Index

Useful Things

→Fast I/O

```
C++:
#pragma GCC optimize("Ofast")
#pragma GCC target("avx,avx2,fma")
ios_base::sync_with_stdio(false),
cin.tie(nullptr), cout.tie(nullptr);
freopen("input.txt", "r", stdin);
Python:
import sys
input = sys.stdin.readline
sys.stdout.write("------")
```

→Vim Setup:

```
"install xclip, vim-gtk3
set nocp ai bs=2 hls ic is lbr ls=2 mouse=a
nu ru sc scs smd so=3 sw=2 ts=2 rnu
filetype plugin indent on
syn on
map gA m'ggVG"+y''
inoremap {<CR> {<CR>}<Esc>ko
nnoremap = mzgg=G`z
vnoremap <C-k> :s/^\s*\zs/\/\/ /<CR>:nohl<CR>
nnoremap <C-k> :s/^\s*\zs/\/\/ /<CR>:nohl<CR>
vnoremap <C-l> :s/^\s*\zs\/\/ //<CR>:nohl<CR>
nnoremap <C-l> :s/^\s*\zs\/\/ //<CR>:nohl<CR>
autocmd FileType cpp map <F9> :w<CR> :!clear;
g++ --std=c++17 % -DONPC -o %:r && ./%:r<CR>
```

→ Vscode Setup

```
{
   "key": "f5",
   "command":
"workbench.action.terminal.sendSequence",
   "args": {
      "text": "clear && g++
${fileBasenameNoExtension}.cpp -o
     ${fileBasenameNoExtension} &&
     ./${fileBasenameNoExtension}
  < in.txt > out.txt\n"
   }
} // need "files.autoSaveDelay": 100
```

→ Policy Based Data Structure

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename T> using o_set = tree<T,
null_type, less<T>, rb_tree_tag,
tree_order_statistics_node_update>;
// find_by_order(k) - returns an iterator to
// the k-th largest element (0 indexed);
// order_of_key(k)-the number of elements in
// the set that are strictly smaller than k;
```

1 Formula

1.1 Area Formulas

Rectangle: $Area = length \times width$
Square: $Area = side \times side$
Triangle: $Area = \frac{1}{2} \times base \times height$
Circle: $Area = \pi \times radius^2$
Parallelogram: $Area = base \times height$
Pyramid Base: $Area = \frac{1}{2} \times base \times slant\ height$
Polygon :

(a) $Area = \frac{1}{2}\left|\sum_{i=1}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)\right|$

(b) $Area = a + \frac{b}{2} - 1$ (for int coordinates)
here, a=#int points inside polygon
b=#int points outside polygon

1.2 Perimeter Formulas

Rectangle: $Perimeter = 2 \times (length + width)$
Square: $Perimeter = 4 \times side$
Triangle: $Perimeter = sum\ of\ all\ sides$
Circle: $Circumference = 2 \times \pi \times radius$

1.3 Volume Formulas

Cube: $Volume = side^3$
Rect Prism: $Volume = length \times width \times height$
Cylinder: $Volume = \pi \times radius^2 \times height$
Sphere: $Volume = \frac{4}{3} \times \pi \times radius^3$
Pyramid: $Volume = \frac{1}{3} \times base\ area \times height$

1.4 Surface Area Formulas

Cube: $Surface\ Area = 6 \times side^2$
Rectangular Prism:
Surface Area = 2 × (length × width + length × height + width × height)
Cylinder:
Surface Area = 2 × π × radius × (radius + height)
Sphere: $Surface\ Area = 4 \times \pi \times radius^2$
Pyramid:
Surface Area = base area + $\frac{1}{2}$ *× perimeter of base × slant height*

Triangles
Side lengths: a, b, c
Semiperimeter: $p = \frac{a+b+c}{2}$
Area: A = $\sqrt{(p(p-a)(p-b)(p-c))}$
Circumradius:$R = \frac{abc}{4A}$
Inradius: $r = \frac{A}{p}$
Length of median (divides triangle into two equal-area triangles):
$ma = \frac{1}{2} * \sqrt{2b^2 + 2c^2 - a^2}$
Length of bisector (divides angles in two):
$sa = \sqrt{\frac{bc}{1-(\frac{a}{b+c})^2}}$

1.5 Trigonometry

Law of sines: $sin\frac{\alpha}{a} = sin\frac{\beta}{b} = sin\frac{\gamma}{c} = \frac{1}{2R}$
Law of cosines: $a^2 = b^2 + c^2 - 2bc\cos\alpha$
Law of tangents: $\frac{a+b}{a-b} = \frac{\tan\frac{\alpha+\beta}{2}}{\tan\frac{\alpha-\beta}{2}}$
$\sin(A+B) = \sin A\cos B + \cos A\sin B$
$\cos(A+B) = \cos A\cos B - \sin A\sin B$
$\sin(A-B) = \sin A\cos B - \cos A\sin B$
$\cos(A-B) = \cos A\cos B + \sin A\sin B$
$\tan(A+B) = \frac{(\tan A + \tan B)}{(1-\tan A\tan B)}$
$\tan(A-B) = \frac{(\tan A - \tan B)}{(1+\tan A\tan B)}$
Double Angle and Half Angle Formulas:
sin 2θ = 2 sin θ cosθ
$\cos 2\theta = \cos^2\theta - \sin^2\theta$
$\tan 2\theta = \frac{(2\tan\theta)}{(1-tan\hat{}2\,\theta)}$
$\sin(\frac{\theta}{2}) = \pm\sqrt{\frac{1-\cos\theta}{2}}$
$\cos(\frac{\theta}{2}) = \pm\sqrt{\frac{1+\cos\theta}{2}}$
$\tan(\frac{\theta}{2}) = \frac{(1-\cos\theta)}{\sin\theta}$
$\sin(r+w) = \sin r\cos w + \cos r\sin w$
$\cos(r+w) = \cos r\cos w - \sin r\sin w$
$\tan(r+w) = \frac{(\tan r + \tan w)}{(1-\tan r\tan w)}$
$\sin r + \sin w = 2\sin(\frac{r+w}{2})\cos(\frac{r-w}{2})$
$\cos r + \cos w = 2\cos(\frac{r+w}{2})\cos(\frac{r-w}{2})$
$(V+W)\tan(\frac{r-w}{2}) = (V-W)\tan(\frac{r+w}{2})$
where V, W are lengths of sides opposite angles r, w.
$a\cos x + b\sin x = r\cos(x-\varphi)$
$a\sin x - b\cos x = r\sin(x-\varphi)$
where $r = \sqrt{a^2 + b^2}$, $\varphi = atan2(b, a)$

1.6 Sum

$c^k + c^{k+1} + ... + c^n = c^{n+1} - c^k$
// (c - 1)   for c ≠ 1
$1 + 2 + 3 + \dots + n = \frac{n*(n+1)}{2}$
$1^2 + 2^2 + ... + n^2 = \frac{n*(n+1)*(2n+1)}{6}$
$1^3 + 2^3 + ... + n^3 = \frac{n^2*(n+1)^2}{4}$
$1^4 + 2^4 + ... + n^4 = \frac{n*(n+1)*(2n+1)*(3n^2+3n-1)}{30}$
Linear, Sn = n / 2 * (2 * a + (n - 1) * d))
Geometric, Sn = a * (r ^ n - 1) / (r - 1)

1.7 Pick's Theorem

A = I-(½)*B-1 [I-inside,B-boundary]

1.8 Approximate Fibo

F(n)=(((1+√5)/2)^n)-((1-√5)/2)^n))/√5

1.9 Stars and Bars

Number of solutions of x_1 + x_2 + x_3 + ... + x_k = n is
C(n-1, k-1) where x_i > 0 and
C(n+k-1, k-1) where x_i >= 0

1.10 Divisibility rules for a number:

3: The sum of its digits is divisible by 3.
4: Last two digits form a number that is divisible by 4.
5: The number ends in 0 or 5.
6: Divisible by both 2 and 3.
7: A rule for 7 is to double the last digit, subtract it from the rest of the number, and check if the result is divisible by 7. Repeat if necessary.
8: Last three digits form a number divisible by 8.
9: Sum of its digits is divisible by 9.
11: Difference between the sum of digits of odd places and even places is divisible by 11
Miscellaneous

1.11 Gen Random Number

```
#define accuracy
chrono::steady_clock::now().time_since_epoch(
).count()
mt19937 rng(accuracy);
int rand(int l, int r) {
  uniform_int_distribution<int> ludo(l, r);
  return ludo(rng);
}
```

1.12 Stress Sh

```
#!/usr/bin/env bash
g++ -g "$1".cpp -DONPC -o "$1"
g++ -g "$2".cpp -DONPC -o "$2"
g++ -g "$3".cpp -DONPC -o "$3"
for ((testNum=0;testNum<$4;testNum++))
do
      ./$3 > stdinput
      ./$2 < stdinput > outSlow
      ./$1 < stdinput > outWrong
      H1=`md5sum outWrong`
      H2=`md5sum outSlow`
      if !(cmp -s "outWrong" "outSlow")
      then
      echo "Error found!"
      echo "Input:"
      cat stdinput
      echo "Wrong Output:"
      cat outWrong
      echo "Slow Output:"
      cat outSlow
      exit
      fi
done
```

```
echo Passed $4 tests
# ./stress.sh wrong correct gen times
```

1.13 Error Checklist

1. Handle integer overflow. a * b + b * c -> 1LL * a * b + 1LL * b * c.
2. Check modular arithmetic is properly use. Ex. check (a % mod + b % mod) % mod is handled.
3. Check the input is signed or unsigned. Use ((a + b) % mod) + mod.
4. convert .size() to int
5. check if int is enough for result
6. check trailing whitespace
7. Check output format for all cases. Check what will print if the answer not found.
8. Check string input (empty string, white space).
9. Handle precision.

1.14 Miscellaneous

1. log n! = log 1 + log 2 + ... + log n
2. gcd(a, b) = gcd(a - b, b) Number of occurrences of a prime number p in n! is ⌊n/p⌋ + ⌊n/p2⌋ + ⌊n/p3⌋ + ... + 0
3. Number of divisors of p^x*q^y where p and q are prime is (x + 1) * (y + 1)
4. Sum of divisors of p^x*q^y where p and q are prime is (1 + p + p^2 + ... + p^x) (1 + q + q^2 + ... + q^y)

2 Number Theory

2.1 Divisor Count

```
int maxVal = 1e6 + 1;
vector<int> countDivisor(maxVal, 0);
void countingDivisor(){
    for (int i = 1; i < maxVal; i++)
        for(int j= i; j<maxVal;j+= i)
            countDivisor[j]++;
}
long long numberOfDivisors(long long num) {
    long long total = 1; for (int i = 2;
(long long)i * i <= num; i++) {   if (num % i
== 0) {   int e = 0;   do {   e++;
num /= i;   } while (num % i == 0);
total *= e + 1; }     }     if (num > 1) {
        total *= 2;
    }    return total;}
// count the number of divisors of all
numbers in a range.
```

2.2 Leap year

```
bool isLeap(int n){
    if (n%100==0)
        if (n%400==0) return true;
        else return false;
    if (n%4==0) return true;
    else return false;}
```

2.3 Num of Leap year in between

```
int calNum(int year) {
return (year / 4) - (year / 100) +
 (year / 400);
}
int leapNum(int l, int r) {
      return calNum(r) - calNum(--l);}
```

2.4 Print Calendar of any year

```
int dayNumber(int day, int month, int year){
   static int t[]={0,3,2,5,0,3,5,1,4,6,2,4};
   year -= month < 3;
   return (year + year / 4 - year / 100 +
      year / 400 + t[month - 1] + day) % 7;
}
string getMonthName(int monthNumber) {
    string months[]={"January", "February",
        "March","April","May","June","July",
        "August","September", "October",
        "November", "December"};
    return (months[monthNumber]);
}
int numberOfDays(int monthNumber, int year){
    if (monthNumber==1 && isLeapYear(year))
        return 29;
    int monthDays[] = {31, 28, 31, 30, 31,
                30, 31, 31, 30, 31, 30, 31};
    return (monthDays[monthNumber]);
}
void printCalendar(int year) {
    printf("       Calendar - %d\n\n",year);
    int days;
    int current = dayNumber(1, 1, year);
    // i--> Iterate through all the months
    // j--> Iterate through all the days of
        the month - i
    for (int i = 0; i < 12; i++) {
        days = numberOfDays(i, year);
        cout << "          |" <<
            getMonthName(i).c_str()
            << "|" << endl;
        printf(" Sun Mon Tue Wed Thu Fri
                 Sat\n");
        int k;
        for (k = 0; k < current; k++)
            printf("    ");
        for (int j = 1; j <= days; j++) {
            printf("%4d", j);
            if (++k > 6) {
                k = 0; cout << endl;
            }
        }
        if (k)
            cout << endl;
        cout <<
        "-----------------------------\n";
        current = k;
    }
}
```

2.5 Power

```
int x = (int)(pow(base, power) + 1e-18);
```

2.6 BINARY EXPONENTIATION:(a^b)

```
int binaryExp(int base, int power, int MOD =
mod) {
    int res = 1;
    while (power) {
        if (power & 1)
            res = (res * base) % MOD;
        base = ((base%MOD)*(base%MOD))%MOD;
        power /= 2;
    }
    return res;
}
```

2.7 BINARY EXPONENTIATION:(a^b^c)

```
//function call:
binaryExp(a, binaryExp(b, c, mod-1), mod)
```

2.8 Check is prime number-O(sqrt(n))

```
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;}
```

2.9 Prime factorization-O(sqrt(n))

```
// smallest prime factor of a number.
int factor(int n){
    int a;
    if (n%2==0)
        return 2;
    for (a=3; a<=sqrt(n); a+=2){
        if (n%a==0)
            return a;
    }
    return n;}
// complete factorization
int r;
while (n>1){
    r = factor(n);printf("%d", r);n /= r;}
```

2.10 Seive

```
const int N = 1e7 + 3;
vector<int> primes;
int notprime[N];
void sieve() {
  primes.push_back(2);
  for (int i = 4; i < N; i += 2) {
      notprime[i] = true;
  }
  for (int i = 3; i < N; i += 2) {
      if (!notprime[i]) {
      primes.push_back(i);
 for (int j = i * i; j < N; j += 2 * i) {
      notprime[j] = true;      }      }  }}
```

2.11 Optimized Seive(upto 1e9)

```
vector<int> sieve(const int N, const int Q =
17, const int L = 1 << 15) {
  static const int rs[] = {1, 7, 11, 13, 17,
19, 23, 29};
  struct P {
    P(int p) : p(p) {}
    int p; int pos[8];
  };
  auto approx_prime_count = [] (const int N)
-> int {
    return N > 60184 ? N / (log(N) - 1.1)
                     : max(1., N / (log(N) -
1.11)) + 1;
  };
  const int v = sqrt(N), vv = sqrt(v);
  vector<bool> isp(v + 1, true);
  for (int i = 2; i <= vv; ++i) if (isp[i]) {
    for (int j = i * i; j <= v; j += i)
isp[j] = false;
  }
  const int rsize = approx_prime_count(N +
30);
  vector<int> primes = {2, 3, 5}; int psize =
3;
  primes.resize(rsize);

  vector<P> sprimes; size_t pbeg = 0;
  int prod = 1;
  for (int p = 7; p <= v; ++p) {
    if (!isp[p]) continue;
    if (p <= Q) prod *= p, ++pbeg,
primes[psize++] = p;
    auto pp = P(p);
    for (int t = 0; t < 8; ++t) {
      int j = (p <= Q) ? p : p * p;
      while (j % 30 != rs[t]) j += p << 1;
      pp.pos[t] = j / 30;
    }
    sprimes.push_back(pp);
  }

  vector<unsigned char> pre(prod, 0xFF);
  for (size_t pi = 0; pi < pbeg; ++pi) {
    auto pp = sprimes[pi]; const int p =
pp.p;
    for (int t = 0; t < 8; ++t) {
      const unsigned char m = ~(1 << t);
      for (int i = pp.pos[t]; i < prod; i +=
p) pre[i] &= m;
    }
  }
  const int block_size = (L + prod - 1) /
prod * prod;
  vector<unsigned char> block(block_size);
unsigned char* pblock = block.data();
  const int M = (N + 29) / 30;

  for (int beg = 0; beg < M; beg +=
block_size, pblock -= block_size) {
    int end = min(M, beg + block_size);
    for (int i = beg; i < end; i += prod) {
      copy(pre.begin(), pre.end(), pblock +
i);
    }
    if (beg == 0) pblock[0] &= 0xFE;
    for (size_t pi = pbeg; pi <
sprimes.size(); ++pi) {
      auto& pp = sprimes[pi];
      const int p = pp.p;
      for (int t = 0; t < 8; ++t) {
        int i = pp.pos[t]; const unsigned
char m = ~(1 << t);
        for (; i < end; i += p) pblock[i] &=
m;
        pp.pos[t] = i;
      }
    }
    for (int i = beg; i < end; ++i) {
      for (int m = pblock[i]; m > 0; m &= m -
1) {
        primes[psize++] = i * 30 +
rs[__builtin_ctz(m)];
      }
    }
  }
  assert(psize <= rsize);
  while (psize > 0 && primes[psize - 1] > N)
--psize;
  primes.resize(psize);
  return primes;
}
```

2.12 smallest number in Seive

```
int spf[N];
void buildSpf() {
  for (int i = 2; i < N; i++) {
    spf[i] = i;
  }
  for (int i = 2; 1LL * i * i < N; i++)
    if (spf[i] == i)
      for (int j = i * i; j < N; j += i)
        if (spf[j] == j) spf[j] = i;}
```

2.14 Modular Operation
```
int add(int x, int y) {
    x += y;
    if (x >= mod) x -= mod;
    if (x < 0) x += mod;
    return x;
}
int mul(int x, int y) {
    return x * 1LL * y % mod;
}
int divide(int x, int y) {
    return mul(x, power(y, mod - 2));
}
```
2.15 nCr(more space, less time)
```
const int N = 1e6 + 5, mod = 1e9 + 7;
int fact[N], ifact[N];

void preFact() {
    fact[0] = 1;
    for (int i = 1; i < N; i++) {
        fact[i] = 1LL * fact[i - 1] * i % 
mod;
    }
    ifact[N - 1] = power(fact[N - 1], mod - 
2);
    for (int i = N - 2; i >= 0; i--) {
        ifact[i] = 1LL * ifact[i + 1] * (i + 
1) % mod;
    }
}
int nCr(int n, int r) {
    if (n < r || n < 0) return 0;
    return 1LL * fact[n] * ifact[r] % mod * 
ifact[n - r] % mod;
}
```

2.16 nCr(less space, more time)
```
const int MOD = 1e9 + 7;
const int MAX = 1e7+10;
vector<int> fact(MAX), inv(MAX);
void factorial(){
    fact[0] = 1;
    for (int i = 1; i < MAX; i++)
        fact[i] = (i * fact[i - 1]) % MOD;
}
binaryExp(int a, int n, int M = MOD); //needs 
to implement
void inverse(){
    for (int i = 0; i < MAX; ++i)
        inv[i] = bigmod(fact[i], MOD - 2);
}
int nCr(int a, int b){
    if (a < b or a < 0 or b < 0)
        return 0;
    int de = (inv[b] * inv[a - b]) % MOD;
    return (fact[a] * de) % MOD;
}
// nCr ends here
int ModInv(int a, int M){ return bigmod(a, M 
- 2, M);}
```
2.17 Factorial mod
```
//n! mod p : Here P is mod value
//For binaryExp we call 1.6 function
int factmod (int n, int p)
{    int res = 1;
    while (n > 1){
        res=(res*binaryExp(p-1,n/p,p))%p;
        for (int i=2; i<=n%p; ++i)
            res=(res*i) %p;
            n /= p;
    }    return int (res % p);
}
```
2.18 Generate combinations
```
// n>=m, choose M numbers from 1 to N.
void combination(int n, int m){
    if (n<m) return ;
    int a[50]= {0};
    int k=0;
    for (int i=1; i<=m; i++) a[i]=i;
    while (true){
        for (int i=1; i<=m; i++)
            cout << a[i] << " ";
        cout << endl;
        k=m;
        while ((k>0) && (n-a[k]==m-k)) k--;
        if (k==0) break;
        a[k]++;
        for (int i=k+1; i<=m; i++)
            a[i]=a[i-1]+1;
    }}
```
2.20 10-ary to m-ary
```
char a[16]={'0','1','2','3','4','5','6','7',
           '8','9','A','B','C','D','E','F'};
string tenToM(int n, int m){
    int temp=n;
    string result="";
    while (temp!=0){
        result=a[temp%m]+result;
        temp/=m;
    }
    return result;
}
```
2.21 m-ary to 10-ary
```
string num = "0123456789ABCDE";
int mToTen(string n, int m){
    int multi=1;
    int result=0;
    for (int i=n.size()-1; i>=0; i--)    {
        result += num.find(n[i])*multi;
        multi*=m;
    }
    return result;
}
```
2.22 Catalan numbers
```
void catalan(int n)
{    int res = 1;
    cout << res << " ";
    for (int i = 1; i < n; i++) {
        res = (res * (4 * i - 2)) / (i + 1);
           cout << res << " ";
    }}
```
2.23 Euler's totient function
```
// the positive integers less than or equal 
to n that are relatively prime to n.
const int N = 5000005;
int phi[N];
unsigned long long phiSum[N];
void phiCalc() {
  for (int i = 1; i < N; i++) phi[i] = i;
  for (int i = 2; i < N; i++) {
      if (phi[i] == i) {
      for (int j = i; j < N; j += i) {
      phi[j] -= phi[j] / i;    }     }  }
  for (int i = 2; i < N; i++) {
      phiSum[i] = (unsigned long long)phi[i] 
* (unsigned long long)phi[i] + phiSum[i - 1];
  }
}
```

2.24 EXT_GCD

```
// return {x,y} such that ax+by=gcd(a,b)
int exgcd(int a, int b, int &x, int &y) {
      if (b == 0) {
      x = 1; y = 0;      return a;    }
      int g = exgcd(b, a % b, y, x);
      y -= a / b * x;
      return g;}
}// bezout's identity: a.x +b.y = gcd(a,b)
```

2.25 Power Set

```
void printPowerSet(char* set, int setSz) {
    // Setsize of power set of a set with
    // setsize. n is (2^n-1)
    unsigned int powSetSz = pow(2, setSz);
    int i, j; // i as counter
    // Run from i 000..0 to 111..1
    for (i = 0; i < powSetSz; i++) {
        for (j = 0; j < setSz; j++) {
    //Check if jth bit in the counter is set
    //If set then print jth element from set
            if (i & (1 << j))cout << set[j];
        }
        cout << endl;
    }
}
```

2.26 Sum of divisors

```
long long SumOfDivisors(long long num){
    ll total = 1;
    for (ll i = 2; i * i <= num; i++){
        if (num % i == 0){
            int e = 0;
            do{e++;num /= i;
            } while (num % i == 0);
            long long sum = 0, pow = 1;
            do{sum += pow;pow *= i;
            } while (e-- > 0);
            total *= sum;
        }
    }
    if (num > 1)total *= (1 + num);
    return total;
}
```

2.27 Large Mod

```
int mod(string& num, int a){
    int res = 0;
    for (int i = 0; i < num.length(); i++)
        res = (res * 10 + num[i] - '0') % a;
     return res;
}
```

2.28 divisor of n!

```
ull factorialDivisors(ull n)
{    ull result = 1;
    for (int i=0; i < allPrimes.size(); i++)
    { ull p = allPrimes[i]; ull exp = 0;
        while (p <= n) { exp = exp + (n/p);
            p = p*allPrimes[i]; }
        result = result*(exp+1);  }    return
result;}
```

2.29 Find numbers in between [L, R] which are divisible by all Array elements

```
void solve(int* arr, int N, int L, int R)
{ int LCM = arr[0];
    for (int i = 1; i < N; i++) {
        LCM = (LCM * arr[i]) /
            (__gcd(LCM, arr[i]));    }
    if ((LCM < L && LCM * 2 > R) || LCM > R)
{        return;    }
    int k = (L / LCM) * LCM;
    if (k < L) k = k + LCM;
    for (int i = k; i <= R; i = i + LCM)
        cout << i << ' ';
}
```

2.30 Disarrangement Formula

```
int disarrange(int n) {
  if (n == 1) return 0;  if (n == 2) return
1;
  return (n - 1) * (disarrange(n - 1) +
disarrange(n - 2));}
//D(n) = (n!)/e
```

2.31 MSLCM

```
//For a given number N, maximum sum LCM
indicates the set of numbers whose LCM is N
and summation is maximum. Let, MSLCM(N)
denote this maximum sum of numbers. Given the
value of N you will have to find the value:
∑(i=2->n) MSLCM(i)
long long MSLCM(long long n) {
      long long l = 1, r, val;
      long long ret = 0;
      while (l <= n) {
            val = n / l,r = n / val;
            ret += val * ((l+r)*(r-l+1)/2);
            l = r+1;}
      return ret-1;}
```

2.33 Count 1's from 0 to n

```
int cntOnes(int n) {  int cnt = 0;  for(int
i=1;i<=n;i<<=1) {
      int x = (n + 1) / (i << 1);
      cnt += x * i;      if((n + 1) % i && n
& i) cnt += (n + 1) % i;
  }  return cnt;}
```

2.34 Millar Rabin

```
using u64 = uint64_t;
using u128 = __uint128_t;
u64 binpower(u64 base, u64 e, u64 mod) {
  u64 result = 1;
  base %= mod;
  while (e) {
      if (e & 1) result = (u128)result * base
% mod;
      base = (u128)base * base % mod;
      e >>= 1;  }
  return result;}
bool check_composite(u64 n, u64 a, u64 d, int
s) {
  u64 x = binpower(a, d, n);
  if (x == 1 || x == n - 1) return false;
  for (int r = 1; r < s; r++) {
      x = (u128)x * x % n;
      if (x == n - 1) return false;  }
  return true;};
bool MillerRabin(u64 n, int iter = 5) {  //
returns true if n is probably prime, else
returns false.
  if (n < 4) return n == 2 || n == 3;
  int s = 0;
  u64 d = n - 1;
  while ((d & 1) == 0) {
      d >>= 1;
      s++;   }
  for (int i = 0; i < iter; i++) {
      int a = 2 + rand() % (n - 3);
      if (check_composite(n, a, d, s)) return
false;
  }
  return true;
}
```

3 Algorithms

3.1 Biginteger Operation

```
struct BigInteger {
    string str;
    // Constructor to initialize
    // BigInteger with a string
    BigInteger(string s) { str = s; }
    // Overload + operator to add
    // two BigInteger objects
    BigInteger operator+(const BigInteger& b)
{
        string a = str, c = b.str;
        int alen=a.length(),clen=c.length();
        int n = max(alen, clen);
        if (alen > clen)
            c.insert(0, alen - clen, '0');
        else if (alen < clen)
            a.insert(0, clen - alen, '0');
        string res(n + 1, '0');
        int carry = 0;
        for (int i = n - 1; i >= 0; i--) {
            int digit=(a[i -'0')+(c[i]-'0')
                      +carry;
            carry = digit / 10;
            res[i + 1] = digit % 10 + '0';
        }
        if (carry == 1) {
            res[0] = '1';
            return BigInteger(res);
        }
        else
           return BigInteger(res.substr(1));
    }

    // Overload - operator to subtract
    // first check which number is greater
and then subtract
    BigInteger operator-(const BigInteger& b)
{
        string a = str;
        string c = b.str;
        int alen=a.length(),clen=c.length();
        int n = max(alen, clen);
        if (alen > clen)
            c.insert(0, alen - clen, '0');
        else if (alen < clen)
            a.insert(0, clen - alen, '0');
        if (a < c) {
            swap(a, c);
            swap(alen, clen);
        }
        string res(n, '0');
        int carry = 0;
        for (int i = n - 1; i >= 0; i--) {
            int digit =(a[i]-'0')-(c[i]-'0')
                       - carry;
            if (digit<0) digit+=10,carry=1;
            else carry = 0;
            res[i] = digit + '0';
        }
        // remove leading zeros
        int i = 0;
        while (i < n && res[i] == '0')i++;
        if (i == n)
            return BigInteger("0");
        return BigInteger(res.substr(i));
    }

    // Overload * operator to multiply
    // two BigInteger objects
    BigInteger operator*(const BigInteger& b)
{
        string a = str, c = b.str;
        int alen=a.length(),clen=c.length();
        int n = alen + clen;
        string res(n, '0');
        for (int i = alen - 1; i >= 0;i--) {
            int carry = 0;
            for(int j=clen-1; j>=0; j--) {
                int digit = (a[i] - '0') *
           (c[j-'0')+(res[i+j+1]-'0')+carry;
                carry = digit / 10;
                res[i+j+1]=digit % 10 + '0';
            }
            res[i] += carry;
        }
        int i = 0;
        while (i < n && res[i] == '0')
            i++;
        if (i == n)
            return BigInteger("0");
        return BigInteger(res.substr(i));
    }

    // Overload << operator to output
    // BigInteger object
    friend ostream& operator<<(ostream& out,
const BigInteger& b) {
        out << b.str;
        return out;
    }};
```

3.2 Find rank k in array

```
int find(int l, int r, int k){
    int i=0,j=0,x=0,t=0;
    if (l==r) return a[l];
    x=a[(l+r)/2];
    t=a[x];
    a[x]=a[r];
    a[r]=t;
    i=l-1;
    for (int j=l; j<=r-1; j++)
        if (a[j]<=a[r]){
            i++;
            t=a[i];
            a[i]=a[j];
            a[j]=t;
        }
    i++;
    t=a[i]; a[i]=a[r]; a[r]=t;
    if (i==k) return a[i];
    if (i<k) return find(i+1, r,k);
    return find(l, i-1, k);
}
```

3.3 InfixToPostFix

```
bool delim(char c) { return c == ' '; }
bool is_op(char c) {
    return c == '+' || c == '-' || c == '*'
```

```
           || c == '/' || c == '^';
}
bool is_unary(char c) {
      return c == '+' || c == '-';
}
int priority(char op) {
    if (op < 0) return 3;
    if (op == '+' || op == '-') return 1;
    if (op == '*' || op == '/') return 2;
    if (op == '^') return 4;
    return -1;
}

void process_op(string& output, char op) {
    if (op < 0) {
        switch (-op) {
            case '+':
                output += "+ ";
                break;
            case '-':
                output += "- ";
                break;
        }
    }
    else {
        switch (op) {
            case '+':
                output += "+ ";
                break;
            case '-':
                output += "- ";
                break;
            case '*':
                output += "* ";
                break;
            case '/':
                output += "/ ";
                break;
            case '^':
                output += "^ ";
                break;
        }
    }
}

string InfixToPostFix(string& s) {
    string output;
    stack<char> op;
    bool may_be_unary = true;
    for (int i = 0; i <(int)s.size(); i++){
        if (delim(s[i]))
            continue;
        if (s[i] == '(') {
            op.push('(');
            may_be_unary = true;
        }
        else if (s[i] == ')') {
            while (op.top() != '(') {
               process_op(output, op.top());
               op.pop();
            }
            op.pop();
            may_be_unary = false;
        }
        else if (is_op(s[i])) {
            char cur_op = s[i];
            if (may_be_unary &&
is_unary(cur_op))
                cur_op = -cur_op;
            while (!op.empty() &&
                    ((cur_op >= 0 &&
priority(op.top()) >= priority(cur_op)) ||
                    (cur_op < 0 &&
priority(op.top()) > priority(cur_op)))) {
                process_op(output, op.top());
                op.pop();
            }
            op.push(cur_op);
            may_be_unary = true;
        }
        else {
            char number;
            while (i < (int)s.size() &&
isalnum(s[i]))
                number = s[i++];
            --i;
            output.push_back(number);
            output.push_back(' ');
            may_be_unary = false;
        }
    }
    while (!op.empty()) {
        process_op(output, op.top());
        op.pop();
    }
    return output;
}
```

3.4 Expression Parsing

```
bool delim(char c) { return c == ' '; }

bool is_op(char c) { return c == '+' || c ==
'-' || c == '*' || c == '/'; }

bool is_unary(char c) { return c == '+' || c
== '-'; }

int priority(char op) {
    if (op < 0)  // unary operator
        return 3;
    if (op == '+' || op == '-')
        return 1;
    if (op == '*' || op == '/')
        return 2;
    return -1;
}

void process_op(stack<int>& st, char op) {
    if (op < 0) {
        int l = st.top();
        st.pop();
        switch (-op) {
            case '+':
                st.push(l);
                break;
            case '-':
                st.push(-l);
                break;
        }
    }
    else {
        int r = st.top();
        st.pop();
        int l = st.top();
        st.pop();
        switch (op) {
            case '+':
                st.push(l + r);
                break;
            case '-':
                st.push(l - r);
                break;
            case '*':
                st.push(l * r);
                break;
            case '/':
                st.push(l / r);
                break;
        }
    }
}

int evaluate(string& s) {
    stack<int> st;
    stack<char> op;
    bool may_be_unary = true;
    for (int i = 0; i < (int)s.size(); i++) {
        if (delim(s[i]))
            continue;

        if (s[i] == '(') {
            op.push('(');
            may_be_unary = true;
        }
        else if (s[i] == ')') {
            while (op.top() != '(') {
                process_op(st, op.top());
                op.pop();
            }
            op.pop();
            may_be_unary = false;
        }
        else if (is_op(s[i])) {
            char cur_op = s[i];
            if (may_be_unary &&
is_unary(cur_op))
                cur_op = -cur_op;
            while (!op.empty() &&
                   ((cur_op >= 0 &&
priority(op.top()) >= priority(cur_op)) ||
                    (cur_op < 0 &&
priority(op.top()) > priority(cur_op)))) {
                process_op(st, op.top());
                op.pop();
            }
            op.push(cur_op);
            may_be_unary = true;
        }
        else {
            int number = 0;
            while (i < (int)s.size() &&
isalnum(s[i]))
                number = number * 10 + s[i++]
- '0';
            --i;
            st.push(number);
            may_be_unary = false;
        }
    }

    while (!op.empty()) {
        process_op(st, op.top());
        op.pop();
    }
    return st.top();
}
```

3.5 2D prefix sum

```
pref[i][j] = a[i][j] + pref[i - 1][j] +
pref[i][j - 1] - pref[i - 1][j - 1];
Sum of region = pref[row2][col2] -
pref[row2][col1 - 1] - pref[row1 - 1][col2] +
pref[row1 - 1][col1 - 1];
```

3.6 KMP Algorithm-O(n+m)

```
vector<int> prefix_function(string s) {
        int n = (int)s.length();
        vector<int> pi(n);
        for (int i = 1; i < n; i++) {
        int j = pi[i - 1];
        while (j > 0 && s[i] != s[j]) j = pi[j
- 1];
        if (s[i] == s[j]) j++;
        pi[i] = j;
        }
        return pi;
}

vector<int> find_matches(string text, string
pat) {
        int n = pat.length(), m =
text.length();
        string s = pat + "$" + text;
        vector<int> pi = prefix_function(s),
ans;
        for (int i = n; i <= n + m; i++) {
        if (pi[i] == n) {
                ans.push_back(i - 2 * n);
        }
        }
        return ans;
}
```

3.7 Kadane's Algorithm O(n)

```
// return maximum subarray sum.
int kadense(int arr[], int n)
{
    int maxsum = arr[0], curr_sum = arr[0];
    for (int i = 1; i < n; i++)
    {
        curr_sum = max(arr[i], curr_sum +
arr[i]);
        maxsum = max(maxsum, curr_sum);
    }
    return maxsum;}
```

4 Data Structure

4.1 SEGMENT TREE

```
constexpr int N = 100005;
i64 arr[N], seg[4 * N];
i64 combine(i64 l, i64 r) {
    return l + r;
}
void build(int ind, int low, int high) {
  if (low == high) {
    seg[ind] = arr[low];
    return;
  }
  int mid = (low + high) / 2;
  build(2 * ind, low, mid);
  build(2 * ind + 1, mid + 1, high);
  seg[ind] = combine(seg[2 * ind], seg[2 *
ind + 1]);
}
i64 query(int ind, int low, int high, int l,
int r) {
  if (low >= l && high <= r) return seg[ind];
  if (low > r || high < l) return 0; // check
it!
  int mid = (low + high) / 2;
  i64 left = query(2 * ind, low, mid, l, r);
  i64 right = query(2 * ind + 1, mid + 1,
high, l, r);
  return combine(left, right);
}
void update(int ind, int low, int high, int
node, int val) {
  if (low == high) {
    seg[ind] = val;
    return;
  }
  int mid = (low + high) / 2;
  if (low <= node && node <= mid) update(2 *
ind, low, mid, node, val);
  else update(2 * ind + 1, mid + 1, high,
node, val);
  seg[ind] = combine(seg[2 * ind], seg[2 *
ind + 1]);
}
```

4.2 FENWICK TREE

```
int fenwick[N];
void update(int ind, int val) {
  while (ind < N) {
    fenwick[ind] += val;
    ind += ind & -ind;
  }
}
int query(int ind) {
  int sum = 0;
  while (ind > 0) {
    sum += fenwick[ind];
    ind -= ind & -ind;
  }
  return sum;
}
// 2D fenwick
int fenwick[N][N];
void update(int x, int y, int val) {
    for (int i = x; i < N; i += i & -i)
        for (int j = y; j < N; j += j & - j)
            fenwick[i][j] += val;
}
int query(int x, int y) {
    int sum = 0;
    for (int i = x; i > 0; i -= i & -i)
        for (int j = y; j > 0; j -= j & -j)
            sum += fenwick[i][j];
    return sum;
}
```

4.3 SEGMENT TREE LAZY

```
const int N = 1e5 + 5;
i64 arr[N], seg[N << 2], lz[N << 2];
i64 combine(i64 l, i64 r) {
    return l + r;
}
void push(int node, int l, int r) {
    if (lz[node] == 0) return;
    int mid = (l + r) >> 1;
    lz[node << 1] += lz[node];
    seg[node << 1] += lz[node] * (mid - l +
1);
    lz[node << 1 | 1] += lz[node];
    seg[node << 1 | 1] += lz[node] * (r -
mid);
    lz[node] = 0;
}
void build(int node, int l, int r) {
    if(l == r) {
        seg[node] = arr[l];
        lz[node] = 0;
        return;
    }
    int mid = (l + r) >> 1;
    build(node << 1, l, mid);
    build(node << 1 | 1, mid + 1, r);
    seg[node] = combine(seg[node << 1],
seg[node << 1 | 1]);
}
void update(int node, int l, int r, int ql,
int qr, int val) {
    if(qr < l || r < ql) return;
    if(ql <= l && r <= qr) {
        seg[node] += 1LL * val * (r - l + 1);
// check it!
        lz[node] += val;
        return;
    }
    push(node, l, r);
    int mid = (l + r) >> 1;
    update(node << 1, l, mid, ql, qr, val);
    update(node << 1 | 1, mid + 1, r, ql, qr,
val);
    seg[node] = combine(seg[node << 1],
seg[node << 1 | 1]);
}
i64 query(int node, int l, int r, int ql, int
qr) {
    if(qr < l || r < ql) return 0; // check
it!
    if(ql <= l && r <= qr) return seg[node];
    push(node, l, r);
    int mid = (l + r) >> 1;
    return combine(query(node << 1, l, mid,
ql, qr), query(node << 1 | 1, mid + 1, r, ql,
qr));
}
```

4.5 TRIE

```
const int N = 26;
char BASE = 'A'
class Node {
    public:
        int EoW;
        Node* child[N];
        Node() {
            EoW = 0;
            for (int i = 0; i < N; i++)
child[i] = NULL;
        }
};
Node *root = new Node();

void insert(Node* node, string s) {
    for (size_t i = 0; i < s.size(); i++) {
        int r = s[i] - BASE;
        if (node->child[r] == NULL)
node->child[r] = new Node();
        node = node->child[r];
    }
    node->EoW += 1;
} // insert(root, s);

int search(Node* node, string s) {
    for (size_t i = 0; i < s.size(); i++) {
        int r = s[i] - BASE;
        if (node->child[r] == NULL) return 0;
        node = node->child[r];
    }
    return node->EoW;
} // search(root, s);

void print(Node* node, string s = "") {
    if (node->EoW) cout << s << "\n";
    for (int i = 0; i < N; i++) {
        if (node->child[i] != NULL) {
            char c = i + BASE;
            print(node->child[i], s + c);
        }
    }
} // print whole trie

void remove(Node* node, string s) {
    for (size_t i = 0; i < s.size(); i++) {
        int r = s[i] - base;
        if (node->child[r] == NULL) return;
        node = node->child[r];
    }
    node->EoW--;
} // remove(root, s)

void delete_trie(Node* node) {
    for (int i = 0; i < N; i++) {
        if (node->child[i] != NULL)
delete_trie(node->child[i]);
    }
    delete node;
} // delete full trie
```

4.5 TRIE BIT

```
const int BT = 32;
class Node {
  public: Node * child[2];
  int cnt;
  Node() {
    cnt = 0;
    for (int i = 0; i < 2; i++) child[i] =
NULL;
  }
};
Node * root = new Node();

void insert(Node * node, int x) {
  for (int i = BT - 1; i >= 0; i--) {
    int r = (x >> i) & 1 LL;
    if (node -> child[r] == NULL) node ->
child[r] = new Node();
    node = node -> child[r];
    node -> cnt++;
  }
} // insert(root, x);

int query(Node * node, int x) {
  int mx = 0;
  for (int i = BT - 1; i >= 0; i--) {
    int r = (x >> i) & 1 LL;
    if (node -> child[r ^ 1] == NULL || node
-> child[r ^ 1] -> cnt == 0) {
      if (node -> child[r] == NULL) return
mx;
      node = node -> child[r];
    } else {
      mx |= (1 << i);
      node = node -> child[r ^ 1];
    }
  }
  return mx;
} // query(root, x), get max xor

void remove(Node * node, int x) {
  for (int i = BT - 1; i >= 0; i--) {
    int r = (x >> i) & 1 LL;
    if (node -> child[r] == NULL) return;
    node = node -> child[r];
    node -> cnt--;
  }
} // remove(root, x)
void clearTrie(Node * node) {
  if (node == nullptr) return;
  clearTrie(node -> child[0]);
  clearTrie(node -> child[1]);
  delete node;
}
// clearTrie(root), delete full trie
// root = new Node()
```

4.6 DSU

```
class DisjointSet{
    vector<int> par, sz, minElmt, maxElmt,
cntElmt;

   public:
    DisjointSet(int n){
        par.resize(n + 1);
        sz.resize(n + 1, 1);
        minElmt.resize(n + 1);
        maxElmt.resize(n + 1);
        cntElmt.resize(n + 1, 1);
        for (int i = 1; i <= n; i++)
            par[i]=minElmt[i]=maxElmt[i]=i;
    }
    int findUPar(int u) {
        if (u == par[u])
            return u;
        return par[u] = findUPar(par[u]);
    }
    void unionBySize(int u, int v){
        int pU = findUPar(u);
        int pV = findUPar(v);
        if (pU == pV)
            return;
        if (sz[pU] < sz[pV])
            swap(pU, pV);
        par[pV] = pU;
        sz[pU] += sz[pV];
        cntElmt[pU] += cntElmt[pV];
        minElmt[pU] = min(minElmt[pU],
                          minElmt[pV]);
        maxElmt[pU] = max(maxElmt[pU],
                          maxElmt[pV]);
    }
    int getMinElementIntheSet(int u){
        return minElmt[findUPar(u)];
    }
    int getMaxElementIntheSet(int u){
        return maxElmt[findUPar(u)];
    }
    int getNumofElementIntheSet(int u){
        return cntElmt[findUPar(u)];
    }
};
```

5 Dynamic Programming

5.0 Knapsack

```
int n, W, w[N], v[N], dp[N][100005];
int rec(int i, int weight)
```

```
{    if (i == n + 1) return 0;
    if (dp[i][weight] != -1) return
dp[i][weight]; int ans = rec(i + 1, weight);
    if (weight + w[i] <= W)  ans = max(ans,
rec(i + 1, weight + w[i]) + v[i]);
    return dp[i][weight] = ans;}
```

5.1 LCS O(n*m)

```
 //LCS DP Table and LCS Length

vector<vector<int>>dp(s.size()+1,vector<int>(
t.size()+1));
    for(int i=1;i<=s.size();i++){
        for(int j=1;j<=t.size();j++){

if(s[i-1]==t[j-1])dp[i][j]=dp[i-1][j-1]+1;
            else
dp[i][j]=max(dp[i-1][j],dp[i][j-1]);
        }
    }
    cout<<dp[s.size()][t.size()]<<"\n";

    //Any LCS String
    string a;
    int i=s.size();
    int j=t.size();
    while(i>0 && j>0){
        if(s[i-1]==t[j-1]){
            a+=s[i-1];
            i--,j--;
        }
        else if(dp[i][j-1]>dp[i-1][j])j--;
        else i--;
    }
    reverse(a.begin(),a.end());
    cout<<a<<"\n";

    //Lexicographically smallest LCS

vector<vector<string>>dpp(s.size()+1,vector<s
tring>(t.size()+1));
    for(int i=1;i<=s.size();i++){
        for(int j=1;j<=t.size();j++){

if(s[i-1]==t[j-1])dpp[i][j]=dpp[i-1][j-1]+s[i
-1];
            else
if(dp[i][j-1]<dp[i-1][j])dpp[i][j]=dpp[i-1][j
];
            else
if(dp[i][j-1]>dp[i-1][j])dpp[i][j]=dpp[i][j-1
];
            else
dpp[i][j]=min(dpp[i][j-1],dpp[i-1][j]);
        }
    }
    cout<<dpp[s.size()][t.size()]<<"\n";

    //number of distinct LCS sequences
    int MOD=1e9+7;

vector<vector<int>>cnt(s.size()+1,vector<int>
(t.size()+1,1));
    for(int i=1;i<=s.size();i++){
        for(int j=1;j<=t.size();j++){

if(s[i-1]==t[j-1])cnt[i][j]=cnt[i-1][j-1];
            else
if(dp[i][j-1]<dp[i-1][j])cnt[i][j]=cnt[i-1][j
];
            else
if(dp[i][j-1]>dp[i-1][j])cnt[i][j]=cnt[i][j-1
];
            else{

cnt[i][j]=cnt[i-1][j]+cnt[i][j-1];

if(dp[i][j]==dp[i-1][j-1])cnt[i][j]-=cnt[i-1]
[j-1];

cnt[i][j]=(cnt[i][j]+MOD)%MOD;
            }   }   }
```

5.2 MCM O(n^3)

```
const int N = 1005;
vector<int> v;
int dp[N][N], mark[N][N];
int MCM(int i, int j) {
    if (i == j)
        return dp[i][j] = 0;
    if (dp[i][j] != -1)
        return dp[i][j];
    int mn = INT_MAX;
    for (int k = i; k < j; k++) {
        int x = mn;
        mn = min(mn, MCM(i, k) + MCM(k + 1,
j) + v[i - 1] * v[k] * v[j]);
        if (x != mn)
            mark[i][j] = k;
    }
    return dp[i][j] = mn;
}

void print_order(int i, int j) {
    if (i == j)
        cout << "X" << i;
    else {
        cout << "(";
        print_order(i, mark[i][j]);
        print_order(mark[i][j] + 1, j);
        cout << ")";
    }
}
// memset(dp, -1, sizeof dp);
// print_order(1, n);
```

5.3 Length of LIS O(nlogn)

```
vector<int> v = {7, 3, 5, 3, 6, 2, 9, 8};
vector<int> seq;
for (auto i : v) {
    auto id = lower_bound(seq.begin(),
seq.end(), i);
    if (id == seq.end())
        seq.push_back(i);
    else
        seq[id - seq.begin()] = i;
}
cout << seq.size() << endl;
```

5.4 LCIS O(n * m)

```
int a[100]= {0}, b[100]= {0}, f[100]= {0};
int n=0, m=0;
int main(void){
    cin >> n;
    for (int i=1; i<=n; i++) cin >> a[i];
    cin >> m;
    for (int i=1; i<=m; i++) cin >> b[i];
    for (int i=1; i<=n; i++){
        int k=0;
        for (int j=1; j<=m; j++){
            if (a[i]>b[j] && f[j]>k)
                  k=f[j];
            else if (a[i]==b[j] && k+1>f[j])
                  f[j]=k+1;
        }
    }
    int and=0;
    for (int i=1; i<=m; i++)
        if (f[i]>ans) ans=f[i];
    cout << and << endl;
    return 0;
}
```

5.5 Maximum submatrix

```
int a[150][150]= {0};
int c[200]= {0};
int maxarray(int n){
    int b=0, sum=-100000000;
    for (int i=1; i<=n; i++){
        if (b>0) b+=c[i];
        else b=c[i];
        if (b>sum) sum=b;
    }
    return sum;
}

int maxmatrix(int n){
    int sum=-100000000, max=0;
    for (int i=1; i<=n; i++){
        for (int j=1; j<=n; j++)
            c[j]=0;
        for (int j=i; j<=n; j++){
            for (int k=1; k<=n; k++)
                c[k]+=a[j][k];
            max=maxarray(n);
            if (max>sum) sum=max;
        }
    }
    return sum;
}
int main(void){
    int n=0;
    cin >> n;
    for (int i=1; i<=n; i++)
        for (int j=1; j<=n; j++)
            cin >> a[i][j];
    cout << maxmatrix(n);
    return 0;
}
```

5.6 SOS DP

```
void SOS (vector<int> &v) {
    const int BITS =
log2(*max_element(v.begin(), v.end())) + 1;
    vector<int> freq(1 << BITS, 0);
    for (int mask : v) freq[mask]++;

    vector<vector<int>> dp(BITS + 1,
vector<int> (1 << BITS, 0));
    for (int mask = 0; mask < 1 << BITS;
mask++) {
        dp[0][mask] = freq[mask];
    }

    for (int bits = 1; bits <= BITS; bits++)
{
        for (int mask = 0; mask < 1 << BITS;
mask++) {
            if ((mask & (1 << (bits - 1))) ==
0) {
                dp[bits][mask] = dp[bits -
1][mask];
            }
            else {
                int other_mask = mask - (1 <<
(bits - 1));
                dp[bits][mask] = dp[bits -
1][mask] + dp[bits - 1][other_mask];
            }
        }
    }

    for (int mask : v) cout << dp[BITS][mask]
<< '\n';
}
// dp[bits][mask] means left most 'bits' of
submasks can be differ
// for dp[1][11] 01, 00 are not allow because
leftmost 1 bit can be differ. 10 and 11 are
allowed.
// for travarsing all the submask of a mask
we can use
// submask = mask
// do {
// submask = (submask - 1) & mask
// } while (submask)
}
```

5.7 All possible SubArraySum in O(1)

```
bitset<100005> bs = 1;
    for (auto i : a)
    {
        bs |= (bs << i); // if previous 1
value pos is possible now ith bit or ith sm
is also possible
    }
    cout << bs.count() - 1 << endl;
    for (int i = 1; i <= 100003; i++)
        if (bs[i])
            cout << i << " ";
    cout << endl;
```

5.8 Range DP

```
int f(int st,int ed){
    if(st>ed)return 0;
    if(st==ed)return dp[st][ed]=1;
    if(dp[st][ed]!=-1)return dp[st][ed];

    int cnt=1+f(st+1,ed);
    for(int i=st+1;i<=ed;i++){
        if(a[st]==a[i]){
            cnt=min(cnt,f(st+1,i-1)+f(i,ed));
        }    }
    return dp[st][ed]=cnt;}
```

5.9 Bitmask DP

```
check: (mask & (1<<i))
set: (mask | (1<<i))
clear: if(ith bit 1) → (mask - (1<<i))
int n,mod=1e9+7;
vector<vector<int>>a(21,vector<int>(21));
vector<vector<int>>dp(21,vector<int>((1<<21),
-1));
int f(int m,int mask){
    if(m==n)return 1;
    if(dp[m][mask]!=-1)return dp[m][mask];
    int cnt=0;
    for(int i=0;i<n;i++){
        if(a[m][i] && !(mask&(1<<i))){
            cnt+=f(m+1,(mask|(1<<i)));
            cnt%=mod;
        }     }
    return dp[m][mask]=cnt;}
//call by f(0,0)
```

5.10 CHT

```
#define x first
#define y second
// Warning: replace __int128 with long double
if using floating point lines or if unsure
about the compiler
// Queries maximum by default, pass false in
constructor to query minimum
// Line = pair (m, c) for a line y = mx + c
// Make sure lines are inserted in
non-decreasing order for maximum queries, and
non-increasing order for minimum queries
template<typename T, class Line = pair<T, T>>
// Make sure multiplications fit in T
class CHT {
private:
  vector<Line> lines;
  bool maximum;
public:
  CHT(bool maximum = true) { this->maximum =
maximum; }
  void addLine(Line line) {
    if (!maximum) line.first = -line.first,
line.second = -line.second;
    insert(line);
  }
  T query(const T &x) {
    assert(!lines.empty());

    int L = 0, R = lines.size() - 1;
    while (L != R) {
      int mid1 = L + (R - L) / 3;
      int mid2 = R - (R - L) / 3;
      if (lines[mid1].x * x + lines[mid1].y
          >= lines[mid2].x * x +
lines[mid2].y
      ) R = mid2 - 1;
      else L = mid1 + 1;
    }

    T res = lines[L].x * x + lines[L].y;
    return maximum ? res : -res;
  }
private:
  bool bad(const Line &line1, const Line
&line2, const Line &line3) {
    return __int128(line3.y - line1.y) *
__int128(line1.x - line2.x)
           <= __int128(line2.y - line1.y) *
__int128(line1.x - line3.x);
  }
  void insert(const Line &line) {
    while (lines.size() > 0 && lines.back().x
== line.x) lines.pop_back();
    lines.push_back(line);
    int sz = lines.size();
    while (sz >= 3 && bad(lines[sz - 3],
lines[sz - 2], lines[sz - 1])) {
      lines.erase(lines.end() - 2);
      sz--;
    }
  }
};
```

5.11 Digit Dp

```
int f(int pos, int up, int under, int nz) {
    if (pos == n) return 0;
    int &ans = dp[pos][under][up][nz];
    if (ans != -1) return ans;
    int ans = 0;
    int start = up ? 0 : s[pos] - '0';
    int end = under ? 9 : t[pos] - '0';
    for (int d = start; d <= end; d++) {
        int n_up = up || (d > s[pos] - '0');
        int n_under = under || (d < t[pos] -
'0');
        int n_nz = nz || (d != 0);
        ans += f(pos + 1, n_under, n_up,
n_nz);
    }
    return ans;
}
```

6 Graph Theory

6.1 SPFA – Optimal BF O(V * E)

```
int q[3001]= {0};// queue for node
it d[1001]= {0}; // record shortest path
from start to ith node
bool f[1001]= {0};
int a[1001][1001]= {0}; // adjacency list
int w[1001][1001]= {0}; // adjacency matrix
int main(void) {
  int n=0, m=0;
  cin >> n >> m;
  for (int i=1; i<=m; i++){
  int x=0, y=0, z=0;
cin >> x >> y >> z;
// node x to node y has weight z
a[x][0]++;
 a[x][a[x][0]]=y;
  w[x][y]=z;
/*
// for undirected graph
 a[x][0]++;
 a[y][a[y][0]]=x;
w[y][x]=z;
*/
}
int s=0, e=0;
cin >> s >> e; // s: start, e: end
SPFA(s);
cout << d[e] << endl;
return 0;
}
void SPFA(int v0){
    int t,h,u,v;
    for (int i=0; i<1001; i++) d[i]=INT_MAX;
    for (int i=0; i<1001; i++) f[i]=false;
    d[v0]=0;
    h=0;
    t=1;
    q[1]=v0;
    f[v0]=true;
    while (h!=t){
        h++;
        if (h>3000) h=1;
        u=q[h];
        for (int j=1; j<=a[u][0]; j++){
            v=a[u][j];
            if (d[u]+w[u][v]<d[v]) // change
to > if calculating longest path
            {
                d[v]=d[u]+w[u][v];
                if (!f[v]){
                    t++;
                    if (t>3000) t=1;
                    q[t]=v;
                    f[v]=true;
                }
            }
        }
        f[u]=false;
    }
}
```

6.2 Dijkstra O(V + ElogV)

```
typedef pair<int, int> pairi;
int N = 20000 + 5;
vector<vector<pairi>> adj(N);
vector<int> dis(N, inf), parent(N);

void dijkstra(int src) {
    priority_queue<pairi, vector<pairi>,
                   greater<pairi>> pq;
    dis[src] = 0;
    pq.push({0, src});
    while (pq.size()) {
        auto top = pq.top();
        pq.pop();
        for (auto i : adj[top.second]) {
            int v = i.first;
            int wt = i.second;
            if (dis[v]>dis[top.second]+wt) {
                dis[v]=dis[top.second]+wt;
                pq.push({dis[v], v});
                parent[v] = top.second
            }
        }
    }
}
int node = n;
while (parent[node] != node){
  path.push_back(node);
  node = parent[node]; }
path.push_back(1);
```

6.3 BellmanFord O(V.E)

```
vector<int> dist;
vector<int> parent;
vector<vector<pair<int, int>>> adj;
// resize the vectors from main function

void bellmanFord(int num_of_nd, int src) {
    dist[src] = 0;
    for (int step=0;step<num_of_nd;step) {
        for (int i = 1; i<=num_of_nd; i++) {
            for (auto it : adj[i]) {
                int u = i;
                int v = it.first;
                int wt = it.second;
                if (dist[u] != inf &&
               ((dist[u] + wt) < dist[v])) {
                    if(step==num_of_nd - 1){
                        cout << "Negative
                              cycle found\n";
                        return;
                    }
                    dist[v] = dist[u] + wt;
                    parent[v] = u;
                } } } }
    for (int i = 1; i <= num_of_nd; i++)
        cout << dist[i] << " ";
    cout << endl;
}
```

6.4 Floyd-Warshall algorithm O(n^3)

```
typedef double T;
typedef vector<T> VT;
typedef vector<VT> VVT;

typedef vector<int> VI;
typedef vector<VI> VVI;

bool FloydWarshall (VVT &w, VVI &prev){
  int n = w.size();
  prev = VVI (n, VI(n, -1));

  for (int k = 0; k < n; k++){
    for (int i = 0; i < n; i++){
      for (int j = 0; j < n; j++){
        if (w[i][j] > w[i][k] + w[k][j]){
          w[i][j] = w[i][k] + w[k][j];
          prev[i][j] = k;}}}}

  // check for negative weight cycles
  for(int i=0;i<n;i++)
    if (w[i][i] < 0) return false;
  return true;
}
```

6.5 Topological sort

```
map<string, vector<string>> adj;
map<string, int> degree;
set<string> nodes;
vector<string> ans;
// adj: graph input, degree: cnt indegree,
// node: unique nodes, ans: path
int c = 0;
void topo_sort() {
    queue<string> qu;
// traverse all the nodes and check if its
degree is 0 or not..
    for (string i : nodes) {
        if (degree[i] == 0) qu.push(i);
    }
    while (!qu.empty()) {
        string top = qu.front();
        qu.pop();
        ans.push_back(top);
        for (string i : adj[top]) {
            degree[i]--;
            if (degree[i] == 0) {
                qu.push(i);}}}}
```

6.6 Kruskal O(ElogE)

```
vector<pair<int, pair<int, int>>>
Krushkal(vector<pair<int, pair<int, int>>>
&edges, int n) {
  sort(edges.begin(), edges.end());
  vector<pair<int, pair<int, int>>> ans;
  DisjointSet D(n);
  for (auto it : edges) {
    if (D.findUPar(it.second.first) !=
D.findUPar(it.second.second)) {
      ans.push_back({it.first,
{it.second.first, it.second.second}});
      D.unionBySize(it.second.first,
it.second.second);
    }
  }
  return ans;
}
```

6.7 Prim – MST O(ElogV)

```
typedef pair<int, int> pii;

class Prims {
    map<int, vector<pii>> graph;
    map<int, int> visited;

   public:
    void addEdge(int u, int v, int w) {
        graph[u].push_back({v, w});
        graph[v].push_back({u, w});
    }

    vector<int> path(pii start) {
       vector<int> ans;
       priority_queue<pii, vector<pii>,
                        greater<pii>> pq;
                        // cost vs node
       pq.push({start.second, start.first});
       while (!pq.empty()) {
            pair<int, int> curr = pq.top();
            pq.pop();
            if (visited[curr.second])
                continue;
            visited[curr.second] = 1;
            ans.push_back(curr.second);
            for (auto i:graph[curr.second]){
               if (visited[i.first])
                   continue;
               pq.push({i.second, i.first});
            }       }
       return ans;
    }};
```

6.8 Eulerian circuit O(V+E)

```
unordered_map<int, int> Start, End, Val;
unordered_map<int, pair<int, int>> Range;
int start = 0;
void dfs(int node){
    visited[node] = true;
    Start[node] = start++;
    for (auto child : adj[node]){
        if (!visited[child])
            dfs(child);
    }
    End[node] = start - 1;
}
dfs(1);
vector<int> FlatArray(start + 5);
for (auto i : Start){
    FlatArray[i.second] = Val[i.first];
    Range[i.first]=
                   {i.second,  End[i.first]};
}
```

6.9 LCA

```
const int N = 1e5 + 5, LG = 17;
vector<int> g[N];
int depth[N], parent[N][LG];
void dfs(int u, int p) {
    depth[u] = depth[p] + 1;
    parent[u][0] = p;
    for (int i = 1; i < LG; i++) {
        parent[u][i] = parent[parent[u][i -
1]][i - 1];
    }
    for (int v : g[u]) {
        if (v == p) continue;
        dfs(v, u);
    }
}
int lca(int u, int v) {
    if (depth[u] < depth[v]) swap(u, v);
    int diff = depth[u] - depth[v];
    for (int i = 0; i < LG; i++) {
        if (diff >> i & 1) {
            u = parent[u][i];
        }
    }
    if (u == v) return u;
    for (int i = LG - 1; i >= 0; i--) {
        if (parent[u][i] != parent[v][i]) {
            u = parent[u][i];
            v = parent[v][i];
        }
    }
    return parent[u][0];
}
int dist(int u, int v) {
    return depth[u] + depth[v] - 2 *
depth[lca(u, v)];
}
int kthParent(int u, int k) {
    for (int i = 0; i < LG; i++) {
        if (k >> i & 1) {
            u = parent[u][i];
        }
    }
    return u;
}
```

6.10 Min cost max flow

```
struct Edge{
    int from, to, capacity, cost;
};
vector<vector<int>> adj, cost, capacity;
const int INF = 1e9;
void shortest_paths(int n, int v0,
vector<int>& d, vector<int>& p) {
    d.assign(n, INF);
    d[v0] = 0;
    vector<bool> inq(n, false);
    queue<int> q;
    q.push(v0);
    p.assign(n, -1);
    while (!q.empty()) {
        int u = q.front();
        q.pop();
        inq[u] = false;
        for (int v : adj[u]) {
            if (capacity[u][v] > 0 && d[v] >
d[u] + cost[u][v]) {
                d[v] = d[u] + cost[u][v];
                p[v] = u;
                if (!inq[v]) {
                    inq[v] = true;
                    q.push(v);
            }}}}}
int min_cost_flow(int N, vector<Edge> edges,
int K, int s, int t) {
    adj.assign(N, vector<int>());
    cost.assign(N, vector<int>(N, 0));
    capacity.assign(N, vector<int>(N, 0));
    for (Edge e : edges) {
        adj[e.from].push_back(e.to);
        adj[e.to].push_back(e.from);
        cost[e.from][e.to] = e.cost;
        cost[e.to][e.from] = -e.cost;
        capacity[e.from][e.to] = e.capacity;
    }
    int flow = 0;
    int cost = 0;
    vector<int> d, p;
    while (flow < K) {
        shortest_paths(N, s, d, p);
        if (d[t] == INF)
            break;
        // find max flow on that path
        int f = K - flow;
        int cur = t;
        while (cur != s) {
            f = min(f,
capacity[p[cur]][cur]);
            cur = p[cur];
        }
        // apply flow
        flow += f;
        cost += f * d[t];
        cur = t;
        while (cur != s) {
            capacity[p[cur]][cur] -= f;
            capacity[cur][p[cur]] += f;
            cur = p[cur];
        }
    }

    if (flow < K)
        return -1;
    else
        return cost;
}
```

6.11 SCC

```
vector<bool> visited; // keeps track of which
vertices are already visited

// runs depth first search starting at vertex
v.
// each visited vertex is appended to the
output vector when dfs leaves it.
void dfs(int v, vector<vector<int>> const
&adj, vector<int> &output) {
  visited[v] = true;
  for (auto u : adj[v])
    if (!visited[u])
      dfs(u, adj, output);
  output.push_back(v);
}

// input: adj -- adjacency list of G
// output: components -- the strongy
connected components in G
// output: adj_cond -- adjacency list of
G^SCC (by root vertices)
void scc(vector<vector<int>> const &adj,
vector<vector<int>> &components,
vector<vector<int>> &adj_cond) {
  int n = adj.size();
  components.clear(), adj_cond.clear();

  vector<int> order; // will be a sorted list
of G's vertices by exit time

  visited.assign(n, false);

  // first series of depth first searches
  for (int i = 0; i < n; i++)
    if (!visited[i])
      dfs(i, adj, order);

  // create adjacency list of G^T
  vector<vector<int>> adj_rev(n);
  for (int v = 0; v < n; v++)
    for (int u : adj[v])
      adj_rev[u].push_back(v);

  visited.assign(n, false);
  reverse(order.begin(), order.end());

  vector<int> roots(n, 0); // gives the root
vertex of a vertex's SCC

  // second series of depth first searches
  for (auto v : order)
    if (!visited[v]) {
      std::vector<int> component;
      dfs(v, adj_rev, component);
      components.push_back(component);
      int root =
*min_element(begin(component),
end(component));
      for (auto u : component)
        roots[u] = root;
    }

  // add edges to condensation graph
  adj_cond.assign(n, {});
  for (int v = 0; v < n; v++)
    for (auto u : adj[v])
      if (roots[v] != roots[u])

adj_cond[roots[v]].push_back(roots[u]);
}
```

6.12 Biparitite

```
const int N=1000;
int adj[N][N];
int n,e;
bool isBicolored(int s){
    int colorArray[n];
    for(int i=0;i<n;i++)
        colorArray[i]=-1; //init no color;
    queue<int>q;
    q.push(s);
    colorArray[s]=1; //assigning first color
    while(!q.empty()){
        int senior = q.front();
        q.pop();
        if(adj[senior][senior]==1)
            return false;
        for(int i=0;i<n;i++){
            int junior=i;
            if(adj[senior][junior]==1){

if(colorArray[junior]==colorArray[senior])
//successor(child/junior) having same color
                    return false;
```

```
                ///if(colorArray[junior]!=-1)
continue;    ///not same color but have a
color
                else
if(colorArray[junior]==-1){        ///No
color assigned
                    q.push(junior);

colorArray[junior]=!colorArray[senior];
///assigning diff color
 }}}} return true;}
```

6.13 2D Convex Hull

```
struct point {
    ll x, y;
    bool operator == (point const& t) const {
        return (x == t.x && y == t.y);
    }
};
int orientation(point a, point b, point c) {
    ll d = (c.y - b.y) * (b.x - a.x) - (c.x -
b.x) * (b.y - a.y);
    if (d < 0) return -1; //clockwise
    else if (d > 0) return 1; //anticlockwise
    return 0; //collinear
}

bool clockwise(point a, point b, point c,
bool include_collinear) {
    int o = orientation(a, b, c);
    if (o < 0) return true;
    if (o > 0) return false;
    return include_collinear;
}
bool collinear(point a, point b, point c) {
    return orientation(a, b, c) == 0;
}
bool cmp_points(point& p1, point& p2) {
    return make_pair(p1.x, p1.y) <
make_pair(p2.x, p2.y);
}
ll distance_sq(point a, point b) {
    ll d = (a.x - b.x) * (a.x - b.x) + (a.y -
b.y) * (a.y - b.y);
    return d;
}

void print(vector<point>& v) {
    for (auto it : v) {
        cout << "(" << it.x << ", " << it.y
<< ")" << "\n";
    }
}
vector<point> convex_hull(vector<point>&
points, bool include_collinear = false) {
    int sz = (int)points.size();
    point p0 = *min_element(points.begin(),
points.end(), cmp_points);
    vector<point> convex_hull_vector =
points;
    sort(convex_hull_vector.begin(),
convex_hull_vector.end(), [&p0](const point&
p1, const point& p2) {
        int o = orientation(p0, p1, p2);
        if (o == 0) {
            return distance_sq(p0, p1) <
distance_sq(p0, p2);
        }
        return o < 0;
    });
    if (include_collinear) {
        int idx = sz - 1;
        while (idx >= 0 && collinear(p0,
convex_hull_vector[idx],
convex_hull_vector.back())) idx--;
        reverse(convex_hull_vector.begin() +
idx + 1, convex_hull_vector.end());
    }
    vector<point> st;
    for (int i = 0; i < sz; i++) {
        while (st.size() >= 2 &&
!clockwise(st[st.size() - 2], st.back(),
convex_hull_vector[i], include_collinear)) {
            st.pop_back();
        }
        st.push_back(convex_hull_vector[i]);
    }
    if (!include_collinear && st.size() == 2
&& st[0] == st[1]) {
        st.pop_back();
    }
    return st;
}
```

6.14 Computational Geometry

```
typedef long long int T;
typedef double Tf;
Tf eps = 1e-10;

// int sgn(Tf x)
// {
//     if (fabs(x) < eps) return 0;
//     if (x > 0) return 1;
//     return -1;
// }

int sgn(T x) {
    if(x == 0) return 0;
    if(x > 0) return 1;
    else return -1;
}
struct point {
    T x, y;
    point(T x = 0, T y = 0) {
        this->x = x;
        this->y = y;
    }
    point operator+(point p) { return point(x
+ p.x, y + p.y); }
    point operator-(point p) { return point(x
- p.x, y - p.y); }
    Tf len() { return sqrt(x * x + y * y); }
    T norm() { return x * x + y * y; }
    point operator*(T a) { return point(x *
a, y * a); }
    point operator/(T a) { return point(x /
a, y / a); }
    // integer
    bool operator==(point p) { return p.x ==
x && p.y == y; }
    // float
    // bool operator==(point p) {
    //     return sgn(p.x - x) == 0 &&
sgn(p.y - y) == 0;
    // }
    bool operator<(point p) {
        if (x == p.x) return y < p.y;
        return x < p.x;
    }
    // float
    // bool operator<(point p) {
    //     if (sgn(p.x - x) == 0) return
sgn(y - p.y) == -1;
    //     return sgn(x - p.x) == -1;
    // }
};
T dot(point a, point b) { return a.x * b.x +
a.y * b.y; }
T cross(point a, point b) { return a.x * b.y
- b.x * a.y; }
T orien(point a, point b, point c) { return
cross(a - b, a - c); }

Tf area(point a, point b, point c) {
    return abs(orien(a, b, c)) / (Tf)2;
}
```

```
struct segment {
    point a, b;
    bool onSegment(point p) {
        if (sgn(orien(a, b, p)) != 0) return
false;
        return sgn(dot(p - a, p - b)) <= 0;
    }
    bool intersection(segment s) {
        if (onSegment(s.a)) return true;
        if (onSegment(s.b)) return true;
        if (s.onSegment(a)) return true;
        if (s.onSegment(b)) return true;
        int s1 = sgn(orien(a, b, s.a));
        int s2 = sgn(orien(a, b, s.b));
        int s3 = sgn(orien(s.a, s.b, a));
        int s4 = sgn(orien(s.a, s.b, b));
        if (s1 * s2 < 0 && s3 * s4 < 0)
return true;
        return false;
    }
};
ostream &operator<<(ostream &os, point p) {
    return os << "(" << p.x << "," << p.y <<
")";
}
```

6.15 Depth and width of tree

```
int l[100]= {0}, int r[100]= {0};
stack<int> mystack;
int n = 0, w = 0, d = 0;
int depth(int n){
    if (l[n]==0 && r[n]==0)
        return 1;
    in depthl=depth(l[n]);
    int depthr=depth(r[n]);
    int dep=depthl>depthr ? depthl:depthr;
    return dep+1;
}
void width(int n){
    if (n<=d){
        int t=0,x;
        stack<int> tmpstack;
        while (!mystack.empty()){
            x=mystack.top();
            mystack.pop();
            if (x!=0){
                t++;
                tmpstack.push(l[x]);
                tmpstack.push(r[x]);
            }
        }
        w=w>t?w:t;
        mystack=tmpstack;
        width(n+1);
    }
}
int main(void){
    cin >> n;
    for (int i=1; i<=n; i++)
        cin >> l[i] >> r[i];
    d=depth(1);
    mystack.push(1);
    width(1);
    cout << w << " " << d << endl;
    return 0;
}
```

6.16 Max Pos and Next Greater

```
for (int i = n - 1; i >= 0; i--) {
  while (!stk.empty() && v[i] >=
v[stk.top()]) stk.pop();
  ind[i] = stk.empty() ? -1 : stk.top();
  stk.push(i);
}
// 3 1 5 4 10
// 2 2 4 4 -1
```

6.16 Articulation Bridges

```
const int N = 2e5 + 5;
vector < pair < int, int >> g[N];
int vis[N], st[N], low[N];
int clk, n;
vector < pair < int, int >> bridges;

void dfs(int u, int p, int edg) {
  vis[u] = 1;
  ++clk;
  st[u] = clk;
  low[u] = clk;
  for (auto[v, id]: g[u]) {
    // handled parallel edges between u, v
    if (v == p && id == edg) continue;
    if (vis[v]) {
      low[u] = min(low[u], st[v]);
    } else {
      dfs(v, u, id);
      low[u] = min(low[u], low[v]);
      if (st[u] < low[v]) {
        // edge u, v has bridge
        bridges.push_back({
          u,
          v
        });
      }
    }
  }
}
void find_bridges() {
  clk = 0;
  for (int i = 1; i <= n; i++) {
    vis[i] = 0;
    st[i] = low[i] = -1;
  }
  for (int i = 1; i <= n; i++) {
    if (!vis[i]) {
      dfs(i, 0, -1);
    }
  }
}
```

6.16 Articulation Points

```
const int N = 2e5 + 5;
vector < int > g[N];
int vis[N], st[N], low[N], adjComp[N];
int clk, n;

void dfs(int u, int p) {
  vis[u] = 1;
  ++clk;
  st[u] = clk;
  low[u] = clk;
  int child = 0;
  for (int v: g[u]) {
    if (v == p) continue;
    if (vis[v]) {
      low[u] = min(low[u], st[v]);
    } else {
      dfs(v, u);
      low[u] = min(low[u], low[v]);
      if (low[v] >= st[u] && p != 0) {
        adjComp[u]++;
      }
      child++;
    }
  }
  if (p == 0 && child > 1) {
    adjComp[u] = child - 1;
  }
}
void find_vertices() {
  clk = 0;
  for (int i = 1; i <= n; i++) {
    vis[i] = 0;
    st[i] = low[i] = -1;
  }
  for (int i = 1; i <= n; i++) {
    if (!vis[i]) {
      dfs(i, 0);
    }
  }
}
```

6.16 Block Cut Tree

```
const int N = 2e5 + 5;

vector < int > g[N], bct[2 * N];
int vis[N], low[N], disc[N], is_art[N];
stack < int > stk;
vector < int > block_of[N];
int clk, n, m, bct_nodes, root;

void dfs(int u, int p) {
  vis[u] = 1;
  disc[u] = low[u] = ++clk;
  stk.push(u);
  int child = 0;

  for (int v: g[u]) {
    if (v == p) continue;
    if (!vis[v]) {
      child++;
      dfs(v, u);
      low[u] = min(low[u], low[v]);

      if ((p == -1 && child > 1) || (p != -1
&& low[v] >= disc[u])) {
        is_art[u] = 1;
        vector < int > block;
        while (stk.top() != v) {
          block.push_back(stk.top());
          stk.pop();
        }
        block.push_back(stk.top());
        stk.pop();
        block.push_back(u);

        int block_id = ++bct_nodes;
        for (int x: block) {
          block_of[x].push_back(block_id);
          if (is_art[x]) {
            bct[x].push_back(block_id);
            bct[block_id].push_back(x);
          }
        }
      }
    } else {
      low[u] = min(low[u], disc[v]);
    }
  }
}

void build_bct() {
  clk = 0;
  bct_nodes = n;
  for (int i = 1; i <= n; i++) {
    vis[i] = is_art[i] = 0;
    block_of[i].clear();
    bct[i].clear();
  }
  for (int i = n + 1; i <= 2 * n; i++)
bct[i].clear();

  for (int i = 1; i <= n; i++) {
    if (!vis[i]) {
      dfs(i, -1);
      if (!stk.empty()) {
        int block_id = ++bct_nodes;
        while (!stk.empty()) {
          int x = stk.top();
          stk.pop();
          block_of[x].push_back(block_id);
          if (is_art[x]) {
            bct[x].push_back(block_id);
            bct[block_id].push_back(x);
          }
        }
      }
    }
  }

  root = n + 1;
  for (int i = 1; i <= n; i++) {
    if (is_art[i]) {
      root = i;
      break;
    }
  }
}
```

7 Random Staff

7.1 HASHING

```
const int N = 2e6 + 9;
const int MOD1 = 127657753, MOD2 = 987654319;
const int p1 = 137, p2 = 277;
int ip1, ip2;
pair<int, int> pw[N], ipw[N];
void prec() {
  pw[0] =  {1, 1};
  for (int i = 1; i < N; i++) {
    pw[i].first = 1LL * pw[i - 1].first * p1
% MOD1;
    pw[i].second = 1LL * pw[i - 1].second *
p2 % MOD2;
  }
  ip1 = power(p1, MOD1 - 2, MOD1);
  ip2 = power(p2, MOD2 - 2, MOD2);
  ipw[0] =  {1, 1};
  for (int i = 1; i < N; i++) {
    ipw[i].first = 1LL * ipw[i - 1].first *
ip1 % MOD1;
    ipw[i].second = 1LL * ipw[i - 1].second *
ip2 % MOD2;
  }
}
struct Hashing {
  int n;
  string s; // 0 - indexed
  vector<pair<int, int>> hs; // 1 - indexed
  Hashing() {}
  Hashing(string _s) {
    n = _s.size();
    s = _s;
    hs.emplace_back(0, 0);
    for (int i = 0; i < n; i++) {
      pair<int, int> p;
      p.first = (hs[i].first + 1LL *
pw[i].first * s[i] % MOD1) % MOD1;
      p.second = (hs[i].second + 1LL *
pw[i].second * s[i] % MOD2) % MOD2;
      hs.push_back(p);
    }
  }
  pair<int, int> get_hash(int l, int r) { //
1 - indexed
    assert(1 <= l && l <= r && r <= n);
    pair<int, int> ans;
    ans.first = (hs[r].first - hs[l -
1].first + MOD1) * 1LL * ipw[l - 1].first %
MOD1;
    ans.second = (hs[r].second - hs[l -
1].second + MOD2) * 1LL * ipw[l - 1].second %
MOD2;
    return ans;
  }
  pair<int, int> get_hash() {
    return get_hash(1, n);
  }
};
string s;
Hashing hs;
int lcp(int i1, int j1, int i2, int j2) {
      int l = 1, r = min(j1 - i1 + 1, j2 - i2
+ 1), ans = 0;
      while (l <= r) {
            int mid = (l + r) / 2;
            if (hs.get_hash(i1 + 1, i1 + mid)
== hs.get_hash(i2 + 1, i2 + mid)) {
                  ans = mid;
                  l = mid + 1;
            }
            else {
                  r = mid - 1;
            }
      }

      return ans;
}
int compare(int i1, int j1, int i2, int j2) {
      int l = lcp(i1, j1, i2, j2);
      int len1 = j1 - i1 + 1, len2 = j2 - i2
+ 1;
      if (len1 == len2 && l == len1) return
0;
      if (l == len1) return -1;
      if (l == len2) return 1;
      return s[i1 + l] < s[i2 + l] ? -1 : 1;
}
```

7.2 Suffix Array

```
struct SuffixArray {
  // p suffix array 1 base, 0 index for $
  // rank will show 1 index value
  // 0 base suffix array -> suf[rank[i] - 1]
= i
  vector < int > p, c, rank, lcp;
  vector < vector < int >> st;
  SuffixArray(string
    const & s) {
    build_suffix(s + char(1));
    build_rank(p.size());
    build_lcp(s + char(1));
    build_sparse_table(lcp.size());
  }
  void build_suffix(string
    const & s) {
    int n = s.size();
    const int MX_ASCII = 256;
    vector < int > cnt(max(MX_ASCII, n), 0);
    p.resize(n);
    c.resize(n);
    for (int i = 0; i < n; i++) cnt[s[i]]++;
    for (int i = 1; i < MX_ASCII; i++) cnt[i]
+= cnt[i - 1];
    for (int i = 0; i < n; i++)
p[--cnt[s[i]]] = i;
    c[p[0]] = 0;
    int classes = 1;
    for (int i = 1; i < n; i++) {
      if (s[p[i]] != s[p[i - 1]]) classes++;
      c[p[i]] = classes - 1;
    }
    vector < int > pn(n), cn(n);
    for (int h = 0;
      (1 << h) < n; ++h) {
      for (int i = 0; i < n; i++) {
        pn[i] = p[i] - (1 << h);
        if (pn[i] < 0) pn[i] += n;
      }
      fill(cnt.begin(), cnt.begin() +
classes, 0);
      for (int i = 0; i < n; i++)
cnt[c[pn[i]]]++;
```

```
      for (int i = 1; i < classes; i++)
cnt[i] += cnt[i - 1];
      for (int i = n - 1; i >= 0; i--)
p[--cnt[c[pn[i]]]] = pn[i];
      cn[p[0]] = 0;
      classes = 1;
      for (int i = 1; i < n; i++) {
        pair < int, int > cur = {
          c[p[i]],
          c[(p[i] + (1 << h)) % n]
        };
        pair < int, int > prev = {
          c[p[i - 1]],
          c[(p[i - 1] + (1 << h)) % n]
        };
        if (cur != prev) ++classes;
        cn[p[i]] = classes - 1;
      }
      c.swap(cn);
    }
  }
  void build_rank(int n) {
    rank.resize(n, 0);
    for (int i = 0; i < n; i++) rank[p[i]] =
i;
  }
  void build_lcp(string
    const & s) {
    int n = s.size(), k = 0;
    lcp.resize(n - 1, 0);
    for (int i = 0; i < n; i++) {
      if (rank[i] == n - 1) {
        k = 0;
        continue;
      }
      int j = p[rank[i] + 1];
      while (i + k < n && j + k < n && s[i +
k] == s[j + k])
        k++;
      lcp[rank[i]] = k;
      if (k) k--;
    }
  }
  void build_sparse_table(int n) {
    int lim = __lg(n);
    st.resize(lim + 1, vector < int > (n));
    st[0] = lcp;
    for (int k = 1; k <= lim; k++)
      for (int i = 0; i + (1 << k) <= n; i++)
        st[k][i] = min(st[k - 1][i], st[k -
1][i + (1 << (k - 1))]);
  }
  int get_lcp(int i) {
    return lcp[i];
  }
  int get_lcp(int i, int j) {
    if (j < i) swap(i, j);
    j--; /*for lcp from i to j we don't need
last lcp*/
    int K = __lg(j - i + 1);
    return min(st[K][i], st[K][j - (1 << K) +
1]);
  }
  // Compare two substrings (l1, r1) and (l2,
r2)
  int compare(int l1, int r1, int l2, int r2)
{
    int len1 = r1 - l1 + 1;
    int len2 = r2 - l2 + 1;
    int pos1 = rank[l1];
    int pos2 = rank[l2];
    if (pos1 == pos2) {
      if (len1 != len2) return len1 < len2 ?
-1 : 1;
      return 0;
    }
    int common = get_lcp(min(pos1, pos2),
max(pos1, pos2));
    int compare_len = min(min(len1, len2),
common);
    if (compare_len == len1 && compare_len ==
len2) return 0;
    if (compare_len == len1) return -1;
    if (compare_len == len2) return 1;
    return pos1 < pos2 ? -1 : 1;
  }
};
// s.compare(suf[mid], min(n - suf[mid], m),
t) -> -1(small), 0(equal), 1(large)
```

7.2 when phi(1) to phi(n) is needed

```
int phi[MX];
//bitset<MX> visited;// declared before in
optimized SIEVE
void sieve_phi(){
    for(int i=1; i<MX; ++i) phi[i] = i;
    visited[1] = 1;
    for(int i=2; i<MX; ++i){
        if(!visited[i]){
            for(int j = i; j<MX; j+=i){
                visited[j] = 1;
                phi[j] = phi[j]/i*(i-1);
            }         }    }
}///O(log(logn))
```

7.4 Knight Moves

```
int X[8]={2,1,-1,-2,-2,-1,1,2};
int Y[8]={1,2,2,1,-1,-2,-2,-1};
```

7.5 bit count in O(1)

```
int BitCount(unsigned int u){
     unsigned int uCount;
     uCount = u - ((u >> 1) & 033333333333) -
((u >> 2) & 011111111111);
     return ((uCount + (uCount >> 3)) &
030707070707) % 63;}
```

7.6 Matrix Exponentiation

```
#include<bits/stdc++.h>
using namespace std;
typedef long long LL;

LL arr[60][60],res[60][60],tmp[60][60],m;

void matMul (LL a[][60], LL b[][60], LL mod)
{
    for(int i=0; i<m; i++)
        for(int j=0; j<m; j++)
        {
            tmp[i][j] = 0;
            for(int k=0; k<m; k++)
            {
                tmp[i][j] +=
(a[i][k]*b[k][j])%mod;
                tmp[i][j] %= mod;
            }         }}

void power(LL n, LL mod)
{
    for(int i=0; i<m; i++)
        for(int j=0; j<m; j++)
            if(i==j) res[i][j] = 1;
            else res[i][j] = 0;

    while(n)    {
        if(n&1)        {
            matMul(res,arr,mod);
            for(int i=0; i<m; i++)
                for(int j=0; j<m; j++)
res[i][j] = tmp[i][j];
            n--;        }
        else       {
            matMul(arr,arr,mod);
            for(int i=0; i<m; i++)
                for(int j=0; j<m; j++)
arr[i][j] = tmp[i][j];
            n/=2;        }    }}
```

7.8 sqrt decomposition(MO's Algo)

```
// https://www.spoj.com/problems/DQUERY/
void remove(int idx) {
      int v = arr[idx];
      freq[v]--;
      if (freq[v] == 0) cnt--;
}
void add(int idx) {
      int v = arr[idx];
      freq[v]++;
      if (freq[v] == 1) cnt++;
}
int get_answer() {
      return cnt;
}
int block_size = 700;
struct Query {
      int l, r, idx;
      bool operator<(Query other) const {
            if (l / block_size != other.l /
block_size) return l / block_size < other.l /
block_size;
            return r < other.r;
      }
};
vector<int> mo_s_algorithm(vector<Query>
queries) {
      vector<int> answers(queries.size());
      sort(queries.begin(), queries.end());
      int cur_l = 0;
      int cur_r = -1;
      // invariant: data structure will
always reflect the range [cur_l, cur_r]
      for (Query q : queries) {
            while (cur_l > q.l) {
                  cur_l--;
                  add(cur_l);
            }
            while (cur_r < q.r) {
                  cur_r++;
                  add(cur_r);
            }
            while (cur_l < q.l) {
                  remove(cur_l);
                  cur_l++;
            }
            while (cur_r > q.r) {
                  remove(cur_r);
                  cur_r--;
            }
            answers[q.idx] = get_answer();
      }
      return answers;
}
```

7.10 PIE(inclusion - exclusion)

```
//count the numbers between 1 and n
(inclusive) that are not divisible by any of
the integers in the given array a
//|A₁ ∪ A₂ ∪ ... ∪ A□| = Σ|Aᵢ| - Σ|Aᵢ ∩ A□| +
Σ|Aᵢ ∩ A□ ∩ A□| - ... + (-1)^(n-1)|A₁ ∩ A₂ ∩
... ∩ A□|
#include <bits/stdc++.h>
using namespace std;

inline int LCM(int a, int b){
    return a * b / __gcd(a, b);
}

int PIE(int div[], int n, int num){
    int sum = 0;
    for(int msk=1; msk < (1<<n); ++msk){
        int bit_cnt = 0;
        int cur_lcm = 1;
        for (int i = 0; i < n; ++i){
            if (msk & (1 << i)){
                ++bit_cnt;
                cur_lcm = LCM(cur_lcm,
div[i]);
            }
        }

        int cur = num / cur_lcm;
        if (bit_cnt & 1) sum += cur;
        else sum -= cur;
    }
    return num - sum;
}

signed main(){
    int n, m;
    while (cin >> n >> m){
        int a[m];
        for(int &i : a)cin >> i;
        cout << PIE(a, m, n) << '\n'; }}
```

7.11 LARGEST POWER OF K IN N!

```
int largestPower(int k, int n){
    int cnt=0;
    lili x=k;
    while(x<=n){
        cnt+=(n/x);
        x*=k;
    }
    return cnt;
}
//Smallest Prime Factor, Greatest Prime
Factor, No of Distinct Prime Factors, No of
Total Prime Factors, No. of Divisors, Sum of
Divisors
const lili N=1e6+5;
lili spf[N];
lili lpf, gpf , tpf, dpf, sum, dv;
void sieve(){
    for(int i=0;i<N;i++){
        spf[i]=0; }
for(lili
i=2;i*i<N;i++){if(spf[i]==0){for(lili
j=i*i;j<N;j+=i){if(spf[j]==0)spf[j]=i;}}}
for(lili i=2;i<N;i++){
        if(spf[i]==0){ spf[i]=i; }}}
lili power(lili a, lili b){
    if(b==0) return 1;
    lili x=power(a,b/2);
    if(b%2==0) x=x*x;
    else x=x*x*a;
    return x;
}
int main(){
    ios_base::sync_with_stdio(false);
    cin.tie(NULL);
    sieve();
    lili m,n; cin>>m;
    while(m--){
        cin>>n;
        lpf=spf[n]; gpf=-1; tpf=0; dpf=0;
dv=1; sum=1;
        while(n>1){
            lili x=spf[n];
            lili cnt=0;
            dpf++;
            while(n%x==0){
                tpf++;
                gpf=max(gpf,x);
                cnt++;
                n/=x;
            }
            dv*=(cnt+1);
            lili k=(power(x,cnt+1)-1)/(x-1);
            sum*=k;
        }
        cout<<lpf<<" "<<gpf<<" "<<dpf<<"
"<<tpf<<" "<<dv<<" "<<sum<<"\n";}}
```

7.12 Binary Search

```
ll lo=0, hi=mx;  ///mx=max possible ans
while(lo<hi){
    ll mid=(lo+hi+1)>>1;
    if(condition)  ///valid condition->and
can be greater than or equal mid
        lo=mid;
    else
        hi=mid-1;  ///ans is less than mid
}
///or
while(lo<hi){
    ll mid=(lo+hi)>>1;
    if(condition)  ///valid condition->and
can be less than or equal mid
        hi=mid;
    else
        lo=mid+1;  ///ans is greater than mid
}

ll lo=0, hi=mx, esp=maxError;
while((hi-lo)>esp){
    ll mid=(lo+hi+esp)/2.0;
    if(condition) lo=mid;
    else      hi=mid-esp;
}
 while((hi-lo)>esp){
    ll mid=(lo+hi)/2.0;
    if(condition)  hi=mid;
    else       lo=mid+esp;
}
```

7.12.1 Ternary Search

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

7.14 N Queen optimal

```
// It just counts the number of ways to place
the order.
const int N = 32;
int mark[N][N];
char grid[N][N];
int n, cnt;
void fillup(int row, int col) {
    for (int i = 1; i < n - row + 1; i++) {
        mark[row + i][col]++;
        if (col - i >= 0)
            mark[row + i][col - i]++;
        if (col + i < n)
            mark[row + i][col + i]++;
    }
}
void fillout(int row, int col) {
    for (int i = 1; i < n - row + 1; i++) {
        mark[row + i][col]--;
        if (col - i >= 0)
            mark[row + i][col - i]--;
        if (col + i < n)
            mark[row + i][col + i]--;
    }
}
void find_way(int row) {
    if (row == n) {
        cnt++;
        return;
    }
    for (int j = 0; j < n; j++) {
        if (grid[row][j] == '*' or
mark[row][j])
            continue;
        fillup(row, j);
        find_way(row + 1);
        fillout(row, j);
    }
}
// input in grid. call find_way(0);
```

7.15 HLD

```
vector<int> g[N];
int par[N], dep[N], sz[N];
void dfs(int u, int p = 0) {
    sz[u] = 1, par[u] = p;
    dep[u] = dep[p] + 1;
    int mx = 0;
    for (auto &v: g[u]) {
        if (v == p) continue;
        dfs(v, u);
        sz[u] += sz[v];
        if (sz[v] > mx) {
            mx = sz[v];
            swap(v, g[u][0]);
        }
    }
}
int T, head[N], st[N], en[N];

void dfs_hld(int u, int p = 0) {
    st[u] = ++T;
    // arr[T] = a[u];
    head[u] = (p != 0 && g[p][0] == u) ?
head[p] : u;
    for (auto v: g[u]) {
        if (v == p) continue;
        dfs_hld(v, u);
    }
    en[u] = T;
}
int lca(int a, int b) {
    for (; head[a] != head[b]; b =
par[head[b]]) {
        if (dep[head[a]] > dep[head[b]])
swap(a, b);
    }
    if (dep[a] > dep[b]) swap(a, b);
    return a;
}
int n;
// process node u upto it's ancestor a
// if excl is true, a won't process
i64 chain_process(int a, int u, bool excl =
false) {
    i64 ans = 0;
    for (; head[u] != head[a]; u =
par[head[u]]) {
        ans = ans + query(1, 1, n,
st[head[u]], st[u]);
    }
    ans = ans + query(1, 1, n, st[a] + excl,
st[u]);
    return ans;
}

// process path from node u to v
// if excl is true, lca won't process
i64 path_process(int a, int b, bool excl =
false) {
    i64 ans = 0;
    for (; head[a] != head[b]; b =
par[head[b]]) {
        if (dep[head[a]] > dep[head[b]])
swap(a, b);
        ans = ans + query(1, 1, n,
st[head[b]], st[b]);
    }
    if (dep[a] > dep[b]) swap(a, b);
    ans = ans + query(1, 1, n, st[a] + excl,
st[b]);
    return ans;
}
// update path from node u to v
// if excl is true, lca won't update
void path_update(int a, int b, int val, bool
excl = false) {
    for (; head[a] != head[b]; b =
par[head[b]]) {
        if (dep[head[a]] > dep[head[b]])
swap(a, b);
        update(1, 1, n, st[head[b]], st[b],
val);
    }
    if (dep[a] > dep[b]) swap(a, b);
    update(1, 1, n, st[a] + excl, st[b],
val);
}
/*
     n is global
     dfs(1);
     head[1] = 1;
     dfs_hld(1);
     build(1, 1, n);
*/
void solve() {
      cin >> n;
      int q;
      cin >> q;
      for (int i = 1; i < n; i++) {
            int u, v, c;
            cin >> u >> v >> c;
            g[u].push_back({v, i});
            g[v].push_back({u, i});
            edge_cost[i] = c;
      }
      dfs(1);
      head[1] = 1;
      dfs_hld(1);
      for (int i = 1; i < n; i++) {
            int u = edge_to_ch[i];
            arr[st[u]] = edge_cost[i];
      }
      build(1, 1, n);
      while (q--) {
            int u, v, val;
            cin >> u >> v >> val;
            path_update(u, v, val, true);
            cout << path_process(u, v, true)
<< '\n';
      }
}
```

7.16 Centroid Decomposition

```
const int N = 1e5 + 5;
vector<int> g[N];
int sz[N], dead[N], cenpar[N];
void dfs_sz(int u, int p) {
    sz[u] = 1;
    for (int v : g[u]) {
        if (v == p || dead[v]) continue;
        dfs_sz(v, u);
        sz[u] += sz[v];
    }
}
int find_centroid(int num, int u, int p) {
    for (int v : g[u]) {
        if (v != p && !dead[v] && 2 * sz[v] >
num) {
            return find_centroid(num, v, u);
        }
    }
    return u;
}
void decompose(int u, int p) {
    dfs_sz(u, p);
    int cent = find_centroid(sz[u], u, p);
    dead[cent] = true;
    cenpar[cent] = p;
    for (int v : g[cent]) {
        if (!dead[v]) {
            decompose(v, cent);
        }
    }
}
```

7.17 GRUNDY

```
int calculateGrundy(int n, vector<int>
&grundy, const vector<int> &moves)
{    if (grundy[n] != -1)  return grundy[n];
 unordered_set<int> s; for (int move : moves)
{    if (n >= move)  {
```

```
s.insert(calculateGrundy(n - move, grundy,
moves));  } }  int g = 0; while (s.count(g))
        g++;  return grundy[n] = g;}
vector<int> computeGrundy(int maxN, const
vector<int> &moves){    vector<int>
grundy(maxN + 1, -1);
    grundy[0] = 0; for (int i = 1; i <= maxN;
++i) { calculateGrundy(i, grundy, moves);   }
return grundy;}
```

7.18 Gaussian Elimination

```
class GaussianElimination{
public:
GaussianElimination(vector<vector<double>>
matrix, vector<double> results)
        : matrix(matrix), results(results),
n(matrix.size()) {}
void solve()  { fElim(); bSub(); }
vector<vector<double>> matrix; vector<double>
results, solution; int n;
void fElim()    {
for (int i = 0; i < n; ++i) {
  int maxRow = i;
  for (int k = i + 1; k < n; ++k)
if(abs(matrix[k][i])>abs(matrix[maxRow][i]))
maxRow = k;
swap(matrix[i], matrix[maxRow]);
swap(results[i], results[maxRow]);
for (int k = i + 1; k < n; ++k) {
double factor = matrix[k][i] / matrix[i][i];
 for (int j = i; j < n; ++j)
   matrix[k][j] -= factor * matrix[i][j];
   results[k] -= factor * results[i]; }   }
}
void bSub()  {
solution.resize(n);
for (int i = n - 1; i >= 0; --i)        {
 solution[i] = results[i];
 for (int j = i + 1; j < n; ++j)
  solution[i] -= matrix[i][j] * solution[j];
            solution[i] /= matrix[i][i];   }
}};
```

7.19 Calculating nth element of recurrence relation in O(logN)

- (f1 f2)(a b, c d) = (f3,f4)
- find a,b,c,d first then (f1 f2)(a b, c d)^n-1=(fn fn+1)