Problems Solvable using this template

- 3. Longest Substring Without Repeating Characters
- 159. Longest Substring with At Most Two Distinct Characters (Medium)
- 340. Longest Substring with At Most K Distinct Characters
- 424. Longest Repeating Character Replacement
- 487. Max Consecutive Ones II
- 713. Subarray Product Less Than K
- 1004. Max Consecutive Ones III
- 1208. Get Equal Substrings Within Budget (Medium)
- 1493. Longest Subarray of 1's After Deleting One Element
- 1695. Maximum Erasure Value
- 1838. Frequency of the Most Frequent Element
- 2009. Minimum Number of Operations to Make Array Continuous
- 2024. Maximize the Confusion of an Exam

The following problems are also solvable using the shrinkable template with the "At Most to Equal" trick

- 930. Binary Subarrays With Sum (Medium)
- 992. Subarrays with K Different Integers
- 1248. Count Number of Nice Subarrays (Medium)
- 2062. Count Vowel Substrings of a String (Easy)

Template 1: Sliding Window (Shrinkable)

The best template I've found so far:

```
int i = 0, j = 0, ans = 0;
for (; j < N; ++j)
{
    // CODE: use A[j] to update state which might make the window invalid
    for (; invalid(); ++i) { // when invalid, keep shrinking the left edge until it's valid again
        // CODE: update state using A[i]
    }
    ans = max(ans, j - i + 1); // the window [i, j] is the maximum window we've found thus far
}
return ans;
```

Template 2: Sliding Window (Non-shrinkable)

```
int i = 0, j = 0;
for (; j < N; ++j) {
    // CODE: use A[j] to update state which might make the window invalid
    if (invalid()) { // Increment the left edge ONLY when the window is invalid. In this way, the window GROWs when it's valid, and SHIFTs
    when it's invalid
        // CODE: update state using A[i]
        ++i;
    }
    // after `++j` in the for loop, this window `[i, j)` of length `j - i` MIGHT be valid.
}
return j - i; // There must be a maximum window of size `j - i`.
```

Essentially, we GROW the window when it's valid, and SHIFT the window when it's invalid.

Note that there is only a SINGLE for loop now!

Apply these templates to other problems

1493. Longest Subarray of 1's After Deleting One Element (Medium)

Sliding Window (Shrinkable)

1. What's state? cnt as the number of 0s in the window.
2. What's invalid? cnt > 1 is invalid.

```
// OJ: https://leetcode.com/problems/longest-subarray-of-1s-after-deleting-one-element/
// Author: github.com/lzl124631x
// Time: O(N)
// Space: O(1)
class Solution {
public:
    int longestSubarray(vector<int>& A) {
        int i = 0, j = 0, N = A.size(), cnt = 0, ans = 0;
        for (; j < N; ++j) {
            cnt += A[j] == 0;
            while (cnt > 1) cnt -= A[i++] == 0;
            ans = max(ans, j - i); // note that the window is of size `j - i + 1`. We use `j - i` here because we need to delete a number.
        }
        return ans;
    }
};
```

Sliding Window (Non-shrinkable)

```
// OJ: https://leetcode.com/problems/longest-subarray-of-1s-after-deleting-one-element/
// Author: github.com/lzl124631x
// Time: O(N)
// Space: O(1)
class Solution {
public:
    int longestSubarray(vector<int>& A) {
        int i = 0, j = 0, N = A.size(), cnt = 0;
        for (; j < N; ++j) {
            cnt += A[j] == 0;
            if (cnt > 1) cnt -= A[i++] == 0;
        }
        return j - i - 1;
    }
};
```

3. Longest Substring Without Repeating Characters (Medium)

Sliding Window (Shrinkable)

1. state: cnt[ch] is the number of occurrence of character ch in window.
2. invalid: cnt[s[j]] > 1 is invalid.

```
// OJ: https://leetcode.com/problems/longest-substring-without-repeating-characters/
// Author: github.com/lzl124631x
// Time: O(N)
// Space: O(1)
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int i = 0, j = 0, N = s.size(), ans = 0, cnt[128] = {};
        for (; j < N; ++j) {
            cnt[s[j]]++;
            while (cnt[s[j]] > 1) cnt[s[i++]]--;
            ans = max(ans, j - i + 1);
        }
        return ans;
    }
};
```

Sliding Window (Non-shrinkable)

Note that since the non-shrinkable window might include multiple duplicates, we need to add a variable to our state.

1. state: dup is the number of different kinds of characters that has duplicate in the window. For example, if window contains aabbc, then dup = 2 because a and b has duplicates.
2. invalid: dup > 0 is invalid

```
// OJ: https://leetcode.com/problems/longest-substring-without-repeating-characters/
// Author: github.com/lzl124631x
// Time: O(N)
// Space: O(1)
class Solution {
public:
    int lengthOfLongestSubstring(string s) {
        int i = 0, j = 0, N = s.size(), cnt[128] = {}, dup = 0;
        for (; j < N; ++j) {
            dup += ++cnt[s[j]] == 2;
            if (dup) dup -= --cnt[s[i++]] == 1;
        }
        return j - i;
    }
};
```

713. Subarray Product Less Than K (Medium)

Sliding Window (Shrinkable)

- state: prod is the product of the numbers in window
- invalid: prod >= k is invalid.

Note that since we want to make sure the window [i, j] is valid at the end of the for loop, we need i <= j check for the inner for loop. i == j + 1 means this window is empty.

Each maximum window [i, j] can generate j - i + 1 valid subarrays, so we need to add j - i + 1 to the answer.

```cpp
// OJ: https://leetcode.com/problems/subarray-product-less-than-k/
// Author: github.com/lzl124631x
// Time: O(N)
// Space: O(1)
class Solution {
public:
    int numSubarrayProductLessThanK(vector<int>& A, int k) {
        if (k == 0) return 0;
        long i = 0, j = 0, N = A.size(), prod = 1, ans = 0;
        for (; j < N; ++j) {
            prod *= A[j];
            while (i <= j && prod >= k) prod /= A[i++];
            ans += j - i + 1;
        }
        return ans;
    }
};
```

The non-shrinkable template is not applicable here since we need to the length of each maximum window ending at each position