3. Longest Substring Without Repeating Characters

Given a string `s` , find the length of the **longest substring** without duplicate characters.

Example 1:

```
Input: s = "abcabcbb"
Output: 3
Explanation: The answer is "abc", with the length of 3.
```

Example 2:

```
Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.
```

Example 3:

```
Input: s = "pwwkew"
Output: 3
Explanation: The answer is "wke", with the length of 3.
Notice that the answer must be a substring, "pwke" is a subsequence and not a substri
```

Constraints:

- `0 <= s.length <= 5 * 10^4`
- `s` consists of English letters, digits, symbols and spaces.

```
class Solution:
    def lengthOfLongestSubstring(self, s: str) -> int:
        se = {}
        start = 0
        maxx_len = 0

        for i in range(len(s)):
            if s[i] in se and se[s[i]] >= start:
                start = se[s[i]] + 1
            se[s[i]] = i
            maxx_len = max(maxx_len, i - start + 1)

        return maxx_len
```

---

424. Longest Repeating Character Replacement

You are given a string `s` and an integer `k`. You can choose any character of the string and change it to any other uppercase English character. You can perform this operation at most `k` times.

Return *the length of the longest substring containing the same letter you can get after performing the above operations*.

Example 1:

```
Input: s = "ABAB", k = 2
Output: 4
Explanation: Replace the two 'A's with two 'B's or vice versa.
```

Example 2:

```
Input: s = "AABABBA", k = 1
Output: 4
Explanation: Replace the one 'A' in the middle with 'B' and form "AABBBBA".
The substring "BBBB" has the longest repeating letters, which is 4.
There may exists other ways to achieve this answer too.
```

Constraints:

- `1 <= s.length <= 10^5`
- `s` consists of only uppercase English letters.

- `0 <= k <= s.length`

```
class Solution:
    def characterReplacement(self, s: str, k: int) -> int:
        count = {}
        res = 0

        l = 0
        maxf = 0
        for r in range(len(s)):
            count[s[r]] = count.get(s[r], 0) + 1
            maxf = max(maxf, count[s[r]])

            if (r - l + 1) - maxf > k:
                count[s[l]] -= 1
                l += 1
            res = max(res, r-l+1)
        return res
```

930. Binary Subarrays With Sum

Given a binary array `nums` and an integer `goal`, return *the number of non-empty **subarrays** with a sum* `goal`.

A **subarray** is a contiguous part of the array.

Example 1:

```
Input: nums = [1,0,1,0,1], goal = 2
Output: 4
Explanation: The 4 subarrays are bolded and underlined below:
[1,0,1,0,1]
[1,0,1,0,1]
[1,0,1,0,1]
[1,0,1,0,1]
```

Example 2:

```
Input: nums = [0,0,0,0,0], goal = 0
Output: 15
```

Constraints:

- `1 <= nums.length <= 3 * 10^4`
- `nums[i]` is either `0` or `1`.
- `0 <= goal <= nums.length`

```
from collections import defaultdict
from typing import List

class Solution:
    def numSubarraysWithSum(self, nums: List[int], goal: int) -> int:
        count = defaultdict(int)
        count[0] = 1
        prefix_sum = 0
        result = 0

        for num in nums:
            prefix_sum += num
            result += count[prefix_sum - goal]
            count[prefix_sum] += 1

        return result
```

1004. 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: [1,1,1,0,0,1,1,1,1,1,1]
Bolded numbers were flipped from 0 to 1. The longest subarray is underlined.
```

Example 2:

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

Constraints:

- `1 <= nums.length <= 10^5`
- `nums[i]` is either `0` or `1`.
- `0 <= k <= nums.length`

```
from typing import List

class Solution:
    def longestOnes(self, nums: List[int], k: int) -> int:
        left = 0
        zeros = 0
        max_len = 0

        for right in range(len(nums)):
            if nums[right] == 0:
                zeros += 1

            while zeros > k: #shirk important
                if nums[left] == 0:
                    zeros -= 1
                left += 1

            max_len = max(max_len, right - left + 1)

        return max_len
```

1248. Count Number of Nice Subarrays

Given an array of integers `nums` and an integer `k`. A continuous subarray is called **nice** if there are `k` odd numbers on it.

Return *the number of **nice** sub-arrays*.

Example 1:

```
Input: nums = [1,1,2,1,1], k = 3
Output: 2
Explanation: The only sub-arrays with 3 odd numbers are [1,1,2,1] and [1,2,1,1].
```

Example 2:

```
Input: nums = [2,4,6], k = 1
Output: 0
Explanation: There are no odd numbers in the array.
```

Example 3:

```
Input: nums = [2,2,2,1,2,2,1,2,2,2], k = 2
Output: 16
```

Constraints:

- 1 <= nums.length <= 50000
- 1 <= nums[i] <= 10^5
- 1 <= k <= nums.length

```
from collections import defaultdict
class Solution:
    def numberOfSubarrays(self, nums: List[int], k: int) -> int:
        cnt = defaultdict(int)
        cnt[0] = 1
        prefix = 0
        res = 0
        for num in nums:
            if num%2 ==1:
                prefix+=1
            res += cnt[prefix-k ]
            cnt[prefix]+=1
        return res
```

1358. Number of Substrings Containing All Three Characters

Given a string `s` consisting only of characters *a*, *b* and *c*.

Return the number of substrings containing **at least** one occurrence of all these characters *a*, *b* and *c*.

Example 1:

```
Input: s = "abcabc"
Output: 10
Explanation: The substrings containing at least one occurrence of the characters a, b
```

Example 2:

```
Input: s = "aaacb"
Output: 3
Explanation: The substrings containing at least one occurrence of the characters a, b
```

Example 3:

```
Input: s = "abc"
Output: 1
```

Constraints:

- `3 <= s.length <= 5 x 10^4`
- `s` only consists of *a*, *b* or *c* characters.

```
from collections import defaultdict

class Solution:
    def numberOfSubstrings(self, s: str) -> int:
        count = defaultdict(int)
        total = 0
        left = 0

        for right in range(len(s)):
            count[s[right]] += 1

            while count['a'] > 0 and count['b'] > 0 and count['c'] > 0:
                total += len(s) - right
                count[s[left]] -= 1
                left += 1

        return total
```

1423. Maximum Points You Can Obtain from Cards

There are several cards **arranged in a row**, and each card has an associated number of points.
The points are given in the integer array `cardPoints`.

In one step, you can take one card from the beginning or from the end of the row. You have to take exactly `k` cards.

Your score is the sum of the points of the cards you have taken.

Given the integer array `cardPoints` and the integer `k`, return the *maximum score* you can obtain.

Example 1:

```
Input: cardPoints = [1,2,3,4,5,6,1], k = 3
Output: 12
Explanation: After the first step, your score will always be 1. However, choosing the
```

Example 2:

```
Input: cardPoints = [2,2,2], k = 2
Output: 4
Explanation: Regardless of which two cards you take, your score will always be 4.
```

Example 3:

```
Input: cardPoints = [9,7,7,9,7,7,9], k = 7
Output: 55
Explanation: You have to take all the cards. Your score is the sum of points of all c
```

Constraints:

- 1 <= cardPoints.length <= 10^5
- 1 <= cardPoints[i] <= 10^4
- 1 <= k <= cardPoints.length

```
from typing import List

class Solution:
    def maxScore(self, cardPoints: List[int], k: int) -> int:
        n = len(cardPoints)
        window_size = n - k
        total_sum = sum(cardPoints)

        # Find the minimum sum subarray of size n - k
        curr_sum = sum(cardPoints[:window_size])
        min_sub_sum = curr_sum

        for i in range(window_size, n):
            curr_sum += cardPoints[i] - cardPoints[i - window_size]
            min_sub_sum = min(min_sub_sum, curr_sum)

        return total_sum - min_sub_sum
```