Todays Content

1. Single Number 1
    - 1a Approach a
    - 1b Approach b
2. Single Number 2
3. Single Number 3

Revise:

1B: Check if i^{th} Bit set in N: (N >> i) & 1 == 1.

2B: Set i^{th} Bit in N: N = N | (1 << i)

3B: a ^ a = 0

Q: Given ar[N] every ele repeats twice except 1, return unique ele.

Eg: ar[] = { 2 3 5 6 3 6 2 } ans = 5

Eg: ar[] = { 7 6 7 9 9 } ans = 6

Ideal: Xor of all ar[] elements & return unique element

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

Ideal: Write Binary of all elements

	0	1	2	3	4	5	6
Eg: ar[] = {	2	3	5	6	3	6	2 } ans =

	2	1	0
2 :	0	1	0
3 :	0	1	1
5 :	1	0	1
6 :	1	1	0
3 :	0	1	1
6 :	1	1	0
2 :	0	1	0
cnt :	3	6	3
	odd	even	odd

#Obs: If No Unique element:

At every bit:

Number of array elements with bit position is set is even number.

Repeat for all bit positions

#Idea:

In unique = 0^{th} bit = 1

In unique = 1^{st} bit = 0

In unique = 2^{nd} bit = 1

	2	1	0
Unique :	~~X~~	~~X~~	~~X~~
5 :	1	0	1

uni = 0;

For every bit pos i : {0,..31}

Iterate & calculate how many ar[] elements have i^{th} bit as set; c = 0

if [c%2 == 1) { In unique ele i^{th} bit is set }

```
int SingleNumber(vector<int>& ar) { TC: O(32*N) ≈ O(N)  SC: O(1)
    int unq = 0;
    int N = ar.size();
    for(int i=0; i<32; i++) { # i: bit position.
        # Iterate on ar[] & calculate how many elements have ith bit set
        int c = 0;
        for(int j=0; j<N; j++) {
            if( (ar[j]>>i)&1 == 1) {
                c++;
            }
        }
        # c: Count of ar[] elements where ith bit set.
        if( c%2 == 1) { # In unique el ith bit is set.
            unq = unq | (1<<i)   # Set ith bit in unq
        }
    }
    return unq;
}
```

Q2: Given an arr[], all the elements will occurs thrice but once.

Find the unique element.

Constraints:

$1 <= N <= 10^5$

$0 <= arr[i] <= 10^9$

Ex1: arr[] = { 4 5 5 4 1 6 6 4 5 6 } ans = 1

#Idea1: For every arr[i]:

Iterate in arr[] get freq & check unique or not?

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

#Idea2: Xor of all elements =

xor of all individual elements

From this we cannot extract unique element.

arr[] = { 4 ^ 5 ^ 5 ^ 4 ^ 1 ^ 6 ^ 6 ^ 4 ^ 5 ^ 6 } = 1 ^ 4 ^ 5 ^ 6

#Ideas: Write binary representation of numbers

Eg2: ar[] = {5 7 5 4 7 11 11 9 11 7 5 4 4} ans=

	3	2	1	0
5 :	0	1	0	1
7 :	0	1	1	1
5 :	0	1	0	1
4 :	0	1	0	0
7 :	0	1	1	1
11 :	1	0	1	1
11 :	1	0	1	1
9 :	1	0	0	1
11 :	1	0	1	1
7 :	0	1	1	1
5 :	0	1	0	1
4 :	0	1	0	0
4 :	0	1	0	0

#Obs: If No Unique element:

At every bit:

Number of array elements with bit position is set is multiple of 3

3 9 6 10 → Not multiple of 3 : Unique ele 0th bit = set

→ multiple of 3 : Unique ele 1st bit = Unset

→ multiple of 3 : Unique ele 2nd bit = Unset

→ Not multiple of 3 : Unique ele 3th bit = set

	3	2	1	0
ele =	~~0~~	~~0~~	~~0~~	~~1~~
9 =	1	0	0	1

```
int SingleNumber(vector<int> &ar) { TC: O(32*N) ≈ O(N)  SC: O(1)
    int unq = 0;
    int N = ar.size();
    for(int i=0; i<32; i++) { # i: bit position.
        # Iterate on ar[] & calculate how many elements have ith bit set
        int c = 0;
        for(int j=0; j<N; j++) {
            if( (ar[j]>>i)&1 == 1) {
                c++;
            }
        }

        # c: Count of ar[] elements where ith bit set.
        if( c%3 != 0) { # In unique el ith bit is set.
            unq = unq | (1<<i)   # set ith bit in unq
        }
    }
    return unq;
}
```

Q3: Given an arr[]: all the elements will occurs twice but two elements

Return two unique elements in increasing order

Constraints:

$1 <= N <= 10^5$

$0 <= arr[i] <= 10^9$

ex1: arr[] = {4 5 4 1 6 6 5 2} return {1 2}

ex2: arr[] = {4 9 9 8} return {4 8}

Idea1: for every arr[]:

Iterate in array get freq & check unique or not.

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

Idea2: Xor of all elements = xor of unique elements

arr[] = {4 ^ 5 ^ 4 ^ 1 ^ 6 ^ 6 ^ 5 ^ 2} = 1 ^ 2 = 3

a ^ b = 3.

↳ Hard to get both unique elements

#Idea3: Xor of all elements

1010 1000 1100 0110 1010 1100

arr[] = { 10 ^ 8 ^ 8 ^ 9 ^ 12 ^ 9 ^ 6 ^ 11 ^ 10 ^ 6 ^ 12 ^ 17 }

→ 1000 → 1001 → 1001 → 1011 → 0110 → 10001

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

11 : 0 1 0 1 1

17 : 1 0 0 0 1

val : 1 1 0 1 0

Obs:

In val 1st bit = Set

Both unique ele are diff at 1st bit.

#Hint: Split arr[] based on 1st bit info

1st bit set → [10 6 11 10 6] Xor of Set = 11

1st bit unset → [8 8 9 12 9 12 17] Xor of unset = 17

In val 3rd bit = Set

Both unique ele are diff at 3rd bit.

#Hint: Split arr[] based on 3rd bit info

3rd bit set → [10 8 8 9 12 9 11 10 12] Xor of Set = 11

3rd bit unset → [6 6 17] Xor of unset = 17

#Steps:

1. Calculate xor of all arr[] elements.
2. Get any set bit position in xor value; = p.
3. Split entire arr[] based on bit position = p

Set → Calculate xor : Unq1

Unset → Calculate xor : Unq2

4. Add both in vector & return.

```
vector<int> SingleNumber(vector<int> &ar){   TC: O(N+32+N+1) = O(N)
    int N = ar.size();                       SC: O(1)
    int xr = 0;
    for(int i=0; i<N; i++){
        xr = xr ^ ar[i];
    }

    int p=0;
    for(int i=0; i<32; i++){
        if( (xr>>i)&1 == 1){ # xr ith bit set
            p=i; break;
        }
    }

    int set=0, unset=0;
    for(int i=0; i<N; i++){
        if( (ar[i]>>p)&1 == 1){ # ar[i] goes to set
            set = set ^ ar[i];
        }else{
            unset = unset ^ ar[i];
        }
    }

    vector<int> v(2);
    if(set<unset){
        v[0]=set; v[1]=unset;
    }else{
        v[0]=unset; v[1]=set;
    }
    return v;
}
```