Lecture :- Bit Manipulation 1

Agenda

- single element ✓ Easy
- Single element 2 ✓ Easy - Medium
- Single element 3 ✓ Medium - hard
- Max and pair ✓ Medium.

Revision

a	b	[0 dominates] a & b	[1 dominates] a | b	[same - 0 diff - 1] a ^ b
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Left shift

$$a << n = a * 2^n$$

Right shift

$$a >> n = \frac{a}{2^n}$$

Check if kth bit is set or not?

Eg: n = 12, k = 2. = true.

if n & (1 << k) == 0 kth bit is 0

≠ 0 kth bit is 1

Qu **Single element**

Given arr[n]. Every el appears twice but only one element appears once. find that unique element

Eg: arr: [1 . 2 . 3 . 2 , 3 . 4 , 1 , 5 . 4]

Take xor of array. = ans

Qu **single element 2**

Given arr[n]. Every el **appears thrice** but only one element appears once. find that unique element.

Eg: arr: [1 2 4 3 3 2 2 3 1 1]

arr: [0 0 1 0]

Approach 1: Hashmap TC: O(n)

SC: O(n)

Approach 2 Expected TC: O(n)

SC: O(1)

XOR approach: [does not work]

arr: [1 2 4 3 3 2 2 3 1 1]

xor = 1 ^ 2 ^ 4 ^ ~~3~~ ^ ~~3~~ ^ ~~2~~ ^ ~~2~~ ^ 3 ^ ~~1~~ ^ ~~1~~

xor = 1 ^ 2 ^ 4 ^ 3 = x Ans.

Xor does not work coz frequency of each el in array is odd.

~~x~~ ^ ~~x~~ ^ ~~x~~ ^ ~~x~~ = 0

~~x~~ ^ ~~x~~ ^ ~~x~~ ^ ~~x~~ ^ 1 = 1.

arr: [1 2 4 3 3 2 2 3 1 1]

(2 1 0)

1 — 0 0 1

2 — 0 1 0

4 — 1 0 0

3 — 0 1 1

3 — 0 1 1

2 — 0 1 0

2 — 0 1 0

3 — 0 1 1

1 — 0 0 1

1 — 0 0 1

---

1 One 6 0 6 ones

9 z 4 z 4 zeroes

1 0 0 = 4

— 6 ones: [1, 3, 3, 3, 1, 1]

not a mul of 3 ⇐ 4 zeroes: [2, 4, 2, 2]

0 0 0 ⟹ same no

0 ⟹ diff / unique no

---

arr: [10 10 10]

(3 2 1 0)

1 0 1 0

1 0 1 0

1 0 1 0

3 1's 0 1's 3 1's 0 1's

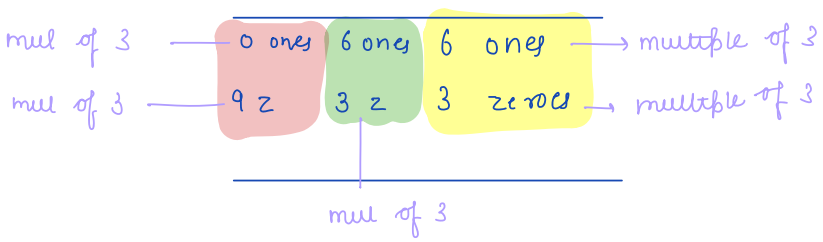
0 0's 3 0's 0 0's 3 0's

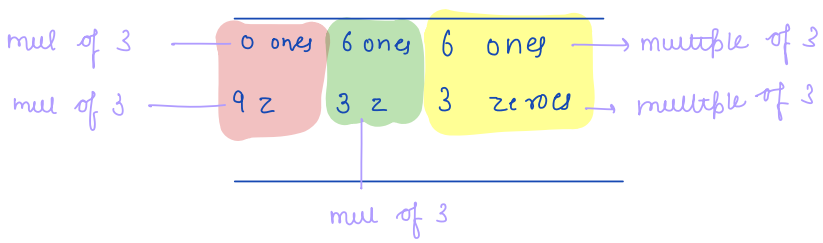
arr : [1 2 3 3 2 2 3 1 1]

(2 1 0)

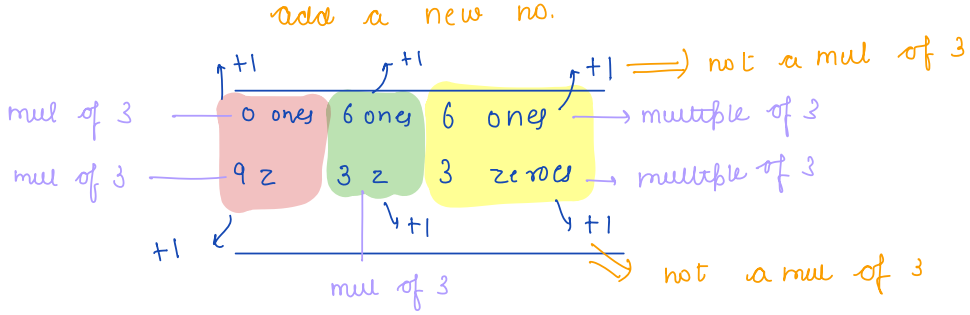
→ all el is coming thrice

1	0	0	1
2	0	1	0
3	0	1	1
3	0	1	1
2	0	1	0
2	0	1	0
3	0	1	1
1	0	0	1
1	0	0	1





add a new no.



Algo: if (count of 1's is multiple of 3) {

unique no. at that idx, it value will be 0.

}

if (count of 0's is multiple of 3) {

unique no. at that idx, it value will be 1.

}

```
int singleElement2 ( int [] arr) {
    int unique = 0;
    for ( i = 31; i >= 0; i--) {
        int ones = 0;
        for( int el: arr) {
            int bitvalue = el & ( 1 << i);
            if ( bitvalue != 0) {
                ones++;
            }
        }
        if (ones % 3 == 0) {
            // At ith bit idx, value should be 0;
            // Do nothing
        } else {
            // At ith idx, value == 1
            unique += Math.pow(2, i);
        }
    }
    return unique;
}
```

TC: $O(32 * n) \simeq O(n)$

SC: $O(1)$

Qu2 **Single element 2** [Amazon, Uber]

Given arr[n]. all el appear twice.

two integers appear once.

find those two unique no.

arr[] = [1 2 3 1 2 4] ans = 3, 4.

arr: [1, 2] ans = 1, 2

Approach: XOR

arr: [1 2 3 1 2 4]

xor = 1 ^ 2 ^ 3 ^ 1 ^ 2 ^ 4

xor = 3 ^ 4. = 7.

```
  0 1 1
^ 1 0 0
-------
  1 1 1
-------
```

Let's say x and y are unique.

xor = x ^ y.

RSB mask: Right most set bit mask.

Eg: 57 : 1 1 1 0 0 1

vans: 0 0 0 0 0 1

RSb (57) = 1

Eg: 76 : 1 0 0 1 1 0 0

vans: 0 0 0 0 1 0 0

$rsb(x) = x \,\&\, x''$

2's complement =

1's complement +1

Dry run: $x = 72$.

72 : 1 0 0 1 0 0 0

$(72)'$ 0 1 1 0 1 1 1

\+ 1

0 1 1 1 0 0 0

72 : 1 0 0 1 0 0 0

$(72)''$: & 0 1 1 1 0 0 0

0 0 0 1 0 0 0

Single element 3

36 :	1	0	0	1	0	0
50 :	1	1	0	0	1	0
24 :	0	1	1	0	0	0
56 :	1	1	1	0	0	0
36 :	1	0	0	1	0	0
24 :	0	1	1	0	0	0
42 :	1	0	1	0	1	0
^ 50 :	^ 1	1	0	0	1	0
56 ^ 42	0	1	0	0	1	0

	5	4	3	2nd	1st	0th
56 ^ 42 =	0	1	0	0	1	0

↓

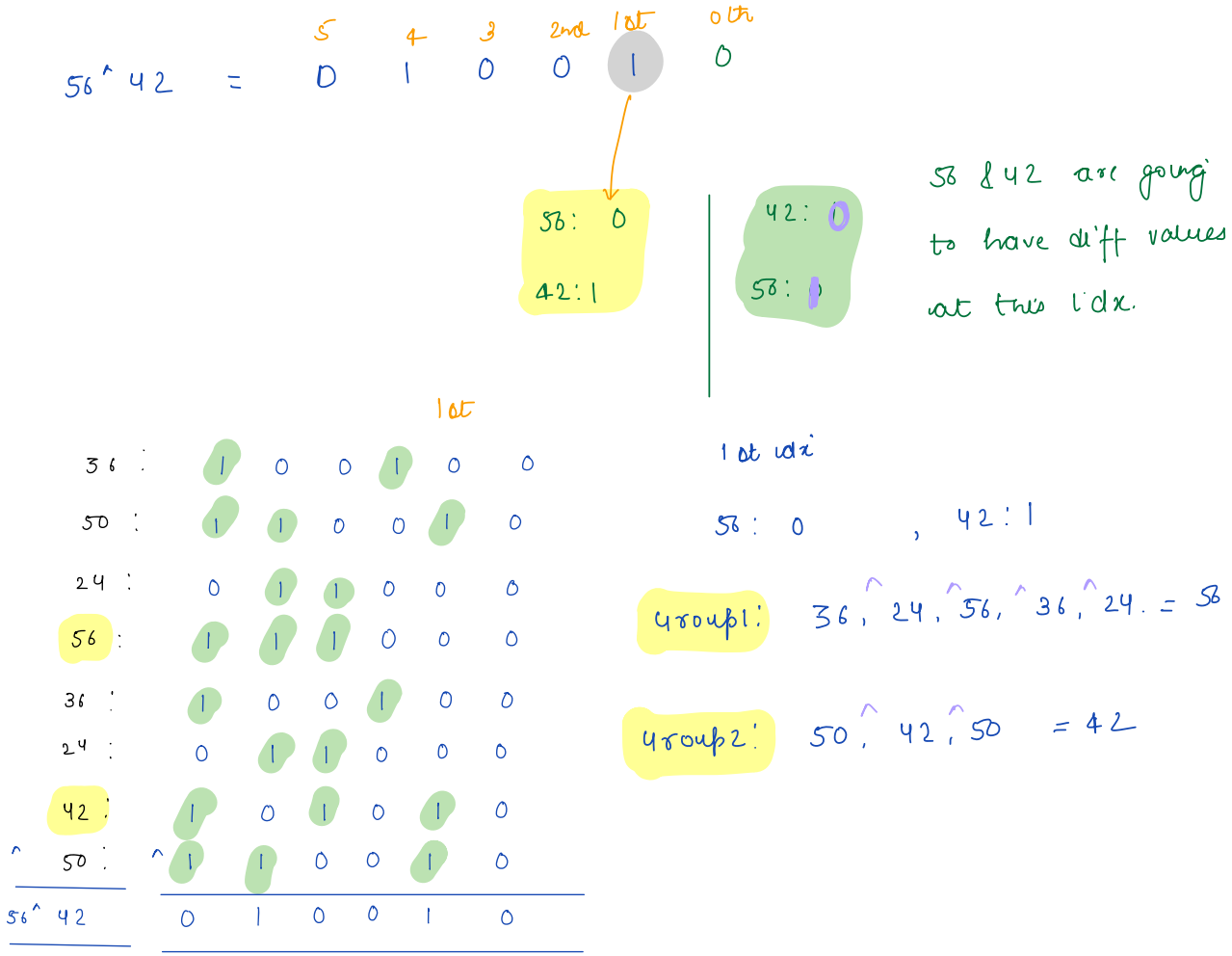
bit value of 56 & 42 would have been same

final xor: 0 1 0 0 1 0

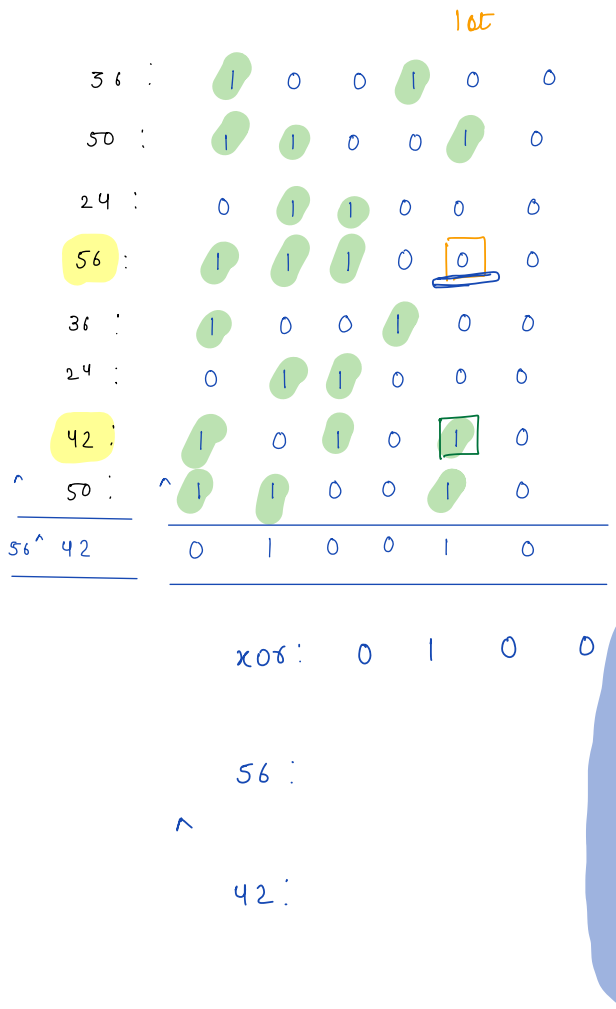
of 56 & 42.

	0	0	0	0	0	1
56 :	1	0	0	1	0	0
42 :	0	1	0	0	1	0
	1	1	or 1	1	1	0
	1	0	1	1	0	0

2 groups.

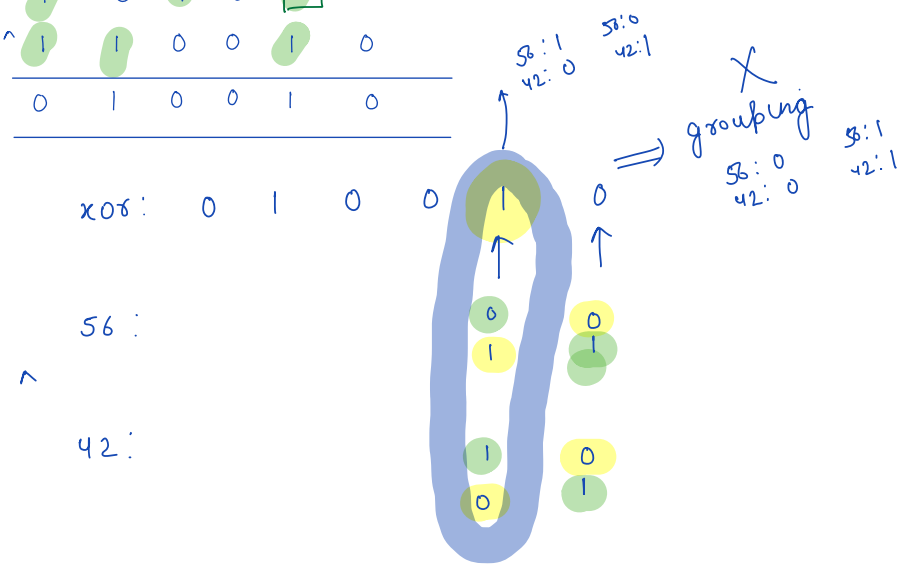

5 4 3 2nd 1st 0th
56 ^ 42 = 0 1 0 0 1 0
56: 0
42: 1
42: 0
56: 1
56 & 42 are going to have diff values at this idx.
1st
36 : 1 0 0 1 0 0
50 : 1 1 0 0 1 0
24 : 0 1 1 0 0 0
56 : 1 1 1 0 0 0
36 : 1 0 0 1 0 0
24 : 0 1 1 0 0 0
42 : 1 0 1 0 1 0
^ 50 : ^ 1 1 0 0 1 0
56 ^ 42 0 1 0 0 1 0
1st idx
56: 0 , 42: 1
group1: 36 ^ 24 ^ 56 ^ 36 ^ 24 = 56
group2: 50 ^ 42 ^ 50 = 42

Doubt:



Group:1 $36 \wedge 24 \wedge 58 \wedge 36 \wedge 24 = 58$

Group2: $50 \wedge 42 \wedge 50 = 42$



Algorithmic:

	lot					
36 :	1	0	0	1	0	0
50 :	1	1	0	0	1	0
24 :	0	1	1	0	0	0
56 :	1	1	1	0	0	0
36 :	1	0	0	1	0	0
24 :	0	1	1	0	0	0
42 :	1	0	1	0	1	0
^ 50 :	^ 1	1	0	0	1	0
56 ^ 42	0	1	0	0	1	0

Let's say x & y are unique.

1. xor of all array

   xor = x ^ y.

2. Grouping:

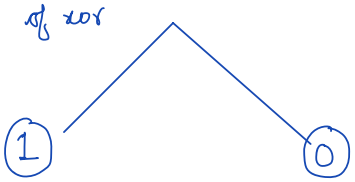
   xor = 56 ^ 42

   = 0 1 0 0 1 0

   Rsb = 0 0 0 0 1 0

   of xor

rsb & el of array of xor



```
void singleElement3(int[] arr){
    int xxory = 0;
    for(int val: arr){
        xxory ^= val;
    }
    int rsb = xxory & twoComplement(xxory);
                xxory & -xxory.
    int x = 0;
    int y = 0;
    for(int val: arr){
        if((val & rsb) == 0){
            x = x ^ val;
        } else{
            y = y ^ val;
        }
    }
    print(x);
    print(y);
}
```

$x'' = -x$ H/w

TC: O(n)

SC: O(1)

Break: 9:32 AM

Qu Max and pair

Given arr[n]. find the pair with max and.

Eg: arr[] = [1 . 2 , 3 , 4 , 5]

ans = 4 & 5 = 4

arr[] = [15, 6 . 3 , 1 . 5]

ans = 6 → 6 & 15

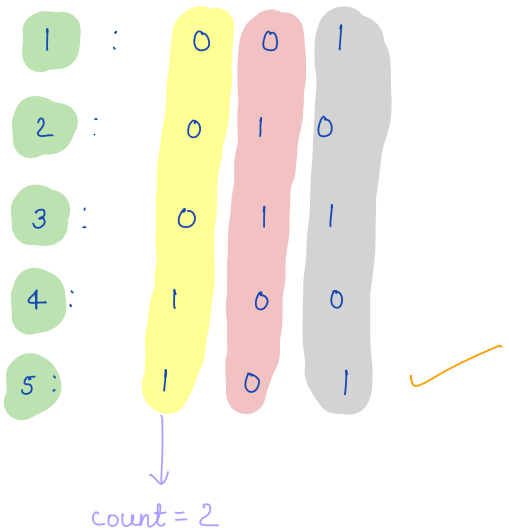
Brute force:

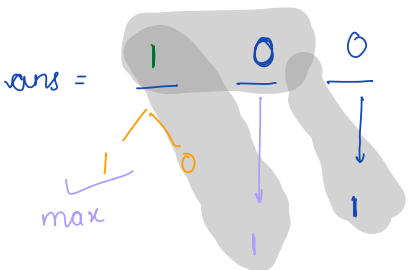
```
int maxAndPair(int[] arr) {
    int n = arr.length;
    int ans = -∞;
    for(i=0; i<n; i++) {
        for(j = i+1; j<n; j++) {
            int and = arr[i] & arr[j];
            ans = max(ans, and);
        }
    }
    return ans;
}
```

Approach 2: TC: O(n)

SC: O(1)

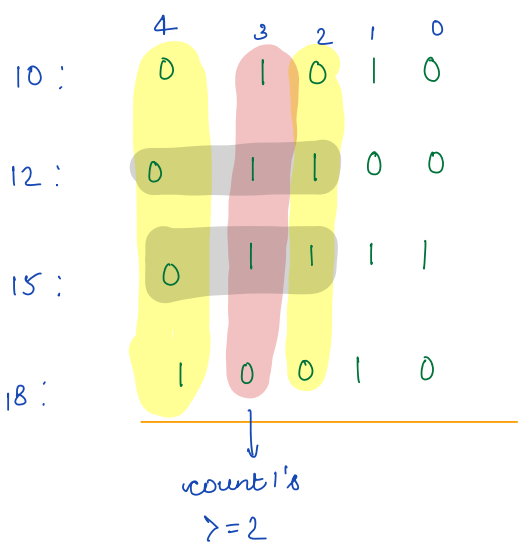
arr[] = [1 2 3 4 5]



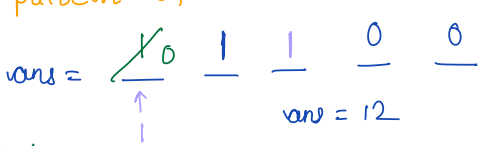


Eg: 2

arr: [10 12 15 18]



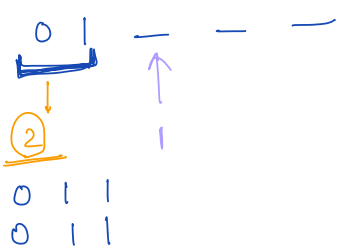
pattern = 0;



i=4 —

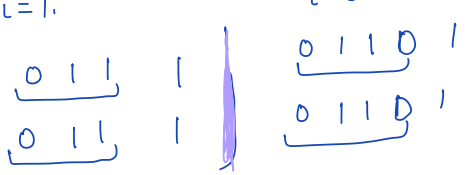
i=3 => pattern = 0

i=2



i=1.

i=0



Code:

```
int maxAndPair(int[] arr) {

    int res = 0;

    for (i = 31; i >= 0; i--) {

        int cnt = 0;                    (1)
        int val = res | (1 << bit);     — Property of left shift

        for(int el : arr) {
(2)         if (val & el == val){
Assignment
                cnt++;
            }
        }

        if (cnt >= 2){
                update your res;        (3)
        }
    }

    return res;
}
```

TC: O(n)

SC: O(1)

Thankyou :)

Doubt:

arr : [1 2 4 3 3 2 2 3 1 1]

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

Added

1 ones | 0 ones | 6 ones | 6 ones

9 z | 3 z | 3 zeros

4 zeros | 4 zeros

1 0 0 = 4

arr

3 : 0 1 1
3 : 0 1 1
3 : 0 1 1
6 : 1 1 0
6 : 1 1 0
6 : 1 1 0

3 0's

3 1's

check for count(1's)

%3 = 0 | %3 ≠ 0

0 | 1

check for cnt(0's)

%3 = 0 | %3 ≠ 0

1 | 0