**B. Karen and Coffee -** [**https://codeforces.com/problemset/problem/816/B**](https://codeforces.com/problemset/problem/816/B)

**🧩 Problem Understanding**

Karen has **n recipes**, and each recipe gives a **temperature range** [li, ri] where coffee tastes good.

A temperature T is **admissible** if **at least k recipes** recommend brewing coffee at that temperature (i.e., T lies inside at least k of the [li, ri] intervals).

Karen then asks **q queries**, each query gives a range [a, b], and you must find **how many admissible integer temperatures** exist within [a, b].

**Example Intuition**

Input:

3 2 4

91 94

92 97

97 99

92 94

93 97

95 96

90 100

* Recipe 1 → [91, 94]
* Recipe 2 → [92, 97]
* Recipe 3 → [97, 99]
* k = 2 → temperature must appear in **at least 2 intervals**

Let’s find how many recipes recommend each temperature:

| **Temperature** | **Recipes Count** | **Admissible?** |
| --- | --- | --- |
| 91 | 1 | No |
| 92 | 2 | ✅ |
| 93 | 2 | ✅ |
| 94 | 2 | ✅ |
| 95 | 1 | No |
| 96 | 1 | No |
| 97 | 2 | ✅ |
| 98 | 1 | No |
| 99 | 1 | No |

→ So admissible temperatures are {92, 93, 94, 97}

Now queries:

1. [92,94] → 3 admissible
2. [93,97] → 3 admissible
3. [95,96] → 0 admissible
4. [90,100] → 4 admissible ✅

**🧠 How to Think About the Problem**

We have up to **200,000 recipes** and **200,000 queries**, and the temperature range can go up to **200,000** — meaning a brute force solution (checking each temperature for each recipe) would be **O(n × 200000)** → far too slow.

We need an **O(n + max\_temp + q)** solution.

**⚙️ Efficient Approach — Prefix Sum + Difference Array**

**Step 1: Use a Difference Array**

We can count how many intervals cover each temperature efficiently using a **difference array technique**.

* For each interval [l, r]:
* diff[l] += 1
* diff[r + 1] -= 1
* Then, take a prefix sum of this array → gives us the **number of recipes recommending each temperature**.

**Step 2: Mark Admissible Temperatures**

Create another array ok[temp] = 1 if count[temp] >= k, otherwise 0.

**Step 3: Prefix Sum for Queries**

Now, make a **prefix sum** array of ok[] called pref[], where:

pref[i] = pref[i-1] + ok[i]

Then for any query [a, b]:

answer = pref[b] - pref[a-1]

✅ Constant-time per query.

**🧮 Complexity Analysis**

| **Operation** | **Complexity** |
| --- | --- |
| Building diff array | O(n) |
| Prefix sum to get counts | O(max\_temp) |
| Building admissible prefix | O(max\_temp) |
| Answering q queries | O(q) |
| **Total** | **O(n + q + max\_temp)** |
| **Memory** | O(max\_temp) ≈ 200k |

Efficient and fits limits easily.

**✅ C++ Implementation**

#include <bits/stdc++.h>

using namespace std;

const int MAX = 200000 + 5;

int main() {

ios::sync\_with\_stdio(false);

cin.tie(nullptr);

int n, k, q;

cin >> n >> k >> q;

vector<int> diff(MAX, 0);

// Step 1: Build difference array

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

int l, r;

cin >> l >> r;

diff[l] += 1;

if (r + 1 < MAX) diff[r + 1] -= 1;

}

// Step 2: Build prefix sum to get count per temperature

vector<int> count(MAX, 0);

count[0] = diff[0];

for (int i = 1; i < MAX; i++)

count[i] = count[i - 1] + diff[i];

// Step 3: Mark admissible temperatures (count >= k)

vector<int> ok(MAX, 0);

for (int i = 1; i < MAX; i++) {

ok[i] = (count[i] >= k ? 1 : 0);

}

// Step 4: Build prefix sum for admissible counts

vector<int> pref(MAX, 0);

for (int i = 1; i < MAX; i++)

pref[i] = pref[i - 1] + ok[i];

// Step 5: Answer queries in O(1)

while (q--) {

int a, b;

cin >> a >> b;

cout << pref[b] - pref[a - 1] << "\n";

}

return 0;

}

**🧾 Summary**

| **Step** | **Description** |
| --- | --- |
| 1️⃣ | Use difference array to track how many intervals cover each temperature |
| 2️⃣ | Prefix sum → count of recipes per temperature |
| 3️⃣ | Mark admissible temperatures (count ≥ k) |
| 4️⃣ | Prefix sum again → answer queries instantly |
| ✅ | Time: O(n + q + 200000), Memory: O(200000) |