🔢 Problem: 713. Subarray Product Less Than K

Given:

- An array `nums` of positive integers.
- An integer `k`.

Goal:

- Count the number of **contiguous subarrays** where the **product of all the elements** is **strictly less than** `k`.

✅ Constraints:

- `1 <= nums.length <= 30,000`
- `1 <= nums[i] <= 1000`
- `0 <= k <= 1,000,000`

💡 Key Observations:

- All numbers are **positive** → No need to worry about zero or negative numbers changing product behavior.
- Products can grow very fast → Naive approach ($O(n^2)$ with all subarrays) would **TLE** for large `n`.
- Need an efficient approach: **Sliding Window** works best here.

🚀 Efficient Approach: Sliding Window

🧠 Intuition:

Use a **sliding window** `[i...j]`:

- Expand `j` to include new elements.
- Shrink `i` when product becomes **≥ k**.
- At each step, count valid subarrays ending at `j`.

✅ Steps:

1. Initialize:
   - `prod = 1` → to keep track of product in the window.
   - `i = 0`, `count = 0`.
2. Iterate `j` from 0 to n-1:
   - Multiply `prod` by `nums[j]`.
   - While `prod >= k`, divide `prod` by `nums[i]` and move `i` forward.
   - At this point, all subarrays ending at `j` and starting from `i` to `j` are valid. Count = `j - i + 1`.
3. Return `count`.

✅ Code (with Fix and Clean-up):

```
class Solution {
public:
    int numSubarrayProductLessThanK(vector<int>& nums, int k) {
        if (k <= 1) return 0;

        int count = 0;
        int prod = 1;
        int i = 0;

        for (int j = 0; j < nums.size(); j++) {
            prod *= nums[j];

            while (prod >= k && i <= j) {
```

```
            prod /= nums[i];
            i++;
        }

        count += (j - i + 1);
    }

    return count;
  }
};
```

🧪 Dry Run Example:

Input:

```
nums = [10, 5, 2, 6], k = 100
```

Initialize:

```
prod = 1, count = 0, i = 0
```

Iteration:

j = 0

- prod = 1 × 10 = 10
- 10 < 100 → count += (0 - 0 + 1) = 1
- count = 1

j = 1

- prod = 10 × 5 = 50
- 50 < 100 → count += (1 - 0 + 1) = 2

- count = 3

j = 2

- prod = 50 × 2 = 100
- 100 >= 100 → shrink window:
  - prod /= 10 → prod = 10, i = 1
- Now, prod = 10 < 100 → count += (2 - 1 + 1) = 2
- count = 5

j = 3

- prod = 10 × 6 = 60
- 60 < 100 → count += (3 - 1 + 1) = 3
- count = 8

✅ Final Answer: **8**

📌 Valid Subarrays:

```
[10], [5], [2], [6],
[10, 5], [5, 2], [2, 6],
[5, 2, 6]
```

🧹 Notes on the Original Code:

In your original code:

```
while (prod >= k) {
    count += (j - i); // ❌ Incorrect to count here.
    prod /= nums[i];
    i++;
}
```

This line is **wrong**:

```
count += (j - i);
```

Because we should only **add to the count after** the `prod < k` condition is satisfied (and for current `j`, not earlier).

That's why the correct logic is:

```
count += (j - i + 1); // ✅ Add number of valid subarrays ending at j
```

No need for post-processing loop either (`while (i < n)`), it was unnecessary and incorrect.

✅ Time and Space Complexity:

- **Time:** O(n)
- **Space:** O(1)

📘 Summary for Future Reference:

Step	Description
1	Use sliding window with product variable
2	Expand window by multiplying current number
3	Shrink window if product ≥ k
4	Count all valid subarrays ending at current index
5	Return total count
