Todays Content

1. Check Bit
2. Count Set Bit
3. Power of 2 & Shift
    a) Set i^{th} Bit
    b) Flip i^{th} Bit
4. Negative numbers & Ranges & Overflows
5. % modular Arthmetic Basics

Rightshift >> {Revise For First 5 Mins}

Say a is 8 bit number? At max it can store 8 bits

$2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

a = 25: 0 0 0 1 1 0 0 1 = 16 + 8 + 1 = 25 = $25/2^0$

a>>1: 0 0 0 0 1 1 0 0 = 8 + 4 = 12 = $25/2^1$

a>>2: 0 0 0 0 0 1 1 0 = 4 + 2 = 6 = $25/2^2$

a>>3: 0 0 0 0 0 0 1 1 = 2 + 1 = 3 = $25/2^3$

a>>4: 0 0 0 0 0 0 0 1 = 1 ⟶ $25/2^4$

Obs: a>>n: $a/2^n$

100>>4 = $100/2^4$ = 6

Sizes: 1 Byte = 8 bits

Int : 4 B : 4 * 8 = 32 bits

long : 8 B : 8 * 8 = 64 bits

Q1: Given N & i, check if i^{th} bit in N is Set = 1 or Not = 0?

	5	4	3	2	1	0
	2^5	2^4	2^3	2^2	2^1	2^0
N = 21 : i = 2	0	1	0	1	0	1
N = 34 i = 3 :	1	0	0	0	1	0

ans = Set

ans = UnSet

Ideal: Convert number into binary & check if i^{th} bit is Set/UnSet

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	i^{th}
N = 10 :	0	0	0	0	1	0	1	0	0 : (N&1) == 1 : Set / Else UnSet
N>>1 :		0	0	0	0	1	0	1	1 : (N>>1) & 1 == 1 : Set / Else UnSet

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	i^{th}
N = 45 :	0	0	1	0	1	1	0	1	2
N>>2 :			0	0	1	0	1	1	: (N>>2) & 1 == 1 : Set / Else UnSet

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	i^{th}
N = 45 :	0	0	1	0	1	1	0	1	3
N>>3 :				0	0	1	0	1	: (N>>3) & 1 == 1 : Set / Else UnSet

boolean checkBit(int N, int i){ TC: O(1) SC: O(1)

```
if ((N>>i) & 1 == 1) { //Set
    return True
}
else { // UnSet
    return False
}
}
```

Q2: Given N, return no: of set bits in N

N = 10 : $2^3\ 2^2\ 2^1\ 2^0$ → 1 0 1 0 ans = 2

N = 26 : $2^4\ 2^3\ 2^2\ 2^1\ 2^0$ → 1 1 0 1 0 ans = 3

Idea1:

N = 45 c = 0 // count of set bits

$2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$ (N & 1) == 1 : Set

0 1 0 1 1 0 1 — 0^{th} Set c = c+1, c = 1

0 0 1 0 1 1 0 — 0^{th} Unset c = 1

0 0 0 1 0 1 1 — 0^{th} Set c = c+1, c = 2

0 0 0 0 1 0 1 — 0^{th} Set c = c+1, c = 3

0 0 0 0 1 0 0 — 0^{th} Unset c = 3

0 0 0 0 0 0 1 — 0^{th} Set c = c+1, c = 4

0 0 0 0 0 0 0 — N == 0, Stop process

```
int countbits2(int N){
    int c=0;
    while(N>0){
        //Check 0th Bit is Set
        if((N&1)==1){
            c=c+1
        }
        N=N>>1;
    }
    return c;
}
```

Issues:

```
N=10;
print(N)//10
N+10;
print(N)//10
```

```
N=10;
N>>1;
print(N)//10
```

Power of left shift:

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1 :	0	0	0	0	0	0	0	1	
1<<3 :	0	0	0	0	1	0	0	0	val = 8

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1 :	0	0	0	0	0	0	0	1	
1<<2 :	0	0	0	0	0	1	0	0	val = 4

<< with OR

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
N = 45 :	0	0	1	0	1	1	0	1	
or						↕			
1<<2 :	0	0	0	0	0	1	0	0	
N | (1<<2) :	0	0	1	0	1	1	0	1	val = 45

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
N = 41 :	0	0	1	0	1	0	0	1	
or						↕			
1<<2 :	0	0	0	0	0	1	0	0	
N | (1<<2) :	0	0	1	0	1	1	0	1	= 41 + 2^2 = 45

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
N = 42 :	0	0	1	0	1	0	1	0	
or						↕			
1<<3 :	0	0	0	0	1	0	0	0	
N | (1<<3) :	0	0	1	0	1	0	1	0	val = 42

	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
N = 50 :	0	0	1	1	0	0	1	0	
or						↕			
1<<3 :	0	0	0	0	1	0	0	0	
N | (1<<3) :	0	0	1	1	1	0	1	0	= 50 + 2^3 = 58

Obs: N | (1<<i)

- : i^{th} bit in N is Set : N remains unchanged
- : i^{th} bit in N is UnSet : N becomes = $N + 2^i$: i^{th} bit set

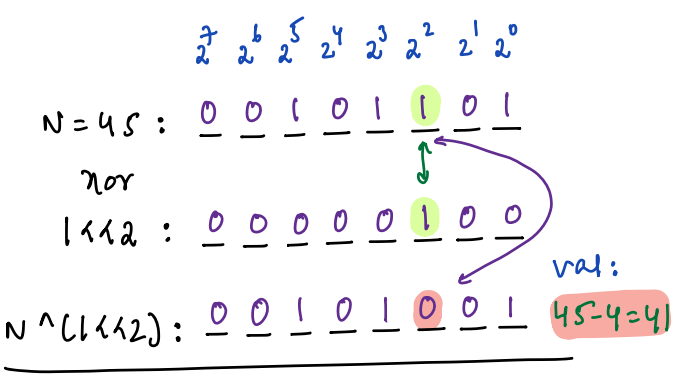
Q: Set i^{th} bit in N.

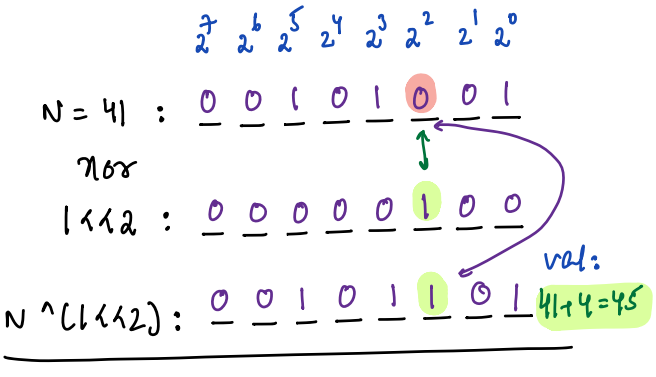
a. If i^{th} bit is already set, leave it set

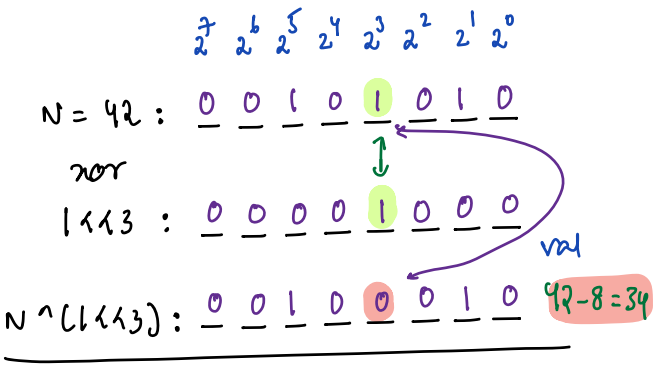
b. If i^{th} bit is unset, make it set in N

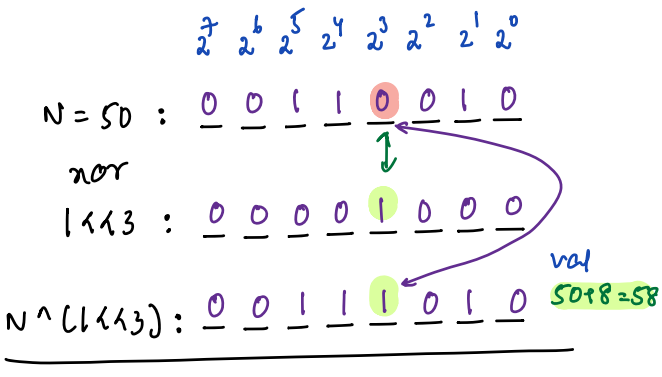
→ N = N | (1<<i)

<< with XOR









obs: N ^ (1<<i)

- : i^{th} bit in N is Set : N becomes = N - 2^i (i bit unset)
- : i^{th} bit in N is UnSet : N becomes = N + 2^i (: i^{th} bit set)

Q: Flip i^{th} bit in N.

a. if i^{th} bit is already set, make it unset

b. if i^{th} bit is unset, make it set

N = N ^ (1<<i)

Cute Math Observations

$2^0 = 1, 2^1 - 1$

$2^0 + 2^1 = 3, 2^2 - 1$

$2^0 + 2^1 + 2^2 = 7, 2^3 - 1$

$2^0 + 2^1 + 2^2 + 2^3 = 15, 2^4 - 1$

Generalization:

$$2^0 + 2^1 + 2^2 + 2^3 + \ldots + 2^k = 2^{k+1} - 1$$

-Ve number :

8 bit :

$-2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

_ _ _ _ _ _ _ _

↓ $2^6 + 2^5 + 2^4 + 2^3 + 2^2 + 2^1 + 2^0 =$

$2^7 > 2^7 - 1$

MSB : Most Significant Bit : Value of MSB position is -ve

$-2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

a = 10 : 0 0 0 0 1 0 1 0 :

$-2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

-10 : 1 0 0 0 1 0 1 0 : $-2^7 + 2^3 + 2^1 = -128 + 10 = -118 \neq -10$

Correct way : $-a = \sim a + 1$: This will be performed by system.

$-2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

a = 10 : 0 0 0 0 1 0 1 0

$-a \begin{bmatrix} \sim a \\ + \\ 1 \end{bmatrix}$: 1 1 1 1 0 1 0 1

0 0 0 0 0 0 0 1

$-2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

-a = -10 : 1 1 1 1 0 1 1 0

→ $-2^7 + 2^6 + 2^5 + 2^4 + 2^2 + 2^1 +$

→ $-128 + 64 + 32 + 16 + 4 + 2$

→ $-128 + 118 = -10$

$-2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

a = 25 : 0 0 0 1 1 0 0 1

$-a \begin{bmatrix} \sim a \\ + \\ 1 \end{bmatrix}$: 1 1 1 0 0 1 1 0

0 0 0 0 0 0 0 1

$-2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

-25 : 1 1 1 0 0 1 1 1

→ $-2^7 + 2^6 + 2^5 + 2^2 + 2^1 + 2^0$

→ $-128 + 64 + 32 + 4 + 2 + 1$

→ $-128 + 103 = -25$

Ranges:

8 bit:

$-2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

1 0 0 0 0 0 0 0 : Min $= -2^7 = -128$

$-2^7\ 2^6\ 2^5\ 2^4\ 2^3\ 2^2\ 2^1\ 2^0$

0 1 1 1 1 1 1 1 : Max $= 2^7 - 1 = 127$

Sum $= 2^0 + 2^1 + 2^2 + \dots 2^6$

16 bit:

$-2^{15}\ 2^{14}\ 2^{13} \dots 2^3\ 2^2\ 2^1\ 2^0$

1 0 0 ... 0 0 0 0 : Min $= -2^{15}$

$-2^{15}\ 2^{14}\ 2^{13} \dots 2^3\ 2^2\ 2^1\ 2^0$

0 1 1 ... 1 1 1 1 : Max $= 2^{15} - 1$

Sum $= 2^0 + 2^1 + 2^2 + \dots 2^{14}$

32 bit:

$-2^{31}\ 2^{30}\ 2^{29} \dots 2^3\ 2^2\ 2^1\ 2^0$

1 0 0 ... 0 0 0 0 : Min $= -2^{31} = -2 * 10^9$

$-2^{31}\ 2^{30}\ 2^{29} \dots 2^3\ 2^2\ 2^1\ 2^0$

0 1 1 ... 1 1 1 1 : Max $= 2^{31} - 1 = 2 * 10^9$

Sum $= 2^0 + 2^1 + 2^2 + \dots 2^{30}$

64 bit:

$-2^{63}\ 2^{62}\ 2^{61} \dots 2^3\ 2^2\ 2^1\ 2^0$

1 0 0 ... 0 0 0 0 : Min $= -2^{63} = -8 * 10^{18}$

$-2^{63}\ 2^{62}\ 2^{61} \dots 2^3\ 2^2\ 2^1\ 2^0$

0 1 1 ... 1 1 1 1 : Max $= 2^{63} - 1 = 8 * 10^{18}$

Sum $= 2^0 + 2^1 + 2^2 + \dots 2^{62}$

Approx:

$2^{10} = 1024 \approx 1000 = 10^3$

$2^{10} \approx 10^3$

Step1: Apply cube on both

$2^{10} \approx 10^3$

$2^{10} * 2^{10} * 2^{10} \approx 10^3 * 10^3 * 10^3$

$2^{30} \approx 10^9$

: Multiply by 2

$2^{30} * 2 \approx 2 * 10^9$

$2^{31} \approx 2 * 10^9$

Step2:

$2^{31} \approx 2 * 10^9$

Apply square

$2^{31} * 2^{31} \approx 2 * 10^9 * 2 * 10^9$

$2^{62} \approx 4 * 10^{18}$

: Multiply by 2

$2 * 2^{62} \approx 2 * 4 * 10^{18}$

$2^{63} \approx 8 * 10^{18}$

% operator: Remainder

a % m = Remainder when a is divided by m

Dividend = Divisor * Quotient + Remainder

Remainder = Dividend - Divisor * Quotient

Remainder = Dividend - greatest multiple of Divisor ≤ Divident

rem

10 % 4 = 10 - (greatest multiple of 4 ≤ 10) 8 = 2

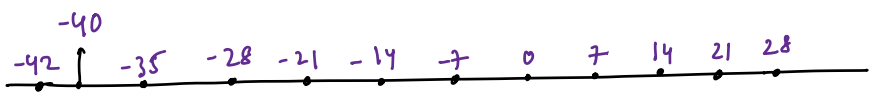
13 % 5 = 13 - (greatest multiple of 5 ≤ 13) 10 = 3

150 % 11 = 150 - (greatest multiple of 11 ≤ 150) 143 = 7

20 % 4 = 20 - (greatest multiple of 4 ≤ 20) 20 = 0

4 % 9 = 4 - (greatest multiple of 9 ≤ 4) 0 = 4

Numberline:



-40 % 7 = -40 - (greatest multiple of 7 ≤ -40) -42 =

= -40 - (-42) = -40 + 42 = 2

-40 % 9 = -40 - (greatest multiple of 9 ≤ -40) -45

= -40 - (-45) = -40 + 45 = 5

-60 % 9 = -60 - (greatest multiple of 9 ≤ -60) -63

= -60 - (-63) = -60 + 63 = 3

	min	max
n % m =	0	[M-1]

Issues in Java :

	Java *	Expected or Python
print (-40%7)	-5 +7 →	2
print (-60%9)	-6 +9 →	3
print (-40%9)	-4 +9 →	5

In Java for -ve numbers if we want to get correct remainder?

```
int rem( int a, int m) {
    if (a<0) {
        return [a%m + m]%m
    }
    else { return a%m }
}
```

a	m	[a%m + m]%m	Expected
-63	9	= -63 %9 + 9 = 9%9 → 0	
-40	9	= -40 %9 + 9 = 5%9 → 5	
-60	9	= -60%9 + 9 = 3%9 → 3	
-40	7	= -40%7 + 7 = 2%7 → 2	

Why %M: Helps us in limiting output data to a required range

-∞ ⋮ ∞ } %M = Expected Output [0, M-1] =

Modular Arithmetics: (%) with (+, -, *, /) → out of scope

$[0 \dots m-1]$ $\quad [0,M-1] + [0,M-1] = \{0..2M-2\} \% M = \{0,M-1\}$

a b m : $(a+b) \% m = (a\%m + b\%m) \% m$

8 6 10 : $(8+6)\%10 = (8\%10 + 6\%10)\%m$

$14\%10 \to 4 = 4 \leftarrow (8+6)\%10$

5 6 3 : $(5+6)\%3 = (5\%3 + 6\%3)\%3$

$(11)\%3 \to 2 = 2 \leftarrow (2+0)\%3$

a b m : $(a*b) \% m = (a\%m * b\%m)\%m$

8 6 10 : $(8*6)\%10 = (8\%10 * 6\%10)\%10$

$48\%10 \to 8 = 8 \leftarrow (8*6)\%10$

a b m : $(a-b) \% m = (a\%m - b\%m + m)\%m$

14 7 4 : $(14-7)\%4 \quad (14\%4 - 7\%4 + 4)\%4$

$7\%4 \to 3 \quad 3 \Leftarrow 3\%4 \leftarrow (2-3+4)\%4$

11 7 3 : $(11-7)\%3 \quad (11\%3 - 7\%3 + 3)\%3$

$4\%3 \to 1 \quad (2-1+3)\%3$

$4\%3 \to 1 \quad (1+3)\%3$

$4\%3 \to 1 \quad 1 = 4\%3$

Q: Sum of ar[] : Include in Content

Note: ans can be very large, return % $[10^9+7]$

Constraints

$1 <= N <= 10^7$

$-10^6 <= ar[i] <= 10^{12}$

```
int sum( long ar[]){
    int N = ar.length;
    long ans = 0;
    for(int i = 0; i < N; i++){
        ans = ans + ar[i]
    }
    return ans % {10^9+7}
}
```

Data already overflowed

```
int m = 10^9+7
int N = ar.length;
long ans = 0;
for(int i = 0; i < N; i++){
    ans = (ans + ar[i]) % m
}
return [ans % m + m] % m
```

If won't overflow