

# AGENDA:

DSA: Bit Manipulations

Truth Table for Bitwise Operators

Basic Bitwise Operator Properties

Problem 1: Single Number

Left Shift Operator ( $<<$ )

Power of Left Shift Operator

Setting the  $i$ -th Bit

Toggling the  $i$ -th Bit

Checking the  $i$ -th Bit

Unsetting the  $i$ -th Bit

Problem 2: Count the total number of SET bits in N

Problem 3: Single Number 3



# Bit-wise Operators: &, |, ^, ~, <<, >>

0 → F unlit OFF low voltage

1 → T set ON high voltage

if both a & b are 0  
else 1  
NOT

a	b	a&b	a b	a^b	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

↓ if a & b are  
↓, 0 otherwise

same same  
zero game

Power of 2s →  $1, 2, 4, 8, 16, \dots$   
 $2^0, 2^1, 2^2, 2^3, 2^4, \dots$

11 →  $2^3 + 2^1 + 2^0$

$8 + 2 + 1$

$1 \ 0 \ 1 \ 1$   
 $2^3 \ 2^2 \ 2^1 \ 2^0$

2	22	0
2	11	1
2	5	0
2	2	1
2	1	0
	0	

$$10110$$

$$2^4 2^3 2^2 2^1 2^0$$

$$16 + 4 + 2 = 22$$

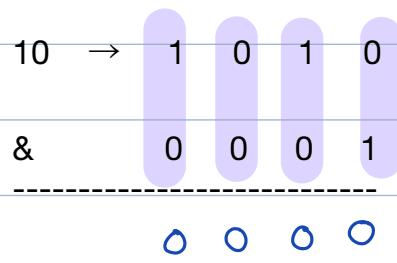
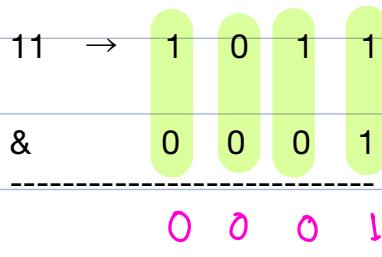
bin(11) →

# Basic Properties

1. Even / Odd Number  $\rightarrow$

$$A \& 1 = 1 \quad \text{odd}$$

$$A \& 1 = 0 \quad \text{even}$$



2.  $A \& 0 \rightarrow 0 \quad \text{A}$



3.  $A \& A \rightarrow A$

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

4.  $A | 0 \rightarrow A$

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

5.  $A | A \rightarrow A$ 

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

6.  $A \wedge 0 \rightarrow A$ 

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

7.  $A \wedge A \rightarrow 0$ 

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

$$2 \wedge 2 = 0$$

$$11 \wedge 11 = 0$$

**Cumulative Property** →

Commutative

$$A \oplus B = B \oplus A$$

$$A \mid B = B \mid A$$

$$A \wedge B = B \wedge A$$

**Associative Property** → Order in which operations are done  
doesn't matter

$$(A \oplus B) \oplus C = A \oplus (B \oplus C)$$

$$(A \mid B) \mid C = A \mid (B \mid C)$$

$$(A \wedge B) \wedge C = A \wedge (B \wedge C)$$



< Question- 1 > : Evaluate the expression:  $a \wedge b \wedge a \wedge d \wedge b$

$$\begin{aligned} & a \wedge b \wedge a \wedge d \wedge b \\ \Rightarrow & a \wedge a \wedge b \wedge b \wedge d \\ \Rightarrow & 0 \wedge 0 \wedge d \\ \Rightarrow & \underline{\underline{d}} \end{aligned}$$

< Question- 2 > : Evaluate the expression:  $1 \wedge 3 \wedge 5 \wedge 3 \wedge 2 \wedge 1 \wedge 5$

$$\begin{aligned} \Rightarrow & 1 \wedge 3 \wedge 5 \wedge 3 \wedge 2 \wedge 1 \wedge 5 \\ \Rightarrow & 1 \wedge 1 \wedge 3 \wedge 3 \wedge 5 \wedge 5 \wedge 2 \\ \Rightarrow & 0 \wedge 0 \wedge 0 \wedge 0 \wedge 2 \\ \Rightarrow & \underline{\underline{2}} \end{aligned}$$

Value of  $120 \wedge 5 \wedge 6 \wedge 6 \wedge 120 \wedge 5$  is - 0



**< Question > :** Given  $\text{arr}[N]$  where every element is present twice except one unique element.

Find that unique element.

$$A = [4, 5, 5, 4, 1, 6, 6]$$

$$\text{Output} = 1$$

Bruteforce  $\longrightarrow$  Create a counter of A and key with frequency == 1 in my array

```
▶ # Bruteforce
# Create a counter out of the given A itself and return the key with freq == 1

import collections
A = [7, 5, 5, 1, 7, 6, 1, 6, 4]

def single_number_1(A):
    counter = collections.Counter(A)

    for k, freq in counter.items():
        if freq == 1:
            return k

    return -1

print(single_number_1(A))

# TC: O(N)
# SC: O(N)
```

## ▶ # Optimised XOR ALL

```
A = [7, 5, 5, 1, 7, 6, 1, 6, 4]
```

```
def single_number_1(A):  
    xor = 0  
  
    for val in A:  
        xor ^= val  
  
    return xor
```

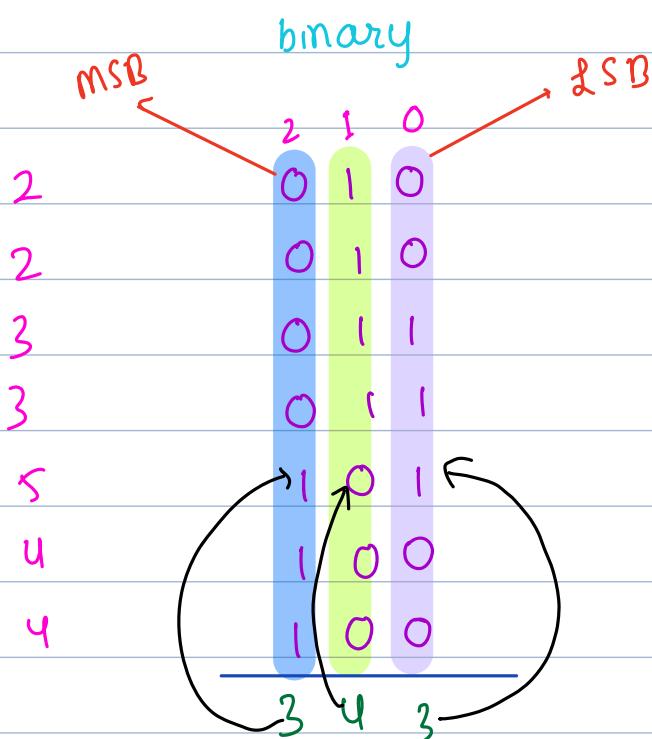
```
print(single_number_1(A))
```

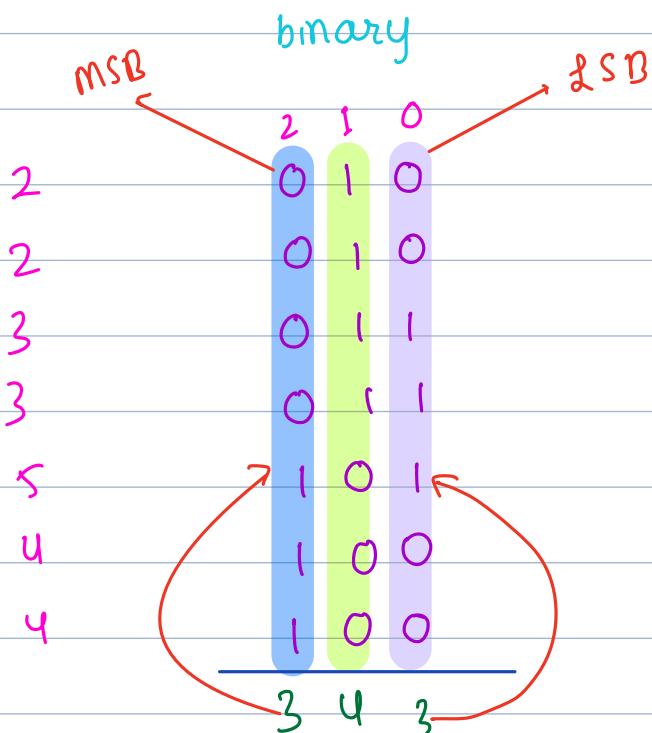
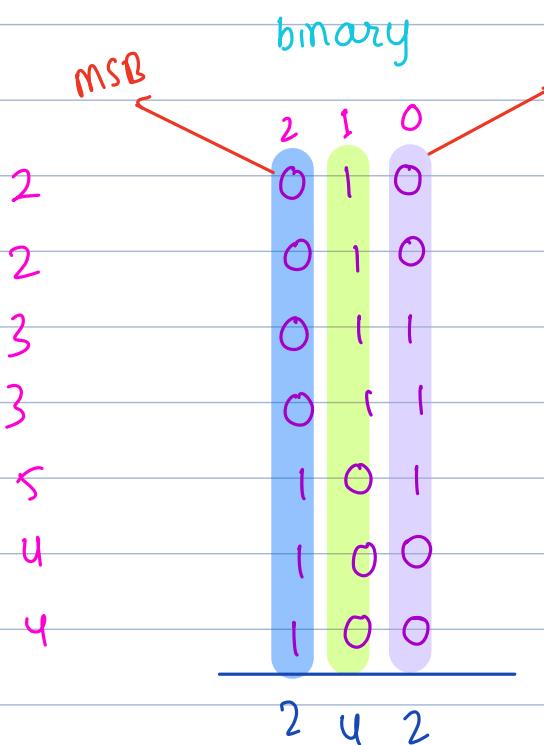
```
# TC: O(N)  
# SC: O(1)
```

→ 4

ans = 5

A = 2 2 3 3 5 u u

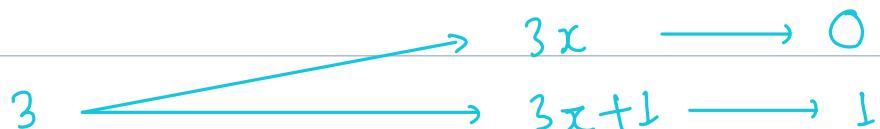
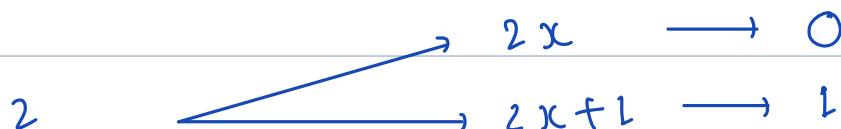




/.2

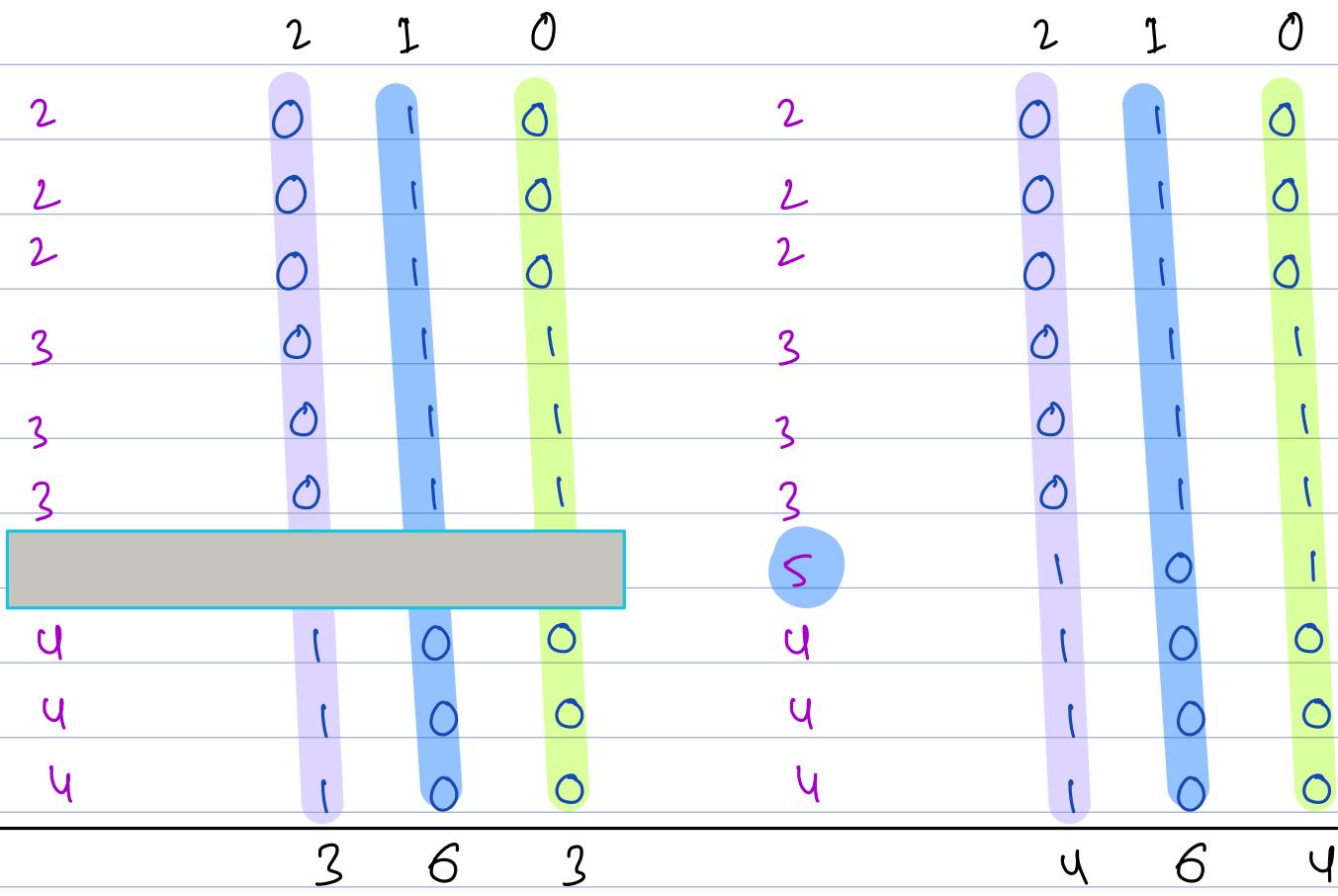
1 0 1

every element appears



4 → XOR all

A = 2 2 2 3 3 3 5 4 4 4

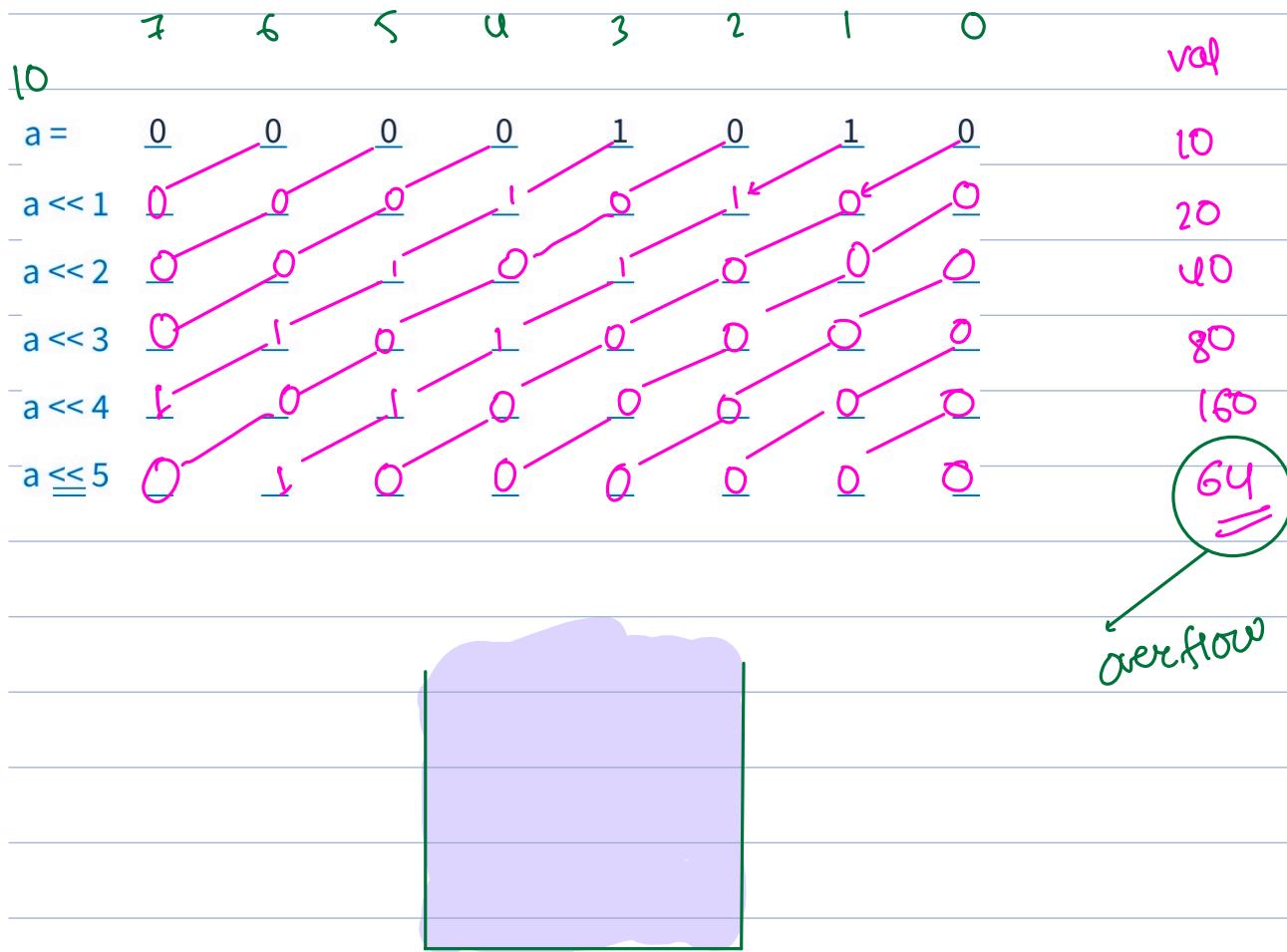


$$\begin{array}{r} 7.3 \\ \times 3 \\ \hline 101 \end{array}$$



# Left Shift Operator ( $<<$ )

8 bit rep



NOTE  $\rightarrow$  Not applicable for python

$\Rightarrow$  left shift by 1  $\Rightarrow$  multiply by 2  $= 2^1$   
left shift by 2  $\Rightarrow$  multiply by 4  $= 2^2$   
left shift by 3  $\Rightarrow$  multiply by  $= 2^3$   
⋮

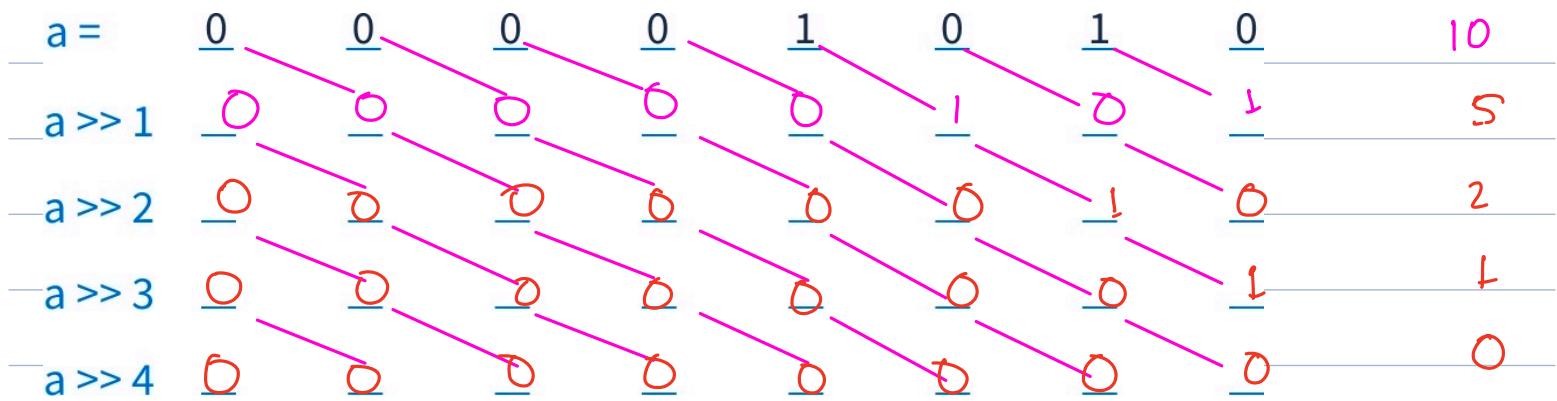
left shift by  $n \Rightarrow$  multiply by  $= 2^n$

$$1 \ll 3 = 1 * 2^3 = 8$$

$\Rightarrow$  6 5 4 3 2 1 0 = 00100000 8



# Right Shift Operator ( >> )



int division

right shift by 1  $\Rightarrow$  divide by 2

right shift by 2  $\Rightarrow$  divide by  $2^2$

right shift by 3  $\Rightarrow$  divide by  $2^3$

right shift by 4  $\Rightarrow$  divide by  $2^4$

right shift by 5  $\Rightarrow$  divide by  $2^5$

⋮

⋮

right shift by n  $\Rightarrow$  divide by  $2^n$

$$10 \gg 2 = \frac{10}{2^2} = \frac{10}{4} = \underline{\underline{2}}$$

$$4 \ll 3 = 4 * 2^3 = 4 * 8 = \underline{\underline{32}}$$



## Power of Left Shift Operator

Break: 9:35

If you want to set  $i^{\text{th}}$  bit

1. OR Operator  $\rightarrow N | (1 \ll i)$

U 3 2 1 0  
↓  
N = 1 0 0 1 1 0 1  
| 1 << U 0 0 1 0 0 0 0  
Final res 1 0 1 1 1 0 + N | (1 << 4)

Setting a bit  $\Rightarrow$  making it 1

Set  $i^{\text{th}}$  bit  $\Rightarrow N | (1 \ll i)$

I want to toggle  $i^{\text{th}}$  bit

6 5 4 3 2 1 0  
N 0 1 0 1 1 0 1  
^  $i \ll 3$  0 0 0 1 0 0 0  
----- 0 1 0 0 1 0 1

toggle 3<sup>rd</sup> bit  
 $0 \leftrightarrow 1$

6 5 4 3 2 1 0  
N 0 1 0 0 1 1 0 1  
^  $i \ll 3$  0 0 0 1 0 0 0  
----- 0 1 0 1 1 0 1

Toggle  $i^{\text{th}}$  bit  $\Rightarrow N \wedge (1 \ll i)$



How to check if the  $i^{\text{th}}$  bit is 0 or 1?

$$\begin{array}{cccccccccc} & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ N = & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 0 \\ \text{Q} 1 \ll 4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{array}$$

$$\begin{array}{cccccccccc} & 7 & 6 & 5 & 4 & 3 & 2 & 1 & 0 \\ N = & 1 & 1 & 0 & 0 & 0 & 1 & 1 & 0 \\ \text{Q} 1 \ll 4 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

Tell me if  $4^{\text{th}}$  bit is 0 or 1?

if  $N \& (1 \ll 4) == 0$  :

$4^{\text{th}}$  bit is 0

else

$4^{\text{th}}$  bit is 1

def check\_bit(N, i):  
 return N & (1 << i) > 0



## Unset i-th bit

[47]

# Code to unset the i-th bit

```
N = 0b11001101
```

```
# Use whatever we have learnt above
# Unset 2nd bit in N
```

.....

```
if the ith bit is already 0/unset don't touch it
if the ith bit is set or 1 -> toggle it to make it 0
.....
```

```
def unset(N, i):
    if check_bit(N, i):
        return N ^ (1 << i) # toggle ith bit
    return N
```

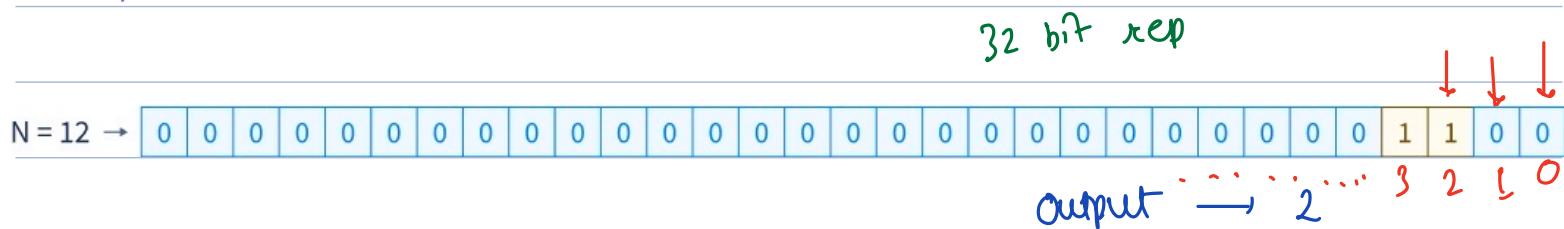
```
N = 0b100110110
i = 12
pb(N)
pb(unset(N, i))
```

→ 100110110  
100110110  
100110110

**< Question > :** Given an integer N. Count the set-bits in N.

$$N \leq 10^9$$

Example :



for all the bit position call check-bit fn

# Code to count the total number of SET bits in N

$N = 0b1001101100110$

```
ans = 0
for i in range(32):  $\rightarrow \log(N)$ 
    # if check_bit(N, i):
    #     ans += 1
    ans += check_bit(N, i)  $\rightarrow O(1)$ 
print(ans)
```

TC:  $O(\log N)$

int	$\approx 2^{32}$	$\log$
long	$\approx 2^{64}$	$\log$

## • Single Element III

Given arr[N], every element repeats twice except for 2 elements. Find the two unique elements.

Eg:  $A[6] = \{ 3, 6, 4, 4, 3, 8 \} = 6, 8$  $A[4] = \{ 4, 9, 9, 8 \} = 4, 8$

BruteForce Hashmap

01010  
 10 ↑ 10 8 8 9 9 11  
 A[12] = 10 10 8 8 9 9 11  
 ↓  
 01000  
 ↓  
 01011

xor all → 11 ^ 17 → 01011  
 ^ 10001  
 11010  
 ↓  
 1st bit

11 & 17 were diff

	4	3	2	1	0
10	0	1	0	1	0
10	0	1	0	1	0
8	0	1	0	0	0
8	0	1	0	0	0
9	0	1	0	0	1
9	0	1	0	0	1
11	0	1	0	0	1
12	0	1	1	0	0
12	0	1	1	0	0
6	0	0	1	1	0
6	0	0	1	1	0
17	1	0	0	0	1

10  
10  
11  
6  
6

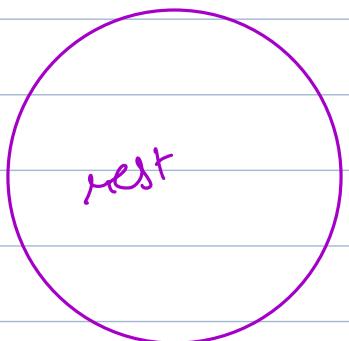
8 8  
9 9  
12 12  
17

set

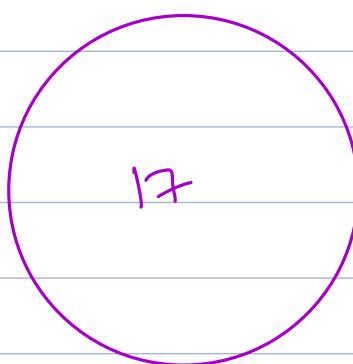
unset

4 3 2 1 0

10 0 1 0 1 0  
10 0 1 0 1 0  
8 0 1 0 0 0  
8 0 1 0 0 0  
9 0 1 0 0 1  
9 0 1 0 0 1  
11 0 1 0 1 1  
12 0 1 1 0 0  
12 0 1 1 0 0  
6 0 0 1 1 0  
6 0 0 1 1 0  
17 1 0 0 0 1



unet



set

```
# Code to find the two unique numbers
```

```
A = [10, 10, 8, 8, 9, 9, 11, 12, 12, 6, 6, 17, -1, -1, 0, 0]
```

```
# xor all
xor = 0
for val in A:
    xor ^= val
```

}  $O(N)$

```
# xor will contain the xor of 11 ^ 17
```

```
diff = -1
for i in range(32):
    if check_bit(xor, i):
        diff = i
        break
```

}  $\log N$

TC:  $O(N)$

SC:  $O(1)$

```
pb(xor)
print(diff)
```

```
a = 0 # first unique
b = 0 # second unique
```

```
for val in A:
    if check_bit(val, diff):
        a ^= val # group set
    else:
        b ^= val # group unset
```

}  $O(N)$

```
print(a, b)
```