge 0 \wedge 1$

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

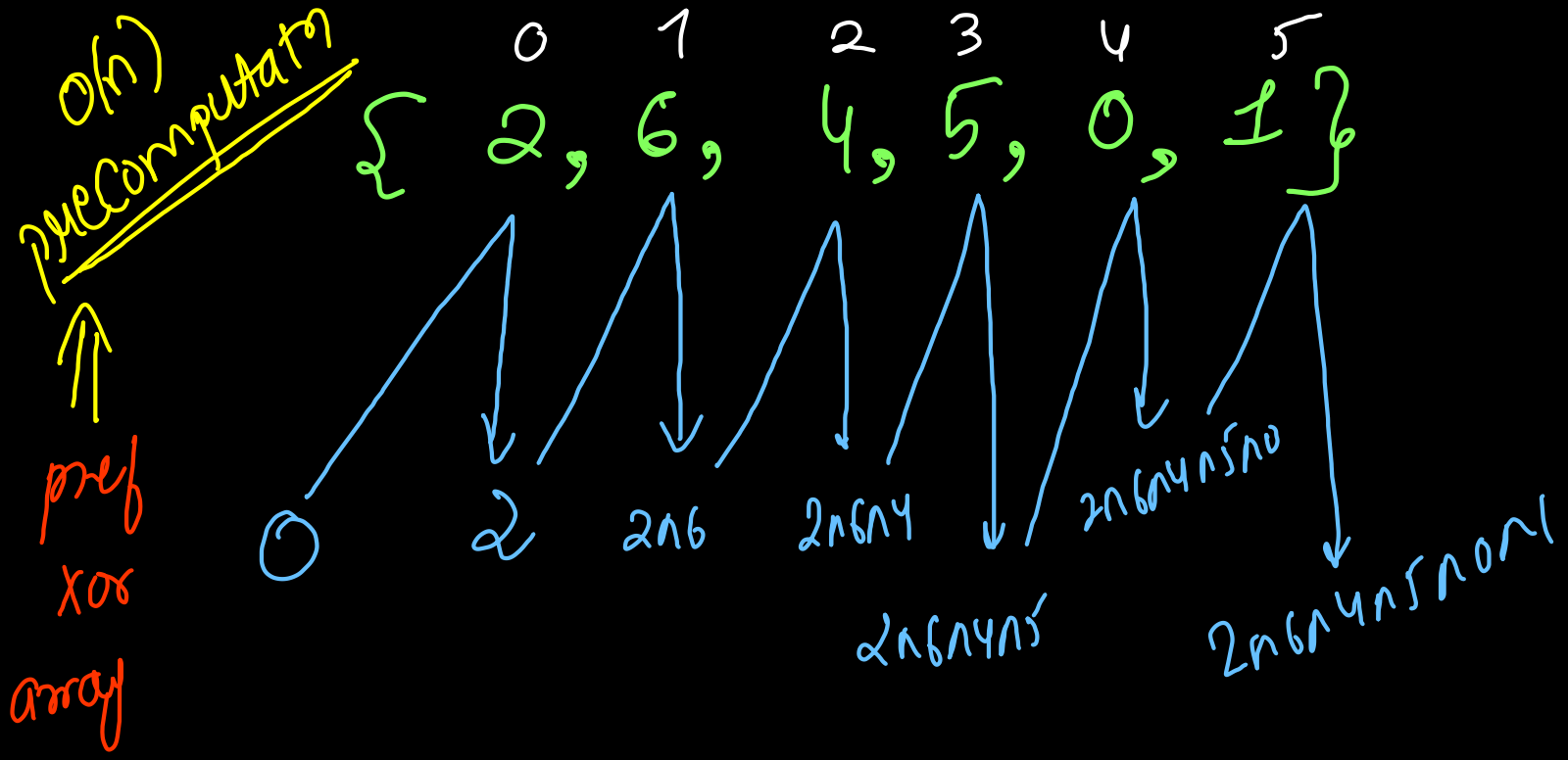
$0\ \ 1\ \ \boxed{2\ \ 3\ \ 4}\ \ 5$

$[\overset{l}{2}, \overset{r}{4}] = 4 \wedge 5 \wedge 0$

$= (4 \wedge 5 \wedge 0 \wedge 2 \wedge 6) \wedge (2 \wedge 6)$

$= pref[4] \wedge pref[2-1]$

$[l, r] = pref[r] \wedge pref[l-1]$



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

query[l, r] = pref[r] ^ pref[l-1] ⇒ each query can be solved in constant time.

$O(q)$

0	1	2	3	4	5
2	6	4	5	0	1

pref xor

$[0,0]$ $[0,1]$ $[0,2]$ $[0,3]$ $[0,4]$ $[0,5]$

$[l, r]$

$[0, 2] \Rightarrow$ prefix[2]

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

$x \wedge b \wedge y \wedge 5 \wedge 0 \wedge 1 \quad \wedge \quad x \wedge b \wedge y$

Approach 1)

```java
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;
}
```

O(n) Precomp

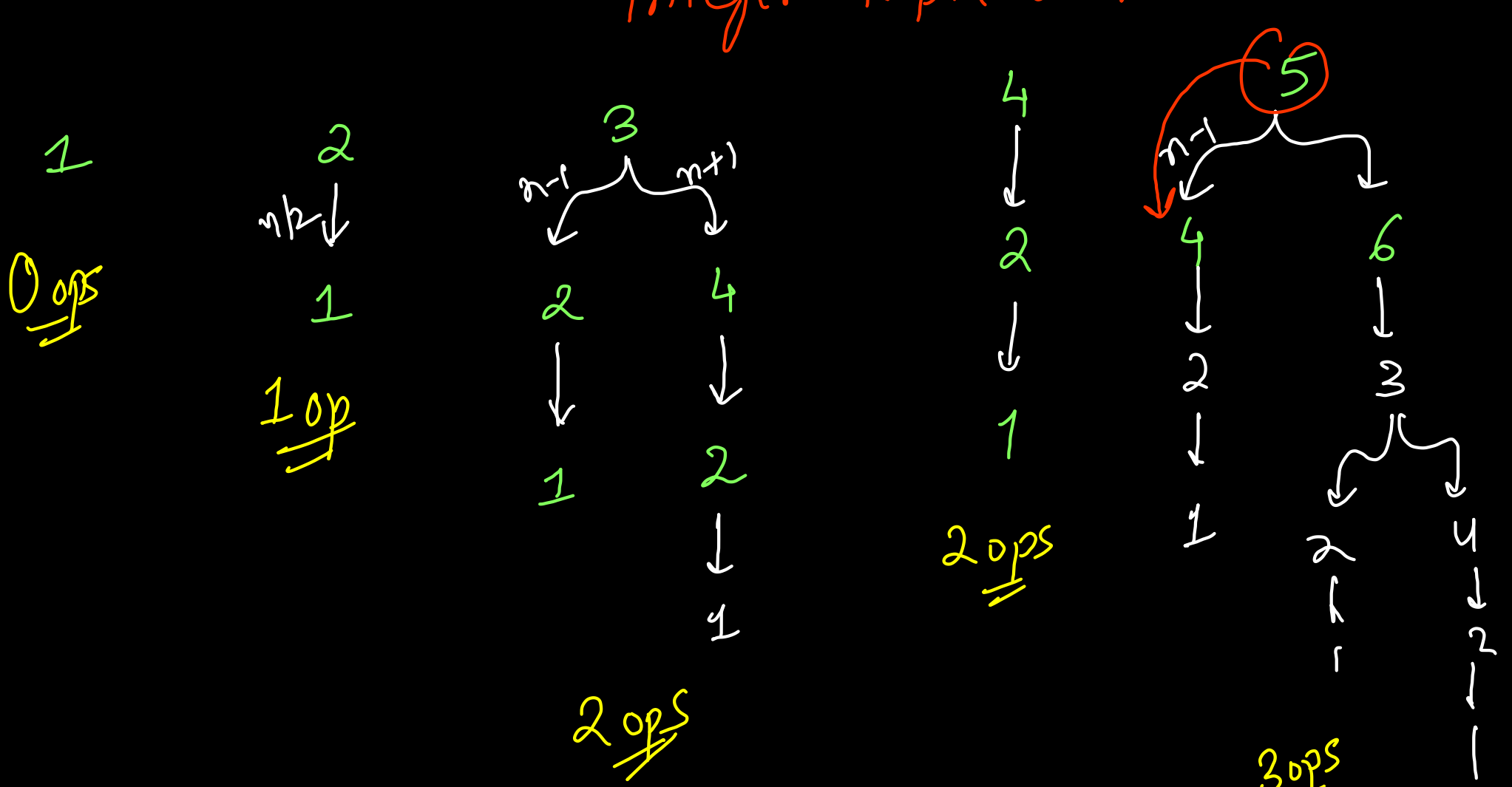
O(q) Work

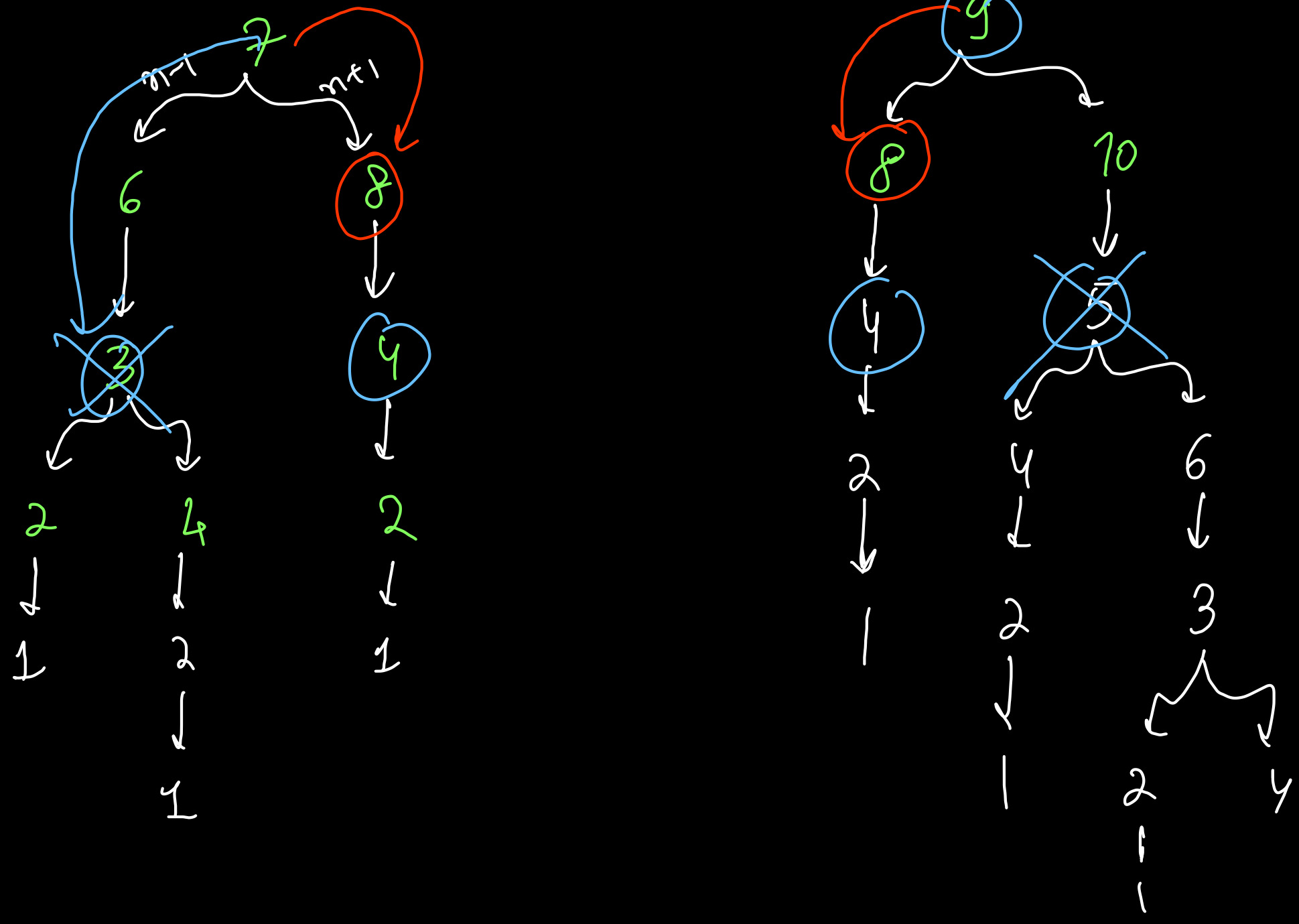
Total time ⇒ O(n+q) linear

Approach 2)

```java
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




7
n-1
n-2
6
8
3
4
2
4
2
1
2
1
1
9
8
10
4
5
2
1
4
6
1
2
3
1
2
4
1

```java
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)
        return 1 + integerReplacement(n - 1);
    else return 1 + integerReplacement(n + 1);
}
```

greedy choice

$O(\log_2 N)$ time

Power Set

Powerset("abcd")

[~~""~~, "a", "b", "c", "d",
"ab", "ac", "ad", "bc", "bd",
"abc", "abd", "acd", "bcd", "abcd"]

- Empty subset ignore
- Lexicographically sorted

decimal	binary (2 1 0 = c b a)	subset
0	0 0 0	""
1	0 0 1	"a"
2	0 1 0	"b"
3	0 1 1	"ab"
4	1 0 0	"c"
5	1 0 1	"ac"
6	1 1 0	"bc"
7	1 1 1	"abc"

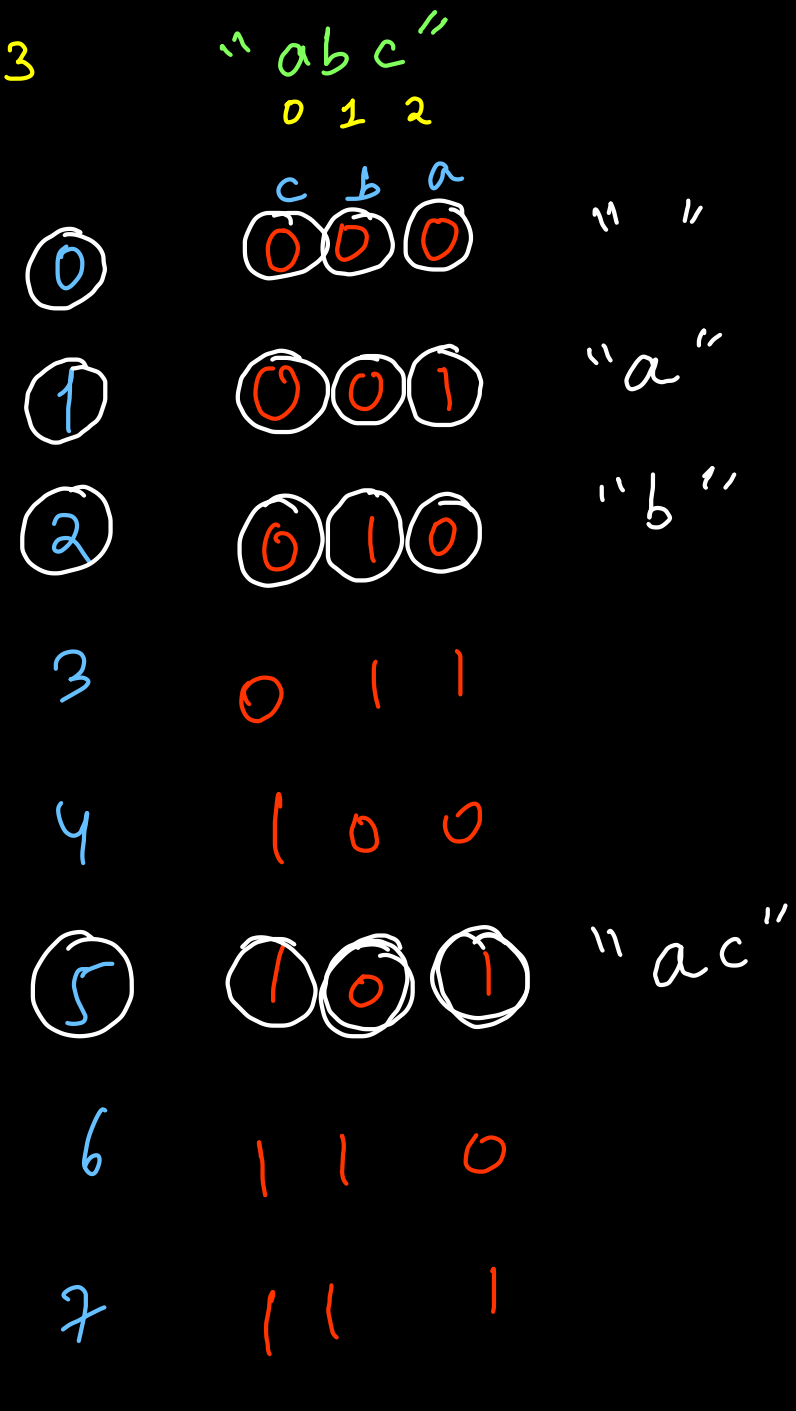
$8 = 1 << 3$

```java
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)
                subset += s.charAt(bit);
        }
        res.add(subset);
    }

    Collections.sort(res);
    return res;
}
```

to ignore empty subset

2^n = total subsets

Check set bit or not



1<<0 = 1

1<<1 = 2

1<<2 = 4

Exponentiation

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

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

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

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

power = n = sum terms of power of 2.

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

x^n

multiplier (x) $= (3*3) * (3*3) * (3*3*3*3)$

2^3	2^2	2^1	2^0
8	4	2	1
1	0	1	1

$11 = 1011$

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

$2^{-5} = (1/2)^5 = (0.5)^5 = 0.5 * 0.5 * 0.5 * 0.5 * 0.5$

$x^n = (1/x)^{-n}$

```java
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;
}
```

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

base $= (6^2)^2 = (6^4)^2 = 6^8$

power $= 13$

ans $= 1 * 6^1 * 6^4 * 6^8 = 6^{13}$

$13 = 1101$

$13 = 8 + 4 + 1$

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

Time $O(\log_2 N)$

Logarithmic

Space $O(1)$

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

power

$13 = 1101_2$

$6 = 110_2$

$3 = 11_2$

$1 = 1_2$

base = 6 ans = 1

6^1 $1 * 6 = 6^1$

6^2

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

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

Approach 1)

```java
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)

```java
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;
}
```