**############################################################**

**1- Check whether K-th bit is set or not**

Given a number **N**and a bit number **K**, check if **Kth** bit of N is **set or not**. A bit is called set if it is 1. Position of set bit '1' should be **indexed starting with 0** from **LSB**side in binary representation of the number.  
  
**Example 1:**

**Input**: N = 4, K = 0

**Output**: No

**Explanation**: Binary representation of 4 is 100,

in which 0th bit from LSB is not set.

So, return false.

**Example 2:**

**Input**: N = 4, K = 2

**Output**: Yes

**Explanation**: Binary representation of 4 is 100,

in which 2nd bit from LSB is set.

So, return true.

**Example 3:**

**Input**: N = 500, K = 3

**Output**: No

**Explanation**: Binary representation of 500 is

111110100, in which 3rd bit from LSB is not set.

So, return false.

**Your task:**

You don't have to read input or print anything. Your task is to complete the **function checkKthbit**that takes **n and k as parameters** and **returns**either **true**(if kth bit is set) or **false**(if kth bit is not set).  
  
**Expected Time Complexity:** O(LogN).  
**Expected Auxiliary Space:** O(1).  
  
**Constraints:**  
1 ≤ N ≤ 109  
0 ≤ K ≤ floor(log2(N) + 1)

bool checkKthBit(int n, int k)

{ int mask=1;

while(k>0)

{ mask=mask<<1;

k--;

}

return n&mask;

}

**############################################################**

**2- Count total set bits**

You are given a number**N**. Find the **total count of set bits**for all numbers from 1 to N(both inclusive).  
  
**Example 1:**

**Input**: N = 4

**Output**: 5

**Explanation**:

For numbers from 1 to 4.

For 1: 0 0 1 = 1 set bits

For 2: 0 1 0 = 1 set bits

For 3: 0 1 1 = 2 set bits

For 4: 1 0 0 = 1 set bits

Therefore, the total set bits is 5.

**Example 2:**

**Input**: N = 17

**Output**: 35

**Explanation**: From numbers 1 to 17(both inclusive),

the total number of set bits is 35.

**Your Task:**The task is to complete the function **countSetBits**() that takes **n as a parameter**and returns the**count of all bits**.  
  
**Expected Time Complexity:** O(log N).  
**Expected Auxiliary Space:** O(1).  
  
**Constraints:**  
1 ≤ N ≤ 108

int highbit (int n)

{ int i = 0;

while ((1 << i) <= n)

i++;

return i-1;

}

int countSetBits (int n)

{ if (n == 0)

return 0;

int bit = highbit (n), base = 2, cpy;

int count = (1 << (bit - 1)) \* bit;

int remain = n - (1 << bit) + 1;

return count + remain + countSetBits(n - (1 << bit));

}

**############################################################**

**3- Number is sparse or not**

Given a number **N**. The task is to check whether it is**sparse or not**. A number is said to be a sparse number if **no two or more consecutive bits are set**in the binary representation.  
  
**Example 1:**

**Input**: N = 2

**Output**: true

**Explanation**: Binary Representation of 2 is 10,

which is not having consecutive set bits.

So, it is sparse number.

**Example 2:**

**Input**: N = 3

**Output**: false

**Explanation**: Binary Representation of 3 is 11,

which is having consecutive set bits in it.

So, it is not a sparse number.

**Your Task:**The task is to complete the function **checkSparse**() that takes **n as a parameter** and **returns true**if the number is sparse else returns **false**.

**Expected Time Complexity:** O(log N).  
**Expected Auxiliary Space:** O(1).  
  
**Constraints:**  
1 <= N <= 106

bool isSparse(int n)

{ if ((n & (n >> 1)) == 0)

return true;

else

return false;

}

**############################################################**

**4- Binary To Gray Code equivalent**

You are given a decimal number **N.**You need to find the gray code of the number N and convert it into decimal.  
To see how it's done, refer [here](https://www.geeksforgeeks.org/gray-to-binary-and-binary-to-gray-conversion/).  
  
**Example 1:**

**Input:** N = 7

**Output**: 4

**Explanation**: 7 is represented as 111 in

binary form. The gray code of 111 is 100,

in the binary form whose decimal equivalent

is 4.

**Example 2:**

**Input:** N = 10

**Output**: 15

**Explanation**: 10 is represented as 1010 in

binary form. The gray code of 1010 is 1111,

in the binary form whose decimal equivalent

is 15.

**Example 3:**

**Input:** N = 0

**Output**: 0

**Explanation**: Zero is represented as zero

in binary, gray, and decimal.

**Your Task:**The task is to complete the function **greyConverter**() which takes **n**as a parameter and returns it's equivalent **gray code**.  
  
**Expected Time Complexity:** O(1).  
**Expected Auxiliary Space:** O(1).  
  
**Constraints:**  
0 <= N <= 109

int greyConverter(int n)

{

return n^(n>>1);

}

**############################################################**

**5- Power of 2**

Given a non-negative integer **N**. The task is to check if N is a power of **2**. More formally, check if**N**can be expressed as **2x**for some **x.**

**Example 1:**

**Input:** N = 1

**Output:** true

**Explanation:**

1 is equal to 2 raised to 0 (20 = 1).

**Example 2:**

**Input:** N = 98

**Output:** false

**Explanation:**

98 cannot be obtained by any power of 2.

**Your Task:**Your task is to complete the function **isPowerofTwo**() which takes **n**as a parameter and returns **true or false** by **checking** is given number can be represented as a power of two or not.  
  
**Expected Time Complexity:** O(log N).  
**Expected Auxiliary Space:** O(1).  
  
**Constraints:**  
0 ≤ N ≤ 1018

bool isPowerofTwo(long long n){

if (n == 0)

return false;

return ((n & (n-1)) == 0)? true: false;

}

**############################################################**

**6- Gray to Binary equivalent**

Given**N** in Gray code equivalent. Find its binary equivalent.   
**Note**: We need to find the binary equivalent of the given gray code and return the decimal equivalent of the binary representation.  
  
**Example 1:**

**Input:** N = 4

**Output**: 7

**Explanation**:

4 is represented as gray 100 and its

binary equivalent is 111 whose decimal

equivalent is 7.

**Example 2:**

**Input**: N = 15

**Output**: 10

**Explanation**:

15 is represented as gray 1111 and its

binary equivalent is 1010 i.e. 10 in decimal.

**Example 3:**

**Input**: N = 0

**Output**: 0

**Explanation**: Zero is zero in gray and in

binary and decimal.

**Your Task:**Your task is to complete the function **grayToBinary**() which accepts an integer **n** as a parameter and returns decimal of the binary equivalent of the given gray number.   
  
**Expected Time Complexity:** O(log N).  
**Expected Auxiliary Space:** O(1).   
  
**Constraints:**  
0 <= N <= 108

int grayToBinary(int n)

{

int ans = n;

while (n > 0)

{

n = n >> 1;

ans = ans ^ n;

}

return ans;

}

**############################################################**

**7- Bit Difference**

You are given two numbers **A** and **B**. The task is to **count the number of bits needed to be flipped**to **convert**A to B.  
  
**Example 1:**

**Input:** A = 10, B = 20

**Output**: 4

**Explanation**:

A  = 01010

B  = 10100

As we can see, the bits of A that need

to be flipped are **0101**0. If we flip

these bits, we get 10100, which is B.

**Example 2:**

**Input**: A = 20, B = 25

**Output**: 3

**Explanation**:

A  = 10100

B  = 11001

As we can see, the bits of A that need

to be flipped are 1**01**0**0**. If we flip

these bits, we get 11001, which is B.

**Your Task:**The task is to complete the function **countBitsFlip**() that **takes A and B** as parameters and **returns**the **count**of the **number of bits to be flipped** to convert**A to B**.  
  
**Expected Time Complexity:** O(log N).  
**Expected Auxiliary Space:** O(1).  
  
**Constraints:**  
1 ≤ A, B ≤ 106

int countBitsFlip (int a, int b)

{

int c = a ^ b;

int count = 0;

while (c > 0)

{ c = c & (c - 1);

count++;

}

return count;

}

**############################################################**

**8- Find first set bit**

Given an integer an **N.** The task is to return the position of **first set bit found from the right side** in the binary representation of the number.  
**Note**: If there is no set bit in the integer N, then return 0 from the function.

**Example 1:**

**Input**: N = 18

**Output**: 2

**Explanation**: Binary representation of

18 is 010010,the first set bit from the

right side is at position 2.

**Example 2:**

**Input**: N = 12

**Output**: 3

**Explanation**: Binary representation

of 12 is 1100, the first set bit

from the right side is at position 3.

**Your Task:**  
The task is to complete the function **getFirstSetBit**() that takes an integer **n** as a **parameter**and **returns**the**position of first set bit**.

**Expected Time Complexity:** O(log N).  
**Expected Auxiliary Space:** O(1).

**Constraints:**  
0 <= N <= 108

unsigned int getFirstSetBit(int n)

{

if(n == 0)

return 0;

unsigned int i = 1;

int mask = 1;

while (n)

{ if(mask & n)

break;

n = n >> 1;

i++;

}

return i;

}

**############################################################**

**9- Rightmost different bit**

Given two numbers **M** and **N**. The task is to find the position of the **rightmost different** bit in the binary representation of numbers.

**Example 1:**

**Input:** M = 11, N = 9

**Output:** 2

**Explanation:** Binary representation of the given

numbers are: 1011 and 1001,

2nd bit from right is different.

**Example 2:**

**Input:** M = 52, N = 4

**Output**: 5

**Explanation**: Binary representation of the given

numbers are: 110100 and 0100,

5th-bit from right is different.

**User Task:**  
The task is to complete the function **posOfRightMostDiffBit**() which takes**two arguments m and n** and **returns** the **position of first different bits in m and n**. If both m and n are the same then return **-1** in this case.

**Expected Time Complexity:** O(max(log m, log n)).  
**Expected Auxiliary Space:** O(1).

**Constraints:**  
1 <= M <= 109  
1 <= N <= 109

int posOfRightMostDiffBit(int m, int n)

{

int c = m ^ n;

int i = 1, mask = 1;

while(c)

{

if(mask & c)

break;

i++;

c = c >> 1;

}

return i;

}

**############################################################**

**10- Swap all odd and even bits**

Given an unsigned integer **N**. The task is to swap all odd bits with even bits. For example, if the given number is 23 (**0**0**0**1**0**1**1**1), it should be converted to 43(0**0**1**0**1**0**1**1**). Here, every even position bit is swapped with adjacent bit on the right side(even position bits are highlighted in the binary representation of 23), and every odd position bit is swapped with an adjacent on the left side.  
  
**Example 1**:

**Input**: N = 23

**Output**: 43

**Explanation**:

Binary representation of the given number

is **0**0**0**1**0**1**1**1 after swapping

0**0**1**0**1**0**11 = 43 in decimal.

**Example 2**:

**Input**: N = 2

**Output**: 1

**Explanation**:

Binary representation of the given number

is **1**0 after swapping 0**1** = 1 in decimal.

**Your Task:**Your task is to complete the function **swapBits**() which takes an integer and **returns an**integer with all the odd and even bits swapped.

**Expected Time Complexity:** O(1).  
**Expected Auxiliary Space:** O(1).  
  
**Constraints:**  
1 ≤ N ≤ 109

unsigned int swapBits(unsigned int n)

{ unsigned int maske = 0xAAAAAAAA;

unsigned int masko = 0x55555555;

return ((n & maske) >> 1) | ((n & masko) << 1);

}

**############################################################**

**11- Maximum subset XOR**

Given an array arr[] of N positive integers. Find an integer denoting the**maximum XOR subset** value in the given array arr[].

**Example 1:**

**Input :**

N = 3

arr[] = {2, 4, 5}

**Output :** 7

**Explanation :**

The subset {2, 5} has maximum

subset XOR value.

**Example 2 :**

**Input :**

N= 3

arr[] = {9, 8, 5}

**Output :** 13

**Explanation :**

The subset {8, 5} has maximum

subset XOR value.

**Your Task :**

You don't need to read input or print anything. Your task is to complete the function **maxSubarrayXOR()** which takes the array and an integer as input and returns the maximum subset XOR value.

**Expected Time Complexity :** O(N\*Log(max(arr[i])))  
**Expected Auxiliary Space :**O(1)

**Contraints :**  
1 <= N <= 105

1 <= arr[i] <= 106

int maxSubarrayXOR (int arr[] , int n)

{

int index = 0;

for (int i = 31; i >= 0; i--)

{ int maxInd = index;

int maxEle = INT\_MIN;

for (int j = index; j < n; j++)

{ if ( (arr[j] & (1 << i)) != 0

&& arr[j] > maxEle )

maxEle = arr[j], maxInd = j;

}

if (maxEle == INT\_MIN)

continue;

swap(arr[index], arr[maxInd]);

maxInd = index;

for (int j=0; j<n; j++)

{ if (j != maxInd &&

(arr[j] & (1 << i)) != 0)

arr[j] = arr[j] ^ arr[maxInd];

}

index++;

}

int res = 0;

for (int i = 0; i < n; i++)

res ^= arr[i];

return res;

}