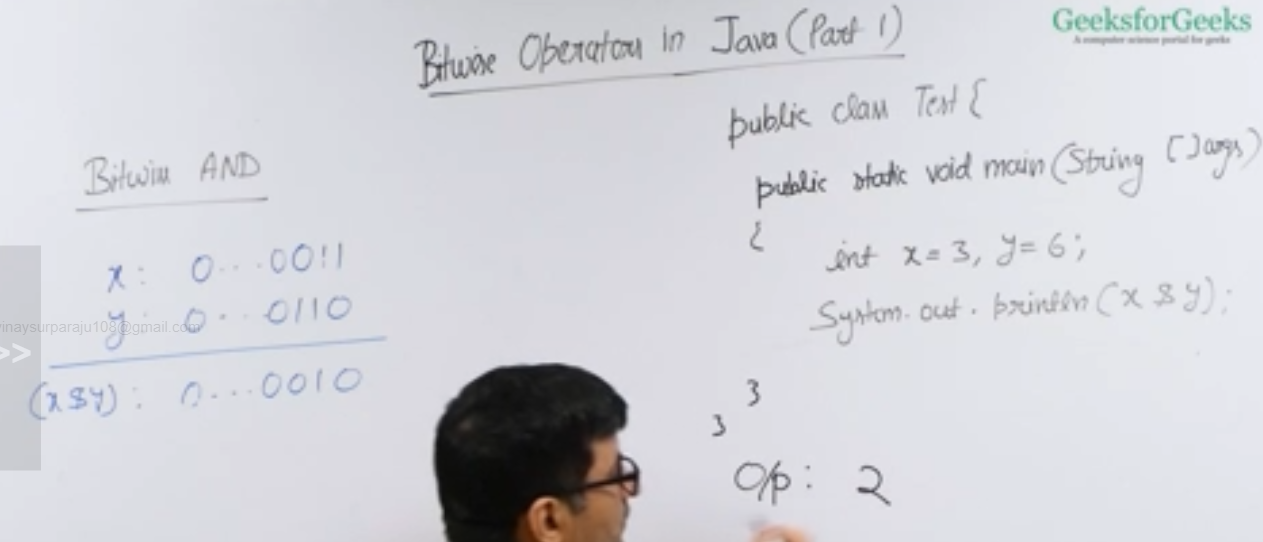
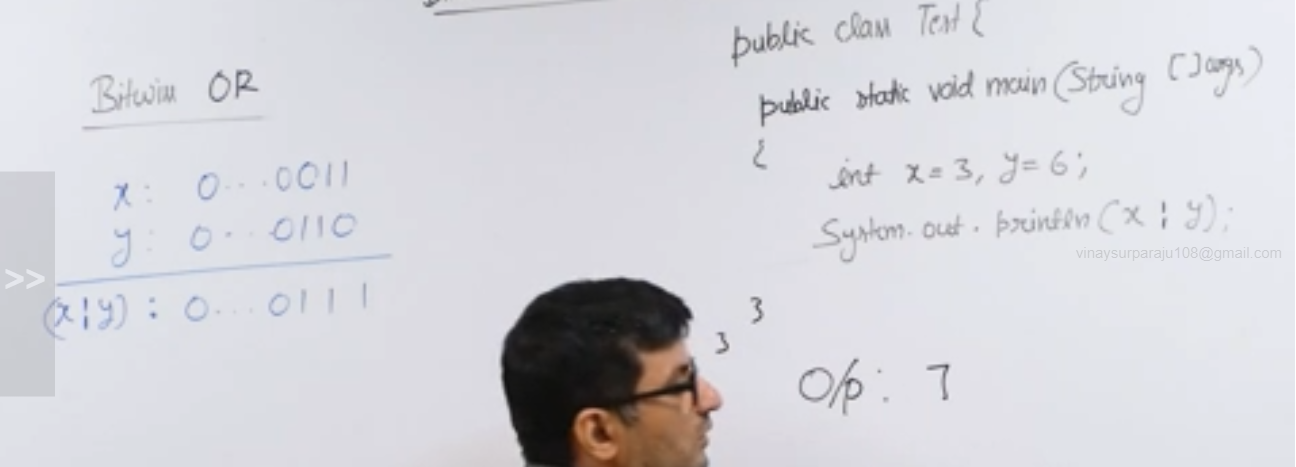
Bitwise Operators in Java:

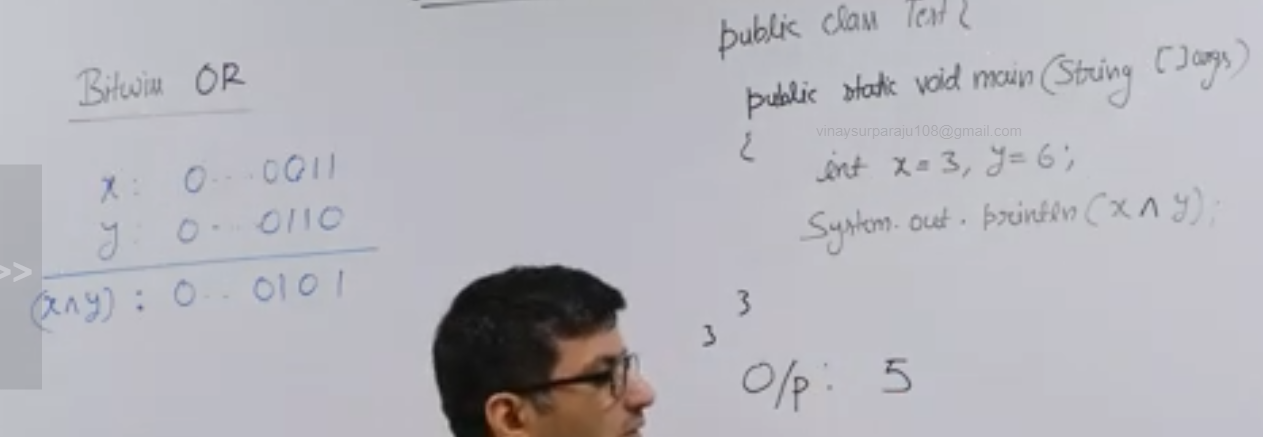
1. Bitwise AND (&) - if two inputs are 1 only then output is 1. It considers binary representation of numbers when bitwise operator is used and returns the output in integer. Eg: 3 & 6



1. Bitwise OR (|)- if either of the input is 1, both the input are 1 then output is 1.



1. Bitwise XOR (^) – if only either of the input is 1, then the output is 1.



1. Bitwise NOT or Complement(~)- takes the input number and prints the output in complement form.

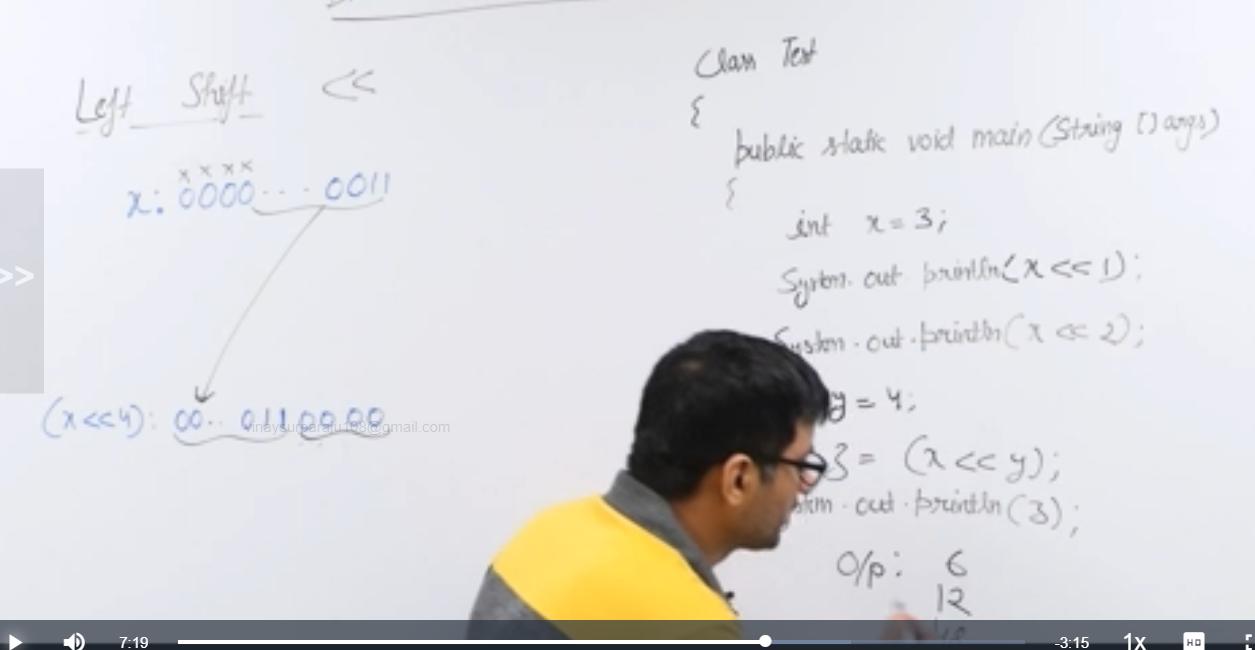
Negative numbers are stored in 2s complement form.If the left most bit is 0 then it’s a positive number and if its 1 then it’s a negative number.

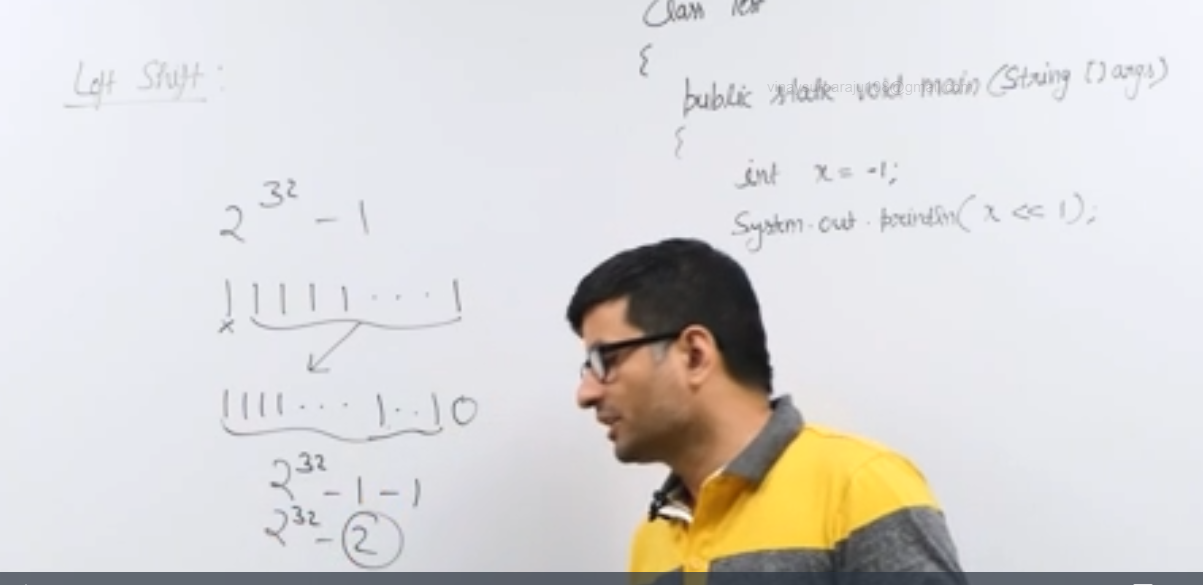
Representation of negative number x = 2pow(32)-x



1. Left shift operator (<<): it is a binary operator, it takes a representation of a number whose binary number is to be shifted and takes another number which represents how many times the number need to be shifted.

The left most bits will be ignored and new zeros will be added to the right as per the shift number





In general, for small input number either for negative or positive numbers, left shift operator works as multiplication with 2.

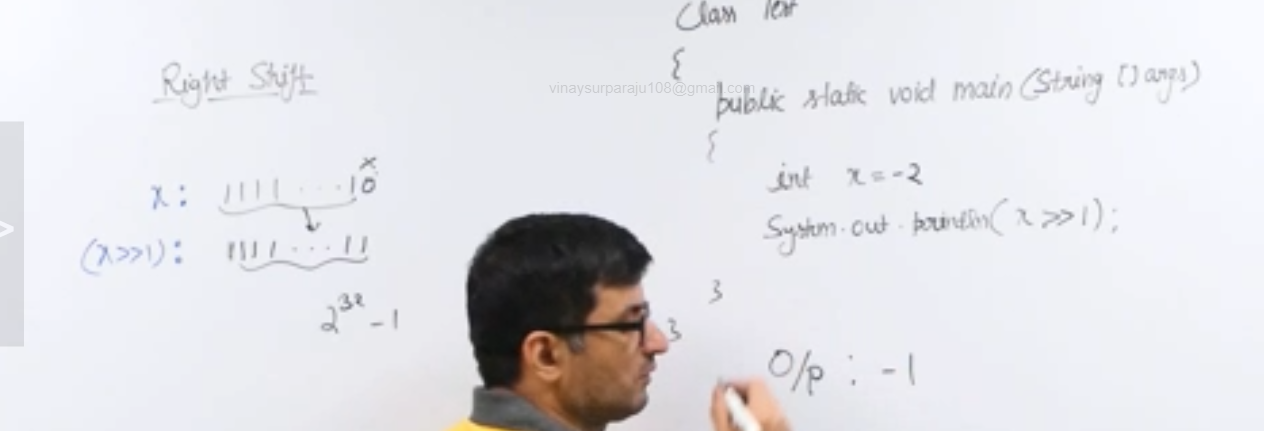
1. Bitwise Right shift operator( >>): It is the opposite of left shift operator.It shifts the bits to right by the value.

The right most bit is ignored and zero will be added to the left most bit

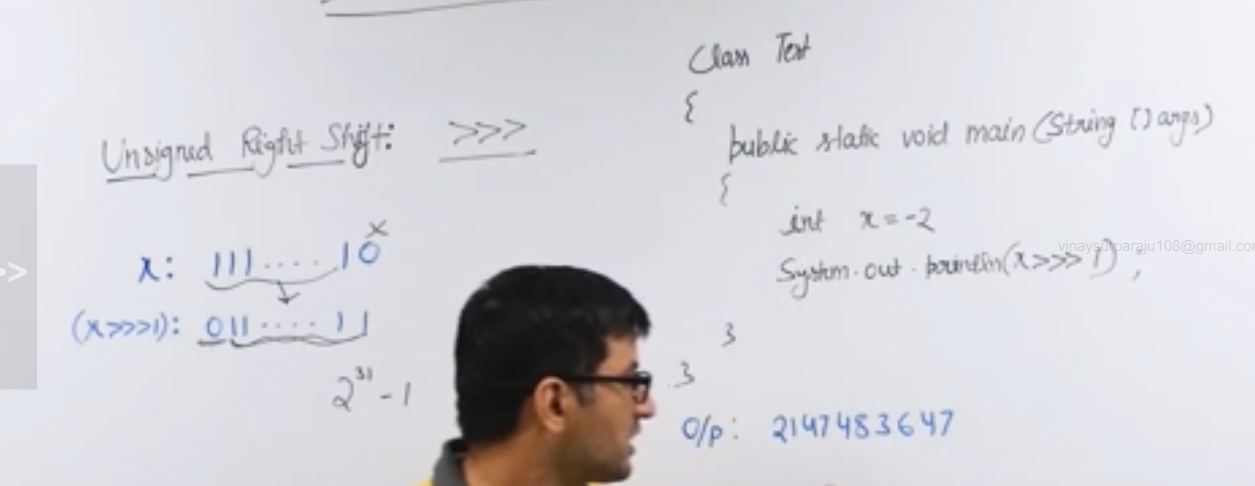


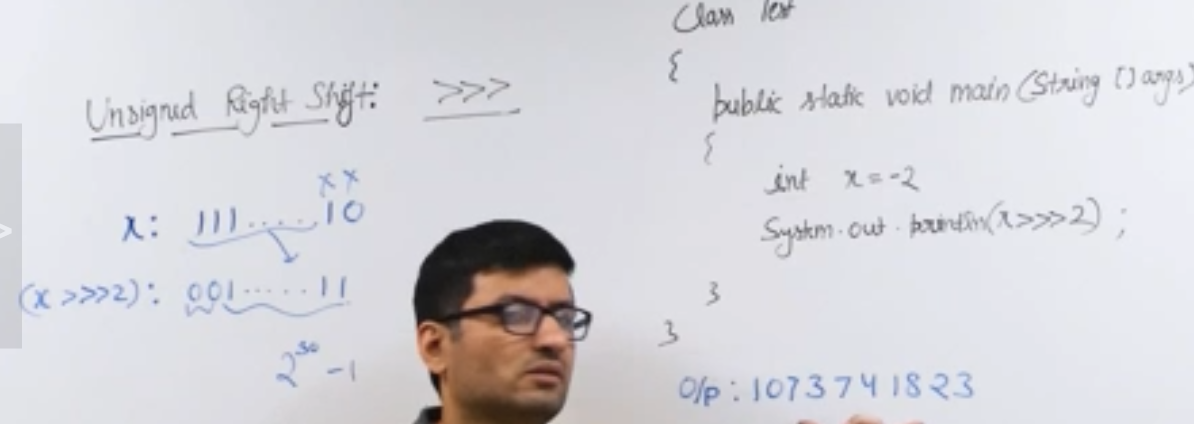
For negative input number,we have signed right shift because it treats negative and positive differently.In case of negative, it fills leading bits with 1 to maintain negative sign

For input greater than -2, right shift operator will return -1 in decimal as output.

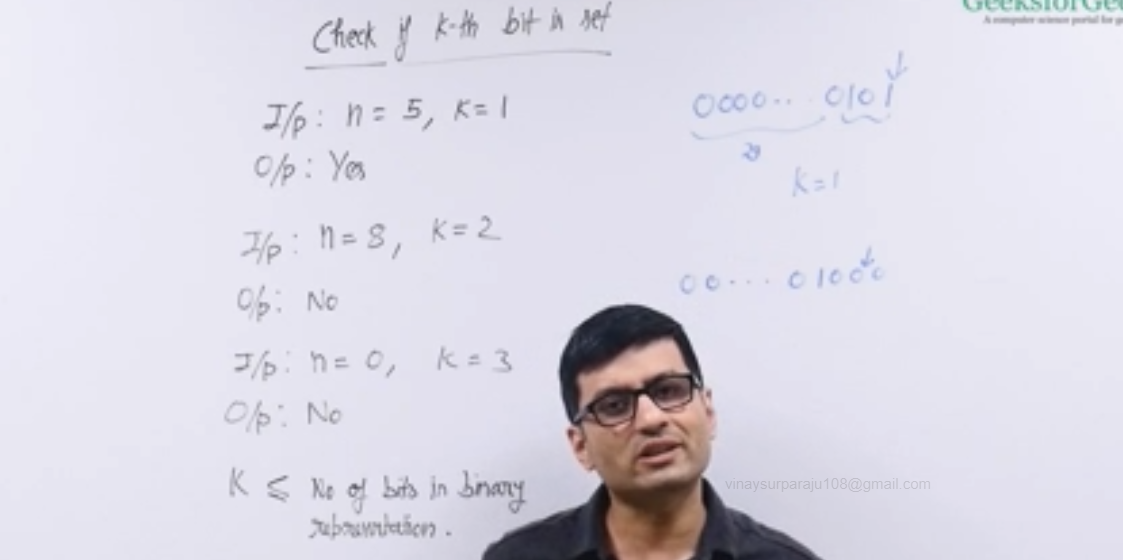


1. Unsigned right shift operator(>>>):Here the leading bits are filled with zero.This difference is only for negative numbers.

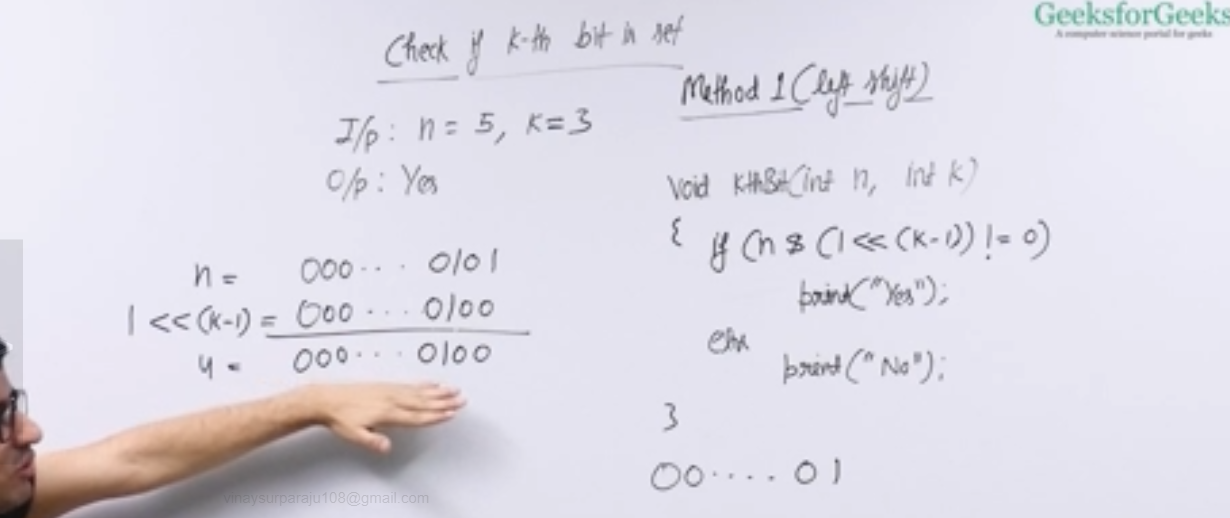




Check if Kth bit is set or not:

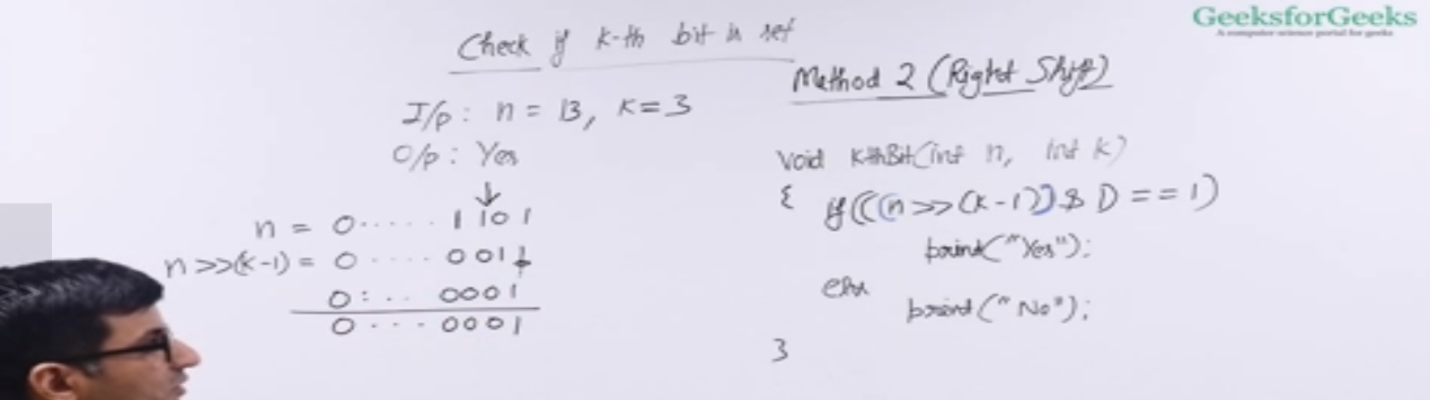


Sol 1:



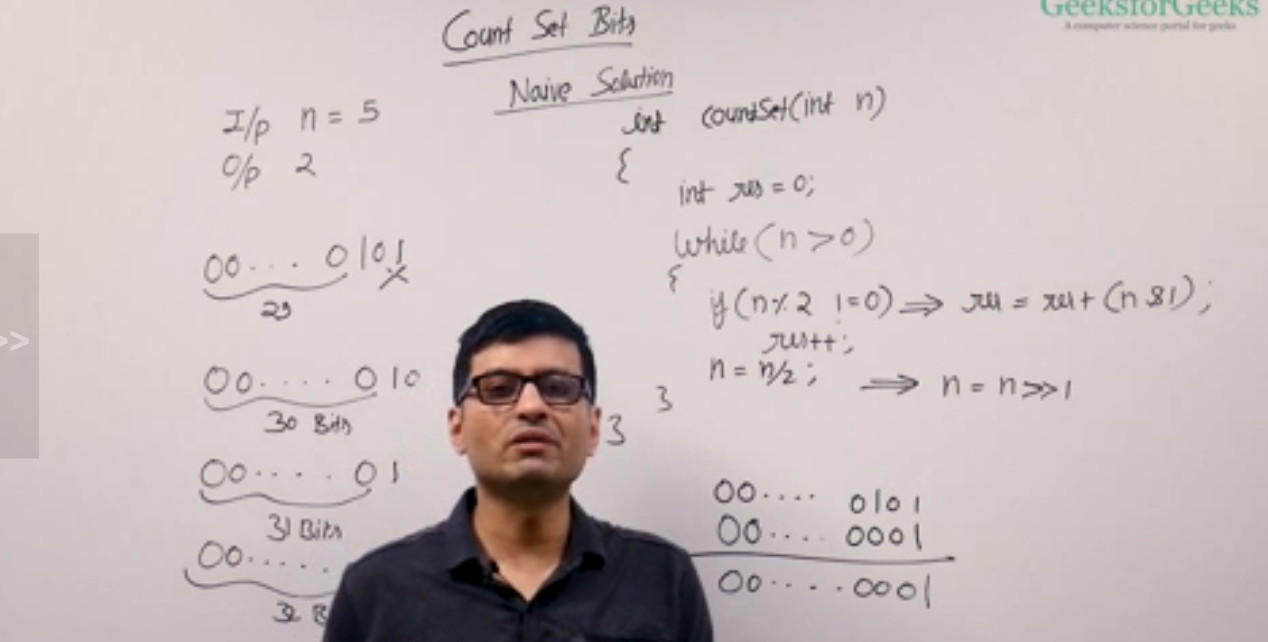
We will left shift 1 with K-1 and then perform bitwise AND with input N. If both have 1 then it is set.

Sol 2:



Count set bits in integer:

Naïve sol: time complexity of Theta(Total bit count n)



Brian Kerringam Algorithm: time complexity of Theta(set bit count)

When we substract 1 from a number, all the bits which are 0 will become 1 after last bit set(left most bit set) and left most bit will become 0.



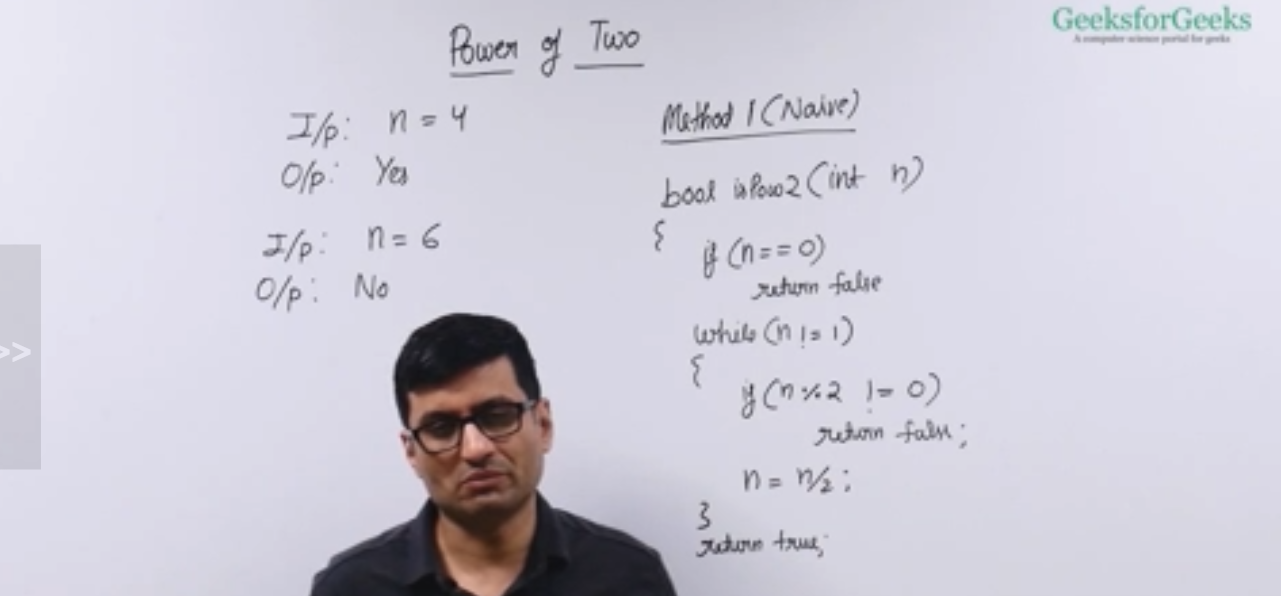
Lookup Table Method:



Power of two:

Naïve sol:

Assuming n is non-negative integer. We divide n by 2 and check if the number is odd, then we will return false, else we divide further until n is 1

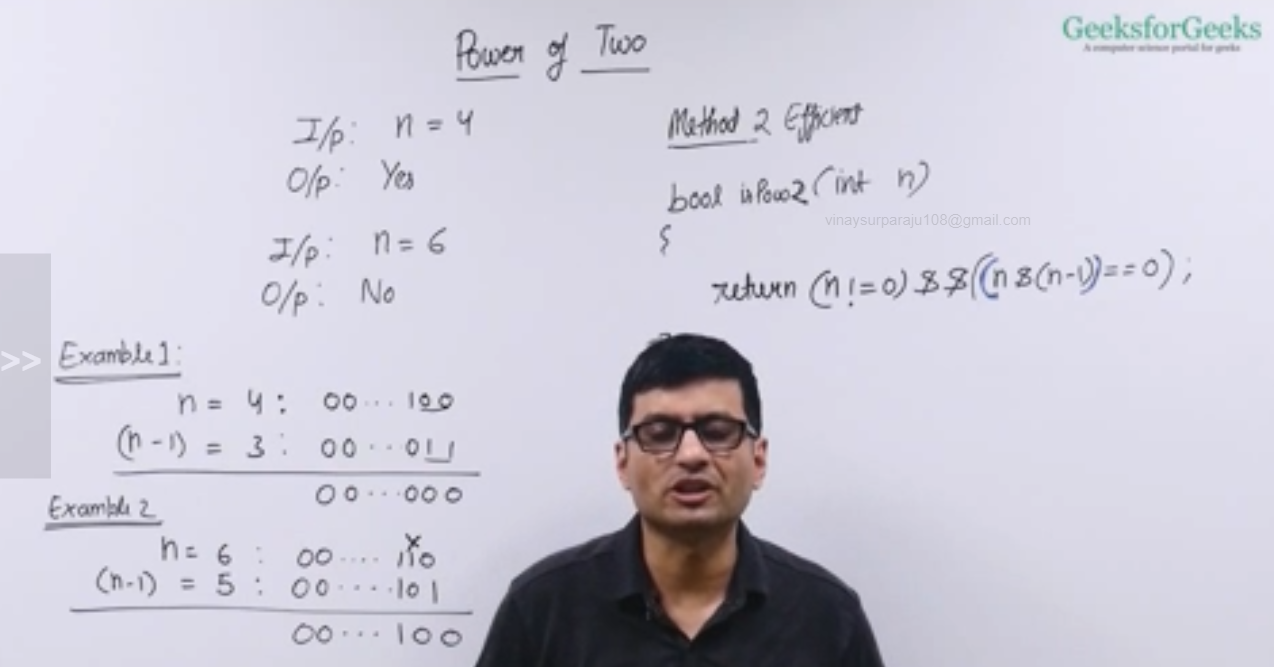


Brian Kerringom method:

For input number N, there will be 1 set bit available, so if there is 1 set bit available then input is power of 2 else it will have more than 1 set bit

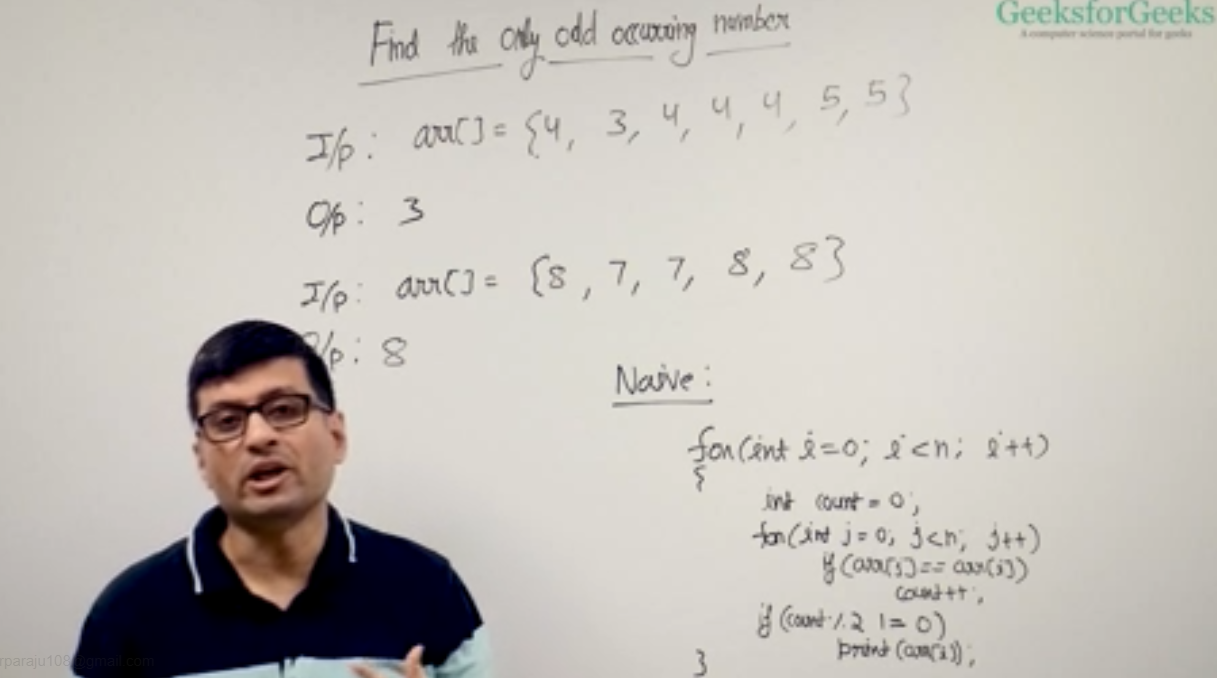
We unset the only set bit and if the result is zero then the number is power of 2 else if the result has more set bits then it’s not power of 2

We can always unset the right most bit of the number by performing bitwise AND of the number n with n-1.

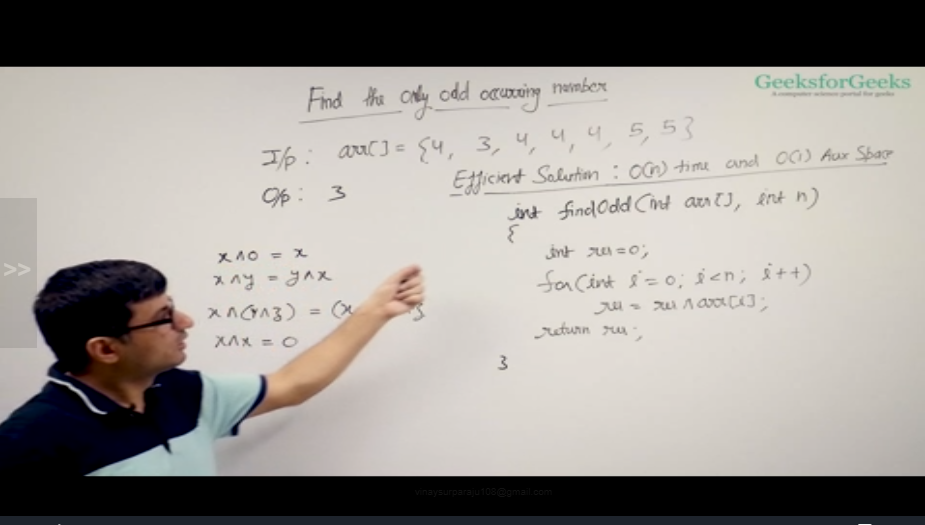


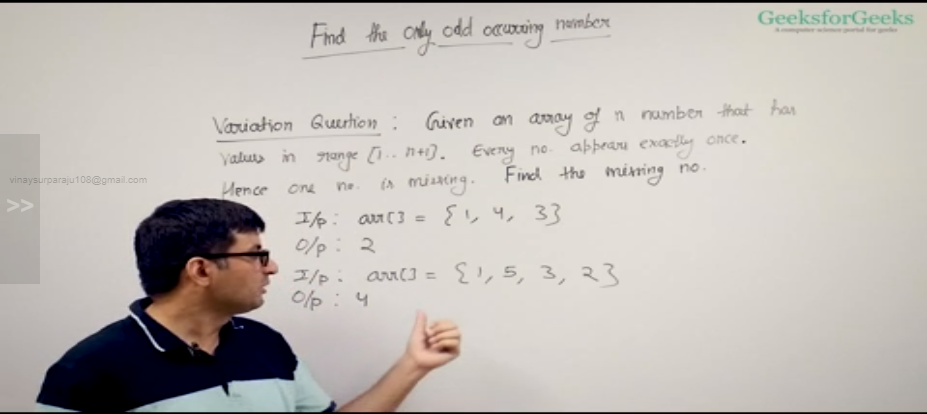
One Odd occurring:

Naïve sol:



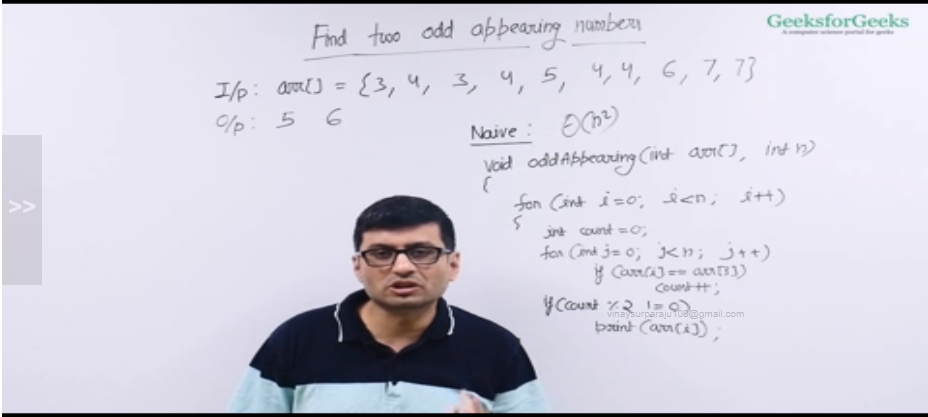
Efficient approach:





Two Odd Occurring number:

Naive sol:



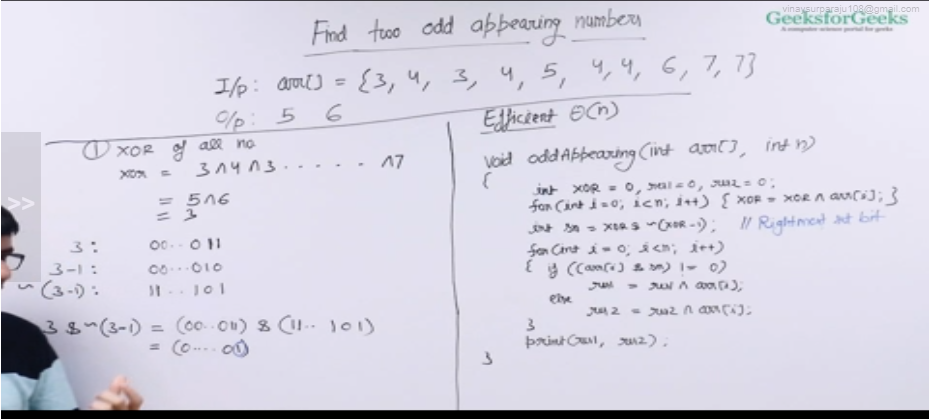
1. Efficient approach: O(n)

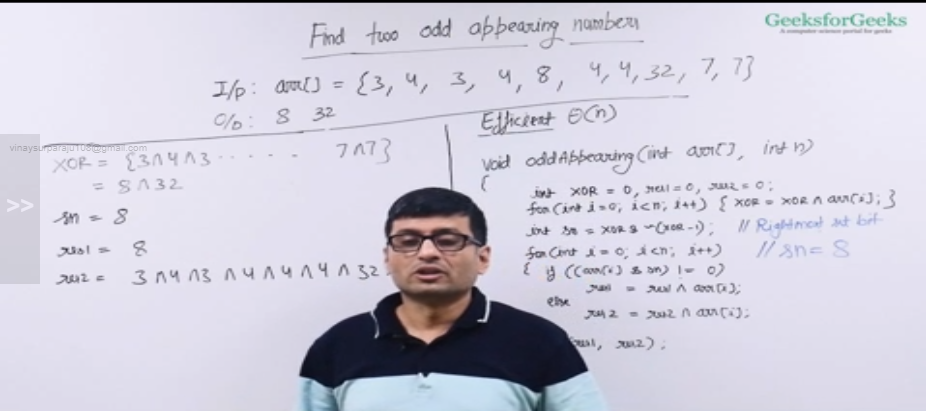
Step1: To perform XOR of all the numbers - even appearing will cancel out each other and odd appearing number remains and its XOR provides a value

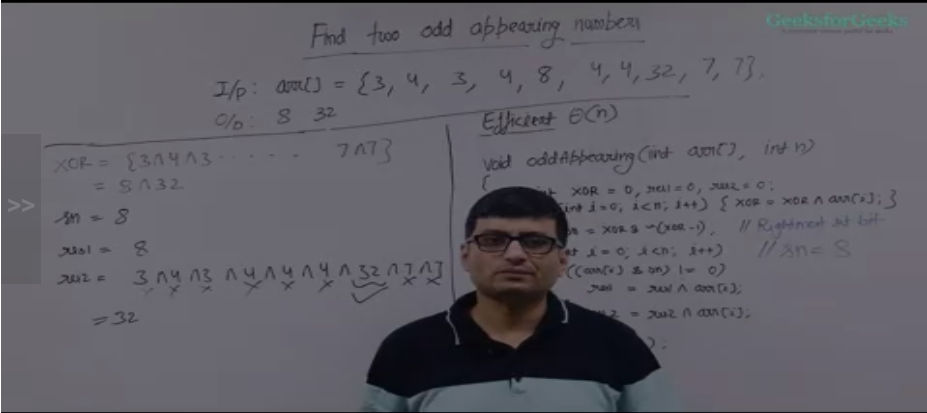
Step 2: Consider the set bit of the value and group the input into 2 groups

Step 3: Group1 will have input values having set bit at the position and group 2 will have input values having no set bit at the position

Step 4: Now perform XOR of all the inputs in each group , we will get the result

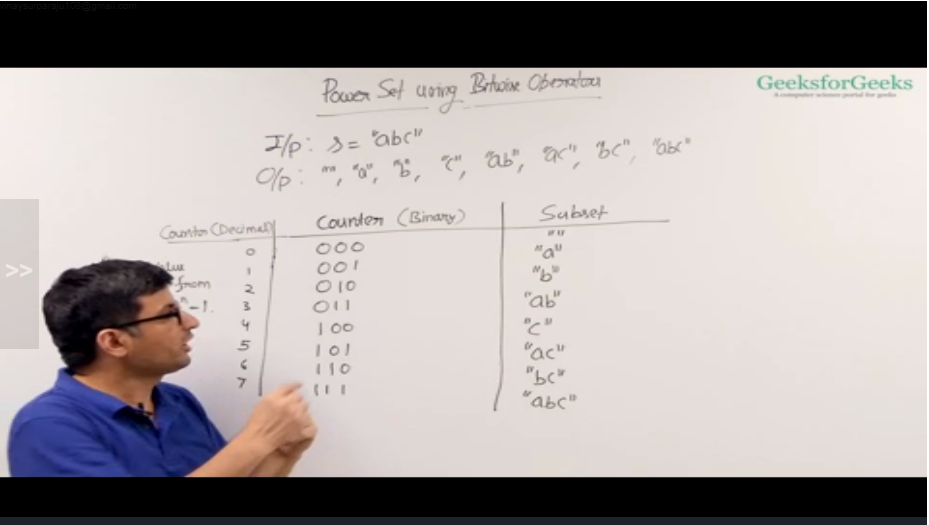




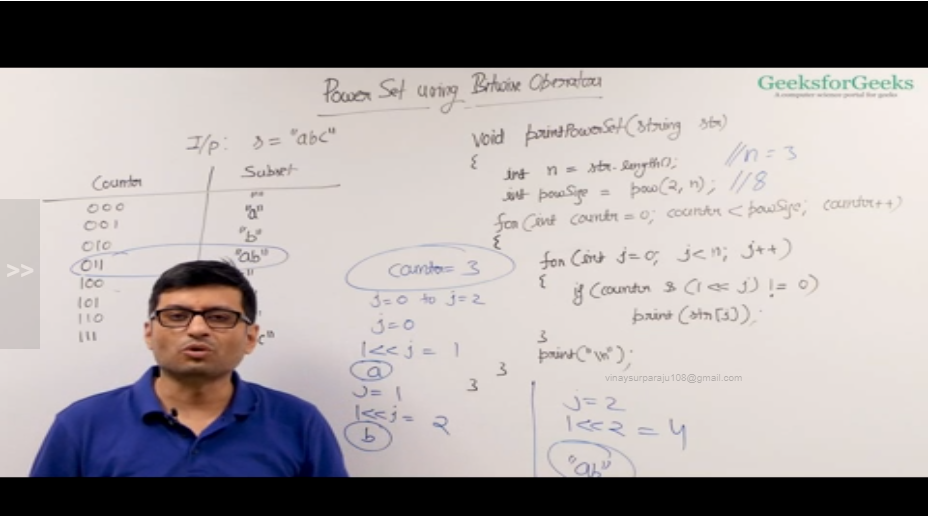


Power set using bitwise operator:

For input have n char, there will be 2pown subsets.



Each set bit corresponds to a character in the string and subsets are displayed



Time complexity of the program is 2pown\*n