Sliding window & Two pointers.

Sliding window.

1. constant window size

w2

arr = [1, 2, 3, 4 , 5, 6, 7, 8 9] . k=4

0 w1 k-1

Pseudo code.

```
l=0  r=0
for  r : 0 → k-1  do
        sum+= arr[r] ;
max = sum;
for  r : k → n-1  do
        sum -= arr[l] ;
        l++;
        sum += arr[r] ;
    max = max( sum, max)
```

2. Longest substring / subarray where <condition>.

Eg: Longest subarray with sum ≤ k.

arr = [2, 5, 1, 7, 10]   k = 14.

Approaches:

i) Brute Force. → Generate all the subarrays.

[2], [2, 5], [2, 5, 1], [2, 5, 1, 7], [2, 5, 1, 7, 10]

[5], [5, 1], [5, 1, 7], [5, 1, 7, 10]

[1], [1, 7], [1, 7, 10]

[7], [7, 10]

[10]

```
for (i=0 → n-1)
{
    for (j=i → n-1)
    {
        for (k=i → j)
        {
            cout<< arr[k]
        }
    }
}
```

ii) Better Approach. (Longest subarray with sum ≤k)

```
        l,r
         ↓
arr = [2, 5, 1, 7, 10]        k=14
```

maxlen=1

i) Expand → r

ii) Shrink → l.

We start off with window of size 1.

sum= 2 ≤ k (14)

maxlen=2

```
arr = [2, 5, 1, 7, 10]
       ↑  ↑
       l  r
```

Sum= 2+5 =7 ≤k (14)

maxlen=3

```
arr = [2, 5, 1, 7, 10]
       ↑     ↑
       l     r
```

Sum = 2+5+1 = 8 ≤k (14)

```
arr = [2, 5, 1, 7, 10]
       ↑        ↑
       l        r
```

sum = 2+5+1+7 = 15 > k. (so, shrink)

maxlen=3

```
arr = [2, 5, 1, 7, 10]
          ↑     ↑
          l     r
```

sum = 5+1+7 = 13 ≤ k

```
l=0   r=0   maxlen=0   sum=0

while (r < n)
{
    sum += arr[r];
    while(sum > k)
    {
        sum -= arr[l];
        l++;
    }

    if ( sum ≤ k)
        maxlen = max(maxlen, r-l+1)
        // Here store the Indices
        // if need to print
        // the subarray.

    r = r+1;
}
print(maxlen);
```

Time Complexity

$$O(\underset{l}{\uparrow N} + \underset{r}{\uparrow N}) = O(2N)$$

l → worst case

r ← always from 0 to N.

Space Complexity

Here $O(1)$ → constant

but depends on the problem.

(iii) Optimal approach.

This is specifically for this problem. the problem is asking to find out the longest subarray "Length" whose sum ≤ k not the subarray itself, so we once reached the length of 3, I just don't allow the window size to shrunk below 3 and I would increase it to 4 and check if it is sum ≤ k if move 'l' by one position ahead and again check until r < n.

INSHORT: INSTEAD of the While inside change it to if

Optimal :

```
l=0   r=      maxLen=0   sum=0

while (r < n)
{
      sum += arr[r];
~~While~~  if (sum > k)
      {
         sum -= arr[l];
         l++;
      }

      if ( sum <= k)
           maxLen = max(maxLen, r-l+1)
           // Here store the Indices
           // if need to print
           // the subarray.

      r = r+1;
}
print (maxLen);
```

3. No. of Subarrays where condition. (Difficult).
(Solved using the pattern 2)

Eg: No. of subarrays where sum $= k$.

In this case we don't know when to shrink and when to Expand

So breakdown the problem into 2.

(no. of subarrays when sum $\leq k$) $= x$
(no. of subarrays when sum $<= (k-1)$) $= y$

Ans: $x - y$.

4) Shortest / Minimum Window < condition.

- Find a valid window satisfying gn condition
- Then keep on shrinking until its valid.