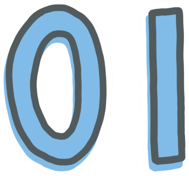
Bit Manipulation 2



Agenda

- Check bit / Count set bits
- Set i^{th} bit
- Unset i^{th} bit
- Negative Numbers
- Ranges
- Importance of constraints

Quick Revision

Quiz 1:

a = 15

print(a<<2)

$$15 << 2 = 15 \times 2^{2}$$

$$= 15 \times 4$$

$$= 60$$

Quiz 2:

Which of the following options output is 2 power n

$$1 << N \qquad a << i = a \times 2^{i}$$

Put $a=1$ →

$$1 << i = 1 \times 2^{i}$$

$$1 << N = 1 \times 2^{N} = 2^{N}$$

Quiz 3:

int a = 29

print(a>>2)

$$29 >> 2$$

$$= \frac{29}{2^{2}} = \frac{29}{4} = 7$$

Set → Put 1

Set i^{th} bit

Given N & i, set the i^{th} bit in N.

$$\begin{array}{lcccc} & 3 & 2 & 1 & 0 \\ N = 10 & 1 & 0 & 1 & 0 \\ i = 2 & 1 & 1 & 1 & 0 \end{array} \Rightarrow 14$$

$$\begin{array}{lccccc} & 4 & 3 & 2 & 1 & 0 \\ N = 23 & 1 & 0 & 1 & 1 & 1 \\ i = 2 & 1 & 0 & 1 & 1 & 1 \end{array} \Rightarrow 23$$

Ans = N | Magic No

Magic Number - A no which has only i^{th} bit set. All other bits are unset.

$2^i = (1 << i)$

$i = 2$ → 00100

$i = 3$ → 01000

$i = 1$ → 00010

```
int setIthBit(int N, int i) {

    return    N | (1 << i)

}
```

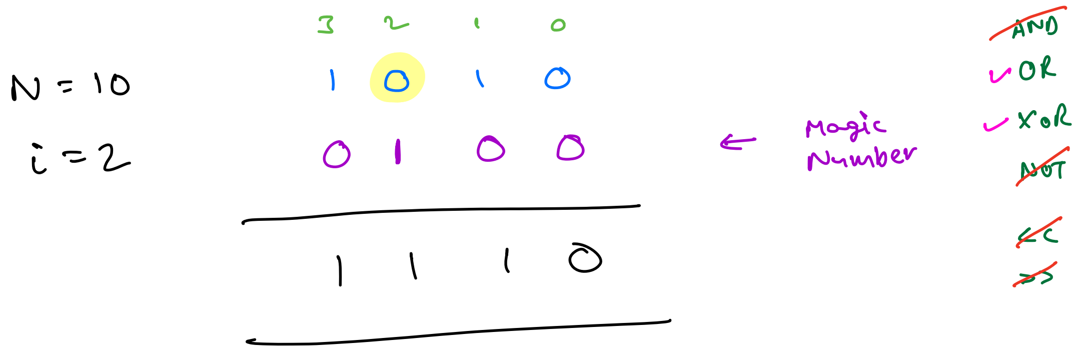
$1 << i$

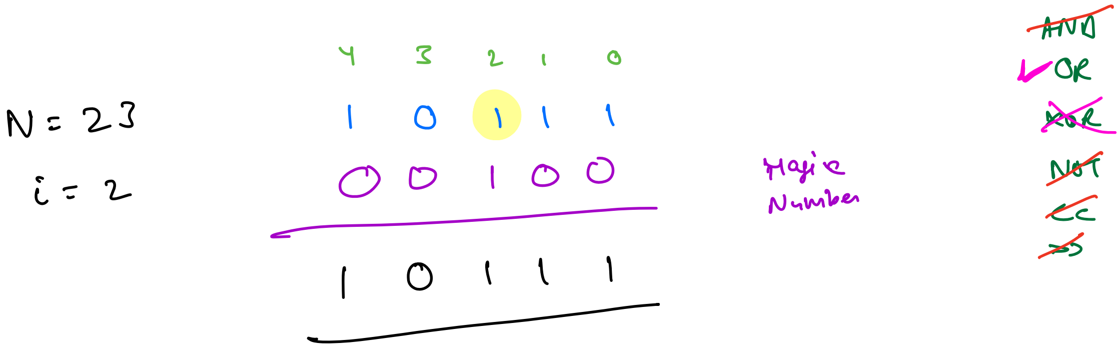
1 = 00001

$i = 1$ → $1 << 1$ ⇒ 00010

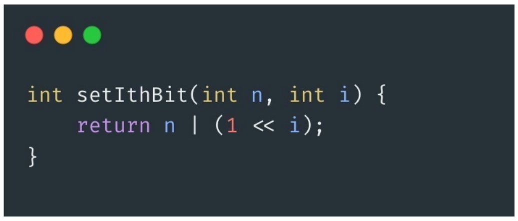
$i = 2$ → $1 << 2$ ⇒ 00100

$i = 3$ → 0●000



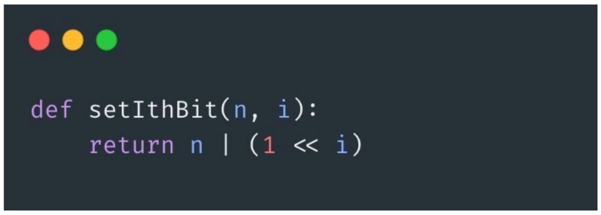


Java



```java
int setIthBit(int n, int i) {
    return n | (1 << i);
}
```

Python



```python
def setIthBit(n, i):
    return n | (1 << i)
```

Unset i^{th} bit

Given N & i, unset the i^{th} bit in N.

N = 10

i = 2

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

⇒ 10

N = 23

i = 2

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

⇒ 19

```
int unsetIthBit(int N, int i) {
        return      N & (~ (1<<i) )
}
```

N = 10

i = 2

$$\begin{array}{cccccc}
 & 3 & 2 & 1 & 0 & \\
 & 1 & 0 & 1 & 0 & \\
\& & 1 & 0 & 1 & 1 & \leftarrow \text{Super Number} \\
\hline
 & 1 & 0 & 1 & 0 & \Rightarrow 10
\end{array}$$

N = 23

i = 2

$$\begin{array}{ccccccc}
 & 4 & 3 & 2 & 1 & 0 & \\
 & 1 & 0 & 1 & 1 & 1 & \\
\& & 1 & 1 & 0 & 1 & 1 & \leftarrow \text{Super Number} \\
\hline
 & 1 & 0 & 0 & 1 & 1 & \Rightarrow 19
\end{array}$$

Super Number — i^{th} bit unset

All other bits are set

Ans = N & Super Number

Super Number = ~(Magic Number)

= ~(1<<i)

Follow up question — HW

Given N & i → **Toggle** ith bit

If bit is 1 → 0

bit is 0 → 1

↳ One line solution only

Check bit

Given N and i, check if i th bit position is set or not.

Example

N = 21

i = 2

4	3	2	1	0
1	0	1	0	1

Set

Example

N = 34

i = 3

5	4	3	2	1	0
1	0	0	0	1	0

Unset

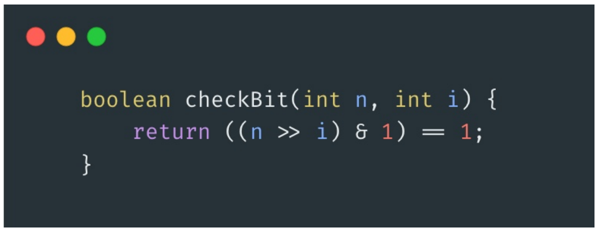
Idea

	6	5	4	3	2	1	0	
N = 82	1	0	1	0	0	1	0	
i = 0								(N&1) == 1
i = 1		1	0	1	0	0	1	((N>>1)&1) == 1
i = 2			1	0	1	0	0	((N>>2)&1) == 1

```
Boolean checkBit(int N, int i) {

        if ( ((N>>i) & 1) == 1 )
                return    true

        else
                return    false

}
```

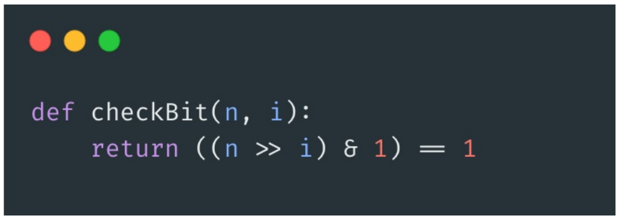
TC : O(1)
SC : O(1)

Java



```java
boolean checkBit(int n, int i) {
    return ((n >> i) & 1) == 1;
}
```

Python



```python
def checkBit(n, i):
    return ((n >> i) & 1) == 1
```

Can we also do it with left shift ?

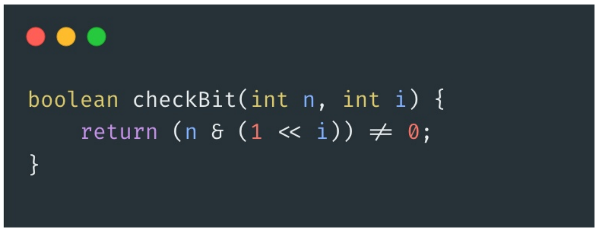
	6	5	4	3	2	1	0	
N= 82	1	0	1	0	0	1	0	
i=0	0	0	0	0	0	0	1	(N&1) ≠ 0
i= 1	0	0	0	0	0	1	0	N&(1<<1)≠0
i=2	0	0	0	0	1	0	0	N&(1<<2) ≠0

if res = 0 → ith bit is unset

res ≠ 0 → ith bit is set

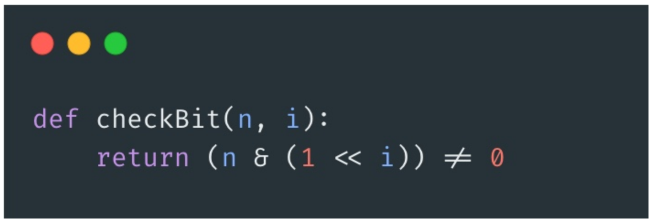
```
Boolean checkBit(int N, int i) {
        return (N & (1 << i) )!=0

}
```

Java



```
boolean checkBit(int n, int i) {
    return (n & (1 << i)) ≠ 0;
}
```

Python



```
def checkBit(n, i):
    return (n & (1 << i)) ≠ 0
```

Count bits

Qualcomm

box

Given an integer N, count how many set bits are there in N (Assume N to be a 32 bit integer)

Example

N = 10   1 0 1 0   ⇒ 2

Example

N = 27   1 1 0 1 1   → 4

Example

N = 45   1 0 1 1 0 1   ⇒ 4

```
int countSetBits(int N) {
        c = 0
        for (i = 0; i < 32; i++) {
                if ( checkBit(N, i) )
                        c++
        }
        return c
}
```

Iterate over all bits & check them

32 iterations

O(1) time

Idea 2

	5	4	3	2	1	0	c=0
N=45	1	0	1	1	0	1	c=1
$N>>1 = \frac{45}{2}$		1	0	1	1	0	c=1
$N>>2 = \frac{45}{2^2}$			1	0	1	1	c=2
$N>>3 = \frac{45}{2^3}$				1	0	1	
$N>>4 = \frac{45}{2^4}$					1	0	
$N>>5 = \frac{45}{2^5}$						1	
$N>>6 = \frac{45}{2^6}$						0	

```
int countSetBits(int N) {
        c=0
        while ( N>0 ) {
            if( (N&1) == 1 )
                c++
            N = N>>1
        }
        return c
}
```

TC : $O(\log_2 N)$

Which approach is better ?

Quiz 4

Approach 1 — 32 iterations — $O(1)$

Approach 2 — — $O(\log_2 N)$

According to Big O — 1 is better

In worst case, both approaches will take 32 iterations

↳ $N = 2^{32} - 1$

In every other case,

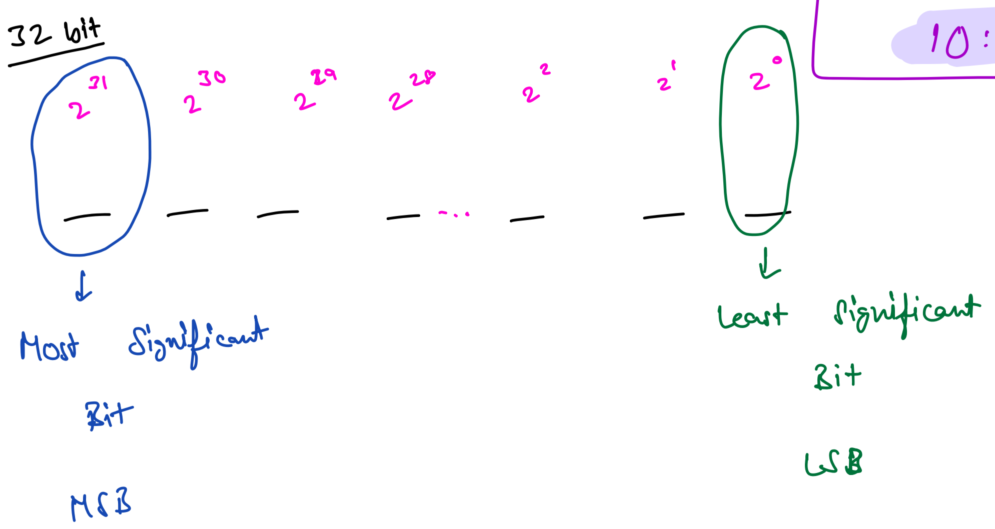
2^{nd} approach is better

Rare case — where Big O gives the wrong ans

Negative Numbers

Break till

10:30 PM



To store -ve numbers in our computers, we consider MSB base value to -ve.

To compute the -ve of a number, we store it in its 2's complement form

8 bit

		7	6	5	4	3	2	1	0
N = 45		0	0	1	0	1	1	0	1
~45	=	1	1	0	1	0	0	1	0
+ 1	+	0	0	0	0	0	0	0	1
		1	1	0	1	0	0	1	1

1) Take inverse / negation of N

2) Add 1 to it

MSB

$$\begin{array}{cccccccc} \downarrow & & & & & & & \\ 1 & 1 & 0 & 1 & 0 & 0 & 1 & 1 \\ \downarrow & \downarrow & & \downarrow & & & \downarrow & \downarrow \end{array}$$

$128 + 64 + 16 + 2 + 1 = 211$

$(-128) + 64 + 16 + 2 + 1 = -45$

Quiz 5 Convert the following N to decimal

4 bit

$$\begin{array}{cccc} \underline{-2^3} & 2^2 & 2^1 & 2^0 \\ 1 & 0 & 1 & 0 \\ \nearrow & & & \end{array}$$

MSB

Value $= -2^3 + 2^1$

$= -8 + 2 = -6$

Quiz 6

Convert the following N to decimal

4bit

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

$$-2^3 = -8$$

Quiz 7

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

$$2^4 + 2^2 + 2^0$$

$$= 16 + 4 + 1$$

$$= 21$$

MSB → 1 → -ve

0 → +ve

Quiz 8

$$\begin{array}{cccccccc} -2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \underline{1} & 0 & 0 & \underline{1} & 0 & \underline{1} & 0 & \underline{1} \end{array}$$

$$-2^7 + 2^4 + 2^2 + 2^0$$

$$= -128 + 21$$

$$= -107$$

Ranges

32 bit → Java, C++ - int

Min → 1 0 0 0 0 0 0 0 .... 0 0

= -2^{31} = -2,147,483,648

↓ Round off

≃ -2×10^{9} -2000000000

Max → 0 1 1 1 1 1 1 1 1 1 1

= $2^{31} - 1$ = 2,147,483,647

≃ 2×10^{9}

32 bit int → $\left[-2 \times 10^{9} , 2 \times 10^{9}\right]$

---

64 bit

C++, Java → long

Min → 1 0 0 0 0 0 0 0 ...0

= -2^{63} ≃ -9×10^{18} (Approx)

Max → 0 1 1 1 1 1 1

= $2^{63} - 1$ ≃ 9×10^{18} (Approx)

Range of long (64 bit int) → $\left[-9 \times 10^{18}, \; 9 \times 10^{18}\right]$

Importance of Constraints

Given an array of N elements, calculate sum of all elements.

Constraints

$1 <= N <= 10^5$

$1 <= A[i] <= 10^6$

```
long
~~int~~   sum = 0

for  i -> [0, N-1]
        sum += A[i]

return  sum
```

Worst case

10^5 terms

A[i] largest — 10^6

Max sum = $10^5 \times 10^6 = 10^{11}$

Can we store 10^{11} in 32 bit int ?

Overflow

Given two numbers, multiply them

Constraints

$1 <= a, b <= 10^6$

$$\text{int } c = a * b$$

Worst case

$$c = 10^6 \times 10^6 = 10^{12}$$

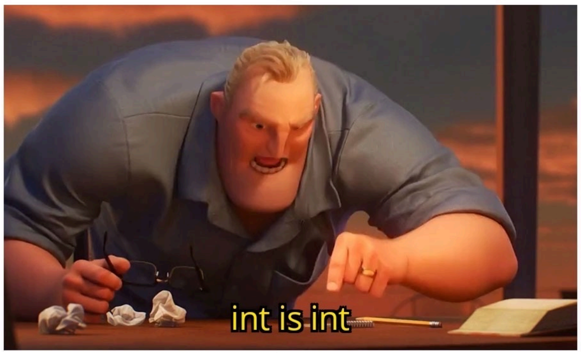
Overflow

$$\text{long } c = \underbrace{(\text{long})}_{\text{Typecast}} a * b$$

C++: int, long, long long

Java: byte, short, int, long, BigInteger

Python:



Even in Python, you shouldn't work with large numbers ($>= 10^{18}$)

↓

Calculations become very very slow

Doubts

Thank
You

10 | (1 << 3)

32 bit

10 = 0000 1 0 1 0

1<<3 = OR 0000 1 0 0 0

0000 1 0 1 0 = 10

int x = -1

8 bit

1 = 0 0 0 0 0 0 0 1

~1 = 1 1 1 1 1 1 1 0

+1 0 0 0 0 0 0 0 1

1 1 1 1 1 1 1 1

↓

-1

int - 32 bits

Min → 1 00 000 0

$= -2^{31}$

font → Jetbrains Mono

```
int abc(int []arr)

x ≠ 5
```

↑

font ligature

Good Night

Thank you

Monday