**********Our Template**********

```
#include <bits/stdc++.h>
#include <ext/pb_ds/assoc_container.hpp>
#include <ext/pb_ds/tree_policy.hpp>
using namespace __gnu_pbds;
using namespace std;

#define ll long long
#define fast ios::sync_with_stdio(false);cin.tie(nullptr)

template<class T> using special_set = tree<T,
null_type, less<T>, rb_tree_tag,
 tree_order_statistics_node_update>;
```

```
// String transformations
string toupper(string a) {
    for (char &c : a)
        if (islower(c))
            c = toupper(c);
    return a;
}
string tolower(string a) {
    for (char &c : a)
        if (isupper(c))
            c = tolower(c);
    return a;
}
```

```
// Factorial calculation
int factorial(int k) {
    int res = 1;
    for (int i = 2; i <= k; ++i)
        res *= i;
    return res;
}
ll calc_combinatoric(ll a, ll b, ll mod) {
    ll fact_a = factorial(a, mod);
    ll fact_b = factorial(b, mod);
    ll fact_a_b = factorial(a - b, mod);
    ll inv_fact_b = power_MOD(fact_b, mod - 2, mod);
    ll inv_fact_a_b = power_MOD(fact_a_b, mod - 2,
mod);
    return (fact_a * inv_fact_b % mod * inv_fact_a_b
% mod);
}
```

```
// Comparator for pairs
bool cmp(pair<int, int> a, pair<int, int> b) {
    if (a.first == b.first) {
        return a.second > b.second;
    } else {
        return a.first < b.first;
    }
}
```

```
// Bitwise operations
bool is_odd(ll x) {
    return x & 1;
}
bool check_kth_bit(ll x, ll k) {
    return x & (1LL << k);
}
ll set_kth_bit_1(ll x, ll k) {
    return x | (1LL << k);
}
ll set_kth_bit_0(ll x, ll k) {
    return x & ~(1LL << k);
}
ll multiply_by_power_of_2(ll x, ll k) {
    return x << k;
}
ll divide_by_power_of_2(ll x, ll k) {
    return x >> k;
}
int count_set_bits(int x) {
    return __builtin_popcount(x);
}
ll count_set_bits_ll(ll x) {
    return __builtin_popcountll(x);
}
ll toggle_kth_bit(ll x, ll k) {
    return x ^ (1LL << k);
}
bool is_power_of_2(ll x) {
    return x && !(x & (x - 1));
}
```

```
bool find_str(string str1, string str2) {
    if (str1.find(str2) != string :: npos) {
        return true;
    } else {
        return false;
    }
}
ll floor_val (ll n, ll a) {
    return n / a;
}
ll ceil_val (ll n, ll a) {
    return (n + a - 1) / a;
}
ll round_val (ll n, ll a) {
    return (n + a / 2) / a;
}

void solve () {
    special_set<int> st;
    ...
}

int main() {
    fast;
    int q;
    cin >> q;

    while (q--) {
        solve();
    }

cerr << "Execution time: " << 1000.f * clock() /
CLOCKS_PER_SEC << " ms." << endl;

    return 0;
}
```

Math Formula

Parallel :

(1) Average / subsequence / normal number of 1 to n,
=> Value = (1st + last) / 2
(2) Nth value = a + (n - 1) * d
a = 1st number
n = Nth
d = normal difference
Nth value = last number
(3) Sum-> Sn = (n/2)(2a+(n-1)*d)
1+2+3+..+n = (n + 1) / 2
1^2+2^2+..+n^2 = n(n + 1)(2n + 1) / 6
1^3+2^3+..+n^3 = n(n + 1) / 2
2+4+6+..+2n = n(n + 1)
1+3+5+..+(2n - 1) = n^2
Harmonic Series : 1 + (1 / 2) + (1 / 3) + (1 / 4) +..

Multi-dimensional :
(1) Nth number = Ar^(n - 1)
(2) Sum of 1 to n number sum,
If (r > l) Sn = a(r^n - 1) / (r - 1)
If (r < l) Sn = a(l - r^n) / (l - r)

SQUARE :
1. (a + b)^2 = a^2 + 2ab + b^2
2. (a + b)^2 = (a - b)^2 - 4ab
3. (a - b)^2 = a^2 - 2ab + b^2
4. (a - b)^2 = (a + b)^2 -4ab
5. a^2 + b^2 = (a + b)^2 - 2ab
6. a^2 + b^2 = (a+b)^2+(a-b)^2 / 2
7. a^2 - b^2 = (a - b)^2 + 2ab
8. a^2 - b^2 = (a + b)(a - b)
9. 2(a^2+b^2) = (a+b)^2 + (a-b)^2
10. 4ab = (a+b)^2 - (a-b)^2
11. Ab = (a+b / 2)^2 - (a-b / 2)^2
12. (a+b+c)^2=(a^2+b^2+c^2)+2ab+2bc+2ca
13. 2(ab+bc+ca) = (a+b+c)^2-(a^2+b^2+c^2)

CUBE:
1. (a+b)^3 = a^3+3a^2b+3ab^2+b^3
2. (a-b)^3 = a^3 - 3a^2b +3ab^2 - b^3
3. a^3 + b^3 = (a + b)^3 - 3ab(a + b)
4. a^3 + b^3 = (a + b)(a^2 - ab + b^2)
5. a^3 - b^3 = (a - b)^3 + 3ab(a - b)
6. a^3 - b^3 = (a - b)(a^2 + ab + b^2)

Ex : 1+2+3+...+n Here, 1 to n all sum,
even_cnt = n / 2
even sum = even_cnt*(even_cnt + 1)
odd_cnt = (n + 1) / 2
odd sum = odd_cnt * odd_cnt

```
          1
        1   1
      1   2   1
    1  3     3  1
  1   4   6   4  1
```

1. nPr = n! / (n - r)!
2. nCr = n! / r! * (n-r)!

=> (x - 2)^3 = x^3(-2)^0+3x^2(-2)^1+3x^1(-2)^3+ 1x^0(-2)^3.

POWER :
1. a^m * a^n = a^m+n
2. a^m/a^n = a^m-n
3. (ab)^m = a^m * b^m
4. (a/b)^m = a^m / b^m
5. (a^m)n = a^mn
6. m*sqrt(a) = a * (1 / m)
7. a != 0 then, a^0 = 1

LOG :
1. Log a^a = 1
2. Log a^1 = 0
3. Log a^a^n =m or, n*loga^a=m
4. Log (m / n) = loga^m - loga^n
5. Log a^mn = loga^m + loga^n
6. Log a^m^n = n * log a^m
7. Log a^m = loga^m * loga^b

RATIO :
(1) a:b = c:d So, a/b = c/d So, b/a = d/c
a:b = b+c
a/b = b/c
b^2 = ac
(2) a:b = c:d , a/b = c/d then, a/c = b/d

Divisibility :
2 = last digit div by 2 or not
3 = sum of digit div by 3 or not
4 = last two div by 4 or not
5 = last dig 0 or 5
6 = If div by both 2 and 3
7 = last dig -> remove double it -> Subtract from remaining -> If by 7 or not
8 = last three dig div by 8 or not
9 = sum of dig div by 9 or not
11 = 4123 % 11
= ((4*10^3)+(1*10^2)+(2*10^1)+(3*10^3))
= (4*(-1) + 1 + (2*(-1) + 3) % 11
If this sum is div by 11 or not.
(odd index will mul by -1 and even will 1)
12 = div by 3 and 4 or not
Ex : n! Time d is it div by 3, 7, 9?(d>=1 && d<=9)

(1) (n! * d) % 3 == 0 (d%3 == 0) or (n>=3)
(2) (d == 7 && n == 3) OR (n>2)
(3) (n < 6) {
ll val = factorial(n);
ll lol = val * d;
If (digitSum(lol) % 9 == 0) “divisible”
} else {
if(n >= 6) “Divisible”

Formula Area :
(1) Square Corner = n * sqrt(2)
(2) Area = n^2
(3) Porishima = 4n
(4) Rectangle Corner = sqrt(a^2+b^2)
(5) Area = a*b
(6) Porishima = 2(a+b)
(7) Samakoni / triangle area = (1/2)*a*b
(8) Shomobahu = sqrt(3)/4 * a^2
(9) Shomodibahu = (b/4)sqrt(4a^2-b^2)
(10) triangle’s area, A = sqrt(s(s-a)(s-b)(s-c))
semi perimeter of triangle, S = (a+b+c)/2
(11) If we know three length a, b, c of triangle inside circle area is,
R=(abc)/sqrt((a+b+c)(b+c-a)(c+a-b)(a+b-c))
(12) circle inside triangle = A / S

Bhaskara, x = (-b +- sqrt(b^2 - 4ac)) / 2a

Modular Arithmetic :
(1) (a + b) % m = ((a % m) + (b % m)) % m
(2) (a * b) % m = (1ll * (a % m) * (b % m)) % m
(3) (a - b) % m = ((a % m) - (b % m) + m) % m
(4) (a^b)%m = 1, b = 0, (a^(b/2))^2, b even
a*(a^((b-1)/2))^2, b odd

Number Theory

(1) Prime Check :

```
bool prime (int n) {
    for (int i = 2; i <= sqrt(n); i++) {
        if (n % i == 0) return false;
    }
    return true;
}
```

(2) sieve of Erothenotenes :

```
vector<bool> is_prime;
vector<ll> primes;

void sieve(ll n) {
    is_prime.resize(n + 1, true);
    is_prime[0] = is_prime[1] = false;

    for (ll i = 2; i * i <= n; i++) {
        if (is_prime[i]) {
            for (ll j = i * i; j <= n; j += i) {
                is_prime[j] = false;
            }
        }
    }

    for (ll i = 2; i <= n; i++) {
        if (is_prime[i]) {
            primes.push_back(i);
        }
    }
}
```

(3) Prime Fact :

```
void primeFact(int n) {
    for (int i = 2; i <= sqrt(n); i++) {
        if (n % i == 0) {
            int cnt = 0;
            while (n % i == 0) {
                cnt++;
                  n = n / i;
            }
            cout << i << "^" << cnt;
            if (n > 1) cout << " * ";
        }
    }
    if (n > 1) {
        cout << n << "^1";
    }
}
```

(4) Binary Exponentiation :

```
ll Binary_expo(ll a,ll b, ll mod) {
    if(b == 0) return 1;
    if(b == 1) return a % mod;

    ll temp = power(a, b/2, mod);

    if(b % 2 == 0) {
        return (temp * temp) % mod;
    }
    else {
        return (((temp * temp) % mod) * a) % mod;
    }
}
```

(5) SOD, NOD :

```
void countAndSumDivisors(ll n) {
    set<ll> st;
    ll NOD = 0, SOD = 0;
    for (ll i = 1; i <= sqrt(n); i++) {
        if (n % i == 0) {
            ll x = i;
            ll y = n / i;
            st.insert(x), st.insert(y);

            SOD += x;
            if (x != y) SOD += y;
        }
    }
    NOD = st.size();
    cout << SOD << " " << NOD << endl;
}
```

(6) Segmented Sieve :

```
vector<int> primes;
// Regular sieve to find all primes up to sqrt(r)
void sieve(int limit) {
    vector<bool> isPrime(limit + 1, true);
    isPrime[0] = isPrime[1] = false;

    for (int i = 2; i * i <= limit; i++) {
        if (isPrime[i]) {
            for (int j = i * i; j <= limit; j += i) {
                isPrime[j] = false;
            }
        }
    }
    for (int i = 2; i <= limit; i++) {
        if (isPrime[i]) {
            primes.push_back(i);
        }
    }
}
// Seg. sieve to find primes in the range [l, r]
void segmentedSieve(ll l, ll r) {
    if (l == 1) l++;

    ll limit = sqrt(r) + 1;
    sieve(limit);

    int size = r - l + 1;
    vector<bool> isPrime(size, true);

    for (int p : primes) {
```

```
        ll start = max((ll)p * p, l + ((p - l % p) % p));
        for (ll j = start; j <= r; j += p) {
            isPrime[j - l] = false;
        }
    }
    for (int i = 0; i < size; i++) {
        if (isPrime[i]) {
            cout << l + i << endl;
        }
    }
}
```

(7) GCD & LCM :

```
ll gcd(ll a, ll b) {
    while (b != 0) {
        ll temp = b;
        b = a % b;
        a = temp;
    }
    return a;
}

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

(8) Big Divisible :

```
void Big_div (string s, int m) {
    int ans = 0;

    for (int i = 0; i<s.size(); i++) {
        ans = ans * 10 + (s[i] - '0') % m;
    }
    if (ans % m == 0) {
        cout << "Yes" << endl;
    } else {
        cout << "No" << endl;
    }
}
```

(9) Big Mod :

```
int bigmod(string base, string expo, int mod) {
    if (mod == 1)
        return 0;

    int res = 1, base = 0, expNum = 0;
    for (int i = 0; i < base.size(); i++) {
        base = (base * 10 + (base[i] - '0')) % mod;
    }
    for (int i = 0; i < expo.size(); i++) {
        expNum = (expNum * 10 + (expo[i] - '0')) %
(mod - 1);
    }
    while (expNum > 0) {
        if (expNum % 2 == 1) {
            res = (res * base) % mod;
        }
        expNum /= 2;
        base = (base * base) % mod;
    }
    return res;
}

int main() {
    string a, b="1";
    ll mod;
    cin >> a >> mod; // mod = 1000000007
    cout<< bigmod(a, b, mod) << endl;

    return 0;
}
```

(10) Big Subtractor :

```
string BigSub(string x, string y) {
    reverse(x.begin(), x.end());
    reverse(y.begin(), y.end());

    int len1 = x.size(), len2 = y.size();

    if (len1 > len2) {
        int gap = len1 - len2;
        while (gap--) {
            y.push_back('0');
        }
    } else if (len2 > len1) {
        int gap = len2 - len1;
        while (gap--) {
            x.push_back('0');
        }
    }
    int n = x.size(), carry = 0;
    string sub = "";

    for (int i = 0; i < n; i++) {
        int p = (x[i] - '0') - carry;
        int q = (y[i] - '0');

        if (p < q) {
            p += 10;
            carry = 1;
        } else {
            carry = 0;
        }
        int r = p - q;
        char lastDigit = r + '0';
        sub.push_back(lastDigit);
    }
    while (sub.size() > 1 && sub.back() == '0') {
        sub.pop_back();
    }
    reverse(sub.begin(), sub.end());
    return sub;
}
```

(11) Big Summation :

```
string BigSum(string x, string y) {
    reverse(x.begin(), x.end());
    reverse(y.begin(), y.end());
    int len1 = x.size(), len2 = y.size();

    if (len1 > len2) {
        int gap = len1 - len2;
        while (gap--) {
            y.push_back('0');
```

```
        }
    } else {
        int gap = len2 - len1;
        while (gap--) {
            x.push_back('0');
        }
    }
    int n = x.size(). carry = 0;
    string sum = "";

    for (int i = 0; i < n; i++) {
        int p = (x[i] - '0') + carry;
        int q = (y[i] - '0');
        int r = p + q;
        char lastDigit = (r % 10) + '0';
        sum.push_back(lastDigit);
        carry = r / 10;
    }
    if (carry > 0) {
        sum.push_back(carry + '0');
    }
    reverse(sum.begin(), sum.end());

    return sum;
}
```

---

(12) Big Multiplication :

```
string multiply(string num1, string num2) {
    int len1 = num1.size(), len2 = num2.size();
    string result(len1 + len2, '0');

    for (int i = len1 - 1; i >= 0; --i) {
        int carry = 0;
        for (int j = len2 - 1; j >= 0; --j) {
            int product = (num1[i] - '0') * (num2[j] - '0')
+ carry + (result[i + j + 1] - '0');
            carry = product / 10;
            result[i + j + 1] = (product % 10) + '0';
        }
        result[i] += carry;
    }
    result.erase(0, result.find_first_not_of('0'));

    if (result.empty()) {
        return "0";
    }
    return result;
}
```

---

(13) Find Factorial :

```
ll multiply(ll x, ll res[], ll res_size) {
    ll carry = 0;
    for (int i = 0; i < res_size; i++) {
        ll prod = res[i] * x + carry;
        res[i] = prod % 10;
        carry = prod / 10;
    }
    while (carry) {
        res[res_size] = carry % 10;
        carry = carry / 10;
        res_size++;
    }
    return res_size;
}

void factorial(ll n) { // if N <= 15000
    ll res[5000];
    res[0] = 1;
    ll res_size = 1;

    for (int x = 2; x <= n; x++) {
        res_size = multiply(x, res, res_size);
    }

    cout << "Factorial of " << n << " is: ";
    for (int i = res_size - 1; i >= 0; i--) {
        cout << res[i];
    }
    cout << endl;
}

int main() {
    int N;
    cout << "Enter a number: ";
    cin >> N;

    factorial(N);
    return 0;
}
```

Problems

---

(1) Least Prime Factor, Greatest Prime Factor, Distinct Prime Factor, Total Prime Factor, SOD, NOD.

```
const int N = 1e6+9;
int spf[N];
// Precompute the smallest prime factor (spf) for
every number up to N
void sieve() {
    for (int i = 2; i <= N; i++) {
        spf[i] = i;
    }
    for (int i = 2; i <= N; i++) {
        if (spf[i] == i) {  // Only prime numbers
            for (int j = i * 2; j <= N; j += i) {
                spf[j] = min(spf[j], i);
            }
        }
    }
}
// Function to get the least prime factor (lpf) of a
number

int LPF(int x) {
    return spf[x];
}
// Function to get the greatest prime factor (gpf)
of a number

int GPF(int x) {
    int ans = 0;
    while (x > 1) {
        int p = spf[x];
```

```
        ans = max(ans, p);
        while (x % p == 0) {
            x /= p;
        }
    }
    return ans;
}
// Function to get the number of distinct prime
factors

int distinctPrimeFactors(int x) {
    int distinct_count = 0;
    while (x > 1) {
        int p = spf[x];
        distinct_count++;
        while (x % p == 0) {
            x /= p;
        }
    }
    return distinct_count;
}
// Function to get the total number of prime

factors (including their powers)
int totalPrimeFactors(int x) {
    int total_count = 0;
    while (x > 1) {
        int p = spf[x];
        while (x % p == 0) {
            total_count++;
            x /= p;
        }
    }
    return total_count;
}
// Function to calculate the number of divisor of x

int numberOfDivisors(int x) {
    int num_divisors = 1;
    while (x > 1) {
        int p = spf[x];
        int power_of_p = 0;
        while (x % p == 0) {
            power_of_p++;
            x /= p;
        }
        num_divisors *= (power_of_p + 1);
    }
    return num_divisors;
}
// Function to calculate the sum of divisors of x

ll sumOfDivisors(int x) {
    ll sum_divisors = 1;
    while (x > 1) {
        int p = spf[x];
        int prime_power = 1;
        while (x % p == 0) {
            prime_power *= p;
            x /= p;
        }
        sum_divisors *= (1LL * (prime_power * p - 1)
/ (p - 1));
    }
    return sum_divisors;
}
int main() {
    sieve();

    int n;
    cin >> n;

    while (n--) {
        int q;
        cin >> q;
        int lpf = LPF(q);
        int gpf = GPF(q);
        int dist_prime_fact =
distinctPrimeFactors(q);
int total_prime_fact = totalPrimeFactors(q);
int NOD = numberOfDivisors(q);
ll SOD = sumOfDivisors(q);

cout << lpf << " " << gpf << " " << dist_prime_fact
<< " "<< total_prime_fact << " " << NOD << " "
<< SOD << endl;
    }
    return 0;
}
```

---

(2) Given an array of N length.Q query and in each query have given [L, R].we need to print gcd excluding part from range [L, R] 1<= N, Q <= 10^5.

```
int main () {
    int tc;
    cin >> tc;

    while (tc--) {
        int n, q;
        cin >> n >> q;
        int arr[n+3];

        for (int i = 1; i <= n; i++) {
            cin >> arr[i];
        }
        int pre[n+2], post[n+2];
        pre[0] = 0, post[0] = 0;

        for (int i = 1; i <= n; i++) {
            pre[i] = gcd(pre[i-1], arr[i]);
        }

        for (int i = 1; i <= n; i++) {
            post[i] = gcd(post[i+1], arr[i]);
        }
        while (q--) {
            int l, r;
            cin >> l >> r;
            int ans = gcd(pre[l-1], post[r+1]);
            cout << ans << endl;
        }
    }
    return 0;
```

---

(3) Given A:B1, B2:C, Find A:C?

```
ll a, b1, b2, c;
cin >> a >> b1 >> b2 >> c;

ll g1 = gcd(a, b1);
a/= g1, b1 /= g1;
```

```
ll g2 = gcd(b2, c);
b2 /= g2, c /= g2;

if (b1 == b2) {
   cout << a << " " << c << endl;
} else {
   ll lcm_b = lcm(b1, b2);
   a *= (lcm_b / b1);
   c *= (lcm_b / b2);
   ll g = gcd(a, c);
   a /= g, c /= g;

   cout << a << " " << c << endl;
}
```

---

(4) You are given a positive integer N. Check whether the number N is representable as the sum of the cubes of two positive integers. such that, a^3+b^3 = N.

```
void solve() {
   ll n, k =0; cin >> n;
   for (ll i=1; i<=cbrt(n); i++) {
      ll p = n-i*i*i;
      k = cbrt(p);
      if (k != 0 && k*k*k == p) {
         cout << "YES" << endl;
         break;
      }
      k = 0;
   }
   if (k ==0) {
      cout << "NO" << endl;
   }
}
```

(5) GIven A, B find maximum gcd of x, y. A <= x, y <= B.

```
cin >> a >> b;
for (i = b; ; i–){
   if((a+i-1) / i < (b / i)) {
      cout << i << endl;
      break;
   }
}
```

---

(6) Count number of pair indices(i, j) such that i<j and aj - ai = j - i;

```
int n;
cin >> n;
ll arr[n+1];
map<ll, ll> mp;
for (int i=1; i<=n; i++){
   cin >> arr[i];
   cnt += mp[arr[i] - i]++;
}
cout << cnt << endl;
```

---

(7) GIven array n & value m. we need to find how many consecutive subset are divisible by m. (Prefix Sum)

```
input -> n, m;
unordered_map<ll, ll> freq;
freq[0] = 0;
ll pref = 0, cnt = 0;
for (int i=0; i<n; i++) {
   pref += v[i];
   ll rem = pref % m;
   if(rem < 0) rem += m;
   if (freq.count(rem) > 0) {
      cnt += freq[rem];
   }
freq[rem]++;
}
cout << cnt << endl;
```

---

(8) Solve x^2 + sqrt(x) = c find value of X.

```
cin >> c;
double l = 0, r = 1e18;
for (int i=0; i<=200; i++) {
   double mid = (l + r) / 2.0;
   if ((mid ** mid) + sqrt(mid) <= c) {
      l = mid;
      ans = mid;
   }
   else {
      r = mid;
   }
}
cout << setprecision(12) << ans < endl;
```

---

(9) A, B given now find the maximum gcd of x, y . (A <= x <= y <= B)

```
cin >> a >> b;

for (int i=b; ; i –) {
   if ((a+i-1) / i < (b / i)) {
      cout << i << endl;
      break;
   }
}
```

Some Tricks :

Divisor :

1. How to find which number has odd number of divisor.
sol-> if (sqrt(n) * sqrt(n) == n) yes;

2. Print 1 to 1e12 those have odd divisor.
sol-> 1^2, 2^2, 3^2...

3. given N and need to print number of divisor.
sol-> N's prime faact's each value power + 1.

->Precompute smallest prime factors (SPF) up to N (using sieve).
->Factorize N using SPF (take O(nlogn))
->Calculate the number of divisors using the

formula.(total *= (count + 1))

4. N have x divisors, task is sum of these divisor,
sol->
-2^3 * 5^2
-1(1+5+5^2)+
2(1+5+5^2)+
2^2(1+5+5^2)+
2^3(1+5+5^2)
-(1+2+2^2+2^3) + (1+5+5^2)

if : p1^e1 * p2^e2 * p3^e3..
formula : ((p1^e1 - 1) / (p1-1)) * ((p2^e2 - 1) / (p2-1)) * ...

Prime :

---

1. Print 1 to N all number those have 1, 2, 3 divisor.
sol-> 1 have 1 divisor, prime number have 2 divisor, prime^2 have 3 divisor so on.

2. Need to print 1 to N all divisors and sum divisors of all value.
Code :
```
int N = 100;
int d[104], sum[104];
// O(N*log(logN))

for (int i=1; i<=N; i++) {
   for (int j=i; j<=n; j+=i) {
      d[j]++;
      sum[j] += i;
   }
}
```

3. For q queury take N and print N's prime fact q, n in worst 1e6. (Sieve Factorization)
code : O(logN)
```
const int N = 1e6+9;
int spf[N];

int main() {
   for (int i=2; i<N; i++) {
      spf[i] = i;
   }
   for (int i=2; i<N; i++) {
      for (int j=i; j<N; j+=i) {
         spf[j] = min(spf[j], i);
      }
   }
   int q; cin >> q;

   while (q--) {
      int n; cin >> n;
      vector<ll> ans;
      while(n > 1) {
         ans.pb(spf[n]);
         n /= spf[n];
      }
      for (auto x : ans) {
         cout << x << " ";
      }
      cout << endl;
   }
}
```

Tricks :

---

1. How many Trailing zero have in N!.
sol-> In N! have 10*10*10 for we find trailing zero. So, 10 mean 2^a * 5^b. ans is min(a, b).
code :
```
int countTrailingZeros(int N) {
   int count = 0;
   for (int i = 5; N / i >= 1; i *= 5) {
      count += N / i;
   }
   return count;
}
```

2. How many divisor have in N!.
sol-> find prime fact of N then apply,
N! = x^n * y^m * z^o = (n+1)*(m+1)*(o+1)

3. Make a array where for all subarray product's are divisible by it's length.
sol-> if subarray length is divisible by any value inside subarray then it can.
ex : 1, 2, 3, 4 is divisible by 4.

4. L to R how many number divisible by m.
sol-> 1 to R divisible by m is cnt1, 1 to L-1 divisible by m is cnt2. now ans is (cnt1 - cnt2)

5. Given an array size N, M=10^5, ai <=10^9. print how many pair is (ai + aj) % M = 0.
sol-> here, (ai + aj) % M = 0 means, ai % M = aj % M.
we will go ahead and check how many present (M - a[i]) before a[i].

0, 1, 2, 3, 1, 2, 5 (m = 5)
a0 = 0,
a1 = 1, 5-1 = 4 not present
a2 = 2, 5-2 = 3 not present
a3 = 3, 5 - 3 = 2 present cnt = 1
a4 = 1, 5 - 1 = 4 not present
a3 = 2, 5 - 2 = 3 present cnt = 2

6. Given array need to print how many subarray are divisible by m.
sol-> first using prefix sum we will make sum array also will mod by m. then i will check how many have,
pj - pi = 0, (pj = pi-1)

7. In N how many odd divisor exist.
sol-> Firstlt we will find N's prime fact then will just remove 2's part and others are ans.

8. Given N and need to say can we show n equal sum of 2 prime.

sol-> firstly we will find out all prime of N.
then, we will check a+b = n,, b = n-a.
code :

```
sieve() {
...
}
bool ok (int n) {
   for (int i=2; i<n; i++) {
      if (spf[i]==i && spf[n-i]==n-i) {
         return true;
      }
   }
   return false;
}
```

9. Given N and need to present N in minimum prime number sum.
sol-> if N is prime then ans is 1, else we will take 2 and remain part N-2 = X.
if X not prime then we will try to make X into 2 prime.

10. How many digit in a number N!.
sol-> if n = 10^5 then,
-log(n!) is huge. so,
-log10(1*2*3*...*n)
-log10^1 + log10^2 + ... + log10^n. check brute force then add 1. and show ans in floor value.

11. n, ai = 10^5 need to print maximum subset of number that's gcd is 1. else print -1.
sol-> we have to find just 2 number so that their gcd become 1.
hint : 2^0 + 2^1 + 2^2 + 2^3 + 2^4...+2^n < 2^n+1.

12. Given N we need to show minimum power of 2. like, 10 : 2^1 + 2^3, 2^1 + 2^2 + 2^2. (ans is 2^1 + 2^3)
sol-> we will find max power then remaining was 1 like, for 10 - 2^3 = 8 remain 2^1.

13. GIven N. need to print a, b. so that Lcm(a, b) is smallest between 1<=a,b<=n all possible pair.
(a+b == n) n<=10^9.(the lcm of a, b is as small as possible).
sol-> if prime (1, n-1), if even (n/2, n/2), else find smallest (s),
then largest divisor (d = n/s) ans pair will be (d, n-d).

14. Given 2 array a, b. our task is output num of minimum inversion or swap so that a become b.
sol-> count at each index value that how many value are strictly greater then that previously value.

Graph

Graph representation,

---

list,

```
vector <int> graph[10000];
while (e--) {
   int u, v;
   cin >> u >> v;
   graph[u].push_back(v);
   graph[v].push_back(u);
}

for (int i = 1; i <= n; i++) { // highlight graph
   cout << i;
   for (int j = 0; j<graph[i].size(); j++) {
      cout << "->" << graph[i][j];
   }
   cout << endl;
}
```

Matrix,

```
int graph[n+1][n+1];

for (int i=1; i<=n; i++) {
   for (int j=1; j<=n; j++) {
      if (i == j) {
         graph[i][j] = 0;
      }
      else {
         graph[i][j] = -1;
      }
   }
}
while (e--) {
   int u, v; cin>>u>>v;
   graph[u][v] = 1;
   graph[v][u] = 1;
}
for (int i=1; i<=n; i++) {
   for (int j=1; j<=n; j++) {
      cout << graph[i][j] << " ";
   }
   cout << endl;
}
```

DFS

```
vector<int> graph[1002];
int vis [1003];

void DFS (int node){
   vis[node] = 1;
   cout << node << " -> ";

   for(int i = 0; i < graph[node].size(); i++){
      int child = graph[node][i];

      if(vis[child] == 0){
         DFS (child);
      }
   }
}
```

BFS

```
vector<int> graph[1002];
int vis[1002] = {0};

void BFS(int root) {
    queue<int> q;
    q.push(root);

    while(!q.empty()) {
        int node = q.front();
        q.pop();

        if(vis[node] != 0) {
            continue;
        }
        vis[node] = 1;
        cout << node << " -> ";

        for (int i=0; i<graph[node].size(); i++) {
            int child = graph[node][i];

            if (vis[child] == 0) {
                q.push(child);
            }
        }
    }
}
```

Connected Component

```
vector <int> v[100005];
int vis[100005];

void DFS (int node) {
    vis[node] = 1;
    for (int child : v[node]) {
        if (vis[child] == 0) {
            DFS(child);
        }
    }
}

int main () {
    ...
    for (int i = 1; i <= n; i++) {
        if (vis[i] == 0) {
            cnt++;
            DFS(i);
        }
    }
    cout << cnt << endl;
}
```

Bipartite Check

```
vector <int> graph[10001];
int vis[10001], col[10001];

bool dfs(int node, int c) {
    vis[node] = 1;
    col[node] = c;

    for (auto child : graph[node]) {
        if (vis[child] == 0) {
            if (dfs(child, c^1) == false) {
                return false;
            }
        }
        else {
            if (col[node] == col[child]) {
                return false;
            }
        }
    }
    return true;
}

int main () {
    ...
    if (dfs(1, 0) == true) {
        cout << "bipartite" << endl;
    }
    else {
        cout << "not bipartite" << endl;
    }
```

Cycle Detection

```
vector <int> graph[10001];
int vis[10001];

bool dfs (int node, int par) {
    vis[node] = 1;

    for (auto child : graph[node]){
        if (vis[child] == 0) {
            if (dfs(child, node) == true){
                return true;
            }
        }
        else {
            if (child != par) {
                return true;
            }
        }
    }
    return false;
}

int main () {
    ...
    if (dfs(1, 0) == false) {
        cout << "No cycle exist"<<endl;
    } else {
        cout << "Cycle exist"<<endl;
    }
}
```

Single source shortest path

```
vector<int> graph[10001];
int vis[10001], dist[10001];

void bfs(int node) {
    queue<int> q; q.push(node);
    vis[node] = 1, dist[node] = 0;
```

```
    while (!q.empty()) {
        int current = q.front();
        q.pop();

        for (auto child : graph[current]) {
            if (!vis[child]) {
                vis[child] = 1;
                dist[child] = dist[current] + 1;
                q.push(child);
            }
        }
    }
}

int main() {
    ...
    bfs(1);
}
```

Finding Diameter of Tree / Longest Path

```
vector <int> graph[10001];
int vis[10001];
int maxD, maxnode;

void dfs(int node, int d) {
    vis[node] = 1;

    if (d > maxD) {
        maxD = d;
        maxnode = node;
    }
    for (auto child : graph[node]) {
        if (vis[child] == 0) {
            dfs(child, d+1);
        }
    }
}
int main () {
    ...
    maxD = -1;
    dfs(1, 0);

    for (int i=1; i<=n; i++) {
        vis[i] = 0;
    }
    maxD = -1;
    dfs(maxnode, 0);

    cout << maxD << endl;
    return 0;
}
```

Disjoint Set Union

```
const int N = 1e5 + 7;
int parent[N];
int component_size[N];

void make(int v) {
    parent[v] = v;
    component_size[v] = 1;
}

int find(int v) {
    if (v == parent[v]) {
        return v;
    }
    return parent[v] = find(parent[v]);
}  // Path Compression

void Union(int a, int b) {
    a = find(a);
    b = find(b);

    if (a != b) { //  // Union by component_size
        if (component_size[a] <
component_size[b]) {
            swap(a, b);
        }
        parent[b] = a;
        component_size[a] +=
component_size[b];
    }
}
int main() {
    int n, q, cc = 0;
    cin >> n >> q;

    for (int i = 1; i <= n; i++) {
        make(i);
    }
    while (q--) {
        int t, u, v;
        cin >> t >> u >> v;

        if(t == 0) {
            Union(u, v);
        }
        else {
            if (find(u) == find(v)) {
                cout << 1 << endl;
            }
            else {
                cout << 0 << endl;
            }
        }
    }

    for (int i = 1; i <= n; i++) {
        if (find(i) == i) {
            cc++;
        }
    }
    cout << cc << endl;

    return 0;
}
```

Prim's Algorithm MST

```
vector<pair<int, int>> graph[1002];
```

```
int vis[1002] = {0};

int MST(int root) {
  priority_queue<pair<int, int>, vector<pair<int,
int>>, greater<pair<int, int>>> pq;
    int sum = 0;
    pq.push({0, root});

    while (!pq.empty()) {
        auto it = pq.top();
        pq.pop();
        int node = it.second;
        int wt = it.first;

        if (vis[node]) {
            continue;
        }
        vis[node] = 1;
        sum += wt;

        for (auto adj : graph[node]) {
            int adjNode = adj.first;
            int edgeWeight = adj.second;

            if (!vis[adjNode]) {
                pq.push({edgeWeight, adjNode});
            }
        }
    }
    return sum;
}

int main() {
    …
    int result = MST(1);
    cout << "Total Cost : " << result << endl;
}
```

Array / Grid Direction

```
int dx[] = { +1, 0, -1, 0, +1, +1, -1, -1};
int dy[] = {0, +1, 0, -1, +1, -1, +1, -1};

int dx[] = { +0, +0, +1, -1, -1, +1, -1, +1};
//King's Move
int dy[] = { -1, +1, +0, +0, +1, +1, -1, -1};
//king's Move

int dx[] = { -2, -2, -1, -1, +1, +1, +2, +2};
//knight'sMove
int dy[] = { -1, +1, -2, +2, -2, +2, -1, +1};
///knight's Move
```

Example Code :

```
const int N = 507;
char graph[N][N];
bool vis [N][N];
int n, m, k;

int dx[] = {0, 0, -1, 1};
int dy[] = {1, -1, 0, 0};

bool valid(int x, int y) {
        return (x>=0 && x<n && y>=0 && y<m
&& !vis[x][y] && graph[x][y]=='.');
}

void dfs(int x, int y) {
        vis[x][y] = true;

        for (int i=0; i<4; i++) {
                int next_x = dx[i] + x;
                int next_y = dy[i] + y;

                if (valid(next_x, next_y)) {
                        dfs(next_x, next_y);
                }
        }
        if (k > 0) {
                graph[x][y] = 'X';
                --k;
        }
}

}

int main() {
    cin >> n >> m >> k;

    for (int i=0; i<n; i++) {
        for (int j=0; j<m; j++) {
                cin >> graph[i][j];
        }
    }

    for (int i=0; i<n; i++) {
        for (int j=0; j<m; j++) {
                if (!vis[i][j] && graph[i][j]=='.') {
                        dfs(i, j);
                }
        }
    }
}
```

Dijkstra

```
#define INF 1000000000
vector<pair<int, int>> graph[1001];
vector<int> dist(1009, INF);

int main() {
    int n, m;
    cin >> n >> m;

    while (m--) {
        int u, v, w;
        cin >> u >> v >> w;
        graph[u].push_back({v, w});
        graph[v].push_back({u, w});
    }
  priority_queue<pair<int, int>, vector<pair<int,
int>>, greater<pair<int, int>>> pq;
    pq.push({0, 1});
    dist[1] = 0;
```

```
    while (!pq.empty()) {
        int node = pq.top().second;
        int curr_d = pq.top().first;
        pq.pop();

        for (auto child : graph[node]) {
            int child_node = child.first;
            int child_wt = child.second;

            if (dist[child_node] > dist[node] +
child_wt) {
                dist[child_node] = dist[node] +
child_wt;
                pq.push({dist[child_node],
child_node});
            }
        }
    }

    for (int i = 1; i <= n; i++) {
        cout << dist[i] << " ";
    }
    cout << endl;

    return 0;
}
```

Floyd warshall

```
const ll INF = 1e18;

int main() {
    ll n, m, q;
    cin >> n >> m >> q;
    ll graph[n+1][n+1];

    for(int i=1;i<=n;i++) {
        for(int j=1;j<=n;j++) {
            graph[i][j] = (i==j)?0:INF;
        }
    }
    for(int i=0; i<m; i++){
        ll v1, v2, w;
        cin >> v1 >> v2 >> w;

        graph[v1][v2] = min (graph[v1][v2], w);
        graph[v2][v1] = min (graph[v2][v1], w);
    }
    for(int k = 1; k<=n; k++) {
        for(int i=1; i<=n; i++) {
            for(int j=1; j<=n; j++) {
                if(graph[i][k] != INF && graph[k][j] !=
INF) {
                    graph[i][j] = min(graph[i][j],
graph[i][k]+graph[k][j]);
                }
            }
        }
    }
    while (q--) {
        int u, v;
        cin >> u >> v;
        if(graph[u][v] >= INF) {
            cout << "-1" << endl;
        }
        else {
            cout << graph[u][v] << endl;
        }
    }
    for(int i=0;i<nodes;i++){
        for(int j=0;j<nodes;j++) {
            cout << graph[i][j] << " ";
        }
        cout << endl;
    }
}
```

Bellman Ford

```
const long long INF = 1e18;
const int MAX_N = 1e5 + 5;

// Global variables
vector<pair<int, int>> adj[MAX_N];
vector<bool> vis(MAX_N, false);
vector<long long> dist(MAX_N, INF);

bool bellmanFord(int n, int src) {
    dist[src] = 0;
    // Relax all edges n-1 times
    for (int i = 1; i < n; i++) {
        for (int u = 1; u <= n; u++) {
            if (dist[u] == INF) continue;
            for (auto edge : adj[u]) {
                int v = edge.first;
                int w = edge.second;
                if (dist[u] + w < dist[v]) {
                    dist[v] = dist[u] + w;
                }
            }
        }
    }
    // Check for negative weight cycles
    for (int u = 1; u <= n; u++) {
        if (dist[u] == INF) continue;
        for (auto edge : adj[u]) {
            int v = edge.first, w = edge.second;
            if (dist[u] + w < dist[v]) {
                return true; // Negative cycle detected
            }
        }
    }
    return false; // No negative cycle
}
int main() {
    cin >> n >> m >> q;

    while(m--) {
        int v1, v2, w;
        cin >> v1 >> v2 >> w;
        adj[v1].push_back({v2, w});
        adj[v2].push_back({v1, w});
    }
    while (q--) {
        int u, v; cin >> u >> v;
```

```
        fill(dist.begin(), dist.end(), INF);
        fill(vis.begin(), vis.end(), false);
        bool neg_cyc = bellmanFord(n, u);

        if (neg_cyc) {
            cout << "contains negative cycle\n";
        } else if (dist[v] == INF) {
            cout << "-1\n";
        } else {
            cout << dist[v] << "\n";
        }
    }
}
```

Example :

1. Three musketeers. Choose three musketeers knowing each other and what is the minimum possible sum of their recognitions.

Code :

```
const int N = 1e4+7;
vector <int> g[N];
int vis[N], D[N], ans = INT_MAX;

void dfs(int node){
  vis[node] = 1;

  for (auto child1: g[node]) {
    if (child1 > node) {
      for (auto child2: g[child1]) {
        auto it = find(g[node].begin(), g[node].end(),
child2);
        if (child2 > child1 && it != g[node].end()) {
          int rec = D[node]+D[child1]+D[child2];
          ans = min(ans, rec);
        }
      }
    }
  }
}

int main() {
    int n, m;
    cin >> n >> m;
    for (int i=1; i<=m; i++) {
        int a, b; cin >> a >> b;
        g[a].push_back(b);
        g[b].push_back(a);
        D[a]++, D[b]++;
    }
    for (int i=1; i<=n; i++) {
        if (!vis[i]) dfs(i);
    }
    if (ans != INT_MAX) {
        cout << ans - 6 << endl;
    } else {
        cout << -1 << endl;
    }
    return 0;
}
```

---

2. A subtree of the tree is called balanced if the number of white vertices equals the number of black vertices. Count the number of balanced subtrees. output a single integer — the number of balanced subtrees.

code :

```
vector<bool> vis;
vector<vector<ll>> graph;
string s;
ll ans;

pair<ll, ll>dfs(ll head) {
    vis[head] = true;
    ll b = 0, w = 0;
    if (s[head - 1] == 'B') b++;
    else w++;

    for (int i=0; i<graph[head].size(); i++) {
        ll child = graph[head][i];
        if (vis[child]) {
            pair<ll, ll>temp = dfs(child);
            b += temp.first;
            w += temp.second;
        }
    }
    if (b == w) ans++;
    return {b, w};
}

void solve() {
    ll n; cin >> n;
    vis.assign(n+1, false);
    graph.assign(n+1, vector<ll>());

    for (int i=2; i<=n; i++) {
        ll x; cin >> x;
        graph[x].push_back(i);
        graph[i].push_back(x);
    }
    cin >> s;
    ans = 0;
    dfs(1);
    cout << ans << endl;
}
```

---

3. Erdos p. Number (BFS)

code :

```
map<string, vector<string>> graph;
map<string, int> dist;
map<string, bool> vis;

void bfs(string start) {
  queue<string> q;
  dist[start] = 0;
  vis[start] = true;
  q.push(start);

  while (!q.empty()) {
    string curr = q.front();
    q.pop();

    for (auto child : graph[curr]) {
```

```
      if (!vis[child]) {
        vis[child] = true;
        dist[child] = dist[curr] + 1;
        q.push(child);
      }
    }
  }
}

  int tc, papers, names;
  string line;
  cin >> tc;

  for (int cs = 1; cs <= tc; cs++) {
    cin >> papers >> names;
    cin.ignore();

  graph.clear();
  dist.clear();
  vis.clear();

  for (int i = 0; i < papers; i++) {
    getline(cin, line);

    int colon_pos = line.find(':');
    if (colon_pos != string::npos) {
      line = line.substr(0, colon_pos);
    }
    int comma_count = 0;

    for (int j = 0; j < line.size(); j++) {
      if (line[j] == ',') {
        comma_count++;
        if (comma_count % 2 == 0) {
          line[j] = ' ';
        }
      }
    }
    vector<string> authors;
    stringstream ss(line);
    string last_name, initials;

    while (ss >> last_name >> initials) {
      authors.push_back(last_name + " " +
initials);
    }

    for (int j = 0; j < authors.size(); j++) {
      for (int k = j + 1; k < authors.size(); k++) {
        graph[authors[j]].push_back(authors[k]);
        graph[authors[k]].push_back(authors[j]);
      }
    }
  }

bfs("Erdos, P.");

cout << "Scenario " << cs << endl;
for (int i = 0; i < names; i++) {
  getline(cin, line);
  if (dist.find(line) == dist.end()) {
    cout << line << " infinity" << endl;
  }
  else {
    cout << line << " " << dist[line] << endl;
  }
}
```

Dynamic Programming

(1) Minimum Number of Coins to Make the Target

```
int n, k; cin >> n >> k;
vector<int> coins(n);
for (int i = 0; i < n; i++) {
    cin >> coins[i];
}
vector<int> dp(k + 1, INT_MAX);
dp[0] = 0;  // 0 coins needed for amount 0

for (int i = 1; i <= k; i++) {
    for (int coin : coins) {
        if (coin <= i && dp[i - coin] != INT_MAX) {
            dp[i] = min(dp[i], 1 + dp[i - coin]);
        }
    }
}
if (dp[n] == INT_MAX) {
    cout << -1 << endl;
} else {
    cout << dp[n] << endl;
}
```

(2) Number of Ways to Make the Target

```
const int MOD = 1e9 + 7;
int n, k;
cin >> n >> k;

vector<int> coins(n);
for (int i = 0; i < n; i++) cin >> coins[i];
vector<int> dp(k + 1, 0);
dp[0] = 1;// 1 way to make amount 0

for (int coin : coins) {
    for (int i = coin; i <= k; i++) {
        dp[i] = (dp[i] + dp[i - coin]) % MOD;
    }
}
cout << dp[k] << endl;
```

(3) Minimum sum path in grid

```
int m, n; cin >> m >> n;
vector<vector<int>> grid(m, vector<int>(n));

for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        cin >> grid[i][j];
    }
}
vector<vector<int>> dp(m, vector<int>(n, 0));
dp[0][0] = grid[0][0];
for (int i = 1; i < m; i++) { // first column
```

```
    dp[i][0] = dp[i-1][0] + grid[i][0];
}
for (int j = 1; j < n; j++) { // first row
    dp[0][j] = dp[0][j-1] + grid[0][j];
}
for (int i = 1; i < m; i++) { // remaining cells
    for (int j = 1; j < n; j++) {
        dp[i][j] = min(dp[i-1][j], dp[i][j-1]) +
grid[i][j];
    }
}
cout << dp[m-1][n-1] << endl;
```

(4) Monkey Banana (Diamond shape)

```
ll arr[400][400];
ll dp[400][400];

ll monkey (ll i, ll j, ll k) {
    if (arr[i][j] == 0) {
        return 0;
    }
    if (i == k-1) {
        return arr[i][j];
    }
    if (dp[i][j] != -1) {
        return dp[i][j];
    }
    ll r1=0, r2=0;

    r1 = arr[i][j] + monkey(i+1, j, k);
    r2 = arr[i][j] + monkey(i+1, j+1, k);

    dp[i][j] = max(r1, r2);

    return dp[i][j];
}

memset(arr, 0, sizeof(arr));
memset(dp, -1, sizeof(dp));

ll n;
cin >> n;
ll k = 2 * n - 1;

for (int i = 0; i < n; i++) {
    for (int j = 0; j <= i; j++) {
        cin >> arr[i][j];
    }
}
ll l = 1;

for (int i = n; i < k; i++) {
    for (int j = l; j < n; j++) {
        cin >> arr[i][j];
    }
    l++;
}
ll ans = monkey(0, 0, k);
printf("Case %d: %d\n", it, ans);
```

(5) Greedy Algo Knapsack :

```
int n, w;
cin >> n >> w;
vector<int> weight(n), profit(n);
vector<double > ratio;

for (int i=0; i<n; i++) {
    cin >> weight[i];
}
for (int i=0; i<n; i++) {
   cin >> profit[i];
}
for (int i=0; i<n; i++) {
  double x = (double)profit[i] / weight[i];
  ratio.push_back(x);
}

for (int i=0; i<n-1; i++) {
    for (int j=i+1; j<n; j++) {
        if (ratio[i] < ratio[j]) {
            swap(ratio[i], ratio[j]);
            swap(profit[i], profit[j]);
            swap(weight[i], weight[j]);
        }
    }
}

int sum=0, total_w=0;
for (int i=0; i<n; i++) {
    if (weight[i] < w) {
        sum += profit[i];
        w -= weight[i];
    }
    else {
       sum += w * ratio[i];
       w = 0;
    }
}
cout << sum << endl;
```

(6) 0/1 Knapsack

```
int n, w;
cin >> n >> w;

vector<ll> wt(n), val(n);

for (int i=0; i<n; i++) {
   cin >> wt[i];
}
for (int i=0; i<n; i++) {
   cin >> val[i];
}
vector<vector<ll>> dp(n+1, vector<ll>(w+1, 0));

for (int i=1; i<n+1; i++) {
   for (int j=0; j<w+1; j++) {
       dp[i][j] = dp[i-1][j];

       if (j-wt[i-1] >= 0) {
           dp[i][j] = max(dp[i][j], dp[i-1][j-wt[i-1]] +
val[i-1]);
```

```
    }
  }
}
cout << dp[n][w] << endl;
```

BIT Masking

---

1. You are given an array arr of 20 integer and another integer S. How many subset with sym equal to S?
code : (Approach: check all possible subset)

```
int n;
cin >> n;
vector<ll> v(n);

for (int i=0; i<n; i++) {
   cin >> v[i];
}
for (int mask=0; mask < (1<<n); mask++) {
   int sum = 0;

   for(int i=0; i<n; i++) {
      if ((mask >> i) && 1) {
         sum += v[i];
      }
   }
   if (sum == s) ans++;
}
cout << ans << endl;
```

Basic Stuff :

Sum of all pair,

1. XOR

```
ll Pair_Xor_Sum (int arr[], int n) {
   ll total_sum = 0;

   for (int bit=0; bit<32; bit++) {
      int cnt_set = 0;
      for (int i=0; i<n; i++) {
         if (arr[i] & (1 << bit)) {
            cnt_set++;
         }
      }
      int cnt_Unset = n - cnt_set;
      total_sum += (1LL << bit) * cnt_set *
cnt_Unset;
   }
   return total_sum;
}
```

---

2. OR

```
ll Pair_OR_Sum (int arr[], int n) {
   ll total_sum = 0;

   for (int bit=0; bit<32; bit++) {
      int cnt_set = 0;
      for (int i=0; i<n; i++) {
         if (arr[i] & (1 << bit)) {
            cnt_set++;
         }
      }
      total_sum += (1LL << bit) * cnt_set * n;
   }
   return total_sum;
}
```

---

3. AND

```
ll Pair_AND_Sum (int arr[], int n) {
   ll total_sum = 0;

   for (int bit=0; bit<32; bit++) {
      int cnt_set = 0;
      for (int i=0; i<n; i++) {
         if (arr[i] & (1 << bit)) {
            cnt_set++;
         }
      }
      total_sum += (1LL << bit) * (cnt_set *
(cnt_set - 1)) / 2;
   }
   return total_sum;
}
```

Sum of all subset,

1. AND

```
ll Subset_AND_Sum (int arr[], int n) {
   ll total_sum = 0;

   for (int bit=0; bit<32; bit++) {
      int cnt_set = 0;
      for (int i=0; i<n; i++) {
         if (arr[i] & (1 << bit)) {
            cnt_set++;
         }
      }
 total_sum += (1LL << bit) * ((1LL << cnt_set) - 1);
   }
   return total_sum;
}
```

---

2. OR

```
ll Subset_OR_Sum (int arr[], int n) {
   ll total_sum = 0;

   for (int bit=0; bit<32; bit++) {
      int subset = 0;
      for (int i=0; i<n; i++) {
         if (bit & (1 << i)) {
            subset |= arr[i];
         }
      }
      total_sum += subset;
   }
   return total_sum;
}
```

---

3. XOR

```
ll Subset_XOR_Sum (int arr[], int n) {
    ll total_sum = 0;

    for (int bit=0; bit<32; bit++) {
        int cnt_set = 0;
        for (int i=0; i<n; i++) {
            if (arr[i] & (1 << bit)) {
                cnt_set++;
            }
        }
    total_sum += (1LL << bit) * cnt_set * (1LL <<
(n-1));
    }
    return total_sum;
}
```

Searching & Sorting

1. Two Sum

```
int n = nums.size();
unordered_map<ll, ll> mp;

for (int i=0; i<n; i++) {
    ll diff = target - nums[i];

    if (mp.find(diff) != mp.end()) {
        return {mp[diff], i};
    }
    mp[nums[i]] = i;
}
```

Another,

```
given array, target
vector<pair<ll, ll>> ans;

for (int i=0; i<v.size(); i++) {
    ans.push_back({v[i], i});
}
sort(ans);
ll l = 0, r = ans.size()-1;
while (l < r) {
    ll curr = ans[l].first + ans[r].first;

    if(curr == target) {
        return {min(ans[l].second, ans[r].second),
                max(ans[l].second, ans[r].second)
                };
    }
    else if (curr > target) {
        r--;
    }
    else {
        l++;
    }
}
```

2. Three Sum

```
int n = nums.size();
sort(nums);
vector<vector<ll>> ans;

for (int i=0; i<n; i++) {
    if (i > 0 && nums[i] == nums[i-1]) {
        continue;
    }
    ll j = i+1, k = n-1;

    while (j < k) {
        ll sum = nums[i] + nums[j] + nums[k];
        if (sum > target) k--;
        else if (sum < target) j++;
        else {
            ans.push_back({nums[i], nums[j],
nums[k]});
            j++;
            k--;
            while(j<k && nums[j] == nums[j-1]) {
                j++;
            }
        }
    }
}
return ans;
```

3. Four Sum

```
int n = nums.size();
vector<vector<int>> ans;
sort(nums.begin(), nums.end());

for (int i = 0; i < n; i++) {
  if (i > 0 && nums[i] == nums[i - 1]) {
    continue;
  }
  for (int j = i + 1; j < n; j++) {
    if (j > i + 1 && nums[j] == nums[j - 1]) {
      continue;
    }
    int p = j + 1, q = n - 1;
    while (p < q) {
      long long sum =
(ll)nums[i]+(ll)nums[j]+(ll)nums[p]+(ll)nums[q];
      if (sum > target) {
        q--;
      } else if (sum < target) {
          p++;
        } else {
            ans.push_back({nums[i], nums[j],
nums[p], nums[q]});
            p++;
            q--;
            while (p < q && nums[p] == nums[p - 1]) {
              p++;
            }
            while (p < q && nums[q] == nums[q + 1]) {
              q--;
            }
        }
    }
  }
}
return ans;
```

4. We need to produce n copies of a document

using two copiers with different speeds. The first copier takes x seconds per copy, while the second takes y seconds. We can use both copiers simultaneously and can make copies from either the original or any existing copies. The challenge is to determine the minimum time required to make all n copies.

```
bool good(ll t) {
    if (t < min(x, y)) {
        return false;
    }
    ll total = 1;
    t -= min(x, y);
    total += floor()t/x) + floor(t/y);
    return total >= n;
}
void solve () {
    cin >> n >> x >> y;
    ll l=0, r = max(x,y)*n;

    while (l+1 < r) {
        ll mid = (l + (r-l)/2);
        if (good(mid)) {
            r = mid;
        }
        else {
            l = mid;
        }
    }
    cout << r << endl;
}
```

---

5. There are n rectangles of the same size: w in width and hin length.
It is required to find a square of the smallest size into which these
rectangles can be packed. Rectangles cannot be rotated.

```
ll w, h, n;

bool good (ll x) {
    return (x / w) * (x / h) >= n;
}
void solve() {
    cin >> w >> h >> n;
    ll l = 0, r = 1;

    while (!good(r)) {
        r *= 2;
    }
    while (r > l + 1) {
        ll mid = (l + r) / 2;
        if (good(mid)) {
            r = mid;
        } else {
            l = mid;
        }
    }
    cout << r << endl;
}
```

---

6. There are n ropes, you need to cut k pieces of the same length from them.
Find the maximum length of pieces you can get.

```
ll n, k;
vector<ll> v;

bool good (double x) {
    int seg = 0;
    for (int i=0; i<n; i++) {
        seg += floor(v[i] / x);
    }
    return seg >= k;
}

void solve() {
    cin >> n >> k;
    v.resize(n);

    for (int i=0; i<n; i++) {
        cin >> v[i];
    }
    double l=0, r = 1e8;

    for (int i=0; i<100; i++) {
        double mid = (l + r) / 2;
        if (good(mid)) {
            l = mid;
        } else {
            r = mid;
        }
    }
    cout << setprecision(20) << l << endl;
}
```

Comparator Sort

```
// second value descending
(1)  bool cmp(pair<int, int> a, pair<int, int> b) {
    return a.second > b.second;
}

// second value sort
(2)  bool cmp(pair<int, int> a, pair<int, int> b) {
    return a.first < b.first;
}

// second value sort
(3)  bool cmp(pair<int, int> a, pair<int, int> b) {
    if (a.first == b.first) {
        return a.second > b.second;
    }
}
```

Algorithms

---

1. KMP Algorithm

```
// O(m+n)
// find how many times pattern exist in text
// the task is to print the indexes of all the occurences of pattern string in the text string.for printing,starting index of a string should be taken as 1.

#include <bits/stdc++.h>
```

```
using namespace std;

void search (string pat, string txt) {
    vector <int> lps(pat.size(), 0);

    for (int i = 1; i < pat.size(); i++) {
        int j = lps[i-1];

        while (j > 0 && pat[j] != pat[i]) {
            j = lps[j-1];
        }

        if (pat[j] == pat[i])j++;

        lps[i] = j;
    }

    int m = pat.size();
    int n = txt.size();
    int i = 0, j = 0, cnt = 0;
    vector <int> ans;

    while (i < n) {
        if (pat[j] == txt[i]) {
            i++;
            j++;
        }
        if (j == m) {
            cnt++;
            ans.push_back(i-(m-1)); // 3 - (-2) = 1
            j = lps[j-1];
        }
        else if (pat[j] != txt[i]) {
            if (j == 0)i++;
            j = lps[j-1];
        }
    }

    cout << "cnt : " << cnt << endl;

    for (auto it : ans) {
        cout << it << " ";
    }
    cout << endl;
}

int main () {
    string txt, pat;
    cin >> txt >> pat;
    search(pat, txt);

    return 0;
}
```

---

2. Function to find the maximum subarray sum using Kadane's Algorithm

```
int maxSubarraySum(vector<int> &arr) {
    int max_curr = arr[0];
    int max_global = arr[0];

    for (int i=1; i<arr.size(); i++) {
        max_curr = max(arr[i], max_curr + arr[i]);

        if (max_curr > max_global) {
            max_global = max_curr;
        }
    }
    return max_global;
}
```
