**Write a code to determine the frequency and uniqueness of n numbers ranging from 0 - 900.**

#include<bits/stdc++.h>

using namespace std;

int main ()

{

int arr [901] = {0};

int n, x;

cin >> n;

for (int i = 0; i < n; i++)

{

cin >> x;

for (int j = 0; j < 901; j++)

{

if (j == x)

{

arr[j]++;

}

}

}

for (int j = 0; j < 901; j++)

{

if (arr[j] != 0)

{

cout << j << " is a unique number which occured " << arr[j] << " times." << endl;

}

}

return 0;

}  
  
  
  
**Write a code to count numbers of consecutive values.**

#include<bits/stdc++.h>

using namespace std;

int main()

{

vector <int> v;

int n;

while (cin >> n)

{

v.push\_back(n);

}

int prev = v[0];

int c = 0;

for (auto & value:v)

{

if (prev == value)

{

c++;

}

else

{

cout << prev << " occured " << c << " times" << endl;

prev = value;

c = 1;

}

}

cout << prev << " occured " << c << " times" <<endl;

return 0;

}  
  
  
  
**Write a code to determine how many values are inputted only once in an array.**

#include <bits/stdc++.h>

using namespace std;

int main()

{

map<int, int> mp;

int x;

while (cin >> x)

{

mp[x]++;

// mp[x] accesses the value associated with the key x in the map.

// If x is not already a key in the map, it will be inserted with a default value of 0.

}

for (auto it : mp)

{

if (it.second == 1)

{

cout << it.first << endl;

}

}

return 0;

}  
  
  
  
**Write a program to check if the string entered by the user is sorted in increasing order or not.**

#include<bits/stdc++.h>

using namespace std;

int main()

{

string s;

cin >> s;

if (is\_sorted(s.begin(), s.end()))

cout << "Yes, it is sorted" << endl;

else

cout << "No, it is not sorted" << endl;

// ASCII values will be counted. So, in case of sorting:

// 1, 2, 3, 4, ...(numbers)... will be smallest

// then comes A, B, C, D, ...(capital letters)...

// then a, b, c, d, ...(small letters)...

return 0;

}  
  
  
  
**Write a code to convert an entire string to uppercase and lowercase.**

#include<bits/stdc++.h>

using namespace std;

int main()

{

string s;

cin >> s;

for (auto &x : s)

{

x = tolower (x);

}

cout << s << endl;

for (auto &x : s)

{

x = toupper (x);

}

cout << s << endl;

return 0;

}  
  
  
**Write a program to check the LCM of two strings.**

#include<bits/stdc++.h>

using namespace std;

int main()

{

string a, b;

cin >> a >> b;

string c = a;

string d = b;

int x = a.size();

int y = b.size();

int k = (x \* y) / \_\_gcd(x, y);

while (a.size() < k)

{

a = a + c;

}

while (b.size() < k)

{

b = b + d;

}

if (b.compare(a) == 0)

{

cout << a << endl;

}

else

{

cout << -1 << endl;

}

return 0;

}  
  
  
  
**Write a code to check if a number is prime.**

#include<bits/stdc++.h>

using namespace std;

sievebool prime (int n)

{

if (n < 2)

return false;

for (int i = 2; i <= sqrt (n); i++)

{

if (n % i == 0)

return false;

}

return true;

}

int main ()

{

int x;

while (cin >> x)

{

if (prime (x) == true)

cout << "Prime" << endl;

else

cout << "Not Prime" << endl;

}

return 0;

}

**Sieve of Eratosthenes. Write a code to determine the amount of prime numbers between 1 to n.**

#include <bits/stdc++.h>

using namespace std;

void sieve(int n)

{

vector <int> prime(n + 1, 0);

// Initialize vector with size n+1 and all elements set to 0.

// We take size upto n + 1 so that prime[n] is counted.

for (int i = 2; i <= sqrt(n); i++)

{

if (prime[i] == 0)

{

for (int j = i \* 2; j <= n; j += i)

{

prime[j] = 1;

}

}

}

for (int i = 2; i <= n; i++)

{

if (prime[i] == 0)

cout << i << " ";

}

cout << endl;

}

int main()

{

while (1)

{

int n;

cin >> n;

sieve(n);

}

return 0;

}  
  
  
  
**Write a code to determine the prime factorization, number of unique prime and their count of a number n.**

#include <bits/stdc++.h>

using namespace std;

void p\_factorial(int n)

{

set <int> unique\_primes; // Set to store unique prime factors

for (int i = 2; i <= sqrt(n); i++)

{

if (n % i == 0)

{

int c = 0;

while (n % i == 0)

{

c++;

n = n / i;

}

cout << i << " ^ " << c << " ,";

unique\_primes.insert(i); // Insert prime factor into set

}

}

if (n > 1) // If n is a prime number greater than sqrt(n)

{

cout << n << " ^ 1";

unique\_primes.insert(n); // Insert the remaining prime factor

}

cout << endl;

// Output the number of unique prime factors

cout << "Number of unique prime factors: " << unique\_primes.size() << endl;

// Print the unique prime factors

cout << "Unique prime factors: ";

for (auto prime : unique\_primes)

{

cout << prime << " ";

}

cout << endl;

}

int main()

{

int t;

cin >> t;

while (t--)

{

int n;

cin >> n;

p\_factorial(n);

}

return 0;

}  
  
  
  
**Binary Exponentiation. Finding the value of base ^ power.**

#include <bits/stdc++.h>

using namespace std;

int power(int b, int p, int mod)

{

int result = 1;

while (p)

{

if (p % 2 == 1)

{

result = (result \* b) % mod;

p--;

}

else

{

b = (b \* b) % mod;

p /= 2;

}

}

return result;

}

int main()

{

int t;

cin >> t;

while (t--)

{

int b, p; // b -> base, p -> power

cin >> b >> p;

cout << power (b, p, 1e9 + 7) << endl;

}

return 0;

}

**Matrix Exponentiation. Matrix, M ^ n.**

#include <bits/stdc++.h>

using namespace std;

const int mod = 1e9 + 7;

void mul(vector <vector <int>> & a, vector <vector <int>> & b, int n)

{

vector <vector <int>> R(n, vector <int> (n, 0));

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

for (int k = 0; k < n; k++)

{

int x = (a[i][k] \* b[k][j]) % mod;

R[i][j] = (R[i][j] + x) % mod;

}

}

}

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

a[i][j] = R[i][j];

}

}

}

void power(vector <vector <int>> & a, int n, int p)

{

vector <vector <int>> I(n, vector <int> (n, 0));

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

if (i == j)

I[i][j] = 1;

else

I[i][j] = 0;

}

}

while (p)

{

if (p % 2 == 1)

{

mul(I, a, n);

p--;

}

else

{

mul(a, a, n);

p /= 2;

}

}

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

a[i][j] = I[i][j];

}

}

}

int main()

{

int t;

cin >> t;

while (t--)

{

int n, p;

cin >> n >> p; // n -> matrix size (matrix with size n x n), p -> power

vector<vector<int>> a(n, vector<int>(n));

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

cin >> a[i][j];

}

}

power(a, n, p);

for (int i = 0; i < n; i++)

{

for (int j = 0; j < n; j++)

{

cout << a[i][j] << " ";

}

cout << endl;

}

}

return 0;

}

**Printing, counting and determining the sum of the number of divisors**

#include <bits/stdc++.h>

using namespace std;

const int mod = 1e9 + 7;

set<int> s;

void count\_divisors(int n, int &sum)

{

sum = 0;

for (int i = 1; i <= sqrt(n); i++)

{

if (n % i == 0)

{

int x = i, y = n / i;

s.insert(x);

s.insert(y);

sum += x;

if (x != y)

{

sum += y;

}

}

}

}

int main()

{

int t;

cin >> t;

while (t--)

{

int n;

cin >> n;

s.clear(); // Clear the set at the beginning of each iteration

int sum;

count\_divisors(n, sum);

cout << s.size() << endl; // Total number of divisors

for (auto it : s)

{

cout << it << " ";

}

cout << endl;

cout << "Sum of divisors: " << sum << endl;

}

return 0;

}

**Fibonacci finding using Matrix Exponentiation.**

#include <bits/stdc++.h>

using namespace std;

const int mod = 1e9 + 7;

// Function to multiply two matrices

void mul(int a[2][2], int b[2][2])

{

int res[2][2] = {};

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

for (int k = 0; k < 2; k++)

{

res[i][j] = (res[i][j] + 1LL \* a[i][k] \* b[k][j]) % mod;

}

}

}

for (int i = 0; i < 2; i++)

{

for (int j = 0; j < 2; j++)

{

a[i][j] = res[i][j];

}

}

}

// Function to find the nth Fibonacci number using matrix exponentiation

void solve(int a, int b, int n)

{

int I[2][2] = { {1, 0}, {0, 1} }; // Identity matrix

int T[2][2] = { {1, 1}, {1, 0} }; // Transformation matrix

// Perform matrix exponentiation

while (n)

{

if (n % 2)

{

mul(I, T);

n--;

}

else

{

mul(T, T);

n /= 2;

}

}

// Result is in I[0][0] \* b + I[0][1] \* a

int ans = (1LL \* I[0][0] \* b + 1LL \* I[0][1] \* a) % mod;

cout << ans << endl;

}

int main()

{

int t;

cin >> t;

while (t--)

{

int a, b, n;

cin >> a >> b >> n; to print array[n] as the nth fibonacci

if (n == 1) cout << a % mod << endl; // F(1) is a (n == 0)

else if (n == 2) cout << b % mod << endl; // F(2) is b (n == 1)

else solve(a, b, n-2); // For F(n), compute solve(a, b, n-2) (a, b, n – 1)

}

return 0;

}

**Euclid's Algorithm (GCD). Finding GCD of a number n.**

#include <bits/stdc++.h>

using namespace std;

int gcd (int a, int b)

{

if (b == 0)

{

return a;

}

return gcd (b, a % b);

}

int main()

{

int t;

cin >> t;

while (t--)

{

int a, b;

cin >> a >> b;

int ans = gcd (a, b);

cout << ans << endl;

}

return 0;

}

**Segmented Sieve. Write a program to find and count the prime numbers from L to R.**

#include <bits/stdc++.h>

using namespace std;

vector<int> v; // Vector to hold primes up to sqrt(R)

void sieve(int n)

{

vector<int> prime(n + 1, 0);

for (int i = 2; i <= sqrt(n); i++)

{

if (prime[i] == 0)

{

for (int j = i \* i; j <= n; j += i)

{

prime[j] = 1;

}

}

}

for (int i = 2; i <= n; i++)

{

if (prime[i] == 0)

{

v.push\_back(i);

}

}

}

void init(int l, int r)

{

if (l == 1)

{

l++;

}

int mx = r - l + 1;

vector<int> ar(mx, 0); // Vector for marking non-primes in range [l, r]

for (int p : v)

{

if (p \* p <= r)

{

int i = (l / p) \* p;

if (i < l)

{

i += p;

}

for (; i <= r; i += p)

{

if (i != p)

{

ar[i - l] = 1;

}

}

}

}

vector<int> primes\_in\_range;

for (int i = 0; i < mx; i++)

{

if (ar[i] == 0)

{

primes\_in\_range.push\_back(l + i);

}

}

cout << "Prime numbers: ";

for (int prime : primes\_in\_range)

{

cout << prime << " ";

}

cout << "\nPrime count: " << primes\_in\_range.size() << endl;

}

int main()

{

sieve(100000);

int t, L, R;

cin >> t;

while (t--)

{

cin >> L >> R;

init(L, R);

}

return 0;

}

**Write a code to determine if a number can be perfectly divided by a number m.**

#include <bits/stdc++.h>

using namespace std;

void mod (string s, int m)

{

int ans = 0;

for (int i = 0; i < s.size(); i++)

{

ans = ans \* 10 + (s[i] - '0');

ans = ans % m;

}

if (ans % m == 0)

cout << "Yes" << endl;

else

cout << "No" << endl;

}

int main()

{

int t;

cin >> t;

while (t--)

{

string s;

cin >> s;

int m;

cin >> m;

mod (s, m);

}

return 0;

}

**Inverse Modulo.**

#include <bits/stdc++.h>

using namespace std;

int power (int base, int p, int m)

{

int res = 1;

while (p)

{

if (p % 2 == 1)

{

res = (res \* base) % m;

p--;

}

else

{

base = (base \* base) % m;

p /= 2;

}

}

return res % m;

}

int main()

{

int t;

cin >> t;

while (t--)

{

int a, b, m;

cin >> a >> b >> m;

int x = power (b, m - 2, m);

int ans = (a \* x) % m;

cout << ans << endl;

}

return 0;

}

**Binomial Co-efficient using Modular Arithmetic.**

#include <bits/stdc++.h>

using namespace std;

const int mod = 1e9 + 7; // Define the modulus value for modular arithmetic

// Function to compute factorial under modulo

long long fact(long long n)

{

long long ans = 1;

for (long long i = 2; i <= n; i++) // Calculate factorial from 2 to n

ans = (ans \* i) % mod; // Keep the result within mod to prevent overflow

return ans; // Return the factorial modulo mod

}

// Function to compute base^p under modulo using binary exponentiation

long long power(long long base, long long p)

{

long long ans = 1;

while (p)

{

if (p % 2 == 1) // If p is odd, multiply the current base with the answer

ans = (ans \* base) % mod;

base = (base \* base) % mod; // Square the base for the next iteration

p /= 2; // Divide p by 2 (right shift in binary)

}

return ans; // Return the result of base^p % mod

}

// Function to compute nCr % mod using factorial and modular inverse

long long nCr(long long n, long long r)

{

if (r > n) // If r is greater than n, nCr is zero

return 0;

long long nFact = fact(n); // Compute n!

long long rFactInverse = power(fact(r), mod - 2); // Compute (r!)^-1 % mod

long long nMinusRFactInverse = power(fact(n - r), mod - 2); // Compute ((n-r)!)^-1 % mod

// Calculate nCr % mod using the formula: nCr = n! / (r! \* (n-r)!)

return nFact \* rFactInverse % mod \* nMinusRFactInverse % mod;

}

int main()

{

int t;

cin >> t; // Read the number of test cases

while (t--)

{

long long n, r;

cin >> n >> r; // Read n and r for each test case

cout << nCr(n, r) << endl; // Output the result of nCr % mod

}

return 0;

}

**Euler's Totient Function. Write a code to count how many times gcd(i, n) == 1 from 1 to number n.**

#include<bits/stdc++.h>

using namespace std;

void phi (int n)

{

int ans = n;

for (int i = 2; i \* i <=n; i++)

{

if (n % i == 0)

{

while (n % i == 0)

{

n /= i;

}

ans \*= (i - 1);

ans /= i;

}

}

if (n > 1)

{

ans \*= (n - 1);

ans /= n;

}

cout << ans << endl;

}

int main ()

{

int t;

cin >> t;

while (t--)

{

int n;

cin >> n;

phi (n);

}

return 0;

}

**Goldbach's Conjecture. Express an even number greater than 2 as the sum of 2 prime numbers.**

#include <bits/stdc++.h>

#include <cmath>

using namespace std;

// Function to check if a number is prime

bool isPrime(int num)

{

if (num <= 1) return false;

if (num <= 3) return true;

if (num % 2 == 0 || num % 3 == 0) return false;

for (int i = 5; i <= sqrt(num); i += 6)

{

if (num % i == 0 || num % (i + 2) == 0) return false;

}

return true;

}

// Function to find two prime numbers that sum up to the given even number

void findGoldbachPair(int num)

{

if (num <= 2 || num % 2 != 0)

{

cout << "Please enter an even number greater than 2." << endl;

return;

}

for (int i = 2; i <= num / 2; i++)

{

if (isPrime(i) && isPrime(num - i))

{

cout << num << " = " << i << " + " << (num - i) << endl;

return;

}

}

// If no pair is found (though this should not happen as per the conjecture)

cout << "No prime pair found." << endl;

}

int main()

{

int num;

cout << "Enter an even number greater than 2: ";

cin >> num;

findGoldbachPair (num);

return 0;

}

**Extended Euclidean Algorithm to solve ax + by = gcd (a, b).**

#include <bits/stdc++.h>

using namespace std;

// Function to implement the Extended Euclidean Algorithm

int gcdExtended(int a, int b, int &x, int &y)

{

if (a == 0)

{

x = 0;

y = 1;

return b;

}

int x1, y1; // To store results of recursive call

int gcd = gcdExtended(b % a, a, x1, y1);

// Update x and y using results of recursive call

x = y1 - (b / a) \* x1;

y = x1;

return gcd;

}

// Driver code

int main()

{

int a, b, x, y;

cout << "Enter two integers a and b: ";

cin >> a >> b;

int gcd = gcdExtended(a, b, x, y);

cout << "gcd(" << a << ", " << b << ") = " << gcd << endl;

cout << "Coefficients x and y are: " << x << " and " << y << endl;

return 0;

}

**Linear Diophantine Equation to solve ax+by=c.**

#include <bits/stdc++.h>

using namespace std;

// Function to implement the Extended Euclidean Algorithm

int gcdExtended(int a, int b, int &x, int &y)

{

if (a == 0)

{

x = 0;

y = 1;

return b;

}

int x1, y1;

int gcd = gcdExtended(b % a, a, x1, y1);

x = y1 - (b / a) \* x1;

y = x1;

return gcd;

}

// Function to find a particular solution to ax + by = c

bool findSolution(int a, int b, int c, int &x0, int &y0, int &g)

{

g = gcdExtended(abs(a), abs(b), x0, y0);

if (c % g != 0)

{

return false; // No solution

}

x0 \*= c / g;

y0 \*= c / g;

if (a < 0) x0 = -x0;

if (b < 0) y0 = -y0;

return true;

}

int main()

{

int a, b, c;

cout << "Enter coefficients a, b and c (ax + by = c): ";

cin >> a >> b >> c;

int x0, y0, g;

if (findSolution(a, b, c, x0, y0, g))

{

cout << "x = " << x0 << ", y = " << y0 << endl;

}

else

{

cout << "No solution exists." << endl;

}

return 0;

}

**Chinese Remainder Theorem.**

#include <bits/stdc++.h>

using namespace std;

// Function to compute the modular inverse of a number a under modulo m

int modInverse(int a, int m)

{

int m0 = m;

int y = 0;

int x = 1;

if (m == 1)

{

return 0;

}

while (a > 1)

{

// q is the quotient

int q = a / m;

int t = m;

// m is the remainder now, process same as Euclid's algo

m = a % m;

a = t;

t = y;

// Update y and x

y = x - q \* y;

x = t;

}

// Make x positive

if (x < 0)

{

x += m0;

}

return x;

}

// Function to find the solution to the system of congruences

int chineseRemainderTheorem(const vector<int>& a, const vector<int>& m)

{

int k = a.size();

int M = 1;

// Compute the product of all moduli

for (int i = 0; i < k; i++)

{

M \*= m[i];

}

int result = 0;

// Solve the system of congruences

for (int i = 0; i < k; i++)

{

int Mi = M / m[i];

int yi = modInverse(Mi, m[i]);

result = (result + a[i] \* Mi \* yi) % M;

}

// Ensure the result is positive

if (result < 0)

{

result += M;

}

return result;

}

int main()

{

// Number of congruences

int k;

cout << "Enter the number of congruences: ";

cin >> k;

vector<int> a(k), m(k);

// Input the congruences

cout << "Enter the remainders (a\_i) and moduli (m\_i):" << endl;

for (int i = 0; i < k; i++)

{

cin >> a[i] >> m[i];

}

// Compute the result using CRT

int result = chineseRemainderTheorem(a, m);

// Output the result

cout << "The solution is: " << result << endl;

return 0;

}

**Binary Search.**

#include <bits/stdc++.h>

using namespace std;

void binary\_search(vector<int>& v, int n, int x)

{

sort(v.begin(), v.end()); // sort the vector

int left = 0;

int right = n - 1;

while (left <= right)

{

int mid = left + (right - left) / 2; // avoid overflow

if (v[mid] == x)

{

cout << "Found" << endl;

cout << "Index of x is: " << mid << endl;

return;

}

else if (v[mid] > x)

{

right = mid - 1;

}

else

{

left = mid + 1;

}

}

cout << "X is not found" << endl;

}

int main()

{

int n;

cin >> n;

vector<int> v(n);

for (int i = 0; i < n; i++)

{

cin >> v[i];

}

int x; // element to be searched.

cin >> x;

binary\_search(v, n, x);

return 0;

}

**First Occurrence, Last Occurrence and count of an element.**

#include <bits/stdc++.h>

using namespace std;

int FO(vector<int>& v, int n, int x)

{

int ans = -1;

int left = 0;

int right = n - 1;

while (left <= right)

{

int mid = left + (right - left) / 2; // avoid overflow

if (v[mid] == x)

{

ans = mid;

right = mid - 1; // continue to search in the left half

}

else if (v[mid] > x)

{

right = mid - 1;

}

else

{

left = mid + 1;

}

}

return ans;

}

int LO(vector<int>& v, int n, int x)

{

int ans = -1;

int left = 0;

int right = n - 1;

while (left <= right)

{

int mid = left + (right - left) / 2; // avoid overflow

if (v[mid] == x)

{

ans = mid;

left = mid + 1; // continue to search in the right half

}

else if (v[mid] < x)

{

left = mid + 1;

}

else

{

right = mid - 1;

}

}

return ans;

}

int main()

{

int n;

cin >> n;

vector<int> v(n);

for (int i = 0; i < n; i++)

{

cin >> v[i];

}

int x; // element to be searched.

cin >> x;

// Sort the vector before searching

sort(v.begin(), v.end());

int first\_occurrence = FO(v, n, x);

int last\_occurrence = LO(v, n, x);

if (first\_occurrence != -1)

{

cout << "First Occurrence: " << first\_occurrence << endl; // first\_occurrence + 1 to output position number

cout << "Last Occurrence: " << last\_occurrence << endl; // last\_occurrence + 1 to output position number

cout << "Count of " << x << " is " << (last\_occurrence - first\_occurrence + 1) << endl;

}

else

{

cout << "X is not found" << endl;

}

return 0;

}

**Number of times an array or a vector is rotated to sort it.**

#include <bits/stdc++.h>

using namespace std;

int findRotationCount(vector<int>& v)

{

int left = 0;

int right = v.size() - 1;

while (left <= right)

{

int mid = left + (right - left) / 2;

// Check if the mid element is the minimum element

if (mid > 0 && v[mid] < v[mid - 1])

{

return mid;

}

else if (v[mid] >= v[left])

{

// The minimum element is in the right part

left = mid + 1;

}

else

{

// The minimum element is in the left part

right = mid - 1;

}

}

return 0; // If the array is not rotated

}

int main()

{

int n;

cin >> n;

vector<int> v(n);

for (int i = 0; i < n; i++)

{

cin >> v[i];

}

int rotation\_count = findRotationCount(v);

cout << "Number of times the array is rotated: " << rotation\_count << endl;

return 0;

}

**Find Floor of an Element in a Vector.**

#include <bits/stdc++.h>

using namespace std;

// Function to find the floor of a given element in a sorted vector

int findFloor(vector<int>& v, int n, int x)

{

int left = 0;

int right = n - 1;

int floorIndex = -1; // Initialize floor index to -1 (indicating no floor found)

while (left <= right)

{

int mid = left + (right - left) / 2; // Calculate mid to avoid overflow

// If the element at mid is less than or equal to x, update floorIndex

if (v[mid] <= x)

{

floorIndex = mid;

left = mid + 1; // Search the right half to find a greater floor

}

else

{

right = mid - 1; // Search the left half

}

}

return floorIndex; // Return the index of the floor element

}

int main()

{

int n;

cin >> n; // Read the number of elements in the vector

vector<int> v(n); // Declare a vector of size n

for (int i = 0; i < n; i++)

{

cin >> v[i]; // Read elements into the vector

}

int x; // Element to find the floor of

cin >> x;

// Sort the vector to ensure it works with unsorted inputs

sort(v.begin(), v.end());

// Find the floor of x in the sorted vector

int floorIndex = findFloor(v, n, x);

// Check if a floor was found and output the result

if (floorIndex != -1)

{

cout << "Floor of " << x << " is: " << v[floorIndex] << endl;

}

else

{

cout << "No floor found for " << x << endl;

}

return 0;

}

**Find Ceiling of an Element in a Vector.**

#include <bits/stdc++.h>

using namespace std;

// Function to find the ceiling of a given element in a sorted vector

int findCeil(vector<int>& v, int n, int x)

{

int left = 0;

int right = n - 1;

int ceilIndex = -1; // Initialize ceiling index to -1 (indicating no ceiling found)

while (left <= right)

{

int mid = left + (right - left) / 2; // Calculate mid to avoid overflow

// If the element at mid is greater than or equal to x, update ceilIndex

if (v[mid] >= x)

{

ceilIndex = mid;

right = mid - 1; // Search the left half to find a smaller ceiling

}

else

{

left = mid + 1; // Search the right half

}

}

return ceilIndex; // Return the index of the ceiling element

}

int main()

{

int n;

cin >> n; // Read the number of elements in the vector

vector<int> v(n); // Declare a vector of size n

for (int i = 0; i < n; i++)

{

cin >> v[i]; // Read elements into the vector

}

int x; // Element to find the ceiling of

cin >> x;

// Sort the vector to ensure it works with unsorted inputs

sort(v.begin(), v.end());

// Find the ceiling of x in the sorted vector

int ceilIndex = findCeil(v, n, x);

// Check if a ceiling was found and output the result

if (ceilIndex != -1)

{

cout << "Ceiling of " << x << " is: " << v[ceilIndex] << endl;

}

else

{

cout << "No ceiling found for " << x << endl;

}

return 0;

}

**Find the next alphabetical element in a string.**

#include <bits/stdc++.h>

using namespace std;

char solve(string s, char ch, int n)

{

// Remove spaces from the string

s.erase(remove(s.begin(), s.end(), ' '), s.end());

// Sort the string to ensure it works with unsorted inputs

sort(s.begin(), s.end());

int left = 0;

int right = s.size() - 1; // Update n after removing spaces

char ans = '\0'; // Initialize with a null character

while (left <= right)

{

int mid = (left + right) / 2;

if (s[mid] > ch)

{

ans = s[mid];

right = mid - 1;

}

else

{

left = mid + 1;

}

}

return ans;

}

int main()

{

string s;

getline(cin, s); // Read the entire line to include spaces

char ch;

cin >> ch;

char ans = solve(s, ch, s.size());

if (ans != '\0')

{

cout << ans << endl;

}

else

{

cout << "No character found greater than " << ch << endl;

}

return 0;

}

**Find the pair whose sum is equal to x.**

#include <bits/stdc++.h>

using namespace std;

vector <pair <int, int>> values;

bool solve(vector <int> &v, int x, int n)

{

int i = 0;

int j = n - 1;

bool found = false;

while (i < j)

{

int sum = v[i] + v[j];

if (sum == x)

{

values.push\_back({v[i], v[j]});

found = true;

i++;

j--;

}

else if (sum > x)

{

j--;

}

else

{

i++;

}

}

return found;

}

int main()

{

int n;

cin >> n;

vector<int> v(n);

for (int i = 0; i < n; i++)

{

cin >> v[i];

}

sort(v.begin(), v.end());

int x;

cin >> x;

bool ans = solve(v, x, n);

if (ans)

{

cout << "Yes" << endl;

for (const auto &p : values)

{

cout << x << " = " << p.first << " + " << p.second << endl;

}

}

else

{

cout << "No" << endl;

}

return 0;

}

**Find the pair whose sum is closest to x.**

#include <bits/stdc++.h>

using namespace std;

void solve(vector <int> &v, int x, int n)

{

int left = 0;

int right = n - 1;

int index1, index2;

int diff = INT\_MAX;

while (left < right)

{

int sum = v[left] + v[right];

if (abs (sum - x) < diff)

{

index1 = left;

index2 = right;

diff = abs (sum - x);

}

if (sum > x)

right--;

else

left++;

}

cout << v[index1] << " and " << v[index2] << " -> " << v[index1] + v[index2] << endl;

}

int main()

{

int n;

cin >> n;

vector<int> v(n);

for (int i = 0; i < n; i++)

{

cin >> v[i];

}

sort(v.begin(), v.end());

int x;

cin >> x;

solve(v, x, n);

return 0;

}

**Find the closest pair from two vectors.**

#include <bits/stdc++.h>

using namespace std;

void solve(vector <int> &v1, vector <int> &v2, int x, int n, int m)

{

int left = 0;

int right = m - 1;

int index1, index2;

int diff = INT\_MAX;

while (left < n && right >= 0)

{

int sum = v1[left] + v2[right];

if (abs (sum - x) < diff)

{

index1 = left;

index2 = right;

diff = abs (sum - x);

}

if (sum > x)

right--;

else

left++;

}

cout << v1[index1] << " and " << v2[index2] << " -> " << v1[index1] + v2[index2] << endl;

}

int main()

{

int n;

cin >> n;

vector<int> v1(n);

for (int i = 0; i < n; i++)

{

cin >> v1[i];

}

int m;

cin >> m;

vector<int> v2(m);

for (int i = 0; i < m; i++)

{

cin >> v2[i];

}

sort(v1.begin(), v1.end());

sort(v2.begin(), v2.end());

int x;

cin >> x;

solve(v1, v2, x, n, m);

return 0;

}

**Find all triplets with sum equal to x. (Using Brute Forces Method)**

#include <bits/stdc++.h>

using namespace std;

void solve(vector <int> &v, int x, int n)

{

for (int i = 0; i < n; i++)

{

for (int j = i + 1; j < n; j++)

{

for (int k = j + 1; k < n; k++)

{

if (v[i] + v[j] + v[k] == x)

{

cout << v[i] << " + " << v[j] << " + " << v[k] << " = " << x << endl;

}

}

}

}

}

int main()

{

int n;

cin >> n;

vector<int> v(n);

for (int i = 0; i < n; i++)

{

cin >> v[i];

}

sort(v.begin(), v.end());

int x;

cin >> x;

solve(v, x, n);

return 0;

}

**Find all triplets with sum equal to x.**

#include <bits/stdc++.h>

using namespace std;

void solve(vector<int> &v, int x, int n)

{

for (int i = 0; i < n - 2; i++)

{

int target = x - v[i];

int left = i + 1;

int right = n - 1;

while (left < right)

{

int sum = v[left] + v[right];

if (sum == target)

{

cout << v[i] << " + " << v[left] << " + " << v[right] << " = " << x << endl;

left++;

right--;

}

else if (sum < target)

{

left++;

}

else

{

right--;

}

}

}

}

int main()

{

int n;

cin >> n;

vector<int> v(n);

for (int i = 0; i < n; i++)

{

cin >> v[i];

}

sort(v.begin(), v.end());

int x;

cin >> x;

solve (v, x, n);

return 0;

}

**Find four elements that sum to a given value.**

#include <bits/stdc++.h>

using namespace std;

void solve(vector<int> &v, int x, int n)

{

for (int i = 0; i < n - 3; i++)

{

for (int j = i + 1; j < n - 2; j++)

{

int target = x - v[i] - v[j];

int left = j + 1;

int right = n - 1;

while (left < right)

{

int sum = v[left] + v[right];

if (sum == target)

{

cout << v[i] << " + " << v[j] << " + " << v[left] << " + " << v[right] << " = " << x << endl;

left++;

right--;

}

else if (sum < target)

{

left++;

}

else

{

right--;

}

}

}

}

}

int main()

{

int n;

cin >> n;

vector<int> v(n);

for (int i = 0; i < n; i++)

{

cin >> v[i];

}

sort(v.begin(), v.end());

int x;

cin >> x;

solve (v, x, n);

return 0;

}

**Ackermann Function**

#include<bits/stdc++.h>

using namespace std;

#define ll long long

#define lli long long int

#define pk() ios\_base::sync\_with\_stdio(0); cin.tie(0)

const int mod = 1e9 + 7;

int acm (int m, int n)

{

if (m == 0)

{

return n + 1;

}

else if (m > 0 && n == 0)

{

return acm (m - 1, 1);

}

else if (m > 0 && n > 0)

{

return acm(m - 1, acm(m, n - 1));

}

else

{

return -1;

}

}

int main ()

{

pk();

int m, n;

cin >> m >> n;

cout << acm(m, n) << "\n";

return 0;

}

**Pizza cutting for maximum slices.**

#include<bits/stdc++.h>

using namespace std;

#define ll long long

#define lli long long int

#define pk() ios\_base::sync\_with\_stdio(0); cin.tie(0)

const int mod = 1e9 + 7;

int main ()

{

pk ();

ll n;

while (cin >> n)

{

if (n < 0)

break; //return 0;

else

{

ll a = (n \* (n + 1))/2 + 1;

cout << a << endl;

}

}

return 0;

}