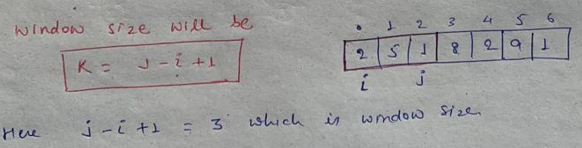
**Sliding Window**

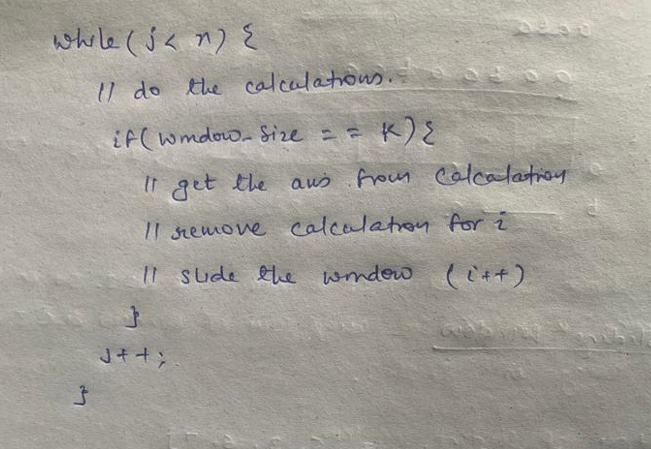
**Identification of Sliding window problems**

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;**



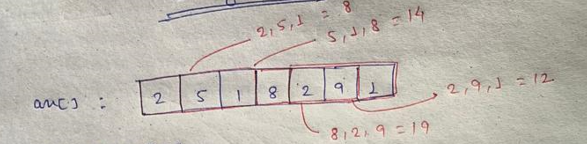
**General format for the fixed size sliding window**



1. **Maximum sub array sum of windows size K**

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

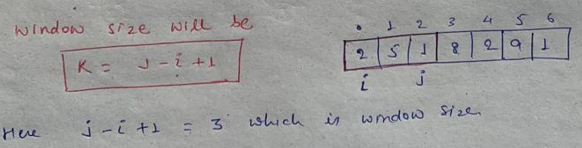
Here array and windows size is given and we have to find sub array



Here there will be many sub array of size 3. We have to have find the subarray of size 3 which have max sum.

So here in this example it should return 19

So here as per identification sliding window can be used here



**Algo:**

Here in the calculation step we have to sum up all the element

Once windows size (j-i+1) will be equal to k = 3

Then find the max between max and sum

Then remove the sum of ith index as we need to slide the window so ith index will be useless now

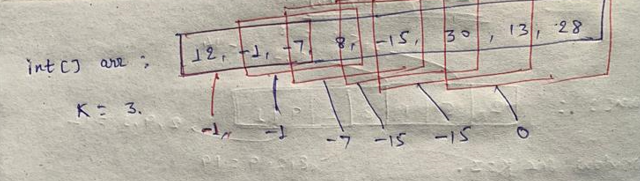
Slide the window

<https://github.com/hareramcse/Datastructure/blob/master/Algorithm/src/com/hs/slidingwindow/MaximumSubArraySumOfSizeK.java>

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

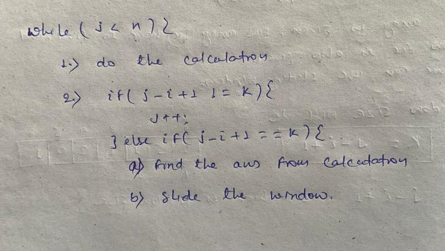
**int** arr[] = { 12, -1, -7, 8, -15, 30, 13, 28 };

**int** k = 3;



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

**Algo:**



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

Once windows size is met then fetch the first element from the queue and 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.

Slide the window.

<https://github.com/hareramcse/Datastructure/blob/master/Algorithm/src/com/hs/slidingwindow/FirstNegativeNumberInWindowsSizeK.java>

1. **Count occurrences of the anagram in window size K.**

String str = "abaacdfghaabagjjkkbaaalhfmaaab";

String pattern = "abaa";

**Anagram means……..in given String pattern should match in any order. Count of char should be same but it can be in any order.**

**Algo:**

Take the map and store the count of each char.

Map<Character, Integer > map ;

Now get the Size of the map, it will say that how many number of diff character are there in map.

While(j<str.length(){

// if arr[j] is present in the map, then decrease the count of that char

// once count is zero of that char then decrease the size of the map.

// once window size is hit

// if count == 0 then ans++

// if arr[j] is present in the map, then increment the count of that char

// once count of that char is == 1 then increment the map size;

// slide the window

}

<https://github.com/hareramcse/Datastructure/blob/master/Algorithm/src/com/hs/slidingwindow/AnagramCount.java>

1. **Max in sliding window of size K**

**int**[] nums = { 1, 3, -1, -3, 5, 3, 6, 7 };

**int** k = 3;

We have to find out maximum in every window of size k

**Algo:**

Will take a Deque to store the usable value from the arr.

I = 0; j = 0;

arr[j] will be 1;

Before putting it in queue we will check

**while** (queue.size() > 0 && queue.peekLast() < nums[j]) {

queue.removeLast();

}

queue.add(nums[j]);

Here queue is empty so it will directly added into queue so queue = {1}

I = 0; j = 1; so arr[j] = 3

Here queue size is not empty and queue.peekLast() which is 1 < arr[j] which is 3

So in this case we don’t need values which is less than the current maximum.

So we will remove other values in queue which is less than 3

After removing values from queue add arr[j] into queue. So queue = {3}

I = 0, j = 2; so arr[j] = -1

Here queue size is not empty but queue.peekLast() which is 3 not less then arr[j] which is -1

So it will directly added into queue. So now in queue = {3, -1}

Here window size is met….. I mean j – I +1 == 3

So once window size is met….

We will remove the peek element from the Deque and store it in resultant array.

Now we have to slide the window.

Before sliding the window, we need to remove the ith index from the current window and if there is any calculation is there in queue then will remove that too.

**if** (queue.peek() == nums[i]) {

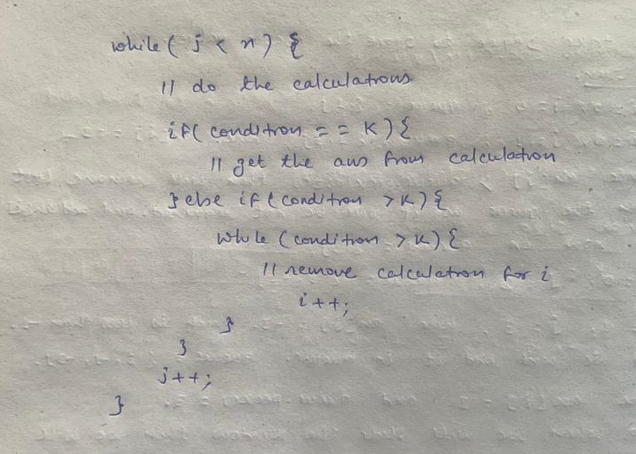
queue.removeFirst();

}

<https://github.com/hareramcse/Datastructure/blob/master/Algorithm/src/com/hs/slidingwindow/MaxInSlidingWindow.java>

**Variable Size window**

**General format**



1. **Largest sub array of sum K**

**int** arr[] = { 2, 4, 6, 1, 3, 9, 7, 5, 8 };

**int** k = 13;

Here sum is given we have to find out maximum subarray which sum is equal to 13

**Algo:**

Largest subarray means largest window size in array which sum is equal to K. so basically, we have to find out largest window size in array which sum is K

**while** (j < n) {

// find the sum of arr[j]

// if sum == k then find the max from max and j – i +1 ;

// else **while** (sum > k), substract the arr[j] from sum

// while subtracting if sum == k find the max from max and j-i+1;

// slide the window

}

<https://github.com/hareramcse/Datastructure/blob/master/Algorithm/src/com/hs/slidingwindow/MaxWindowsSizeOfSumK.java>

1. **Largest sub string of K unique character**

String str = "aabacbebebe";

**int** k = 3;

**while** (j < s.length()) {

// store the count of the char into map

// if map.size == k

// then get the max of ans and j-i+1;

**//** if map.size < k

// while map.size > k

// decrease the count of s.charAt(i) by 1

// after decreasing the count if its count becomes 0 then remove this char from map.

// i++

}

<https://github.com/hareramcse/Datastructure/blob/master/Algorithm/src/com/hs/slidingwindow/LargestSubstringOfKUniqueChar.java>

1. **Largest sub string of unique character**

String str = "aabacbebebe";

Take a Map<Character, Integer> map;

**while** (j < s.length()) {

// store the count of the char into map

// if map.size == j-i+1

// then get the max of ans and j-i+1;

**//** if map.size < j-i+1

// decrease the count of s.charAt(i) by 1

// after decreasing the count if its count becomes 0 then remove this char from map.

// i++

}

<https://github.com/hareramcse/Datastructure/blob/master/Algorithm/src/com/hs/slidingwindow/LargestSubstringOfUniqueChar.java>

1. **Minimum window substring**

String str = "timetopractice";

String t = "toc";

Minimum window substring means………….. in str find the substring in which t string should present in same order and in atleast same quantity

OP : toprac

**Algo:**

Take a Map<Character, Integer> map;

Put the char and its count of t String into this map.

Get the size of the map and store it in count variable.

**while** (j < s.length()) {

// char from the s String and check if it is present in the map

// if it is present then decrement the count of that char

// once count == 0 then decrement the count

// if count == 0 it means in map is empty and all the char in

// t String is present in s String. So here we have result

// if windows size j – i + 1 < min

// then store the substring(i, j+1) into res

// and then min = j -i + 1;

// while ( count == 0 && i <= j) it means if there is extra char which can be removed to make the window size smaller

// to remove the char we check the string starting from the ith index

// get the ith char from the s String and check in the map

// if the char is present in the map

// if char count == 0 then increment the count

// and increment the count of the char into map

// i++

// if count == 0 && j – i + 1 < min

// then find the substring(i, j+1) and store the result in res

// min = j - i + 1;

}