# 🔥 Priyansh's Revision Notes for: **Leetcode 1493 - Longest Subarray of 1's After Deleting One Element**

---

## 🔶 🧠 Problem Statement (Simplified)

Given a binary array `nums` (only contains `0s` and `1s`), **delete any one element** from it. After deletion, return the **length of the longest contiguous subarray that contains only 1's**.

> Important:
>
> - Deletion is *mandatory* (you **must** delete one element, even if all are 1s).
> - After deletion, you have to find the longest subarray consisting only of 1s.

---

## 🔶 📌 Example Walkthroughs

### ✅ Example 1:

```
Input:  [1,1,0,1]
Delete index 2 (value 0)
Result: [1,1,1] → Length = 3
```

### ✅ Example 2:

```
Input:  [0,1,1,1,0,1,1,0,1]
Delete index 4 (value 0)
Result: [1,1,1,1,1] → Length = 5
```

## ✅ Example 3:

```
Input:  [1,1,1]
Deletion required → remove one 1
Result: [1,1] → Length = 2
```

---

## 🔶 🔍 Intuition (My Thought Process)

Maine is problem ko sliding window approach se solve karne ka socha — kyunki hume continuous subarray ki baat ho rahi hai.

- **Window [i...j] maintain karo**, jahan:
    - Sirf ek hi `0` allowed ho (jisko hum "delete" kar denge).
- Agar `0` mila aur abhi tak delete nahi kiya → toh usko "ignore" karne ke liye x_factor (delete power) ka use karo.
- Agar doosra `0` mila → toh ab window shrink karo from the left (`i++`) until ek `0` remove ho jaye.
- Har valid window par, length calculate karo `j - i - 1` (kyunki ek element humne delete kiya hai).
- Maximum length ko update karte raho.

---

## 🔶 🚀 Approach (Step-by-Step)

1. **Initialize Pointers**:

   `i = 0`, `j = 0` → window start and end

   `x_factor = 1` → means ek `0` ko ignore kar sakte hain (delete karne ke liye)

2. **Traverse array with `j`**:
    - Agar `nums[j] == 1`: simple, window extend karo → `j++`
    - Agar `nums[j] == 0`:
        - Aur agar `x_factor > 0`: use delete power → `j++`, `x_factor--`
        - Agar `x_factor == 0`: window mein already ek `0` delete ho chuka, toh ab window shrink karo from left:

    - Pehle window ka length calculate karo `len = j - i - 1`
    - `maxlen` update karo
    - Then left pointer `i` ko aage badhao jab tak wo 0 encounter na kare.
    - Uske baad `i++` to remove that 0 and regain `x_factor = 1`

3. **Loop ke baad**:
   - Final window ka length calculate karo (j-i-1), and maxlen update karo.

---

## 🔶 💻 Code (With Comments)

```
class Solution {
public:
    int longestSubarray(vector<int>& nums) {
        int n = nums.size();
        int i = 0; // window start
        int j = 0; // window end
        int len;
        int x_factor = 1; // power to delete one 0
        int maxlen = INT_MIN;

        while (j < n) {
            if (nums[j] == 1) {
                j++; // window expand
            } else {
                if (x_factor == 1) {
                    // Use delete power
                    j++;
                    x_factor--;
                } else {
                    // Already used delete power → shrink window
                    len = j - i - 1;
                    maxlen = max(len, maxlen);

                    // Move i ahead to remove one 0 from window
                    while (nums[i] == 1) i++;
                    i++; // move past the 0
                    x_factor++; // regain delete power
                }
            }
        }

        // Final check after loop
        len = j - i - 1;
        maxlen = max(len, maxlen);

        return maxlen;
    }
};
```

---

## 🔶 ✅ Dry Run Example: `[0,1,1,1,0,1,1,0,1]`

1. `i = 0, j = 0 → nums[0] = 0 → x_factor = 1 → use it → j++, x_factor = 0`
2. `j = 1 to 3 → all 1s → j++`
3. `j = 4 → nums[4] = 0 → x_factor = 0 → can't delete → calculate len = 4 - 0 - 1 = 3`
4. Move i ahead till `nums[i] == 0`, i = 0 → `i++` → now i = 1, `x_factor = 1`
5. Repeat...

Final maxlen = **5**

---

## 🔶 📘 Final Notes / Takeaways

- Ye problem ek **variant of longest subarray with at most K zeroes** hai — but yahan `K = 1`, aur ek `0` delete *karna hi hai*.
- Sliding window with two pointers kaafi efficient hai, `O(N)` time complexity.
- Edge Case: Jab poori array 1s se bhari ho, tab bhi ek element delete karna hi padega → final answer `n-1` hoga.

---

## ✅ TC: `O(N)`

### ✅ SC: `O(1)`

(No extra space used except variables)

---

If you're revising this later, just remember:

> "Window banayi thi delete power ke saath, jahan sirf ek 0 allowed hai. Doosra 0 aate hi window chhoti kar di, aur har valid window ka size check kiya after removing 1 element."

---