Contents

**1. Number Theory**

**a. Sieve**

```
#define LIM 10000005
#define pb push_back
bool status[LIM]; /// NEED A GLOBAL
ARRAY
vector <int> prime;
void make_primes(int N)
{
    int i, j, sqrtN;
    status[0] = 1, status[1] = 1;
    sqrtN = int( sqrt((double) N )) + 1;
/// have to check primes up to (sqrt(N))
    for(i = 4; i <= N; i+=2) status[i] =
1;
    for(i = 3; i <= sqrtN; i+= 2){  ///
so, i is a prime, so, discard all the
multiples
        if(status[i] == 0){
            for(j = i*i; j <= N; j+=i+i)
/// j = i * i, because it's the first
number to be colored
                status[j] = 1;
        }
    }
    for(i = 0; i < LIM; i++)
if(status[i] == 0) prime.pb(i);
    //for(i = 0; i < 10000005; i++) cout
<< v[i] << endl;
}
```

**b. Segmented Sieve**

```
void sg_sieve(ll m, ll n)
{
    ll i;
    memset(status2,0,sizeof status2);
    for(i = 0; v[i]*v[i] <= n; i++){
        ll tem = (m+v[i]-1)/v[i];
        ll sg = tem*v[i];
        if(sg == v[i]) sg += v[i];
        for(ll st = sg; st <= n; st +=
v[i]){
            status2[st-m+1] = 1;
        }
    }
    for(i = m ; i <= n; i++){
        if(status2[i-m+1] == 0 and i !=
1) cout << i << endl;
    }
    cout << endl;
}
```

**c. Bitwise sieve 64bit**

```
#define ll long long
#define mx 100000000
#define one (1LL)
vector<ll> prime;
ll status[(mx / 64) + 2];

bool checkprime(ll n) {
    if(status[n / 64] & (one << (n %
64))) return false;
    return true;
}

void setbit(ll n) {
    status[n / 64] = status[n / 64] |
(one << (n % 64));
}

void mkprime(ll n) {
    setbit(0);
    setbit(1);

    prime.push_back(2);
    ll sqrtn = sqrt(n);
    for(ll i = 3; i <= sqrtn; i += 2) {
        if(checkprime(i)) {
            for(ll j = i * i; j <= n; j
+= i + i) {
                setbit(j);
            }
        }
    }

    for(ll i = 3; i <= n; i += 2) {
        if(checkprime(i))
prime.push_back(i);
    }
}
```

**d. Prime factorization**

```
#define ll long long  /// MACRO NEEDED
#define LIM 10005     /// MACRO NEEDED
#define mod 100000007 /// MACRO NEEDED
#define pb push_back  /// MACRO NEEDED
#define ff first      /// MACRO NEEDED
#define ss second     /// MACRO NEEDED

bool status[LIM]; /// NEED A GLOBAL
ARRAY
vector <int> prime;
```

```
vector <pair<int,int> > prime_pow;

void make_prime(int N)
{
    int i, j, sqrtN;
    status[0] = 1, status[1] = 1;
    sqrtN = int( sqrt((double) N )) + 1;
/// have to check primes up to (sqrt(N))
    for(i = 4; i <= N; i+=2) status[i] =
1;
    for(i = 3; i <= sqrtN; i+= 2){  ///
so, i is a prime, so, discard all the
multiples
        if(status[i] == 0){
            for(j = i*i; j <= N; j+=i+i)
/// j = i * i, because it's the first
number to be colored
                status[j] = 1;
        }
    }
    for(i = 0; i < LIM; i++)
if(status[i] == 0) prime.pb(i);
    //for(i = 0; i < 10000005; i++) cout
<< v[i] << endl;
}
void factor(int n)
{
    for(int i = 0; i < prime.size() &&
prime[i]*prime[i] <= n; i++){
        int cnt = 0;
        while(n%prime[i] == 0){
            n /= prime[i];
            cnt++;
        }
        if(cnt) prime_pow.pb({prime[i],
cnt});
    }
    if(n != 1) prime_pow.pb({n, 1});
}
make_prime() /// NEED TO CALL THE
FUNCTION FROM MAIN
factor()     /// NEED TO CALL THE
FUNCTION FROM MAIN
```

e. **Number of Divisor Sieve Method(NOD)**

```
bool status[LIM+5];
ll nod[LIM+5];
void NOD()
{
    ll i, j, tem, cnt;
    for(i = 0; i <= LIM; i++)
        nod[i] = 1;
    for(i = 4; i <= LIM; i+=2){
        status[i] = 1;
        tem = i;
        cnt = 0;
        while(tem%2 == 0){
            tem /= 2;
            cnt++;
        }
        nod[i] *= (cnt+1);
    }
    for(i = 3; i <= LIM; i+=2){
        if(status[i] == 0){
            for(j = i+i; j <= LIM; j +=
i){
                status[j] = 1;
                tem = j;
                cnt = 0;
                while(tem%i == 0){
                    tem /= i;
                    cnt++;
                }
                nod[j] *= (cnt+1);
            }
        }
    }
    nod[0] = 0, nod[1] = 1;
}
```

f. **Sum of NOD**

**Problem :** Find the snod of n = 100000 ?
nod(1) + nod(2) + nod(3) + … + nod(n) = ??

```
//code of O(n) solution
int snod( int n ) {
    int res = 0;
    for ( int i = 1; i <= n; i++ ) {
        res += n / i;
    }
    return res;
}
//code of O(sqrt(n)) solution
int snod( int n ) {
    int res = 0;
    int u = sqrt(n);
    for ( int i = 1; i <= u; i++ ) {
        res += ( n / i ) - i; //Step 1
    }
    res *= 2; //Step 2
    res += u; //Step 3
```

```
    return res;
}
```

### g. SOD (Sum of Divisor)

**Problem** : Find the sod of n?

sod(12) = 1 + 2 + 3 + 4 + 6 + 12 = 28;

if $N = p_1^{a_1} \times p_2^{a_2} \times \ldots p_k^{a_k}$, then

$$SOD(N) = (p_1^0 + p_1^1 + p_1^2 \ldots p_1^{a_1}) \times (p_2^0 + p_2^1 + p_2^2 \ldots p_2^{a_2}) \times \ldots (p_k^0 + p_k^1 + p_k^2 \ldots p$$

```
//code of O(sqrt(n)) solution
int sod(int n){
    int sqrtn = sqrt(n);
    int res = 0;
    for( int i = 1; i <= sqrtn; i++){
        if(n % i == 0){
            res += i; //"i" is a divisor
            res += n / i; //"n/i" is
also a divisor
        }
    }
    if(sqrtn * sqrtn == n) res -= sqrtn;
//same number counted two times on first
loop
    return res;
}
//code of sod using prime factorization
int sod(int n){
    int res = 1;
    int sqrtn = sqrt(n);
    for(int i = 0; i < prime.size() &&
prime[i] <= sqrtn; i++){
        if(n % prime[i] == 0){
            int tempSum = 1; // Contains
value of (p^0+p^1+...p^a)
            int p = 1;
            while(n % prime[i] == 0){
                n /= prime[i];
                p *= prime[i];
                tempSum += p;
            }
            sqrtn = sqrt(n);
            res *= tempSum;
        }
    }
    if(n != 1){
        res *= (n + 1); // Need to
multiply (p^0+p^1)
    }
    return res;
}
```

### h. GCD Extreme

```
#include <bits/stdc++.h>
#define ll long long
#define pb push_back
#define LIM 200001
using namespace std;
ll arr[LIM], ans[LIM];
vector <ll> v[LIM], prime;
void make_prime()
{
    ll i, j;
    for(i = 2; i < LIM; i++){
        ll f = 1;
        for(j = 2; j*j <= i; j++)
            if(i%j == 0) f = 0;
        if(f) prime.pb(i);
    }

}
void ETF()
{
    ll i, j;
    for(i = 0; i < LIM; i++)
        arr[i] = i;
    for(i = 0; i < prime.size(); i++){
        for(j = prime[i]; j < LIM; j +=
prime[i])
            arr[j] =
(arr[j]/prime[i])*(prime[i]-1);
    }

}
void divi()
{
    ll i, j;
    for(i = 2; i < LIM; i++){
        for(j = i+i; j < LIM; j += i){
            v[j].pb(i);
        }
    }
}
void GCD()
{
    ll i, j;
    for(i = 1; i < LIM; i++){
        //cout << i << endl;
        for(j = 0; j < v[i].size();
j++){
            ans[i] +=
v[i][j]*arr[i/v[i][j]];
```

```
            //cout << v[i][j] << " " <<
i/v[i][j] << " " << arr[i/v[i][j]] <<
endl;
        }
        ans[i] += arr[i];
    }
    for(i = 2; i < LIM; i++)
        ans[i] += ans[i-1];
}
int main()
{
    make_prime();
    ETF();
    divi();
    GCD();
    ll n;
    while(cin >> n){
        if(!n) break;
        cout << ans[n]-1 << endl;
    }
    return 0;
}
```

i. **LCM Extreme**

```
#include <bits/stdc++.h>
#define ull unsigned long long
#define pb push_back
#define LIM 3000005
using namespace std;
ull phi[LIM], ans[LIM];
void divi()
{
    ull i, j;

    for(i = 2; i < LIM; i++){
        for(j = i; j < LIM; j += i){
            ans[j] += j*((i*phi[i])/2);
        }
    }
}
void ETF()
{
    ull i, j;

    for(i = 0; i < LIM; i++)
        phi[i] = i;

    for(i = 2; i < LIM; i++){
        if(phi[i] == i){
            for(j = i; j < LIM; j += i)
                phi[j] =
(phi[j]/i)*(i-1);
        }
    }

//    for(i = 0; i < LIM; i++){
//        cout << i << ": " << phi[i] <<
endl;
//    }

}

void GCD()
{
    ull i;

    for(i = 1; i < LIM; i++)
        ans[i] += ans[i-1];

}
int main()
{
    //make_prime();
    ETF();
    divi();
    GCD();
    ull t, n, i, cas = 1;
    cin >> t;
    while(t--){
        scanf("%llu", &n);
        printf("Case %llu: %llu\n",
cas++, ans[n]);
    }
    return 0;
}
```

j. **Extended GCD**

```
void ext_gcd(int a, int b, int *X, int
*Y)
{
    int x, y, r, x1, y1, x2, y2, r1, r2,
q;
    x2 = 1, y2 = 0;
    x1 = 0, y1 = 1;
    for(r2 = a, r1 = b; r1 != 0; r2 =
r1, r1 = r, x2 = x1, x1 = x, y2 = y1, y1
= y){
        q = r2/r1;
        r = r2%r1;
        x = x2-q*x1;
        y = y2-q*y1;
```

```
    }
    *X = x2, *Y = y2;
}
```

**k. Linear Diophantine Equation**

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

bool find_any_solution(ll a, ll b, ll c,
ll &x0, ll &y0, ll &g) {
    g = gcd(abs(a), abs(b), x0, y0);
    if (c % g) {
        return false;
    }

    x0 *= c / g;
    y0 *= c / g;
    if (a < 0) x0 = -x0;
    if (b < 0) y0 = -y0;
    return true;
}
void shift_solution(ll & x, ll & y, ll
a, ll b, ll cnt) {
    x += cnt * b;
    y -= cnt * a;
}

ll find_all_solutions(ll a, ll b, ll c,
ll minx, ll maxx, ll miny, ll maxy) {
    ll tem;
    if(a == 0 and b == 0){
        if(c == 0) return
(abs(minx-maxx)+1)*(abs(miny-maxy)+1);
        else return 0;
    }
    else if(a == 0){
        tem = c/b;
        if(c%b == 0 and tem >= miny and
tem <= by) return abs(minx-maxx)+1;
        else return 0;
    }
    else if(b == 0){
        tem = c/a;
        if(c%a == 0 and tem >= minx and
tem <= maxx) return abs(miny-maxy)+1;
        else return 0;
    }
    ll x, y, g;
    if (!find_any_solution(a, b, c, x,
y, g))
        return 0;
    a /= g;
    b /= g;

    ll sign_a = a > 0 ? +1 : -1;
    ll sign_b = b > 0 ? +1 : -1;

    shift_solution(x, y, a, b, (minx -
x) / b);
    if (x < minx)
        shift_solution(x, y, a, b,
sign_b);
    if (x > maxx)
        return 0;
    ll lx1 = x;

    shift_solution(x, y, a, b, (maxx -
x) / b);
    if (x > maxx)
        shift_solution(x, y, a, b,
-sign_b);
    ll rx1 = x;

    shift_solution(x, y, a, b, -(miny -
y) / a);
    if (y < miny)
        shift_solution(x, y, a, b,
-sign_a);
    if (y > maxy)
        return 0;
    ll lx2 = x;

    shift_solution(x, y, a, b, -(maxy -
y) / a);
    if (y > maxy)
        shift_solution(x, y, a, b,
sign_a);
    ll rx2 = x;

    if (lx2 > rx2)
        swap(lx2, rx2);
    ll lx = max(lx1, lx2);
```

```
    ll rx = min(rx1, rx2);

    if (lx > rx)
        return 0;
    return (rx - lx) / abs(b) + 1;
}
```

l. **Euler Totient Function**

$$\begin{aligned}\varphi(n) &= \varphi(p_1^{k_1})\,\varphi(p_2^{k_2})\cdots\varphi(p_r^{k_r}) \\ &= p_1^{k_1-1}(p_1-1)\,p_2^{k_2-1}(p_2-1)\cdots p_r^{k_r-1}(p_r-1) \\ &= p_1^{k_1}\left(1-\frac{1}{p_1}\right)p_2^{k_2}\left(1-\frac{1}{p_2}\right)\cdots p_r^{k_r}\left(1-\frac{1}{p_r}\right) \\ &= p_1^{k_1}p_2^{k_2}\cdots p_r^{k_r}\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\cdots\left(1-\frac{1}{p_r}\right) \\ &= n\left(1-\frac{1}{p_1}\right)\left(1-\frac{1}{p_2}\right)\cdots\left(1-\frac{1}{p_r}\right).\end{aligned}$$

m. **Euler totient function using sieve**

```
#define MAX 10000000

unsigned long long phi[MAX + 7];

/**
* It took 0.902 secs to generate up to
1e7.
**/
void generatePhi() {
   phi[1] = 0;
   for (int i = 2; i <= MAX; i++) {
       if(!phi[i]) {
           phi[i] = i-1;
           for(int j = (i << 1); j <=
MAX; j += i) {
               if(!phi[j]) phi[j] = j;
               phi[j] = phi[j] * (i-1) /
i;
           }
       }
   }
}
```

n. **Digits of factorial**

$Base\ Conversion,\ log_b x\ = \frac{log_c x}{log_c b}$

```
ll digitfac(ll n, ll base){
    double x = 0;
    for (ll i = 1; i <= n; i++){
        x += log10(i) / log10(base); //
Base Conversion
    }
    ll res = x + 1 + eps; // eps means a
very small number such as 10^(-9)
    return res;
}
```

**2. Math**

a. **BigMod**

```
ll bigmod(ll a, ll b)

{
    if(b == 0) return 1;
    ll x = bigmod(a, b/2);
    x = (x*x)%mod;
    if(b%2 == 1) x = (x*a)%mod;
    return x;
}
```

b. **Trigonometry functions**

```
double PI = acos(-1); //
3.141592653589793238
double angle = 60.0; // angle value in
degree unit and double datatype
cos_val = cos ( angle * PI / 180.0 );
sin_val = sin ( angle * PI / 180.0 );
tan_val = tan ( angle * PI / 180.0 );

cos_inverse_val = acos ( val ) * 180.0 /
PI; //val in double, cos_inverse_val in
degree unit, double data type
sin_inverse_val = asin ( val ) * 180.0 /
PI;
tan_inverse_val = atan ( val ) * 180.0 /
PI;
tan_inverse_y_by_x_val = atan2 (y,x) *
180 / PI;
```

c. **Matrix Exponent**

```
#define ll long long
#define MAX 100000
#define mat_int long long
using namespace std;

int pw[] = {1, 10, 100, 1000, 10000};

struct matrix{
    int dimR, dimC, MOD = 10000;
    vector <vector <mat_int> > mat;
    matrix(int _dimR, int _dimC){
        dimR = _dimR;
        dimC = _dimC;
```

```
        mat.clear();
        mat.resize(dimR, vector
<mat_int> (dimC, 0));
        if(dimR == dimC){
            for(int i = 0; i < dimR;
i++)
                mat[i][i] = 1;
        }
    }

    matrix operator * (const matrix
&oth)
    {
        int nr = dimR;
        int nc = oth.dimC;
        matrix newMat(nr, nc);
        for(int i = 0; i < nr; i++){
            for(int j = 0; j < nc; j++){
                mat_int sum = 0;
                for(int k = 0; k < dimC;
k++)
                {
                    sum += (mat[i][k] *
oth.mat[k][j]) % MOD;
                    sum %= MOD;
                }
                newMat.mat[i][j] = sum;
            }
        }
        return newMat;
    }

    matrix operator ^ (ll p)
    {
        matrix res(dimR, dimC);
        matrix x = *this;

        while(p){
            if(p % 2 == 1) res = x *
res;
            x = x * x;
            p /= 2;
        }

        return res;
    }

    void print()
    {
        cout << "......" << endl;
        for(int i = 0; i < dimR; i++){
            for(int j = 0; j < dimC;
j++){
                cout << mat[i][j] << "
";
            }
            cout << endl;
        }
        cout << "......" << endl;
    }

};
```

**d. Inclusion-Exclusion**

```
ll inclusionExclusion(ll n)
{
    ll ans = 0;
    for(ll i = 1; i < (1LL <<
prime.size()); i++){
        if(__builtin_popcountll(i)%2){
            ll val = 1;
            for(ll pos = 0; pos <
prime.size(); pos++){
                if((i&(1LL << pos))) val
*= prime[pos];
            }
            ans += n/val;
        }
        else{
            ll val = 1;
            for(ll pos = 0; pos <
prime.size(); pos++){
                if((i&(1LL << pos))) val
*= prime[pos];
            }
            ans -= n/val;
        }
    }
    return n - ans;
}
```

**3. Counting**

**a. Derangement**

```
ll Derange(ll n)
{
    if(n==0) return 1;
    if(n==1) return 0;
    if(n==2) return 1;

    if(derange[n]!=-1) return
derange[n];
```

```
    return
derange[n]=((n-1)%mod*((Derange(n-1))%mo
d+(Derange(n-2)%mod))%mod)%mod;
}
```

**b. Catalan Number DP Style**

```
#define ll long long             ///
MACROS NEEDED
#define LIM 1005                 ///
MACROS NEEDED
#define mod 1000000007           ///
MACROS NEEDED

ll catalan[LIM];
void cata()
{
    catalan[0] = 1, catalan[1] = 1;  ///
Two Base Cases
    for(ll i = 2; i <= LIM; i++){
        ll sum = 0;
        for(ll j = 0; j < i; j++){
            sum +=
(catalan[j]*catalan[i-j-1])%mod;  ///
As, catalan[i] =
sum_of(catalan[j]*catalan[i-j-1])
            sum %= mod;  /// Answer with
mod 1e9+7
        }
        catalan[i] = sum;
    }
}
```

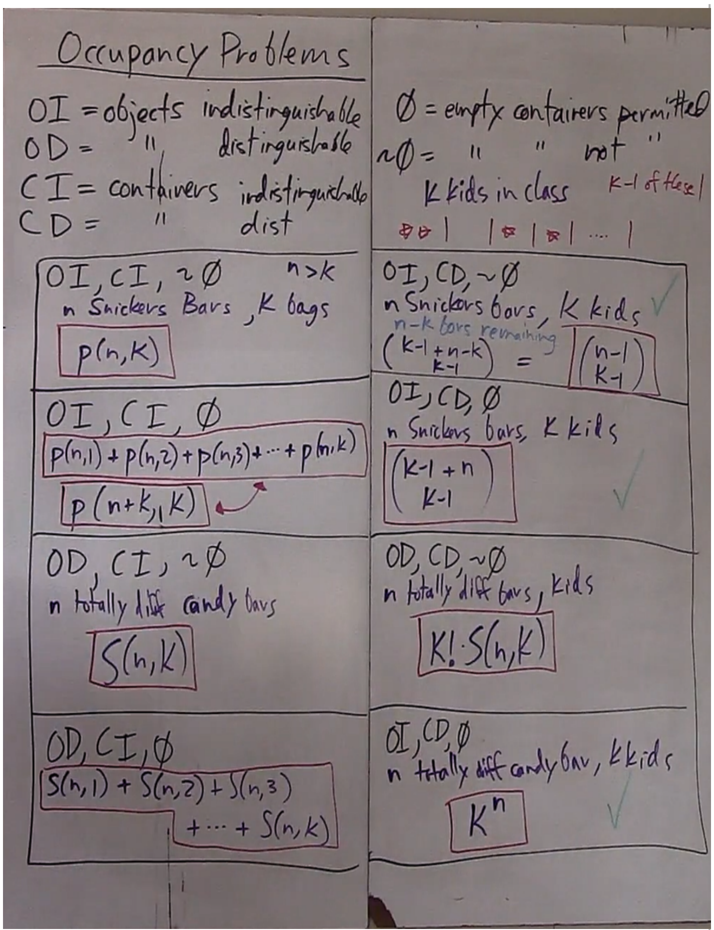
**c. Catalan Numbers Inverse mod**

```
#define ll long long   /// MACRO
NEEDED
#define LIM 300005     /// MACRO
NEEDED
#define mod 1000000007 /// MACRO
NEEDED

ll fac[LIM];
ll bigmod(ll a, ll b)
{
    if(b == 0) return 1;
    ll x = bigmod(a, b/2);
    x = (x*x)%mod;
    if(b%2 == 1) x = (x*a)%mod;
    return x;
}
void find_fac()
{
    ll i;
    for(i = 1, fac[0] = 1; i <
LIM; i++)
        fac[i] = (i*fac[i-1])%mod;
/// factorial generation upto LIM
}
ll ncr(ll n, ll r)
{
    ll up, down, ans;
    up = fac[n];
    down = (fac[r]*fac[n-r])%mod;
    ans = (up*bigmod(down,
mod-2))%mod;  /// modular
multiplicative inverse
    return (ans+mod)%mod;
}
ll cat(ll n)
{
    ll down;
    down = bigmod(n+1,mod-2);
/// modular multiplicative inverse
    return
(ncr(2*n,n)*down+mod)%mod;   ///
catalan[i] = (1/(i+1))*(2*i)Ci
}

/// fac();        NEED TO CALL THE
FUNCTION FROM MAIN
/// val = cal()   NEED TO CALL THE
FUNCTION FROM MAIN
```

**d. Occupancy Problems picture**




**e. Pascal Triangle (NCR DP style)**

```
ll ncr[1005][1005];
void precal()
{
    ll i, j;
    for(i = 0; i < 1005; i++)
        ncr[i][0] = 1;
    for(i = 1; i < 1005; i++)
            for(j = 1; j < 1005; j++)
                ncr[i][j] = (ncr[i-1][j]+ncr[i-1][j-1])%mod;
}
```

**f. NCR Inverse mod**

```
ll fac[LIM];
ll bigmod(ll a, ll b)
{
    if(b == 0) return 1;
    ll x = bigmod(a, b/2);
    x = (x*x)%mod;
    if(b%2 == 1) x = (x*a)%mod;
    return x;
}
void find_fac()
{
    ll i;
    for(i = 1, fac[0] = 1; i < LIM; i++)
        fac[i] = (i*fac[i-1])%mod;

}
ll ncr(ll n, ll r)
{
    ll up, down, ans;
    up = fac[n];
    down = (fac[r]*fac[n-r])%mod;
    ans = (up*bigmod(down, mod-2))%mod;
    return (ans+mod)%mod;
}
// add find_fac() in code
// add ncr() in code
```

## 4. Data Structure

**a. Segment tree**

```
#include <bits/stdc++.h>
using namespace std;
#define MAX 100005
#define left st, (st + en) / 2, nd + nd
#define right (st + en) / 2 + 1, en, nd + nd + 1
int tree[4 * MAX + 5];
int n, arr[MAX + 5];

void build(int st, int en, int nd)
{
    if(st == en)
    {
        tree[nd] = arr[st];
        return;
    }
    build(left); /// left subtree
    build(right); /// right subtree
    tree[nd] = tree[nd + nd] + tree[nd + nd + 1];
}

int query(int st, int en, int nd, int L, int R)
{
    if(L <= st && en <= R) return tree[nd]; /// if the query segment is completely overlapping our tree segment/node.
    if(en < L || R < st) return 0;
    return query(left, L, R) + query(right, L, R);
```

```
}

void update(int st, int en, int
nd, int idx, int v)
{
    if(en < idx || idx < st)
return;
    if(st == en)
    {
        tree[nd] += v;
        return;
    }
    update(left, idx, v);
    update(right, idx, v);
    tree[nd] = tree[nd + nd] +
tree[nd + nd + 1];
}

int main()
{
    /// build(0, n - 1, 1);
    cin >> n;
    for(int i = 0; i < n; i++) cin
>> arr[i];
    build(0, n - 1, 1); /// O(4 *
N)
    int Q;
    cin >> Q;
    while(Q--)
    {
        int com;
        cin >> com;
        if(com == 0) /// update
        {
            int idx, v;
            cin >> idx >> v;
            update(0, n - 1, 1,
idx, v);
        }
        else if(com == 1) ///
query
        {
            int L, R;
            cin >> L >> R;
            cout << query(0, n -
1, 1, L, R) << "\n";
        }
    }
    return 0;
}
```

**b. Segment tree with lazy**

```
#include <bits/stdc++.h>
#define ll long long
#define LIM 100005
#define left st, (st + ed) / 2, nd
+ nd
#define right (st + ed) / 2 + 1,
ed, nd + nd + 1
using namespace std;

ll arr[LIM], tree[4*LIM],
lazy[4*LIM];

void build(ll st, ll ed, ll nd)
{
    if(st == ed)
    {
        tree[nd] = arr[st];
        return;
    }
    build(left); /// left subtree
    build(right); /// right
subtree

    tree[nd] = tree[nd + nd] +
tree[nd + nd + 1];
}

void update(ll st, ll ed, ll nd,
ll L, ll R, ll val)
{
    if(lazy[nd] != 0){  /// IF ANY
UPDATE IS PENDING
        tree[nd] += (ed - st +
1)*lazy[nd]; /// UDPATE THE TREE
        if(st != ed){
            lazy[nd + nd] +=
lazy[nd];  /// PROPAGATE PENDING
UPDATES TO THE LEFT CHILD
            lazy[nd + nd + 1] +=
lazy[nd];  /// PROPAGATE PENDING
UPDATES TO THE RIGHT CHILD
        }
        lazy[nd] = 0;   /// ALL
PENDINGS ARE CLEAR
    }

    if(ed < L or R < st) return;
/// INVALID STATE

    if(L <= st and ed <= R){
```

```
        tree[nd] += (ed - st + 1)*val;  /// UPDATE THE RANGE WITH THE INSERTED VALUE 'val'
        if(st != ed){
            lazy[nd + nd] += val; /// PROPAGATE PENDING UPDATES TO THE LEFT CHILD
            lazy[nd + nd + 1] += val;/// PROPAGATE PENDING UPDATES TO THE RIGHT CHILD
        }
        return;
    }
    update(left, L, R, val); /// UPDATE LEFT CHILD
    update(right, L, R, val); /// UPDATE RIGHT CHILD

    tree[nd] = tree[nd + nd] + tree[nd + nd + 1];   /// KEEP THE ANSWERS OF CHILDS IN PARANT NODE

}

ll query(ll st, ll ed, ll nd, ll L, ll R)
{
    if(ed < L or R < st) return 0; /// INVALID STATE

    if(lazy[nd] != 0){  /// IF ANY UPDATE IS PENDING
        tree[nd] += (ed - st + 1)*lazy[nd]; /// UPDATE THE TREE
        if(st != ed){
            lazy[nd + nd] += lazy[nd];  /// PROPAGATE PENDING UPDATES TO THE LEFT CHILD
            lazy[nd + nd + 1] += lazy[nd]; /// PROPAGATE PENDING UPDATES TO THE RIGHT CHILD
        }
        lazy[nd] = 0;   /// ALL PENDINGS ARE CLEAR
    }

    if(L <= st and ed <= R){
        return tree[nd];    /// OVERLAPPED STATE
    }
    return query(left, L, R) + query(right, L, R);  /// KEEP THE ANSWERS OF CHILDS IN PARANT NODE
}
```

**c. <u>Sparse table</u>**

```
#include <bits/stdc++.h>
#define ll long long
#define LIM 100005
using namespace std;

int arr[LIM], dp[LIM][22], log_array[LIM];  /// NEVER USE LOG/SQRT ETC. IN SIZE DECLARATION

/// dp[array_size][log2(array_size)]

/// dp[starting_index][range]

void construct(int n)   /// n = ARRAY SIZE; O(nlogn)
{
    log_array[1] = 0;
    for(int i = 2; i <= n; i++)
        log_array[i] = 1 + log_array[i / 2];    /// GENARATING 2 BASED LOG VALUES 1 TO n

    for(int i = 0; i < n; i++)
        dp[i][0] = arr[i]; /// SINGLE RANGE RESULTS or BASE CASE

    int k = log_array[n] + 1;

    for(int j = 1; j <= k; j++){
        for(int i = 0; i + (1 << (j - 1)) < n; i++){
            dp[i][j] = min(dp[i][j - 1], dp[i + (1 << (j - 1))][j - 1]);
        }
    }
    // dp[0...n-1][1]
    // dp[0...n-1][2]
    // ...
    // dp[0...n-1][K]

}
```

```
int query(int L, int R) /// O(1)
{
    int len = R - L + 1;
    //if(len <= 0) return INT_MAX;
    int lg = log_array[len];
    return min(dp[L][lg], dp[R -
(1 << lg) + 1][lg]);   /// RETURN
ANSWER FROM 2 OVERLAPPING RANGES
}
```

**d. <u>LCA (Lowest Common Ancestor)</u>**

```
#include <bits/stdc++.h>
#define ll long long
#define pb push_back
using namespace std;

int parent[1005][22], depth[1005];
vector <int> edg[1005];

void dfs(int u, int pr) /// O(n)
{
    if(pr != -1)
        depth[u] = depth[pr] + 1;
    parent[u][0] = pr;

    for(int i = 0; i <
edg[u].size(); i++){
        int v = edg[u][i];
        if(v != pr) dfs(v, u);
    }

}

void build(int n)   /// O(nlogn)
{
    int lg = log2(n) + 1;

    for(int k = 1; k <= lg; k++){
        for(int i = 1; i <= n;
i++){
            if(parent[i][k-1] !=
-1)
                parent[i][k] =
parent[parent[i][k-1]][k-1];
        }
    }

}

int findLCA(int n, int u, int v)
/// O(nlogn)
{
    if(depth[u] > depth[v])
swap(u, v);

    int diff = depth[v] -
depth[u];

    int lg = log2(n) + 1;
    for(int i = 0; i <= lg; i++){
        if(diff & (1 << i))
            v = parent[v][i];
    }

    if(u == v) return v;

    for(int i = lg; i >= 0; i--){
        if(parent[u][i] !=
parent[v][i]){
            u = parent[u][i];
            v = parent[v][i];
        }
    }

    return parent[u][0];
}

void init(int n)
{
    int lg = log2(n) + 1;
    for(int i = 0; i <= n; i++){
        edg[i].clear();
        depth[i] = 0;
        for(int j = 0; j <= lg;
j++)
            parent[i][j] = -1;
    }
}
```

**e. <u>DSU (Disjoint Set Union)</u>**

```
int findP(int u)
{
    if(u == pr[u]) return u;
    return pr[u] = findP(pr[u]);
}

void connect(int u, int v)
{
    u = findP(u);
    v = findP(v);
```

```
    if(u != v)
        pr[u] = v;
}

/// findP(value)  NEED TO CALL FROM MAIN
/// connect(x, y) NEED TO CALL FROM MAIN
/// INITIALIZE VALUE OF pr[] FROM MAIN
```

**5. Graph Theory**

**a. Dijkstra Shortest Path**

```
#define pb push_back
#define pii pair<int,int>
#define ff first
#define ss second

vector <int> v[1005], w[1005];
int arr[10005];

void dj(int dis, int node)
{
    priority_queue <pii, vector< pii >,
greater<pii > > pq;
    pii up;
    memset(arr,127,sizeof arr);
    pq.push({dis,node}); arr[node] = 0;
    while(!pq.empty()){
        up = pq.top(); pq.pop();
        for(int i = 0; i <
v[up.ss].size(); i++){
            int ver = v[up.ss][i];
            int wei = w[up.ss][i];
            if(wei+up.ff < arr[ver]){

pq.push({wei+up.ff,ver});
                arr[ver] = wei+up.ff;
            }
        }
    }
}
```

**b. Floyd Warshall**

```
void floyd_warshall() {
    for (int k = 0; k < NODE; k++)
{// remember that loop order is
k->i->j
        for (int i = 0; i < NODE;
i++) {
            for (int j = 0; j <
NODE; j++) {
                AdjMat[i][j] =
min(AdjMat[i][j], AdjMat[i][k] +
AdjMat[k][j]);
                p[i][j] = p[k][j];
// update the parent matrix for
path print
            }
        }
    }
}
```

**c. Articulation points and bridges**

```
/// ARTICULATION BRIDGES
#define ll long long  /// MACROS
NEEDED
#define pb push_back  /// MACROS
NEEDED
#define LIM 100005    /// MACROS
NEEDED
ll dis[LIM], vis[LIM], par[LIM],
low[LIM], cnt, n;
vector <ll> ver[LIM];
vector <pair<ll,ll> > bridges;
void dfs(ll u)
{
    dis[u] = low[u] = cnt++;
    for(ll i = 0; i <
ver[u].size(); i++){
        ll v = ver[u][i];
        if(v != par[u]){
            if(vis[v] == 0){  ///
TREE EDGES
                par[v] = u;   ///
SAVING PARTENTS
                vis[v] = 1;
                dfs(v);
                low[u] =
min(low[u], low[v]); /// IF TREE
EDGE, LOW[U] = MIN(LOW[U] OF ALL
OF ITS CHILD)
            }
            else if(vis[v] == 1){
/// BACK EDGES
                low[u] =
min(low[u], dis[v]); /// IF BACK
EDGE, LOW[U] = MIN(MIN OF
DIS[CHILD] AND LOW[U])
            }
            if(dis[u] < low[v])
bridges.pb({min(u,v), max(u,v)});
/// ARTICULATION EDGES
```

```
        }
    }
    vis[u] = 2;   /// FORWARD
EDGES
}




/// ARTICULATION POINTS
int n, m, tim, dis[LIM], low[LIM],
par[LIM];
vector <int> edg[LIM], artpoint;

void dfs(int u)
{
    dis[u] = low[u] = tim++;
    int child = 0, mx = -inf;
    for(int i = 0; i <
edg[u].size(); i++){
        int v = edg[u][i];
        if(v != par[u]){
            if(dis[v]){     ///
BACK EDGE
                low[u] =
min(low[u], dis[v]); /// IF BACK
EDGE, LOW[U] = MIN(MIN OF
DIS[CHILD] AND LOW[U])
            }
            else{           ///
TREE EDGE
                par[v] = u; ///
SAVING PARENT
                dfs(v);
                low[u] =
min(low[u], low[v]);   /// IF TREE
EDGE, LOW[U] = MIN(LOW[U] OF ALL
OF ITS CHILD)
                if(low[v] >=
dis[u] && u != 1)
                    mx = max(mx,
dis[v]);
                child++;
            }
        }
    }
    if(u != 1 && mx >= dis[u])
artpoint.pb(u);  /// IF THE NODE
IS NOT ROOT
    if(u == 1 && child > 1)
artpoint.pb(u);     /// IF THE
NODE IS ROOT CHECK IF (# OF CHILD
> 1)
}
```

**d. MST (Min/Max Spanning Tree)**

```
#define ll long long  /// MACRO
NEEDED
#define pb push_back  /// MACRO
NEEDED

int n, pr[105], sz[105];
struct abc{
    int w, u, v;
};
vector <abc> edg;

bool cmp(abc x, abc y)
{
    return x.w < y.w;
}

int findP(int u)
{
    if(u == pr[u]) return u;
    return pr[u] = findP(pr[u]);
}

void connect(int u, int v)
{
    u = findP(u);
    v = findP(v);
    if(u != v){
        if(sz[u] < sz[v]){  ///
MARGING SMALLER COMPONENT WITH
LARGER COMPONENT
            pr[u] = v;
            sz[v] += sz[u];
        }
        else{
            pr[v] = u;
            sz[u] += sz[v];
        }
    }
}

int kruskal(int state)  /// STATE
= 0 (MIN), STATE = 1(MAX)
{
    int i;
    for(i = 0; i <= n; i++){
        pr[i] = i;
```

```
            sz[i] = 1;
        }
        sort(edg.begin(), edg.end(),
    cmp);
        if(state)
            reverse(edg.begin(),
    edg.end());
        int cost = 0;
        for(i = 0; i < edg.size();
    i++){
            int parx =
    findP(edg[i].u);
            int pary =
    findP(edg[i].v);
            if(parx != pary){
                cost += edg[i].w;
                connect(parx, pary);
            }
        }
        return cost;
    }
```

**e. King, Horse & Adjacent Walk**

```
#define ll long long

ll king_xx[] = {1, -1, 0, 0, 1, 1, -1,
-1};
ll king_yy[] = {0, 0, 1, -1, 1, -1, 1,
-1};

ll horse_xx[] = {1, 2, 2, 1, -1, -2, -2,
-1};
ll horse_yy[] = {2, 1, -1, -2, -2, -1,
1, 2};

ll adj_xx[] = {1, -1, 0, 0};
ll adj_yy[] = {0, 0, 1, -1};
```

## 6. Strings

**a. Trie**

```
string s;
int n, trie[LIM][15], cnt = 2;
bool flag[LIM], endpoint[LIM];

void add()
{
    int u = 1;
    for(int i = 0; i < s.size(); i++){
        int edg = s[i] - '0';
        if(!trie[u][edg]){
            trie[u][edg] = cnt++;
        }
        u = trie[u][edg];
        flag[u] = 1;
    }
    endpoint[u] = 1;
}

bool query()
{
    int u = 1;
    for(int i = 0; i < s.size(); i++){
        int edg = s[i] - '0';
        if(trie[u][edg])
           u = trie[u][edg];
        else
            return 1;
        if(endpoint[u]) return 0;
    }

    return (flag[u] == 0);
}

void reset(int u)
{
    for(int i = 0; i < 10; i++){
        if(trie[u][i])
reset(trie[u][i]);
        trie[u][i] = flag[u] =
endpoint[u] = 0;
    }
}
```

**b. KMP**

```
#include <bits/stdc++.h>
#define ll long long
#define LIM 1000006
using namespace std;

char temp[LIM];
string text, pat;
int prefix[LIM];

void calcPrefix()
{
    prefix[0] = prefix[1] = 0;

    for(int i = 2; i < pat.size(); i++){
        int p = i - 1;
        while(1){
            if(pat[i] == pat[ prefix[p]
+ 1 ]){
```

```
                prefix[i] = prefix[p] + 1;
                break;
            }
            else if(!p) break;
            else p = prefix[p];
        }
    }
}

int kmp() /// returns the # of matched substrings
{
    int p = 0, ans = 0;
    for(int j = 0; j < text.size(); j++){
        while(p && pat[p + 1] != text[j]){
            p = prefix[p];
        }
        if(pat[p + 1] == text[j]){
            p++;
        }
        if(p + 1 == pat.size()){
            ans++;
        }
    }
    return ans;
}

int main()
{
    int cas = 1, t;
    scanf("%d", &t);
    while(t--){
        scanf("%s", temp); text = temp;
        scanf("%s", temp); pat = temp;
        pat = "#" + pat;
        calcPrefix();
        printf("Case %d: %d\n", cas++, kmp());
    }

    return 0;
}
```

c. **Hashing**

```
#define LIM 1000006

ll prefHash[2][LIM], basePower[2][LIM];
ll mod[2] = {1000000007, 1000000009};
ll base[2] = {31, 101};
string str;
char temp[LIM];

void preCal()
{
    prefHash[0][0] = prefHash[1][0] = str[0] - 'a' + 1;
    basePower[0][0] = basePower[1][0] = 1;

   for(int j = 0; j < 2; j++){
        for(int i = 1; i < str.size(); i++){
            basePower[j][i] = (basePower[j][i - 1]*base[j]) %mod[j];
            prefHash[j][i] = (prefHash[j][i - 1]*base[j] + str[i] - 'a' + 1) %mod[j];
        }
    }
}

ll evaluateHash(int idx, int L, int R)
{
    if(L == 0) return prefHash[idx][R];
    return (prefHash[idx][R] - (prefHash[idx][L - 1]*basePower[idx][R - L + 1]) %mod[idx] + mod[idx]) %mod[idx];
}
```

d. **Sub-string palindrome check using Hashing**

```
#define ll long long
#define LIM 1000006

ll prefHash[2][LIM], prefHashRev[2][LIM], basePower[2][LIM];
ll mod[2] = {1000000007, 1000000009};
ll base[2] = {31, 101};
string str, revStr;
char temp[LIM];

void preCal()
{
    prefHash[0][0] = prefHash[1][0] = str[0] - 'a' + 1;
    prefHashRev[0][0] = prefHashRev[1][0] = revStr[0] - 'a' + 1;
    basePower[0][0] = basePower[1][0] = 1;
```

```
   for(int j = 0; j < 2; j++){
        for(int i = 1; i < str.size();
i++){
            basePower[j][i] =
(basePower[j][i - 1]*base[j]) %mod[j];
            prefHash[j][i] =
(prefHash[j][i - 1]*base[j] + str[i] -
'a' + 1) %mod[j];
            prefHashRev[j][i] =
(prefHashRev[j][i - 1]*base[j] +
revStr[i] - 'a' + 1) %mod[j];
        }
    }
}

ll getHash(int idx, int L, int R)
{
    if(L == 0) return prefHash[idx][R];
    return (prefHash[idx][R] -
(prefHash[idx][L - 1]*basePower[idx][R -
L + 1]) %mod[idx] + mod[idx]) %mod[idx];
}

ll getHashRev(int idx, int L, int R)
{
    if(L == 0) return
prefHashRev[idx][R];
    return (prefHashRev[idx][R] -
(prefHashRev[idx][L -
1]*basePower[idx][R - L + 1]) %mod[idx]
+ mod[idx]) %mod[idx];
}

bool isPalindrome(int L, int R)
{
    int sz = str.size();
    if(getHash(0, L, R) == getHashRev(0,
sz - R - 1 , sz - L - 1) &&
       getHash(1, L, R) == getHashRev(1,
sz - R - 1, sz - L - 1)) return 1;
    return 0;
}

/// preCal() -> CALL FROM MAIN
/// isPalindrome(start, end) -> CALL
FROM MAIN
```

**7. Template**

```
//start_template
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>

#define ll long long
#define ull unsigned long long
#define pb push_back
#define pii pair<int, int>
#define ff first
#define ss second
#define nl '\n'
#define mod 1000000007
#define inf 1000000009
#define MAXX 1000000000000015
#define LIM 300005
#define eps 1e-9
#define pi acos(-1)

using namespace std;
using namespace __gnu_pbds;
/*
Activate it for pbds set
 x.find_by_order(pos) ----> returns the
value at "pos" index in the set 0-based
 x.order_of_key(value)     ----> returns
the position of "value" in the set
0-based
 */
#define ordered_set tree<pii,
null_type,less<pii>, rb_tree_tag,
tree_order_statistics_node_update>
void FAST_IO() {
ios_base::sync_with_stdio(false);
cin.tie(0); cout.tie(0); }
/*
Partition Numbers:
Formula: P(n,k) = P(n-1,k-1) + P(n-k,k)
Base Case:
P(n,k) = 0 if n < k
P(n,k) = 0 if k = 0 [P(n,0) = 0]
P(n,k) = 1 if(k = n) [P(n,n) = 1]

Catalan Numbers:
Formula: C(n) = SumOf( C(k) * C(n-1-k)
), n >= 2, 0 <= k < n
Base Case: C(0) = C(1) = 1
Formula2: C(n) = (1/(n + 1)) * ncr(2*n,
n)

Starling Numbers of 2nd kind:
Formula: S(n,k) = S(n-1,k-1) + k *
S(n-1,k)
```

```
Base Case:
S(n,k) = 0 if k > n
S(n,k) = 1 if k = 1 [S(n,1) = 1]
S(n,k) = 1 if n = k [S(n,n) = 1]
S(n,k) = 0 if k = 0 [S(n,0) = 0]

Moves on Grid:
int king_xx[] = {1, -1, 0, 0, 1, 1, -1,
-1};
int king_yy[] = {0, 0, 1, -1, 1, -1, 1,
-1};

int horse_xx[] = {1, 2, 2, 1, -1, -2,
-2, -1};
int horse_yy[] = {2, 1, -1, -2, -2, -1,
1, 2};

int adj_xx[] = {1, -1, 0, 0};
int adj_yy[] = {0, 0, 1, -1};

/// bit manipulations
/// bool checkbit(int mask,int
bit){return mask & (1<<bit);}
/// int setbit(int mask,int bit){ return
mask  (1<<bit) ; }
/// int clearbit(int mask,int
bit){return mask & ~(1<<bit);}
/// int togglebit(int mask,int
bit){return mask ^ (1<<bit);}
*/

int main()
{
   ///MUST READ THE POINTS BELOW BEFORE
SUBMIT
   FAST_IO();




   return 0;
   ///MUST READ THE POINTS BELOW BEFORE
SUBMIT
}
/*
    1. LOOK SPECIAL CASE N = 1.
    2. LOOK FOR OVERFLOW.
    3. LOOK FOR OUT OF BOUNDS.
    4. ALWAYS TEST WITH HAND-MADE
TEST-CASES BEFORE SUBMIT.
*///end_template
```

**8. Mo’s ordering**

```
struct query {
    ll id, l, r;
    bool operator < (const query &
other) const {
        int block_a = l / block_size;
        int block_b = other.l /
block_size;
        if(block_a == block_b) return r
< other.r;
        return block_a < block_b;
    }
};
ll arr[mx + 5], left, right, res[mx +
5], freq[mx + 5];
query queries[50005];
void add(ll idx) {
    freq[arr[idx]]++;
    if(freq[arr[idx]] == 1)
unique_values++;
}
void remov(ll idx) {
    freq[arr[idx]]--;
    if(freq[arr[idx]] == 0)
unique_values--;
}
//in main function
for(ll i = 0; i < q; i++) {
            while(queries[i].l < left)
add(--left);
            while(queries[i].l > left)
remov(left++);
            while(queries[i].r < right)
remov(right--);
            while(queries[i].r > right)
add(++right);

            res[queries[i].id] =
unique_values;
        }
```

**9. Maximum Subarray Sum (Kadane’s Algo)**
**//General version**

```
vector<int> a(n);
int ans = a[0], sum = 0;
for (int r = 0; r < n; ++r) {
    sum += a[r];
    ans = max(ans, sum);
    sum = max(sum, 0);
}
cout << sum << nl;
```

**//With index of the subarray of max-sum(left-right)**

```
vector<int> a(n);
int ans = a[0], ans_l = 0, ans_r = 0;
int sum = 0, minus_pos = -1;
for (int r = 0; r < n; ++r) {
    sum += a[r];
    if (sum > ans) {
        ans = sum;
        ans_l = minus_pos + 1;
        ans_r = r;
    }
    if (sum < 0) {
        sum = 0;
        minus_pos = r;
    }
}
```

**10. Dynamic Programming**

a. **How many zeroes**

```
ll dp[15][5][5][5], dp2[15][5][5];
ll f2(ll pos, ll issmall, ll num)
{
    /// HOW MANY NUMBERS SMALLER THAN n
IS POSSIBLE FROM THIS STATE
(pos,issmall)
    if(pos == s[num].size()) return 1;
    if(dp2[pos][issmall][num] != -1)
return dp2[pos][issmall][num];
    ll lo = 0, hi = s[num][pos]-'0', sum
= 0;
    if(issmall) hi = 9;
    for(; lo <= hi; lo++){
        sum += f2(pos+1,issmall |
(lo<hi), num);
    }
    return dp2[pos][issmall][num] = sum;
}
ll f(ll pos, ll issmall, ll hasstarted,
ll num)
{
    if(pos == s[num].size()) return 0;
    if(dp[pos][issmall][hasstarted][num]
!= -1) return
dp[pos][issmall][hasstarted][num];
    ll lo = 0, hi = s[num][pos]-'0', sum
= 0;
    if(issmall) hi = 9;
    for(; lo <= hi; lo++){
        ll tem = f(pos+1, issmall |
(lo<hi), hasstarted | (lo != 0), num);
        /// ALREADY STARTED AND
RIGHT-NOW WE ARE PUTTING 0 AT CURRENT
POSITION
        /// SO WE WILL HAVE TO FIND OUT
IN HOW MANY WAYS THIS ZERO WILL
CONTRIBUTE
        if(hasstarted and lo == 0) tem
+= f2(pos+1,issmall | (lo<hi),num);
        sum += tem;
    }
    return
dp[pos][issmall][hasstarted][num] = sum;
}
```

**11.Searching**

a. **Ternary Search**

```
double ternary_search(double l, double
r) {
    double eps = 1e-9;
//set the error limit here
    while (r - l > eps) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        double f1 = f(m1);
//evaluates the function at m1
        double f2 = f(m2);
//evaluates the function at m2
        if (f1 < f2)
            l = m1;
        else
            r = m2;
    }
    return f(l);
//return the maximum of f(x) in [l, r]
}
```