Todays Content

1. 1st non repeating char for every prefix substring
2. 1st negative elements in all subarrays for size = k

C++

queue<type> que;

que.push( )	Insert x at rear/back end of enque
que.pop()	delete ele at front end
que.front()	Return ele at front end
que.back()	Return ele at back end.
que.size()	Return no: of ele in queue.

1Q Given input of stream of characters

For every new input char print 1st non-repeating char for entire data

Note: If no non repeating character print #

Stream1	a	b	c	c	a	e	b	e	a	g
Output:	a	a	a	a	b	b	e	#	#	g

	✓	✓	✓		✓			✓	✓	✓	✓			
Stream 2	a	b	c	~~b~~	d	~~c~~	~~a~~	e	g	h	~~g~~	~~h~~	~~d~~	~~e~~
Output	a	a	a	a	a	a	d	d	d	d	d	d	e	#

1st

Note: # All possible ans char

~~a~~ ~~b~~ ~~c~~ ~~d~~ e g h

Operations : Queue

Insert back

Delete front

Access front

Hashmap

a : 2 # If freq > 1 = repeatin.

b : 2 # If freq > 1 = repeatin.

c : 2 # If freq > 1 = repeatin.

d : 2 # If freq > 1 = repeatin.

e : 2 # If freq > 1 = repeatin.

g : 2 # If freq > 1 = repeatin.

h : 2 # If freq > 1 = repeatin.

#Pseudo Code:

```
void 1nrRepeating(string s){   TC: O(N)   SC: O(N)
    queue<char> q;
    unordered_map<char, int> um;          Total Iterations = Outer + Inner
    for(int i=0; i<s.size(); i++){                             N    +   N   = 2N
        #New char is s[i];                                               1 pop = 1 iter
        um[s[i]]++;                                                      N push oper
        if(um[s[i]]==1){  #non repeating                                 N pop oper
            q.push(s[i]);
        }

        while(q.size()>0 && um[q.front()] > 1){
            q.pop();
        }

        if(q.size()==0){
            print("#");
        }
        else{
            print(q.front());
        }
    }
}
```

|

28 First -ve element in all subarrays of size = k.

Note: For a subarray if no -ve element print 0

	0	1	2	3	4	5	6
arr[] = {	1	-1	-2	3	-4	5	2 } k = 4

#Subarrays	Output:
[0 3]	-1
[1 4]	-1
[2 5]	-2
[3 6]	-4

Idea1: for every subarray of len k:

Iterate & calculate 1st -ve element.

TC: $O(N-K+1) * O(K)$ SC: $O(1)$

If $k \approx N/2$

$O(N - N/2 + 1) * O(N/2) \approx O(N^2)$

Idea2 : Sliding Window ?

Dry Run:

```
             0   1   2   3    4   5   6   7   8    9
arr[] = {    1  -1  -2   3  [-4   5   2  -6   2]   8 }   k = 5
```

	Del arr[]	Add arr[]	All possible ans	Output
[0 4]			-1 -2 -4	-1
[1 5]	arr[0] = 1 *	arr[5] = 5 *	-1 -2 -4	-1
[2 6]	arr[1] = -1	arr[6] = 2 *	~~-1~~ -2 -4	-2
[3 7]	arr[2] = -2	arr[7] = -6 ✓	~~-2~~ -4 -6	-4
[4 8]	arr[3] = 3 *	arr[8] = 2 *	-4 -6	-4
[5 9]	arr[4] = -4	arr[9] = 8 *	~~-4~~ -6	-6

Container : queue

Add back

Delete front

Ans front

```
void firstNegative(vector<int> arr, int k){ TC: O(N)  SC: O(K)

    queue<int> q;
    for( int i=0; i< k; i++ ){
        if( arr[i] < 0 ){
            q.push(arr[i]);
        }
    }
    print( q.front());

    int s=1, e=k;
    while( e < arr.size() ){
        # remove arr[s-1], add arr[e];
        if( arr[s-1] == q.front() ){
            q.pop();
        }

        if ( arr[e] < 0 ){
            q.push(arr[e]);
        }

        if( q.size() ==0 ){
            print(0);
        }
        else{
            print( q.front());
        }
    }
}
```