BINARY EXPONENTIATION THROUGH RECURSION

The pow function in C++ is powerful, it has some limitations and inefficiencies in certain scenarios, which makes **binary exponentiation** a better choice in these cases. Also, pow uses floating-point arithmetic for non-integer powers, which can introduce small precision errors.

Binary exponentiation is a specific algorithm often used in competitive programming and other contexts where we need to compute powers in O(logN) time. It is really useful where the constraints are large.

BRUTE FORCE

```
int a, int b,sum=0;
cin>>a>>b;
//compute a^b
for(int i=1;i<=b;i++)
{sum*=a
}
```

uses O(n) time.

BINARY EXPONENTIATION

Dividing the power by two. Then multiplying with it through recursion. Like we want to compute the power 2^16.

Here 16 divided by 2 is 8. Compute 2^8.
Then 8 divided by 2 is 4. Compute 2^4.
Then 4 divided by 2 is 2. Compute 2^2.
Then 2 divided by 2 is 1. Compute 2^1.

Here comes the question. What if the power is odd? Then compute it like this,

2 divided by 2 is 1. Compute 2^1.
1 is odd. So,
1 is divided by 2 is 0.
So, computing 2^1 will be, 2*2^0.
An extra 2 will be multiplied.

And in the function, the base case will be returning 1 if the power is zero.

```
ll binaryExponen(ll a,ll b)
{
    if(b==0){
        return 1;
    }
    ll res = binaryExponen(a, b/2);
    if(b&1){
        return a*res*res;
    }
    else{
        return res*res;
    }
}
```

Note: This approach won't work correctly for decimal point values(float or doubles).