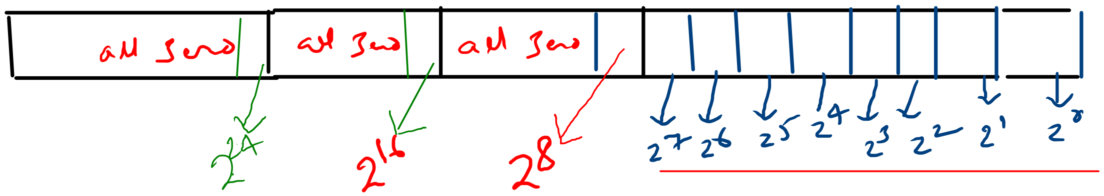
"Bitwise Utilities"

Understanding number of leading & trailing zeros in an integer.

In Java integer is 32 bits i.e 4 bytes.

e.g. $x = 83$ → lies between $2^{6} \left(\lfloor \log_2 83 \rfloor\right)$ & $2^{7} \left(\lceil \log_2 83 \rceil\right)$



as we know

$$\sum_{i=0}^{n} 2^{n} = 2^{n+1} - 1$$

So, 2^8 bit position is always greater than the number formed by using bit positions 0 to 7.

If $x < 2^{8}$ → x lies in first byte

If $x >= 2^{24}$ → x lies in 4th byte

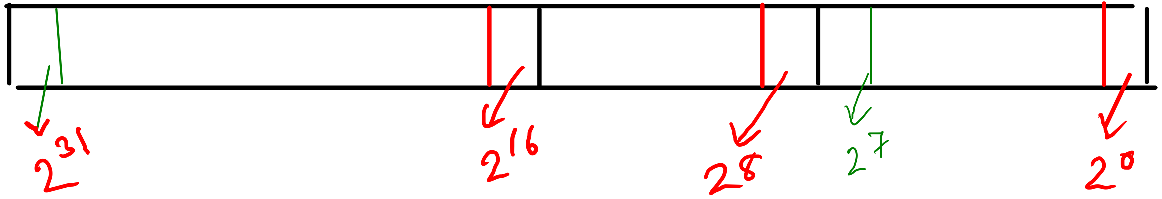
If $x >= 2^{16}$ → x doesn't lie in first 16 bits.

If $x >= 2^{8}$ → x doesn't lie in 1st byte (0 to 7)

If $x >= 2^{4}$ → x doesn't lie in 1st 4 bits (0 to 3)

If $x >= 2^{2}$ → x doesn't lie in 1st 2 bits (0 to 1)

If $x >= 2^{1}$ → x doesn't lie on 0th bit



[$x >= 2^{16}$ → means x doesn't lie in first 2 bytes]

We will start our inspection at $x >= 2^{16}$ because once we get to know x doesn't lie in first 2 bytes (means can have at max $31-16=15$ leading zeros) then we need to inspect 3rd & 4th byte, this can be easily done by shifting 3rd & 4th byte to 1st & 2nd byte position.

So, checks will be

(I) $x >= 2^{16}$ or $x >= (1 << 16)$

(II) $x >= 2^{8}$ or $x >= (1 << 8)$

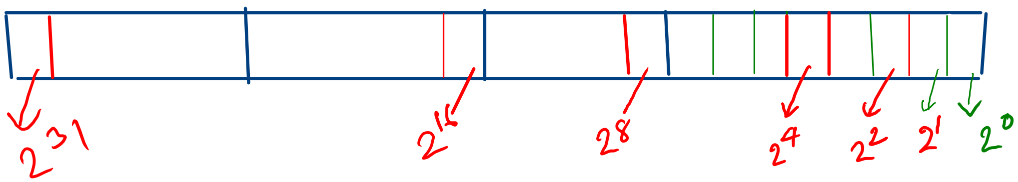
(III) $x >= 2^{4}$ or $x >= (1 << 4)$

(IV) $x >= 2^{2}$ or $x >= (1 << 2)$

(V) $x >= 2^{1}$ or $x >= (1 << 1)$

x = given number, Max_index = 31

initialize LeadingZeros = 31



Algo: If $x >= 2^{16}$: → means max leading zeros may be

$31 - 16 = 15$

→ and further need to inspect 3^{rd} & 4^{th} byte.

→ zerofill right shift 16. so that 3^{rd} & 4^{th} bytes comes at 2^{nd} & 1^{st} byte position.

→ $x >>> 16$

then

step 2: If $x >= 2^8$: Here we might be inspecting a fresh number or $x>>>16$ shifted bytes of step I that lies on 2^{nd} byte position.

since x is $>= 2^8$, so x cannot be at 1^{st} byte

so, max leading_zeros = current_leading_zeros − 8

→ now, we shift the $x >>> 8$

Similarly we do for $x >= 2^4$, $x >= 2^2$ & $x >= 2^1$.