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

```
        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

```
        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: Take 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 : 4

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

Ideas:

a) Get freq of every element using a loop == 1

TC: O(N^2) SC: O(1)

Optimization: Get freq of element using hashmap

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

b) Sort ar[] get freq of every element = 1

ar[10] = 6 6 7 9 7 6 9 9 7 8

Sort ar[]

ar[] = 6 6 6 7 7 7 8 9 9 9

↳ unique element

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

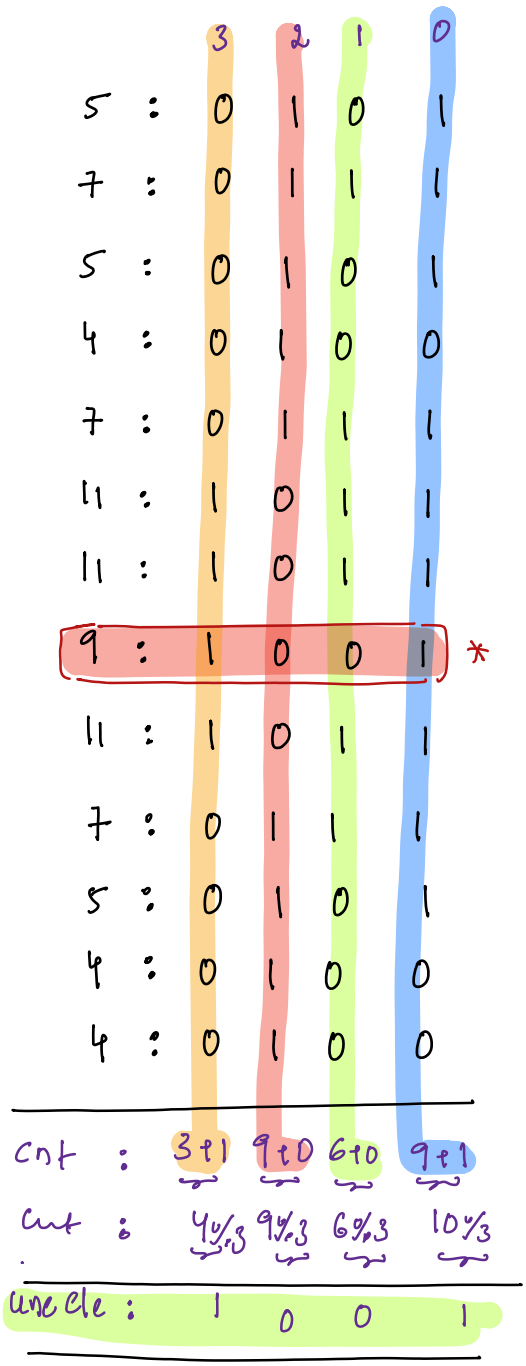
c) XOR of all elements:

ar[10] = 6 ^ 6 ^ 7 ^ 9 ^ 7 ^ 6 ^ 9 ^ 9 ^ 7 ^ 8

= 9 ^ 6 ^ 7 ^ 8 } We cannot get unique element

Idea3:

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



obs: Iterate & count no: of set bits in i^{th} bit position = c

c%3 == 1 : i^{th} bit in unique is set

else : i^{th} bit in unique is unset

Pseudo Code:

TC: 32 * N → O(N)

SC: O(1)

```
int TripleTrouble(int arr[], int n){
    int unq = 0
    for(i=0; i<32; i++) // ith bit pos
    {
        // No: of element in arr[] with ith bit set
        c = 0
        for(j=0; j<n; j++){
            if(checkBit(arr[j], i)) { c = c+1 }
        }
        if( c%3 == 1) { // ith bit in unique element = 1
            unq = unq + (1<<i) }
    }
    return unq;
}
```

Implement

Extention:

Every element repeats thrice except 1 element repeats 2 time

Only change if(c%3 == 2)

→ Every element repeats 4 times

→ Except 1 element, repeat 1 time
↳ xor of all elements

→ Except 1 element, repeat 2 time
↳ xor of all element *
↳ for above code c%4 == 2

→ Except 1 element, repeat 3 time
↳ xor of all elements

Unique Element - [1]

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

Eg:

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

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

Ideas:

a) Get freq of every element using loop

TC: O(N²) SC: O(1)

Optimize: Get freq of every element using hashmap

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

b) Sort arr[], iterate & get freq

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

c) *OR of all elements

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

xr = 6 ^ 8 = 14 → How can we get 2 elements?

Obs: xr of both unique elements

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

nr of elements

→ 11 ^ 17

	4	3	2	1	0
17:	1	0	0	0	1
11:	0	1	0	1	1
17^11:	1	1	0	1	0

Split arr[], based on data on 1st bit

Set

Ele with 1st bit set

10 6 11 10 6

nr of all ele = 11

Unset

Ele with 1st bit Unset

8 8 9 12 9

12 17

nr of all ele = 17

Obs: In 17^11 1st bit set?

At 1st bit both unique elements have different bits

Split arr[], based on data on 3rd bit

Set

Ele with 3rd bit set

10 8 8 9 12 9

11 10 12

nr of all ele: 11

Unset

Ele with 3rd bit Unset

6 6 17

nr of all ele: 17

Obs: In 17^11 3rd bit set?

At 3rd bit both unique elements have different bits

Pseudocode:

```
int UniqueII(int ar[], int n){   TC: O(N)  SC: O(1)

    Step:1 [ int v=0
             i=0; i<n; i++){ v= v^ar[i]}

    Step:2 [ Get set bit pos in v
             pos=0
             i=0; i<32; i++){
                 if(checkbit(v, i)){ pos=i; break}
             }

    Step:3 [ // Split arr[] based on pos bit
             int set=0, unset=0
             i=0; i<n; i++){
                 if(checkbit(ar[i], pos)){
                     set = set^ar[i]
                 }
                 else{ unset = unset^ar[i] }
             }

    Step:4 [ 2 unique elements are set & unset]
}
```

3Q)

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

↳ {1-6}
ar[4] = { 3 6 1 4 } → { 2, 5 }

↳ {1-7}
ar[5] = { 1 6 4 7 5 } → { 2, 3 }

Ideas:

↳ Hashmap a Srt n bool ch[N+2] or Sending elements to correct positions

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

↳ N=5, contain [1, N+2] = {1-7}

ar[5] = { 1 6 4 7 5 } { 1 2 3 4 5 6 7 }

Obs: In complete data

→ Every element repeating twice, except unique elements

↳ 2 missing elements

Pseudocode : TODO/ Doubts TC: O(N) SC: O(1)

Step1: v = 0

```
for(i=0; i<n; i++){
    v = v^ar[i]
}
for(i=1; i<=(n+2); i++){
    v = v^i
}
// v is xor of all elements
```

Step2: pos = 0

```
for(i=0; i<32; i++){
    if(checkbit(v, i)){ pos = i; break;}
}
```

Step3: int set = 0, unset = 0

```
for(i=0; i<n; i++){
    if(checkbit(ar[i], pos)){
        set = set^ar[i]
    }
    else{ unset = unset^ar[i]}
}
for(i=1; i<=(n+2); i++){
    if(checkbit(i, pos)
        set = set^i;
    }
    else{ unset = unset^i;}
}
```

Step:4 Set & unset are 2 missing elements

Unique Element - [1] [2.0] TC: O(N) SC: O(1)

Given ar[N], every element repeats even except 2 elements

find the 2 unique elements which occur odd times

ar[10] = { 8 2 6 8 6 2 6 9 7 9 }

Idea: xor of all ele = 8 ^ 2 ^ 6 ^ 8 ^ 6 ^ 2 ^ 6 ^ 9 ^ 7 ^ 9 = 6 ^ 7

= xor of 2 elements

↳ Split based on Set bit & take xor

Given ar[N], it contains all elements [1, N] except 2 elements such that 1 element is missing 1 is repeating find both

[1 - 6]

ar[6] = { 3 5 1 4 6 5 } = { 2 (missing) 5 (repeating) }

[1 - 5]

ar[5] = { 2 1 3 3 5 } = { 4 (missing) 3 (repeating) }

Ideas:

↳ Hashmap / Sort / 2 - equations / Bring element in corr position

Bits:

N = 6 → [it should all Elements 1 - N]

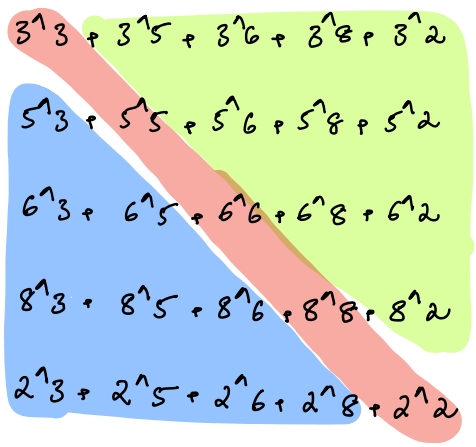
ar[6] = { 3 5 1 4 6 5 } { 1 2 3 4 5 6 }

Consider complete data, every element repeats twice except 2 elements → { 5 repeats 3 2 repeats 1 }

4Q) Given ar[N], calulate sum of xor of all pairs ?

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

All pairs:



Ideal: sum of xor of all pairs

```
s=0
for(i=0; i<n; i++){
    for(j=0; j<n; j++){
        s = s + ar[i]^ar[j]
    }
}
```

TC: O(N2) SC: O(1)

Idea2: Iterate only on upper or lower triangle

```
for(i=0; i<n; i++){
    for(j=i+1; j<n; j++){
        s = s + ar[i]^ar[j]
    }
}
return 2*s
```

TC: O(N2) SC: O(1)

	3	2	1	0
2 :	0	0	1	0
3 :	0	0	1	1
5 :	0	1	0	1
6 :	0	1	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
	4	6	6	6
S =	*	*	*	*
	2^3	2^2	2^1	2^0
	=	=	=	=

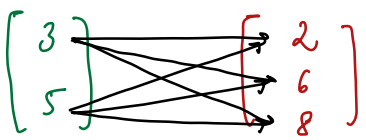
S = 32 + 24 + 12 + 6 = 74

return 2S = 148

obs: if both bits are different its xor is 1

In arr[] how many elem 0th bit

Set — 2 | Unset — 3



In arr[] in how many xor pairs 0th bit set = 6 pairs = $6 * 2^0 = 6$

In arr[] how many elem 1th bit

Set — 3 | Unset — 2

In arr[] in how many xor pairs 1st bit set = 6 pairs = $6 * 2^1 = 12$

In arr[] how many elem 2th bit

Set — 2 | Unset — 3

In arr[] in how many xor pairs 2nd bit set = 6 pairs = $6 * 2^2 = 24$

In arr[] how many elem 3th bit

Set — 1 | Unset — 4

In arr[] in how many xor pairs 1st bit set = 4 pairs = $4 * 2^3 = 32$

Final obs:

Repeat above procedure for all 32 bits:

- Say no: of arr[] with i^{th} bit { Set : C, Unset : N-C }
- # No: of arr pair with i^{th} bit Set = $C * (N-C)$
- # Total Contribution of i^{th} bit = $C * (N-C) * 2^i$

```
int sumPairs(int arr[], int n){        TC: O(32*N) → O(N)
                                       SC: O(1)
    int ans = 0
    for(i=0; i<32; i++) // ith bit pos
    |   // No: of element in arr[] with ith bit set
    |   c = 0
    |   for(j=0; j<n; j++){            → Implement
    |   |   if(checkBit(arr[j], i)) { c = c+1; }
    |   }
    |   ans = ans + c*(n-c)*(1<<i)
    |                    |
    |                    2^i
    }
    return 2*ans;
}
```

$c*(n-c)$: # arr pairs with i^{th} bit set