

# BitMap

15 June 2022 20:36



AlgoU  
Session 4 B...

Alex and Sam are very good friends. Alex is doing programming a lot these days. He has set a target A for himself. Initially, Alex score is zero. Alex can only double his score by doing a question or Alex can seek help from Sam for doing questions which will contribute 1 to Alex score. Alex wants his score to be exactly A. Also, he does not want to take much help from Sam.

Find and return the minimum number of times Alex needs to take help from Sam to achieve a score of exactly A.

$0 \leq A \leq 10^9$

Input 1:

$A = 5$

Output 1:

2

## Bit Manipulation Background

- Representing numbers as binary
- When is representing nos. as binary useful?
- Iterate through all subsets of {1, 2, 3, 4}
- Bitwise operations - OR, AND, XOR, leftshift, rightshift

$\{1, 2, 3, 4\} = 3$

1	$\rightarrow (001)$
2	$\rightarrow (010)$
3	$\rightarrow (011)$
4	$\rightarrow (100)$
5	$\rightarrow (101)$
6	$\rightarrow (110)$
7	$\rightarrow (111)$
8	$\rightarrow (1000)$

$8 \rightarrow 1000_2$

$12 \rightarrow 1100_2$

$12 = 8 + 4 + 2 + 0$

## Bit Manipulation Background

- Representing numbers as binary
- When is representing nos. as binary useful?
- Iterate through all subsets of  $\{1, 2, 3, 4\}$
- Bitwise operations - OR, AND, XOR, leftshift, rightshift

$$\begin{array}{c} \{1, 2, 3\} \\ \rightarrow 1 \rightarrow (0\ 001) \end{array} \quad \left. \right\}$$

$$\begin{array}{c} \{1\} \\ \{2\} \\ \{2, 3\} \\ \vdots \\ \{1 \rightarrow 2^{n-1}\} \end{array}$$

## Bit Manipulation Background

- Representing numbers as binary
- When is representing nos. as binary useful?
- Iterate through all subsets of  $\{1, 2, 3, 4\}$
- Bitwise operations - OR, AND, XOR, leftshift, rightshift

$$\begin{array}{c} \text{Set } S_1 \\ \begin{array}{cccc} & 1 & 0 & 1 & 0 \\ \text{S} & \frac{1}{1} & \frac{0}{0} & \frac{1}{0} & \frac{0}{0} \end{array} \end{array}$$

$$\begin{array}{c} n \rightarrow (n)_2 \\ \text{OR} \\ \begin{array}{c} a = (1010) \\ b = (1100) \\ \hline 1110 \rightarrow \text{OR} \end{array} \\ \text{AND} \\ \begin{array}{c} 1000 \\ 000110 \\ \hline 0000 \end{array} \\ \text{XOR} \\ \begin{array}{c} 1010 \\ 0100 \\ \hline 1110 \end{array} \end{array}$$

$$\begin{array}{c} \text{XOR} \\ \begin{array}{c} 1010 \\ 0100 \\ \hline 1110 \end{array} \end{array} \rightarrow \begin{array}{c} 1010 \\ 0100 \\ \hline 1110 \end{array} \wedge \begin{array}{c} 1010 \\ 0100 \\ \hline 1110 \end{array} \rightarrow \{1, 2, 3, 5\}$$

## Introduction

Bitwise Operations are faster and closer to the system and sometimes optimize the program to a good level.

1 byte comprises of 8 bits and any integer or character can be represented using bits in computers, which we call its binary form.

# Introduction

Bitwise Operations are faster and closer to the system and sometimes optimize the program to a good level.

1 byte comprises of 8 bits and any integer or character can be represented using bits in computers, which we call its binary form (contains only 1 or 0) or in its base 2 form.

Example:

$$1) 14 = \{1110\}_2$$

$$= 1 * 2^3 + 1 * 2^2 + 1 * 2^1 + 0 * 2^0$$

$$= 14.$$

$$2) 20 = \{10100\}_2$$

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

$$= 20.$$

$\star 2^x$  without using  $\star / +$

$$\hookrightarrow A \underline{\ll n} \rightarrow A \times 2^n$$

$$1 \ll 3$$

$$0001$$

$$0010 \rightarrow \times 2$$

$$0100 \rightarrow \times 2$$

$$\downarrow 000$$

## Bitwise Operators

NOT ( $\sim$ ):

AND ( $\&$ )

OR ( $\|$ )

XOR ( $^$ )

$$A \gg n = \frac{A}{2^n}$$

$$8 \gg 1 \rightarrow 8 / 2^2 = 4$$

$$\hookrightarrow 1000$$

$$\oplus$$

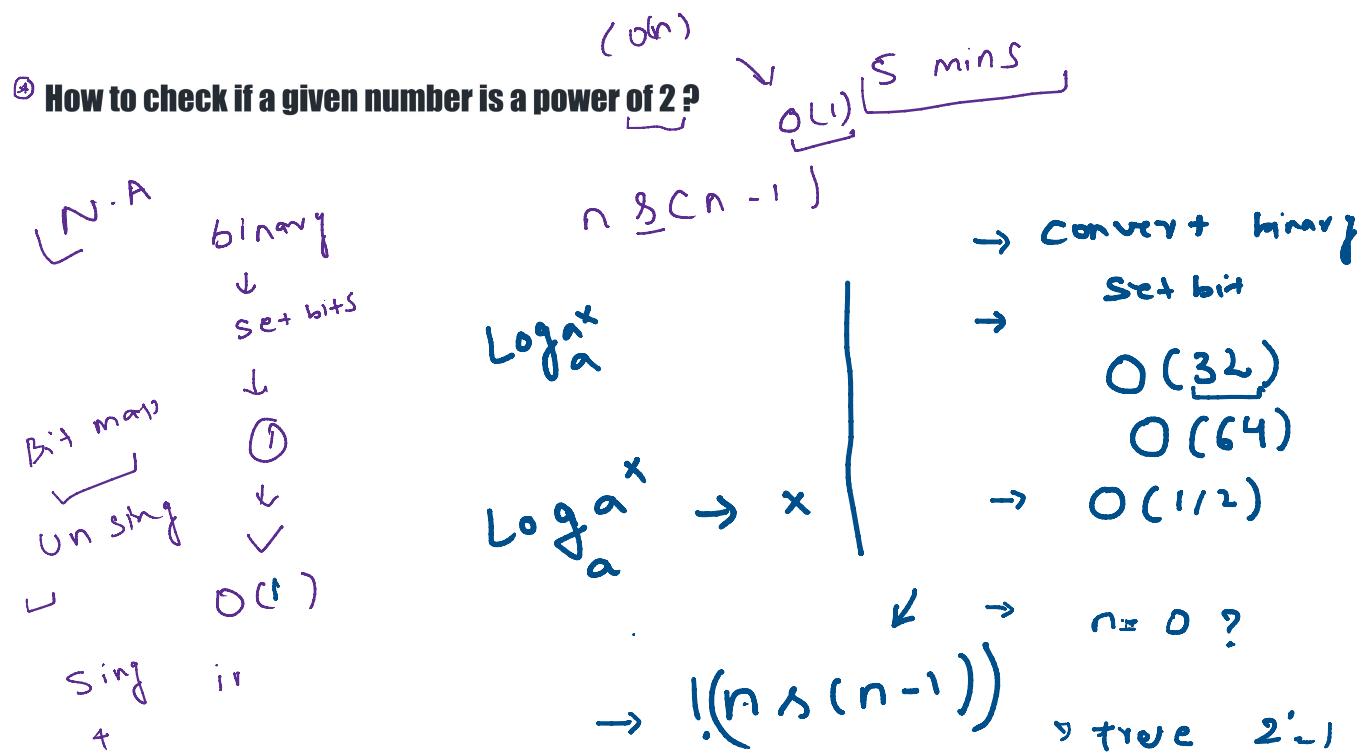
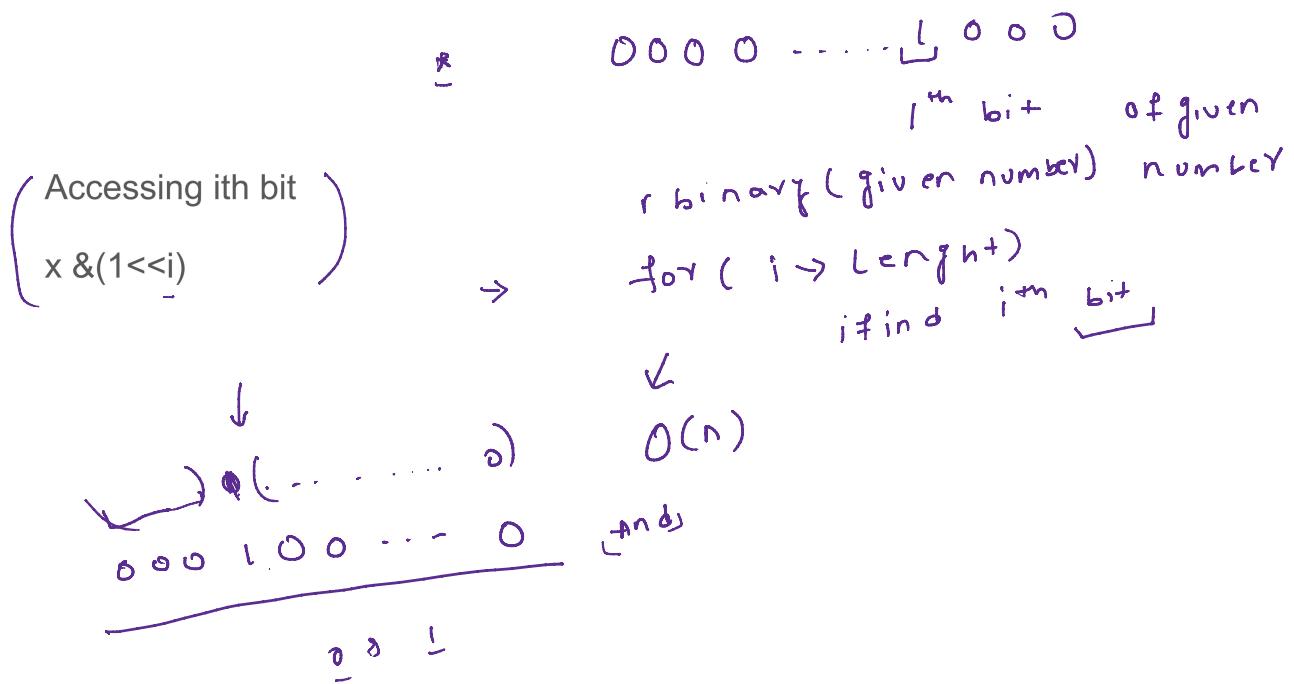
$$0100$$

**Left Shift ( $\ll$ ):** Left shift operator is a binary operator which shifts the some number of bits, in the given bit pattern, to the left and append 0 at the end. Left shift is equivalent to multiplying the bit pattern with  $2^k$  ( if we are shifting k bits ).

$$1 \ll n = 2^n$$

**Right Shift ( $\gg$ ):** Right shift operator is a binary operator which shifts the some number of bits, in the given bit pattern, to the right and append 1 at the end. Right shift is equivalent to dividing the bit pattern with  $2^k$  ( if we are shifting k bits ).

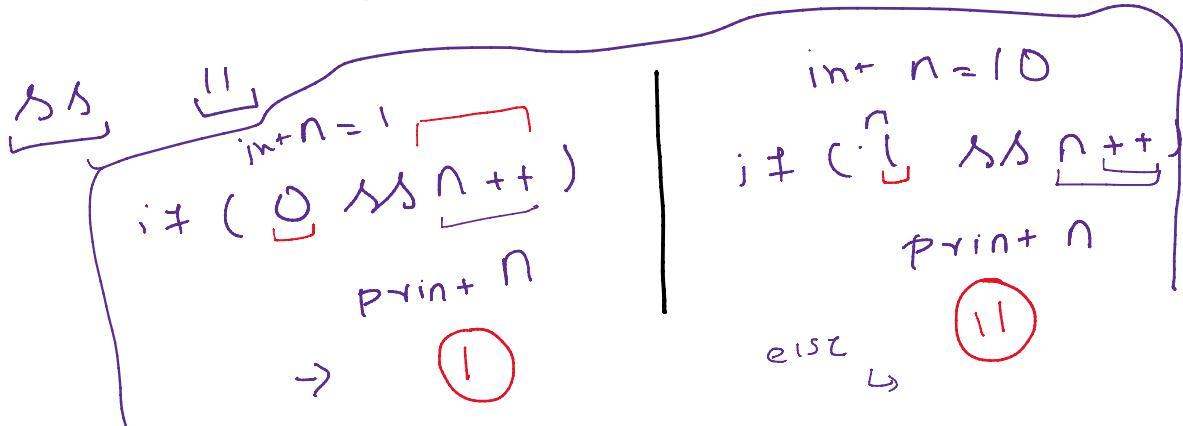
$$16 \gg 4 = 1$$



$$\begin{aligned} \rightarrow n &\rightarrow 16 \rightarrow 10000 \\ 15 &\rightarrow 0111 \\ 2^4 &\rightarrow \underline{1} \text{ set bit} \\ &\rightarrow \underline{1} \dots \underline{0} (n-1)) \end{aligned}$$

$$2 \rightarrow 100\cdots0$$

$(n \geq 1) \cdot (n \leq 2^{(n-1)})$



### How to check if a given number is a power of 2?

```
bool isPowerOfTwo(int x){
    if(x == 0) return false;
    else {
        while(x % 2 == 0) x /= 2;
        return (x == 1);
    }
}
```

$$[0] \quad \begin{smallmatrix} 1 & 0 & 0 & 0 \\ \dots \end{smallmatrix}$$

int (n) = 10

unsigned =  $n = 10$

0  $\underline{\dots} \dots \dots \dots$

int  $\rightarrow 2^{-32} \quad 2^{32}-1$

unsigned  $\rightarrow 2^{32}$

## How to check if a given number is a power of 2 ?

```
bool isPowerOfTwo(int x)  
{  
    // x will check if x == 0 and !(x & (x - 1)) will check if x is a power of 2 or not  
    return (x && !(x & (x - 1)));  
}
```

$\left( \begin{array}{l} \text{__builtin_popcount}(x) \\ \text{__builtin_popcountll}(x) \end{array} \right)$

$\begin{array}{rcl} 2 & \rightarrow & 1 \\ 3 & \rightarrow & 2 \end{array}$

Count set bits

② XOR

### Quickest way to swap two numbers

$a \wedge= b;$   
 $b \wedge= a;$   
 $a \wedge= b;$

$$\begin{array}{l} a = c = a \\ a = b \\ b = c \end{array}$$

$a = 10 \quad b = 1$

$$\begin{array}{l} a = a + b \\ b = a - b \\ a = a - b \end{array}$$

$$\begin{array}{l} a = 11 \\ b = 11 - 1 = 10 \\ a = 11 - 10 \\ = 1 \end{array}$$

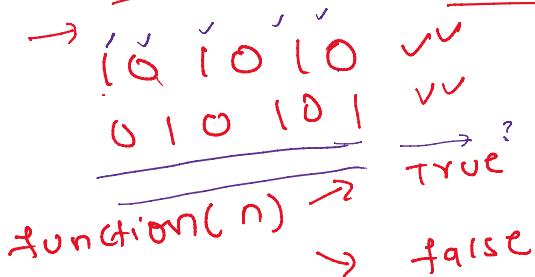
$a = 1 \quad b = 10$

$$\begin{array}{l} P_1 \rightarrow a \wedge a = 0 \\ P_2 \rightarrow a \wedge 0 = a \\ a' = a \wedge b \\ b = b \wedge a' \\ = b \wedge a \wedge b \end{array}$$

$$\begin{array}{l} = b \wedge b \wedge a \\ = 0 \wedge a \\ b = a \end{array}$$

④ Check if a number has bits in alternate pattern

5 min



$[N.A] \rightarrow \vee$

$$n = 101010 \quad x = n \wedge (n \gg 1)$$

$$\text{xor} \quad 010101 - \leftarrow x \wedge (x+1)$$

$$\begin{array}{r} 111111 \\ \hline 100000 \\ \hline n \wedge (n-1) \end{array}$$

Check if a number has bits in alternate pattern

```
bool allBitsAreSet(unsigned int n)
{
    }
```

```
    if (((n + 1) & n) == 0) return true;
    return false;
}
```

```
bool bitsAreInAltOrder(unsigned int n)
{
    unsigned int num = n ^ (n >> 1);
    return allBitsAreSet(num);
}
```

→ check  
if all set

can we replace  
XOR with ~~A~~  $\oplus$

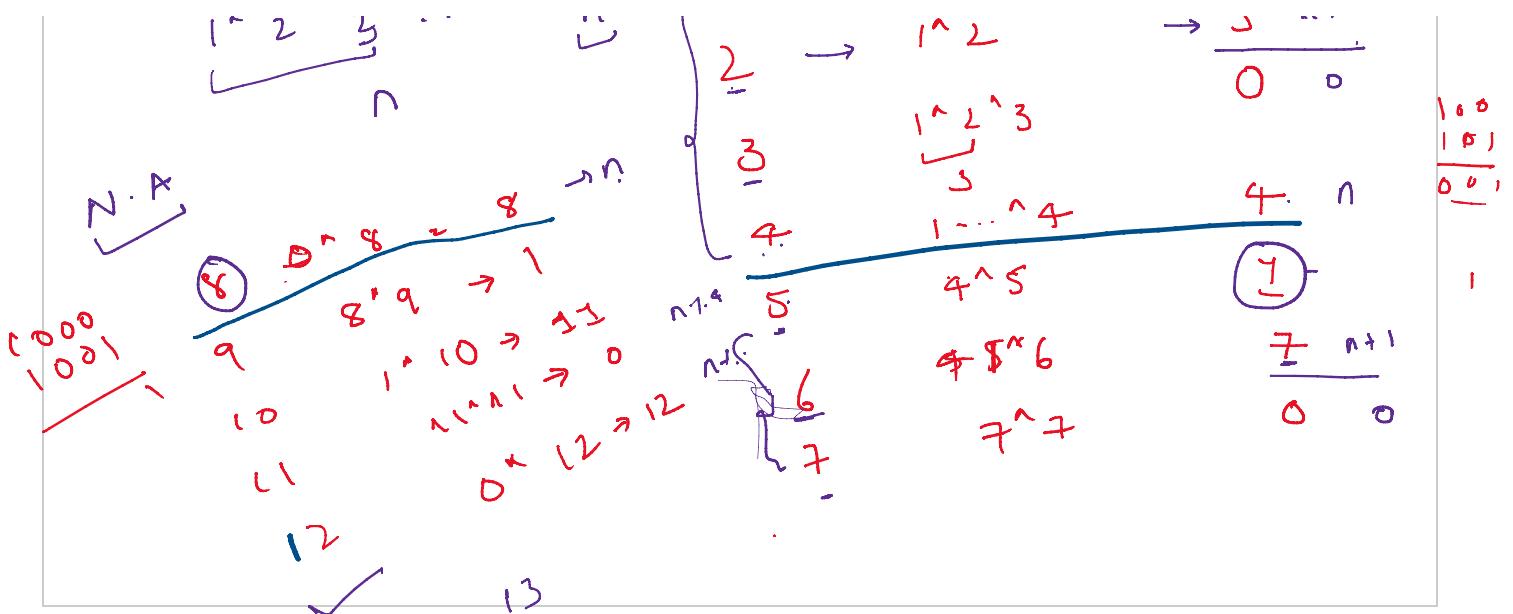
$$\begin{array}{r} 101010 \\ \oplus 010101 \\ \hline 111111 \end{array}$$

④ Compute XOR from 1 to n

$$\begin{array}{c} 1^{\wedge} 2^{\wedge} 3^{\wedge} \dots \quad \vdots \quad \{ \quad \begin{array}{l} n \\ 2 \\ \dots \end{array} \quad \rightarrow \quad \begin{array}{l} 1 \\ 1^{\wedge} 2 \\ \dots \end{array} \quad \rightarrow \quad \begin{array}{l} \text{sum} \\ \text{out} \end{array} \\ \hline \end{array}$$

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

$$\rightarrow \frac{3 \cdot n + 1}{0 \quad 0}$$



## Compute XOR from 1 to n

```

int computeXOR(int n)
{
    if (n % 4 == 0) return n;
    if (n % 4 == 1) return 1;
    if (n % 4 == 2) return n + 1;
    else
        return 0;
}

```

$$\textcircled{1} \rightarrow h = \log_2 n.$$

$$B.S \rightarrow \log_2 n$$

## Compute XOR from 1 to n

```

int computeXOR(int n)
{
    ✓✓ if (n % 4 == 0) return n;
    ✓✓ if (n % 4 == 1) return 1;
    ✓✓ if (n % 4 == 2) return n + 1;
}

```

$$\left. \begin{array}{l} T(n-1) \% 4 == 0 \rightarrow n-1 \\ T(n-1) \% 4 == 1 \rightarrow 1 \\ T(n-1) \% 4 == 2 \rightarrow n \\ T(n-1) \% 4 == 3 \rightarrow 0 \end{array} \right\} \begin{array}{l} P_1 \\ P_2 \end{array}$$

$1^n 2^n 3^n \dots n^n$

```

    if (n % 4 == 1) return 1;
    if (n % 4 == 2) return n + 1;
    else
        return 0;
}

```

*case(1)*

$$T(n-1) \leq 3^n - 1$$

$$\begin{aligned} n \% 4 = 0 &= 1^n 2^n 3^n \dots n^n \\ &= T(n-1 \% 4 = 3)^n \\ &= 0^n \\ &= n \end{aligned}$$

$$\begin{aligned} T(n \% 4 = 0) &= 1^n 2^n \dots n^n \\ &= T(n-1 \% 4 = 0)^n \\ &\quad + (n-1 \% 4 = 0) \\ &\quad \downarrow \\ &= (n-1)^n \\ &\quad + 1 \\ &\quad + 0 \\ &\quad \times \\ &\quad 1 \\ &\quad 0 \end{aligned}$$

*{ n-1 → even }*

$$(n \% 4 = 2) = T((n-1) \% 4 = 1)^n$$

$$\begin{aligned} &\times \quad \downarrow^n \\ &1 0 1 0 \dots 0 = n+1 \\ &\times \quad 0 0 0 0 0 0 1 \\ &\quad \diagup \quad \diagdown \\ &\quad \times \quad 1 \end{aligned}$$

$$\begin{aligned} n \% 4 = 3 &= T(n \% 4 = 2)^n \\ &= n^n \end{aligned}$$

$$2^0$$

- Q. In an array, every number appears twice except one. Find that number.

$A = \{ 1, 1, 3, 4, 3, 4, 9 \} \rightarrow \textcircled{1}$

$\hookrightarrow P_1 = a^na = 0$

$P_2 = 0^na = a$

$\hookrightarrow \text{Aditya} \rightarrow O(n \log n) \text{ O(1)}$

$\hookrightarrow \text{Savan} \rightarrow O(n) \text{ O}(n)$

Time Complexity:

```

for(i = 0; i < n; i++) {
    hashmap[i] += freq;
    if(freq == 1)
        print(i);
}
    
```

- \* You are given an array in which every number appears thrice except one(which occurs exactly once). Find that number

$A = \{ 1, 1, 1, 2, 9, 2, 9, 2, 9, 10 \} \rightarrow \textcircled{10}$

$B = \{ 1, 1, 1, 3, 3, 3, 4 \} \rightarrow 4$

Time Complexity:

$N \cdot A$   $\rightarrow$   $O(N \cdot 3^D)$

BITMAP  $\rightarrow$   $O(2^D)$

Why?

Diagram illustrating the bit manipulation approach:

Input array:  $\{ 1, 1, 1, 2, 9, 2, 9, 2, 9, 10 \}$

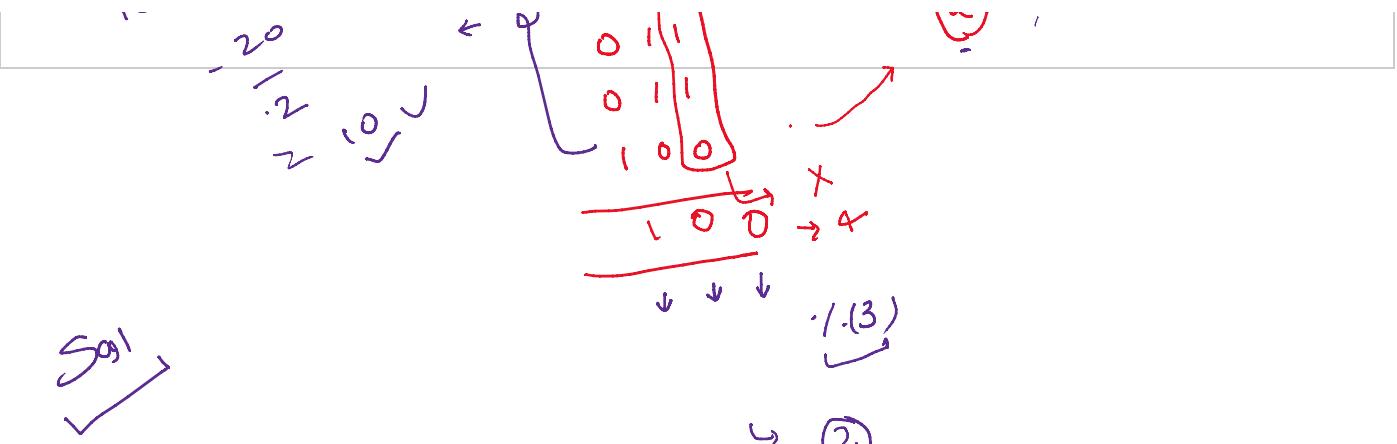
Bitmask:  $\{ 000, 001, 010, 011, 100, 101, 110, 111 \}$

Counting 1s in each position:

- Position 0: 3 ones (marked with a red bracket)
- Position 1: 3 ones (marked with a red bracket)
- Position 2: 3 ones (marked with a red bracket)
- Position 3: 3 ones (marked with a red bracket)
- Position 4: 1 one (marked with a red bracket)
- Position 5: 1 one (marked with a red bracket)
- Position 6: 1 one (marked with a red bracket)
- Position 7: 1 one (marked with a red bracket)

Result:  $\{ 1, 1, 1, 3, 3, 3, 4 \}$

Final answer:  $\textcircled{10}$



$$\rightarrow \xrightarrow{T.C} (O(n)) \\ \downarrow O(1) \quad \left. \begin{array}{l} \downarrow \\ O(n) \end{array} \right\} \rightarrow$$

$$\sum 1, 1, 1, 3, 3, 3, 4 \quad \xrightarrow{\text{Sum-original} = 16}$$

① Sum-original = 16

②  $\sum 1, 3, 4$

③  $\sum 3, 9, 12 = 24$

④  $24 - 16 = 8 \xrightarrow{2} 4$

⑤  $S.C \quad S.L \quad \rightarrow O(n) \quad O(n)$

$$\sum 3, 3, 3, 4 \quad \xrightarrow{\text{Sum} = 12} 0$$

$O(n) \quad O(n)$

Q: 011  
011  
011  
10  
11

✓

→ Each element rep ~ 5 times, 1 element

→