// C++ program for the above approach

#include <bits/stdc++.h>

using namespace std;

// Comparator

bool compareBy(const pair<int, int>& a,

const pair<int, int>& b)

{

if (a.second != b.second)

return a.second < b.second;

return a.first < b.first;

}

// Function to find maximum shops

// that can be visited by K persons

int maximumShops(int\* opening, int\* closing,

int n, int k)

{

// Store opening and closing

// time of shops

pair<int, int> a[n];

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

a[i].first = opening[i];

a[i].second = closing[i];

}

// Sort the pair of array

sort(a, a + n, compareBy);

// Stores the result

int count = 0;

// Stores current number of persons visiting

// some shop with their ending time

multiset<int> st;

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

// Check if current shop can be

// assigned to a person who's

// already visiting any other shop

bool flag = false;

if (!st.empty()) {

auto it = st.upper\_bound(a[i].first);

if (it != st.begin()) {

it--;

// Checks if there is any person whose

// closing time <= current shop opening

// time

if (\*it <= a[i].first) {

// Erase previous shop visited by the

// person satisfying the condition

st.erase(it);

// Insert new closing time of current

// shop for the person satisfying ?he

// condition

st.insert(a[i].second);

// Increment the count by one

count++;

flag = true;

}

}

}

// In case if no person have closing

// time <= current shop opening time

// but there are some persons left

if (st.size() < k && flag == false) {

st.insert(a[i].second);

count++;

}

}

// Finally print the ans

return count;

}

// Driver Code

int main()

{

// Given starting and ending time

int S[] = { 1, 8, 3, 2, 6 };

int E[] = { 5, 10, 6, 5, 9 };

// Given K and N

int K = 2, N = sizeof(S)

/ sizeof(S[0]);

// Function call

cout << maximumShops(S, E, N, K) << endl;

}