

Max Consecutive Ones III

Given a binary array `nums` and an integer `k`, return the maximum number of consecutive `1`'s in the array if you can flip at most `k` `0`'s.

Example 1:

Input: `nums = [1,1,1,0,0,0,1,1,1,1,0]`, `k = 2`
Output: 6
Explanation: `[1,1,1,0,0,0,1,1,1,1,0]`
Bolded numbers were flipped from 0 to 1. The longest subarray is underlined.

Example 2:

Input: `nums = [0,0,1,1,0,0,1,1,1,0,1,1,0,0,0,1,1,1]`, `k = 3`
Output: 10
Explanation: `[0,0,1,1,1,1,1,1,1,0,0,0,1,1,1]`
Bolded numbers were flipped from 0 to 1. The longest subarray is underlined.

Constraints:

- $1 \leq \text{nums.length} \leq 10^5$
- `nums[i]` is either 0 or 1.
- $0 \leq k \leq \text{nums.length}$

```
class Solution {
public:
    int longestOnes(vector<int>& nums, int k) {
        int left = 0, right = 0;
        int maxLength = 0;
        int zeroCount = 0;

        while (right < nums.size()) {
            if (nums[right] == 0) {
                zeroCount++;
            }

            while (zeroCount > k) {
                if (nums[left] == 0) {
                    zeroCount--;
                }
                left++;
            }

            maxLength = max(maxLength, right - left + 1);
            right++;
        }

        return maxLength;
    }
};
```

Time Complexity: $O(N)$
Space Complexity: $O(1)$

→ Approach:-

- Just use sliding window.

```
class Solution {
public:
    int longestOnes(vector<int>& nums, int k) {
        queue<int> q;
        int n = nums.size();
        int i = 0, j = 0;
        int longestOnesCnt = 0;

        while (j < n) {
            if (nums[j] == 0) {
                q.push(j);
            }

            if (q.size() > k) {
                i = q.front() + 1;
                q.pop();
            }

            longestOnesCnt = max(longestOnesCnt, j - i + 1);
            j++;
        }

        return longestOnesCnt;
    }
};
```

Time Complexity: $O(N)$
Space Complexity: $O(k)$