**B. Sereja and Suffixes -** [**https://codeforces.com/problemset/problem/368/B**](https://codeforces.com/problemset/problem/368/B)

**🔍 Problem Understanding**

We’re given:

* An integer array a of size n.
* m queries — each query gives an index li.

For each li, we need to find **how many distinct numbers** appear in the suffix starting from index li to the end of the array (n).

Formally:  
We want the count of unique elements in a[li...n].

**Example**

n = 10, m = 10

a = [1,2,3,4,1,2,3,4,100000,99999]

queries: 1 2 3 4 5 6 7 8 9 10

Let’s analyze:

| **l** | **subarray (a[l...n])** | **distinct count** |
| --- | --- | --- |
| 1 | [1,2,3,4,1,2,3,4,100000,99999] | 6 |
| 2 | [2,3,4,1,2,3,4,100000,99999] | 6 |
| 3 | [3,4,1,2,3,4,100000,99999] | 6 |
| 4 | [4,1,2,3,4,100000,99999] | 6 |
| 5 | [1,2,3,4,100000,99999] | 6 |
| 6 | [2,3,4,100000,99999] | 5 |
| 7 | [3,4,100000,99999] | 4 |
| 8 | [4,100000,99999] | 3 |
| 9 | [100000,99999] | 2 |
| 10 | [99999] | 1 |

**🧠 Naive Idea (TLE ❌)**

For each query li, scan from li to n and count distinct numbers using a set.

for each query li:

set<int> st;

for i from li to n:

st.insert(a[i]);

print st.size();

➡️ **Time complexity:** O(m \* n) = up to 10¹⁰ operations (too slow)

**⚡ Efficient Approach — Reverse Traversal (O(n + m))**

**Observation:**

If we know the number of distinct elements in suffix starting at index i+1,  
then we can compute for index i easily:

* If a[i] has not appeared before (in suffix after i), then distinct count increases by 1.
* Otherwise, it remains the same.

**✅ Step-by-Step Logic**

1. Create an array suffixDistinct[n+1] where:
2. suffixDistinct[i] = number of distinct elements in a[i...n]
3. Maintain a frequency array or boolean visited array for elements a[i].
4. Traverse from **right to left (n → 1)**:
   * If element a[i] was **not seen before**, increment current distinct count.
   * Store that count in suffixDistinct[i].
5. Answer each query li instantly by outputting suffixDistinct[li].

**🧩 Example Walkthrough**

a = [1,2,3,4,1,2,3,4,100000,99999]

n = 10

Start from right:

| **i** | **a[i]** | **seen before?** | **count** | **suffixDistinct[i]** |
| --- | --- | --- | --- | --- |
| 10 | 99999 | no | 1 | 1 |
| 9 | 100000 | no | 2 | 2 |
| 8 | 4 | no | 3 | 3 |
| 7 | 3 | no | 4 | 4 |
| 6 | 2 | no | 5 | 5 |
| 5 | 1 | no | 6 | 6 |
| 4 | 4 | yes | 6 | 6 |
| 3 | 3 | yes | 6 | 6 |
| 2 | 2 | yes | 6 | 6 |
| 1 | 1 | yes | 6 | 6 |

✅ Works perfectly.

**✅ Final C++ Solution**

#include <bits/stdc++.h>

using namespace std;

int main() {

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

cin.tie(nullptr);

int n, m;

cin >> n >> m;

vector<int> a(n + 1);

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

cin >> a[i];

vector<int> distinctSuffix(n + 1);

vector<bool> seen(100001, false); // since ai ≤ 10^5

int distinctCount = 0;

// Traverse from end to start

for (int i = n; i >= 1; --i) {

if (!seen[a[i]]) {

seen[a[i]] = true;

distinctCount++;

}

distinctSuffix[i] = distinctCount;

}

// Answer queries

while (m--) {

int l;

cin >> l;

cout << distinctSuffix[l] << '\n';

}

return 0;

}

**🧮 Complexity Analysis**

| **Operation** | **Complexity** |
| --- | --- |
| Build suffix counts | **O(n)** |
| Each query answer | **O(1)** |
| **Total** | **O(n + m)** |
| Space | **O(n + max(a[i])) ≈ O(10⁵)** |

✅ Fits easily within 1 second.

**🏁 Summary**

| **Step** | **Concept** |
| --- | --- |
| 1️⃣ | Traverse from right to left |
| 2️⃣ | Track seen numbers |
| 3️⃣ | Store running distinct count in suffixDistinct[i] |
| 4️⃣ | Answer queries in O(1) |