Todays Content:

- Check Bit Revise ✓
- Unique Element I ✓
- Unique Element II
- Unique Element III ✓
    a) Updated version of Unique element-III ✓
- Sum of all xor pairs

Problems:

$0 <= N <= 10^9$
$0 <= i <= 30$
} For given N check if i^{th} bit is set or Not?
1 0

```
bool CheckBit(N, i){
    return [(N>>i)&1==1]
}
```

	6	5	4	3	2	1	0
N=53 :	0	1	1	0	1	0	1

i=0 :

N&1==1 : 0^{th} bit set

i=1

	6	5	4	3	2	1	0
N=53 :	0	1	1	0	1	0	1
N>>1 :		0	1	1	0	1	0

(N>>1)&1==1 : 1^{st} bit set

i=2

	6	5	4	3	2	1	0
N=53 :	0	1	1	0	1	0	1
N>>2 :			0	1	1	0	1

(N>>2)&1==1 : 2^{nd} bit set

Unique element:

Every element repeats twice except 1, find unique element?

Ex1: ar[7] = { 3 2 3 7 2 8 7 } : ans = 8

Ex2: ar[9] = { 3 6 2 3 5 4 5 6 4 } : ans = 2

Idea: xor of all elements:

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

Unique element II

Given ar[N], every element repeats thrice except 1, find the unique element?
↳ comes 1 times

Constraints:

$1 <= N <= 10^6$

$1 <= ar[i] <= 10^9$

ar[7] = 6 5 6 4 5 6 5 → ans = 4

ar[13] = 5 7 5 4 7 11 11 9 11 7 5 4 4 → ans = 9

a) for every element, iterate in array get freq == 1

TC: N * {O(N)} ⇒ O(N²) SC: O(1)

optimize: Optimize using hashmap, to get freq of elements

TC: O(N + N) = O(N) SC: O(N)

b) Sort the arr[], & iterate on arr[] & get unique elements

TC: O(NlogN + N) ⇒ O(NlogN) SC: O(1)

c) xor of all elements:

ar[7] = 6 ^ 5 ^ 6 ^ 4 ^ 5 ^ 6 ^ 5 = 6 ^ 5 ^ 4 ≠ final ans

ar[13] = 5 ^ 7 ^ 5 ^ 4 ^ 7 ^ 11 ^ 11 ^ 9 ^ 11 ^ 7 ^ 5 ^ 4 ^ 4 =

= 4 ^ 9 ^ 11 ^ 7 ^ 5

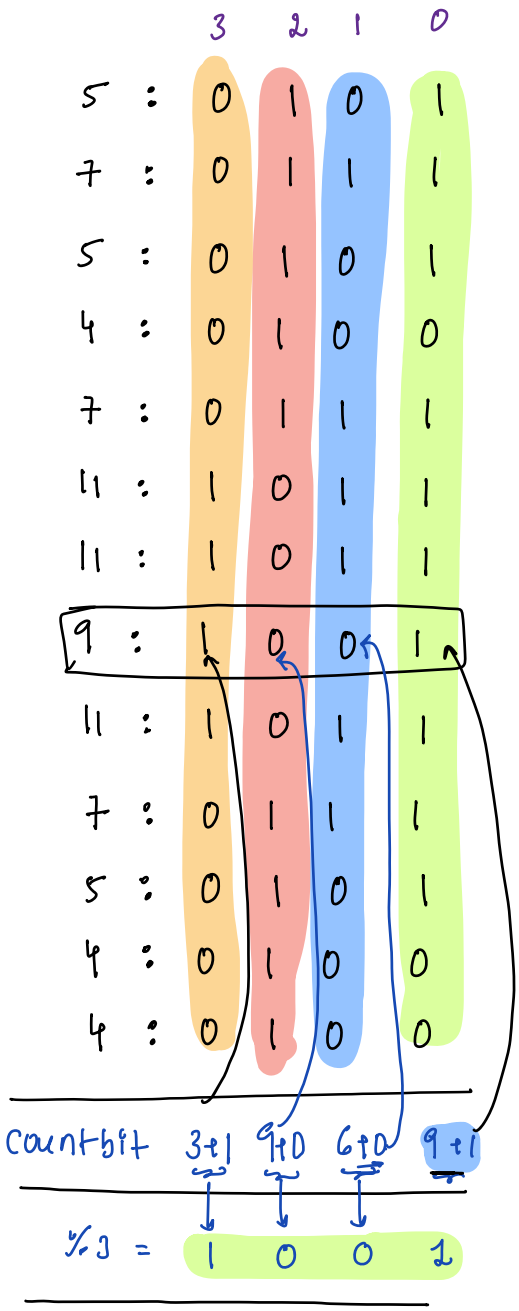
Ideas:

In general any Bit manipulation based question try to solve bit by bit?

Contribution technique works
↳ based on bit by bit pos?

Idea3:

ar[13] = 5 7 5 4 7 11 11 9 11 7 5 4 4



: Idea: at every bit calculate no: of set bits :

Say at i^{th} bit no: of arr[] elements with i^{th} bit set = c_i

if(c_i % 3 != 0) => In unique element i^{th} bit set

Pseudo Code:

```
int TripleTrouble(int ar[], int n){        TC: 32*N → O(N)
                                           SC: O(1)
    ans = 0;
    i = 0; i<32; i++){
        // for ith bit, calculate no: of ar[] with ith bit set ?
        ci = 0
        j = 0; j<n; j++){
            if(checkbit(ar[j], i) == True){        1<<i = 2^i
                ci = ci + 1
            }
        }
        if (ci % 3 != 0){ // for unique ele ith bit is set ?
            ans = ans + (1<<i) // 2^i, because decimal for ith bit = 2^i
        }
    }
    return ans;
}
```

Extention:

Every element repeats thrice except 1 element repeats 2 time

→ Every element repeats 4 times

→ Except 1 element, repeat 1 time

: xor of all elements

→ Except 1 element, repeat 2 time

: xor of all elements : { a^a^a^b^a^b^c^c^c^c }

: if (ci % 4 != 0){ // for unique ele ith bit is set }

→ Except 1 element, repeat 3 time :

: xor of all elements : { a^a^a^b^a^b^c^c^c^c^b }

Unique Element - [1]

Given ar[N], every element repeats twice except 2 elements which occur 1 time

find the 2 unique elements

Eg:

ar[6] = { 3 6 4 4 3 8 } : ans = {6, 8}

ar[4] = { 4 9 9 8 } : ans = { 4, 8 }

ar[6] = { 2 4 4 6 6 2 } not possible

Ideas:

a) for every element, iterate in array get freq == 1

TC: N * {O(N)} ⇒ O(N²) SC: O(1)

optimize: Optimize using hashmap, to get freq of elements

TC: O(N + N) = O(N) SC: O(N)

b) Sort the arr[], iterate on arr[] & get unique elements

ar[8]: { 6 4 6 3 7 3 9 7 }

Sort : { 3 3 4 6 6 7 7 9 } : ans = {4, 9}

TC: O(NlogN + N) ⇒ O(NlogN) SC: O(1)

c) XOR of all elements

ar[6] = { 3 ^ 6 ^ 4 ^ 4 ^ 3 ^ 8 } = 6 ^ 8 = 14

ar[4] = { 4 ^ 9 ^ 9 ^ 8 } = 4 ^ 8 = 12

Obs: xor of all ele = {xor of unique elements}

	1010	1000	1100	0110	1010	1100
ar[12] =	10	8 8	9	12	9	6 11	10	6	12	17
	1000	1001	1001	1011	0110	10001

Step 1: xor of all ar[]

v = 11 ^ 17

	4	3	2	1	0
17 :	1	0	0	0	1
11 :	0	1	0	1	1
v =	1	1	0	1	0

Obs: for v 1st bit is 1

Both unique have diff diff value at bit pos = 1

Obs: for v 3rd bit is 1

Both unique have diff diff value at bit pos = 3

Obs: for v 4th bit is 1

Both unique have diff diff value at bit pos = 4

Iterate on arr[], split it based on 1st bit data

- Set: all elements with 1st bit set = 1 → [10 6 11 10 6] → xor of all ele: 11
- Unset: all elements with 1st bit unset = 0 → [8 8 9 12 9 12 17] → xor of all ele: 17

Iterate on arr[], split it based on 3rd bit data

- Set: all elements with 3rd bit set = 1 → [10 8 8 9 12 9 11 10 12] → xor of all ele = 11
- Unset: all elements with 3rd bit unset = 0 → [6 6 17] → xor of all ele = 17

Pseudocode:

TC: O(N+32+N) ≈ O(N)
SC: O(1)

```
int UniqueII(int arr[], int n){

    Step1: xor of all ele
        int v=0;
        i=0; i<n; i++) { v= v^arr[i] }
        // v = xor of 2 unique elements

    Step2: find bit pos based on which we need to split
        p = __                    { Take any set bit, that's fine }
        i=0; i<32; i++){
            if(checkBit(v, i) == True){
                // Both unique elements different data at ith bit pos
                p = i; break
            }
        }

    Step3: Split arr[] based on bit pos = p
        set = 0, unset = 0
        i=0; i<n; i++){
            if(checkbit(arr[i], p) == True){
                // arr[i] goes to set space
                set = set^arr[i]
            }
            else{
                // arr[i] goes to unset space
                unset = unset^arr[i]
            }
        }
    // 2 unique elements are set & unset
}
```

3Q)

Given ar[N], contains all elements from [1, N+2] except 2 ele

find 2 missing elements?

ar[4] = {3 6 1 4} : missing ele = 2 5 (range {1-6})

ar[5] = {1 6 4 7 5} : missing ele = 2 3 (range {1-7})

ar[4] = {3 3 4 2} : missing ele = 1, 5, 6 (range {1-6})

Ideas:
a) 2 nested loops
b) Using hashmap
c) Sort
d) Bringing ele to it's corct pos → Doubt Session
e) Create mathematical equations

XOR idea:

ar[5] = {1^6^4^7^5} {1^2^3^4^5^6^7} (range {1, 7})

Idea: every element repeats twice, except 2, find 2 unique elements

are 2 missing elements

→ TC: O(N) SC: O(1) : TODO

→ Will discuss in todays doubt Session

Pseudocode : TODO/ Doubts

→ maths equations :

arr[N] :→ in arr[] missing elements are a & b

↳ {1, N+2}

All elements from [1, N+2] — all arr[] elements

{1 2 3 .. a .. b .. N+2} {1 2 3 .... N+2}

eq1 : Sum of all [1, N+2] − Sum of arr[] elem = a+b

eq2 : Sum of square [1, N+2] − sum of square arr[] ele = a^2+b^2

Say : $a+b = V_1$ ✓

$a^2+b^2 = V_2$ ✓

↳ square = $(a+b)^2 = V_1^2 = a^2+b^2+2ab = V_1^2$

$= V_2 + 2ab = V_1^2$

$(a-b)^2 = a^2+b^2-2ab$ $\quad \boxed{2ab = V_1^2 - V_2}$

$= V_2 - (V_1^2 - V_2)$

$= V_2 - V_1^2 + V_2$

$(a-b)^2 = 2V_2 - V_1^2$

$a-b = \sqrt{2V_2 - V_1^2}$

$a+b = V_1$

Calculate a & b

$a = \frac{V_1 + \sqrt{2V_2 - V_1^2}}{2}$

$b = a - V_1$

4Q) Given ar[N], calculate sum of xor of all pairs ? → {Google}

ar[5] = {3 5 6 8 2}

All pairs:

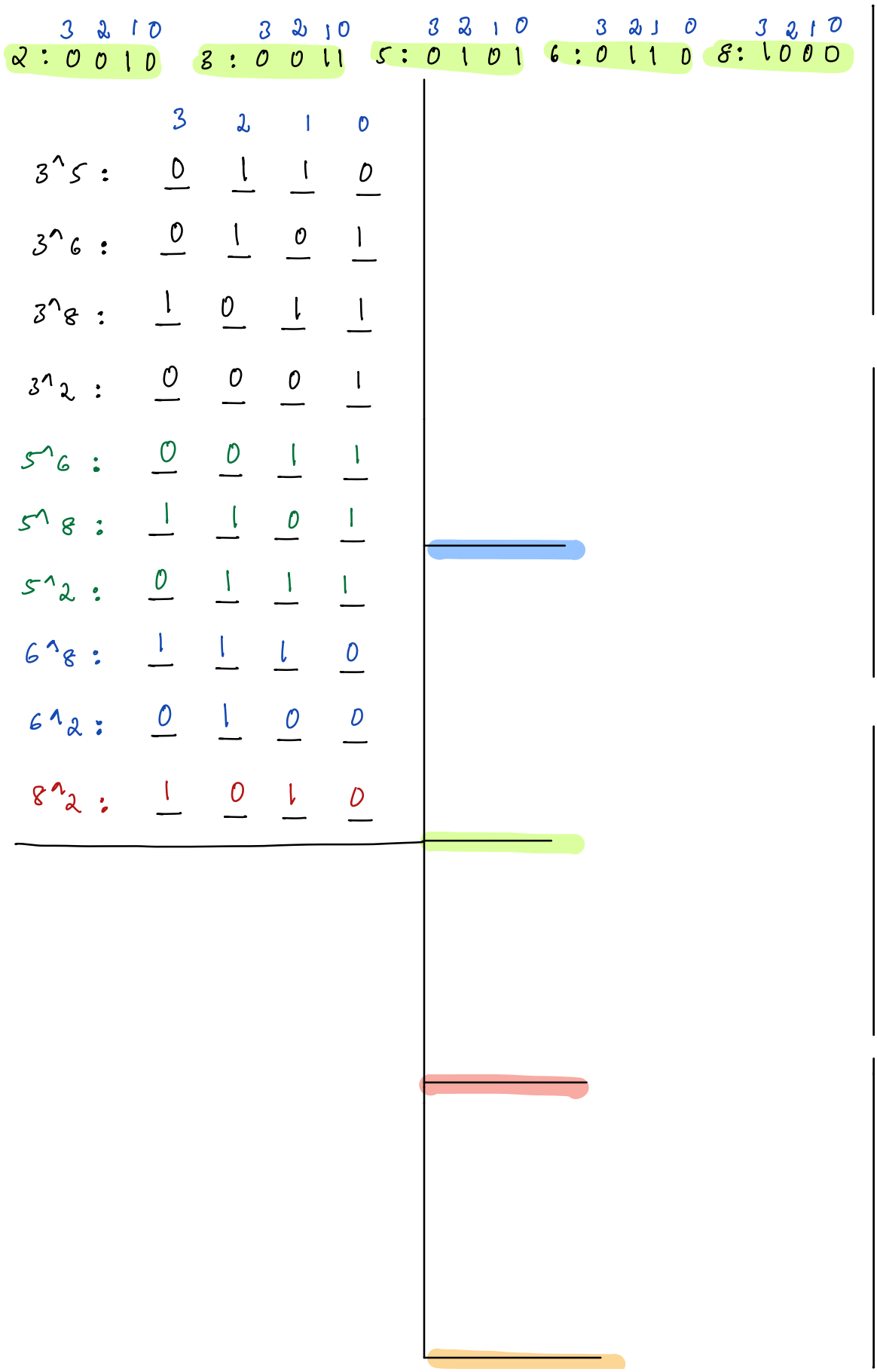
3^3 + 3^5 + 3^6 + 3^8 + 3^2

5^3 + 5^5 + 5^6 + 5^8 + 5^2

6^3 + 6^5 + 6^6 + 6^8 + 6^2

8^3 + 8^5 + 8^6 + 8^8 + 8^2

2^3 + 2^5 + 2^6 + 2^8 + 2^2


3 2 1 0
2 : 0 0 1 0
3 2 1 0
3 : 0 0 1 1
3 2 1 0
5 : 0 1 0 1
3 2 1 0
6 : 0 1 1 0
3 2 1 0
8 : 1 0 0 0
3 2 1 0
3^5 : 0 1 1 0
3^6 : 0 1 0 1
3^8 : 1 0 1 1
3^2 : 0 0 0 1
5^6 : 0 0 1 1
5^8 : 1 1 0 1
5^2 : 0 1 1 1
6^8 : 1 1 1 0
6^2 : 0 1 0 0
8^2 : 1 0 1 0