**1. Find first set bit**

Given an integer an **N.** The task is to print the position of **first set bit found from right side** in the binary representation of the number.

**Input:**  
The first line of the input contains an integer T, denoting the number of test cases. Then T test cases follow. The only line of the each test case contains an integer N.

**Output:**  
For each test case print in a single line an integer denoting the position of the first set bit found form right side of the binary representation of the number.**If there is no set bit print "0"**.  
  
**Constraints:**  
1 <= T <= 200  
0 <= N <= 106  
  
**Example:**  
**Input:**  
2  
18  
12

**Output:**  
2  
3

CODE:

using namespace std;

int fun(int n)

{

if(n==0 || n==1)

return n;

int cnt=0;

while(n)

{

int r=n%2;

cnt++;

if(r==1)

return cnt;

n=n/2;

}

return 0;

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

cout<<fun(n)<<endl;

}

return 0;

}

**2. Rightmost different bit**

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

**Input:**  
The input line contains T, denoting the number of testcases. Each testcase follows. First line of each testcase contains two space separated integers M and N.

**Output:**  
For each testcase in new line, print the position of rightmost different bit in binary representation of numbers. If both M and N are same then print **-1** in this case.

**Constraints:**  
1 <= T <= 100  
1 <= M <= 103  
1 <= N <= 103

**Example:**  
**Input:**  
2  
11 9  
52 4

**Output:**  
2  
5

CODE:

using namespace std;

int fun(int n,int m)

{

int cnt=0;

while(n || m)

{

int r1=n%2;

int r2=m%2;

cnt++;

if(r1!=r2)

return cnt;

n=n/2;

m=m/2;

}

return cnt;

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n,m;

cin>>n>>m;

cout<<fun(n,m)<<endl;

}

return 0;

}

**3. 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 **RSB**side in binary representation of the number. Consider N = 4(100):  0th bit = 0, 1st bit = 0, 2nd bit = 1.

**Input:**  
The first line of input contains an integer T denoting the number of test cases. Then T test cases follow. Each test case consists of two lines. The first line of each test case contain an integer  **N**. The second line of each test case contains an integer  **K**.

**Output:**  
Corresponding to each test case, print "**Yes**" (without quotes) if**Kth** bit is set else print "**No**" (without quotes) in a new line.

**Constraints:**  
1 ≤ T ≤ 200  
1 ≤ N ≤ 109  
0 ≤ K ≤ floor(log2(N) + 1)

**Example:  
Input**:  
3  
4  
0  
4  
2  
500  
3

**Output**:  
No  
Yes  
No

CODE:

int main()

{

int t;

cin>>t;

while(t--)

{

int n,k;

cin>>n>>k;

if(n & 1<<k )

cout<<"Yes"<<endl;

else

cout<<"No"<<endl;

}

return 0;

}

**4. Toggle bits given range**

Given a non-negative number **N** and two values **L** and **R**. The problem is to toggle the bits in the range L to R in the binary representation of N, i.e, to toggle bits from the rightmost Lth bit to the rightmost Rth bit. A toggle operation flips a bit 0 to 1 and a bit 1 to 0. Print N after the bits are toggled.

**Example 1:**

**Input:**

**N** = 17 , **L** = 2 , **R** = 3

**Output:**

23

**Explanation:**

(17)10 = (10001)2. After toggling all

the bits from 2nd to 3rd position we get

(10111)2 = (23)10

**Example 2:**

**Input:**

**N** = 50 , **L** = 2 , **R** = 5

**Output:**

44

**Explanation:**

(50)10 = (110010)2. After toggling all

the bits from 2nd to 3rd position we get

(101100)2 = (44)10

**Your Task:**  
You don't need to read input or print anything. Your task is to complete the function **toggleBits()** which takes 3 Integers N,L and R as input and returns the answer.

**Expected Time Complexity:** O(1)  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
1 <= N <= 105  
1 <= L<=R <=Number of Bits in N

CODE:

#include <bits/stdc++.h>

using namespace std;

// } Driver Code Ends

class Solution {

public:

int toggleBits(int n , int l , int r) {

int cnt=0;

int i=0;

int ans=0;

while(n)

{

int rem=n%2;

cnt++;

//cout<<r<<" ";

//cout<<"cnt="<<cnt;

if(cnt>=l && cnt<=r)

{

// cout<<"hi";

if(rem==0)

rem=1;

else

rem=0;

}

//cout<<-r<<" ";

ans+=rem\*pow(2,i);

i++;

n=n/2;

}

// cout<<endl;

return ans;

// code here

}

};

// { Driver Code Starts.

int main() {

int t;

cin >> t;

while (t--) {

int N,L,R;

cin>>N>>L>>R;

Solution ob;

cout << ob.toggleBits(N,L,R) << endl;

}

return 0;

}

**5. Set kth bit**

Given a number **N** and a value **K**. From the right, set the Kth bit in the binary representation of N. The position of Least Significant Bit(or last bit) is 0, the second last bit is 1 and so on.

**Example 1:**

**Input:**

N = 10

K = 2

**Output:**

14

**Explanation:**

Binary representation of the given number

10 is: 1 0 1 0, number of bits in the

binary reprsentation is 4. Thus 2nd bit

from right is 0. The number after changing

this bit to 1 is: 14(1 1 1 0).

**Example 2:**

**Input:**

N = 15

K = 3

**Output:**

15

**Explanation:**

The binary representation of the given

number 15 is: 1 1 1 1, number of bits

in the binary representation is 4. Thus

3rd bit from the right is 1. The number

after changing this bit to 1 is

15(1 1 1 1).

**Your Task:**   
You don't need to read input or print anything. Your task is to complete the function **setKthBit**() which takes two integer N and K as input parameter and returns an integer after setting the K'th bit in N.

**Expected Time Complexity:** O(1)  
**Expected Auxiliary Space:** O(1)

**Constraints:**  
1 <= N <= 109  
0 <= K < X, where X is the number of bits in the binary representation of N

CODE:

#include<bits/stdc++.h>

using namespace std;

// } Driver Code Ends

class Solution

{

public:

int setKthBit(int n, int k)

{

return (n= (n | 1<<k));

// Write Your Code here

}

};

// { Driver Code Starts.

int main()

{

int t;

cin >> t;

while (t--)

{

int N, K;

cin >> N >> K;

Solution ob;

int ans = ob.setKthBit(N,K);

cout << ans << endl;

}

return 0;

}

**6. Power of 2**

Given a positive integer **N**. The task is to check if N is a power of **2**. That is**N**is **2x**for some **x.**

**Input:**  
The first line contains T denoting the number of test cases. Each test case contains a single positive integer N.

**Output:**  
Print "**YES**" if it is a power of 2 else "**NO**" (Without the double quotes).

**Constraints:**  
1 <= T <= 100  
0 <= N <= 1018

**Example:**  
**Input :**  
2  
1  
98

**Output :**  
YES  
NO

**CODE:**

using namespace std;

#define ll long long

string check(ll n)

{

ll cnt=0;

int r=n&1;

while(n)

{

n=n&(n-1);

cnt++;

}

if(cnt==1)

return "YES";

return "NO";

}

int main()

{

ll t;

cin>>t;

while(t--)

{

ll n;

cin>>n;

cout<<check(n)<<endl;

}

return 0;

}

**7. Bit Difference**

You are given two numbers **A** and **B**. Write a program to **count number of bits needed to be flipped**to **convert**A to B.

**Input:**  
The first line of input contains an integer T denoting the number of test cases. T testcases follow. The first line of each test case is**A and B**separated by a space.

**Output:**  
For each testcase, in a new line, print the number of bits needed to be flipped.

**Constraints:**  
1 ≤ T ≤ 100  
1 ≤ A, B ≤ 103

**Example:**  
**Input:**  
1  
10 20  
**Output:**  
4

CODE:

using namespace std;

int fun(int n,int m)

{

int cnt=0;

while(n || m)

{

int r1=n%2;

int r2=m%2;

// cnt++;

if(r1!=r2)

cnt++;

n=n/2;

m=m/2;

}

return cnt;

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n,m;

cin>>n>>m;

cout<<fun(n,m)<<endl;

}

return 0;

}

**8. Rotate Bits**

Given an integer **N** and an integer **D**, you are required to write a program to **rotate the binary representation** of the integer **N by D**digits to the **left**as well as **right**and print the **results**in **decimal values** after each of the rotation.  
**Note**: Integer N is stored using **16 bits**. i.e. 12 will be stored as 0000.....001100.

**Input**:  
First line of input contains a single integer **T** which denotes the number of test cases. Each test case contains two space separated integers **N** and **D** where N denotes the number to be rotated and D denotes the number of digits by which the number is required to be rotated.

**Output:**  
For each testcase, in a new line, print the value of number N after rotating it to **left**by D digits in **one line,** and **second line** prints the value of number N after rotating it to the **right**by D digits.

**Constraints:**  
1 <= T <= 100  
1 <= N <  216  
1 <= D <= 105

**Example:  
Input:**  
2  
229 3  
28 2  
**Output:**  
1832  
40988  
112  
7

CODE:

using namespace std;

#define ll long long

unsigned short int left(int n,unsigned int d)

{

return ((n<<d)|(n>>(16-d)));

}

unsigned short int right(int n,unsigned int d)

{

return ((n>>d)|(n<<(16-d)));

}

int main()

{

int t;

cin>>t;

while(t--)

{

unsigned short int n,d;

cin>>n>>d;

d=d%16;

cout<<left(n,d)<<endl;

cout<<right(n,d)<<endl;

}

return 0;

}

**9. 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 right side(even position bits are highlighted in binary representation of 23), and every odd position bit is swapped with adjacent on left side.

**Input:**  
The first line of input contains **T**, denoting the number of testcases. Each testcase contains single line.

**Output:**  
For each testcase in new line, print the converted number.

**Constraints:**  
1 ≤ T ≤ 100  
1 ≤ N ≤ 100

**Example:**  
**Input**:  
2  
23  
2

**Output**:  
43  
1

**CODE:**

ll swapeo(ll n)

{

ll i=0;

ll ans=0;

while(n)

{

ll r1=n%2;

n=n/2;

ll r2=n%2;

swap(r1,r2);

ans+=r1\*pow(2,i);

i++;

ans+=r2\*pow(2,i);

i++;

n=n/2;

}

return ans;

}

int main()

{

ll t;

cin>>t;

while(t--)

{

ll n;

cin>>n;

cout<<swapeo(n)<<endl;

}

return 0;

}

**10. 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).

**Input:**  
The first line of input contains an integer T denoting the number of test cases. T testcases follow. The first line of each test case is**N**.

**Output:**  
For each testcase, in a new line, print the total count of all bits.

**Constraints:**  
1 ≤ T ≤ 100  
1 ≤ N ≤ 103

**Example:**  
**Input:**  
2  
4  
17  
**Output:**  
5  
35

CODE:

using namespace std;

#define ll long long

int main()

{

ll t;

cin>>t;

while(t--)

{

ll n;

cin>>n;

ll ans=0;

for(ll i=1;i<=n;i++)

ans+=\_\_builtin\_popcount(i);

cout<<ans<<endl;

}

return 0;

}

**11. Longest Consecutive 1's**

iven a number **N.**The task is to find the length of the longest consecutive 1s in its binary representation.

**Input:**  
The first line of input contains an integer **T**denoting the number of test cases. Then **T** test cases follow. Each test case contains an integer **N.**

**Output:**  
For each test case, in a new line, print the length of the longest consecutive 1's in N's binary representation.

**Constraints:**  
1 <= T < 100  
1 <= N <= 103

**Example:  
Input**:  
2  
14  
222

**Output**:  
3   
4

CODE:

using namespace std;

int len(int n)

{

int cnt=0;

int ans=0;

while(n)

{

int r=n%2;

if(r)

cnt++;

else

cnt=0;

ans=max(cnt,ans);

n=n/2;

}

return ans;

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

cout<<len(n)<<endl;

}

return 0;

}

**12. Number is sparse or not**

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

**Input:**  
The first line of input contains an integer **T** denoting the number of test cases. The first line of each test case is number 'N'.

**Output:**  
Print '**1**' if the number is sparse and '**0**' if the number is not sparse.

**Constraints:**  
1 <= T <= 100  
1 <= N <= 103

**Example:**  
**Input:**  
2  
2  
3

**Output:**  
1  
0

CODE:

using namespace std;

bool check(int n)

{

while(n)

{

int r1=n%2;

n=n/2;

int r2=n%2;

n=n/2;

// cout<<"n="<<n<<" r1="<<r1<<" r2="<<r2<<endl;

if(r1==1 && r1==r2)

return false;

n=n\*2+r2;

}

return true;

}

int main()

{

int t;

cin>>t;

while(t--)

{

int n;

cin>>n;

if(check(n))

cout<<1;

else

cout<<0;

cout<<endl;

}

return 0;

}

**13. Party of Couples**

represented by the same number. Find out the only singe person in the party of couples.

**Example 1:**

**Input:** N = 5

arr = {1, 2, 3, 2, 1}

**Output:** 3

**Explaination:** Only the number 3 is single.

**Example 2:**

**Input:** N = 11

arr = {1, 2, 3, 5, 3, 2, 1, 4, 5, 6, 6}

**Output:** 4

**Explaination:** 4 is the only single.

**Your Task:**  
You do not need to read input or print anything. Your task is to complete the function **findSingle()** which takes the size of the array N and the array arr[] as input parameters and returns the only single person.

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

**Constraints:**  
1 ≤ N ≤ 104  
1 ≤ arr[i] ≤ 106

CODE:

//Initial template for C++

#include<bits/stdc++.h>

using namespace std;

// } Driver Code Ends

//User function Template for C++

class Solution{

public:

int findSingle(int n, int arr[]){

int ans=0;

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

ans=ans^arr[i];

// code here

return ans;

}

};

// { Driver Code Starts.

int main()

{

int t;

cin>>t;

while(t--)

{

int N, X;

cin >> N;

int arr[N];

for(int i = 0; i < N; i++){

cin >> arr[i];

}

Solution ob;

cout << ob.findSingle(N, arr) << endl;

}

return 0;

}