# Template

// -0.0+0.0=-0 insted of -0.0+0.0=-0

#pragma GCC optimization("Ofast")

// optimize loops which iterations can be determined at compile time

#pragma GCC optimization("unroll-loops")  
#include<bits/stdc++.h>

using namespace std;

#define endl '\n'

#define pb push\_back

#define F first

#define S second

using ull = unsigned long long;

using ll = long long;

inline void fun(){}//fast function

int main(){

ios\_base::sync\_with\_stdio(0);cin.tie(0);cout.tie(0);//IO speed

int tc=1;

//freopen("inputs.txt", "r", stdin);

//cin>>tc;

int n;

while(tc--){

cin>>n;

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

}

}

return 0;

}

# Which data or container?

**(-10^9 to +10^9):**

**int:** -2,147,483,648 **to** 2,147,483,647

**unsigned int:** 0 **to** 4,294,967,295

**(-10^18 to +10^18):**

**long long:** -9,223,372,036,854,775,808 **to** 9,223,372,036,854,775,807

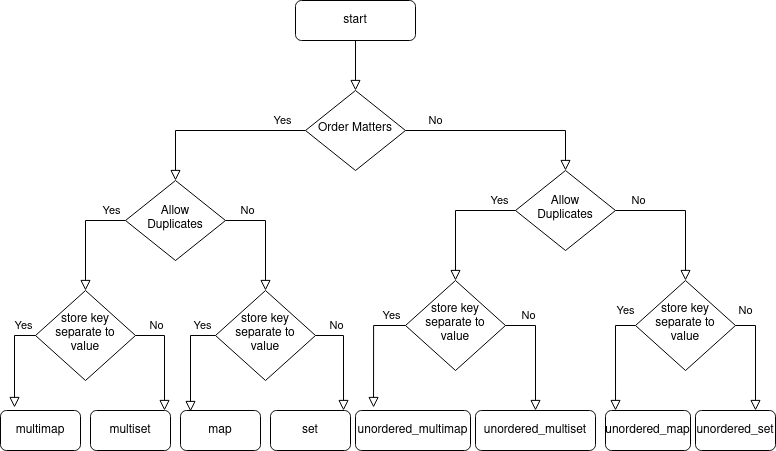
**unsigned long long:** 0 **to** 18,446,744,073,709,551,615

**(7 digit precision):**

**float:**  1,175494351 E - 38 **to** 3,402823466 E + 38

**(15 digit precision):**

**double:** 2,2250738585072014 E - 308 **to** 1,7976931348623158 E + 308



# Misc Math Functions

min({a, b, c, d}); max({a, b, c, d});

bool flag; cout << (flag ? "YES" : "NO");//ternary op

numeric\_limits<unsigned int>::max(); numeric\_limits<float>::min();

cout << fixed << setprecision(digits)<< var;// digits == 0 ? (round)

round(num); trunc(num); ceil(num); floor(num); abs(num);

pow(base, exp); powl(base,exp);

sqrt(num); sqrtl(num);

pow(p, 1.0 / n);// nth root of p

var<<exp;//var\*(2^exp)

var>>exp;//var/(2^exp)

#### first 25 primes

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

# Algorithms

int gcd(int a, int b){ //Greatest common divisor < stdc 17

if (b==0) return a;

else return gcd(b, a%b);

}

int lcm(int a, int b){//Least common multiple < stdc 17

return a\*b/gcd(a,b);

}

bool isPrime(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;

}

int factor(int n) {

int a;

if (n % 2 == 0)

return 2;

for (a = 3; a <= sqrt(n); a++) {

if (n % a == 0)

return a;

}

return n;

}

void fullFactorization(int n) {

int r;

while (n > 1) {

r = factor(n);

std::cout << r << " ";

n /= r;

}

}

const double sqrt5 = sqrt(5);// global constants for fibonnacci

const double phi = (1 + sqrt5) / 2;

const double psi = (1 - sqrt5) / 2;

int fibonacci(int n) {//O(1) of the first 46 fib numbers

return (int)round( (pow(phi, n) - pow(psi, n)) / (sqrt5) );

}

# Circular array

int dist1 = abs(idx1-idx2);

int dist2 = n - dist1;

for (int i = idx1; i != idx2; i = (i - 1 + n) % n) {//Left

}

for (int i = idx1; i != idx2; i = (i + 1 + n) % n) {//Right

}

# Sorting

sort(vec.begin(), vec.end());//ascending

sort(vec.begin(), vec.end(), greater<int>());//non ascending

inline bool myOrder(pair<int, int> p1, pair<int, int> p2) {

return p1.first < p2.first;

}

vector<pair<int, int>> vec = {{3, 1}, {2, 5}, {1, 4}};

sort(vec.begin(), vec.end(), myOrder);

for(auto elem : vec)

cout << "(" << elem.first << ", " << elem.second << ") ";

# Standard Template Library (STL)

## Iterators

for(vector<int>::iterator it=stl.begin(); it!=stl.end(); it++)

cout<<\*it<<" ";//(it) is memory pointer (\*it) its value

for (const auto& pair : mapVar)

cout << pair.first << ": " << pair.second;

## Capacity: stl.size(); //lenght of STL container

stl.empty(); // boolean function

## Modifiers:

stl.clear();//clears the container but complexity O(n) better use swap(v,vv[i])

stl.erase(it);stl.erase(itBegin,itEnd);// removes elements from range

stl.resize(size);stl.resize(size,initializer);stl.insert(it,elem)

stl.emplace\_back(elem)//faster push\_back (avoid copies), inmediate values

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| .fun() | .pb(elem) | .pop\_back() | .insert(elem) | .find(elem) |
| vector | x | x |  | find(v.beg(),v.end(),elem) |
| string | x | x |  | x |
| map |  |  | (make\_pair(e,e)) | x |
| set |  |  | (elem) | x |

## Operations

if (stl.find(elem) != stl.end())//found

stl.count(elem);//count the occurrences of the element, sets and maps only

## Algorithms

int sum = accumulate(v.begin(), v.end(), sumStart);

int product = accumulate(v.begin(), v.end(), 1,multiplies<int>());

merge(v1.begin(), v1.end(), v2.begin(), v2.end(), back\_inserter(vecM));

fill(v.begin(), v.end(), value);iota(vec.begin(), vec.end(),start);

auto maxElement = max\_element(vec.begin(), vec.end());// use: cout << (maxElement != vec.end() ? \*maxElement : “NO”);

# Strings

string substr1=str1.substr(start, end);

int found1 = str.find(substr1);//first find at

int found2 = str.find(substr1, found1 + 1);//second find at

if (found1 != string::npos) //found

str1.replace(start, str2.size(), str2);

str1.erase(start, end);

str1.append(str2); str3 = str1 + str2;

vector<string> allStrings(string s){ //circular string

vector<string> v1(s.size());vector<string> v2(s.size());

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

v1[i] = s.substr(i,s.size()-i) + s.substr(0,i);

v2[i] = s.substr(s.size()-i,i) + s.substr(0,s.size()-i);

}

return v;

}

char ch;

isalpha(ch);isdigit(ch);isupper(ch);islower(ch);//char bools

tolower(ch);toupper(ch);//alphabet

char numC='1'; int numI=numC-'0';//toInt

int numI=1; char numC=numI+'0';//toChar

Usar string stram para evitar complejidad linear espacial de opeador +

std::stringstream ss; ss << str1.substr(0, 1) << str1.substr(3, 4);