***Finding GCD of two numbers***

***Euclidean Algo***

#include <iostream>

**using** **namespace** std;

// Recursive function to return gcd of a and b

**int** gcd(**int** a, **int** b)

{

    // Everything divides 0

**if** (a == 0)

**return** b;

**if** (b == 0)

**return** a;

    // base case

**if** (a == b)

**return** a;

    // a is greater

**if** (a > b)

**return** gcd(a-b, b);

**return** gcd(a, b-a);

}

// Driver program to test above function

**int** main()

{

**int** a = 98, b = 56;

    cout<<"GCD of "<<a<<" and "<<b<<" is "<<gcd(a, b);

**return** 0;

}

***Another Approach***

// C++ program to find GCD of two numbers

#include <iostream>

**using** **namespace** std;

// Recursive function to return gcd of a and b in single line

**int** gcd(**int** a, **int** b)

{

**return** b == 0 ? a : gcd(b, a % b);

}

// Driver program to test above function

**int** main()

{

**int** a = 98, b = 56;

    cout<<"GCD of "<<a<<" and "<<b<<" is "<<gcd(a, b);

**return** 0;

}

***Xor of 1 to n***

**int** computeXOR(**int** n)

{

**if** (n % 4 == 0)

**return** n;

**if** (n % 4 == 1)

**return** 1;

**if** (n % 4 == 2)

**return** n + 1;

**else**

**return** 0;

}

***Left shift --- << Equivalent to multiplying a number by 2 to the power i.***

***Right shift-- >> Equivalent to dividing a number by 2 to the power i.***

***Checking for ith set bit***

***We take a variable f equal to 1 and then left shift it I times so that ith index is set. Then we take the bitwise and of the number and variable. It will result in either 1 or 0 of that particular index only.***

***Counting number of set bits.***

***We will make use of right shift this time. If N & 1 is non zero then we increment the count variable and keep shifting once till the number becomes zero***

***Faster approach.***

***Int n;***

***Cin>>n;***

***Int cnt=0;***

***While(n>0){***

***Cnt++;***

***N=N&N-1;***

***}***

***Xor and properties…***

1. ***A^B***

***Identity element in xor is zero---- 0^A=A.***

1. ***A^A=0***
2. ***Ordering of xor values doesn’t matter.***

***Pair sum Xor of an array…***

***O(n2) approach is clear.***

***Other approach… on writing the matrix it is clear that except for the principal diagonal elements, other elements appear twice in the matrix. Therefore they will result in 0 and hence, the answer is the xor of twice of each element of the 1d array representing the diagonal elements only.***

***Total sum of Xor pairs…It means we have to take ai to an pairs only for each element and not starting from a number less than that.***

***Eg-***

***Int a[1001];***

***Int n,res=0;***

***Cin>>n;***

***For(int i=1;i<=n;i++){***

***Cin>>a[i];***

***}***

***For(int i=0;i<31;i++){***

***Int cntz=0,cnt1=0;***

***For(int j=1;j<=n;j++){***

***If(a[j]&(1<<i))***

***Cnt1++;***

***Else***

***Cntz++;***

***}***

***Int p=cntz\*cnt1;***

***Res+=(1<<i)\*p;***

***}***

***Cout<<res<<endl;***