Today's Content

- → Find 2 unique elements
- → Find 2 missing elements
- → Calculate sum of xor of all pairs
- → Find max A & B

18) Given N Elements every element repeats twice, except 2 unique elements, find the 2 unique elements.

freq = 1

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

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

Idea1: get freq of all Elements
Elements with freq = 1
unique element

Hashmap

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

Search, Insert, Delete, Update — Avg Case O(1)

Idea2: Sort & compare adjacent elements, get unique elements

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

Idea3: XOR of all Elements =

{ 3, 6, 4, 4, 3, 8 } = 6 ^ 8 = 14

```
1000
0110
1110
```

a, b

1) only arr of Elements at worst case.

2) arr of all Elements = arr of 2 unique Elements

(1010) (1000) (1100) (0110) (1010) (1100)

ar[3] : 10 8 8 9 12 9 6 11 10 6 12 17

(1000) (1001) (1001) (1011) (0110) (10001)

xor of all Elements = 11 ^ 17

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

= 26 =

obs:

At bit pos : 1, 3, 4

Both unique elements 11 & 17 have different bits.

Bit pos = 1 (Reference) ? ✓

set → Elements, bit pos=1 set: {10, 6, 11, 10, 6} → xor of all Elements → 11

unset → Elements, bit pos=1 unset: {8, 8, 9, 12, 9, 12, 17} → xor of all Elem → 17

Bit pos = 3 (Reference) ? ✓

set → Elements whose bit pos=3 set: {10, 8, 8, 9, 12, 9, 11, 10, 12} → xor = 11

unset → Elements whose bit pos=3 unset: {6, 6, 17} → 17

Pseudo Code

1) xor of all Elements?

→ N

```
val = 0;
for(i = 0; i < N; i++){
    val = val ^ ar[i]
}
```

2) find set bit pos in val?

↳ [0 30]

→ 31

```
pos = -1
for(i = 0; i <= 30; i++){
    if(checkBit(var, i)){
        pos = i; break;
    }
}
```

3) Using pos, break array into set & unset & xor of all

→ N

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

TC : O(N + 31 + N) → O(N)

SC : O(1)

4) 2 unique Elements → {set & unset}

28) Given N Array Elements, array contains all elements from [1, N+2] except 2 elements. find 2 missing elements
→ [TODO] { Doubt Seeing

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

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

Missing Elements

Ideas:

1) Use hashset & find missing elements → TC: O(N) SC: O(N)

2) Sort & missing elements → TC: O(N log N), SC: O(1)

✓ 3) Use bool ch[N+2], & get missing Elements } TC: N, SC: N

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

Every element is repeating twice except 2 unique Elements.

4th Sol: TC: O(N) SC: O(1)?
↳ Try it out }

1) Step 1: val = {xor of all Elements} ^ {xor of all [1, N+2]}

Todo 1:

```
val = 0;
```

Step 1:

```
for(i=0; i<N; i++){
    val = val ^ ar[i]
}
```

```
for(i=1; i<=(N+2); i++){
    val = val ^ ar[i]
}
```

Step 2:

```
pos = -1
for(i=0; i<30; i++){
    if(checkBit(var, i)){
        pos = i; break;
    }
}
```

Step 3: set = 0, unset = 0 } split entried data

```
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 }
}
```

// finally 2 unique elements are set & unset

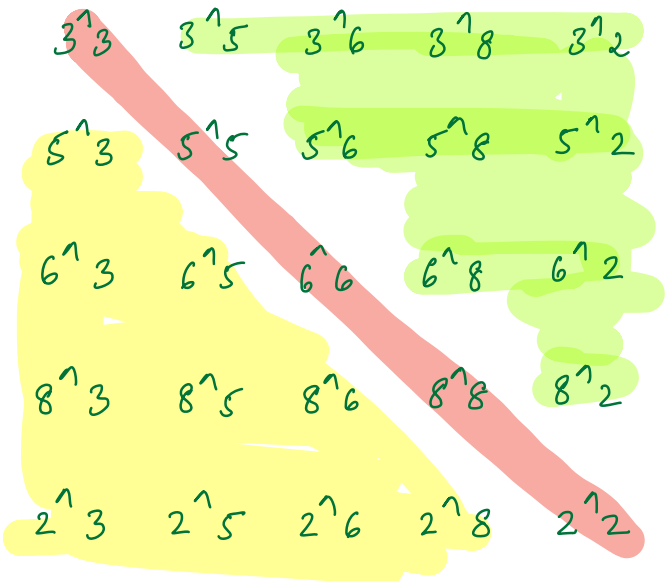
38) Given N Elements. calculate sum of xor of all pairs?

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

Idea1: Sum of all pairs →

```
sum = 0 }  TC: O(N^2)  SC: O(1)
for(i=0; i<N; i++){
    for(j=0; j<N; j++){
        sum = sum + ar[i]^ar[j]
    }
}
```

Idea2:



Idea3: Iterate on upper part & get sum →

ans = 2 * sum

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

// Work it out

Idea: 3 mins

Idea: All pairs with binary representation.

2 : 0010, 3 : 0011, 5 : 0101, 6 : 0110, 8 : 1000

Idea3:

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

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

Idea4: nr: ↦ 0 : same, ↳ 1 : different

In given array, how many Elements 0^{th} bit set = ?

set → {2}, unset → {3} = 6 pairs

In given array Elem 1^{st} set : set 3, unset 2 = 6

In given array Elem 2^{nd} set : set 2, unset 3 = 6

In given array Elem 3^{rd} set : 1, 4 = 4

Idea: At i^{th} pos, C_i number of pairs have i^{th} pos a set →

$$\sum_{i=0}^{30} C_i * 2^i$$

82 + 24 + 12 + 6

74 } TC → 31 * (N²)

→ 2 * 74 → 148 } TC → O(N²), SC → O(1)

$$ans = 2 * \sum_{i=0}^{30} C_i * 2^i$$

Pseudo Code

```
Sum = 0
for(i = 0; i <= 30; i++){
    c = 0
    for(j = 0; j < N; j++){
        if(checkbit(ar[j], i)){ c++ }
    }
    sum = sum + c * (N - c) * (1 << i)
}
return 2*sum
```

TC: 31 * N + O(N) SC: O(1)

11PM: break

Eg

int: 32 bits: [0 31]

31 → MSB bit → -ve value

4Q) Given N Array elements, choose 2 indices i, j such that i ≠ j && [Ar[i] & Ar[j] is max] [bitwise and] → google

indices

Ex1: ar[3] = {27, 18, 20} → {27 & 18} → 18 → and

(27, 18)	(27, 20)	(18, 20)
18 : 10010	27 : 11011	18 : 10010
27 : 11011	20 : 10100	20 : 10100
18 = 10010	16 = 10000	16 ← 10000

Ex2: ar[4] = {21, 18, 24, 17} → {24 21 18 17}

21 : 10101
18 : 10010
24 : 11000
17 : 10001

21 : 10101
17 : 10001
10001 → 17

→ ans = 17

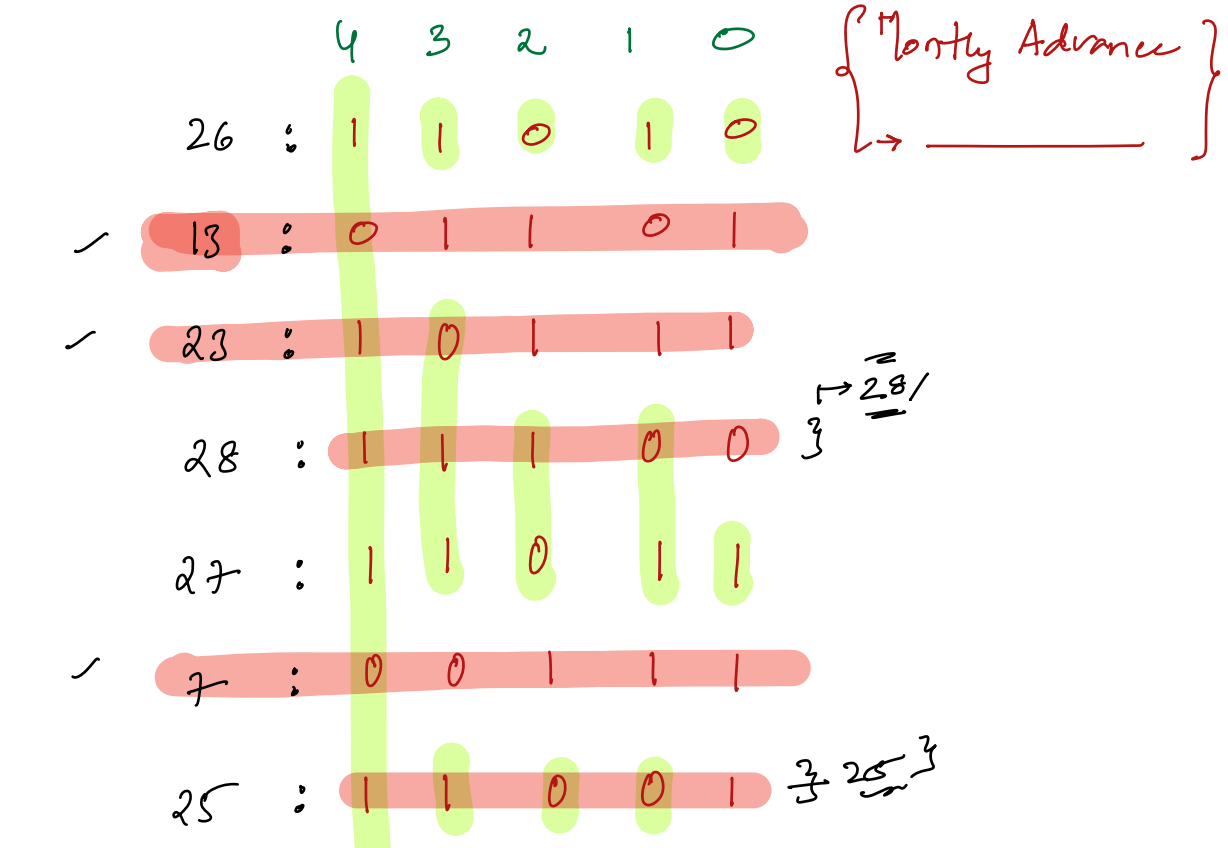
Idea1: Sort, & of adjacent pairs ? ✗

Idea2: Max value can be in & wrt of element ? ✗

Idea3: For every pair get (Bitwise &) overall max

TC: O(N²)

Ex3: arr[7] : { 26, 13, 23, 28, 27, 7, 25 }



Set bits : >=2 >=2 =1 >=2 >=2

find arr

1 1 0 1 1

Remove : 13, 7 23 28, 25

no delete in this can

Pseudo code

```
int ans=0

for(i=30; i>=0; i--){
    c=0;
    for(j=0; j<N; j++){
        if(checkBit(ar[j], i){ c++ }
    }

    if( cnt >= 2) {   // ith bit have set
        ans = ans + (1<<i)  or  ans = ans | (1<<i)
        for(j=0; j<N; j++){
            if( ! checkBit(ar[j], i)){
                ar[j] = 0 }
        } }
}
return ans;
```

checking how many elements have i^{th} bit-set if greater >= 2

→ Remove those elements, whose i^{th} bit is set,

Instead of deleting simply make the Element as = 0

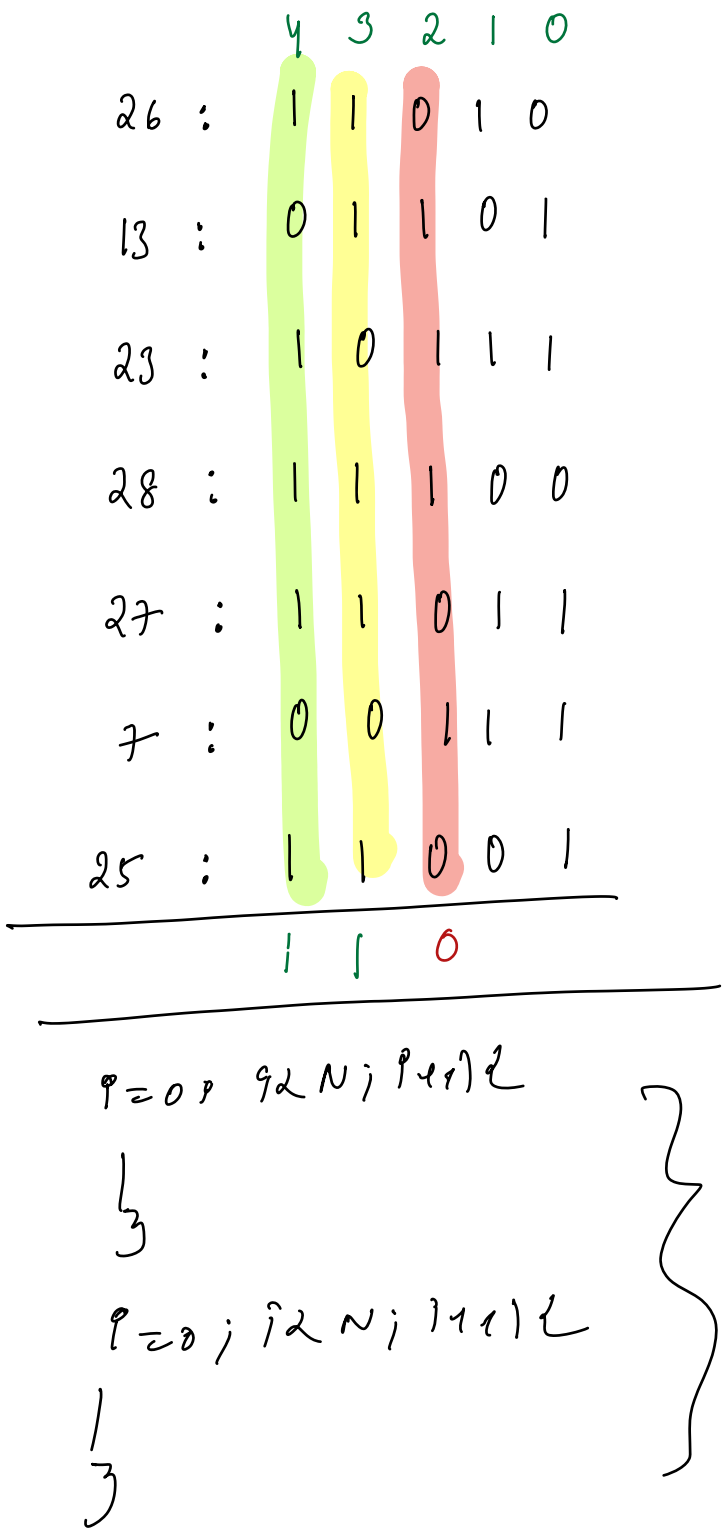
{ Obs: Elements which are non zero, those are your pairs }

{ max bit with and we can get }

TC → $31[N+N] \Rightarrow 62N \Rightarrow O(N)$

SC → $O(1)$

Doubts




```
for(int i=0; i<N; i++){

}

for(int i=0; i<N; i++){

}
```

XOR

1 1 0 1

1 0 1 1

0 1 1 1

→ Subsets using bits

→ doubts: Sunday / Tuesday

→ ( )

{2, 3, 5}

→ {2, 3}

0 1 1

(Doubts)

→ Subsets / Subsequence

2 3 5

→ { 2^n Subseqs / Subseq / }

{2, 3} → order doesn't matter

{3, 2} (1)

$N_{P_0} + N_{P_1} + N_{P_2} - \cdots - N_{P_N}$

$N_{C_0} + N_{C_1} + \cdots N_{C_N}$ $\}$ 2^N

→ prime subsequence:

→ $\{2, 3, *, *, 5\}$