Today's Content:

a) How to store -ve numbers

b) Significance of MSB

c) Datatype Ranges

8 bit number

sign bit → Concept of storing -ve numbers is wrong

	7	6	5	4	3	2	1	0
10 :	0	0	0	0	1	0	1	0
-10 :	1	0	0	0	1	0	1	0

sign bit

4 :	0	0	0	0	0	1	0	0
-4 :	1	0	0	0	0	1	0	0
10 :	0	0	0	0	1	0	1	0

$10 - 4 = 6$

$10 + (-4) = 6$

	...			2^4	2^3	2^2	2^1	2^0
6 :	1	0	0	0	1	1	1	0

sign bit

$-\{0+8+4+2+0\}$

$\Rightarrow -\{14\}$

0 0 0 0 0 0 0 0 → 0

1 0 0 0 0 0 0 0 → -0

sign

2's Complement :

$a = -a$ (in binary) $=$ 2's complement of a $=$ 1's complement $a + 1$ $= \sim a + 1$

1's complement : $1 \rightleftarrows 0$

8 bit numbers

$-10 = $ 1's of $10 + 1$

$-10 = \sim 10 + 1$

10 : 0 0 0 0 1 0 1 0

~ 10 : 1 1 1 1 0 1 0 1

$+1$: 0 0 0 0 0 0 0 1

-10 : -2^7 2^6 2^5 2^4 2^3 2^2 2^1 2^0

1 1 1 1 0 1 1 0

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

$\Rightarrow 128 + 118 = 246$

$-128 + 2^6 + 2^5 + 2^4 + 2^2 + 2^1$

$\Rightarrow -128 + 118 = -10$

8 bit number

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

$2^6 + 2^5 + 2^4 + \dots 2^0$

127

128

MSB : → Most Significant Bit (MSB)

MSB bar value value is Negative.

// 4 bit Number :

3 2 1 0

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

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

4 : 0 0 0 0 0 1 0 0

~ 4 : 1 1 1 1 1 0 1 1

$+1$: 0 0 0 0 0 0 0 1

-4 : 1 1 1 1 1 1 0 0

8 bit nuber

	-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
-4 :	1	1	1	1	1	1	0	0
+								
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
1	0	0	0	0	0	1	1	0

descend

4 2 + 4 = 6

Convert Binary to Decimal Below Value?

4bit Number :

Decimal

-2^3	2^2	2^1	2^0		
1	0	1	1	=	-8 + 2 + 1 = -5
1	0	1	0	=	-8 + 2 = -6
0	0	1	1	=	0 + 2 + 1 = 3
1	0	0	0	=	-8
1	1	1	1	=	-8 + 4 + 2 + 1 = -1

MSB
-ve

8 bit number:

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

0 0 0 1 0 1 0 1 $\rightarrow 2^4 + 2^2 + 2^0 \rightarrow 21$

1 0 0 1 0 1 0 1 $\rightarrow -2^7 + 2^4 + 2^2 + 2^0 \rightarrow -128 + 21$

$\rightarrow -107$

1 0 0 1 0 0 0 1 $\rightarrow -2^7 + 2^4 + 2^0 \rightarrow -111$

N bit Number

$N-1 \quad \ldots \quad 2 \quad 1 \quad 0$

-2^{N-1}

Will MSB Base Value be always -ve?

Keywords:

unsigned | signed

Declare unsigned 4 bit number

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

_ _ _ _

Declare signed 4 bit

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

_ _ _ _

Declare unsigned 8 bit

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

_ _ _ _ _ _ _ _

Declare signed 8 bit

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

_ _ _ _ _ _ _ _

→ How will system know signed / unsigned?

Unsigned : C/C++/C#

unsigned int n

unsigned long n

unsigned short n

← keyword

Base value of MSB is +ve

→ We cannot store -ve numbers in unsigned :

Default

int n

long n

short n

By default they are signed : Base value of MSB bit is -ve

In Java : There is no such that as unsigned, Then are only signed datatypes.

It only supports signed datatypes

Ranges in Signed Datatypes

2 bit Signed Number:

$[-2 \quad 1]$

$[Min \quad Max]$

-2^1	2^0		decimal
0	0	=	0
0	1	=	1
1	0	=	-2
1	1	=	-1

→ Can we store -5 or 10 in 2 bit number: {No}

3 bit Signed Number:

-2^2	2^1	2^0		Decimal
0	0	0	=	0
0	0	1	=	1
0	1	0	=	2
0	1	1	=	3
1	0	0	=	-4
1	0	1	=	-3
1	1	0	=	-2
1	1	1	=	-1

→ Min Max

[-4, 3]

10 mins

→ 8:45a

N bit signed Number

# N bits	Min	Max
2 bits: _ _	-2	1
3 bits: _ _ _	-4	3
4 bits: _ _ _ _ (-2^3 2^2 2^1 2^0)	-8	7
5 bits: _ _ _ _ _	-16	15
N bits: _ _ _ _ _	-2^{N-1}	$2^{N-1}-1$

4 bits:

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

5 bits:

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

N bits:

0	1	1	1	1	1	1
1	0	0	0	0	0	0
-2^{N-1}	2^{N-2}	$\cdots$	2^3	2^2	2^1	2^0

Max N bit Nuber: $\underbrace{2^0 + 2^1 + 2^2 + 2^3 + 2^{N-4} + 2^{N-3} + \cdots 2^{N-2}}_{\text{GP}}$

GP: $\left\{ \begin{array}{l} a = 1 \\ r = 2 \\ t = N-1 \end{array} \right\} \rightarrow \left\{ \dfrac{a * (r^t - 1)}{r-1} \right\}$

Substitute:

$$\frac{1 * (2^{N-1} - 1)}{2-1} = \boxed{2^{N-1} - 1}$$

// N bit Signed Number = $[-2^{N-1}, 2^{N-1}-1]$

Min Max

Min Max

Byte → # N bits → -2^{N-1}, $2^{N-1}-1$

byte/char → 1 → 8 bits = -2^7, 2^7-1

= [-128, 127]

Can we store 180, 240, 10^5 *

↳ overflow

N

short int → 2 → 16 bits = $\{-2^{15}, 2^{15}-1\}$

[-32768, 32767]

int → 4 → 32 bits = $\{-2^{31}, 2^{31}-1\}$

→ Close approx ← $\{-2*10^9, 2*10^9\}$

very unfull

long int → 8 → 64 bits = $\{-2^{63}, 2^{63}-1\}$

Close approx ← $\{-8*10^{18}, 8*10^{18}\}$

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

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

Apply Cube on both sides

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

$$\underbrace{2^{10} * 2^{10} * 2^{10}}_{} = 10^3 * 10^3 * 10^3$$

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

Multiply with 2 on both sides

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

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

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

Apply square on both sides

$$(2^{30})^2 = (10^9)^2$$

$$2^{30} * 2^{30} = 10^9 * 10^9$$

$$\boxed{2^{60} = 10^{18}}$$

Multiply 2^3 on both sides

$$2^3 * 2^{60} = 2^3 * 10^{18}$$

$$\boxed{2^{63} = 8 * 10^{18}}$$

Importance of Constraints:

1Q) Given N array elements calculate sum of them.

Constraints:

$1 \le N \le 10^5$

$1 \le ar[i] \le 10^6$

$1 \le sum \le 10^{11}$

Can we store in int *

We need to use long

```
long
~~int~~ sum (int ar[])
{
    int n = ar.length
    ~~int~~ long sum = 0
    i = 0; i < n; i++)
    {
        sum = sum + ar[i]
    }
    return sum
}
```

→ Submit:

Few Test case wrong ans

Constraints → TLE → Time Limit Exceeding

Constraints → Datatypes → error

D) Very Very Imp

given 2 number a & b return their prod

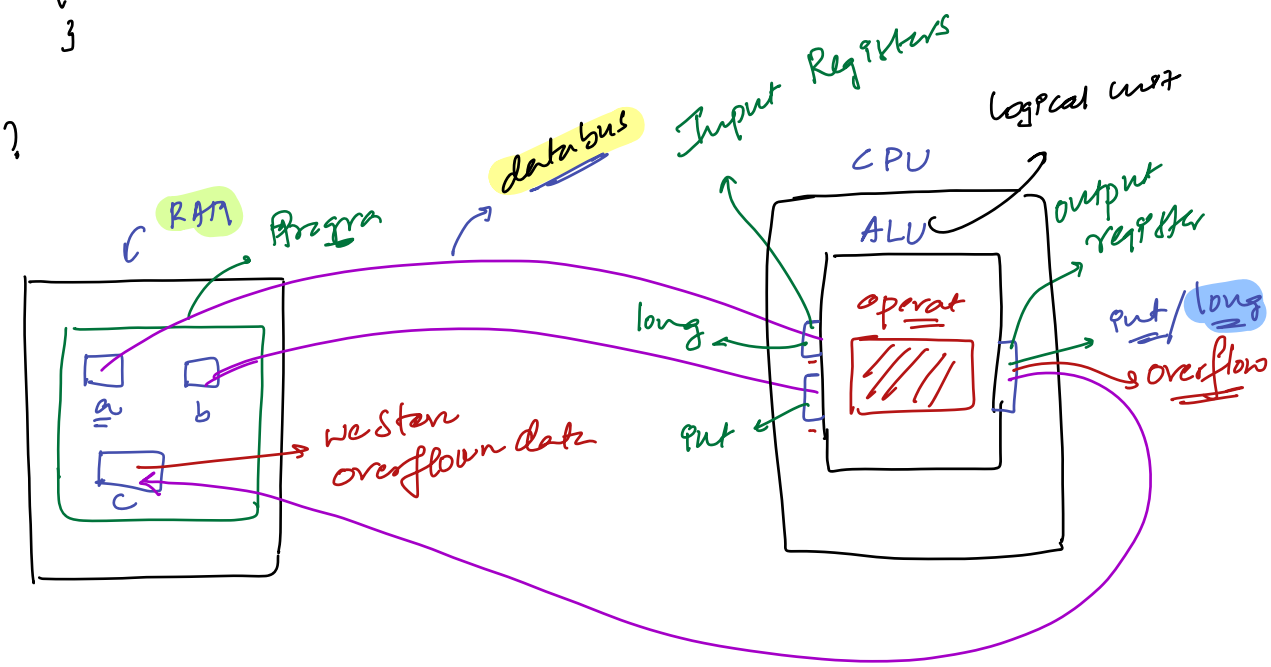
Constraints

1 ≤ a, b ≤ 10^6 } a × b ≤ 10^{12} → We cannot store the prod in int

```
prod (int a, int b) {
    int c = (a*b)          ✗
    long c = (a*b)         ✗
    long c = long(a*b)     ✗
    long c = (long) a * b  ✓
    long a1 = a  b1 = b
    return a1 * b1         ✓
}
```

BODMAS
↳ brackets

why?



// Take care :

- When we multiply int * int
- When we multiply long * long

⇒ Monetary : {Maths}

Doubts?

↳ Why 2's Complement is $-a = \sim a + 1 = -a$ {Indepth}

8 bit

-2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
1	1	1	1	1	1	1	1	= -1

→ 10

↳ unset ⇒ we reduce 1 0

1	1	1	1	0	1	0	1	= -11

$\sim 10 + 1 = -10$

$-11 + 1 = -10$

Unset i^{th} bit

8-bit number:

	7	6	5	4	3	2	1	0
A:	1	0	0	1	1	0	1	1
B:	1	1	1	0	1	1	1	1
	1	0	0	0	1	0	1	1

$$1\ 1\ 1\ 0\ 1\ 1\ 1\ 1 = \sim(0\ 0\ 0\ 1\ 0\ 0\ 0\ 0)$$

$$B = \sim(1 << 4)$$

// given A, unset i^{th} bit:

return $A \,\&\, [\sim(1 << i)]$