BASIC MATHS FOR DSA

1. Prime number

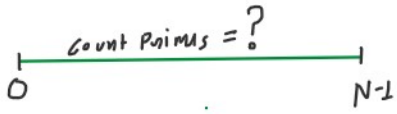
N is prime number when it has only two factor 1 and itself

PROGRAM 01: Count primes (Leetcode-204)

Example 1:
Input: N = 10
Output: 4
Explanation: There are 4 prime numbers less than 10, they are 2, 3, 5, 7.

Example 2:
Input: N = 0
Output: 0

Example 3:
Input: N = 1
Output: 0



APPROACH 01: Naive

N = 10

```cpp
// ✅APPROACH 01: Naive --> Total T.C. = O(N^2) and S.C. = O(1)
class Solution {
public:

    bool naiveAppIsPossible(int n){
        if(n<=1) return false;

        for(int i=2; i<n; i++){ // T.C. of inner loop is O(N)
            if(n%i == 0){
                return false;
            }
        }
        return true;
    }

    int countPrimes(int n) {
        int count = 0;

        for(int i=0; i < n; i++){ // T.C. of outer loop is O(N)
            if(naiveAppIsPossible(i)){
                count++;
            }
        }
        return count;
    }
    // T.C. of countPrimes() is O(N) * O(N) = O(N^2)
};
```

Count ~~0~~ ~~1~~ ~~2~~ ~~3~~ 4

i=0	isprime(i)	count
0	X	0
1	X	0
2	✓	1
3	✓	2
4	X	2
5	✓	3
6	X	3
7	✓	4
8, 9, 10	X	4 → output

APPROACH 02: SQRT

N = 10 and output = 4

```cpp
// ✅APPROACH 02: SQRT --> T.C. = O(N√N) and S.C. = O(1)
class Solution {
public:

    bool sqrtAppIsPossible(int n){
        if(n<=1) return false;

        int sqrtN = sqrt(n); // T.C. of sqrt() is O(logN)

        for(int i=2; i<=sqrtN; i++){ // T.C. of inner loop is O(√N)
            if(n%i == 0){
                return false;
            }
        }
        return true;
        // T.C. of sqrtAppIsPossible() is O(logN) + O(√N) = O(√N)
    }

    int countPrimes(int n) {
        int count = 0;

        for(int i=0; i < n; i++){ // T.C. of outer loop is O(N)
            if(sqrtAppIsPossible(i)){
                count++;
            }
```

We can optimize the **inner loop** using these step
Step 01: find the square root of Nth value, and check is prime or not
I II

Count ~~0~~ ~~1~~ ~~2~~ ~~3~~ 4

N के factors ढूंढो -

N=0	N<=1	√N	i=2	i<=√N	P/NP	count
0,1	✓	—	—	—	NP	0
2	X	1	2	X	P	1
3	X	1	2	X	P	2
4	X	2	2	✓	NP	2
—	X	2	2	✓	—	—

```
        int count = 0;

        for(int i=0; i < n; i++){ // T.C. of outer loop is O(N)
            if(sqrtAppIsPossible(i)){
                count++;
            }
        }
        return count;
    }
    // T.C. of countPrimes is O(√N) * O(N) = O(N√N)
};
```

3	X	1	2	X	P	—
4	X	2	2	✓	NP	2
5	X	2	2	✓	—	—
			3	X	P	3
6	X	2	2	✓	NP	3
7	X	2	2	✓	—	—
			3	X	P	4
8	X	2	2	✓	NP	4
9	X	3	2	✓	—	—
			3	✓	NP	4
10	X	3	2	✓	NP	4

MAIN DRY RUN SE OBSERVE KARPA RHA HUN

JAB N NON-PRIME NUMBER HOTA HAI TO [2, √N] KE B/W AT LEAST 1 OR MORE FACTOR MILTE HAI. LIKE-

Inner loop

NON-PRIME(N)

- N = 4
- N = 6
- N = 8
- N = 9
- N = 10

Factors b/w [2, √N]

2 (1 factor)	[2, 2]
2 (1 factor)	[2, 2]
2 (1 factor)	[2, 2]
3 (1 factor)	[2, 3]
2 (1 factor)	[2, 3]

APPROACH 03: Sieve of Eratoshenes N = 10 output = 4

Step 01: Assume that all number are prime

Step 02: Find prime number and remove the multiple of prime number until N-1 size

STEP:01

0	1	2	3	4	5	6	7	8	9	10
T	T	T	T	T	T	T	T	T	T	T

i=2

STEP:02

i
Prime
2, 3, 5, 7 } MARK TRUE

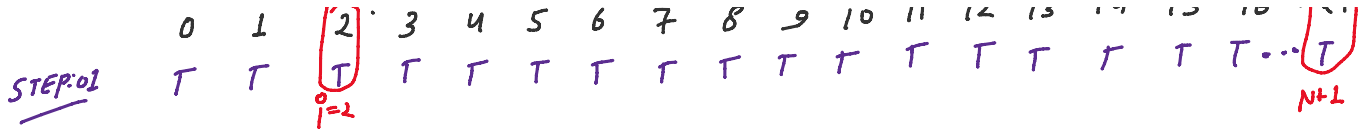
j = 2 * i
Non-prime
4, 6, 8, 10
6, 9
10
} MARK FALSE

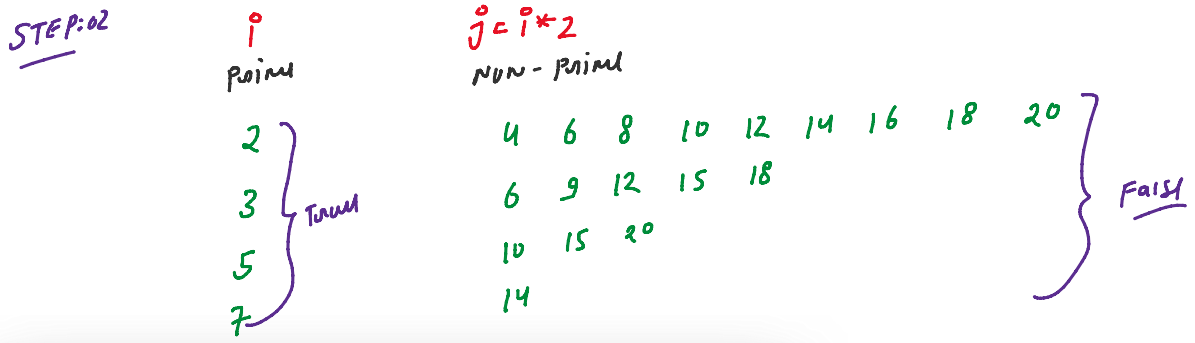
EX:2 N = 21

STEP:01

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16 ..	21
T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T	T ...	T

i=2 (0) N+1






```cpp
// ✅APPROACH 03: Sieve of Eratoshenes --> T.C. = O(N * (log(log N))) and S.C. = O(N)
class Solution {
public:
    int countPrimes(int n) {
        if(n == 0) return 0;

        vector<bool> prime(n+1,true); // Step 01: all are prime marked already (true)
        prime[0]= prime[1]= false;

        int count = 0;

        for(int i=2; i < n; i++){ // T.C. of outer loop is O(N)

            // Step 02
            if(prime[i]){
                count++;
                for(int j=2*i; j<n; j+=i){ // T.C of inner loop O(log(log N))
                    prime[j] = false;
                }
            }
        }
        return count;
    }
    // T.C. of countPrimes is O(N) *  O(log(log N)) = O(N * (log(log N)))
}
```

Time Complexity

$\Rightarrow \left(\frac{n}{2} + \frac{n}{3} + \frac{n}{4} + \cdots + \cdots\right)$

$\Rightarrow n\left[\frac{1}{2} + \frac{1}{3} + \frac{1}{4} + \cdots + \cdots\right]$

↳ H.P. of Prime

$\Rightarrow O(n[\log(\log N)])$

Taylor series ans ⇒ Sinx tanx cosx etc

2. Find GCD/HCF using Euclids Algorithm (GFG)

A	B	GCD / HCF	LCM
24	72	24	72
5	7	1	35
3	6	3	6
4	8	4	8

FIND GCD using Euclid's Algorithm

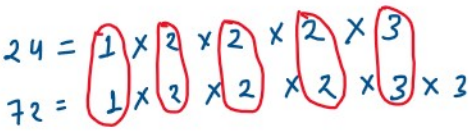
$GCD(A,B) = GCD(A-B, B)$ OR $= GCD(A\%B, B)$ — when $A > B$

$GCD(B-A, A)$ OR $GCD(B\%A, A)$ — when $A < B$

EX1 $A = 24, B = 72$

$24 = 1 \times 2 \times 2 \times 2 \times 3$

$72 = 1 \times 2 \times 2 \times 2 \times 3 \times 3$



$$GCD = 1 \times 2 \times 2 \times 2 \times 3$$
$$= 24$$

Using Euclid's Algorithm

{ Apply this till one of the parameter become Zero }

$$= GCD(B-A, A), \text{ when } A < B$$
$$= GCD(72-24, 24)$$
$$= GCD(48, 24)$$
$$= GCD(24, 24)$$
$$= GCD(0, 24)$$

$GCD(24, 72) = 24$

```cpp
// PROGRAM 02: Find GCD/HCF using Euclids Algorithm (GFG)

class Solution
{
    public:
    // GCD of two positive integer numbers
    int gcd(int A, int B)
    {
        if(A==0) return B;
        if(B==0) return A;

        while(A != B){
            if(A>B){
                A = A - B;
            }
            else{
                B = B - A;
            }
        }

    }
};

/*
Time Complexity: O(A) when A<B or O(B) when A>B
Space Complexity: O(1), no extra space used
*/
```

3. Find LCM (GFG)

```cpp
// PROGRAM 03: Find LCM (GFG)
#include <iostream>
using namespace std;

// GCD of two positive integer numbers
int gcd(int A, int B)
{
    if(A==0) return B;
    if(B==0) return A;

    while(A != B){
       if(A>B){
           A = A - B;
       }
       else{
           B = B - A;
       }
    }
}

int main() {
    int A = 4, B = 8;
    int lcm = (A*B)/gcd(A,B);
    cout<<lcm<<endl;
    return 0;
}
```

$$LCM * GCD = A * B$$

$$LCM = \frac{A * B}{GCD}$$

4. MODULO ARITHMATIC

Ex $5\%4 \Rightarrow [0,1,2,3]$

$5\%3 \Rightarrow [0,1,2]$

What is Modulo Arithmetic?
Generally, to avoid the overflow *while* storing integer we *do* modulo with a large number.

Range: (a % b) -> [0 to b-1]

How modulo is used: A few distributive properties of modulo are as follows:

1. (a + b) % M = ((a % M) + (b % M)) % M
2. (a * b) % M = ((a % M) * (b % M)) % M
3. (a - b) % M = ((a % M) - (b % M)) % M

So, modulo is distributive over +, * and - but not over /.

NOTE: The result of (a % b) will always be less than b.

In the *case* of computer programs, due to the size of variable limitations, we perform modulo M at each intermediate stage so that range overflow never occurs.

5. Fast exponentiation (GFG)

$A^B = ?$

$2^{10} = 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2 \times 2$

$= 1024$

APPROACH 01: Naive solution

i = 0	i < B	Ans = Ans * A
0	✓	= 1 * 2 ⇒ 2
1	✓	= 2 * 2 ⇒ 4
2	✓	= 4 * 2 ⇒ 8
3	✓	= 8 * 2 ⇒ 16
4	✓	= 16 * 2 ⇒ 32
5	✓	= 32 * 2 ⇒ 64
		= 64 * 2 ⇒ 128

4 ✓
5 ✓
6 ✓
7 ✓
8 ✓
9 ✓
10 ✗ END

= 32 * 2 ⇒ 64
= 64 * 2 ⇒ 128
= 128 * 2 ⇒ 256
= 256 * 2 ⇒ 512
= 512 * 2 ⇒ 1024 Output

```cpp
// PROGRAM 04: Fast exponentiation (GFG)
#include<iostream>
using namespace std;

// APPROACH 01: Naive solution
int slowExponentiation(int A, int B){
    int ans = 1;
    for(int i=0; i<B; i++){
        ans *= A;
    }

    return ans;
}

int main(){
    int A, B;
    cin>>A>>B;

    int ans = slowExponentiation(A,B);

    cout<<ans<<endl;

    return 0;
}

/*
Time Complexity: O(B), where B is a exponentiation power
Space Complexity: O(1)
*/
```

APPROACH 02: Better solution

Find = A^B

when A Even → $(A^{B/2})^2$

Ex: $= 2^{10}$
$= (2^5)^2$

when B odd → $(A^{B/2})^2 * A$

$= 2^{11}$
$= (2^5)^2 * 2^1$

Ex 2^5

$= (2^4) * 2$
$= (2^2 * 2^2) * 2$
$= ((2^1 * 2^1) * (2^1 * 2^1)) * 2$

DRY RUN A = 5 and B = 4 O/P = 625

① B = 4
A = A * A = 5 × 5 = 25
B = B/2 = 4/2 = 2

② B = 2
A = 25 × 25 = 625
B = 2/2 = 1

5^4 ↓ $(5^2 * 5^2)$

② B=2 $A = 25 \times 25 = 625$ $B = 2/2 = 1$

③ B = 1 $Ans = Ans * A = 1 * 625 = 625$ $B = 1/2 = 0$

output

④ B=0 ✗ END

$(5^2 * 5^2)$

$(5^1 * 5^1) * (5^1 * 5^1)$

DRY RUN A = 2 and B = 5 O/P = 32

① B=5 $Ans = Ans * 2 = 1 * 2 = 2$ $B = B/2 = 5/2 = 2$

$A = A * A = 2 * 2 = 4$

② B = 2 $A = 4 \times 4 = 16$ $B = 2/2 = 1$

③ B = 1 $Ans = 2 * 16 = 32$ $B = 1/2 = 0$ END

$A = A * A = 16 * 16 = 256$

```cpp
// PROGRAM 04: Fast exponentiation (GFG)
#include<iostream>
using namespace std;

// ✅APPROACH 02: Better solution
int fastExponentiation(int A, int B){
    int ans = 1;

    while(B > 0){

        if(B & 1){ // When B is odd number
            ans = ans * A;
        }

        A = A * A;
        B = B>>1; // B is divided by 2 until B>0
    }

    return ans;
}

int main(){
    int A, B;
    cin>>A>>B;

    int ans = fastExponentiation(A,B);

    cout<<ans<<endl;

    return 0;
}
```

```
    cout<<ans<<endl;

    return 0;
}
/*
✅APPROACH 02: Better solution
Time Complexity: O(Log B), where B is a exponentiation power
Space Complexity: O(1)
*/
```

PROGRAM 05: Modular Exponentiation for large numbers (GFG)

```
// PROGRAM 05: Modular Exponentiation for large numbers (GFG)
#include<bits/stdc++.h>
using namespace std;

class Solution
{
    public:
        long long int PowMod(long long int x,long long int n,long long int M)
        {
            long long int ans = 1;

            while(n > 0){
                if(n & 1){
                    // n is odd
                    ans = (ans * x) % M;
                }

                x = (x*x) % M;
                n = n>>1;
            }
            return ans % M;
        }
};

int main(){
    int T;
    cin >> T;
    while(T--)
    {
        long long int x, n, m;
        cin >> x >> n >> m;
        Solution ob;
        long long int ans = ob.PowMod(x, n, m);
        cout << ans <<"\n";
    }
    return 0;
}
```