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

struct node {
	string str;
	int num;
	double doub;
	char x;

	node(str_, num_, doub_, x_) {
		str = str_;
		num = num_;
		doub = doub_;
		x = x_;
	}
};
array<int, 3> arr; // -> {0, 0, 0}

// max size of 10^7 -> int, double, char
int arr[10000000];

// max size of 10^8 -> bool
bool arr[100000000];

bool comp(int el1, int el2) {
	if(el1 <= el2) {
		return true;
	}
	return false;
}
```

```
bool comp(pair<int,int> el1, pair<int,int> el2) {
	if(el1.first < el2.first) {
		return true;
	}
	if(el1.first == el2.first) {
		if(el1.second > el2.second) {
			return true;
		}
	}
	return false;
}

// arr
// pair<int,int> arr[] = {{1, 4},{5, 2},{5, 9}};
// after sorting arr[] = {{1, 4}, {5, 9}, {5, 2}}
sort(arr, arr+3, comp);

// sorts in ascending according to first
// if first is equal then sorts according to second in ascending
sort(arr, arr+3);
// i want you to sort this in such a way
// that the element who have first element in pair smaller
// appears first, and if first is equal then sort according
// to second and keep the larger second
```

```
int main() {

        // max size of 10^6 -> int, double, char
        int arr[1000000];



        // max size of 10^7 -> bool
        bool arr[10000000];

        double val = 10.0;
        cout << val << endl; // prints 10.0

        cout << raj::getVal() << endl; // prints 50



        int
        double
        char



        // create a data type where you store
        {string, int, double, char}



   // wrong way of defining
        node raj;
        raj.str = "striver";
        raj.num = 79;
        raj.doub = 91.0;

        node *raj = new node("striver", 79, 91.0, "");
```

```
node raj = node("striver", 79, 91.0, "");

{arr[], int, double};



// Arrays -> int arr[100];


array<int, 3> arr;  // -> {?, ?, ?}


array<int, 5> arr = {1}; // -> {1, 0, 0, 0, 0}


int arr[10000] = {0};



array<int, 5> arr;
arr.fill(10); -> /// {10, 10, 10, 10, 10}


arr.at(index);

for(int i = 0;i<5;i++) {
        cout << arr.at(i) << " ";
}


// iterators
```

```
// begin(), end(), rbegin(), rend()


//


array<int, 5> arr = {1, 3, 4, 5, 6};
for(auto it = arr.begin(); it!=arr.end();it++) {
	cout << *it << " ";
}




for(auto it = arr.rbegin(); it>arr.rend();it++) {
	cout << *it << " ";
}

for(auto it = arr.end() - 1; it>=arr.begin();it--) {
	cout << *it << " ";
}

// for each loop
for(auto it: arr) {
	cout << it << " ";
}

string s = "xhegcwe";
// x h e g c w e
for(auto c:s) {
	cout << c << " ";
}
```

```
// size
cout << arr.size();

// front
cout << arr.front(); // arr.at(0);

// back
cout << arr.back(); // arr.at(arr.size() - 1);




// VECTOR

int arr[50];

// segmentation fault if you push_back 10^7 times

vector<int> arr; // -> {}
cout << arr.size() << endl; // -> print 0
arr.push_back(0); // {0}
arr.push_back(2); // {0,2}
cout << arr.size() << endl; // -> print 2
arr.pop_back(); // {0}
cout << arr.size() << endl; // print 1

arr.push_back(0); // {0,0}
arr.push_back(2); // {0,0,2}



vec.clear(); // --> erase all elements at once {}
```

```
vector<int> vec1(4, 0); // -> {0,0,0,0}
vector<int> vec2(4, 10); // -> {10,10,10,10}

// copy the entire vec2 into vec3
vector<int> vec3(vec2.begin(), vec2.end()); // -> []
vector<int> vec3(vec2);



vector<int> raj;
raj.push_back(1); // raj.emplace_back(1); // emplace_back takes lesser time than push back
raj.push_back(3);
raj.push_back(2);
raj.push_back(5); // -> {1, 3, 2, 5}

vector<int> raj1(raj.begin(), raj.begin() + 2);  // -> {1, 3}



// lower bound , upper bound

// swap swap(v1, v2)
// begin(), end(), rbegin(), rend()



// to defining 2d vectors

vector<vector<int>> vec;

vector<int> raj1;
```

```
raj1.push_back(1);
raj1.push_back(2);

vector<int> raj2;
raj2.push_back(10);
raj2.push_back(20);

vector<int> raj3;
raj3.push_back(19);
raj3.push_back(24);
raj3.push_back(27);

vec.push_back(raj1);
vec.push_back(raj2);
vec.push_back(raj3);

// it is vector itself
for(auto vctr: vec) {
	for(auto it: vctr) {
		cout << it << " ";
	}
	cout << endl;
}


for(int i = 0;i<vec.size();i++) {
	for(int j = 0;j<vec[i].size();j++) {
		cout <<  vec[i][j] << " ";
	}
	cout << endl;
}
```

```
// define 10 x 20
vector<vector<int>> vec(10, vector<int> (20, 0));
vec.push_back(vector<int>(20, 0));
cout << vec.size() << endl; // 11 prints

vec[2].push_back(1);


vector<int> arr[4];
arr[1].push_back(0);


// 10 x 20 x 30 // int arr[10][20][30]
vector<vector<vector<int>>> vec(10, vector<vector<int>> vec(20, vector<int> (30, 0));)



// 2nd day

// arrays vectors struct

// set map

// set

// given n elements, tell me the number of unique elements
arr[] = {2, 5, 2, 1, 5} // 3 unique elements -> {1. 2. 5}

set<int> st;
```

```
int n;
cin >> n;
for(int i = 0;i<n;i++) {
        int x;
        cin >> x;
        st.insert(x);
}

cout << st.size();

// st -> {1, 2, 5}
// erase functionality
// log n
st.erase(st.begin()); // st.erase(iterator) // st -> {2, 5}

// log n
// 77777777777
st.erase(st.begin(), st.begin() + 2); // -> []
// st.erase(startIterator, endIterator)

st.erase(5) // st.erase(key) // delete the 5 -> {1, 2}


set<int> st = {1, 5, 7, 8};

auto it = st.find(7); // log n // it will be iterator to 7

auto it = st.find(9); // it = st.end();

st.emplace(6); // st.insert(6)
```

```
cout << st.size() << endl;



set<int> st;
st.insert(5); // -> {5}
st.insert(5); // -> {5}

for(auto it=st.begin(); it!=st.end();it++) {
        cout << *it << " ";
}

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

// delete the entire set
st.erase(st.begin(), st.end()); // makes sure the entire set is deleted



unordered_set<int> st;

st.insert(2);
st.insert(3);
st.insert(1);

// average time complexity is O(1)
// tle -> switch to set
// but the worst case is linear in nature, O(set size)

multiset<int> ms;
```

```
ms.insert(1);
ms.insert(1);
ms.insert(2);
ms.insert(2);
ms.insert(3); // ms.emplace(3)
// st -> {1, 1, 2, 2, 3}

ms.erase(2); // all the instances will be erased

auto it = ms.find(2); // returns an iterator pointing to the first element of 2
ms.clear(); // deleted the entire set
ms.erase(ms.begin(), ms.end()); // deletes the entire set
// log n in size


for(auto it=st.begin(); it!=st.end();it++) {
        cout << *it << " ";
}

for(auto it : st) {
        cout << it << endl;
}
// finds how many times 2 occurs
st.count(2);


ms.erase(ms.find(2));
ms.erase(ms.find(2), ms.find(2) + 2);
```

```
// Key Value
// raj -> 27
// hima -> 31
// sandeep -> 67
// tank -> 89
// map only stores unique keys
// log n is the tc of map
map<string, int> mpp;
mpp["raj"] = 27;
mpp["hima"] = 31;
mpp["praveer"] = 31;
mpp["sandeep"] = 67;
mpp["tank"] = 89;
mpp["raj"] = 29;
mpp.emplace("raj", 45);
mpp.erase("raj"); // mpp.erase(key)
mpp.erase(mpp.begin()); // mpp.erase(iterator)
mpp.clear(); // entire map is cleaned up
mpp.erase(mpp.begin(), mpp.begin()+2); // cleans up a given range
auto it = mpp.find("raj"); // points to where raj lies
auto it = mpp.find("simran"); // points to end since she does not exists

if(mpp.empty()) {
        cout << "Yes it is empty";
}
mpp.count("raj"); // always returns 1 as it stores only 1
// instance of raj

pair<int,int> pr;
```

```
pr.first = 1;
pr.second = 10;

// printing map
for(auto it: mpp) {
        cout << it.first << " " << it.second << endl;
}

for(auto it = mpp.begin(); it!=mpp.end();it++) {
        cout << it->first << " " << it->second << endl;
}

// does not stores in any order
unordered_map<int,int> mpp;
// unordered_map<pair<int,int>,int> mpp; xxxxxx
// o(1) in almost all cases
// o(n) in the worst case, where n is the container size


// Pair class
pair<int,int> pr = {1,2};
pair< pair<int,int>, int> pr = {{1,2}, 2};
cout << pr.first.second << endl;
pair<pair<int,int>, pair<int,int>> pr = {{1,2},{2, 4}};
cout << pr.first.first; -> 1
cout << pr.second.second; -> 4

vector<pair<int,int>> vec;
set<pair<int,int>> st;
map< pair<int,int>, int> mpp;
```

```
multimap<string, int> mpp;
mpp.emplace("raj", 2);
mpp.emplace("raj", 5);



// Stack and Queue
stack<int> st; // lifo ds
// pop
// top
// size
// empty
// push and emplace

st.push(2);
st.push(4);
st.push(3);
st.push(1);


cout << st.top() // prints 2
st.pop(); // deletes the last entered element
cout << st.top(); // prints 3
st.pop();
cout << st.top();

bool flag = st.empty(); // returns true if stack is empty, or false
```

```
// deleted the entire stack
while(!st.empty()) {
        st.pop();
}

cout << st.size() << endl; // number of elements in the stack

stack<int> st;
if(!st.empty()) {
        cout << st.top() << endl; // throw error
}



// queue // fifo operation ds
// push
// front
// pop
// size
// empty

queue<int> q;
q.push(1);
q.push(5);
q.push(3);
q.push(6);

cout << q.front();; // prints 1
q.pop();
cout << q.front(); // prints 5

// linear time
```

```
while(!q.empty()) {
        q.pop();
}
queue<int> q;
for(int i = 0;i<10;i++) q.push(i);



// priority_queue
// push
// size
// top pop empty
priority_queue<int> pq;
pq.push(1);
pq.push(5);
pq.push(2);
pq.push(6);

cout << pq.top(); // print 6
pq.pop();
cout << pq.top(); // print 5

priority_queue<pair<int,int>> pq;
pq.push(1, 5);
pq.push(1, 6);
pq.push(1, 7);

priority_queue<int> pq;
pq.push(-1); // pq.push(-1 * el);
pq.push(-5);
pq.push(-2);
```

```
pq.push(-6);

cout << -1 * pq.top() << endl; // prints 1

// min priority queue is
priority_queue<int, vector<int>, greater<int>> pq;
pq.push(1);
pq.push(5);
pq.push(2);
pq.push(6);

cout << pq.top() << endl; // prints 1



priority_queue<pair<int,int>, vector<pair<int,int>>, greater<pair<int,int>>> pq;



dequeue<int> dq;
// push_front()
// push_back()
// pop_front()
// pop_back()
// begin, end, rbegin, rend
// size
// clear
// empty
// at



list<int> ls;
```

```
// push_front()
// push_back()
// pop_front()
// pop_back()
// begin, end, rbegin, rend
// size
// clear
// empty
// at
// remove -> O(1)
ls.push_front(1);
ls.push_front(2);
ls.push_front(3);
ls.remove(2); -> // o(1) operation



// given N elements, print the elements that occurs maximum
// number of times
// input
// 5
// 1 3 3 3 2

// output
// 3



int n;
cin >> n;
map<int,int> mpp;
int maxi = 0;
```

```
for(int i = 0;i<n;i++) {
        int x;
        cin >> x;
        mpp[x]++;
        if(mpp[x] > mpp[maxi]) {
                maxi = x;
        }
}
cout << x << endl;
```

```
// given N elements, print all elements in sorted order
// input
// n = 6
/// 6 6 3 2 3 5

// output
// 2 3 3 5 6 6

int n;
cin >> n;
multiset<int> ms;
for(int i=0;i<n;i++) {
        int x;
        cin >> x;
        ms.insert(x);
}

for(auto it : ms) {
        cout << it << endl;
```

```
}



// Day 3
// Bitset
// int -> 16 bits
// char -> 8 bits
int a[100];
char a[100];

// bitset -> 1 bit

bitset<5> bt; // stores 1 or 0
cin >> bt; // 10111

// all
// true // false
cout << bt.all(); // returns a true or a false

// any
// true
cout << bt.any(); // bt -> 10011

// false
cout << bt.any(); // bt -> 00000

// count
// for bt -> 10100
// prints 2
```

```
cout << bt.count(); // print the number of set bits

// flip
// bt -> 10100
bt.flip(2); // bt will become 10000

bt.flip();

// none
// if none is set, then true, else false
// bt -> 10000
cout << bt.none(); // false

// bt -> 00000
cout << bt.none(); //true

// set
bt.set(); // 11111

bt.set(2); // sets the 2nd index

bt.set(2, 0);


// reset
bt.reset() // turn all indexes to 0

bt.reset(2); // turn the 2nd index to 0

// size
cout << bt.size(); // prints 5
```

```
        // test

        cout << bt.test(1); // check if the bit is set or not at index  1



        // Algorithms
        // sorting
        // array, vector
        int n;
        cin >> n;
        int arr[n];
        for(int i = 0;i<n;i++) cin >> arr[i];
        // takes n log n
        sort(arr, arr+n); // in increasing order
        // sort from 1 to 3
        sort(arr + 1, arr + 4);

vector<int> vec(5, 0);
for(int i = 0;i<n;i++) {
        cin >> vec[i];
}

sort(vec.begin(), vec.end()); // []

// vec -> {1, 6, 2, 7, 4}
//        0  1  2  3  4
// sort it so that only indexes from 1 to 3
// final vec -> {1, 2, 6, 7, 4}
sort(vec.begin() + 1, vec.begin() + 4); // [1, 4)
```

```
// If I wanna reverse
// reverse(startIterator, endIterator) -> []
reverse(arr, arr+n);


reverse(arr + 1, arr + 4);


reverse(vec.begin(), vec.end());


reverse(vec.begin() + 1, vec.begin() + 4);




// If i want to fine the maximum elements in any index range
// i to j give me the maximum

// *max_element(firstIterator, lastIterator);
int maxi = INT_MIN;
for(int k = i;k<=j;k++) {
        if(a[k] > maxi) {
                maxi = a[k];
        }
}


int el = *max_element(arr, arr+n);
int el = *min_element(arr, arr+n);


int el = *max_element(vec.begin(), vec.end());
int el = *min_element(vec.begin(), vec.end());
```

```
// I give you a range and I want you to find the sum in that range
// i - j, tell me the sum in that range i to j
int sum = 0;
for(int k = i;k<=j;k++) {
        sum += arr[k];
}

// accumulate(startIterator, endIterator, initialSum);
int sum = accumulate(arr, arr+n, 0);
int sum = accumulate(vec.begin(), vec.end(), 0);



// arr[] -> [1, 6, 7, 1, 2, 1, 3]
// x = 1
// tell me how many times the element 1 occurs in the array
int cnt = 0;
// O(N)
for(int i = 0;i<n;i++) {
        if(arr[i] == x) {
                cnt++;
        }
}
cout << cnt;

/// count(firstIterator, lastIterator, x)
int cnt = count(arr, arr+n, 1);
int cnt = count(vec.begin(), vec.end(), 1);
```

```
// arr[] -> {1, 2, 5, 1, 2, 4, 4}
// i want you to find the first occurrence of 2
// it is in the index 1

int ind = -1;
for(int i = 0;i<n;i++) {
        if(arr[i] == x) {
                ind = i;
                break;
        }
}
cout << ind;



// arr[] -> {1, 2, 5, 1, 2, 4, 4}
auto it = find(arr, arr+n, 2); // return an iterator
// pointing to the first instance of it, or else it
// returns pointing to the end() if it is not there

int ind = it - arr;



auto it = find(vec.begin(), vec.end(), 2);
int ind = it - vec.begin();



// arr[] -> {1, 5, 6, 2, 3, 5, 6}
// x = 4
auto it = find(vec.begin(), vec.end(), 4);
if(it == vec.end()) {
```

```
        cout << "element is not present";
}
else {
        cout << "Element is first present at: " << it - vec.begin();
}


// binary search
// this stl only works on SORTED arrrays
// arr[] -> {1, 5, 7, 9, 10}
// x = 9
// true -> 9 exists in my arr
// x = 8
// false -> 8 does not exist in my arr


// binary_search(firstIterator, lastIterator, x)
// returns a true or returns a false
// works in log n complexity
bool res = binary_search(arr, arr+n, 8);
bool res = binary_search(vec.begin(), vec.end(), 8);



// lower_bound function
// returns an iterator pointing to the first
// element which is not less than x
// arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
// x = 10
// x = 6
// x = 13
```

```
// this works in log N

auto it = lower_bound(arr, arr+n, x);
ind = it - arr;

auto it = lower_bound(vec.begin(), vec.end(), x);
int ind = it - vec.begin();

int ind =  lower_bound(vec.begin(), vec.end(), x) - vec.begin();



// upper bound
// returns an iterator which points to an element which is
// just greater than x
// arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
// x = 7
// x = 6
// x = 12 -> end() iterator
// x = 15 -> end() iterator

auto it = upper_bound(arr, arr+n, x);
ind = it - arr;

auto it = upper_bound(vec.begin(), vec.end(), x);
int ind = it - vec.begin();

int ind =  upper_bound(vec.begin(), vec.end(), x) - vec.begin();
```

```
// Q1. find me the first index where the element X lies
// find function can be used but that takes O(N) times
// the array is sorted..

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

int x;
cin >> x;

// arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
// find x = 7
int ind = lower_bound(arr, arr+n, x) - arr;

// find x = 6
int ind = lower_bound(arr, arr+n, x) - arr;




// There are couple of ways to do it
// 1st way
if(binary_search(arr, arr+n, x) == true) {
        cout << lower_bound(arr, arr+n, x) - arr;
}
else cout << "does not exists";
```

```
// 2nd way
int ind = lower_bound(arr, arr+n, x) - arr;
// arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
//////////////0  1  2  3  4  5   6   7   8   9   10
// find x = 13 -> ind = 11, which is out of bound
// hence arr[11] will give you runtime error
if(ind != n && arr[ind] == x) {
        cout << "Found at: " << ind;
}
else {
        cout << "Not found";
}



// Find me the last occurrence of x in an arr
// arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}
///index/////0  1  2  3  4  5   6   7   8   9   10

// last occurrence of x = 10, ans = 7th index
// last occurrence of x = 6, ans = does not exists
// last occurrence of x = 0,
// last occurence of x = 13
int ind = upper_bound(arr, arr+n, x) - arr;
ind -= 1;
if(ind>=0 && arr[ind] == x) {
        cout << "last occurrence: " << ind;
}
else {
        cout << "Does not exists";
}
```

// Q3. tell me the number of times the x appears in arr

// arr[] -> {1, 5, 7, 7, 8, 10, 10, 10, 11, 11, 12}

///index/////0 1 2 3 4 5   6   7   8   9   10

// x = 10, ans = 3

// x = 7, ans = 2

// Next Permutation

// string s = "abc"

// all permutations are as follows:

// abc

// acb

// bac

// bca

// cab

// cba

// s = "bca"

bool res = next_permutation(s.begin(), s.end());

// s = "cba"

bool res = next_permutation(s.begin(), s.end());

```
// if I give you any random string s = "bca"
// i want you to print all the permutations

string s = "bca";
sort(s.begin(), s.end()); // this makes the string as "abc"
do {
        cout << s << endl;
} while(next_permutation(s.begin(), s.end()));



int arr[] = {1, 6, 5};
int n = 3;
sort(arr, arr + n); // this makes the array as {1, 5, 6}
do {
        for(int i = 0;i<n;i++) cout << arr[i] << " ";
        cout << endl;
} while(next_permutation(arr, arr+n)) ;



// prev permutation
bool res = prev_permutation(s.begin(), s.end());



// COMPARATOR
sort(arr, arr+n); // sorts everything in ascending order
sort(arr, arr+n, comp);



// descending
sort(arr, arr+n, comp);
// greater<int> is an inbuilt comparator
```

```
    // which works only if you wanna do this in descending
    sort(arr, arr+n, greater<int>);




    // question of pair
    sort(arr, arr+n, greater<pair<int,int>>);



    vector<vector<int>> vec(n, vectorM<int>(n, 0));
}
```