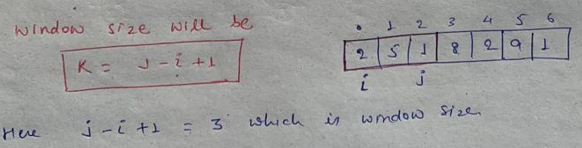
**Sliding Window**

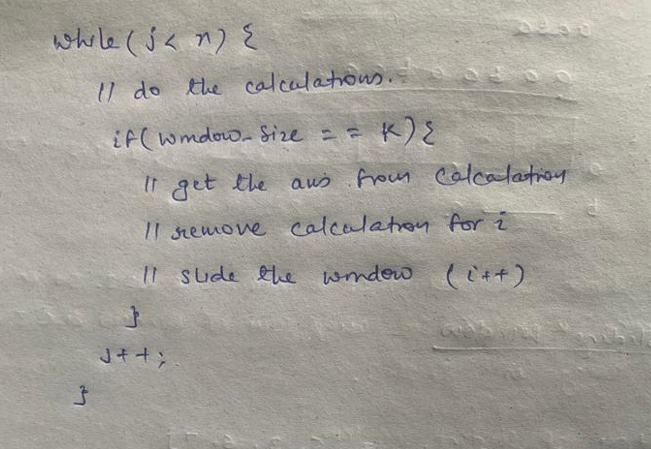
**Identification of Sliding window problems**

1. Array or String will be given
2. Sub Array or Sub String we have to find
3. Windows Size will be given or we have to find

**Int[] arr = {2,5,1,8,2,9,1}; k = 3;**



**General format for the fixed size sliding window**



**438. Find All Anagrams in a String**

Given two strings s and p, return *an array of all the start indices of*p*'s anagrams in*s. You may return the answer in **any order**.

An **Anagram** is a word or phrase formed by rearranging the letters of a different word or phrase, typically using all the original letters exactly once.

**Input:** s = "cbaebabacd", p = "abc"

**Output:** [0,6]

**Explanation:**

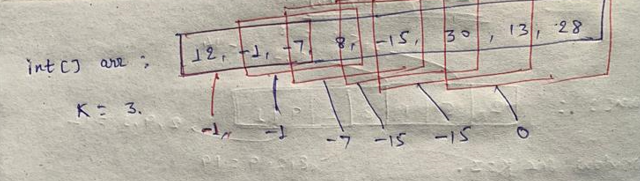
The substring with start index = 0 is "cba", which is an anagram of "abc".

The substring with start index = 6 is "bac", which is an anagram of "abc".

**First negative number in every window size of K**

**int** arr[] = { 12, -1, -7, 8, -15, 30, 13, 28 };

**int** k = 3;



Here we can see that op should be -1, -1, -7, -15, 1-5, 0

Here in the calculation step we have to just add the -ve number in the queue.

Once windows size is met then peek the first element from the queue add it to result. if arr[i] == queue.peek() then remove it from the queue itself. If queue is empty, it means there is no any -ve number in the windows size of k, so add 0 to the result.

Slide the window.

**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*.

**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.

**Maximum subarray sum of size k**

**int** arr[] = { 2, 5, 1, 8, 2, 9, 1 };

**int** k = 3;

Op: 19

**239. Sliding Window Maximum**

You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.

Return *the max sliding window*.

**Input:** nums = [1,3,-1,-3,5,3,6,7], k = 3

**Output:** [3,3,5,5,6,7]

**Explanation:**

Window position Max

--------------- -----

[1 3 -1] -3 5 3 6 7 **3**

1 [3 -1 -3] 5 3 6 7 **3**

1 3 [-1 -3 5] 3 6 7  **5**

1 3 -1 [-3 5 3] 6 7 **5**

1 3 -1 -3 [5 3 6] 7 **6**

1 3 -1 -3 5 [3 6 7] **7**

**Sol:**

We will use Deque for faster insertion/removal of the element. It will insert/delete in O(1) time.

In 1st iteration 1 will come…we will add 1 to deque.

In 2nd iteration 2 will come…. we will check q.peek < curr element, if yes then we will remove the last element from queue and add current element.

So in queue now we have only 3

In 3rd iteration -1 will come….as it is not greater than q.peek it will be added into queue

So in queue now we will have 3, -1

Now window size is hit…so we will simply peek the element from the queue and it will return 3.

Then we will remove the ith char to slide the window. So in queue we will have only -1

For 2nd window.

In 4th iteration -3 will come…..it is smaller than q.peek so it will be added into queue

So in queue we will have -1, -3 and window size is hit

We will peek the element from queue and add it to result. And remove the ith char from queue.

Do the same process for all other window size.

**567. Permutation in String**

Given two strings s1 and s2, return true*if*s2*contains a permutation of*s1*, or*false*otherwise*.

In other words, return true if one of s1's permutations is the substring of s2.

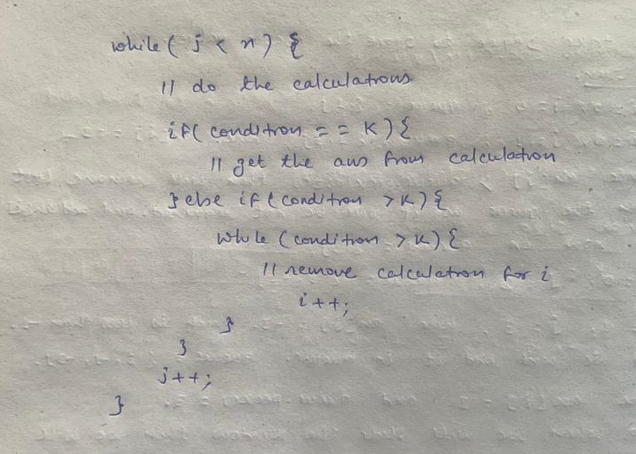
**Input:** s1 = "ab", s2 = "eidbaooo"

**Output:** true

**Explanation:** s2 contains one permutation of s1 ("ba").

**Variable Size window:**

General format



**Largest sub string of K unique character**

String str = "aabacbebebe";

**int** k = 3;

**3. Longest Substring Without Repeating Characters**

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

**Input:** s = "abcabcbb"

**Output:** 3

**Explanation:** The answer is "abc", with the length of 3.

**Sol:**

Same as previous question…here in stead of k we have to check the condition with map size

**76. Minimum Window Substring**

Given two strings s and t of lengths m and n respectively, return *the****minimum window substring****of*s*such that every character in*t*(****including duplicates****) is included in the window. If there is no such substring, return the empty string*""*.*

The testcases will be generated such that the answer is **unique**.

A **substring** is a contiguous sequence of characters within the string.

**Input:** s = "ADOBECODEBANC", t = "ABC"

**Output:** "BANC"

**Explanation:** The minimum window substring "BANC" includes 'A', 'B', and 'C' from string t.

**560. Subarray Sum Equals K**

Given an array of integers nums and an integer k, return *the total number of subarrays whose sum equals to* k.

A subarray is a contiguous **non-empty** sequence of elements within an array.

**Input:** nums = [1,1,1], k = 2

**Output:** 2