Number Theory

- **gcd():** Return a and b gcd(**Greatest Common Divisor**) value. => O(logn)
  [int gcd**= __gcd(a,b);** gcd(m*a, m*b) = m*gcd(a, b)**;** gcd(a/d, b/d) = gcd(a, b)/d**;**]
- **lcm()**: Return a and b lcm(**Least Common Multiple**) value. => O(logn)
  [int lcm**= (a*b)/__gcd(a,b);** lcm(m*a, m*b) = m*lcm(a, b)**;**]
  - lcm(a, b, c) = $\frac{\boldsymbol{abc * gcd(a,b,c)}}{\boldsymbol{gcd(a,b) * gcd(a,c) * gcd(b,c)}}$.

Extended Euclid: //=> **O(log(min(a, b)))**

```
// For this Eq. (a*x) + (b*y) = gcd(a, b);
ll extended_euclid(ll a, ll b, ll &x, ll &y) {
   if (b == 0) {
      x = 1, y = 0;
      return a;
   }
   ll x1, y1;
   ll gcd = extended_euclid(b, a % b, x1, y1);
   x = y1,  y = x1 - y1 * (a / b);
   return gcd;
}
```

Sum and Count of Divisor: **=>O(sqrt(n))**

Ex(sum): 20 => 22 (1+2+4+5+10+20).

Ex(count): 20 => 6 (1,2,4,5,10,20).

```
 void divisor()    {
   ll n, sum = 0, i, c = 0;
   cin >> n;
   vector<ll> divisors;
   for (i = 1; i * i <= n; i++) {
      if (n % i == 0) {
         sum += i, ++c;
         divisors.push_back(i);
         if (i != n / i)
            sum += n / i, ++c, divisors.push_back(n / i);
      }
   }
   cout<<"Count = "<< c <<", Sum = "<< sum <<endl;
   sort(divisors.begin(), divisors.end());
   for(auto &i: divisors) cout << i << " ";
}

int numberOfDivisors(LL n) {
   int sz = primes.size(), cnt = 1;

   for(int i = 0; i < sz && primes[i] * primes[i] <= n; ++i) {
      if(n % primes[i] == 0) {
         int pw = 0;
         while(n % primes[i] == 0) {
            ++pw;
            n /= primes[i];
         }
         cnt = pw + 1;
      }
   }
   if(n != 1) cnt <<= 1;
   return cnt;
}
```

Number of divisors: **=> O(nlog(n))**

Ex: 32 => 2 4 8 16 32

```
const int N = 1e5 + 10;
vector<int> divisor[N];
int main() {
   for (int i = 2; i < N; i++) {
      for (int j = i; j < N; j += i)
         divisor[j].push_back(i);
   }
   int n;  cin >> n;
   for (auto &it : divisor[n]) cout << it << " ";
   cout << endl;
}
```

Bitwise Sieve Algorithm (find prime number):
=> O(nloglogn)

```
const int N = 1e8 + 7;
int marked[N / 64 + 2];

#define on(x) (marked[x / 64] & (1 << ((x % 64) / 2)))
#define mark(x) marked[x / 64] |= (1 << ((x % 64) / 2))

void sieve() {
   for (int i = 3; i * i < N; i += 2) {
      if (!on(i)) {
         for (int j = i * i; j <= N; j += i + i) {
            mark(j);
         }
      }
   }
}
bool isPrime(int num) {
   return num > 1 && (num == 2 || ((num & 1) && !on(num)));
}
```

Sieve Algorithm (find prime number):
=> O(nloglogn)

```
const int N = 1e7 + 10;
bool marked[N];
```