Agenda

1) Bitwise operators
2) Basic &, |, ^ properties
3) Left shift operator
4) right shift operator
5) power of left shift operator
   - set i^{th} bit
   - toggle i^{th} bit
   - unset i^{th} bit
6) Questions

Bitwise Operators

And (&), OR (|), XOR (^), NOT (~)

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

Interesting properties (consider A as a number)

Gives rightmost bit of A

1) A & 1

14 → 1 1 1 0
1 → & 0 0 0 1
0 0 0 0

13 → 1 1 0 1
1 → & 0 0 0 1
0 0 0 1

odd

0th bit 1 → any of A & 1 is 1

0th bit 0 → any of A & 1 is 0

even

```
if ((A & 1) == 0) {
    print ("Even");
}
else {
    print ("Odd");
}
```

2) A & 0 → 0

A & A → A

A | 0 → A

A | A → A

A ^ 0 → A

A ^ A → 0

a ^ b ^ a ^ d ^ b

→ a ^ a ^ b ^ b ^ d

0 ^ 0 ^ d

0 ^ d ⇒ d

3) Commutative (A, B → no.)

A & B = B & A

A | B = B | A

A ^ B = B ^ A

4) Associative (A, B, C → no.)

A & (B & C) = (A & B) & C

A | (B | C) = (A | B) | C

A ^ (B ^ C) = (A ^ B) ^ C

---

1 ^ 3 ^ 5 ^ 3 ^ 2 ^ 1 ^ 5

= 1 ^ 1 ^ 3 ^ 3 ^ 5 ^ 5 ^ 2

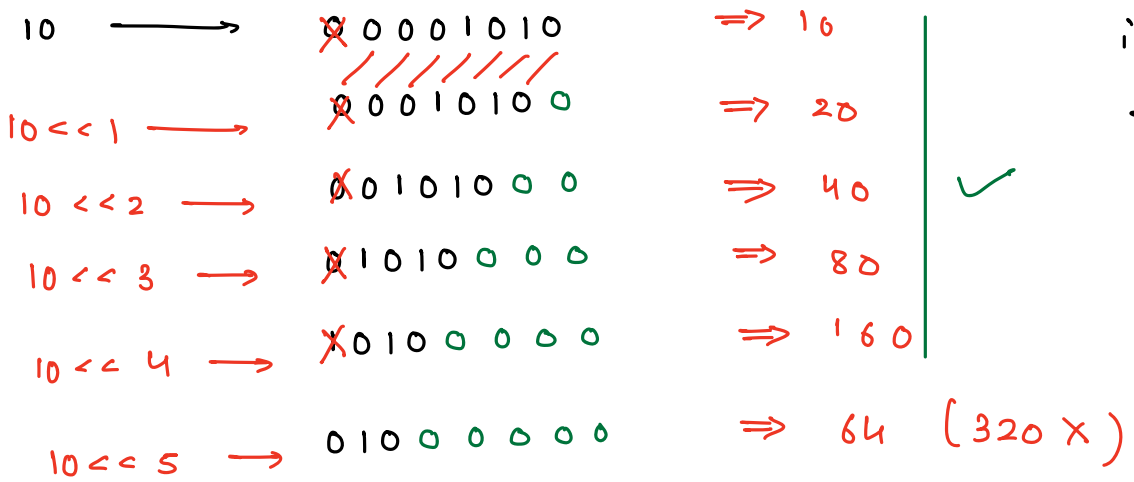
0 ^ 0 ^ 0 ^ 2

= 2

print(8 | 5); // 13

left shift operator (<<)

→ assume 10 in 8-bit format



every left shift is multiplying the no. by 2.

left shift (overflow can happen)

↳ range (n bit no. -2^{n-1} to $2^{n-1}-1$)

$$A << K \Rightarrow A * 2^K$$

⇓ (if A==1)

$$1 << K \Rightarrow 2^K$$

→ O(1) calculation

It also means that only the ith bit is ON, rest everything is OFF.

$12 << 3 \Rightarrow 12 * 2^3 = 96$

12 → 24 → 48 → 96

Right shift operator (>>)

every right shift is dividing the no. by 2.

→ assume 20 in 8-bit format

20 ⟶ 00010100 ⇒ 20

20 >> 1 ⟶ 00001010 ⇒ 10

20 >> 2 ⟶ 00000101 ⇒ 5

20 >> 3 ⟶ 00000010 ⇒ 2

20 >> 4 ⟶ 00000001 ⇒ 1

20 >> 5 ⟶ 00000000 ⇒ 0

$$A >> k = \frac{A}{2^k}$$

$25 >> 3 \Rightarrow \frac{25}{8} = 3$

12

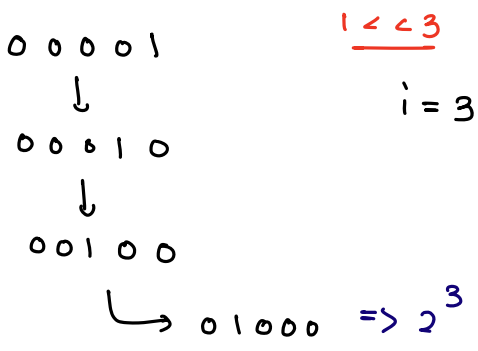
6

3

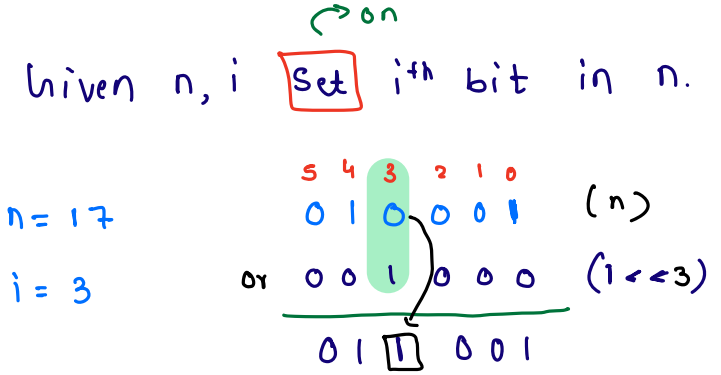
Power of left shift operator

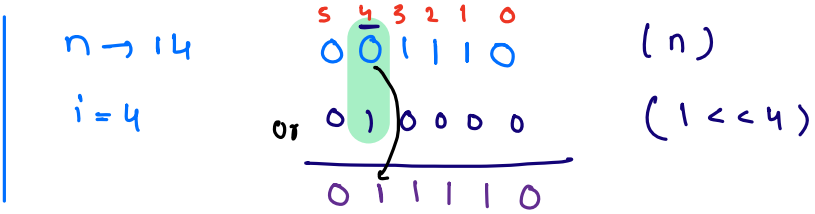
what is $1 << i = 2^i$

i^{th} bit is on



1) Given n, i Set i^{th} bit in n.



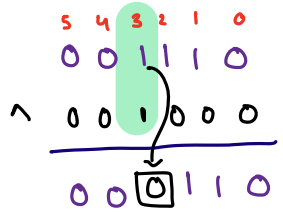



```
function set ( n, i ) {
    n = n | (1<<i);
    return n;
}
```

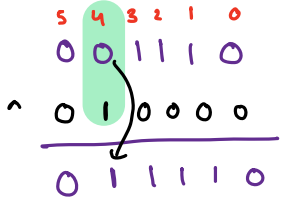
TC: O(1)

2) toggle i^{th} bit of n.

n = 14, i = 3



n = 14, i = 4

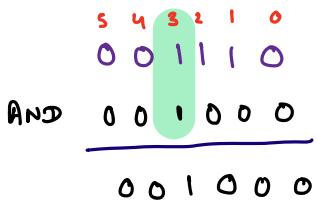



```
function toggle (n, i) {
    n = n ^ (1<<i);
    return n;
}
```

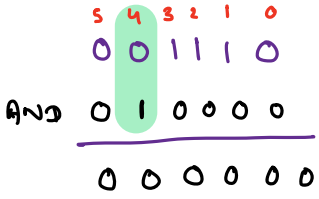
TC: O(1)

3) check i^{th} bit of n

n = 14, i = 3



n = 14, i = 4




```
function checkbit (n, i) {
    if ( (n & (1<<i)) == 0 ) {
        return false; // the ith bit is off
    }
    else {
        return true; // the ith bit is on
    }
}
```

TC: O(1)

4) unset i^{th} bit of n

↓

off

```
function unset(n, i) {
    if (checkbit(n, i) == true) {
        n = n ^ (1<<i);
        return n;
    }
    else {
        return n;
    }
}
```

TC: O(1)

exercise: 4 case, revise them using eg.

Q.1 Given n, calculate total no. of set bits. (n → 32 bit no.)

↳ on (1)

n = 10 00 1 0 1 0 ans = 2

n = 13 0 1 1 0 1 ans = 3

logic: go on every bit 0 to 31, check if it is on or not.
if it is on, increase count variable.

```
function countSetBits ( n ) {                    TC : O(1)

    count = 0;

    for ( i → 0 to 31 ) {
    |   //check ith bit of n is set
    |   if ( (n & (1<<i)) != 0) {
    |           count++;
    |   }
    }

    return count;
}
```

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.

0 1 2

Input 1 : A = $[4369, 8738, 349525]$

Output 1 : $[2]$

Train No. (Index)	Binary Representation	# Count of set bits
0	00000000000000010001000000001	3
1	00000000000000010001000000010	3
2	00000001010101010101010101	11

A = [10 , 19 , 12 , 7]
0 1 2 3

0 → 001010 → 2

1 → 010011 → 3

2 → 001100 → 2

3 → 000111 → 3

ans = [1, 3]

```
Junction train (A) n {

    maxCount = 0;

    for (i → 0 to n-1) {
        count = countSetBits(A[i]);
        if (count > maxCount) {
            maxCount = count;
        }
    }

    ans = [ ];

    for (i → 0 to n-1) {
        count = countSetBits(A[i]);
        if (count == maxCount) {
            ans.add(i);
        }
    }
}
```

TC: O(n)

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?

⇒ A, B, C are given. Build a no. formed by A 0's followed by B 1's and C 0's

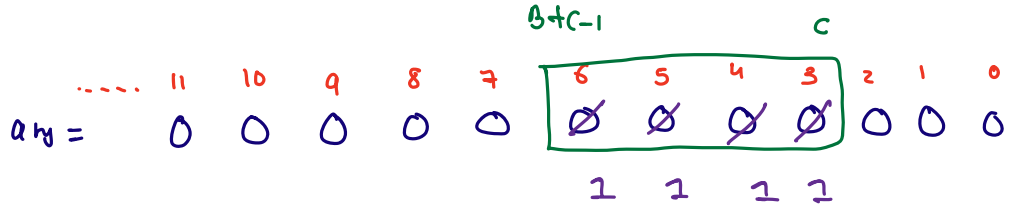
A = 4, B = 3, C = 2

ans = 28

⇓

ans = 0 (set bits from C to B+C-1)

A = 5 B = 4 C = 3




```
function solve (A, B, C) {
    ans = 0;
    // set bits from C to B+C-1
    for (i → C to B+C-1) {
        ans = ans | (1<<i);
    }
    return ans;
}
```