Cheat Sheet (Number Theory)

Max or Min

```
int max(int a, int b) { return a>b ? a:b; }
int min(int a, int b) { return a<b ? a:b; }
```

GCD

```
int gcd(int a,int b) {return (b==0) ? a : gcd(b,a%b); }
```

LCM

```
int lcm(int a, int b) { return a/gcd(a,b)*b; }
```

Extended Euclidean

```
int gcd(int a,int b,int &x,int &y) {
    if (b == 0) {
        x = 1; y = 0; return a; }
    int x1, y1;
    int d = gcd(b, a % b, x1, y1);
    x = y1; y = x1 - y1 * (a / b); return d;
}
```

If Prime

```
bool prime(int n) {
    if (n<2) return false;
    if (n<=3) return true;
    if (!(n%2) || !(n%3)) return false;
    for (int i=5;i*i<=n;i+=6)
        if (!(n%i) || !(n%(i+2))) return false;
    return true;
}
```

b^p

```
int pw(int b, int p) {
    int r = 1;
    while(p) {
        if (p & 1) r *= b;
        p >>= 1; b *= b;
    }
    return r;
}
```

$a^b \pmod p$

```
int powmod(int a,int b,int p){
    int r=1; a %= p;
    while(b){
        if(b&1) r = (r*a)%p;
        b>>=1; a = (a*a)%p;
    }
    return r;
}
```

Inverse Modulo

```
// Require Extended Euclidean
int modInv(int a,int m){
    int x, y, g = gcd(a, m, x, y);
    if (g != 1) return -1;
    else return (x%m + m) % m;
}
```

Sieve

```
// primes contains all Prime numbers
// isprime[i] represents smallest prime number which
divides the number i
const int mx = 1<<20;
int isprime[mx];
vector<int> primes;
void sieve() {
    for(int i=2 ; i < mx ; ++i) {
        if(isprime[i]>0) continue;
        isprime[i] = 1; int j=i*i;
        while(j< mx) {
            if(isprime[j]==0) isprime[j]=i;
            j+=i;
        }
        primes.push_back(i);
    }
}
```

*Some functions are dependent on others

Register on codedigger.tech for Practice Competitive Problems.

Subscribe to YouTube Channel – codealittle for learning more stuff.

Number of Divisors

```
//Require Sieve
int numDiv(int n){
    int pf_idx = 0, pf = primes[pf_idx], ans = 1;
    while(pf * pf <= n){
        int power = 0;
        while(n % pf == 0) { n /= pf; power++; }
        ans *= (power+1);
        pf = primes[++pf_idx];
    }
    if(n != 1) ans*=2;
    return ans;
}
```

Sum of Divisors

```
// Require Sieve and pw
int sumDiv(int n){
    int pf_idx = 0, pf = primes[pf_idx], ans = 1;
    while(pf * pf <= n){
        int power = 0;
        while(n % pf == 0) { n /= pf; power++; }
        ans *= (pw(pf, power + 1) - 1)/(pf-1);
        pf = primes[++pf_idx];
    }
    if(n != 1) ans *= (pw(n, 2) - 1)/(n-1);
    return ans;
}
```

Factor of N

```
set<int> factor(int n) {
    set<int> fact;
    for(int i = 1 ; i*i <= n ; ++i)
        if(n%i == 0)
            fact.insert(i) , fact.insert(n/i);
    return fact;
}
```

Pre-Calculation of Factorial

```
// Require Inverse Modulo
const int mx = 1<<20;
int fact[mx] , ifact[mx];
void preCalcFac(int m){
    fact[0] = 1;
    for(int i = 1; i < mx; ++i)
        fact[i] = (fact[i - 1] * i) % m;
    ifact[mx - 1] = modInv(fact[mx - 1], m);
    for(int i = mx-2; i >= 0 ; --i)
        ifact[i] = (ifact[i + 1] * (i + 1)) % m;
}
```

nCr

```
int nCr(int n,int r){
    int res = n;
    for(int i = 2 ; i<=r ; ++i){
        res*=(n-i+1); res/=i;
    }
    return res;
}
```

$nCr \pmod m$

```
// Require Pre-Calculation of Factorial
int nCr(int n,int r,int m) {
   if (r==0) return 1;
   int ans = (fact[n]*ifact[r])%m;
   ans = (ans*ifact[n-r])%m;
   return ans;
}
```

Euler's Totient Function

```
// the positive integers less than or equal to n that
are relatively prime to n.
int phi (int n){
    int result = n;
    for (int i=2; i*i<=n; ++i)
        if(n%i==0){
            while(n%i==0) n /= i;
            result -= result / i;
        }
    if (n > 1) result -= result / n;
    return result;
}
```

Made with Love by Shivam Singhal