③ BIT MAGIC

Bitwise operator
- And &
- Or |
- XOR ^
- Left shift operator <<
- right shift operator >>
- Not ~

→ Bitwise And &

x	y	x&y
0	0	0
0	1	0
1	0	0
1	1	1

→ Bitwise OR |

x	y	x|y
0	0	0
0	1	1
1	0	1
1	1	1

→ Bitwise XOR ^

x	y	x^y
0	0	0
0	1	1
1	0	1
1	1	0

→ Left shift operator <<

$x = 3$

$x << i$ → ignore i leading bits and add i trailing zeros.

★ If leading y bits are 0, then $x << y$ === $x * 2^y$

→ Right shift operator >>

$x = 33$

$x >> i$ → ignore i trailing bits and add i leading zeros

★ $x >> y$ === $\left\lfloor \frac{x}{2^y} \right\rfloor$

Priyansh

→ Bitwise Not ~

2^s compliment of x in n bit representation = $2^n - x$

To check K^{th} bit of a number n is set(1)

① if (n & (1 << (K-1)) != 0)

```
cout << "Yes" ;
else
cout << "No" ;
```

② if ((n >> (K-1)) & 1 == 1)

```
Cout << "Yes";
else
cout << "No" ;
```

Count set bits

① Brian Kernigam's Algorithm { n & (n-1) }

```
int countsetbit (int n) {
    int res = 0 ;
    while (n > 0) {
        n = n & (n-1) ;
        res++ ;
    }
    return res;
}
```

TC : O(total no of set bits)

② Look up table for 32 bit number {0-256 pre computed ans}

```
int res = 0 ;
int table[256];
void initialise () {
    table[0] = 0;
    for (int i=1; i<256; i++)
        table[i] = (i & 1) + table[i/2];
```

```
int count (int n){
    int res = table[n & 0xff];
    n = n >> 8;
    res += table[n & 0xff];
    n = n >> 8;
    res += table[n & 0xff];
    n = n >> 8;
    res += table[n & 0xff];
    return res;
}
```

Power of two (last power of 2 for given number n)

```
return ((n != 0) and (n & (n-1) == 0));
```

XOR (^) properties

- $x \wedge 0 = x$
- $x \wedge y = y \wedge x$
- $x \wedge (y \wedge z) = (x \wedge y) \wedge z$
- $x \wedge x = 0$

To get the rightmost set bit $x \,\&\, \sim(x-1)$

find one odd appearing no. [given a vector where every no. comes even time find the odd appearing no.]

```
int find (int arr[], int n) {
    int res = 0;
    for (int i = 0; i < n; i++){
        res ^= arr[i];
    return res;
}
```

Priyansh

**

find two odd appearing no. [given a vector where every no. comes even no. of time and only 2 no. are odd times appearing].

```
void find (int arr[], int n){
    int res = 0, res1 = 0, res2 = 0;
    for(int i=0; i<n; i++)
        res ^= arr[i];
    int Sn = res & ~(res-1);
    for(int i=0; i<n; i++)
    {
        if((arr[i] & Sn) != 0)
            res1 ^= arr[i];
        else
            res2 ^= arr[i];
    }
    cout << res1 << res2;
}
```

(rightmost set bit) ←

No. of subset in a string of given length n = 2^n

Power Set {a,b,c} => { {}, {a}, {b}, {c}, {a,b}, {a,c}, {b,c}, {a,b,c} }

```
void powerset (string str){
    int n= str.length();
    int ext = pow(2,n);
    for(int i=0; i<ext; i++) {
        for(int j=0; j<n; j++) {
            if (i & (1<<j) != 0)
                cout << str[j];
        }  cout<<endl;
    }
}
```

TC : $\Theta(2^n * n)$

classmate
Date
Page

****** find the Rightmost set bit

① ffs();

② $\log_2(n \& \sim(n-1)+1)$;

****** Total no. of set bit in a power of 2 ⇒

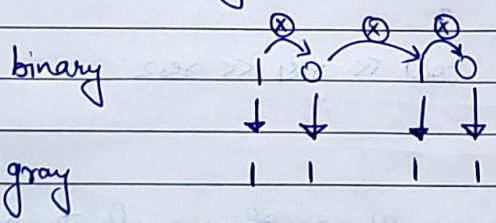
total set bit in 0 to 2^n = $n * 2^{(n-1)}$

****** If n is any given no. Then,

total set bit in 0 to n = $[x \cdot 2^{(x-1)}] + [n - 2^x + 1] + solve(n-2^x)$

where x is max^m power of 2 ie $2^x \leq n$

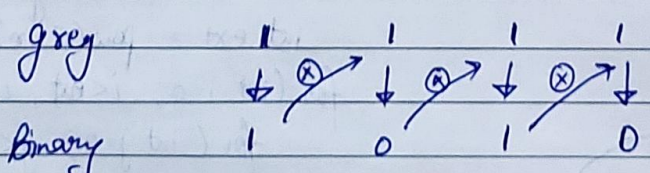
Binary code ⟶ Gray code



MSB same
rest do XOR with last digit

Code: $n \wedge (n >> 1)$

Gray code ⟶ Binary code



MSB same
rest do XOR with last result

```
Code:  int res = n;
       while (n > 0) {
           n = n >> 1;
           res = res ^ n;
```


Priyansh

classmate
Date
Page

* $x = x \& (x-1)$: turn off the rightmost set bit.

* flip a given number : given no. (n) eg: [redacted] 1011
  take value (v) all set bit 1111
  flip no. = (v) - (n) = 1111 - 1011 = 0100

* Swap (a) & (b).

  a ^= b
  b ^= a
  a ^= b.

** find MSB position in given no. (n) ⇒ $\log_2(n)$

*** find value of XOR for 1.2.3 ---> N

if N%4 == 0 then 1^2^3 ___ ^N = N
1 then 1^2^3 ___ ^N = 1
2 then 1^2^3 ___ ^N = N+1
3 then 1^2^3 ___ ^N = 0

Priyansh