**[7] Bit Manipulations –**

1)Bitwise Operators -

2) Check if even or odd -

3) Get ith Bit of a num -

4) Set ith Bit - (sets the value to 1) -

5)Clr ith Bit - (sets the value to 0)

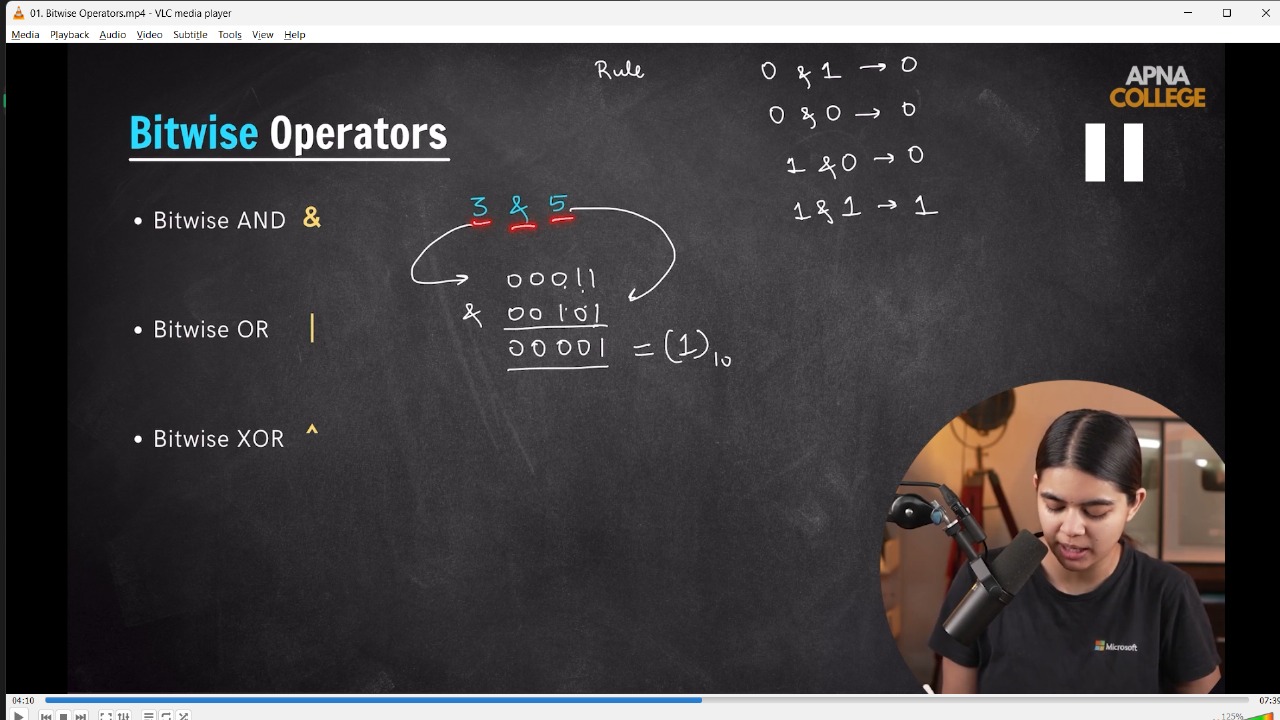
6) No. is power of 2 -

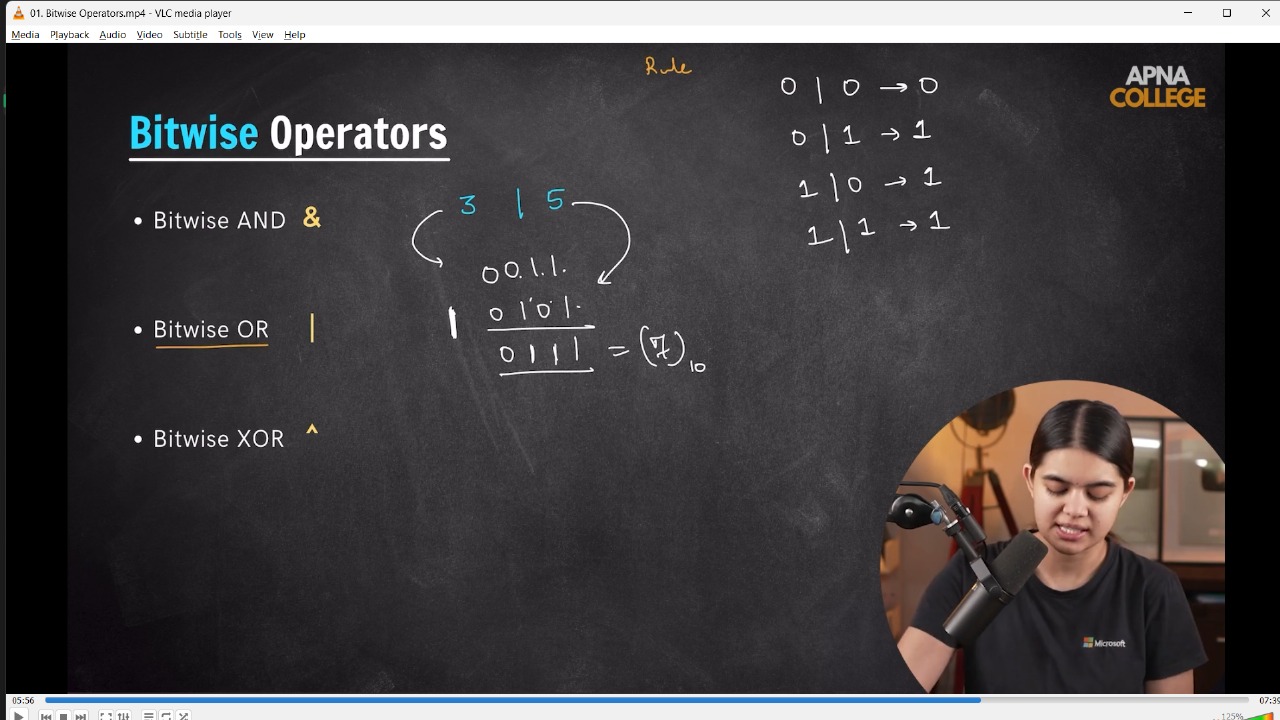
7) Count of Set Bits -

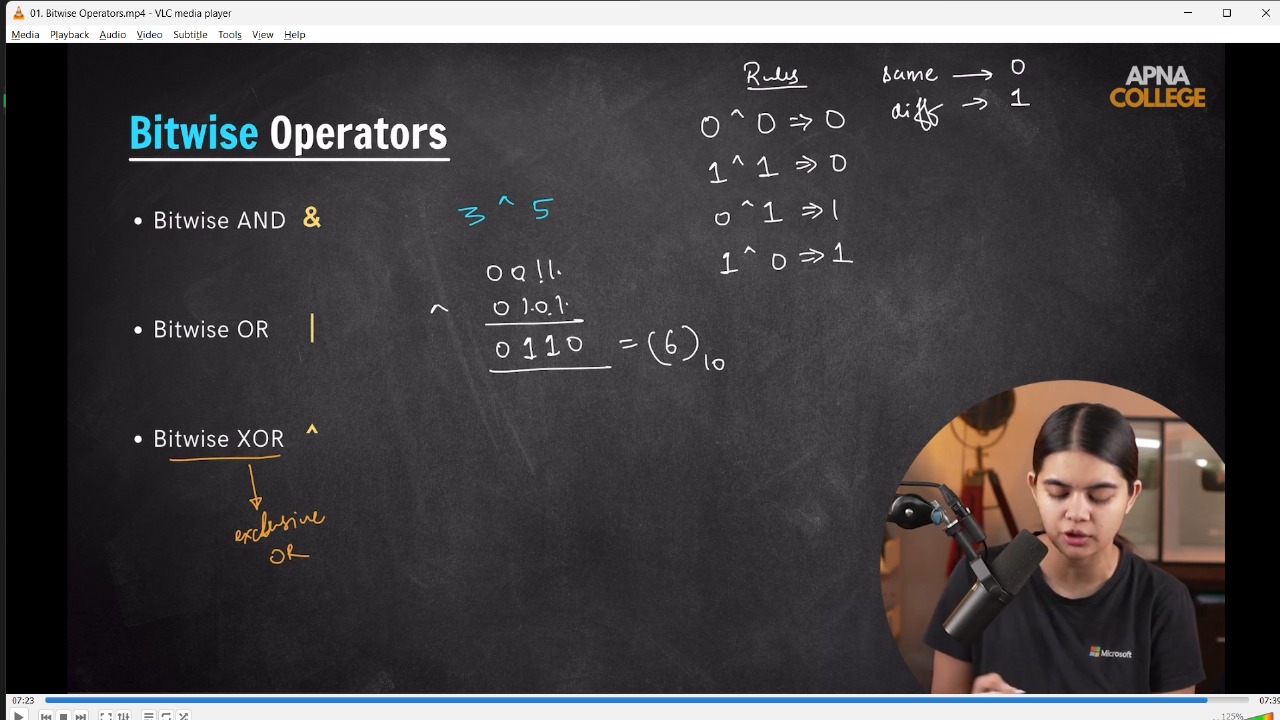
8) Fast Exponentiation -

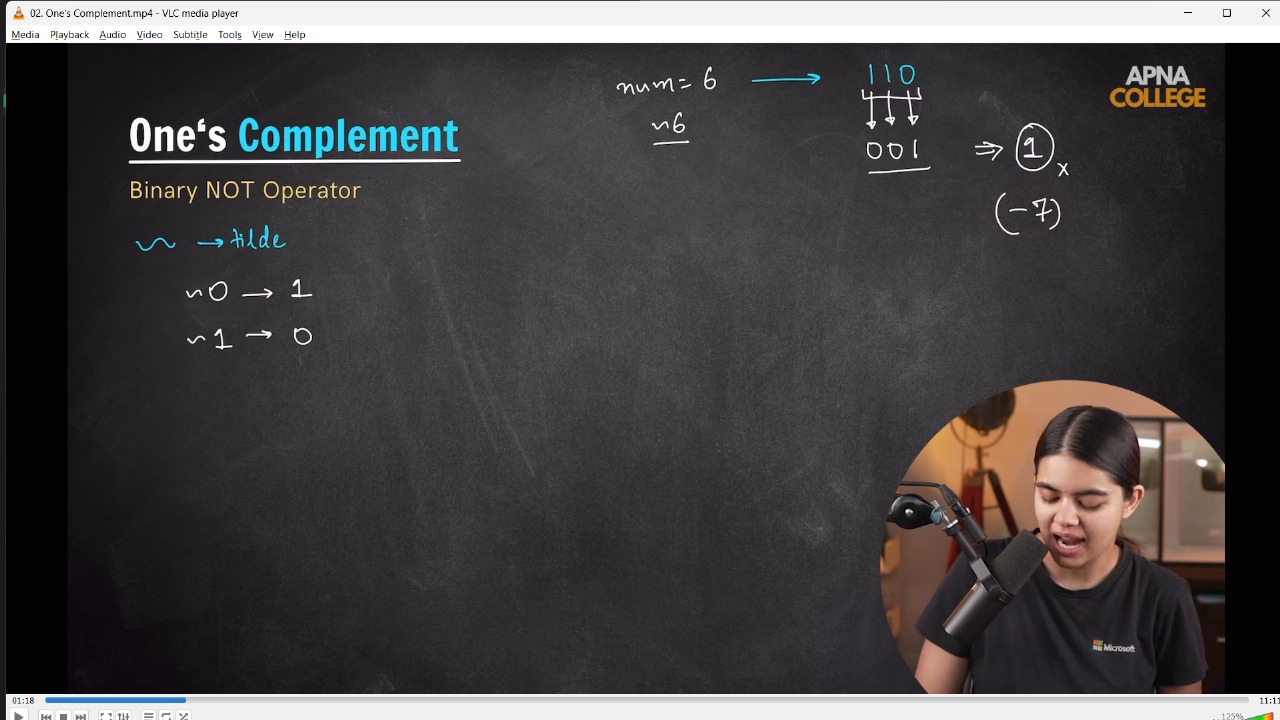
7) Bit Manipulations –

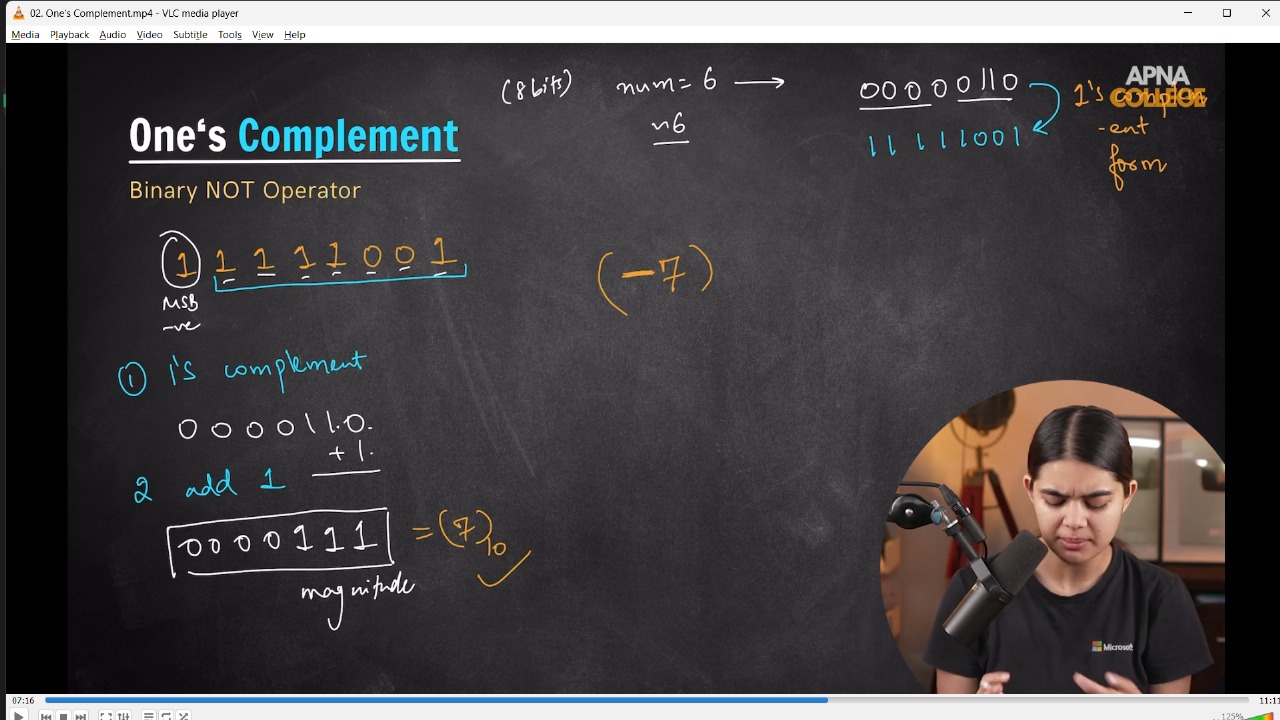
**1)Bitwise Operators  -**

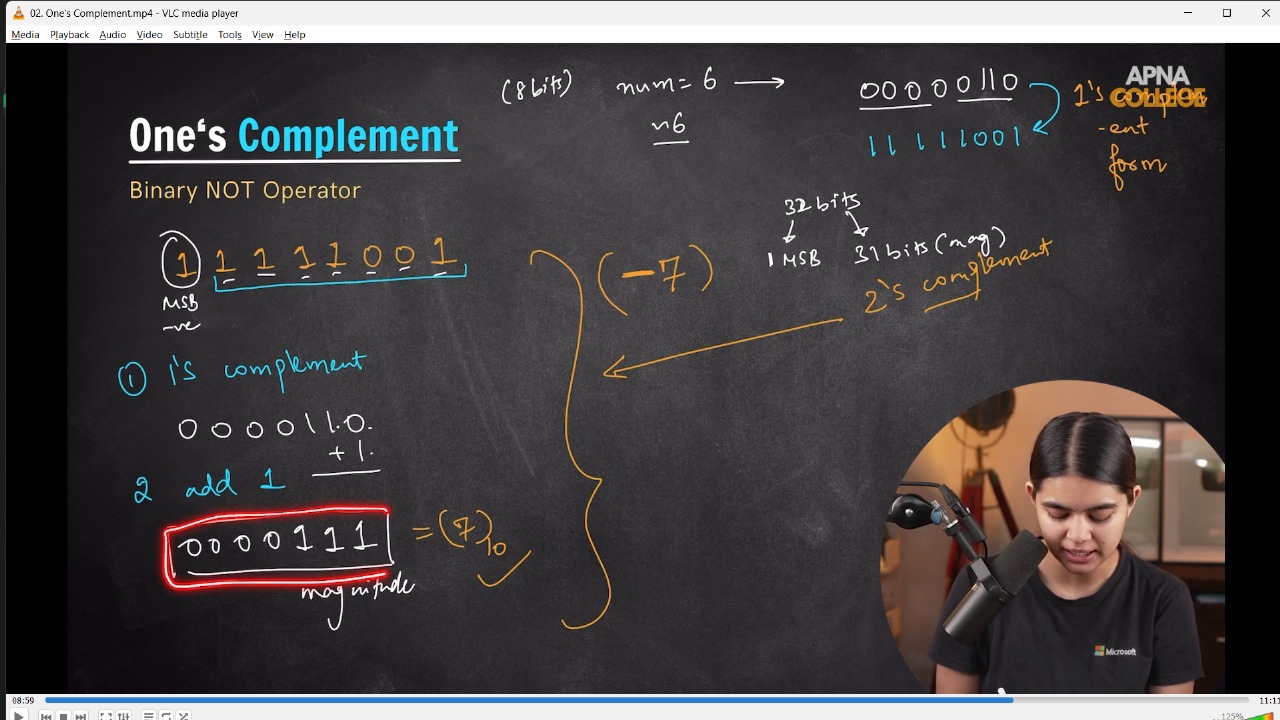


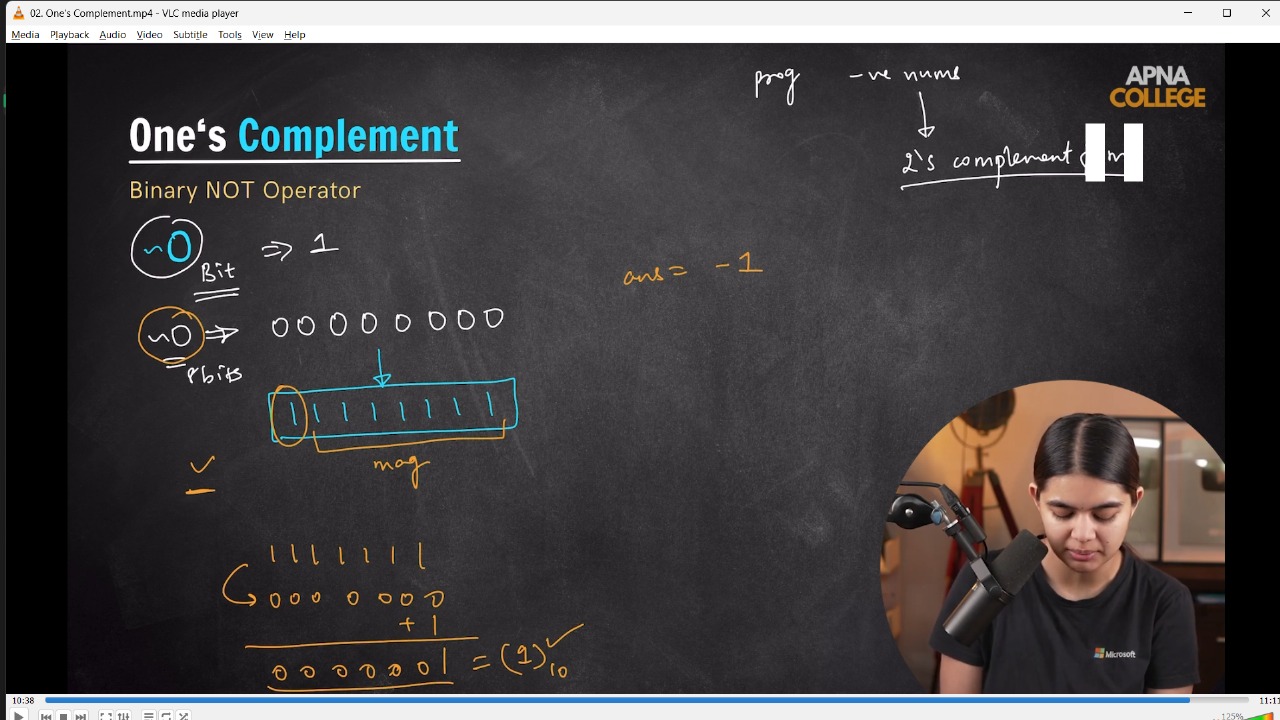


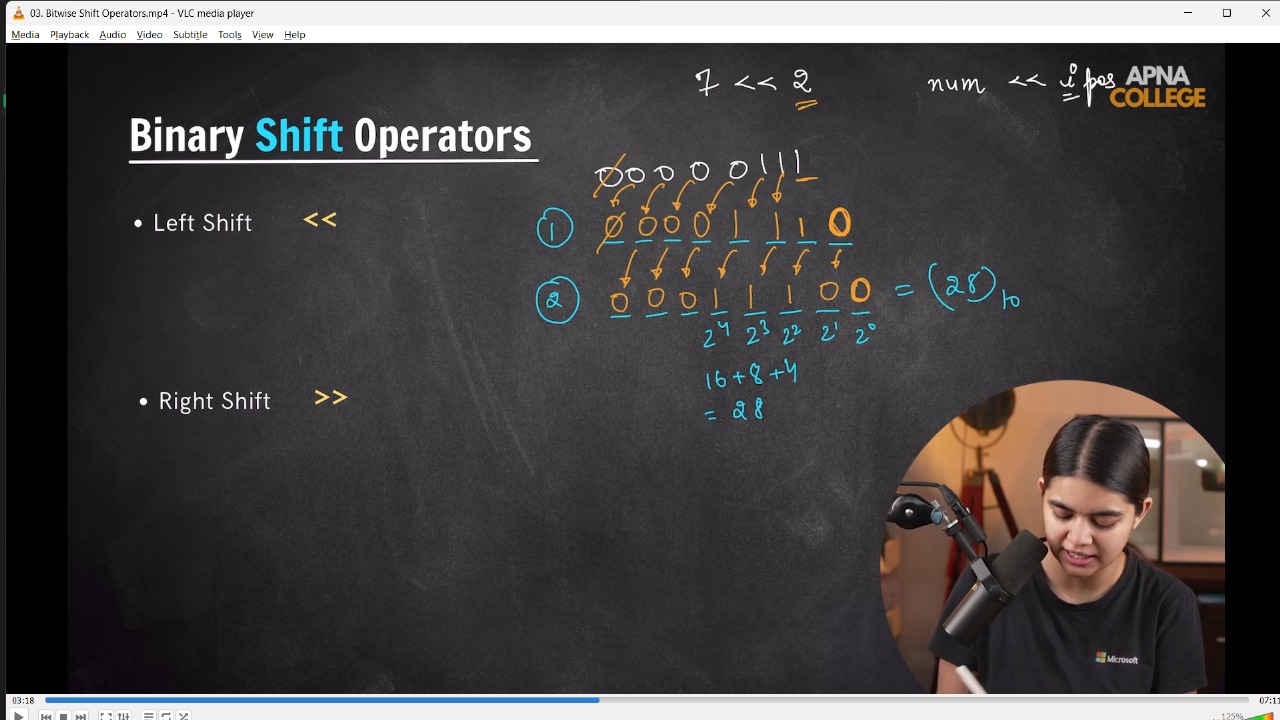


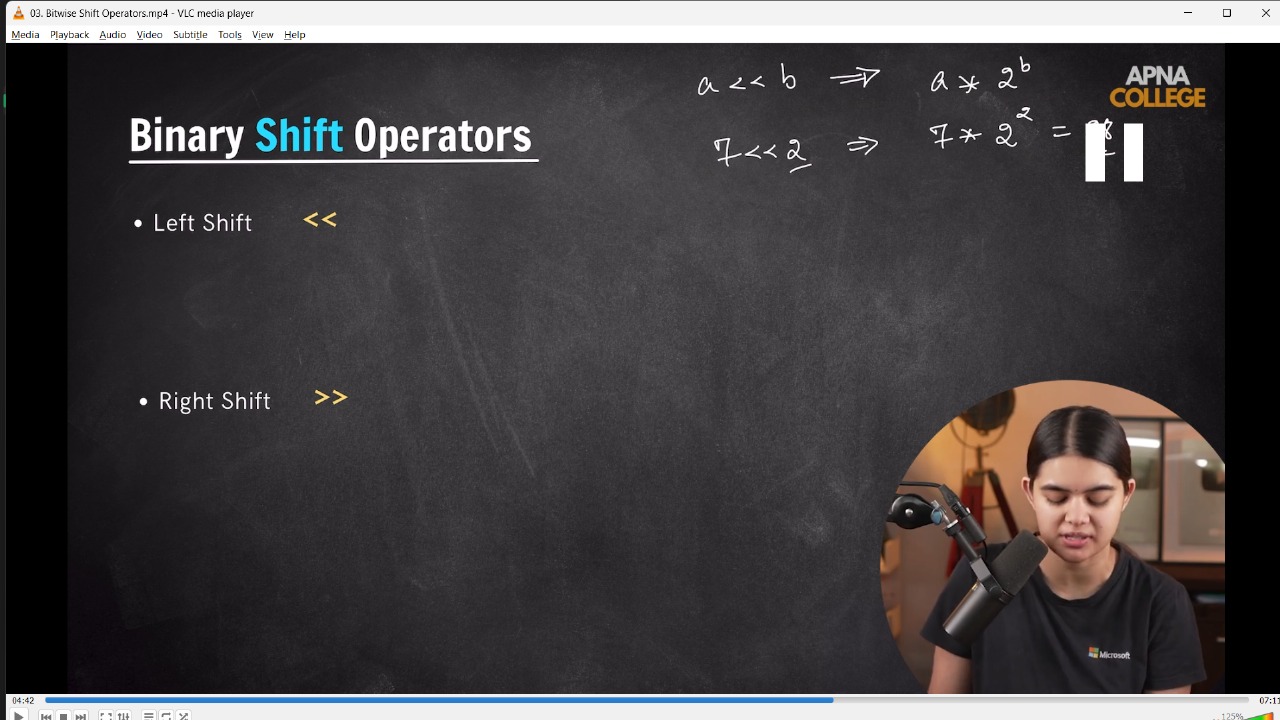


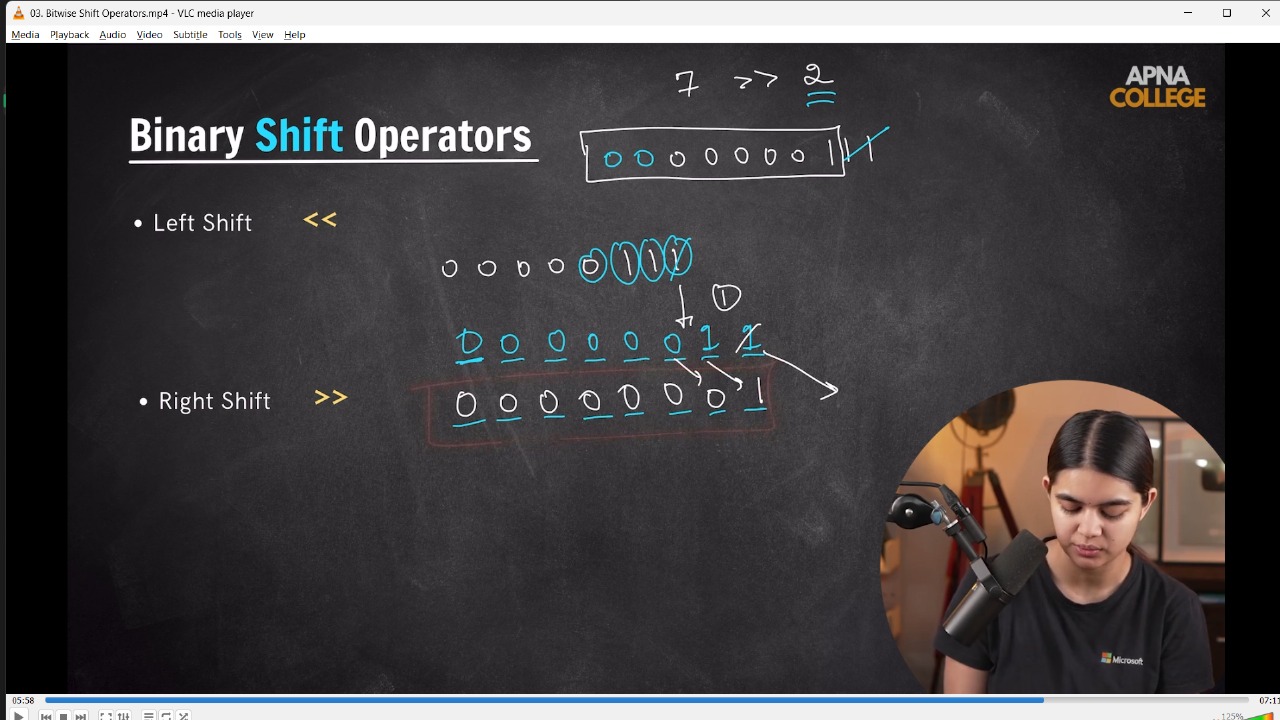


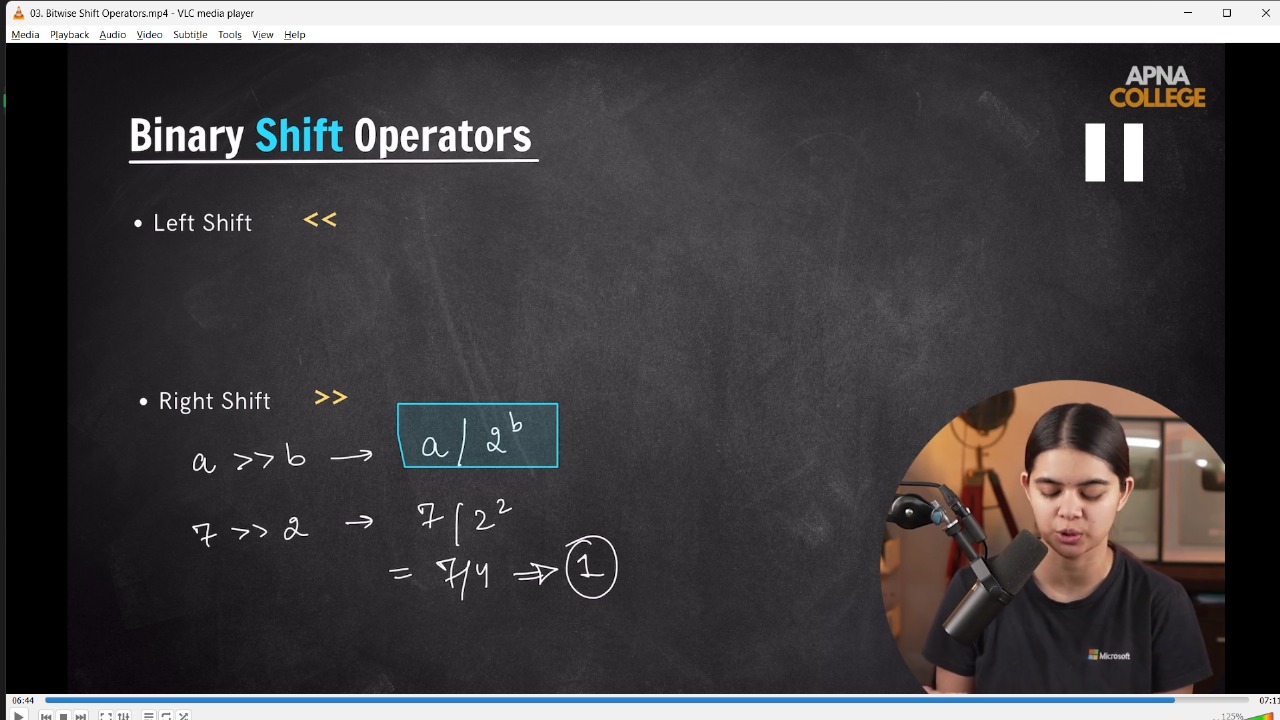


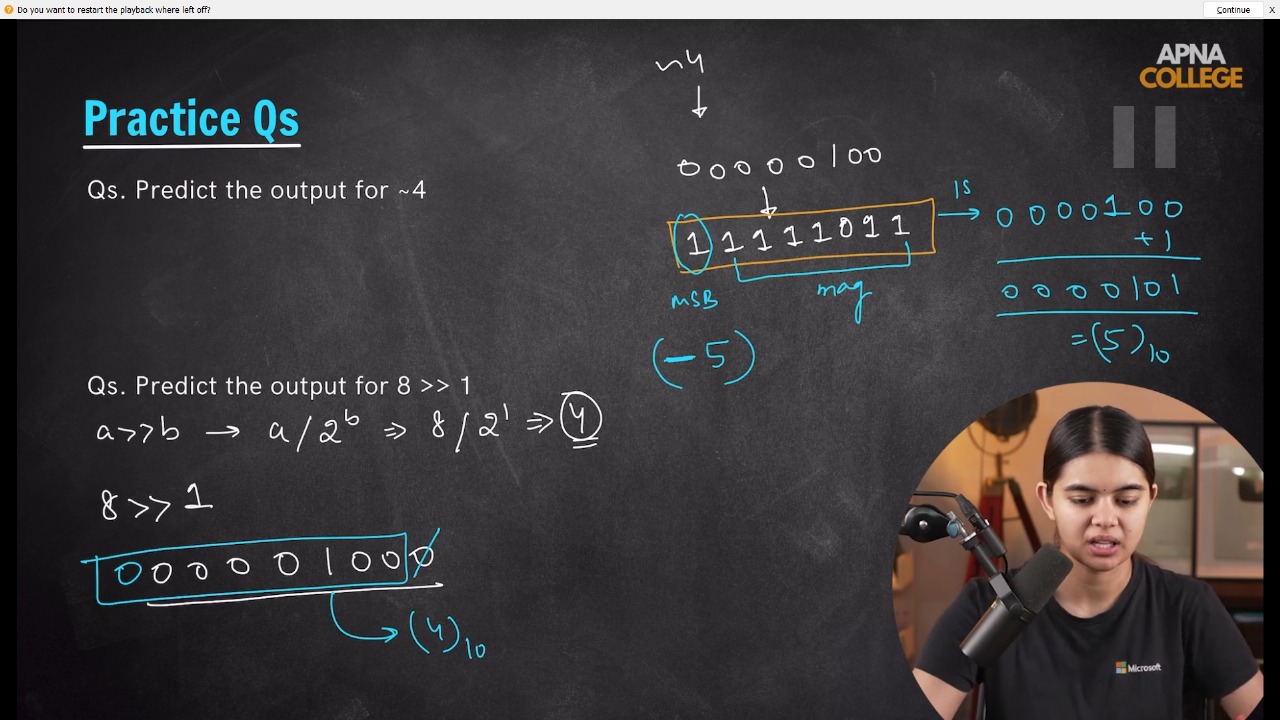
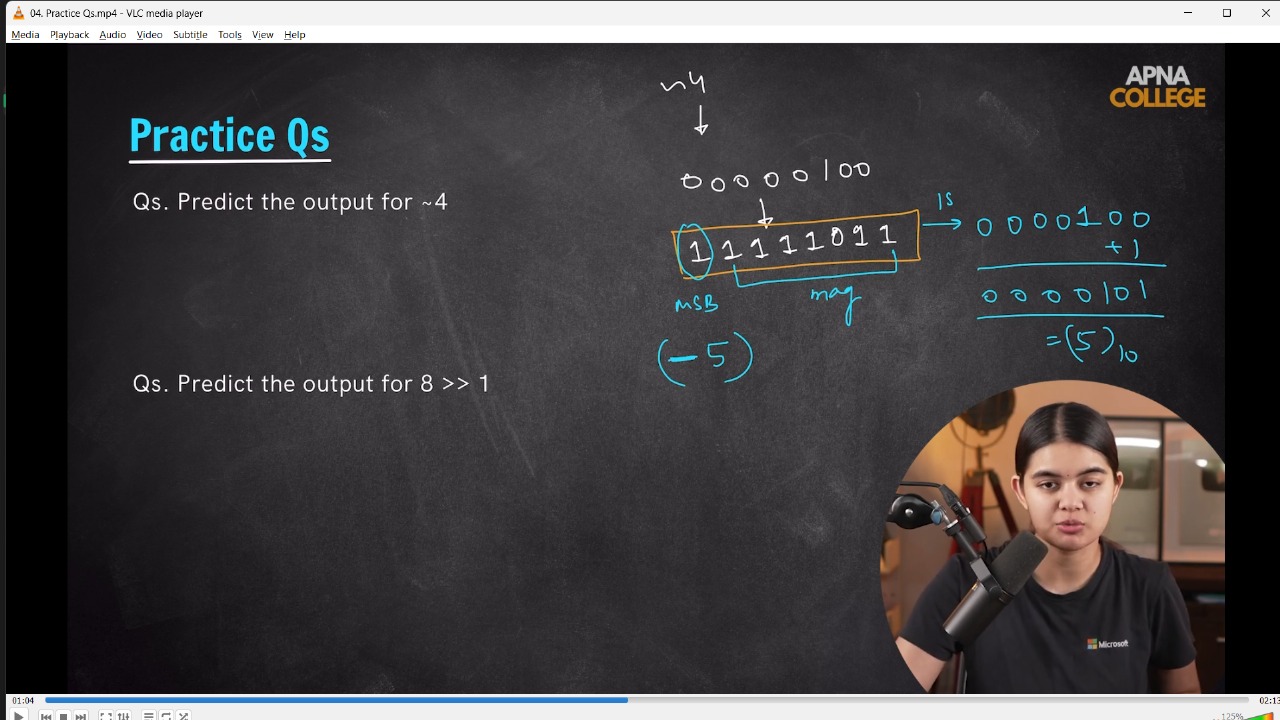












// int main()

// {

//     // Bitwise AND

//     cout << (3 & 5) << endl; //(0001)&(00101) = (00001) = 1

//     // Bitwise OR

//     cout << (3 | 5) << endl; //(0001)|(00101) = (0111) = 7

//     // Botwise XOR

//     cout << (3 ^ 5) << endl; //(0001)^(00101) = (0111) = 6  - Same bit =0, diff. bit = 1

//     // Bitwise NOT(tild operator)

//     cout << (~6) << endl; //-7

//     cout << (~0) << endl; //-1

//     // Binary Shift Operator -

//     // Binary Left SHift Operator -

//     cout << (7 << 2) << endl; // 28 - shorcut is simple. If a<<b then = a\*2^b

//     cout << (7 >> 2) << endl; // 1 - shortcur is simple. If a>>b then = a/2^b

//     // Some Practice Quns -

//     // 1) Output for ~4 -

//     cout << (~4) << endl; //-5

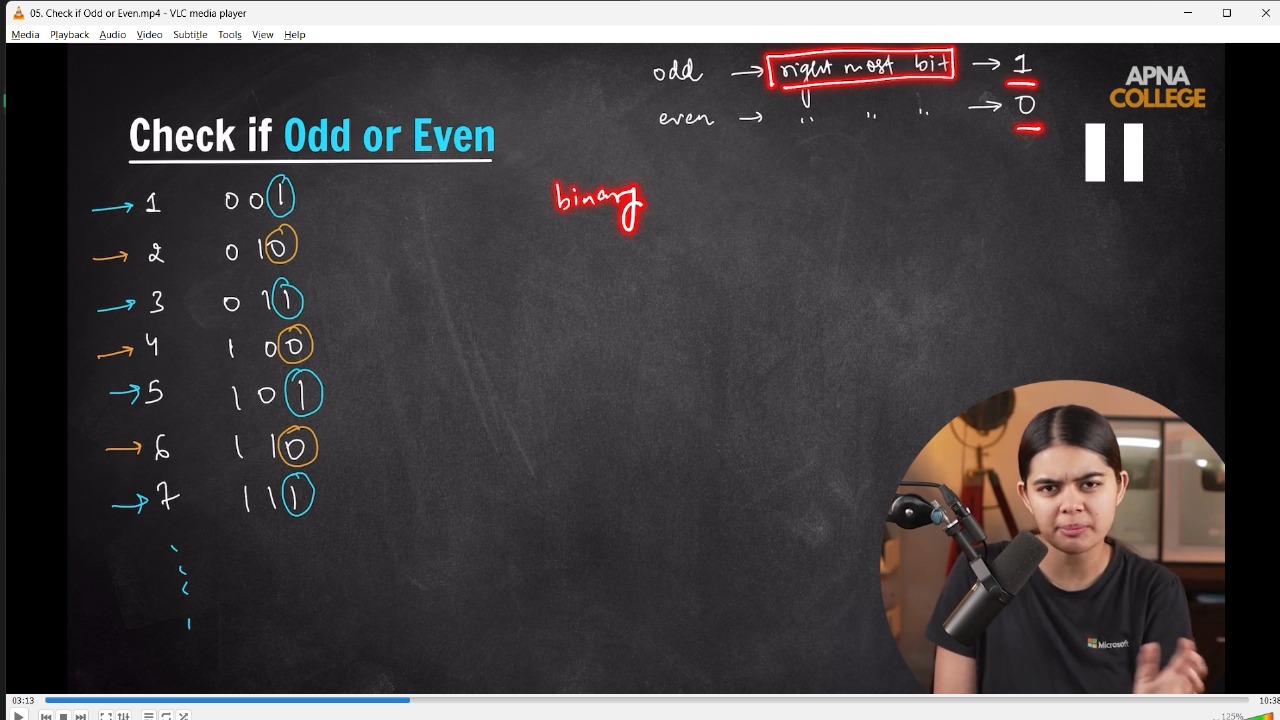
//     // 2) Output for  - 8>>1

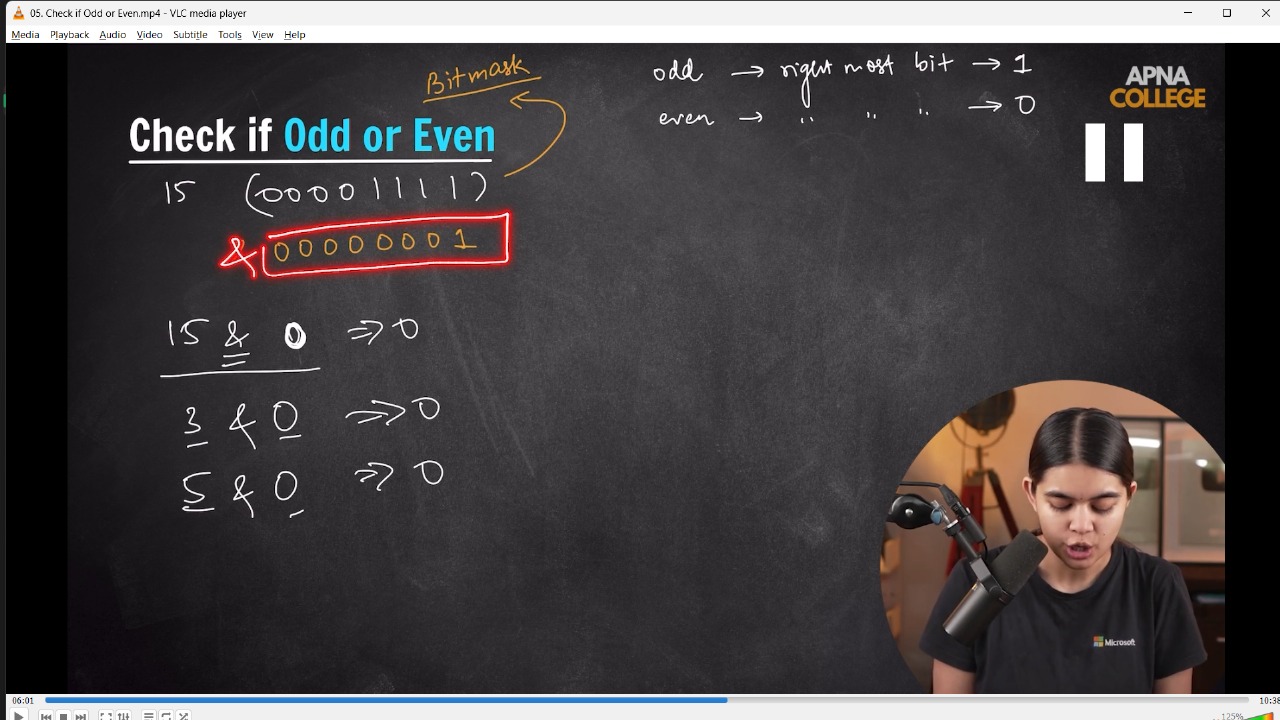
//     cout << (8 >> 1) << endl; // 4

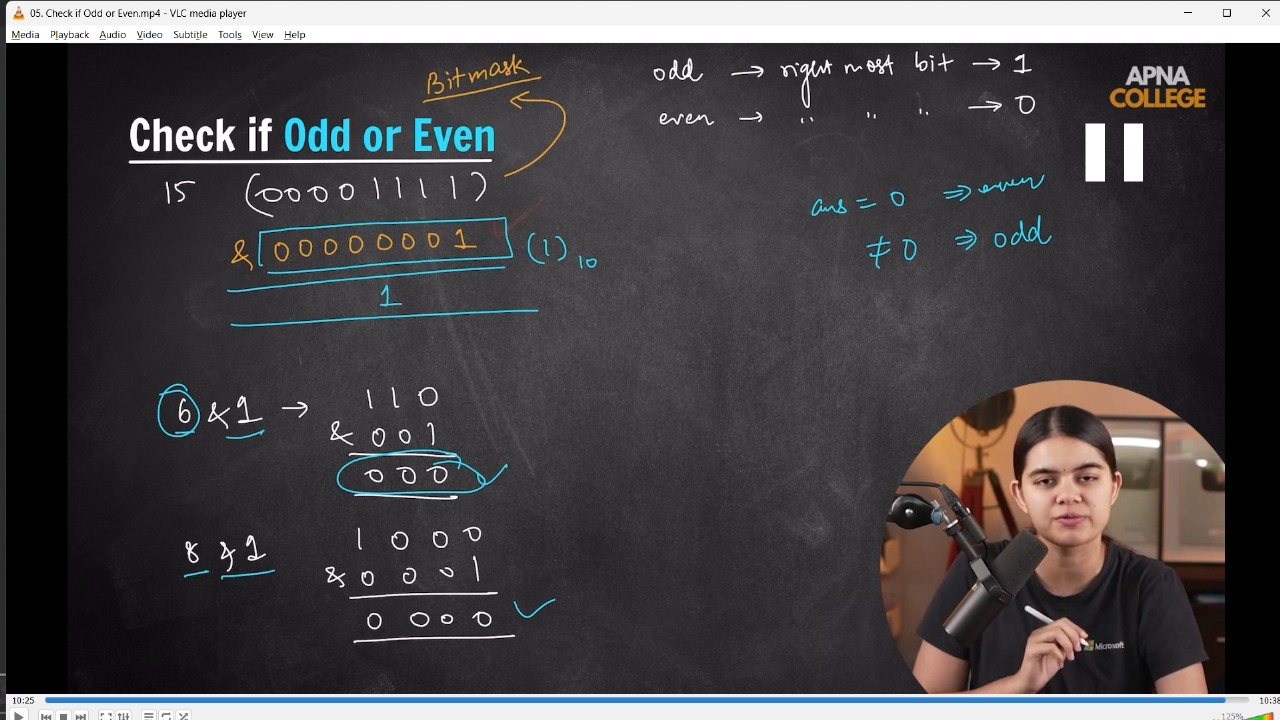
// }

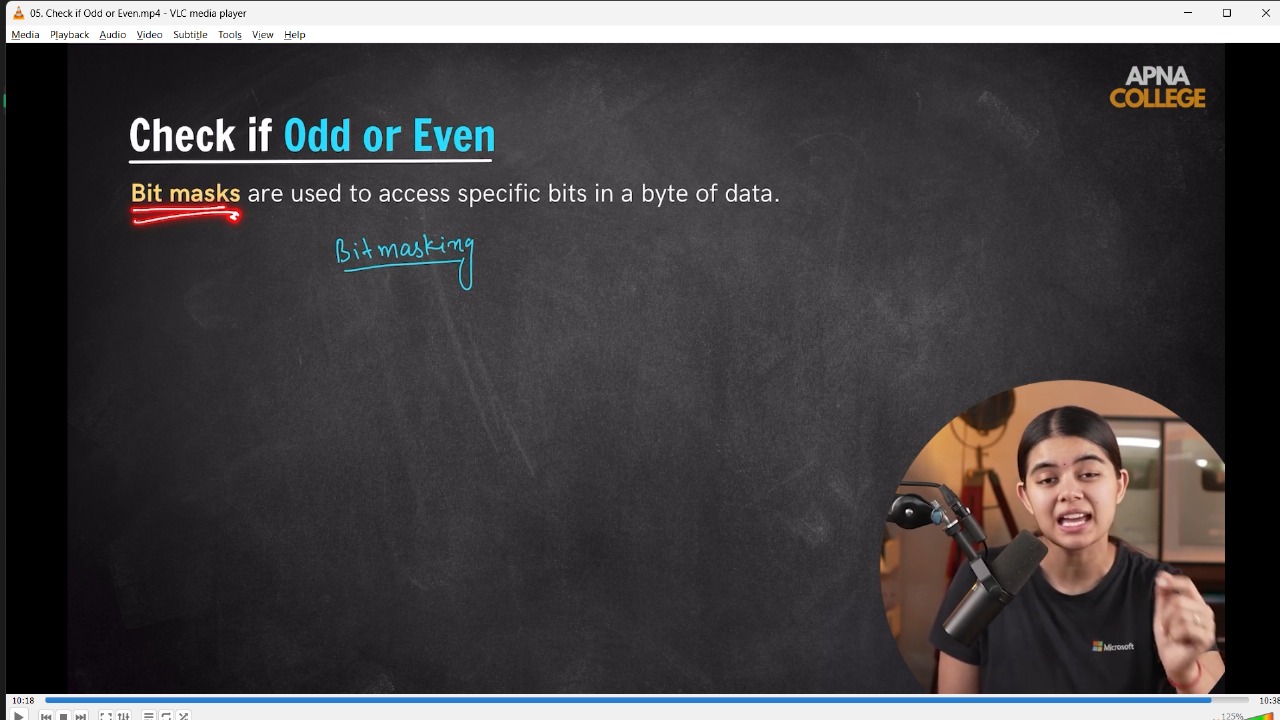
// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**2) Check if even or odd -**









/\* //📌 Logic - In the last of every odd no in Binary(right most bit) form it consist 1.

while in the last of every even no in Binary number(right most bit) it cobsist 0.

So, for check even odd just need to chck the last bit is 0 or 1?

It could be done using (&) Bitwise AND Operator.

Any No. & 0 = will always 0.

Any No. | 0 = Will always give that no.

Any Even No. & 1 = Will alyas 0

Any Odd No. & 1 != 0

 \*/

// void oddOrEven(int num)

// {

//     if ((num & 1) == 0)// here this 1 with AND is known as BitMask - are used to acess specifi c bits in a byute of data

//     {

//         cout << "Clearly, it's an Even Number" << endl;

//     }

//     else

//     {

//         cout << "Ohh No, it's an Odd Number dear" << endl;

//     }

// }

// int main()

// {

//     int num;

//     cout << "Enter the number you want tto check for Even or Odd - " << endl;

//     cin >> num;

//     oddOrEven(num);

// /\*

// Enter the number you want tto check for Even or Odd -

// 12

// Clearly, it's an Even Number

// Enter the number you want tto check for Even or Odd -

// 15

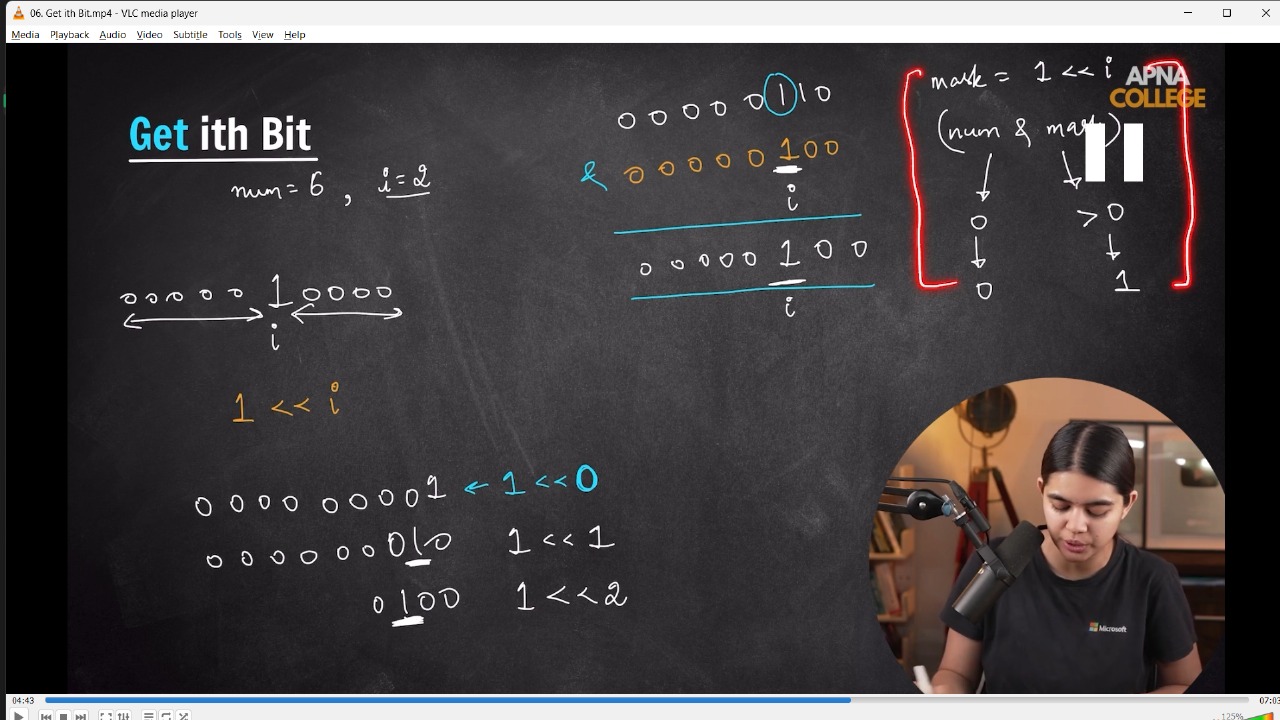
// Ohh No, it's an Odd Number dear

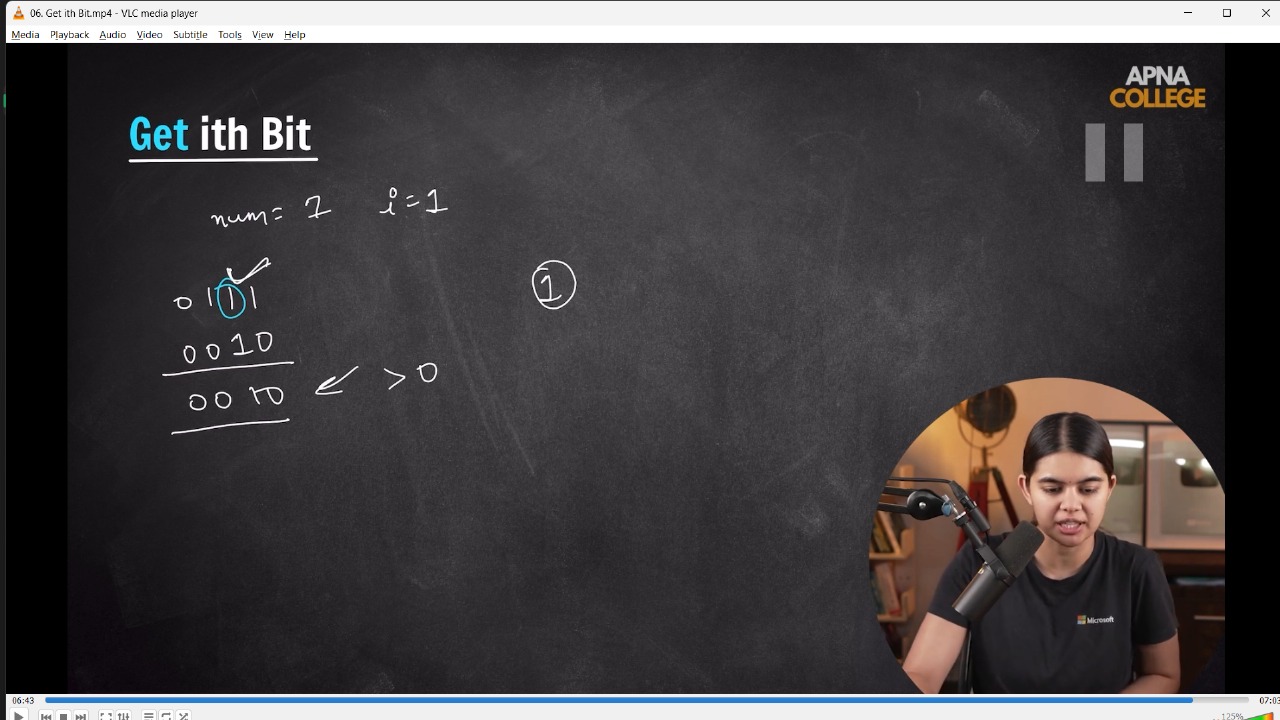
//  \*/

// 1}

// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**3) Get ith Bit of a num**





// int getIthBit(int num, int i)

// {

//     int BitMask = 1 << i;

//     if (!(num & BitMask))

//     {

//         return 0;

//     }

//     else

//     {

//         return 1;

//     }

// }

// int main()

// {

//     cout << getIthBit(6, 2) << endl; // 1 - in BNS 6 is - 0110 - so the ith - 2th means 3rd bit is 1

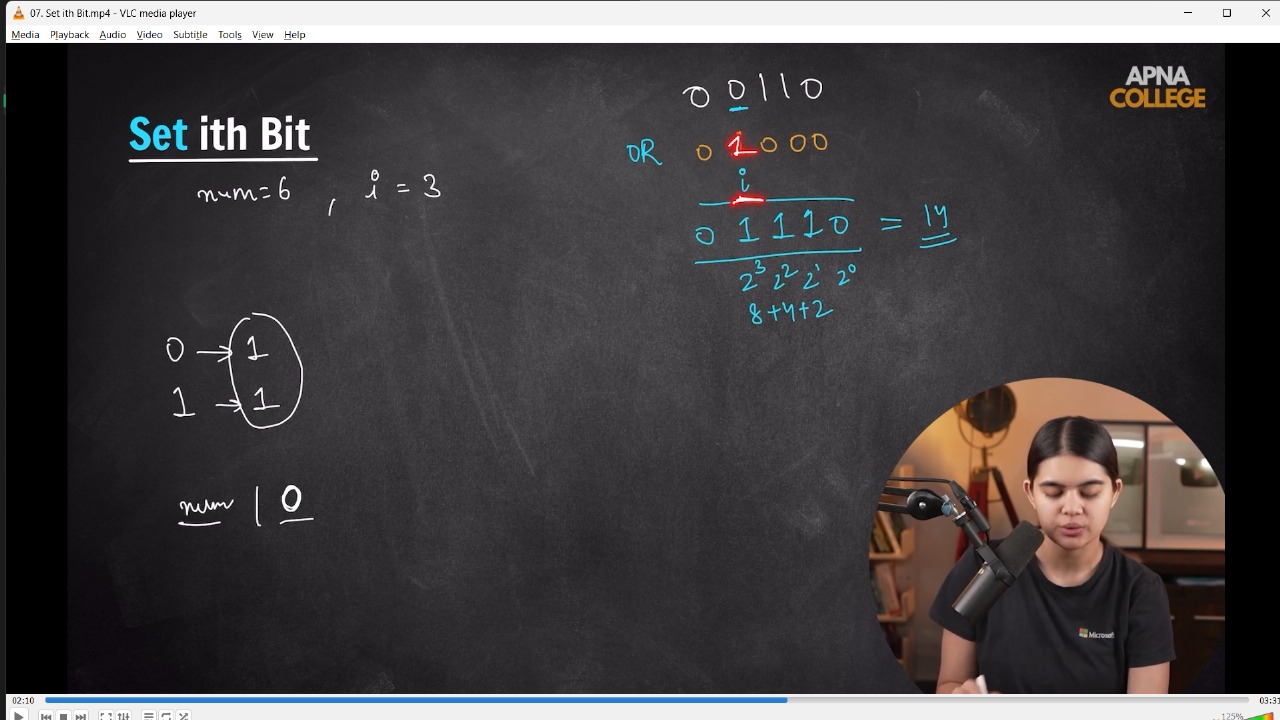
//     cout << getIthBit(7, 1) << endl; // 1 - in BNS 7 is - 0111 - so the ith - 1st means 2nd bit is 1

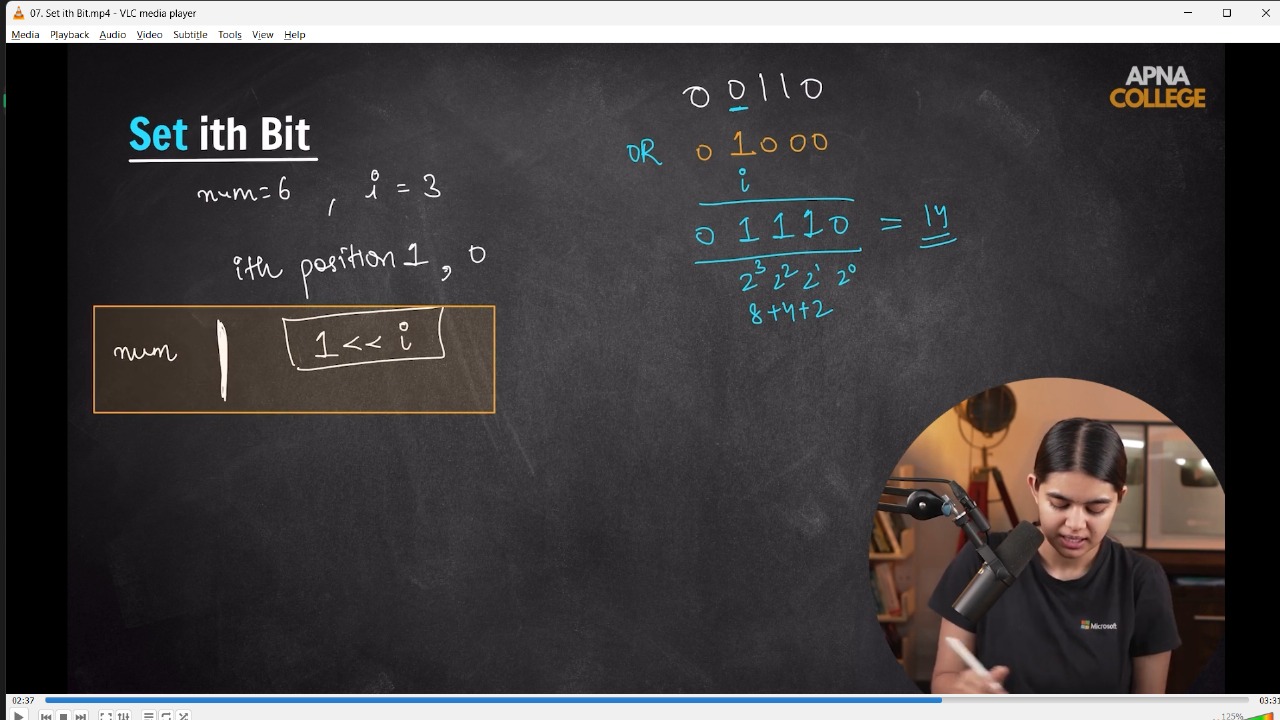
//     cout << getIthBit(5, 1) << endl; // 0 - in BNS 5 is - 0101 - so the ith - 1st means 2nd bit is 0

// }

// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**4) Set ith Bit - (sets the value to 1)**





// int setIthBit(int num, int i)

// {

//     int BitMask = 1 << i;

//     return (num | BitMask);

// }

// int main()

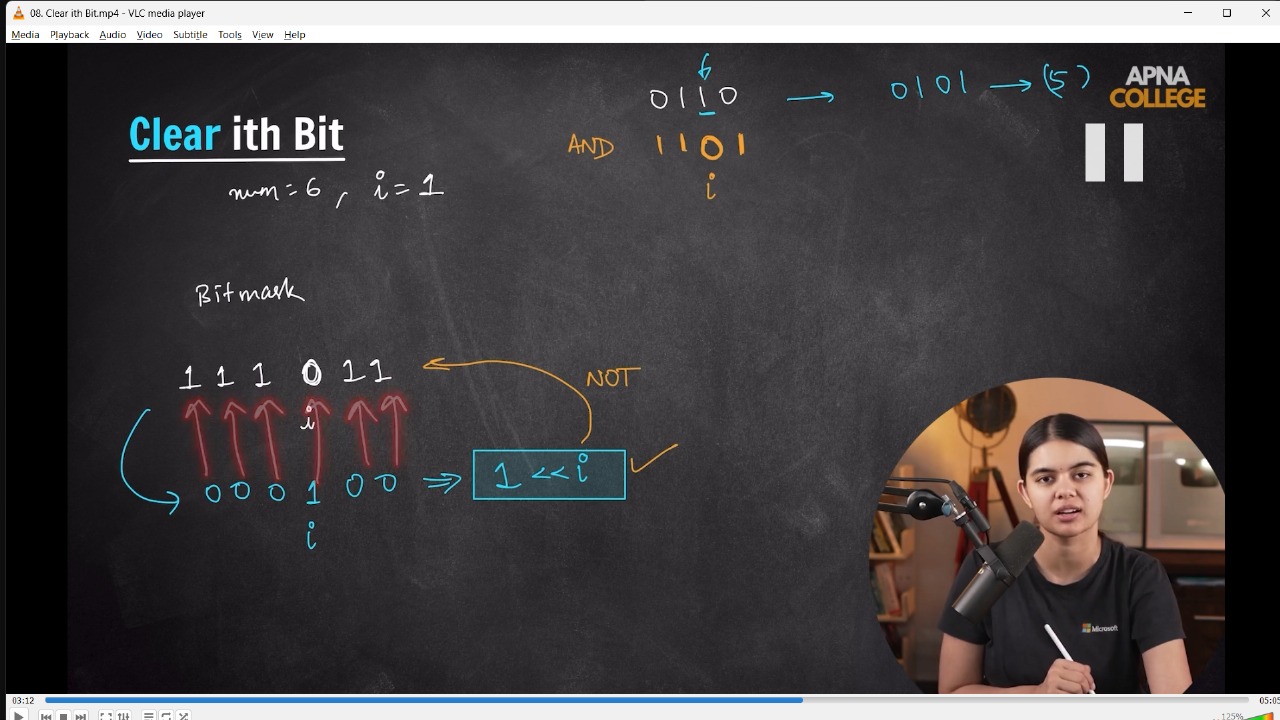
// {

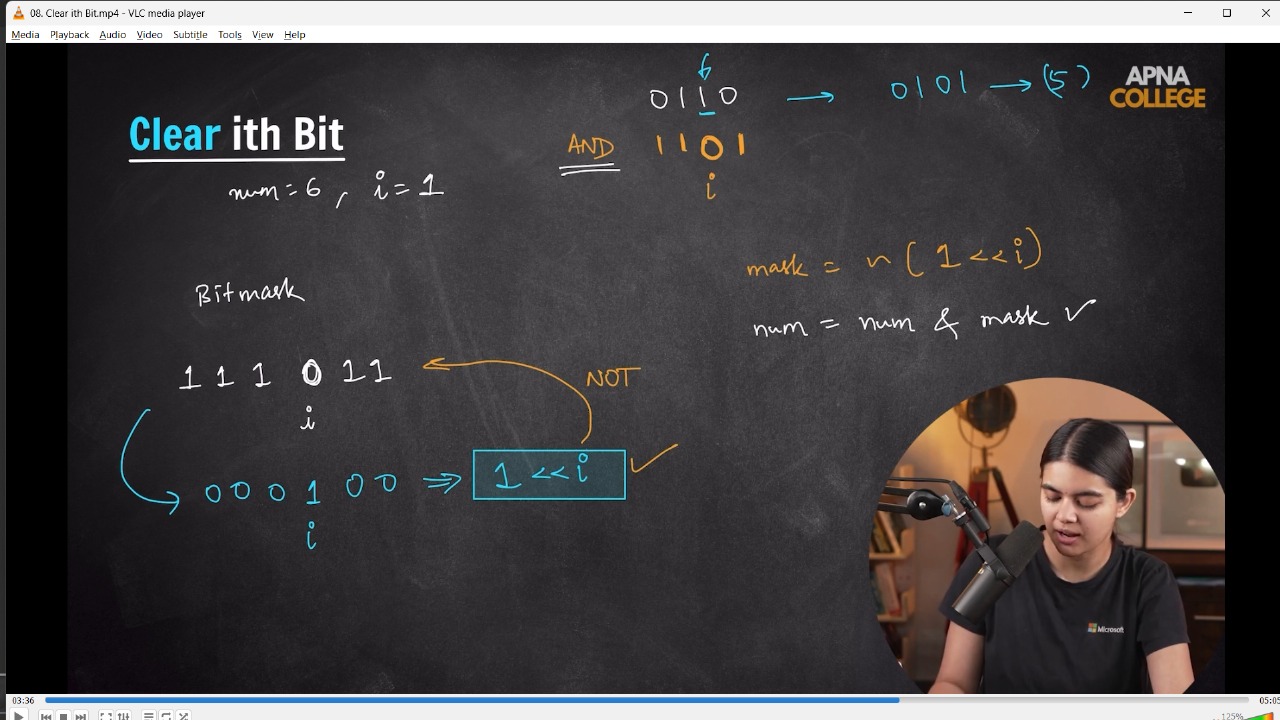
//     cout<<setIthBit(6,3)<<endl;//14 - in BNS 6 is - 0110 - so the ith - 3rd bit changes to 1 then it becomes 1110 which is 14

// }

// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**5)Clr ith Bit - (sets the value to 0)**





// int ClearIthBit(int num, int i)

// {

//     int BitMask = ~(1 << i);

//     return num & BitMask;

// }

// int main()

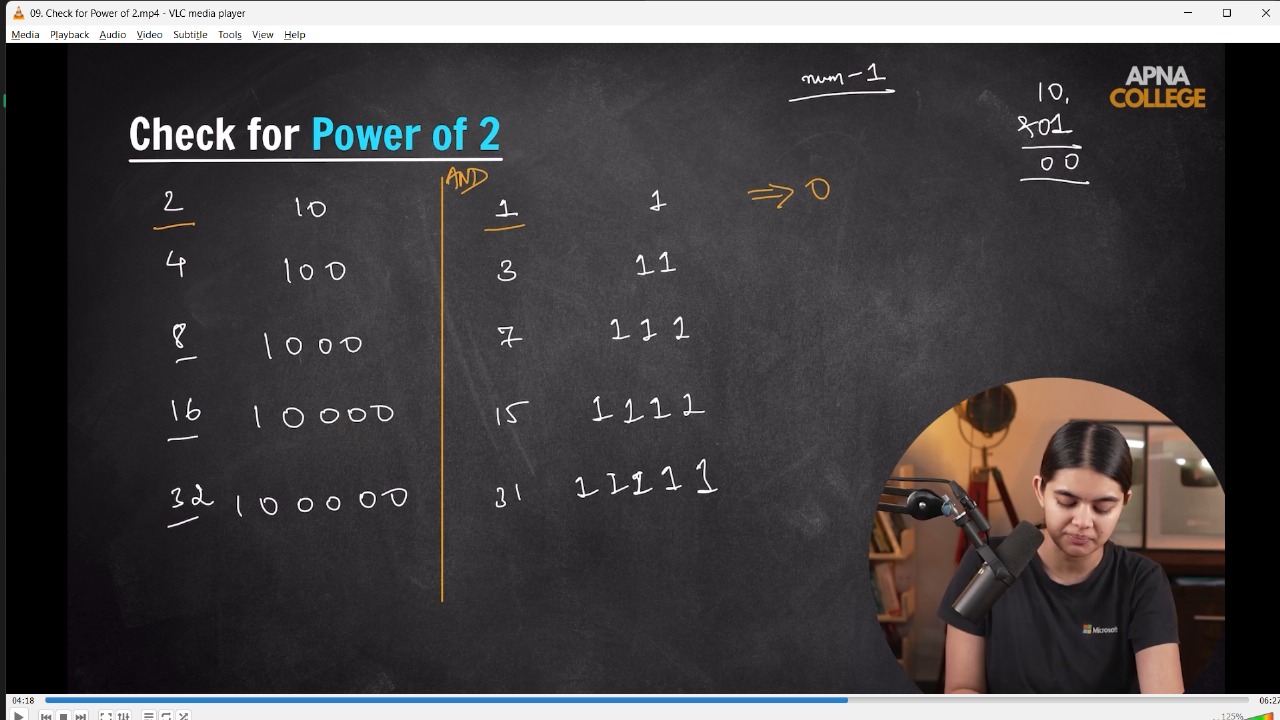
// {

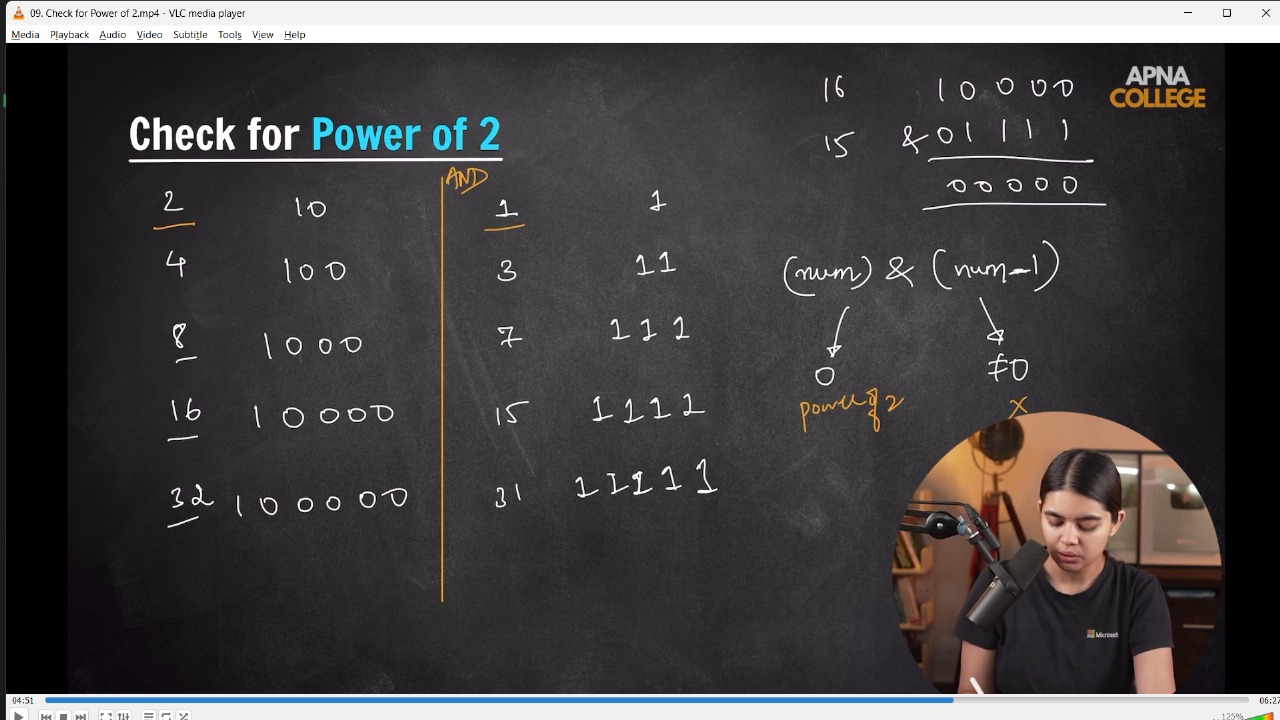
//     cout << ClearIthBit(6, 1) << endl;//4 - in BNS 6 is - 0110 - so the ith - 1st bit changes to 0 by using some op. then it becomes to 0100 which is 4

// }

// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**6) No. is power of 2 -**





/\*

📌 Logic - EVery no. which is power of two having only one 111 at LMSB, rest all the bits are 0

while in Odd no - the no. just above (-1) of power of 2 no. includes only 1 in the whole BN.

So, the diff of it if ==0 then its an even no. otherwise odd no.

\*/

// bool ispowerof2(int num)

// {

//     if (!(num & (num - 1)))

//     {

//         return true;

//     }

//     else

//     {

//         return false;

//     }

// }

// int main()

// {

//     cout<<ispowerof2(4)<<endl;//1

//     cout<<ispowerof2(31)<<endl;//0

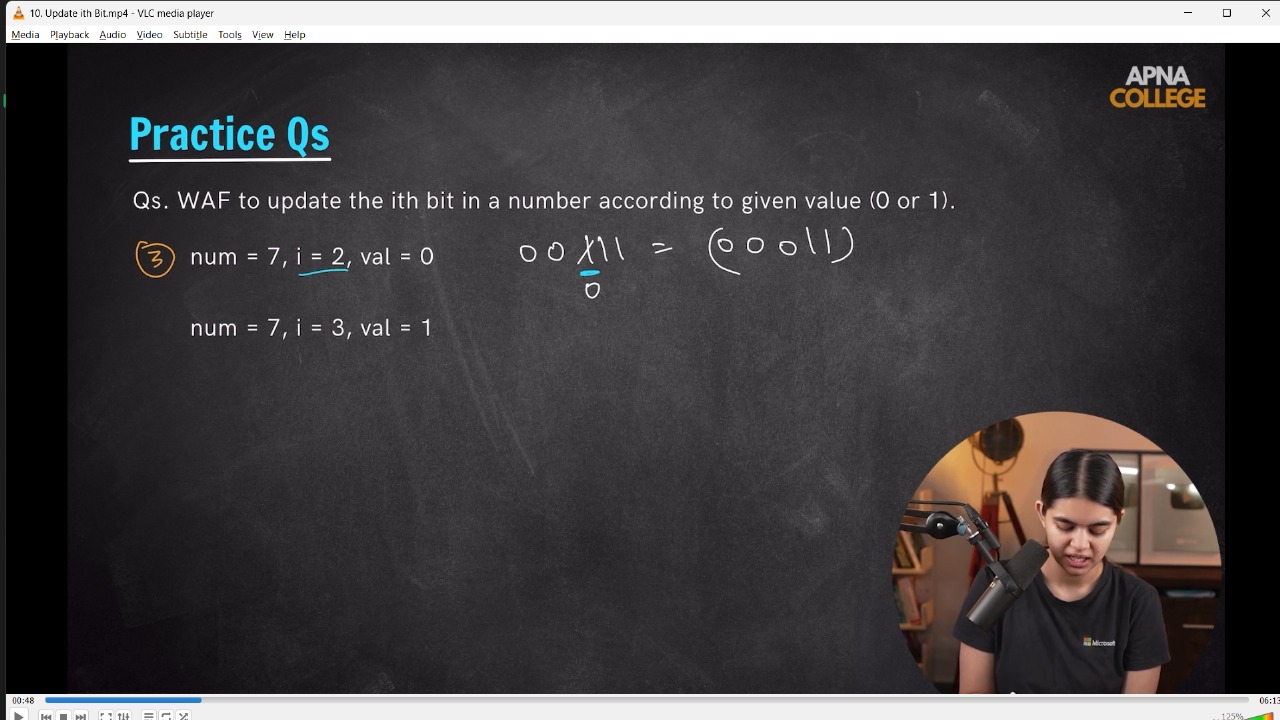
// }

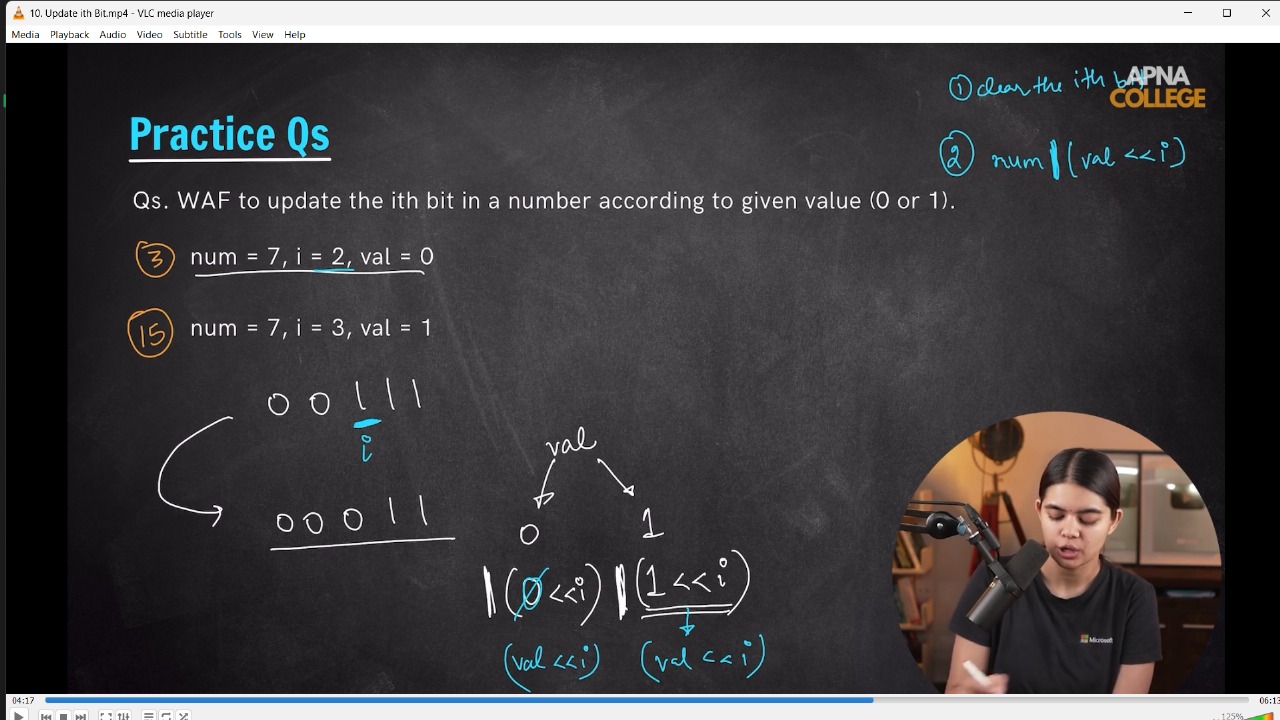
// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**// // QUn -**

**// // 1) num = 7, i=2, val = 0**

**// // 2) num = 7, i=3, val = 1;**





// void updateIthBit(int num, int i, int val)

// {

//     num = num & ~(1 << i); // clear the ith bit

//     num = num | (val << i);

//     cout << num << endl;

// }

// int main()

// {

//     updateIthBit(7, 2, 0); // 3 - on num 7 whoose value in BNS is 0111 and 2nd value in BNS should be 0 - so its : 0011 which is = 3

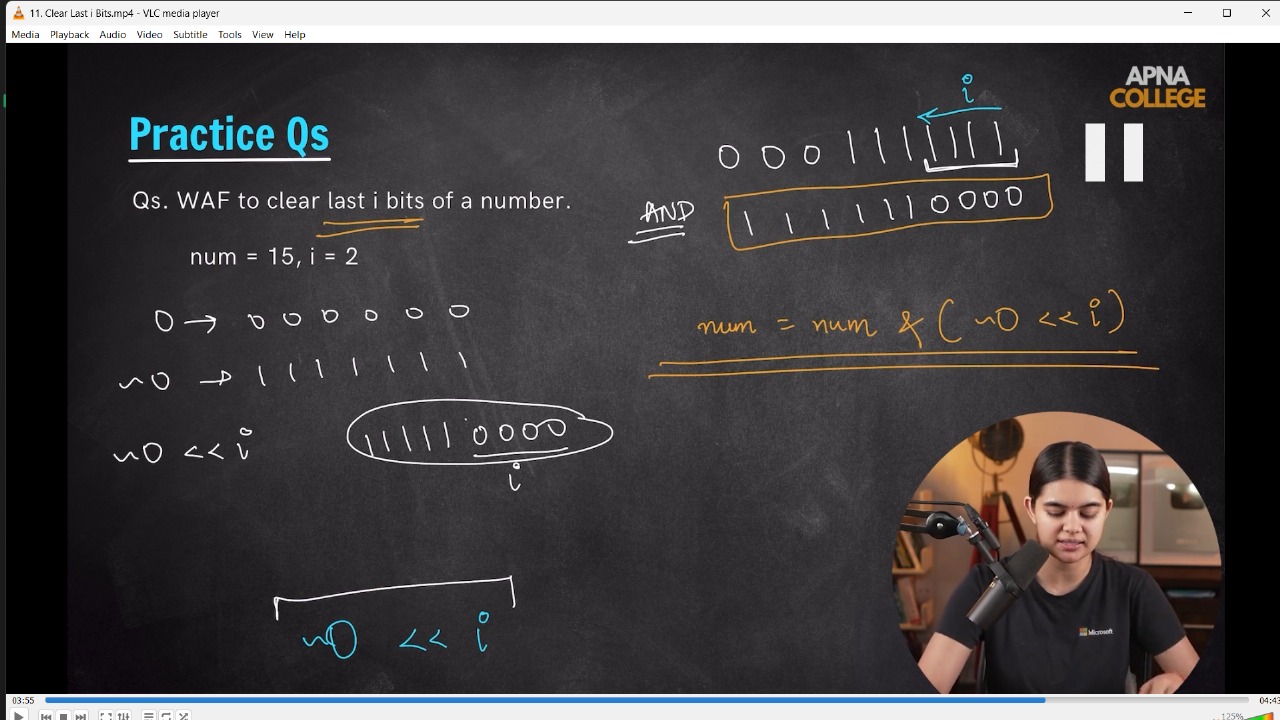
//     updateIthBit(7, 3, 1); // 15 - on num 7 whoose value in BNS is 0111 and 3rd value in BNS should be 1 - so its : 1111 which is = 15

// }

// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**// // Qun - WAF to clear last i bits of a numver - num = 15, i=2**





// void clearBits(int num, int i)

// {

//     int bitMask = (~0) << i;

//     num = num & bitMask;

//     cout << num << endl;

// }

// int main()

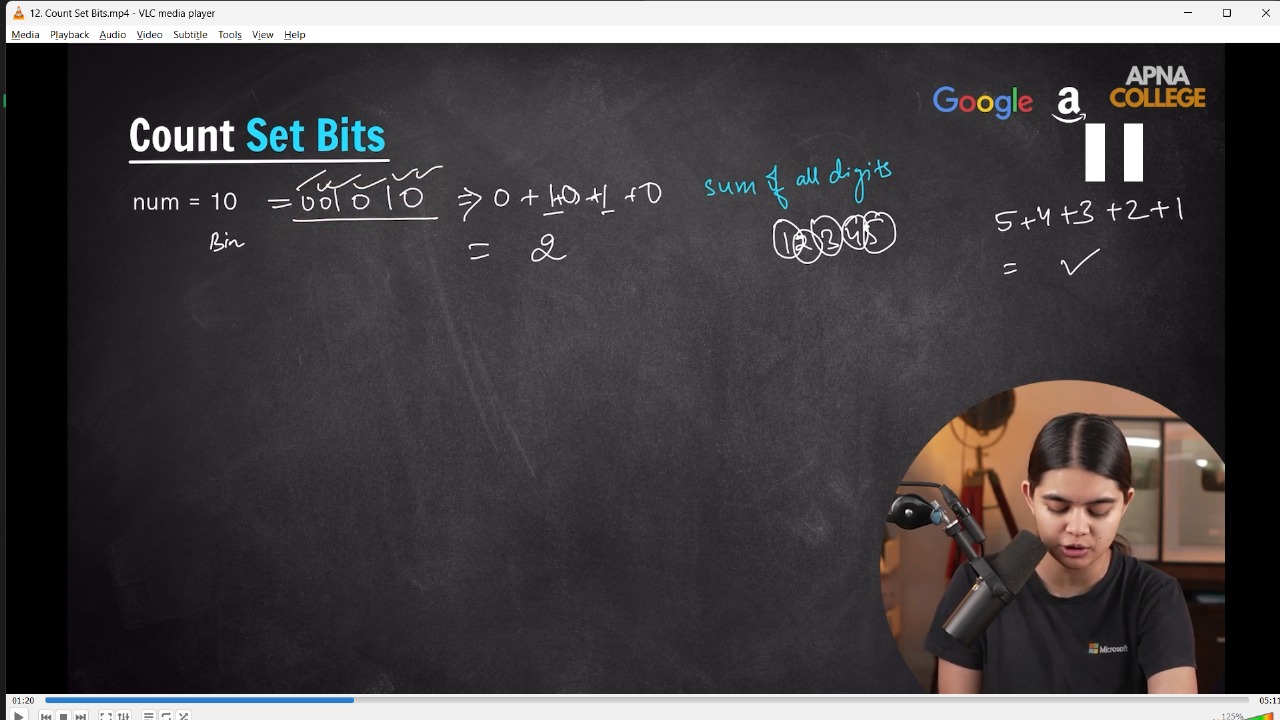
// {

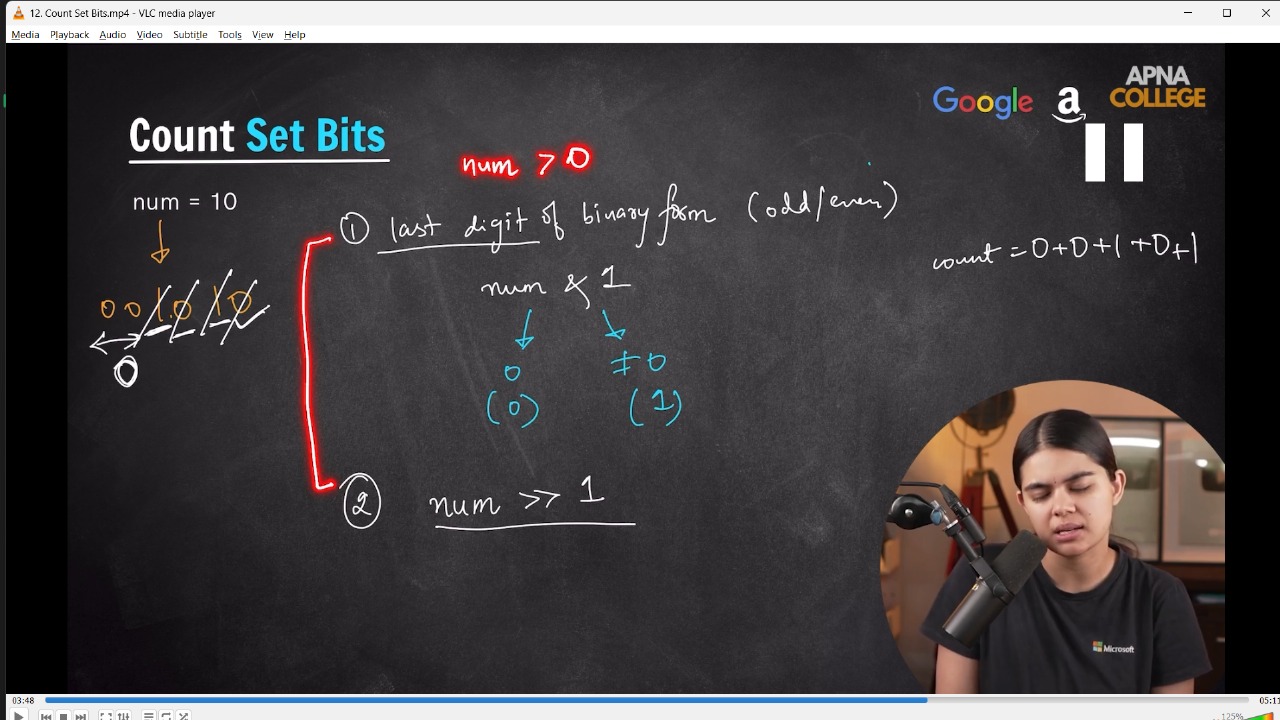
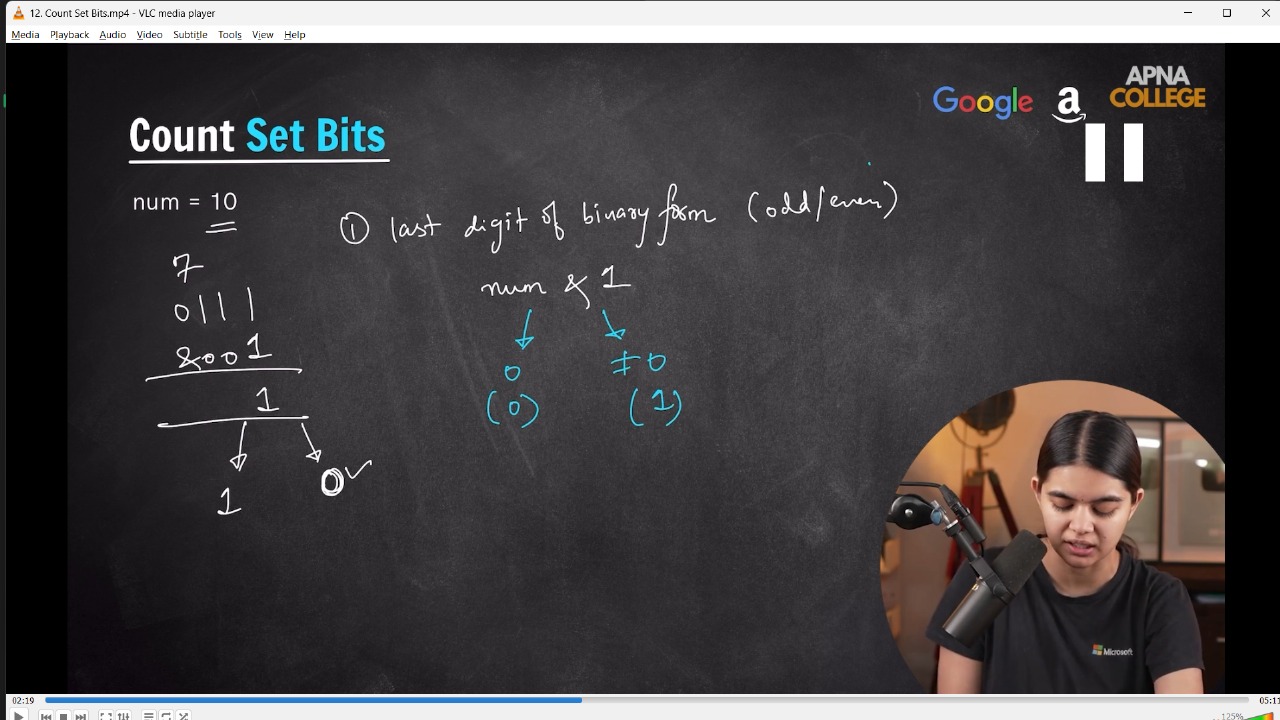
//     clearBits(15, 2);//12 - the num 15 in BNS is 1111, so 2bits clr from last then the no will be 1100 - which is equal to 12.

// }

// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**//7) Count of Set Bits -**





// int CountSetBits(int num)

// {

//     int count = 0;

//     while (num>0)

//     {

//         int lastDig = num&1;

//         count+=lastDig;

//         num = num>>1;

//     }

//     cout<<count<<endl;

//     return count;

// }

// int main()

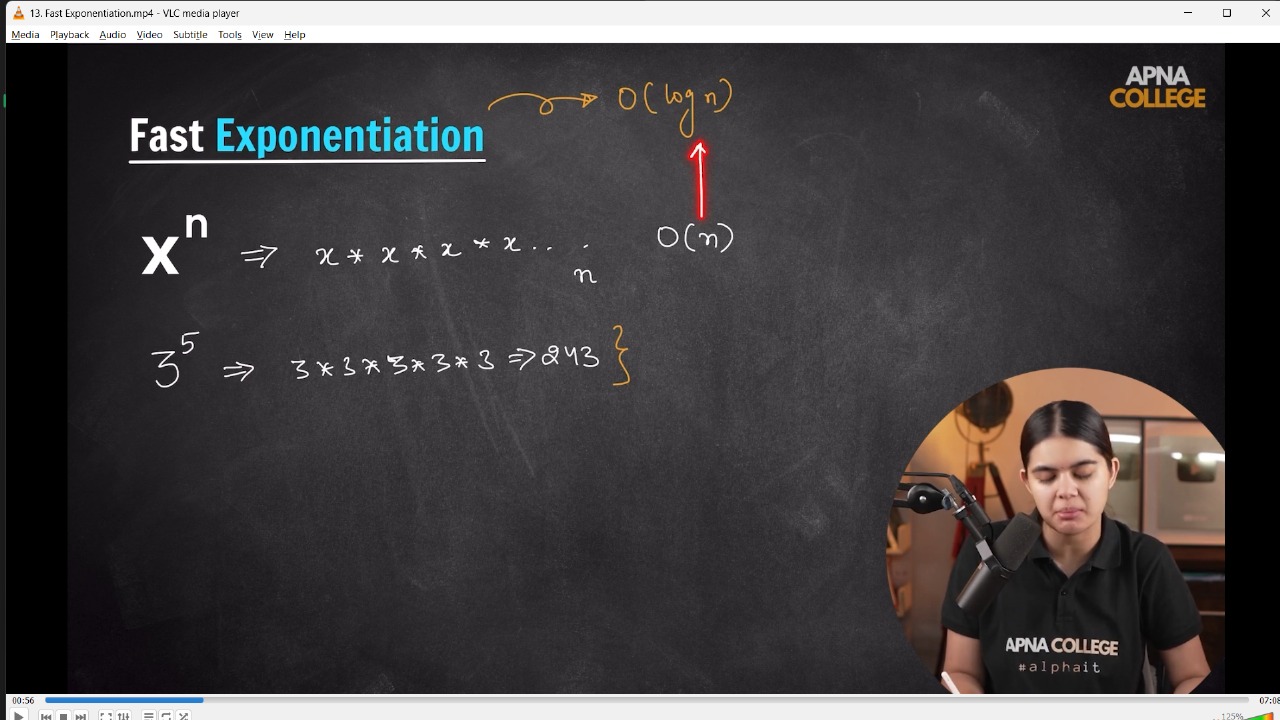
// {

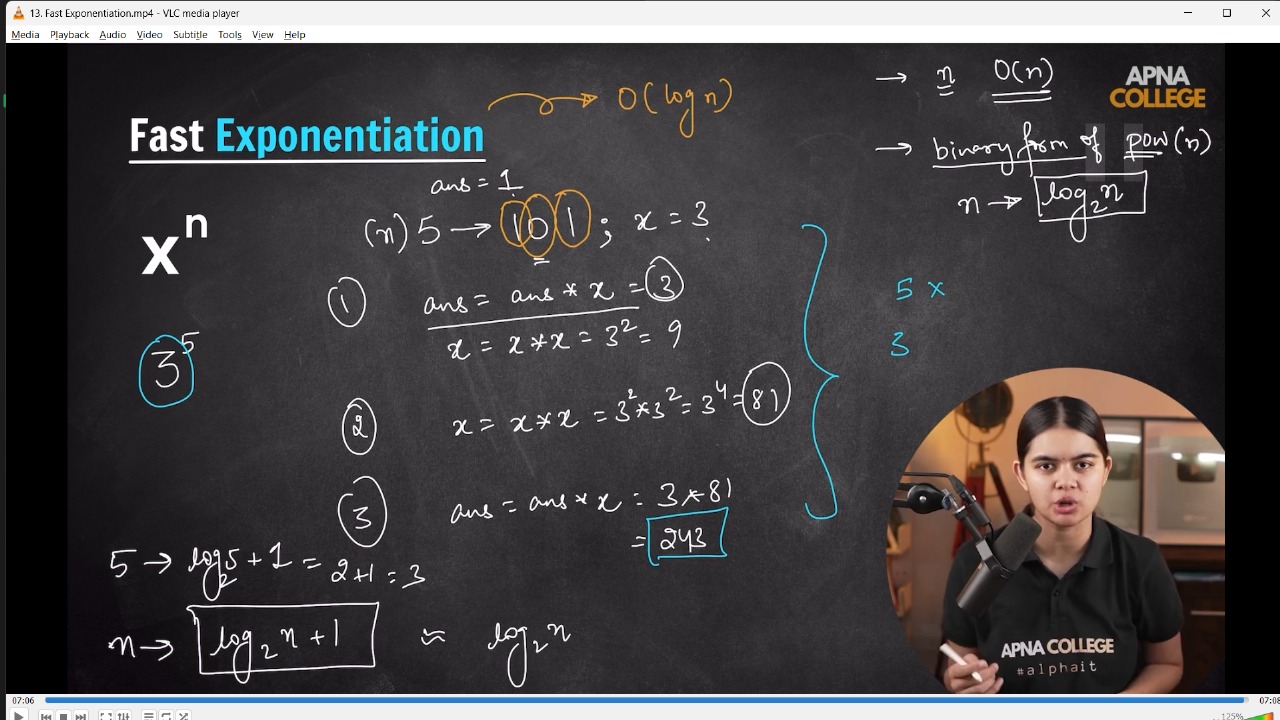
//     CountSetBits(10);//2 - in BNS 10 is 001010, so its count is 0+0+1+0+1+0 = 2

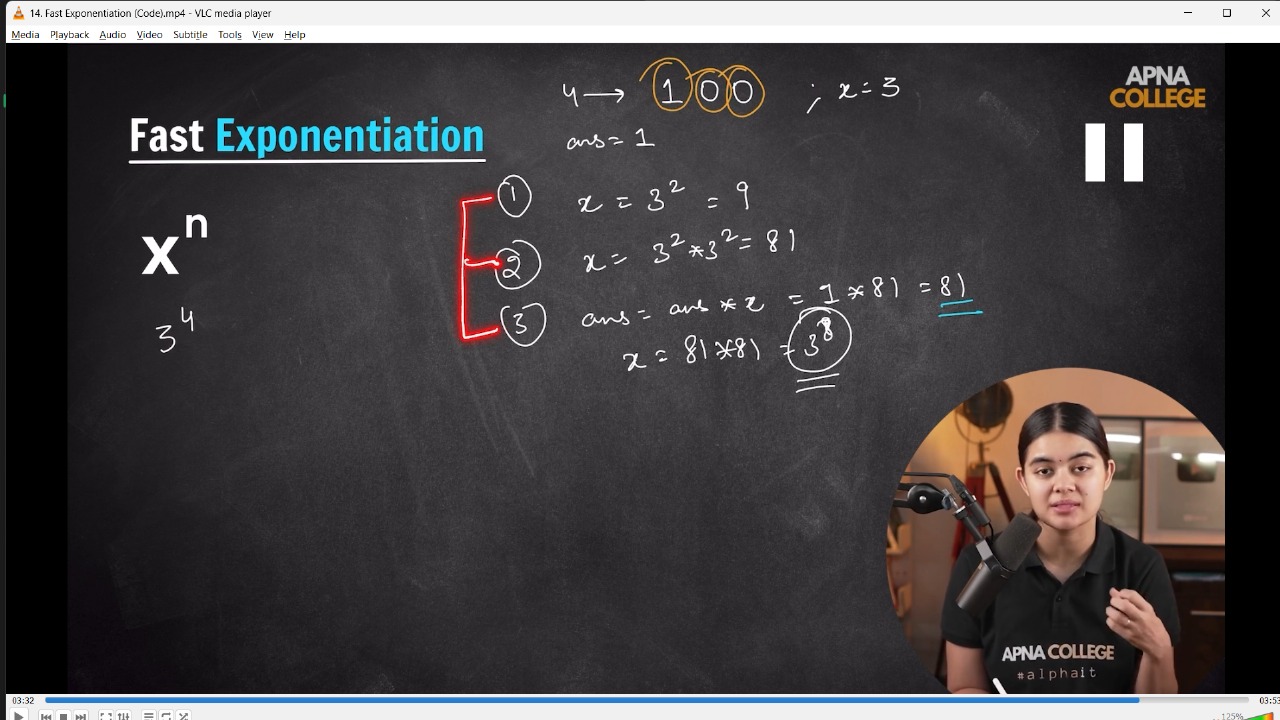
// }

// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

**//8) Fast Exponentiation -**







// void fastExpo(int x, int n)

// {

//     int ans = 1;

//     while (n > 0)

//     {

//         int lastbit = n & 1;

//         if (lastbit)

//         {

//             ans = ans \* x;

//         }

//         x = x \* x;

//         n = n >> 1;

//     }

//     cout<<ans<<endl;

// }

// int main()

// {

//     fastExpo(3,5);//243

// }

// \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_