Class Objectives

1. Understanding Bitwise Operators
2. Key points to keep in mind while using bitwise operators
3. Solving Problems that involve bitwise as well as all topics we have covered so far.

By default we will deal with SIGNED numbers only.

Bitwise OR: |
Bitwise AND: &
Bitwise XOR: ^
Bitwise 1s One's Complement: ~
Left Shift: <<
Right Shift: >>

A	**B**	**A | B**	**A & B**	**A ^ B**	**~A**
0	0	0	0	0	1
0	1	1	0	**1**	1
1	0	1	0	**1**	0
1	1	1	1	0	0

A	**B**	**A | B**	**A & B**	**A ^ B**	**~A**
25	43	59	9	50	-26
0001 1001	0010 1011	0011 1011	0000 1001	0011 0010	1110 0110

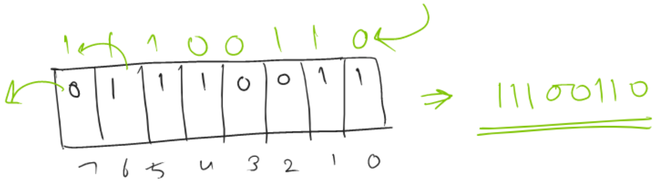
~The answer depends on whether A is signed or unsigned.
If A is UNSIGNED 8-bit number: ~A = **1**110 0110 = 128 + 64 + 32 + 4 + 2 = 230
If A is SIGNED 8-bit number: ~A = 1110 0110 = -128 + 64 + 32 + 4 + 2 = -26

-A = Two's Complement of A = Ones Complement of A + 1 = ~A + 1
-A = ~A + 1
-A - 1 = ~A

A	0001	1001	
B	0010	1011	
A | B	0011	1011	⇒
A&B	0000	1001	⇒ 9
A^B	0011	0010	⇒ 50

Assume A is 8-BIT UNSIGNED NUMBER.

A	25 (0001 1001)	**B**	5 (0000 0101)
A >> 1	12	**B << 1**	10
A >> 2	6	**B << 2**	20
A >> 3	3	**B << 3**	40
A >> k	$A/2^k$	**B << k**	$B * 2^k$



Left Shift: • Move every bit one pos left
• 7th pos bit will be dropped
• 0th pos bit will be filled with 0

Right Shift: • Move every bit one pos Right
• 0th bit is dropped
• 7th bit will be filled with a 0] UNSIGNED

TODO EXERCISE FOR YOU: Figure out what happens on Left and Right Shift of a SIGNED Number.

Thinking Exercise: Given any number N, can we write N as a sum of powers of 2 (non-repeated)?

47 = 0010 1111 = $2^5 + 2^3 + 2^2 + 2^1 + 2^0$ = **YES**

Can any number N be written as the sum of powers of another number X (non-repeated)? => NO

Can any number N be written as the sum of powers of another number X (repeated)? => YES

47 as a sum of powers of 5?

$$\begin{array}{c|l} 5 & 47 \\ \hline 5 & 9 - 2 \\ \hline 5 & 1 - 4 \\ \hline & 0 - 1 \end{array}$$

$$(47)_{10} = (142)_5$$

$$\boxed{1\times5^2 + \overset{\text{coeff}}{4}\times5^1 + 2\times5^0}$$

$$25 + 20 + 2 = 47$$

Bit Manipulation is about Writing and Observing and Leveraging Our Knowledge

A ^ A	**0**	**A ^ 0**	**A**	**A ^ 1**	**$A + (-1)^A$**
A & A	**A**	**A & 0**	**0**	**A & 1**	**A % 2**
A | A	**A**	**A | 0**	**A**	**A | 1**	**A + (A+1)%2**

A ^ 1 => A + 1 for EVEN, A - 1 for ODD
A & 1 => 1 for ODD, 0 for EVEN.
A | 1 = A for ODD, A + 1 for EVEN.

A + 1 - (A%2)
A + (A+1)%2

Given A = 10, B = 5, C = 4, Confirm if below expressions hold true

1. **A & B = B & A** **// COMMUTATIVE   A|B and A^B**
2. **(A & B) & C = A & (B & C) = (A & C) & B** **// ASSOCIATIVE**

POINTS TO KEEP IN MIND

1. Bitwise Operators are COMMUTATIVE and ASSOCIATIVE - Order of performing operations doesn't matter.
2. Bitwise Operators have LESSER PRECEDENCE than many other operators so use them with brackets.
3. Bitwise Operators are faster than ARITHMETIC OPERATORS.
    a. if(a%2==0) print "EVEN"
    b. if(**(a&1)==** 0) print ""EVEN" ⇒ Faster than previous.

MASKING A NUMBER

Given N, we want only the 0th and 3rd bit to be extracted in another number.

N = 0001 **1**01**0**

X = 0000 1000

Get X such that X has only 0th and 3rd bit of N Extracted, every other bit must be 0.

X = N & (A Number having 0th and 3rd bit set) = N & (0000 1001) = N & 9

9 is called MASK because it helps us extract only those bits of N which are set in 9.

[PROBLEM] Given X and Y, create a 8-bit binary number whose Xth and Yth bit only is set.

X = 2, Y = 4	0001 0100 = 20
X = 5, Y = 5	0010 0000 = 32
X = 5, Y = 3	

~~int getXthAndYthBitSetNumber(int X, int Y){~~
~~return (1<<X) + (1<<Y);~~
~~}~~

```
int getXthAndYthBitSetNumber(int X, int Y){
      return (1<<X) | (1<<Y);
}
```

```
int getXthAndYthBitSetNumber(int X, int Y){
      int n = (1 << max(x, y));
      n = n | (1 << min(x, y));
      return n;
}
```

~~int getXthAndYthBitSetNumber(int X, int Y){~~
~~int n = (1 << abs(x - y)); //~~
~~n = n << min(x, y);~~
~~return n;~~
~~}~~

[PROBLEM] Given X and Y, create a 32-bit binary number that has X 1s followed by Y 0s in it in binary form.

X = 3, Y = 2	11100 ⇒ 16 + 8 + 4 = 28
X = 5, Y = 1	111110 ⇒ 62
X = 4, Y = 0	1111 ⇒ 15

```
int getXOnesFollowedByYZeroes(int X, int Y){
     int n = ((1 << X) - 1);
     n = (n << Y);
     return n;
}
```

```
int getXOnesFollowedByYZeroes(int X, int Y){
    int n =(1 << (X+Y)) - (1 << Y);
    return n;
}
```

N = 2^k ⇒ (1 << k) = 100000 (k=5)
N-1 = 2^k - 1 = 011111 (k=5)

N can be a 64-bit number.
K can be upto 60

```
// n = 5, k = 0 (right most bit is 0th bit) ⇒ TRUE
// n = 20, k = 2   ⇒ (0001 0100) ⇒ TRUE
// n = 31, k = 7   ⇒ (0001 1111) ⇒ FALSE

// Convert to Binary Form (string or array), Check Kth Position

bool isKthBitSet(long long n, int k){
        return ((n >> k) & 1) == 1;    // return ((n >> k) & 1) != 0;
}
```

```
bool isKthBitSet(long long n, int k){
        return (n & (1LL << k)) != 0;    // return (n & (1<<k)) == (1 << k);
}
```

```
bool isKthBitSet(long long int n, int k){
        return (n | (1LL << k)) == n;
}
```

[PROBLEM] Given N and K, set Kth bit in N and return the updated number.

0 <= n <= 10^{18}
0 <= k <= 60

N = 32, K = 0,   RETURN 33
N = 33, K = 0,   RETURN 33

```
long long setKthBit(long long N, int k){
       return ( (1LL << k) | N);
}
```

```
long long setKthBit(long long N, int k){
       return (((N>>k) | 1)<<k) | N;
}
```

[PROBLEM] Given N and K, toggle Kth Bit

N = 32, k = 0,   ANS⇒ 33
N = 32, k = 5,   ANS ⇒ 0
N = 18, k = 3,   ANS ⇒ (0001 0010) toggle kth => 26

```
long long toggleKthBit(long long N, int k){
   long long ans = (1LL << k) ^ N;
   return ans;
}
```

PROBLEM SOLVING SESSION

[PROBLEM 1] Given a string S, tell if all characters of S are distinct or not.
0 <= len(S) <= 10^5
S can have any characters - lowercase, upper case, digits, special chars (#, @ …)

S = “alphabet”, ANS = FALSE
S = “pink”, ANS = TRUE

```
bool hasDistinctChars(string s){
}
```

Approach 1: (TC: ALPHABET_SIZE (256), SC: ALPHABET_SIZE)

1. map<char, int> => Freq of every character
2. Loop over string and keep incrementing the freq of each character, at any moment if freq of char becomes > 1 return false

Approach 1.1: Use set in place of map.
Space wise set will be a better choice than a map.

Approach 1.2: **Use a boolean array of size 256** (ALPHABET SIZE), all 0s initially. For each character increment its frequency by going to arra[idx] where idx is ASCII value of char.
HASHTABLE (DAT - Direct Access Table)

What if NO EXTRA SPACE can be taken?

TC: N^2 very high upper bound because charset size is 256 we will do better than N^2

Assuming only lowercase alphabets

```
bool hasDistinctChars(string s){
        for(int i=0; i < s.length(); i++)
            for(int j = i + 1; j < s.length(); j++)
                    if(s[i] == s[j])
                         return false;
       return true;
}
```

```
bool hasDistinctChars(string s){
        for(int j = 0; j <= 255; j++){
            char curChar = (char)j;
            int freqC = 0;
            for(int i = 0; i < s.length(); i++){
                    if(s[i] == curChar )
                       freqC++;
                    if(freqC > 1) return false;
            }
       }
       return true;
}
```

```
bool hasDistinctChars(string s){
        if(s.length() > 256) return false;
        for(int j = 0; j <= 255; j++){
            char curChar = (char)j;
            int freqC = 0;
            for(int i = 0; i < s.length(); i++){
                    if(s[i] == curChar )
                       freqC++;
                    if(freqC > 1) return false;
            }
       }
       return true;
}
```

Assuming string has only lower case alphabets

APPROACH 3: Sort the string and compare adj characters.

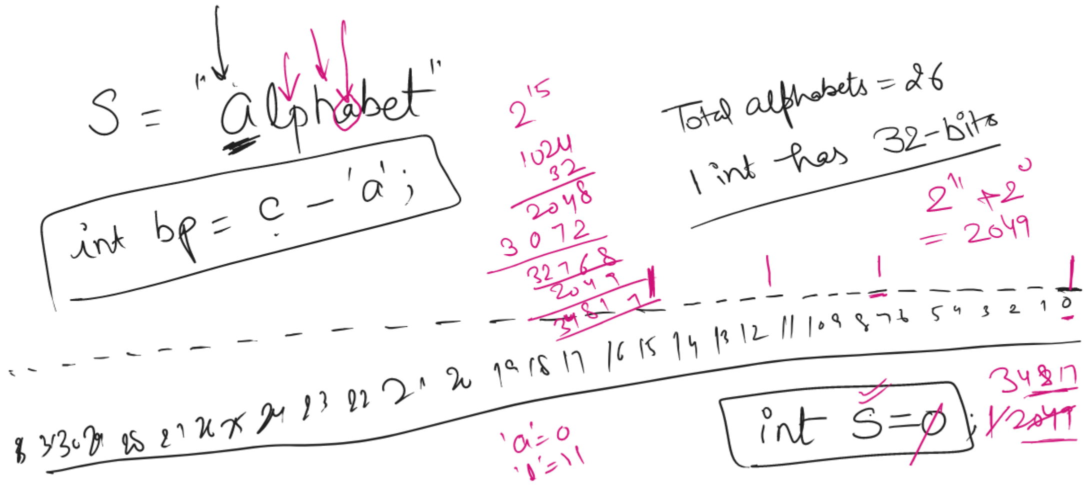
S = “alphabet” , SS = “aabehlpt”
TC: NlogN + N = O(NlogN)
SC: 1

APPROACH 4: Use one integer as a SET

S contains only lower case alphabets

bool isKthBitSet(long long n, int k);
long long setKthBit(long long n, int k);

```
bool hasDistinctCharacters(string s){
        long long intSet = 0;
        for(char c in s){
            int bitPos = c - ‘a’;
            if( isKthBitSet(intSet, bitPos) )
                  return false;
            intSet = setKthBit(intSet, bitPos);
        }
        return true;
}
```



Only lower case alphabets: 1 integer is good enough (32-bits)
Lowercase and uppercase alphabets: 1 long long is good enough (64-bits)
LowerCase, upperCase and Digits :   1 long long for alphabets, 1 for digits?
If it can be any char of 256 chars, we need long long ints to make a set.

0:63 => first long long
64:127 => second long long
128:191 => third long long
192:255 => fourth long long

[PROBLEM] Given an array having every integer repeated 2 times except one integer that occurs only once. Find out non-repeating integers.

1 <= N <= 10^6
0 <= A[i] <= 10^9

A = {1, 3, **1**, 9, 2, 6, 6, 2, 3, 11, 4, 11, 4}

ANS = 9

Approach 1:
Nested Loops approach
For each A[i] search from beginning and count how many times it occurs

TC: N^2, SC: 1

Approach 2:
Map with freq of each value
1 loop to update freq of every value
Second loop to find which value occurs only once
TC: 2N, SC: N

Approach 3:
Set of ints
For each value check in the set

1. If exists remove
2. else add

Finally, set should have one value left, that is the answer.
TC: N, SC: N/2

A = {1, 2, 3, 4, 5, 6, 5, 4, 3, 2, 1}

Approach 4
Sort and Solve
Sort the array
Compare adj elements to find the single occurring element
TC: NlogN + N = O(NlogN)
SC: 1

4.1 If we do binary search after sorting NlogN + logN = O(NlogN)

Approach 5
Bitwise XOR of all array elements.
-> XOR is ASSOCIATIVE
-> A^A = 0, A^0 = A
Calculate XOR of All elements of array, final answer will be desired value.

TC: N, SC: 1

[PROBLEM] Given an array of size N which has distinct values in range [1, N+1] one element from range [1, N+1] is missing. Find the missing element.

N = 10
A = {9, 2, 11, 6, 5, 4, 1, 7, 8, 10}

ANS = 3

We know sum of 1 to N+1, using formula
Sum of 1 to N+1 - Sum of all array elements = MISSING NUMBER

TC: N, SC: 1

What will be the result of XOR of 1 to 11 and XOR of all elements of the array.

SUBSET GENERATION (Combinations)

A = {2, 9, 3}, N = 3

2^N subsets or combinations
N! permutations

How many subsets will this array have?
{}
{2}
{9}
{3}
{2, 9}
{9, 3}
{2, 3}
{2, 3, 9}

2	1	0
3	9	2

0	0	0	0	→	{ }
1	0	0	1	→	{2}
2	0	1	0	→	{9}
3	0	1	1	⇒	{2, 9}
4	1	0	0	→	{3}
5	1	0	1	→	{2, 3}
6	1	1	0	→	{9, 3}
7	1	1	1	→	{2, 3, 9}

```
// Avoid Printing Empty Subset
// 2
// 9
// 2 9
// 3
// 2 3
// 9 3
// 2 3 9
void printAllSubsets(list<int> a, int n){
        totalSubsets = (1 << N);
        for i = [1, totalSubsets-1]
             for bp = [0, N-1]
                  if( isKthBitSet(i, bp) )
                      print a[bp],” “
             print “\n”
}
```

TC: $N * 2^N$

SC: 1

N = 10, will the above code get accepted? $10 * 2^{10}$ =10 * 10^3 = 10^4 => **ACCEPTED**
N = 20, will above code complexity get accepted? 10^7 ⇒ Accepted
N = 40, will above code complexity get accepted? **NO**

Complete before exiting session
https://leetcode.com/problems/largest-combination-with-bitwise-and-greater-than-zero
https://leetcode.com/problems/subsets
https://leetcode.com/problems/count-number-of-maximum-bitwise-or-subsets/

16	0	0	0	1	0	0	0	0	
17	0	0	0	1	0	0	0	1	
71	0	1	0	0	0	1	1	1	✓
62	0	0	1	1	1	1	1	0	✓
12	0	0	0	0	1	1	0	0	✓
24	0	0	0	1	1	0	0	0	
14	0	0	0	0	1	1	1	0	✓