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


********************* NOTE **************************
// C++ STL – by striver_79 (edited by k1ng_of_haskul)
// accompanying video lecture:
// https://www.youtube.com/watch?v=zBhVZzi5RdU&ab_channel=takeUforward
*****************************************************
_____________________________________________________________________________
// ## 'Pairs'

pair<int, int> p = {1, 3};
// prints 1 3
cout << p.first << " " << p.second;
pair<int, pair<int, int>> p = {1, {3, 4}};
// prints 1 4 3
cout << p.first << " " << p.second.second << " " << p.second.first;
pair<int, int> arr[] = { {1, 2}, {2, 5}, {5, 1}};
// Prints 5
cout << arr[1].second;
_____________________________________________________________________________
//## 'vector'

 // A empty vector
vector<int> v;          // {}
v.push_back(1);         // {1}
v.emplace_back(2);      // {1, 2}
vector<pair<int, int>>vec;
v.push_back({1, 2});
v.emplace_back(1, 2);
// Vector of size 5 with
// everyone as 100
vector<int> v(5, 100);        // {100, 100, 100, 100, 100}
// A vector of size 5 initialized with 0 might take garbage value,
// dependent on the vector
vector<int> v(5);             // {0, 0, 0, 0, 0}
vector<int> v1(5, 20);        // {20, 20, 20, 20, 20}
vector<int> v2(v1);           // {20, 20, 20, 20, 20}
// Take the vector to be {10, 20, 30, 40}
//iterators
vector<int>::iterator it = v.begin();
it++;
cout << *(it) << " "; // prints 20
it = it + 2;
cout << *(it) << " "; // prints 30
vector<int>::iterator it = v.end();
vector<int>::iterator it = v.rend();
vector<int>::iterator it = v.rbegin();
cout << v[0] << " " << v.at(0);
cout << v.back() << " ";
// Ways to print the vector
for (vector<int>::iterator it = v.begin(); it != v.end(); it++) {
cout << *(it) << " ";
}
```

```
for (auto it = v.begin(); it != v.end(); it++) {
cout << *(it) << " ";
}
for (auto it : v) {
cout << it << " ";
}
// {10, 20, 12, 23}
v.erase(v.begin()+1);
// {10, 20, 12, 23, 35}
v.erase(v.begin() + 2, v.begin() + 4);          // {10, 20, 35} [start, end)
// Insert function
vector<int>v(2, 100);                           // {100, 100}
v.insert(v.begin(), 300);                       // {300, 100, 100};
v.insert(v.begin() + 1, 2, 10);                 // {300, 10, 10, 100, 100}
vector<int> copy(2, 50);                        // {50, 50}
v.insert(v.begin(), copy.begin(), copy.end()); // {50, 50, 300, 10, 10, 100, 100}
// {10, 20}
cout << v.size();             // 2
//{10, 20}
v.pop_back();                 // {10}
// v1 -> {10, 20}
// v2 -> {30, 40}
v1.swap(v2);                  // v1 -> {30, 40} , v2 -> {10, 20}
v.clear();                    // erases the entire vector
cout << v.empty();            //true or false
```

```
//## ‘List’

list<int> ls;
ls.push_back(2);              // {2}
ls.emplace_back(4);           // {2, 4}
ls.push_front(5);             // {5, 2, 4};
ls.emplace_front();           //{2, 4};
// rest functions same as vector
// begin, end, rbegin, rend, clear, insert, size, swap
```

```
//## ‘deque’

deque<int>dq;
dq.push_back(1);              // {1}
dq.emplace_back(2);           // {1, 2}
dq.push_front(4);             // {4, 1, 2}
dq.emplace_front(3);          // {3, 4, 1, 2}
dq.pop_back();                // {3, 4, 1}
dq.pop_front();               // {4, 1}
dq.back();
dq.front();
// rest functions same as vector
// begin, end, rbegin, rend, clear, insert, size, swap
```

```
//## ‘stack’

stack<int> st;
```

```
st.push(1);             // {1}
st.push(2);             // {2, 1}
st.push(3);             // {3, 2, 1}
st.push(3);             // {3, 3, 2, 1}
st.emplace(5);          // {5, 3, 3, 2, 1}
cout << st.top();       // prints 5  "** st[2] is invalid **"
st.pop();               // st looks like {3, 3, 2, 1}
cout << st.top();       // 3
cout << st.size();      // 4
cout << st.empty();
stack<int>st1, st2;
st1.swap(st2);
```

```
//## ‘queue’

queue<int> q;
q.push(1);              // {1}
q.push(2);              // {1, 2}
q.emplace(4);           // {1, 2, 4}
q.back() += 5
cout << q.back();       // prints 9
// Q is {1, 2, 9}
cout << q.front();      // prints 1
q.pop();                // {2, 9}
cout << q.front();      // prints 2
// size swap empty same as stack
```

```
// # ‘Priority Queue’

priority_queue<int>pq;
pq.push(5);             // {5}
pq.push(2);             // {5, 2}
pq.push(8);             // {8, 5, 2}
pq.emplace(10);         // {10, 8, 5, 2}
cout << pq.top();       // prints 10
pq.pop();               // {8, 5, 2}
cout << pq.top();       // prints 8
// size swap empty function same as others
// Minimum Heap
priority_queue<int, vector<int>, greater<int>> pq;
pq.push(5);             // {5}
pq.push(2);             // {2, 5}
pq.push(8);             // {2, 5, 8}
pq.emplace(10);         // {2, 5, 8, 10}
cout << pq.top();       // prints 2
```

```
// ## ‘Set’

set<int>st;
st.insert(1);           // {1}
st.emplace(2);          // {1, 2}
st.insert(2);           // {1, 2}
st.insert(4);           // {1, 2, 4}
st.insert(3);           // {1, 2, 3, 4}
// Functionality of insert in vector can be used also, that only increases efficiency
```

```
// begin(), end(), rbegin(), rend(), size(), empty() and swap()
// are same as those of above

// {1, 2, 3, 4, 5}
auto it = st.find(3);
// {1, 2, 3, 4, 5}
auto it = st.find(6);
// {1, 4, 5}
st.erase(5);            // erases 5 // takes logarithmic time
int cnt = st.count(1);
auto it = st.find(3);
st.erase(it);           // it takes constant time
// {1, 2, 3, 4, 5}
auto it1 = st.find(2);
auto it2 = st.find(4);
st.erase(it1, it2);     // after erase {1, 4, 5} [first, last)
// lower_bound() and upper_bound() function works in the same way
// as in vector it does.
// This is the syntax
auto it = st.lower_bound(2);
auto it = st.upper_bound(3);
```

```
// ## ‘Multi Set’

// Everything is same as set
// only stores duplicate elements also
multiset<int>ms;
ms.insert(1);           // {1}
ms.insert(1);           // {1, 1}
ms.insert(1);           // {1, 1, 1}
ms.erase(1);            // all 1's erased
int cnt = ms.count(1);
// only a single one erased
ms.erase(ms.find(1));
ms.erase(ms.find(1), ms.find(1)+2);
// rest all function same as set
```

```
// ## ‘unordered set’

unordered_set<int> st;
// lower_bound and upper_bound function does not works, rest all functions are same
// as above,  it does not stores in any particular order it has a better complexity
// than set in most cases, except some when collision happens
```

```
// ## ‘map’

// {key, value}
map<int, int> mpp;
map<int, pair<int, int>> mpp;
map< pair<int, int>, int> mpp;
// key values can be anything
mpp[1] = 2;
mpp.emplace({3, 1});
mpp.insert({2, 4});
```

```
mpp[{2,3}] = 10;
{
{1, 2}
{2, 4}
{3, 1}
}
for(auto it : mpp) {
cout << it.first << " " << it.second << endl;
}
// same options for using iterators as we did in vector for the insert function
cout << mpp[1];               // prints 2
cout << mpp[5];               // prints 0, since it does not exists
auto it = mpp.find(3);        // points to the position where 3 is found
cout << *(it).second;
auto it = mpp.find(5);        // points to the end of the map since 5 not there
// lower_bound and upper_bound works exactly in the
// same way as explained in the other video
// This is the syntax
auto it = mpp.lower_bound(2);
auto it = mpp.upper_bound(3);
// erase, swap, size, empty, are same as above


_____________________________________________________________________________
// ## ‘multimap’

// everything same as map, only it can store multiple keys
// only mpp[key] cannot be used here
_____________________________________________________________________________
// ## ‘unordered map’

// same as set and unordered_Set difference.
_____________________________________________________________________________
// ## ‘comp’

bool comp(pair<int,int>p1, pair<int,int>p2) {
if(p1.second < p2.second) {
return true;
} else if(p1.second == p2.second){
if(p1.first>p2.second) return true;
}
return false;
}
_____________________________________________________________________________
// Miscellaneous

sort(a+2, a+4);         // [first, last)
sort(a, a+n, greater<int>);
pair<int,int> a[] = {{1,2}, {2, 1}, {4, 1}};
// sort it according to second element
// if second element is same, then sort
// it according to first element but in descending
sort(a, a+n ,comp);
// {4,1}, {2, 1}, {1, 2}};
```

```
int num = 7; // 111
int cnt = __builtin_popcount();
long long num = 165786578687;
int cnt = __builtin_popcountll();
string s = "123";
do {
cout << s << endl;
} while(next_permutation(s.begin(), s.end()));
// 123
// 132
// 213
// 231
// 312
// 321
int maxi = *max_element(a,a+n);
```