Bit Manipulation 1

Truth table for bitwise operator

a	b	a&b	a|b	a^b	~a
0	0	0	0	0	1
0	1	0	1	1	1
1	0	0	1	1	0
1	1	1	1	0	0

Basic Property:

AND

1) A & 1 → 1 (if A is odd)
→ 0 (if A is even)

A = 181

```
  10110101
& 00000001
----------
  00000001
```

↓ 1

A = 180

```
  10110100
& 00000001
----------
  00000000
```

↓ 0

2) A & 0 = 0

3) A & A = A

```
  10110100
& 10110100
----------
  10110100
```

OR

1) $A | 0 = A$

```
  1 0 1 1 0 0 1 0
| 0 0 0 0 0 0 0 0
-----------------
  1 0 1 1 0 0 1 0
```

2) $A | A = A$

```
  1 0 1 1 0 0 1 0
| 1 0 1 1 0 0 1 0
-----------------
  1 0 1 1 0 0 1 0
```

XOR

1) $A \wedge 0 = A$

```
  1 0 1 1 0 0 1 0
^ 0 0 0 0 0 0 0 0
-----------------
  1 0 1 1 0 0 1 0
```

2) A ^ A = 0

```
  1 0 1 1 0 0 1 0
^ 1 0 1 1 0 0 1 0
-----------------
  0 0 0 0 0 0 0 0
-----------------
```

Commutative property

Order of operands does not affect the result of bitwise operations.

A & B = B & A
A | B = B | A
A ^ B = B ^ A

3 & 4

4 & 3

3 & 4

100

& 011
& 100
000

& 011
000

7 & (5 & 3)

5 & 3
& 101
011
001

7 & 001
001 → 1

(7 & 5) & 3

7 & 5
& 111
101
101

& 101
011
001 → 1

Associative Property

Grouping of operands does not affect the result of the operation.

$$(A \& B) \& C = A \& (B \& C)$$

$$(A \mid B) \mid C = A \mid (B \mid C)$$

$$(A \hat{} B) \hat{} C = A \hat{} (B \hat{} C)$$

Quiz 1: $a \hat{} b \hat{} a \hat{} 0 \hat{} b$

↳ $a \hat{} a \hat{} b \hat{} b \hat{} 0$

$0 \hat{} 0 \hat{} 0$

$0 \hat{} 0$

0

Quiz 2: $1 \hat{} 3 \hat{} 5 \hat{} 3 \hat{} 2 \hat{} 1 \hat{} 5$

$1 \hat{} 1 \hat{} 3 \hat{} 3 \hat{} 5 \hat{} 5 \hat{} 2$

$0 \hat{} 0 \hat{} 0 \hat{} 2$

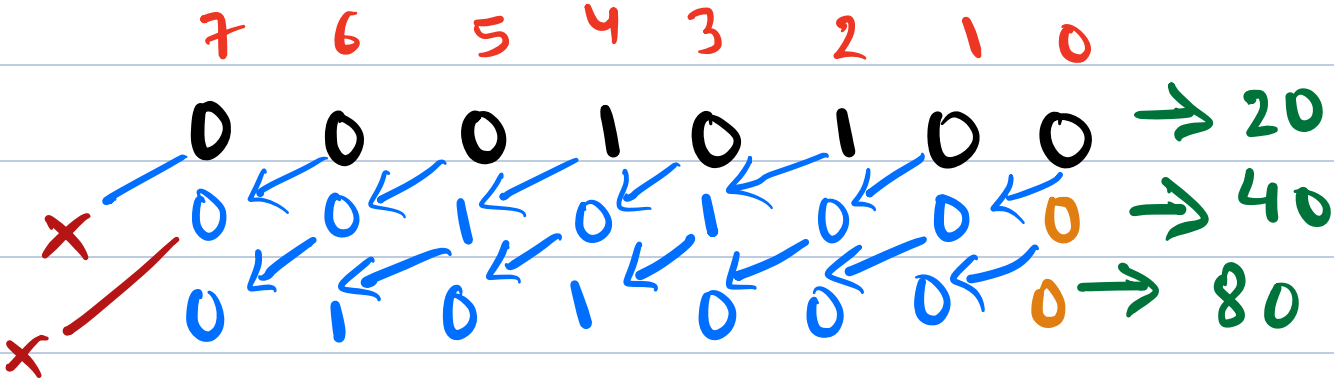
$= 2$

Left Shift Operator (<<) → It shifts the bits of a number to the left by a specified number of positions.

Eg.

Suppose we have 8 bit system.

$N = 20 << 2$


7 6 5 4 3 2 1 0
0 0 0 1 0 1 0 0
→ 20
0 0 1 0 1 0 0 0
→ 40
0 1 0 1 0 0 0 0
→ 80

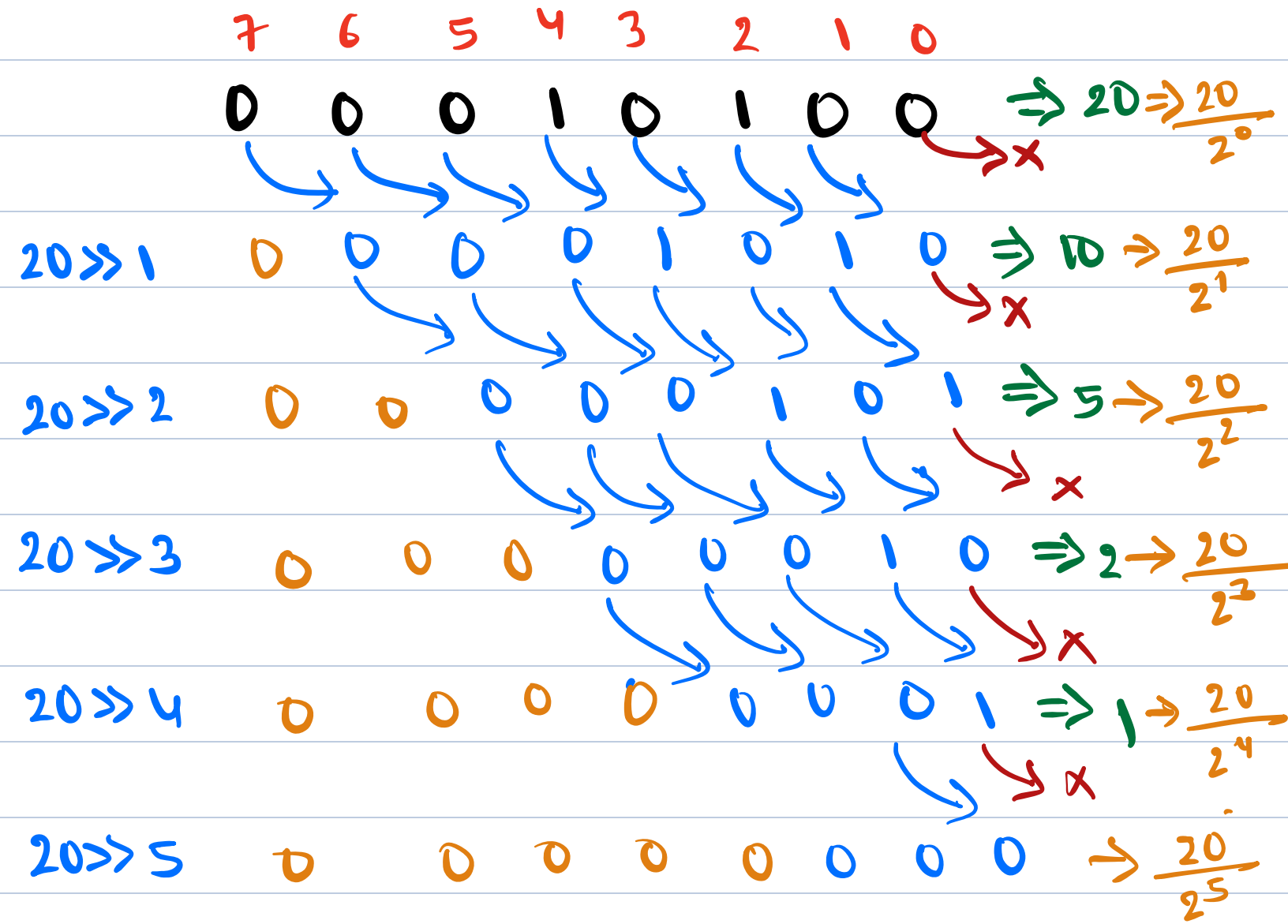

a=10
0 0 0 0 1 0 1 0 ⇒ 10
= 10 * 2^0
a<<1
0 0 0 1 0 1 0 0 ⇒ 20
= 10 * 2^1
a<<2
0 0 1 0 1 0 0 0 ⇒ 40
= 10 * 2^2
a<<3
0 1 0 1 0 0 0 0 ⇒ 80
10 * 2^3
a<<4
1 0 1 0 0 0 0 0 ⇒ 160
10 * 2^4
a<<5
0 1 0 0 0 0 0 0 ⇒ 64
(overflow happened)

$$a << n = a * 2^n$$

$$1 << n = 2^n$$

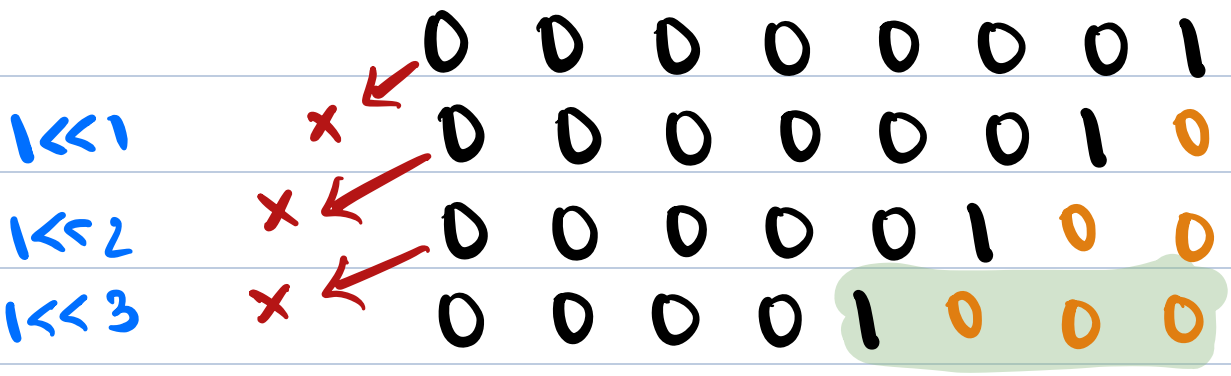
Right shift operator (>>) It shifts the bits of a number to the right by a specified number of positions

Eg. 20 >> 2



$$a >> n = a / 2^n$$

Quiz 3: $1 << 3 = ?$



Power of Left Shift operator

1) Set the i^{th} bit of a number using OR(|) Operator.

$N = N \mid (1 << i)$ $\quad i = 2$

Eg.

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

i = 2

Eg.

5	4	3	2	1	0	
1	0	1	0	1	0	
0	0	0	0	0	1	(1)
0	0	0	1	0	0	(1<<2)

	1	0	1	0	1	0 → A
OR	0	0	0	1	0	0 → (1<<2)
	1	0	1	1	1	0

2) Toggle the i^{th} bit of a number with XOR(^) operator.

$$N = N \wedge (1 << i)$$

i = 2

	5	4	3	2	1	0	
	1	0	1	1	0	1	→ A
XOR	0	0	0	1	0	0	→ (1<<2)
	1	0	1	0	0	1	

i = 2

```
  1 0 1 0 1 0 → A
XOR 0 0 0 1 0 0 → (1<<2)
    ---------
    1 0 1 1 1 0
```

3) Check the status of i^{th} bit using & operator.

N = N & (1<<i) → 0 (if it's unset)

→ >0 (it is set bit)

```
  1 0 1 1 0 1 → A   i = 2
& 0 0 0 1 0 0 → (1<<2)
  -----------
  0 0 0 1 0 0
```

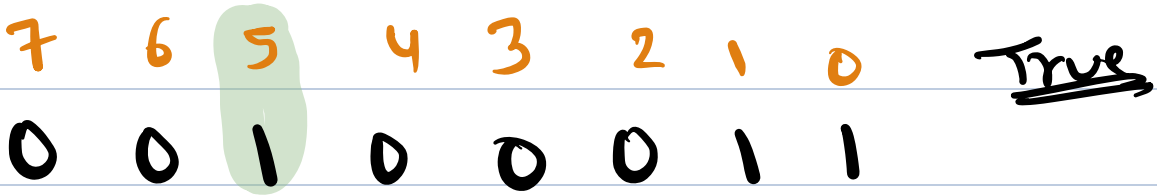
i = 2

```
    1 0 1 0 0 1 → A
AND 0 0 0 1 0 0 → (1<<2)
    -----------
    0 0 0 0 0 0
```

Q.) Given a number N & index i. check Whether the bit i is set or not.

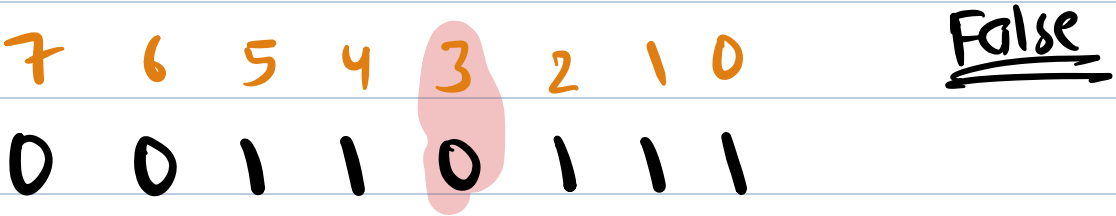
Eg.

N = 35 i = 5



Eg

N = 55 i = 3



Code:

```
boolean checkBit(int N, int i)
{
    if (N & (1<<i) > 0)
    {
        return true;
    }
    return false;
}
```

TC: O(1)

N = 10
i = 3

0 0 0 1 0 1 0

i = 1	0 0 0 0 0 0 1
i<<1	0 0 0 0 0 1 0
i<<2	0 0 0 0 1 0 0
i<<3	0 0 0 1 0 0 0

AND

0 0 0 1 0 1 0
0 0 0 1 0 0 0
---
0 0 0 1 0 0 0

True

Q. Count the total SET bits in N.

Eg. N = 12

0 0 0 ... 1 1 0 0 Ans = 2

Code:

Approach 1:

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

Approach 2: Use the right shift operator

×2

ans=0 N=12

```
ans = 0
while(N>0)
{
  if(N&1>0)
  {
    anstt;
  }
  N = N>>1;
}
return ans;
```

```
...0 0 0 1 1 0 0   ✗
...0 0 0 0 1 1 0   ✗
...0 0 0 0 0 1 1   ✗
...0 0 0 0 0 0 1   ✗
...0 0 0 0 0 0 0
```

O(1) → O(log N)

Scenerio

IRCTC (India's train ticketing system) wants to improve how it shows train options to its users. They've decided that trains which run more **frequently** should appear higher up in the search results. To figure this out, they look at a **28-day period** to see how often each train runs.

Problem

For **each** train, they've come up with a **special number**. This isn't just any number, though. If you were to write it down in binary form (which is like a special code of 0s and 1s), each of the 28 **digits** corresponds to a day in that **period**. A **'1'** means the train runs on that day, and a **'0'** means it doesn't.

Task

Your task is to help **IRCTC** by writing a program. Given a list **A** of these **special numbers** for different **trains**, your program should find the train that runs the most.

A = [4369, 8738, 349525]

Train No. (Index)	Binary Representation	Count Set bits
0	0000000000000001000100000001	3
1	0000000000000010001000000010	3
2	0000000101010101010101010101	14

A = [255, 127, 62]

Train No. (Index)	Binary Representation	Count Set bits
0	00000000000000000000011111111	8
1	00000000000000000000001111111	7
2	00000000000000000000000111111	6

Break : 10.46 PM

Q. Unset the i^{th} bit of a number N if it is is set.

Eg. N = 6 i = 2

```
3 2 1 0
0 1 1 0  →  0 0 1 0 ⇒ 2
```

Approach: 1) check Bit.

2) If it is set → toggle it to unset.

Code:

N = 25 i = 3

```
4 3 2 1 0
1 1 0 0 1  → N
0 1 0 0 0  → (1<<3)
-----------
1 0 0 0 1
```

^

```
int unset(int N, int i)
{
    if (checkBit(N,i) == True)
    {
        N = N ^ (1<<i);
    }
    return N;
}
```

TC: O(1)

A group of computer scientists is working on a project that involves encoding binary numbers. They need to create a binary number with a specific pattern for their project. The pattern requires A 0's followed by B 1's followed by C 0's. To simplify the process, they need a function that takes A, B, and C as inputs and returns the decimal value of the resulting binary number. Can you help them by writing a function that can solve this problem efficiently?

Constraints:

```
0 <= A, B, C <= 20
```

Eg. A = 4 B = 3 C = 2

⇒ 28

Eg. A = 3 B = 1 C = 0

0 0 0 1 ⇒ 1

Eg. A = 2 B = 1 C = 3

0 0 1 0 0 0 ⇒ 8

8 bit system

0 0 0 0 0 0 0 0

A B C

0 0 0 0 1 1 1 0 0 ⇒ 28

8 7 6 5 4 3 2 1 0

A=4 B=3 C=2

C+B-1 C C-1 0

0 0 0 0 0 ... 0 1 1 1 1 ... 1 0 0 0 0

```
long solve (int A, int B, int C)
{
    long ans = 0
    for (i=C; i<=C+B-1; i++)
    {
        ans = ans | (1<<i);
    }
    return ans;
}
```

A=3, B=3, C=1

TC: O(B)

i = ~~2~~ 3

```
          7  6  5  4  3  2  1  0
OR        0  0  0  0  0  0  0  0   <- ans
          0  0  0  0  0  0  1  0   (1<<1)
          ------------------------
          0  0  0  0  0  0  1  0

OR        0  0  0  0  0  0  1  0   <- ans
          0  0  0  0  0  1  0  0   (1<<2)
          ------------------------
          0  0  0  0  0  1  1  0

          0  0  0  0  0  1  1  0   <- ans
          0  0  0  0  1  0  0  0   (1<<3)
          ------------------------
          0  0  0  0  1  1  1  0
```

14