1004. Max Consecutive Ones III

■ Problem Understanding

You are given:

- A binary array `nums` (only 0s and 1s)

- An integer `k` (maximum number of 0s you can flip to 1)

Goal: Find the **longest consecutive sequence of 1s** if you can flip at most `k` zeros to 1.

■ Example 1

Input: nums = [1,1,1,0,0,0,1,1,1,1,0], k = 2

You can flip at most 2 zeros.

If you flip 2 zeros at index 5 and 10:

[1,1,1,0,0,1,1,1,1,1,1]

The longest stretch of 1s is length **6**.

Output = 6.

■ 1. Brute Force Approach ($O(N^2)$)

Idea:

- Try every possible subarray.

- Count how many 0s are inside it.

- If 0s ≤ k → valid subarray → update max length.

■ Brute Force Code

C++

```cpp
```

```
int longestOnes(vector& nums, int k) {
int n = nums.size(), ans = 0;
for (int i = 0; i < n; i++) {
int zeros = 0;
for (int j = i; j < n; j++) {
if (nums[j] == 0) zeros++;
if (zeros > k) break;
ans = max(ans, j - i + 1);
}
}
return ans;
}
```

Java

```java
class Solution {
public int longestOnes(int[] nums, int k) {
int n = nums.length, ans = 0;
for (int i = 0; i < n; i++) {
int zeros = 0;
for (int j = i; j < n; j++) {
if (nums[j] == 0) zeros++;
if (zeros > k) break;
ans = Math.max(ans, j - i + 1);
}
}
return ans;
}
}
```

Python

```python
```

```
def longestOnes(nums, k):
n = len(nums)
ans = 0
for i in range(n):
zeros = 0
for j in range(i, n):
if nums[j] == 0:
zeros += 1
if zeros > k:
break
ans = max(ans, j - i + 1)
return ans
```

■ 2. Optimized: Two Pointers + Sliding Window (O(N))

Idea:

Maintain a window [left, right] that has at most `k` zeros.

- Expand `right` pointer → include nums[right].
- If zeros > k → shrink window from left.
- Keep track of max window size.

■ Optimized Code

C++

```cpp
int longestOnes(vector& nums, int k) {
int left = 0, zeros = 0, ans = 0;
for (int right = 0; right < nums.size(); right++) {
if (nums[right] == 0) zeros++;
while (zeros > k) {
if (nums[left] == 0) zeros--;
```

```
left++;
}
ans = max(ans, right - left + 1);
}
return ans;
}
```

Java

```java
class Solution {
public int longestOnes(int[] nums, int k) {
int left = 0, zeros = 0, ans = 0;
for (int right = 0; right < nums.length; right++) {
if (nums[right] == 0) zeros++;
while (zeros > k) {
if (nums[left] == 0) zeros--;
left++;
}
ans = Math.max(ans, right - left + 1);
}
return ans;
}
}
```

Python

```python
def longestOnes(nums, k):
left = zeros = ans = 0
for right in range(len(nums)):
if nums[right] == 0:
zeros += 1
while zeros > k:
```

```
if nums[left] == 0:
zeros -= 1
left += 1
ans = max(ans, right - left + 1)
return ans
```

■ Complexity

Approach	Time Complexity	Space Complexity
Brute Force	$O(N^2)$	O(1)
Two Pointer (Optimized)	O(N)	O(1)

■ Summary

Concept	Description
What we do	Find longest subarray with ≤ k zeros
Technique	Sliding window
Window validity	zeros ≤ k
When invalid	Move left pointer until valid
Best case performance	O(N)