Todays Content

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

Revise:

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

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

3Q: 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

Idea1: Calculate XOR of all elements & return unique element.

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

Idea2: Write all ar[] numbers in their binary

	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 elements:
For every bit pos:
Total number of set bit is even

#Repeat proves for all bits.

In unique ele 0^{th} bit = 1.
In unique ele 1^{st} bit = 0
In unique ele 2^{nd} bit = 1

	2	1	0
Unique :	⊗	0	⊗
	1	0	1 = 5

#Idea2:

For every bit pos p: 0 to 31.

Iterate & calculate count of ar[] elements with p^{th} bit is set ; = c.

if (c%2==1) { for that bit position: p unique ele 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++){
        # for bit pos: i, calculate no: of elements with ith bit set;
        int c=0;
        for(int j=0; j<N; j++){
            if((ar[j]>>i)&1==1){
                c++;
            }
        }
        if(c%2==1){ # Unique element ith bit is set
            unq = unq | (1<<i);
        }
    }
    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 frequency & check unique or not?

TC: O(N²) SC: O(1)

Issu: TLE

#Idea2: Take xor of all elements

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

Issu: Final value is xor of all elements, we cannot get unique element.

Ideas:

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

Not multiple of : Unique ele 0th bit = set

4 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 3rd bit = set

ele :	3	2	1	0
	~~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++){
        # for bit pos: i, calculate no: of elements with i^th bit set;
        int c=0;
        for(int j=0; j<N; j++) {
            if((ar[j]>>i)&1 == 1){
                c++;
            }
        }
        if( c%3 != 0) { # Unique element i^th bit is set
            unq = unq | (1<<i);
        }
    }
    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} = {1, 2}

Ex2: arr[] = {4 9 9 8} = {4, 8}

Idea1: For every arr[]:

Iterate in arr[], get freq & check if it's unique or not.

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

Issue: TLE

Idea2: Calculate xor of all elements = xor of unique elements

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

a ^ b = 3

↳ We cannot estimate a, b based on their xor value.

#Idea3: Xor of all elements

1010 1000 1100 0110 1010 1100

arr[] = { 10 → 1000 ^ 8 → 1000 ^ 8 → 1001 ^ 9 → 1001 ^ 12 → 1100 ^ 9 → 1001 ^ 6 → 0110 ^ 11 → 1011 ^ 10 → 1010 ^ 6 → 0110 ^ 12 → 1100 ^ 17 → 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 i^{th} bit = Set

Both unique ele are diff at i^{th} bit

#Hint: Split arr[] based on i^{th} bit Information

i^{th} bit Set

[10 6 11 10 6

Xor of set = 11]

i^{th} bit Unset

[8 8 9 12 9 12 17

Xor of Unset = 17]

In val 3^{rd} bit = Set

Both unique ele are diff at 3^{rd} bit

#Hint: Split arr[] based on 3^{rd} bit Information

3^{rd} bit Set

[10 8 8 9 12

9 11 10 12

Xor of set = 11]

3^{rd} bit Unset

[6 6 17

Xor of Unset = 17]

#Steps:

1. Calculate xor of all elements; # xor of 2 unique elements.
2. Get a set bit position in xor value; # At that set bit both unique ele diff
3. On that set bit position split entire arr() into set & unset
4. Calculate xor at set & unset category & return 2 unique elements.

```
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;
}
```