Index

→Useful Things

→Fast I/O

```
C++:
#pragma GCC optimize("O3")
ios_base::sync_with_stdio(false),
cin.tie(nullptr), cout.tie(nullptr);

Python:
import sys
input = sys.stdin.readline
sys.stdout.write("------")
```

→VSCode setup

- **run a file which name is test.cpp**
  `g++ .\test.cpp -o test && .\test`
- **run a file with input output**
  `g++ .\test.cpp -o test && .\test < in.txt > out.txt`
- **Assign Key**

```
{
  "key":"f5",
  "command":"workbench.action.terminal.sendSequence",
  "args":{
    "text":"g++ ${fileBasenameNoExtension}.cpp -o
          ${fileBasenameNoExtension} &&
          ./${fileBasenameNoExtension}
          < in.txt > out.txt\n"
  }
}
// add this json in keybindings.json file
// to run, open terminal along with your cpp code
```

→ Sublime Build Command

```
{
"shell_cmd":"g++ -std=c++17 \"${file}\" -o
            \"${file_base_name}.exe\" &&
            \"${file_path}/${file_base_name}
            .exe\" < in.txt > out.txt 2>
            error.txt",
"working_dir":"${file_path}",
"selector": "source.c++, source.cpp",
"shell": "true",
}
```

1 Formula

1.1 Area Formulas

Type	Area
Rectangle	length × width
Square	side × side
Triangle	0.5 × base × height
Parallelogram	base × height
Pyramid (excluding base)	0.5 × perimeter of base × slant height
Polygon	1. $\frac{1}{2}\left\|\sum_{i=1}^{n-1}(x_i y_{i+1} - x_{i+1} y_i)\right\|$ 2. $a + \frac{b}{2} - 1$ (for int coordinates)

a=#int points strictly inside polygon
b=#int points on sides polygon

1.2 Volume Formulas

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

1.3 Surface Area Formulas

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

1.4 Triangles

Side lengths: a, b, c

Semiperimeter	$s = \frac{a+b+c}{2}$
Area	$A = \sqrt{(s(s-a)(s-b)(s-c))}$
Circumradius	$R = \frac{abc}{4A}$
Inradius	$r = \frac{A}{s}$
Length of median	$m_a = \frac{1}{2} * \sqrt{2b^2 + 2c^2 - a^2}$
Length of bisector	$sa = \sqrt{\frac{bc}{1-(\frac{a}{b+c})^2}}$

1.5 Trigonometry

sin law: $sin\frac{\alpha}{a} = sin\frac{\beta}{b} = sin\frac{\gamma}{c} = \frac{1}{2R}$
cos law: $a^2 = b^2 + c^2 - 2bc\cos\alpha$
tan law: $\frac{a+b}{a-b} = \frac{tan\frac{\alpha+\beta}{2}}{tan\frac{\alpha-\beta}{2}}$
$sin\,(A + B) = sin\,A\,cos\,B + cos\,A\,sin\,B$
$cos\,(A + B) = cos\,A\,cos\,B - sin\,A\,sin\,B$
$sin\,(A - B) = sin\,A\,cos\,B - cos\,A\,sin\,B$
$cos\,(A - B) = cos\,A\,cos\,B + sin\,A\,sin\,B$
$tan\,(A + B) = \frac{(tan\,A + tan\,B)}{(1 - tan\,A\,tan\,B)}$
$tan\,(A - B) = \frac{(tan\,A - tan\,B)}{(1 + tan\,A\,tan\,B)}$
$sin\,2\theta = 2\,sin\,\theta\,cos\theta$
$cos\,2\theta = cos^2\theta - sin^2\theta$
$tan\,2\theta = \frac{(2\,tan\,\theta)}{(1 - tan\text{^}2\,\theta)}$
$sin\,(\frac{\theta}{2}) = \pm\sqrt{\frac{1 - cos\theta}{2}}$
$cos\,(\frac{\theta}{2}) = \pm\sqrt{\frac{1 + cos\theta}{2}}$
$tan\,(\frac{\theta}{2}) = \frac{(1 - cos\,\theta)}{sin\,\theta}$
$sin\,r + sin\,w = 2\,sin\,(\frac{r+w}{2})\,cos\,(\frac{r-w}{2})$
$cos\,r + cos\,w = 2\,cos\,(\frac{r+w}{2})\,cos\,(\frac{r-w}{2})$
$(V + W)\,tan(\frac{r-w}{2}) = (V - W)\,tan(\frac{r+w}{2})$ where V, W are lengths of sides opposite
$a\,cos\,x + b\,sin\,x = r\,cos(x - \varphi)$ $a\,sin\,x - b\,cos\,x = r\,sin(x - \varphi)$ where $r = \sqrt{a^2 + b^2}$, $\varphi = atan2(b, a)$

1.6 Sum

$c^k + c^{k+1} + \ldots + c^n = c^{n+1} - c^k$
// (c - 1)   for c ≠ 1

$1 + 2 + 3 + \ldots + n = \frac{n * (n+1)}{2}$

$1^2 + 2^2 + \ldots + n^2 = \frac{n * (n+1) * (2n+1)}{6}$

$1^3 + 2^3 + \ldots + n^3 = \frac{n^2 * (n+1)^2}{4}$

$1^4 + 2^4 + \ldots + n^4 = \frac{n * (n+1) * (2n+1) * (3n^2 + 3n - 1)}{30}$

sum of first n odd num = n^2

1.7 Logarithmic Basic

$log_b 1 = 0$	$log_b b = 0$
$b^{log_b a} = a$	$x^{log_b y} = y^{log_b x}$
$log_a b = \frac{1}{log_b a}$	$log_a x = \frac{log_b x}{log_b a}$
$log_b (AB) = log_b A + log_b B$	
$log_b (\frac{A}{B}) = log_b A - log_b B$	
$log_a c = log_a b * log_b c$	
$log_b A^x = x\,log_b A$	

1.8 Series

1.8.1 Catalan Series

Series: 1, 1, 2, 5, 14, 42, 132, 429, ...
Equation:

$$C_n = \frac{1}{n+1}\binom{2n}{n} = \frac{(2n)!}{(n+1)!\,n!} \qquad \text{for } n \ge 0.$$

$$C_n = C_0.C_{n-1-0} + C_1.C_{n-1-1} + ... + C_k.C_{n-1-k} + ... + C_{n-1}.C_0$$

1.8.2 Arithmetic Series

- $a_n = a + (n - 1) * d$
- $s_n = \frac{n}{2}(2 * a + (n - 1) * d)$

1.8.3 Geometric Series

- $a_n = a * r^{n-1}$
- $s_n = \frac{a(1 - r^n)}{1 - r}$

1.8.4 Derangement Series

Series : 0, 1, 2, 9, 44, 265, 1854, 14833, 133496, 1334961, 14684570

$$D_n = n!\sum_{k=0}^{n}\frac{(-1)^k}{k!}$$

$$D_n = floor\left[\frac{n!}{e} + \frac{1}{2}\right]$$

1.8.5 nth Fibo Golden Ratio

$$f_n = \left[\frac{\left(\frac{1+\sqrt{5}}{2}\right)^n - \left(\frac{1-\sqrt{5}}{2}\right)^n}{\sqrt{5}}\right]$$

1.1 Facts

- $ceil\left[\frac{a}{b}\right] = floor\left[\frac{a-1}{b}\right] + 1$
- $natural\, num\, sum = \frac{l+r}{2} * (r - l + 1)$
- $floor\left[\frac{floor\left[\frac{n}{a}\right]}{b}\right] = floor\left[\frac{n}{ab}\right]$

2 Number Theory

2.1 Prime number under 1000

```
2   3   5   7   11  13  17  19  23  29  31 37
41  43  47  53  59  61  67  71  73  79 83  89
97  101 103 107 109 113 127 131 137 139 149
151 157 163 167 173 179 181 191 193 197 199
211 223 227 229 233 239 241 251 257 263 269
271 277 281 283 293 307 311 313 317 331 337
347 349 353 359 367 373 379 383 389 397 401
409 419 421 431 433 439 443 449 457 461 463
467 479 487 491 499 503 509 521 523 541 547
557 563 569 571 577 587 593 599 601 607 613
617 619 631 641 643 647 653 659 661 673 677
683 691 701 709 719 727 733 739 743 751 757
761 769 773 787 797 809 811 821 823 827 829
839 853 857 859 863 877 881 883 887 907 911
919 929 937 941 947 953 967 971 977 983 991
997
```

2.2 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]++;
}
// count the number of divisors of all
numbers in a range.
```

2.3 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;
}
```

2.4 Num of Leap year in between

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

2.5 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;
    }
} //Function call: printCalendar(year);
```

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)

```
int binaryExp(int base, int power, int
modulo){
    int and = 1;
    while (power){
        if (power % 2 == 1)
            ans = (ans * base) % modulo;
        base = (base * base) % modulo;
        power /= 2;
    }
    return ans;
} //function call:
binaryExp(a, binaryExp(b, c, mod-1), mod)
```

2.8 Power

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

2.9 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;
}
```

2.10 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;
}
```

// some facts about spf
suppose you have a number N = 120;
you represent it as N = 2^3 * 3^1 * 5^2
Now from this representation we can easily calculate the number of divisors of number N.
Let's see how it works:

(i). we can take 2^3 in 4 different ways like 2^0, 2^1, 2^2, 2^3. In the same way we can take 3^1 in 2 ways(3^0, 3^1) and 5^2 in 3 ways(5^0, 5^1, 5^2).
(ii). Total number of divisor is = 4 * 2 * 3

suppose, N = $p_1^a \times p_2^b \times p_3^c$

number_of_divisors = $(a + 1) * (b + 1) * (c + 1)$

As like calculating the number of divisors, we can also calculate the sum of all divisors.

sum_of_divisors

$$\sigma(N) = \frac{p_1^{a+1} - 1}{p_1 - 1} * \frac{p_2^{b+1} - 1}{p_2 - 1} * \frac{p_3^{c+1} - 1}{p_3 - 1}$$

2.11 smallest prime factor(SPF) 1 to N

```
const int N = 1e7 + 5;
int spf[N];
void smallestPrimeFactorUsingSeive() {
    for (int i = 2; i < N; i++) {
        if (spf[i] == 0) {
            for (int j = i; j < N; j += i) {
                if (spf[j] == 0)
                    spf[j] = i;
            }
        }
    }
}
```

2.12 Seive

```
const ll N = 1e7 + 5;
ll isprime[N];
vector<int> primes;
void sieveOfEratosthenes() {
    for (ll i = 2; i < N; i++)
        isprime[i] = 1;
    for (ll i = 4; i < N; i += 2)
        isprime[i] = 0;
    for (ll i = 3; i * i < N; i += 2) {
        if (isprime[i]) {
            for (ll j = i * i; j<N; j+=i*2)
                isprime[j] = 0;
        }
    }
    for (ll i = 2; i < N; i++)
        if (isprime[i])
            primes.push_back(i);
}
```

2.13 Segmented Sieve

```
// find all the prime num in a range [L,R] of
small size (R-L+1==1e7) where R can be very
large e.g 1e12.
// before that you need to call sieve and
store the primes up to sqrt(R)
void segmentedSieve(ll L, ll R) {
    bool isPrime[R - L + 1];
    for (ll i = 0; i <= R - L + 1; i++)
        isPrime[i] = true;
    if (L == 1)
        isPrime[0] = false;
    for (ll i=0;primes[i]*primes[i]<=R;i++){
        ll curPrime = primes[i];
        ll base = curPrime * curPrime;
        if (base < L) {
            base = ((L + curPrime - 1) /
curPrime) * curPrime;
        }
        for (ll j=base;j<=R; j += curPrime)
            isPrime[j - L] = false;
    }
    for (ll i = 0; i <= R - L; i++) {
        if (isPrime[i] == true)
            cout << L + i << " ";
    }
    cout << endl;
}
```

2.14 Bitwise Seive

```
const ll N = 10000006;
bitset<N> sieve;
void bitwiseSieve() {
    sieve.flip();
    ll finalBit = sqrt(sieve.size()) + 1;
    for (ll i = 2; i < finalBit; i++) {
        if (sieve.test(i))
            for (ll j = 2 * i; j<N; j += i)
                sieve.reset(j);
    }
}
// to check any number is prime or not,
// check sieve.test(number) is true or not
```

2.15 nth prime number

```
// Time complexity O(log(logn))
vector<int> nth_prime;
const int MX = 86200005;
bitset<MX> visited;
void optimized_prime(){
    nth_prime.push_back(2);
    for(int i=3; i<MX; i+=2){
            if(visited[i])
                continue;
            nth_prime.push_back(i);
            if(1ll*i*i > MX)
                continue;
            for(int j = i*i; j< MX; j+= i+i)
                visited[j] = true;
    }
}
```

2.16 Seive up to 1e9 in 500ms

```
// takes 0.5s for n = 1e9
// copy from YouKnowWho
vector<ll> sieve(ll N, ll Q=17,ll L=1<<15){
    ll rs[] = {1,7,11,13, 17, 19, 23, 29};
    struct P {
        P(ll p) : p(p) {}
        ll p;
        ll pos[8];
    };
    auto approx_prime_count = [](ll N) {
        return N > 60184 ? N / (log(N)-1.1)
        : max(1., N / (log(N) - 1.11)) + 1;
    };
    ll v = sqrt(N), vv = sqrt(v);
    vector<bool> isp(v + 1, true);
    for (ll i = 2; i <= vv; ++i) {
        if (isp[i]) {
            for (ll j = i*i;j<=v;j+=i)
                isp[j] = false;
        }
    }
    ll rsize = approx_prime_count(N + 30);
    vector<ll> primes = {2, 3, 5};
    ll ps = 3;
    primes.resize(rsize);
    vector<P> sprimes;
    size_t pbeg = 0;
    ll prod = 1;
    for (ll p = 7; p <= v; ++p) {
        if (!isp[p])
            continue;
        if (p <= Q) {
            prod *= p;
            ++pbeg;
            primes[ps++] = p;
        }
        auto pp = P(p);
        for (ll t = 0; t < 8; ++t) {
            ll 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];
        ll p = pp.p;
        for (ll t = 0; t < 8; ++t) {
            unsigned char m = ~(1 << t);
            for (ll i=pp.pos[t];i<prod;i+=p)
                pre[i] &= m;
        }
    }
    ll bs = (L + prod - 1) / prod * prod;
    vector<unsigned char> block(bs);
    unsigned char* pb = block.data();
    ll M = (N + 29) / 30;
    for (ll s = 0; s < M; s += bs,pb-=bs) {
        ll f = min(M, s + bs);
        for (ll i = s; i < f; i += prod) {
        copy(pre.begin(), pre.end(), pb+i);
        }
        if (s == 0)  pb[0] &= 0xFE;
        for (size_t pi = pbeg; pi <
                    sprimes.size(); ++pi) {
            auto& pp = sprimes[pi];
            ll p = pp.p;
            for (ll t = 0; t < 8; ++t) {
                ll i = pp.pos[t];
                unsigned char m = ~(1 << t);
                for (; i < f; i += p)
                    pb[i] &= m;
                pp.pos[t] = i;
            }
        }
        for (ll i = s; i < f; ++i) {
            for (ll m=pb[i];m>0;m&=m-1){
```

```
     primes[ps++]=i*30+rs[__builtin_ctz(m)];
            }
        }
    }
    assert(ps <= rsize);
    while (ps > 0 && primes[ps - 1] > N)
        --ps;
    primes.resize(ps);
    return primes;
}
```

2.17 Legendre formula

```
// maximum power of prime p that divides n!
int legendre(int n, int p) {
    int ans = 0;
    while (n) {
        n /= p;
        ans += n;
    }
    return ans;
}
```

2.18 EXT_GCD

```
// return {x,y} such that ax+by=gcd(a,b)
pair<int,int>ext_gcd(int a, int b){
    if (b == 0)
        return {1, 0};
    else{
        pair<int,int> tmp=ext_gcd(b, a % b);
        return {tmp.second,
          tmp.first - (a / b) * tmp.second};
    }
}
```

2.19 PHI of N

```
// the positive integers less than or equal
to n that are relatively prime to n.
```

if $n = p_1^{a_1} * p_2^{a_2} * \ldots * p_k^{a_k}$ then

$$\phi(n) = n * (1 - \frac{1}{p_1}) * (1 - \frac{1}{2}) * \ldots * (1 - \frac{1}{p_k})$$

```
int phi(int n) {
    int result = n;
    for (int i = 2; i * i <= n; i++) {
        if (n % i == 0) {
            while (n % i == 0)
                n /= i;
            result -= result / i;
        }
    }
    if (n > 1)
        result -= result / n;
    return result;
}
```

2.20 PHI of 1 to N

```
const int N = 1e5 + 5;
vector<int> phi(N);
void phi_1_to_n() {
    for (int i = 0; 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;
        }
    }
}
Fact: Summation of phi of divisors of N is
equal to N. For example N = 10.
Divisors of 10 are 1, 2, 5, 10. Hence,
```

$\phi(1) + \phi(2) + \phi(5) + \phi(10) = 1 + 1 + 4 + 4 = 10$

2.21 nCr(more space, less time)

```
int mod = 1e9 + 7;
const int MAX = 1e7 + 5;
vector<int> fact(MAX), ifact(MAX), inv(MAX);
void factorial() {
    inv[1] = fact[0] = ifact[0] = 1;
    for (int i = 2; i < MAX; i++)
        inv[i]=inv[mod%i]*(mod-mod/i)%mod;
    for (int i = 1; i < MAX; i++)
        fact[i] = (fact[i - 1] * i) % mod;
    for (int i = 1; i < MAX; i++)
        ifact[i]=ifact[i-1]*inv[i] % mod;
}

int nCr(int n, int r) {
    if (r < 0 || r > n)
        return 0;
    return (int)fact[n] * ifact[r] % mod *
ifact[n - r] % mod;
}
// first call factorial() function
// then for nCr just call nCr(n,r)
```

2.22 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;
}
//For binaryExp we call 1.6 function
void inverse(){
    for (int i = 0; i < MAX; ++i)
        inv[i]=binaryExp(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 binaryExp(a, M - 2, M);
}
```

2.23 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);
}
```

2.24 Modular Operation

Addition:

```
int mod_add(int a, int b, int MOD = mod){
    a = a % MOD, b = b % MOD;
    return (((a + b) % MOD) + MOD) % MOD;
}
```

Subtraction:

```
int mod_sub(int a, int b, int MOD = mod){
    a = a % MOD, b = b % MOD;
    return (((a - b) % MOD) + MOD) % MOD;
}
```

Multiplication:

```
int mod_mul(int a, int b, int MOD = mod){
    a = a % MOD, b = b % MOD;
    return (((a * b) % MOD) + MOD) % MOD;
}
```

Division:

```
//call binary Exponential Function here.
int mminvprime(int a, int b) { return
binaryExp(a, b - 2, b); }
//call modular multiplication here.
int mod_div(int a, int b, int MOD = mod) {
    a = a % MOD, b = b % MOD;
    return (mod_mul(a, mminvprime(b, MOD),
MOD) + MOD) % MOD;
}
//only for prime MOD
```

2.25 Number of Set Bit from 1 to N

```
// number of set bits till 1 to N in log(N)
ll setBitsTillN(ll n) {
    ll cnt = 0;
    for (ll i = 1; i <= n; i = i << 1) {
        ll x = (n + 1) / (i << 1);
        cnt += x * i;
        if ((n + 1) % i and (n & i))

           cnt += (n + 1) % i;
    }
    return cnt;
}
```

2.26 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;
    }
}
```

2.27 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;
}
```

2.28 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;
}
```

2.29 Convex Hull

```
#define pld pair<ld, ld>
ll orientation(pld a, pld b, pld c) {
//Cal the determinant to determine orientat
    double v = a.first*(b.second-c.second)+
               b.first*(c.second-a.second)+
               c.first*(a.second-b.second);
    return (v < 0) ? -1 : (v > 0) ? 1 : 0;
// -1: clockwise, 1: counter-clockwise, 0:
collinear
}
bool cw(pld a,pld b,pld c, bool collinear) {
// Check if points (a, b, c) are clockwise or
collinear (if allowed)
    ll o = orientation(a, b, c);
    return o < 0 || (collinear && o == 0);
}

bool ccw(pld a, pld b,pld c,bool collinear){
// Check if points (a, b, c) are counter
   clockwise or collinear (if allowed)
    ll o = orientation(a, b, c);
    return o > 0 || (collinear && o == 0);
}

void convex_hull(vector<pld> &a, bool
collinear = false) {
    if (a.size() == 1)
```

```
        return;  // Single point, no convex
                   hull needed
// Step 1: Sort points by x-coordinate, then
by y-coordinate
    sort(a.begin(), a.end(),[](pld a,pld b){
        return make_pair(a.first, a.second)
             < make_pair(b.first, b.second);
    });
    pld p1 = a[0], p2 = a.back();
// Leftmost and rightmost points
    vector<pld> up = {p1}, down = {p1};
// Upper and lower hulls
    for (ll i = 1; i < (int)a.size(); i++) {
// Add to upper hull if last point or forms
clockwise turn
        if (i == a.size() - 1 ||
            cw(p1, a[i], p2, collinear)) {
            while (up.size() >= 2 &&
!cw(up[up.size() - 2], up[up.size() - 1],
a[i], collinear)) { up.pop_back(); }
            up.push_back(a[i]);
        }
// Add to lower hull if last point or forms
counter-clockwise turn
        if (i == a.size() - 1 ||
            ccw(p1, a[i], p2, collinear)) {
            while (down.size() >= 2 &&
!ccw(down[down.size() - 2], down[down.size()
- 1], a[i], collinear))
                { down.pop_back(); }
            down.push_back(a[i]);
        }
    }
// Handle special case:all points are
collinear
    if (collinear && up.size()==a.size()) {
        reverse(a.begin(), a.end());
        return;
    }
// Merge upper and lower hulls into the
result
    a.clear();
    for (ll i = 0; i < (int)up.size(); i++)
        a.push_back(up[i]);
    for (ll i = down.size() - 2; i > 0; i--)
        a.push_back(down[i]);
}
```

2.30 Line Intersection

```
int ori(const pair<int, int> &o, const
pair<int, int> &a, const pair<int, int> &b)
{
    int ret = (a.first - o.first) * (b.second
- o.second) -
                (a.second - o.second) *
(b.first - o.first);
    return (ret > 0) - (ret < 0);
}

bool isIntersect(const pair<int, int> &p1,
const pair<int, int> &p2,
                 const pair<int, int> &q1,
const pair<int, int> &q2)
{
    if (ori(p1, p2, q1) == 0 && ori(p1, p2,
q2) == 0 && ori(q1, q2, p1) == 0 && ori(q1,
q2, p2) == 0)
    {
        if (ori(p1, p2, q1))
            return false;
        return (p1.first - q1.first) *
(p2.second - q1.second) <= 0 ||
               (p1.second - q1.second) *
(p2.first - q1.first) <= 0 ||
               (q1.first - p1.first) *
(q2.second - p1.second) <= 0 ||
               (q1.second - p1.second) *
(q2.first - p1.first) <= 0;
    }
    return (ori(p1, p2, q1) * ori(p1, p2, q2)
<= 0) &&
           (ori(q1, q2, p1) * ori(q1, q2, p2)
<= 0);
}
```

3 Algorithms

3.1 KMP Algorithm-O(n+m)

```
vector<int> createLPS(string pattern) {
    int n = pattern.length(), idx = 0;
    vector<int> lps(n);
    for (int i = 1; i < n;) {
        if (pattern[idx] == pattern[i]) {
            lps[i] = idx + 1;
            idx++, i++;
        }
        else {
            if (idx != 0)
                idx = lps[idx - 1];
            else
                lps[i] = idx, i++;
        }
    }
    return lps;
}
int kmp(string text, string pattern) {
    int cnt_of_match = 0, i = 0, j = 0;
    vector<int> lps = createLPS(pattern);
    while (i < text.length()) {
        if (text[i] == pattern[j])
            i++, j++;// i->text,j->pattern
        else {
            if (j != 0)
                j = lps[j - 1];
            else
                i++;
        }
        if (j == pattern.length()) {
            cnt_of_match++;
            // the index where match found ->
(i - pattern.length());
            j = lps[j - 1];
        }
    }
    return cnt_of_match;
}
```

3.2 2D prefix sum

```
class NumMatrix {
    int row, col;
    vector<vector<int>> sums;
public:
    NumMatrix(vector<vector<int>> &matrix) {
        row = matrix.size();
        col = row>0 ? matrix[0].size() : 0;
        sums = vector<vector<int>>(row+1,
               vector<int>(col+1, 0));
        for(int i=1; i<=row; i++) {
            for(int j=1; j<=col; j++) {
                sums[i][j] =
      matrix[i-1][j-1] + sums[i-1][j] +
      sums[i][j-1] - sums[i-1][j-1] ;
            }
        }
    }
    int sumRegion(int row1, int col1,
                  int row2, int col2) {
                return sums[row2+1][col2+1] -
                sums[row2+1][col1] -
                sums[row1][col2+1] +
                sums[row1][col1];
    }
};
```

3.3 Kadane's Algorithm O(n)

```
// return maximum subarray sum.
int maxSubArraySum(vector<int> &a) {
    int size = a.size();
    int maxTill = INT_MIN, maxEnd = 0;
    for (int i = 0; i < size; i++) {
        maxEnd = maxEnd + a[i];
        if (maxTill < maxEnd)
            maxTill = maxEnd;
        if (maxEnd < 0)
            maxEnd = 0;
    }
    return maxTill;
}
```

3.4 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;
    }
};
```

3.5 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;
}
```

3.6 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();
}
```

4 Data Structure

4.1 SEGMENT TREE

```
class SEGMENT_TREE {
   public:
    vector<int> v;
    vector<int> seg;
    SEGMENT_TREE(int n) {
        v.resize(n + 5);
        seg.resize(4 * n + 5);
    }
    //! initially: ti = 1, low = 1, high = n
      //(number of elements in the array);
    void build(int ti, int low, int high) {
        if (low == high) {
            seg[ti] = v[low];
            return;
        }
        int mid = (low + high) / 2;
        build(2 * ti, low, mid);
        build(2 * ti + 1, mid + 1, high);
        seg[ti] = (seg[2*ti]+seg[2*ti+1]);
    }
    //! initially: ti = 1, low = 1, high = n
      //(number of elements in the array),
    //(ql & qr)=user input in 1 based
index;
    int find(int ti, int tl, int tr, int ql,
            int qr) {
        if (tl > qr || tr < ql) {
            return 0;
        }
        if (tl >= ql and tr <= qr)
            return seg[ti];
        int mid = (tl + tr) / 2;
        int l = find(2*ti, tl, mid, ql, qr);
        int r = find(2*ti+1,mid+1,tr,ql,qr);
        return (l + r);
    }
    //! initially: ti = 1, tl = 1, tr = n
      //(number of elements in the array),
      //id = user input in 1 based indexing,
      //val = updated value;
    void update(int ti, int tl, int tr, int
               id, int val) {
        if (id > tr or id < tl)
            return;
        if (id == tr and id == tl) {
            seg[ti] = val;
            return;
        }
        int mid = (tl + tr) / 2;
        update(2 * ti, tl, mid, id, val);
        update(2*ti+1,mid + 1, tr, id, val);
        seg[ti] = (seg[2*ti]+seg[2*ti + 1]);
    }
};
// use 1 based indexing for input and
//queries and update;
```

4.2 FENWICK TREE

```
// Sum
struct FenwickTree {
    vector<int> bit;  // binary indexed tree
    int n;
    FenwickTree(int n) {
        this->n = n;
        bit.assign(n, 0);
    }
    FenwickTree(vector<int>a):
                  FenwickTree(a.size()) {
        for (size_t i=0; i < a.size(); i++)
            add(i, a[i]);
    }
    int sum(int r) {
        int ret = 0;
        for (; r >= 0; r = (r&(r + 1)) - 1)
            ret += bit[r];
        return ret;
    }
    int sum(int l, int r) {
        return sum(r) - sum(l - 1);
    }
    void add(int idx, int delta) {
        for (; idx<n; idx = idx | (idx + 1))
            bit[idx] += delta;
    }
};
```

```
// minimum
struct FenwickTreeMin {
    vector<int> bit;
    int n;
    const int INF = (int)1e9;
    FenwickTreeMin(int n) {
        this->n = n;
        bit.assign(n, INF);
    }
    FenwickTreeMin(vector<int> a) :
                  FenwickTreeMin(a.size()) {
        for (size_t i=0; i < a.size(); i++)
            update(i, a[i]);
    }
    int getmin(int r) {
        int ret = INF;
        for (; r>=0; r = (r & (r + 1)) - 1)
            ret = min(ret, bit[r]);
        return ret;
    }
    void update(int idx, int val) {
        for (; idx<n; idx = idx | (idx + 1))
            bit[idx] = min(bit[idx], val);
    }
};
```

4.3 SEGMENT TREE LAZY

```
const int N = 1e5 + 100;
int tree[N << 2], lz[N << 2];
void propagate(int u, int st, int en) {
    if (!lz[u])
        return;
    tree[u] += lz[u] * (en - st + 1);
    if (st != en) {
        lz[2 * u] += lz[u];
        lz[2 * u + 1] += lz[u];
    }
    lz[u] = 0;
}
void update(int u, int st, int en, int l,
            int r, int x) {
    propagate(u, st, en);
    if (r < st or en < l)
        return;
    else if (st >= l and en <= r) {
        lz[u] += x;
        propagate(u, st, en);
    }
    else {
        int mid = (st + en) >> 1;
        update(2 * u, st, mid, l, r, x);
        update(2*u + 1, mid+1, en, l, r, x);
        tree[u] = tree[2*u]+tree[2*u+1];
    }
}
int query(int u,int st,int en,int l,int r){
    propagate(u, st, en);
    if (r < st or en < l)
        return 0;
    else if (st >= l and en <= r)
        return tree[u];
    else {
        int mid = (st + en) >> 1;
        int left=query(2*u, st, mid, l, r);
        int right=query(2*u+1,mid+1,en,l,r);
        return left + right;
    }
}
```

4.4 TRIE

```
class TrieNode {
   public:
    int isEnd;
    TrieNode *child[26];
    TrieNode() {
        isEnd = 0;
        for (int i = 0; i < 26; i++)
            child[i] = NULL;
    }
};
class Trie {
    TrieNode *root;

   public:
    Trie() : root(new TrieNode()) {}
    void insert(string word) {
        TrieNode *curr = root;
        for (char ch : word) {
            if(curr->child[ch-'a'] == NULL)
                curr->child[ch - 'a'] =
                    new TrieNode();
            curr = curr->child[ch - 'a'];
        }
        curr->isEnd++;
    }
    bool search(string word) {
        TrieNode *curr = root;
        for (char ch : word) {
            if(curr->child[ch-'a'] == NULL)
                return false;
            curr = curr->child[ch - 'a'];
        }
        return curr->isEnd;
    }
    bool startsWith(string prefix) {
        TrieNode *curr = root;
        for (char ch : prefix) {
            if (curr->child[ch-'a']==NULL)
                return false;
            curr = curr->child[ch - 'a'];
        }
        return true;
    }
    bool isJunc(TrieNode *curr) {
        for (int i = 0; i < 26; i++) {
            if (curr->child[i] != NULL)
                return true;
        }
        return false;
    }
    // 1 means junction delete kore asche
    bool dlt(string s, int idx,
             TrieNode *curr) {
        if (idx >= s.size())
            return 0;
        if (idx == s.size() - 1) {
           if (isJunc(curr->child[s[idx] -
                                  'a'])) {
                curr->child[s[idx] -
                            'a']->isEnd = 0;
                return false;
           }
           else {
             delete curr->child[s[idx]-'a'];
             curr->child[s[idx]-'a']= NULL;
                return true;
            }
        }
        bool res = dlt(s, idx + 1,
                       curr->child[s[idx] - 'a']);
        if (res) {
         if(isJunc(curr->child[s[idx]-'a']))
              return false;
         else if (!curr->child[s[idx] -
                    'a']->isEnd) {
             delete curr->child[s[idx]-'a'];
             curr->child[s[idx]-'a']=NULL;
             return true;
            }
        }
        return false;
    }
    bool dlt(string s) {
        if (search(s)) {
            dlt(s, 0, root);
            return true;
        }
        return false;
    }
   void print(string start, TrieNode *curr){
        if (curr->isEnd)
            cout << start << endl;
        for (int i = 0; i < 26; i++) {
            if (curr->child[i] != NULL) {
               start.push_back(i + 'a');
               print(start, curr->child[i]);
               start.pop_back();
            }
        }
    }
    void print() { print("", root); }
};
```

4.5 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)];
    }
};
```

3.6 String Hashing

```
// include binary exponential here.
const ll N = 2e5 + 5;
const ll MOD1 = 127657753, MOD2 = 987654319;
const ll p1 = 137, p2 = 277;
ll ip1, ip2;
pair<ll, ll> pw[N], ipw[N];
void prec() {
    pw[0] = {1, 1};
    for (ll 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 = binaryExp(p1, MOD1 - 2, MOD1);
    ip2 = binaryExp(p2, MOD2 - 2, MOD2);
    ipw[0] = {1, 1};
    for (ll 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 {
    ll n;
    string s;                   // 0 - indexed
    vector<pair<ll, ll>> hs;    // 1 - indexed
    Hashing() {}
    Hashing(string _s) {
        n = _s.size();
        s = _s;
        hs.emplace_back(0, 0);
        for (ll i = 0; i < n; i++) {
            pair<ll, ll> 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<ll, ll> get_hash(ll l, ll r) {
        // 1 - indexed
        assert(1 <= l && l <= r && r <= n);
        pair<ll, ll> 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<ll, ll> get_hash() {
        return get_hash(1, n);
    }
};
```

4.7 Order Set

```
#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;
```

4.8 GP Hash Table

```
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
template <typename p, typename q>
using ht = gp_hash_table<p, q>;
```

5 Dynamic Programming

5.1 LCS O(n*m)

```
string s = "abbced", t = "bedc";
cin >> s >> t;
int n = s.size(), m = t.size();
int dp[n + 5][m + 5];
memset(dp, 0, sizeof(dp));
for (int i = 1; i <= n; i++) {
    for (int j = 1; j <= m; j++) {
        if (s[i - 1] == t[j - 1]) {
            dp[i][j] = 1 + dp[i - 1][j - 1];
        }
        else {
            dp[i][j] = max(dp[i - 1][j],
                           dp[i][j - 1]);
        }
    }
}
```

Many problems can be solved using LCS techniques.

- Longest Increasing Substring
  To solve this, we just care about when two char equals. Rest of the things should be neglected.
- Longest Palindromic Subsequence(LPS)
  To solve this, we just take a new string which is the reverse of the original string. Then just call the LCS function to find LPS.
- Minimum insertions to make a string palindrome
  To solve this, we just basically do string length - LPS.
  Why this? Let's take an example:
  string s = aabca;
  Let's say **aca** is our LPS. Now we find how many char we need to insert to make the string palindrome while our LPS is fixed.
  a ab c a now to make the string palindrome we just need to insert the reverse of **ab** after c. So the new string looks like **a ab c ba a**
- Minimum Number of Deletions and Insertions to make the string equals
  To solve this we just find the LCS of those string then just do:
  n + m - 2 * LCS.length()
  where n, m = strings length

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;
/*
here we basically check is the current
element from v is greater than the last
element of the sequence.
 if it is then push it to the seq array and
if not then replace that index value.
let's take an example: v = 7 3 5 3 6 2 9 8
1st iteration seq = 7;
2nd iteration seq = 3;
3rd iteration seq = 3 5;
4th iteration seq = 3 3;
5th iteration seq = 3 3 6;
6th iteration seq = 2 3 6;
7th iteration seq = 2 3 6 9;
8th iteration seq = 2 3 6 8;
*/
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

```
// # of elements in the list for which you //
want to find the sum over all subsets
int n = 20;
// the list for which you want to find the //
sum over all subsets
vector<int> a(1 << n);

//answer for sum over subsets of each subset
vector<int> sos(1 << n);

for (int i = 0; i < (1 << n); i++) {
      // iterate over all other sets and
checks whether they're a subset of i
      for (int j = 0; j < (1 << n); j++) {
            if ((i & j) == j) {
                  sos[i] += a[j];
            }
      }
}
```

5.7 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;
}
```

5.8 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;
```

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;
            }
        }
    }
}
```

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)

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

class Graph {
    vector<pair<int, edge>> G, T;
    vector<int> parent;
    int cost = 0;

   public:
    Graph(int n) {
        for (int i = 0; i < n; i++)
            parent.push_back(i);
    }

    void add_edges(int u,int v,int wt) {
        G.push_back({wt, {u, v}});
    }

    int find_set(int n) {
        if (n == parent[n])
            return n;
        else
```

```
            return find_set(parent[n]);
    }

    void union_set(int u, int v) {
        parent[u] = parent[v];
    }

    void kruskal() {
        sort(G.begin(), G.end());
        for (auto it : G) {
         int uRep=find_set(it.second.first);
        int vRep=find_set(it.second.second);
            if (uRep != vRep) {
                cost += it.first;
                T.push_back(it);
                union_set(uRep, vRep);
            }
        }
    }

    int get_cost() { return cost; }
    void print() {
        for (auto it : T)
            cout << it.second.first << " "
                 << it.second.second << "->"
                 << it.first << endl;
    }
};

// g.add_edges(u, v, wt);
// g.kruskal();
```

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

```
// query O(logn)
// preprocessing O(n)
struct LCA {
    vector<ll> hi, euler, first, st;
    vector<bool> visit;
    ll n;
    LCA(vector<vector<ll>> &adj, ll root=0){
        n = adj.size();
        hi.resize(n);
        first.resize(n);
        euler.reserve(n * 2);
        visit.assign(n, false);
        dfs(adj, root);
        ll m = euler.size();
        st.resize(m * 4);
        build(1, 0, m - 1);
    }
    void dfs(vector<vector<ll>> &adj,
             ll node, ll h = 0) {
        visit[node] = true;
        hi[node] = h;
        first[node] = euler.size();
        euler.push_back(node);
        for (auto to : adj[node]) {
            if (!visit[to]) {
                dfs(adj, to, h + 1);
                euler.push_back(node);
            }
        }
    }
    void build(ll node, ll b, ll e) {
        if (b == e) {
            st[node] = euler[b];
        }
        else {
            ll mid = (b + e) / 2;
            build(node << 1, b, mid);
            build(node << 1 | 1, mid+1, e);
            ll l = st[node << 1];
             ll r = st[node << 1 | 1];
            st[node] = (hi[l]<hi[r])?l : r;
        }
    }
    ll query(ll node,ll b,ll e,ll L,ll R) {
        if (b > R || e < L)
            return -1;
        if (b >= L && e <= R)
            return st[node];
        ll mid = (b + e) >> 1;
        ll left=query(node << 1,b,mid,L,R);
        ll right = query(node << 1 | 1,
                           mid + 1, e,  L, R);
        if (left == -1)
            return right;
        if (right == -1)
            return left;
       return hi[left]<hi[right]?left:right;
    }


    ll lca(ll u, ll v) {
        ll left = first[u],;
         ll right = first[v];
        if (left > right)
            swap(left, right);
        return query(1, 0, euler.size() - 1,
left, right);
    }
};
```

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

```
unordered_map<int, vector<int>> adj, InvAdj;
stack<int> order;
unordered_map<int, bool> visited;
unordered_map<int, vector<int>> all_scc;
unordered_map<int, int> compId;
void dfs_for_start(int curr){
    visited[curr] = 1;
    for (auto i : adj[curr])
        if (!visited[i])
            dfs_for_start(i);
    order.push(curr);
}
vector<int> curr_comp;
void dfs_for_scc(int curr){
    visited[curr] = 1;
    for (auto i : InvAdj[curr])
        if (!visited[i])
            dfs_for_scc(i);
    curr_comp.push_back(curr);
}
inline void scc(){
    int n, e, u, v;
    cin >> n >> e;
    for (int i = 0; i < e; i++){
        cin >> u >> v;
        adj[u].push_back(v);
        InvAdj[v].push_back(u);
    }
    for (int i = 1; i <= n; i++)
        if (!visited[i])
            dfs_for_start(i);
    visited.clear();
    while (!order.empty()){
        if (!visited[order.top()]){
            curr_comp.clear();
            dfs_for_scc(order.top());
            int sz = all_scc.size() + 1;
            all_scc[sz] = curr_comp;
            for (auto i : curr_comp)
                compId[i] = sz;
        }
        order.pop();
    }
}
no. of ways and min cost of connecting the
sccs
const int MOD = 1e9 + 7, N = 1e5 + 2, INF =
1e18 + 2;
int n, m, comp[N];
vector<int> adj[N], rev[N];
bitset<N> vis;
void DFS1(int u, stack<int> &TS){
    vis[u] = true;
    for (int v : adj[u])
        if (!vis[v])
            DFS1(v, TS);
    TS.push(u);
}
void DFS2(int u, const int scc_no, int
&min_cost, int &ways, vector<int> &cost){
    vis[u] = true;
    comp[u] = scc_no;
    for (int v : rev[u])
        if (!vis[v]){
            if (min_cost == cost[v])
                ++ways;
            else if (min_cost > cost[v]){
                ways = 1;
                min_cost = cost[v];
            }
            DFS2(v, scc_no, min_cost, ways,
                 cost);
        }
}
signed main(){
    FIO cin >> n;
    vector<int> cost(n + 1);
    for (int i = 1; i <= n; ++i)
        cin >> cost[i];
    cin >> m;
    while (m--){
        int u, v;
        cin >> u >> v;
        adj[u].push_back(v);
        rev[v].push_back(u);
    }
    int tot = 0, ways = 1;
    stack<int> TS;
    for (int i = 1; i <= n; ++i)
        if (!vis[i])
            DFS1(i, TS);
    vis.reset();
    int scc_no = 0;
    while (!TS.empty()){
        int u = TS.top();
        TS.pop();
        if (!vis[u]){
            int tmp_cst = cost[u], tmp_ways =
1;
            DFS2(u, ++scc_no, tmp_cst,
                 tmp_ways, cost);
            tot += tmp_cst;
            ways = (ways * tmp_ways) % MOD;
        }
    }
    cout << tot << ' ' << ways;
}//TC: O(V+E)
```

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;}
```

7 Random Staff

7.1 Knight Moves

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

7.2 Rand function

```
#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);
}
```

7.3 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;
}
```

7.4 Matrix Exponentiation

```
#define vvi vector<vector<ll>>
ll n, m;
vvi matixMulti(vvi &a, vvi &b) {
    vvi res(n, vector<ll>(n, 0));
    for (ll i = 0; i < n; i++) {
        for (ll j = 0; j < n; j++) {
            for (ll k = 0; k < n; k++) {
                res[i][j] = (res[i][j] +
(a[i][k] * b[k][j]) % mod) % mod;
            }
        }
    }
    return res;
}

vvi martixExp(vvi &base, ll power) {
    vvi identity(n, vector<ll>(n, 0));
    for (ll i = 0; i < n; i++)
        identity[i][i] = 1;

    while (power > 0) {
        if (power % 2) {
            identity = matixMulti(base,
identity);
        }
        base = matixMulti(base, base);
        power /= 2;
    }
    return identity;
}
```

7.5 sqrt decomposition(MO's Algo)

```
// https://www.spoj.com/problems/DQUERY/
#include <bits/stdc++.h>
using namespace std;
const int SIZE_1 = 1e6 + 10, SIZE_2 = 3e4 +
10;
class query{
public:
    int l, r, indx;
};
int block_size, cnt = 0;
int frequency[SIZE_1], a[SIZE_2];
void add(int indx){
    ++frequency[a[indx]];
    if (frequency[a[indx]] == 1)
        ++cnt;
}
void sub(int indx){
    --frequency[a[indx]];
    if (frequency[a[indx]] == 0)
        --cnt;
}
```

```
bool comp(query a, query b){
    if (a.l / block_size == b.l / block_size)
        return a.r < b.r;
    return a.l / block_size < b.l /
block_size;
}
signed main(){
    int n; cin >> n;
    for(int i = 0; i < n; ++i) cin>>a[i];
    int q; cin >> q;
    int ans[q] = {};
    query Qur[q];
    for (int i = 0; i < q; ++i){
        int l, r; cin>>l>>r;

        Qur[i].l = l - 1;
        Qur[i].r = r - 1;
        Qur[i].indx = i;
    }
    block_size = sqrt(n); // sqrt(q) dileo
hobe, but n is more accurate
    sort(Qur, Qur + q, comp);

    int ML = 0, MR = -1;
    for(int i = 0; i < q; ++i) {
        int L = Qur[i].l;
        int R = Qur[i].r;
        // fixing right pointer
        while (MR < R) add(++MR);
        while (MR > R) sub(MR--);
        // fixing left pointer
        while (ML < L) sub(ML++);
        while (ML > L) add(--ML);
        ans[Qur[i].indx] = cnt;
    }
    for (int i = 0; i < q; ++i)
        cout << and[i] << '\n';
}//sqrt(n)
```

7.6 Meet in the middle

```
#include <bits/stdc++.h>
using namespace std;
int les_equal(vector<int> &s, int key){
    int size = s.size();
    int lo = 0, hi = size - 1, ans = 0;
    while (hi >= lo){
        int mid = lo + (hi - lo) / 2;
        if (s[mid] <= key){
            ans = max(ans, mid);
            lo = mid + 1;
        }
        else hi = mid - 1;
    }
    return ans;
}
signed main(){
    FIO int n, n1, n2, t;
    cin >> n >> t;
    n1 = (n + 1) / 2;
    n2 = n / 2;
    int a1[n1]; for(int &i: a1) cin>>i;
    int a2[n2]; for(int &i: a2) cin>>i;
    vector<int> set1, set2;
    for(int mask=0; mask < (1<<n1); ++mask){
        int temp_sum = 0;
        for (int i = 0; i < n1; ++i){
            int f = 1 << i;
            if (f & mask)
                temp_sum += a1[i];
        }
        set1.push_back(temp_sum);
    }
    for(int mask=0; mask < (1<<n2); ++mask){
        int temp_sum = 0;
        for (int i = 0; i < n2; ++i){
            int f = 1 << i;
            if (f & mask)
                temp_sum += a2[i];
        }
        set2.push_back(temp_sum);
    }
    sort(set2.begin(), set2.end());

    // for(auto itr: set2) cout<<itr<<' ';
    // cout<<'\n';
    // for(auto itr: set1) cout<<itr<<' ';
    // cout<<'\n';
    int and = 0;
    for (auto it : set1){
        int left = t - it;
        if (left < 0) continue;

        int indx = les_equal(set2, left);
        int temp_sum_set2 = (indx != -1 ? (it
+ set2[indx]) : 0);
        if (temp_sum_set2 <= t)
            ans = max(ans, temp_sum_set2);
    }
    cout<<ans;
}//TC: O(2^(LK+1))
```

7.7 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;
}
```

7.8 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);
```

Debug:

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

void __print(ll x) {cerr << x; }
void __print(float x) {cerr << x; }
void __print(char x) {cerr<<'\''<<x<<'\''; }
void __print(string x){cerr<<'\"'<<x<<'\"';}
void __print(bool x){
      cerr << (x ? "true" : "false"); }

template <typename T, typename V>
void __print(const pair<T, V> &x) {
    cerr << '{';
    __print(x.first);
    cerr << ',';
    __print(x.second);
    cerr << '}';
}
template <typename T>
void __print(const T &x) {
    int f = 0;
    cerr << '{';
    for (auto &i : x)
        cerr << (f++ ? "," : ""), __print(i);
    cerr << "}";
}
void _print() { cerr << ""; }
template <typename T, typename... V>
void _print(T t, V... v) {
    __print(t);
    if (sizeof...(v))
        cerr << ", ";
    _print(v...);
}

#define dbg(x...)                         \
    cerr << "[ " << #x << " ] = [ "; \
    _print(x);                            \
    cerr << " ]" << endl;
```