- **N & (N-1)** :- unsets the closest set bit to LSB.

N = 1 0 1 0 1 1 0 0 0 → unset

N & (N-1)

N = 1 0 1 0 1 1 0 0 0

N-1 = 1 0 1 0 1 0 1 1 1

1 0 1 0 1 0 0 0 0

1 0 1 1 0 1 0 0 1 0

1 0 1 1 0 1 0 0 0 0

cnt total set bits

```
int cnt = 0;
while (N > 0)
{
    N = N & (N-1);
    cnt++;
}
```

O(count of set bits)

- check if the given number N is power of 2 or not?

36 - ?

2	32
2	16
2	8
2	4
2	2
	1

2	36
2	18
	9

36 — not a power of 2

power of 2 ≡ One set bit

$2^3 + 2^5$

1010000

10000

```
if( N & (N-1) == 0) return true
else return false;
```

Q All numbers from 1 → N+1 are present except for one number which is missing. Find this missing no.

size N arr

1 4 2 3 6

N=5

1 → 6

ans = 5

1 → N sum of natural no.

1 + 2 + 3 + 4 + 5 + 6

(1 + 4 + 2 + 3 + 6)

$6*(6+1)/2$

missing no = $\frac{n*(n+1)}{2}$ − sum of array

n+1

T.C: O(N)
S.C: O(1)

1 ^ 4 ^ 2 ^ 3 ^ 6 ^ 1 ^ 2 ^ 3 ^ 4 ^ 5 ^ 6

missno = (xor of array) ^ (xor 1 → N+1)

T.C: O(N)
S.C: O(1)

Q array elements. Every elements occurs twice except for 2 elemets which are unique!

① 3 4 3 5 4 7

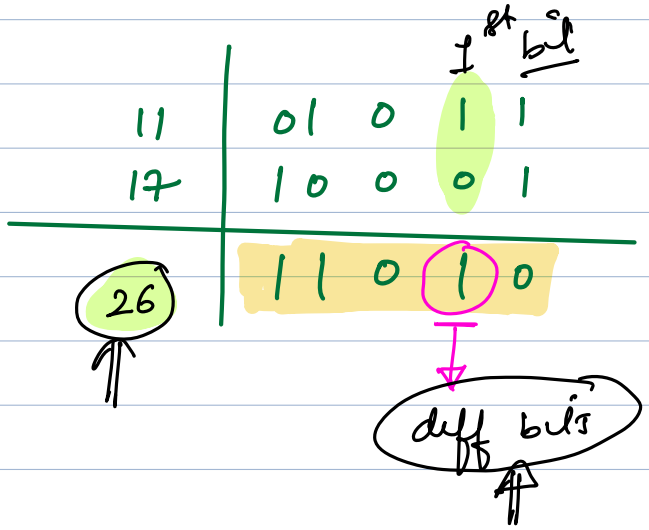
② 10 8 8 9 12 9 6 11 10 6 12 17

Sol^n ① Sort the data, check the adjacent elements

TC($n \log_2 n$)

② freq Hashmap → T.C: O(N)
S.C: O(N)

Xor of all array elements = 11 ^ 17

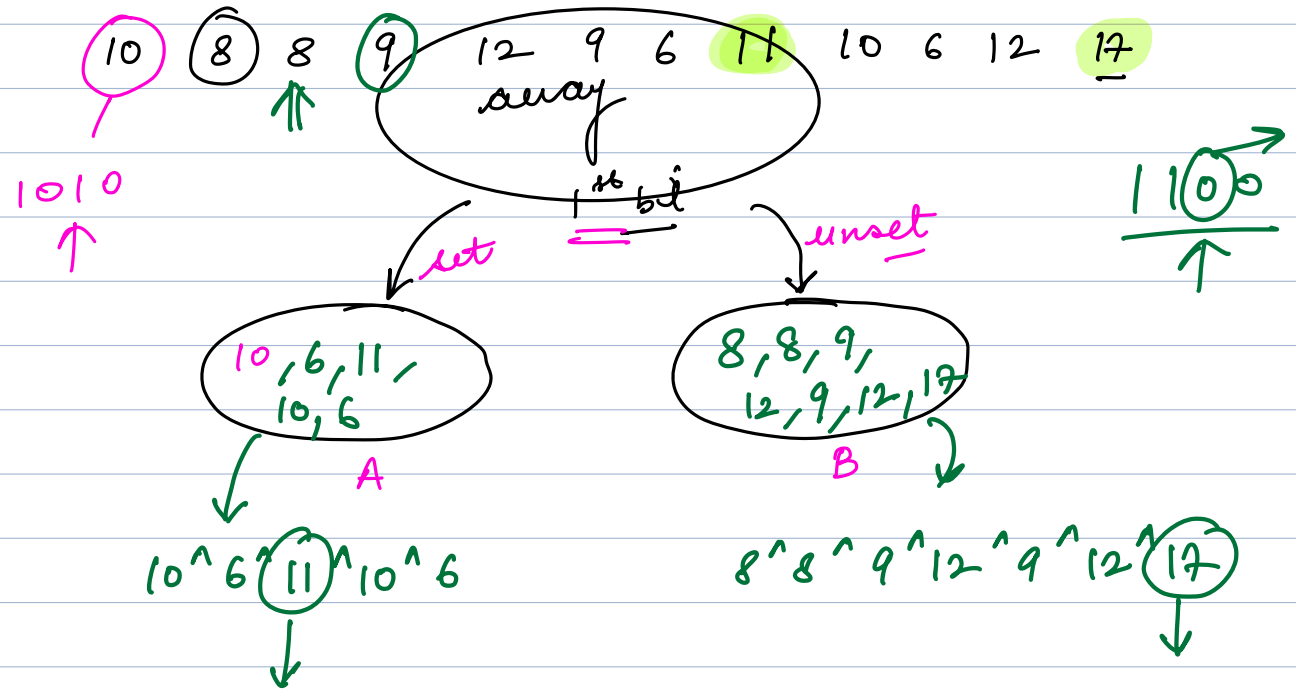


a ^ b = c

a ^ c = b

Both the unique no: will differ at ith bit

↳ any set bit in xor of array



(1) Take xor of complete array — xorArray

(2) In xorArray, find any location which is set

idx

(3) Divide the array into 2 on the basis of idx.

```
xor = 0;
for(i=0; i<n; i++)
    xor = xor ^ arr[i];
idx = -1;
for(i=0; i<32; i++)
{
    if(xor & (1<<i) != 0)
    { idx = i; break; }
}
```

```
xorA = 0;      xorB = 0;
(set)          (unset)
for( i=0; i<n; i++)
{
    if( arr[i] & (1 << idx) != 0)
        xorA = xorA ^ arr[i];
    else
        xorB = xorB ^ arr[i];
}
xorA, xorB
```

10:25 - Break!

Q You are given with an array.

Find sum of xor of all pairs.

{ 3, 5, 6, 8 }

	3	5	6	8
3	3^3	3^5	3^6	3^8
5	5^3	5^5	5^6	5^8
6	6^3	6^5	6^6	6^8
8	8^3	8^5	8^6	8^8

contributions

→ upper triangle sum

→ 0

B.F :- $O(N^2)$

XOR
OR
AND

3 ^ 5	0 1 1 0	= 6
3 ^ 6	0 1 0 1	= 5
3 ^ 8	1 0 1 1	= 11
5 ^ 6	0 0 1 1	= 3
5 ^ 8	1 1 0 1	= 13
6 ^ 8	1 1 1 0	= 14

0th

3 ^ 5 = 4 + 2

4 + 1

8 + 2 + 1

2 + 1

8 + 4 + 1

8 + 4 + 2

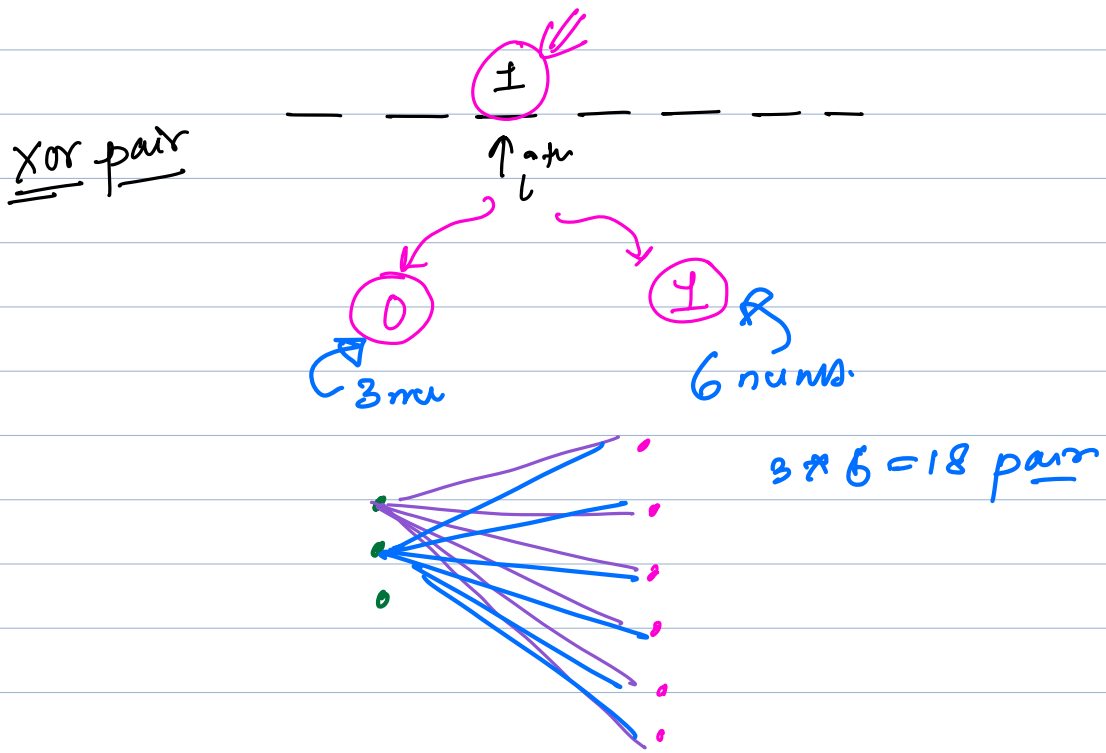
3 4 4 4

2^0

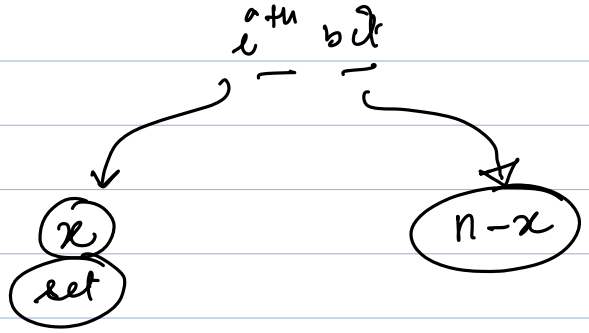
52 * 2 = 104

$3*2^3 + 4*2^2 + 4*2^1 + 4*2^0$

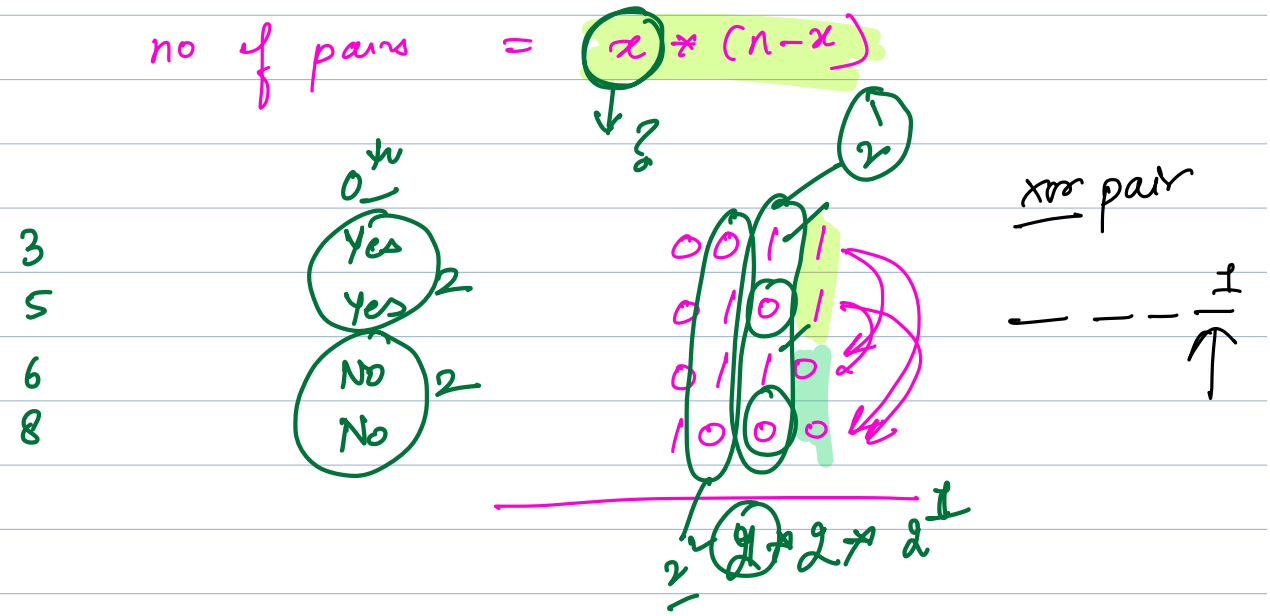
contribution of i^{th} bit = no of pairs in which i^{th} bit is set (XOR) * 2^i, $1 << i$



n elements



no of pairs = $x * (n-x)$



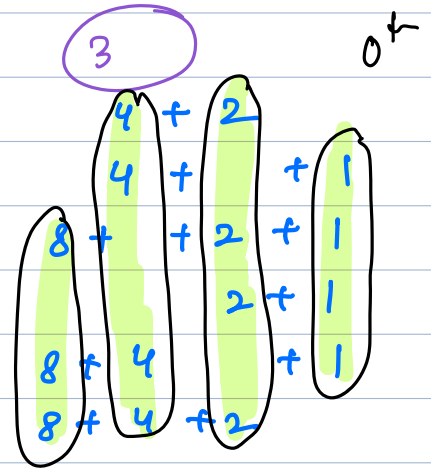

```
for( i=0; i<32; i++)
{
    int x=0;
    for(j=0; j<n; j++)
    {
        if(arr[j] & (1<<i) != 0)
            x++;
    }
    ans = ans + x*(n-x)*(1<<i);
}
return ans*2;
```

O(32*N) ≈ O(N)

3 ^ 5	0 1 1 0	= 6
3 ^ 6	0 1 0 1	= 5
3 ^ 8	1 0 1 1	= 11
5 ^ 6	0 0 1 1	= 3
5 ^ 8	1 1 0 1	= 13
6 ^ 8	1 1 1 0	= 14

0th

52



Different pair sum

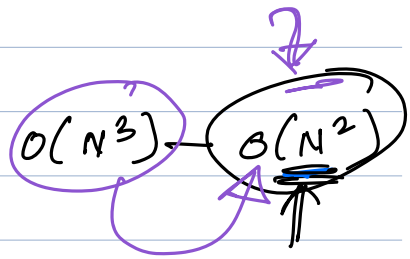
Q Subarrays OR

sum of all OR of all subarrays

1, 4, 3, 2, 6

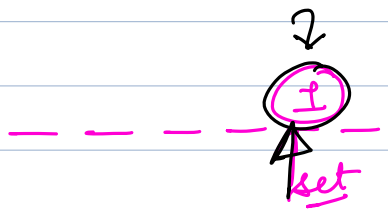
1 — 1 4 — 1 4 3 — 1 4 3 2 — 1 4 3 2 6 —
4 — 4 3 — 4 3 2 — 4 3 2 6 —
3 — 3 2 — 3 2 6 —
2 — 2 6 —
6 —

B-f:- consider all subarrays — O(N^3) — O(N^2)



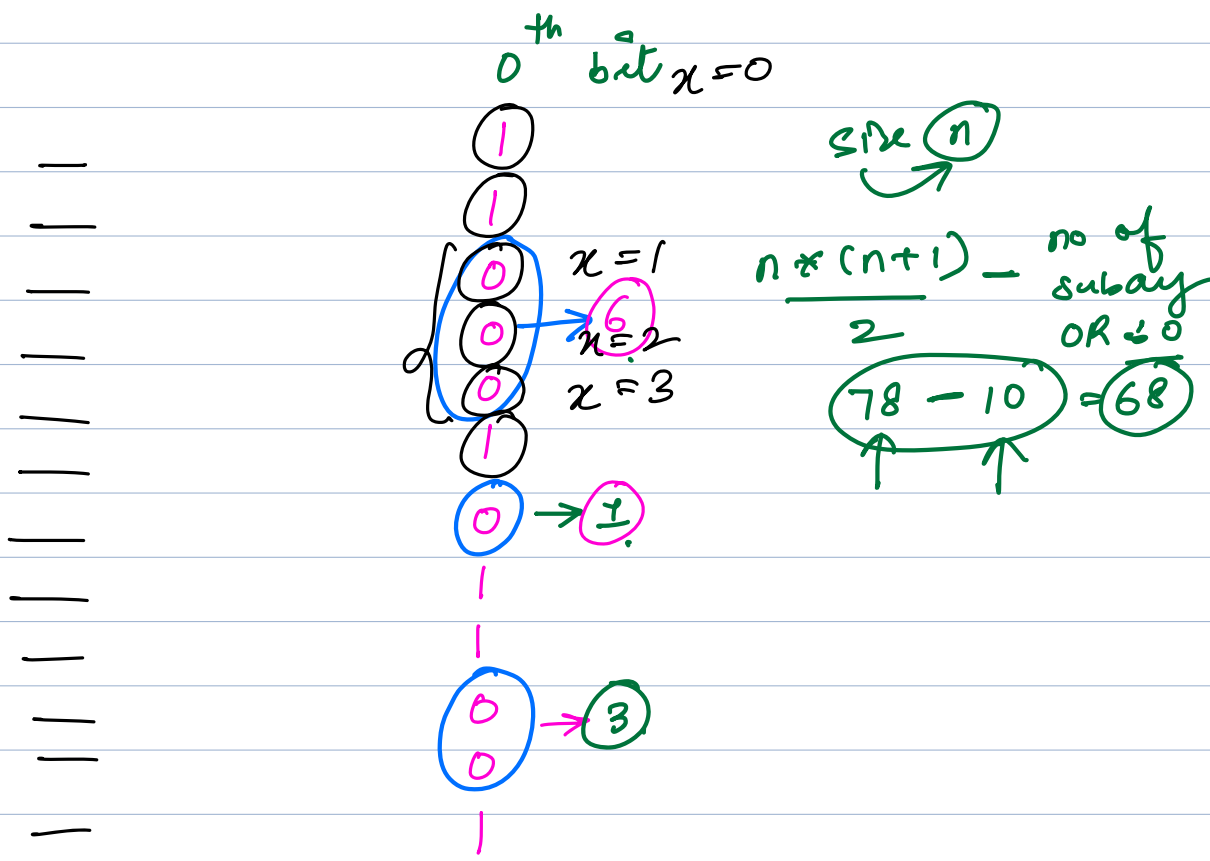
contribution

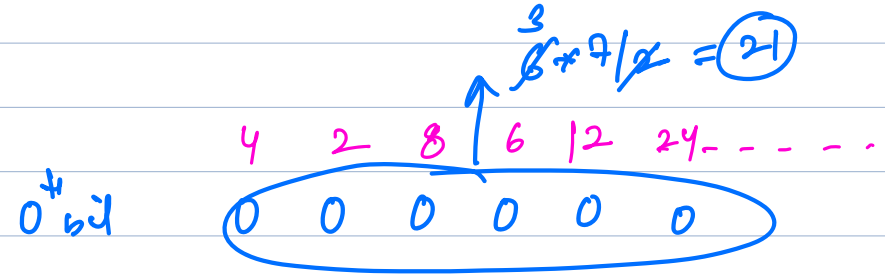
subarray OR



In how many subarray OR's i^{th} bit is set

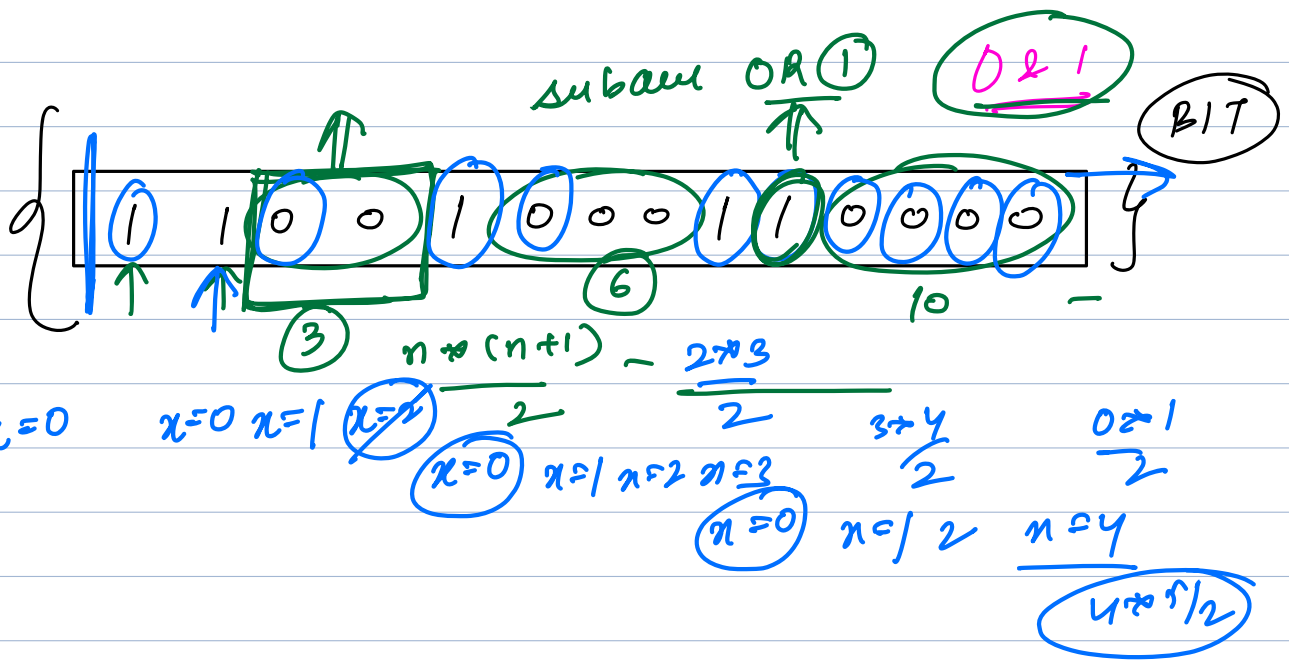
cont of i^{th} = count $* 2^i$





1
1
0 x=1
0 x=2
0 x=3
0 x=4

```
for( i=0; i<32; i++)
{
    i^th bit    total = n*(n+1)/2;   x=0;
    for(j=0; j<n; j++)
    {
        if( arr[j] & (1<<i) == 0)
            x++;
        else{
            total -= x*(n+1)/2;   x=0
        }
    }
    total -= x*(n+1)/2;
    ans = ans + total * 2^i  -> 1<<i
}
```



2	3	4	7

2, 3
3

3, 4

2, 3, 4

2, 3, 4, 7

7

4, 7

3, 4, 7

2
3
4
7

3 → 0 1 0 −1
0 1 1

1 0 0 −1
1 1 1

0+

10−3 = 7
10−1 = 9
10−2 = 8 * 1

2
4 * 5 = 10
4 *