Ques single no. 1

We are given an integer array where every number occurs twice except for one number which occurs just once. Find that number.

ex [4, 5, 5, 4, 1, 6, 6] → Ans 1.

[7, 5, 5, 1, 7, 6, 1, 6, 4] → Ans → 4.

Value of 120 ^ 5 ^ 6 ^ 6 ^ 120 ^ 5 is -

120 ^ 120 ^ 6 ^ 6 ^ 5 ^ 5 ⇒ 0.

Soln → 1

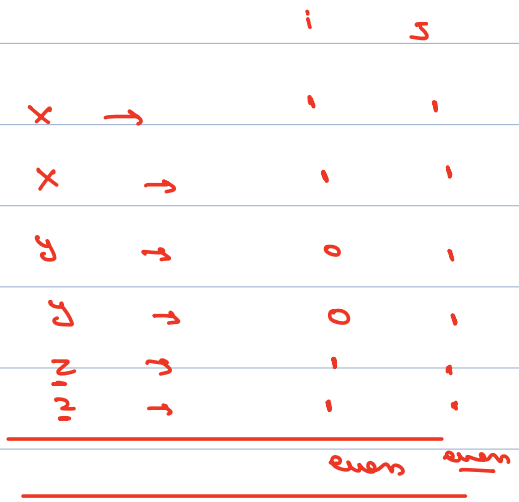
```
int x = 0;

for (int i = 0; i < arr.size(); ++i) {
    x = x ^ arr[i]; // XOR operation
}

print(x);
```

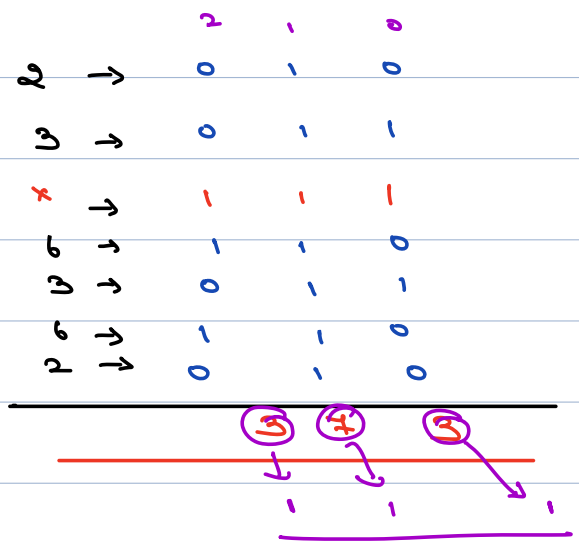
T.C → O(n)
S.C → O(1).

Soln → 2

A = [2, 3, 5, 6, 3, 6, 2]

	2	1	0	
2 →	0	1	0	
3 →	0	1	1	
5 →	1	0	1	x
6 →	1	1	0	
3 →	0	1	1	
6 →	1	1	0	
2 →	0	1	0	
	3 → 1	6 → 0	3 → 1	→ 5 only.





```
int ans = 0;
for (i=0; i<31; i++) {
    int cnt=0;
    for (j=0; j<n; j++) {
        if (checkBit(arr[j], i)) {
            cnt++
        }
    }
    if (cnt%3 != 0) {   //if count is not divisible by 3;
        ans = ans | (1<<i)   //setting ith bit in ans;
    }
}
print(ans);
```

T.C → O(n)

S.C → O(1)

Set a bit.

Ques) single number 2 :-

Given an integer array, all the elements will occur thrice but one. Find the unique element.

Input: [4, 5, 5, 4, 1, 6, 6, 4, 5, 6] Ans → 1

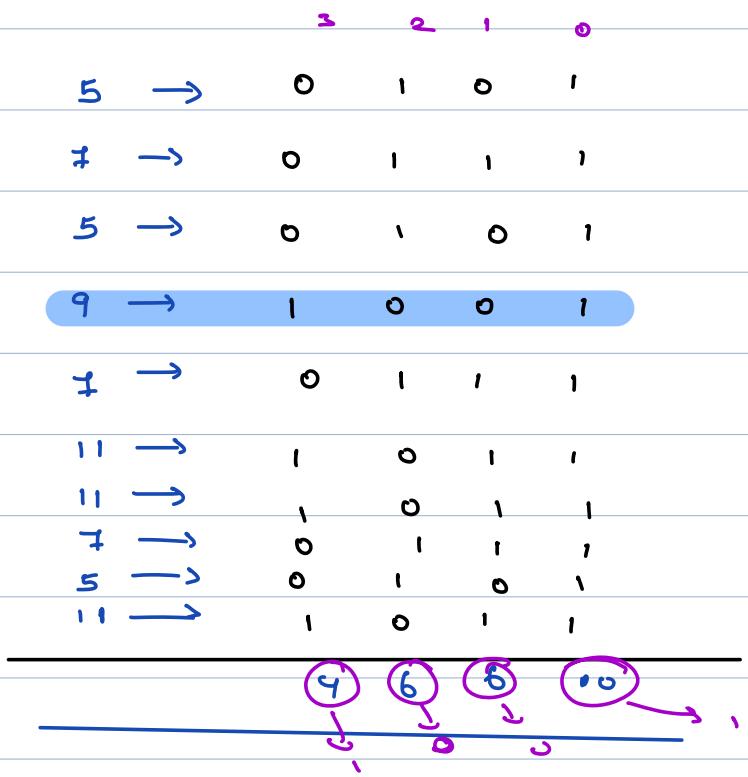
Brute force :-

two for loops. T.C → O(n²)
S.C → O(1)

Approach-2

HashMap → T.C → O(n)
S.C → O(n).

Approach 3 :- [5, 7, 5, 9, 7, 11, 11, 7, 5, 11]

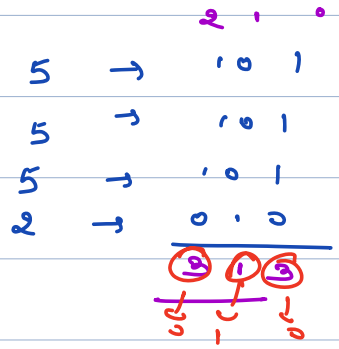


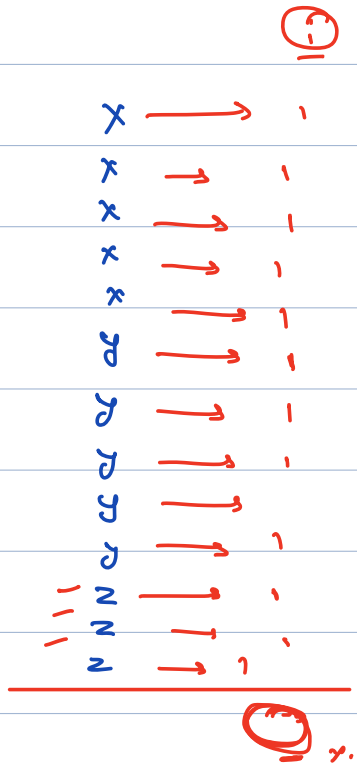
```
int ans = 0;
for (i=0; i<31; i++) {
    int cnt=0;
    for (j=0; j<n; j++) {
        if (checkBit(arr[j], i)) {
            cnt++
        }
    }
    if (cnt%3==1) {
        ans = ans | 1<<i;
    }
}
print(ans);
```

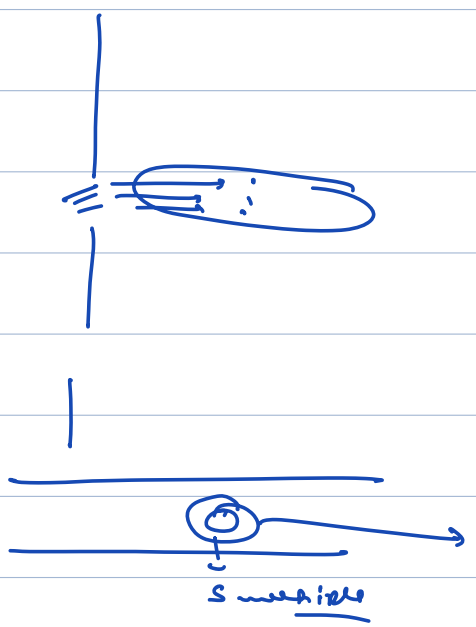
T.C → O(n)

S.C → O(1);

Every no' is coming 3 times, except 1 unique no', which is coming 2 times.







8:03 Am - 8:13 Am

Ques Single Number 3

unique.

Given an integer array, all the elements will occur twice except two. Find those two elements.

Input: [4,5,3,4,1,6,5,6,3]

Input → [3, 6, 4, 4, 3, 8] → 6, 8

[4, 9, 9, 8] → 4, 8

arr[] → 3, 7, 6, 7, 3, 8, 9, 9 → 6, 8.

idea → Take xor, → 14.

Take xor → 7, (2, 5), (1, 6), (3, 4)

arr[12] →

10	8	8	9	12	9	6	11	10	6	12	17
(1010)	(1000)	(1000)	(1001)	(1100)	(1001)	(0110)	(1011)	(1010)	(0110)	(1100)	(10001)

3 2 1 0

11 → 0 1 0 1 1

17 → 1 0 0 0 1

11^17 → 1 1 0 1 0

Split my array based on 1st bit

y=0 set

y=1 unset

10, 6, 11, 10, 6

8, 8, 9, 12, 9, 12, 17

divide based on 3rd bit :-

set

unset

10, 10, 8, 8, 9, 9, 12, 12, 11,

6, 6, 17

Pseudo code

1) `V = 0;` // Take xor of all.

```
for (i = 0; i < n; i++) {
    V ^= arr[i]
}
```

2) from V get a set bit pos.

```
for (i = 0; i < 31; i++) {
    if (checkBit(V, i)) {
        pos = i;
        break;
    }
}
```

3) Split array based on this pos into set and unset

```
set = 0; unset = 0;
for i -> 0 to n-1:
    if (checkBit(arr[i], pos)) {
        set = set ^ arr[i];
    else:
        unset = unset ^ arr[i];
}
print(set + " " + unset);
```

T.C → O(n)

S.C → O(1)

Ques) Maximum And Pair

Given N array elements, choose two indices(i, j) such that **(i != j)** and **(arr[i] & arr[j])** is maximum.

(5, 4, 6, 8, 5) → (0,4) is Maxm.

(27, 18, 20)

(27, 18) → 18

(27, 20) → 16

(18, 20) → 16

e.g2)

arr[5] = {21, 18, 24, 17, 16}

→ (21, 17)

21 : 1 0 1 0 1

18 : 1 0 0 1 0

24 : 1 1 0 0 0

17 : 1 0 0 0 1

16 : 1 0 0 0 0

z → 1 0 0 0 0

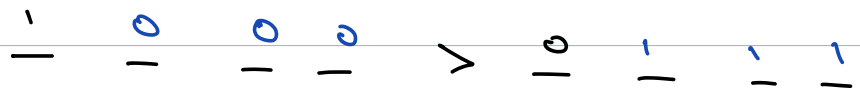
x → 1 0 1 1 1

y → 0 0 1 1 1

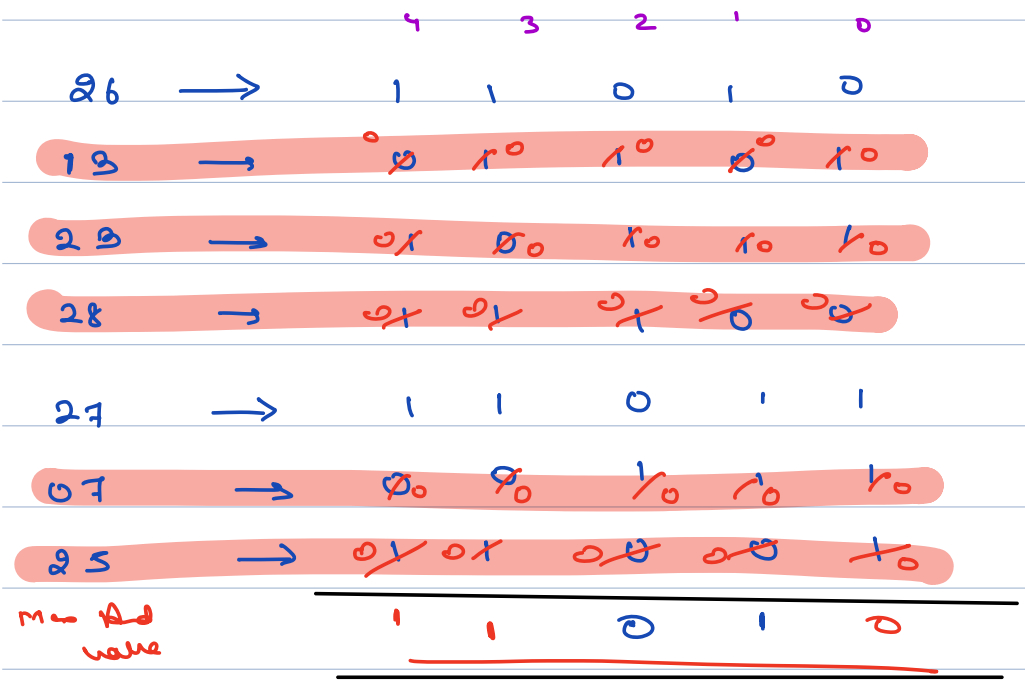
→ 1 0 0 0 0

0 0 1 1 1

idea → having more no. of set bits at same place ✗.



arr[] = { 26, 19, 23, 28, 27, 7, 25 }



```
ans = 0;
for (i=30; i>=0; i--) {
    //find count of set bits at this place
    int c=0;
    for (j=0; j<n; j++) {
        if (checkBit (arr[j], i)) {
            c++;
        }
    }
    if (c>=2) {
        // we can form a pair whose and
        will have a 1 at this place.
        ans = ans | (1<<i)  (adding bit)
        remove unmarked elems.
        for (j=0; j<n; j++) {
            if (checkBit (arr[j], i) == false)
                arr[j]=0;
        }
    }
}
```

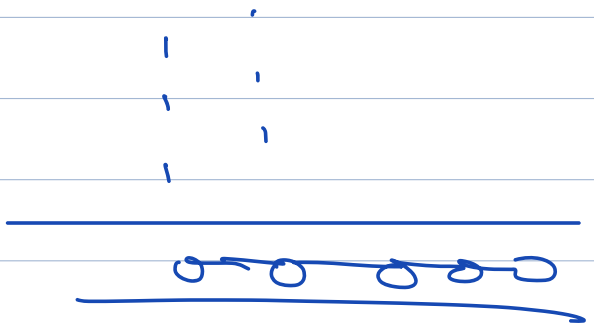
T.C → O(n)

S.C → O(1)

Ques) find count of pairs whose bitwise and is max.

↳ brand the array find arr[i,j] > 0,

as well the $\frac{n \times (n-1)}{2}$



B set bits Count bits.

3 set bits 4 unset bits

↓

(111) << 4.

3 set bits

= 111 → 7

1000 → 8

1111 → 15

1 0000 ← 16

(1<<3) − 1

↓

1000

−1 → 111.

4 set bits.

1<<4 1 0000 − 1

1111

B set bits

((1<<B) − 1) << C.

(1 0000 − 1)

Bits ~

1 1 1 1 ~ 1 Bits