# Lecture 2
# Math & Number Theory - 1

Anik Sarker[1]

Postgraduate Student,
Department of CSE, BUET
ECE Building, West Palasi, Dhaka-1205, Bangladesh

**Abstract.** This lecture is a part of competitive programming training lectures prepared for Eastern University, Dhaka. This lecture introduces some Math and Number Theory based concepts : Binary Exponentiation, Divisibility, Harmonic Number Properties, GCD & LCM, Primality Testing, Sieve of Eratosthenes, Prime Factorization, Totient Function, Divisor Sum Function

## 1 Binary Exponentiation

1. We can calculate $a^n \mod m$ in $O(n)$. Easy right?
2. We can actually do it in $O(log\ n)$

$$a^n\ mod\ m = \begin{cases} 1 & \text{if } i = 0 \\ a^{\frac{n}{2}} * a^{\frac{n}{2}} & \text{if i > 0 and i mod 2 = 0} \\ a^{\frac{n}{2}} * a^{\frac{n}{2}} * a & \text{if i > 0 and i mod 2 = 1} \end{cases} \tag{1}$$

```
#define ll long long int
// returns a^n mod m
// Complexity : O(n)
ll BrutePowerCalc(ll a, ll n, ll mod){
    ll cur = 1;
    for(int i=1 ; i<=n; i++){
        cur = ((cur % mod) * (a % mod)) % mod;
    }
    return cur;
}
```

```
// Complexity : O(log_2 n)
ll FastPowerCalc(ll a, ll n, ll mod){
    if(n == 0) return 1;
    ll ret = FastPowerCalc(a, n/2, mod);
    ret = (ret * ret) % mod;

    if(n % 2 == 1) ret = (ret * a) % mod;
    return ret;
}
```

## 2 Divisibility

1. a mod b = a - b * $\lfloor \frac{a}{b} \rfloor$
2. (a * b) mod m = ((a mod m) * (b mod m)) mod m
3. (10 * a + b) mod m = ((10 * a) mod m + b mod m) mod m

```
// returns a biginteger s modulo m, where s >= 0, m > 0
// Complexity : O(|s|)
ll StringMod(string s, ll mod){
    ll curMod = 0;
    for(int i=0; i<s.size(); i++){
        int digit = s[i] - '0';
        curMod = ((curMod * 10) % mod + digit % mod) % mod;
    }
    return curMod;
}
```

## 3 Harmonic Number Property 1

* $\frac{n}{1} + \frac{n}{2} + \frac{n}{3} + \ldots + \frac{n}{n} \leq O(n\ logn)$

## 4 GCD and LCM

1. **Euclid's algorithm:**

$$gcd(a,b) = \begin{cases} a & \text{if } b = 0 \\ gcd(b, a\ mod\ b) & \text{otherwise} \end{cases} \quad (2)$$

   Complexity : $O(log\ (min(a,b)))$
   No worries, use STL __gcd function.

```
// returns GCD and LCM of 2 numbers
ll GCD(ll a, ll b) {return __gcd(a,b);}
ll LCM(ll a, ll b) {return (a / __gcd(a,b)) * b;}
```

## 5 Primality Testing

* Check a single number if it is prime
* $O(n)$ - check for each number in 2 to n-1
* $O(\sqrt(n)))$ - check for each number in 2 to $\sqrt(n)$
* $O(log^3(n)))$ - Miller-Rabin Primality Testing, **Too Hard !**

```
// returns if a number is prime or not
// Complexity : O(n)
bool PrimeTestSlow(ll n){
    for(ll i=2; i<n; i++){
        if(n % i == 0) return false;
    }
    return true;
}

// Complexity : O(sqrt(n))
bool PrimeTestFast(ll n){
    for(ll i=2; i*i <= n; i++){
        if(n % i == 0) return false;
    }
    return true;
}
```

## 6 Prime Factorization and Functions

1. Any number n can be uniquely expressed as n $= p_1^{d_1} * p_2^{d_2} * \ldots p_k^{d_k}$
2. Prime factorization can be done in $O(\sqrt{(}n)))$

```
// returns prime factors and their powers in a vector of pairs
// Complexity : O(n)
#define pll pair<ll, ll>
#define vll vector<pll>

vll PrimeFactorizationSlow(ll n){
    vll res;
    for(ll i=2; i<=n; i++){
        ll d = 0;
        while(n % i == 0) {n = n / i; d++;}
        if(d > 0) res.push_back({i,d});
    }
    return res;
}
```

```
// Complexity : O(sqrt(n))
vll PrimeFactorizationSlow(ll n){
    vll res;
    for(ll i=2; i*i <=n; i++){
        ll d = 0;
        while(n % i == 0) {n = n / i; d++;}
        if(d > 0) res.push_back({i,d});
    }
    if(n > 1) res.push_back({n,1});
    return res;
}
```

– **Totient / Phi Function :**
  Count of numbers less than n and coprime to n.
  $\phi(n) = n * (1 - \frac{1}{p_1}) * (1 - \frac{1}{p_2}) * \ldots * (1 - \frac{1}{p_k})$

– **Divisor Count Function :**
  Count of numbers which divides n.
  $d(n) = (d_1 + 1) * (d_2 + 1) * \ldots * (d_k + 1)$

– **Divisor Sum Function :**
  Sum of numbers which divides n.
  $\sigma(n) = \frac{p_1^{d_1+1}-1}{p_1-1} * \frac{p_2^{d_2+1}-1}{p_2-1} * \ldots * \frac{p_k^{d_k+1}-1}{p_k-1}$

## 7 Sieve of Eratosthenes

1. Check for all the numbers in range [1, n] - if it is prime
2. Check for each number independently? - $O(n\sqrt(n))$
3. We can do better.
4. Iterate x from 2 to n and mark the multiples of x as composite.
5. Complexity : O(n log n), can be optimized to $O(n\ log\ log\ n)$ and $O(n)$

```
// returns all primes in range [1,n]
// Complexity : O(n log n)
const int MAXN = 1000005;
bool mark[MAXN];
vector<ll> Primes;

void SieveOfEratosthenes(ll n){
    for(int i=2; i<=n; i++){
        if(mark[i] == false) Primes.push_back(i);
        for(int j = 2*i; j<=n; j+=i) mark[j] = true;
    }
}
```