🔢 LeetCode 1004 — Max Consecutive Ones III

🧠 Problem Intuition (Priyansh Style)

Given a binary array `nums` and an integer `k`, we are allowed to **flip at most** `k` **zeroes into ones**.

Our goal is to find the **maximum number of consecutive 1s** that we can get in the array after using those `k` flips wisely.

🎯 Key Idea:

- Flip means: `0 → 1`, max `k` times.
- We need to find the **longest subarray** containing only `1s` if we are allowed to flip at most `k` zeroes.

⚙️ Sliding Window Approach

We'll use two pointers `i` and `j` to create a window `[i, j)`.

This window will represent the current subarray we're examining.

🪜 Step-by-step Thought Process:

1. Initialize:
   - `i = 0`, `j = 0` → Window boundaries
   - `flips = 0` → Count of 0s flipped so far
   - `maxlen = INT_MIN` → Track max window length
2. Traverse array using `j`:
   - If `nums[j] == 1` → No flip needed, just move `j++`
   - If `nums[j] == 0`:
     - If `flips < k` → Flip and `flips++`, move `j++`
     - Else → Flips exceeded:
       - First calculate `len = j - i`
       - Update `maxlen = max(maxlen, len)`
       - Shrink the window from `i` till we remove one flipped 0:

- Skip 1s (`nums[i] == 1`)
- Once a 0 is found, reduce `flips--`, then move `i++`

3. After loop ends, do one final check:
   - `len = j - i`
   - `maxlen = max(maxlen, len)`
4. Return `maxlen`

💻 Code (As Written by You — Untouched ✅)

```
class Solution {
public:
    int longestOnes(vector<int>& nums, int k) {
        int n = nums.size();
        int i = 0,j = 0 ,flips = 0;
        int len = INT_MIN;
        int maxlen = INT_MIN;
        while(j < n){
            if(nums[j] == 1) j++;
            else{
                if(flips < k){
                    j++;
                    flips++;
                }
                else{
                    len = j - i;
                    maxlen = max(len,maxlen);
                    while(nums[i] == 1){
                        i++;
                    };
                    flips--;
                    i++;
                }
            }
        }
        len = j - i;
        maxlen = max(len,maxlen);
        return maxlen;
```

```
    }
};
```

✅ No changes done — because your logic is **solid** and approach is **clean**.

🧪 Dry Run Example

Example:

```
nums = [1,1,1,0,0,0,1,1,1,1,0], k = 2
```

Step	i	j	flips	Window	Action
1	0	3	0	[1,1,1]	1s → window expand
2	0	4	1	[1,1,1,0]	Flip first 0
3	0	5	2	[1,1,1,0,0]	Flip second 0
4	0	6	2	[1,1,1,0,0,0]	Flip limit reached (k=2)
5	0	6	2		Need to shrink from i
	0	6	2	i=0 (1) → skip	
	1	6	2	i=1 (1) → skip	
	2	6	2	i=2 (1) → skip	
	3	6	1	i=3 (0) → unflip	flips--, i++
...				Continue...	Keep expanding and tracking

➡️ Final Answer: **6**

✅ Final Takeaways (Priyansh Vibe 💭)

- Use sliding window when you need to find **longest subarray** with constraints (like flips).
- `j` grows the window.

  `i` shrinks it when **flips exceed limit**.
- Keep tracking max length → `maxlen = max(maxlen, j - i)`
- Don't forget the **final window check after the loop**.
