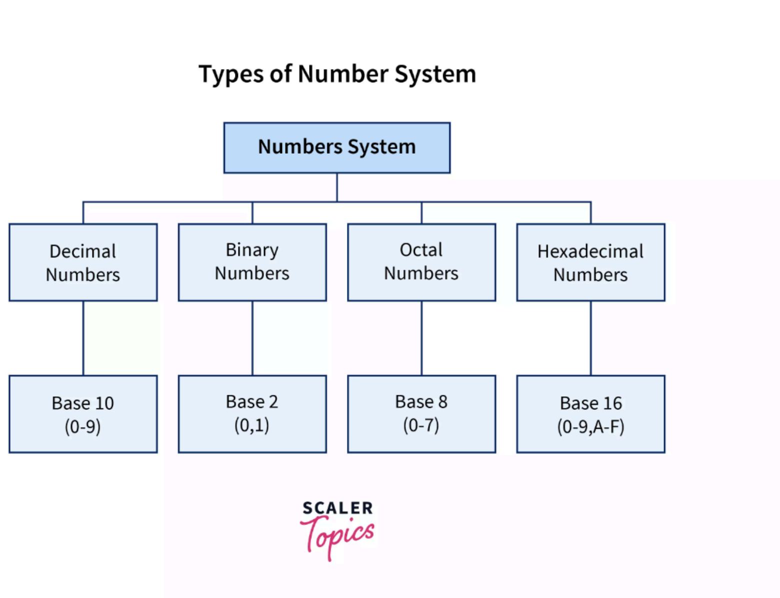
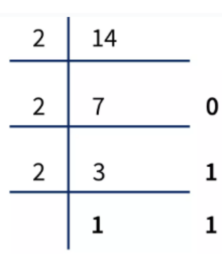
Bit Manipulation

Basics:

1. A number system or the numeral system is a way of representing numbers.The number system is one of the most important concepts of mathematics and a building block for programming as it tells us how the data is represented in various formats according to the situation and usage.



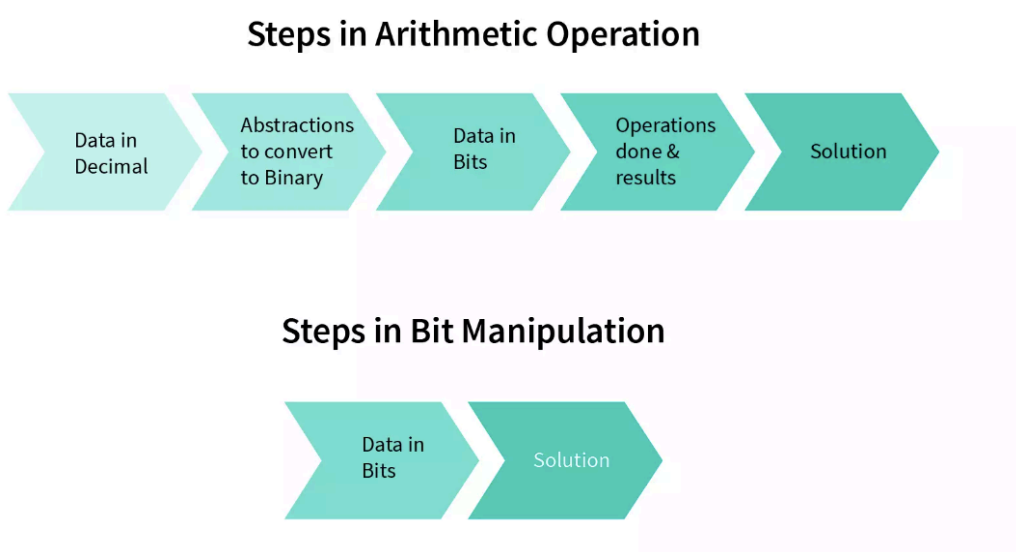
2. **Binary No. System**
The binary number system or the base 2 number system uses only two digits i.e. 0 and 1 to represent any number. For example, the **decimal number 14** is represented as **1110 in the binary number system**. We need to keep dividing the decimal number by 2 until the quotient becomes 0. When the remainders obtained in the division are represented in reverse order, a binary number generated.



Base 2 Number System Example

Practice question - [Decimal to Binary]

3. **Bitwise Operators**
Bitwise operators are one among those operators which perform their tasks at binary level directly on **binary digits** (also known as bits), the smallest form of data in a computer, or its data types. These bitwise operators are faster and an efficient way to interact with computers to make heavy computation in a linear time because it works directly with the bits rather than through a level of abstraction of software.



4. **Types of Bitwise Operators**

Operators	**Symbol**
Bitwise AND	&
Bitwise OR	|
Bitwise XOR	^
Bitwise Not	~
Left shift operator	<<
Right shift operator	>>

5. **Truth Table for AND, OR, XOR operators**

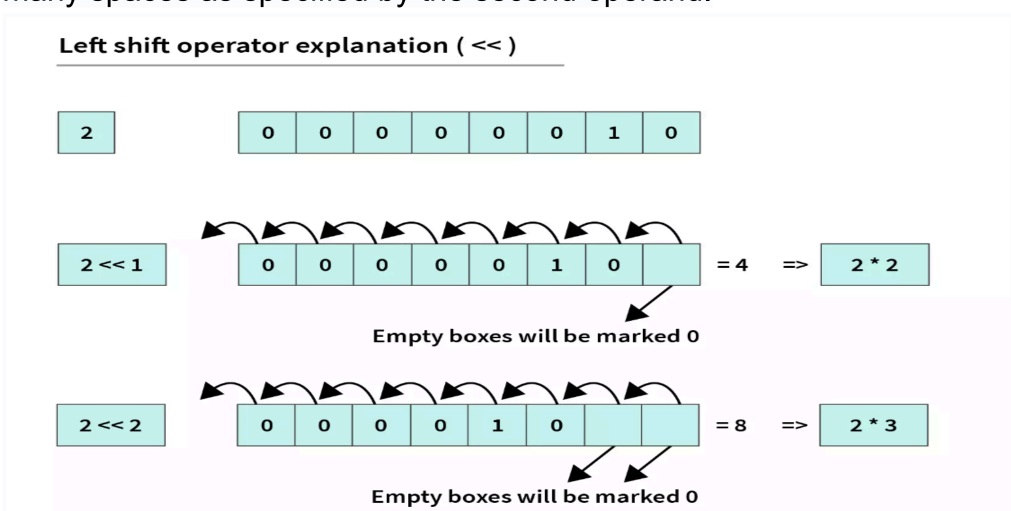
x	y	x&y	x|y	x^y
0	0	0	0	0
0	1	0	1	1
1	0	0	1	1
1	1	1	1	0

Practice Question - [Reverse Bits]()
- [Swap Bits]()

Shift operators

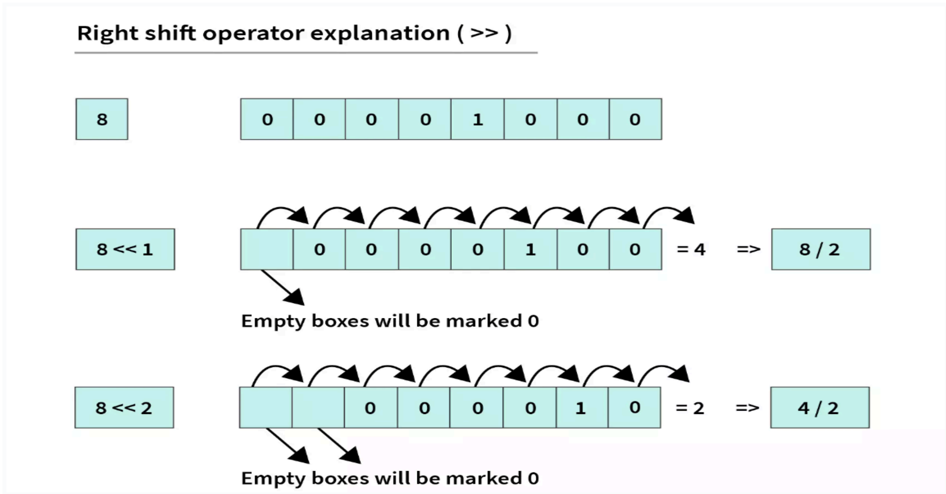
Left Shift operator

A left shift operator shifts each digit in a number's binary representation left by as many spaces as specified by the second operand.



Right Shift Operator

A right shift operator shifts each digit in a number's binary representation right by as many spaces as specified by the second operand.



6. **Properties of Bitwise Operators**
   a. **AND**
      i. A & 1 = 0 or 1 (depending on the last bit of A)
      ii. A & 0 =0
      iii. A & A=A

   b. **OR**
      i. A | 0 = A
      ii. A | 1 = A (if A is odd)
          =A+1(if A is even)
      iii. A | A = A
   c. **XOR**
      i. A ^ 0 = A
      ii. A ^ A = 0

Practice Question - [Single Number]
[Flip bit]
[Number of 1 bits]

Frequently asked Interview Question

1. Given an integer N. Check if the i_{th} is set to 1.

Example: N = 20, i = 2

Let's represent N in the binary number system.
$20 = (10100)_2$

We can see that the bit at 2_{nd} index is set. So, we will return true.

Solution:
We can use bit masking to solve this problem. These are:

1. Right shift the number by i.
2. Now, the bit at 0_{th} is the required answer.
3. We can simply find the 0_{th} bit using AND operation with 1.

$N = (10100)_2$
$N>>2 = (00101)_2$
$(N>>2)\&1 = (1)_2$

```
function(N, K):
    if (N>>K)&1 > 0:
        return true
    else:
        return false
```

- Time complexity: O(1)
- Space complexity: O(1)

2..Single Number 2

Given an array of integers, every element appears thrice except for one, which occurs once.

Find that element that does not appear thrice.

Example:

A = [1, 12, 1, 12, 4, 3, 12, 1, 3, 3]
Here, only 4 occurs once, rest all elements occur thrice. So, our answer should be 4.

Solution:

Observation:
We can work on the count of set bits on each indices, to find if the bit is set in the unique element.
If count = 3 * x, then the bit is unset
If count = 3 * x + 1, then the bit is set
count = 3 * x + 2 is not possible.

Pseudocode:

```
function(A):
    ans = 0
    for bit from 0 to 31:
        count = 0
        for element in A:
            if ((element>>bit)&1) > 0:
                count = count + 1
        if count%3 == 1:
            ans = ans|(1<<bit)
```

3. **Given an integer array A of size N, in which every element occurs exactly twice except 2 elements. Find those two elements.**

Solution:

Brute force: For every element check if it is unique or not.

Pseudo Code:

```
function(A):
    N = length of A
    for i from 0 to N - 1:
        flag = 1
        for j from i + 1 to N - 1:
            if A[i] is equal to A[j]:
                A[i] is not unique
                set flag = 0
                break loop

        if flag == 1:
            A[i] is an answer
```

Optimization:
Suppose X and Y are the unique elements. We can think about using properties of XOR. One such property is A ^ A = 0. So, if we xor all the values of the array A, we will have the xor of the element which occurs once. Let it be Z.

Now, let's try to separate the array into two arrays such that X and Y occur in different arrays. For this we will find a set bit in Z and separate all the elements of A based on this set bit. Now, these two subarrays will have a single unique element. We can easily solve these sub problems using xor operation.

Pseudo Code:

```
function(A):
    ini_xor = 0
    for element in A:
        ini_xor = ini_xor ^ element

    set_bit = -1

    find set bit in ini_xor
    for i from 0 to MAX_BIT:
        if (ini_xor>>i)&1 > 0:
            ith bit is set
            set_bit = i
            break from loop

    xor_1 = 0, xor_2 = 0

    for element in A:
        if set_bit is set in element:
            xor_1 = xor_1 ^ element
        else
            xor_2 = xor_2 ^ element

    return (xor_1, xor_2)
```

- Time complexity: O(N)
- Space complexity: O(N)

Video document -
https://drive.google.com/file/d/1zEJN4CuoTR7K6b0ecEiKzEP1Rzp3jHoR/view?usp=share_link