LeetCode 2483 – Minimum Penalty for a Shop

✅ Problem Statement

You are given a string `customers` consisting of characters `'Y'` and `'N'`:

- `'Y'` → customers come at that hour.
- `'N'` → no customers come at that hour.

If the shop **closes at hour** `j` (`0 <= j <= n`):

- For every hour when the shop is **open** and no customers come → **penalty += 1**
- For every hour when the shop is **closed** and customers come → **penalty += 1**

Find the **earliest hour** to close the shop to **minimize penalty**.

✅ Example

Input:

`customers = "YYNY"`

Output:

`2`

Explanation:

```
j=0 → penalty = 3
j=1 → penalty = 2
j=2 → penalty = 1 ✅
j=3 → penalty = 2
j=4 → penalty = 1
Answer = 2 (earlier than 4)
```

✅ Constraints

- `1 <= customers.length <= 10^5`
- `customers` contains only `'Y'` and `'N'`.

✅ Approach 1: Prefix + Suffix Arrays

We calculate:

- `pre[i]` = number of `'N'` before index `i` (penalty when shop is open)
- `suff[i]` = number of `'Y'` from index `i` onward (penalty when shop is closed)

Total penalty at hour i:

```
penalty = pre[i] + suff[i]
```

Choose smallest penalty, earliest index.

✅ Steps

1. Create `pre` array:
   - Traverse from left, count `'N'` → store in `pre`.
2. Create `suff` array:
   - Traverse from right, count `'Y'` → store in `suff`.
3. For every `i` in `0...n`:

   ```
   penalty = pre[i] + suff[i]
   track minimum
   ```

✅ Dry Run for "YYNY"

```
customers = "YYNY", n=4

pre:  [0, 0, 0, 1, 1]
suff: [3, 2, 1, 1, 0]
```

Penalties:
i=0 → 3
i=1 → 2
i=2 → 1 ✅
i=3 → 2
i=4 → 1
Answer = 2

✅ Code (Prefix + Suffix)

```
class Solution {
public:
    int bestClosingTime(string c) {
        int n = c.length();
        vector<int> pre(n + 1), suff(n + 1);

        // Fill prefix array (count of 'N')
        int p_n = 0;
        for (int i = 0; i <= n; i++) {
            pre[i] = p_n;
            if (i < n && c[i] == 'N') p_n++;
        }

        // Fill suffix array (count of 'Y')
        int s_y = 0;
        for (int i = n; i >= 0; i--) {
            if (i < n && c[i] == 'Y') s_y++;
            suff[i] = s_y;
        }

        int mn = INT_MAX, min_dex = 0;
        for (int i = 0; i <= n; i++) {
            int penalty = pre[i] + suff[i];
            if (penalty < mn) {
                mn = penalty;
```

```
                min_dex = i;
            }
        }
        return min_dex;
    }
};
```

✅ Complexity

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

✅ Approach 2: Optimized (O(1) Space)

We start by calculating penalty if the shop is closed at hour 0:

```
penalty = count of 'Y' in string
```

Then iterate:

- If `c[i] == 'Y'` → penalty-- (since keeping shop open avoids penalty)
- If `c[i] == 'N'` → penalty++ (since keeping shop open adds penalty)

  Track minimum.

✅ Code (Optimized)

```
class Solution {
public:
    int bestClosingTime(string customers) {
        int penalty = 0;
        for (char ch : customers)
            if (ch == 'Y') penalty++;

        int minPenalty = penalty, bestHour = 0;
```

```
        for (int i = 0; i < customers.size(); i++) {
            if (customers[i] == 'Y') penalty--;
            else penalty++;
            if (penalty < minPenalty) {
                minPenalty = penalty;
                bestHour = i + 1;
            }
        }
        return bestHour;
    }
};
```

✅ Complexity

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

✅ Summary

Approach	Time	Space
Prefix+Suffix	O(n)	O(n)
Optimized	O(n)	O(1)

🔥 **Recommended:** Use the **optimized version** for large inputs.