Date _____ Page _____

Day - 196

Sliding Window - 4

* Length of longest subarray with atmost k frequency:

1	2	3	1	2	3	3	4	1	2

k = 2

⇉ Ex:

1	2	3	1	2	3

→ size = 6

1	2	3	3	4	1	2

→ size = 7

⇉ Basic approach will be same.

⇉ Now, we will use the concept of sliding window protocol.

⇉ Start from taking as first element in the window & check the frequency.

⇉ If the frequency is less than k then increase the window size otherwise decrease the window size from starting.

Code

```
unordered_map<int, int> count;
int start=0, end=0, len=0;
while(end<n){
    count[nums[end]]++;
```

Date _____ Page _____

~~len = max~~

~~while~~

```
while( count[nums[end]] > k){
        count[nums[start++]]--;
}
len = max(len, end - start + 1);
end++;
}
return len;
```

* Count subarray where max element appear at least k times

1	2	3	2	3	1	2	3	3	2	k = 2

⇉ Ex:

1	2	3	2	3

⇉ max element → 3 → 2 times.

⇉ We have to count all these types of subarrays.

⇉ We will SWP approach.

⇉ So, first we get a window that satisfy the conditions. Now, if we add other elements in the window the result will be same.

⇉ So, we can directly add all these subarrays

⇉ After that, we will decrease the element from starting from the window, then again check for conditions.

Date _____ Page _____

⇉ And we will repeat this process.

⇉ If the window is not valid then we increase the window.

⇉ So, first we increase the window size until count == k.

⇉ Then decrease the window size.

Code

```
int start = 0, end = 0, maxEle, count = 0,
total = 0;
while( end < n){
    if( nums[end] == maxEle)
        count ++;
    while ( count == k){
        total += n - end;
        if ( nums [start] == maxEle)
            count --;
        start ++;
    }
    end ++;
}
return total;
```

Date _____ Page _____

* Subarray with k different Integer:

| 1 | 2 | 1 | 2 | 3 |

k = 2

⇉ We have to return good sub arrays.

⇉ Good subarrays are those arrays that have k different integer.

Ex: | 2 | 1 | 2 | ✓

1 → 1

2 → 2

| 1 | 2 | 3 | →

1 → 1 (more than 2)

2 → 1

3 → 1

⇉ Here, we directly ~~don't~~ can't use the sWP approach. Using this, can result in missing of some answers.

⇉ So, we can redefine our problem statement as Total no. of subarray, atleast k diff. int.

⇉ Atleast (2): sub(2) + sub(3) + sub(4) + sub(5)

Atleast (3): sub(3) + sub(4) + sub(5)

= sub(2)

⇉ So, by using this approach, we can find out exactly ~~=~~ (k).

⇉ For finding atleast, we will use the previous question approach.