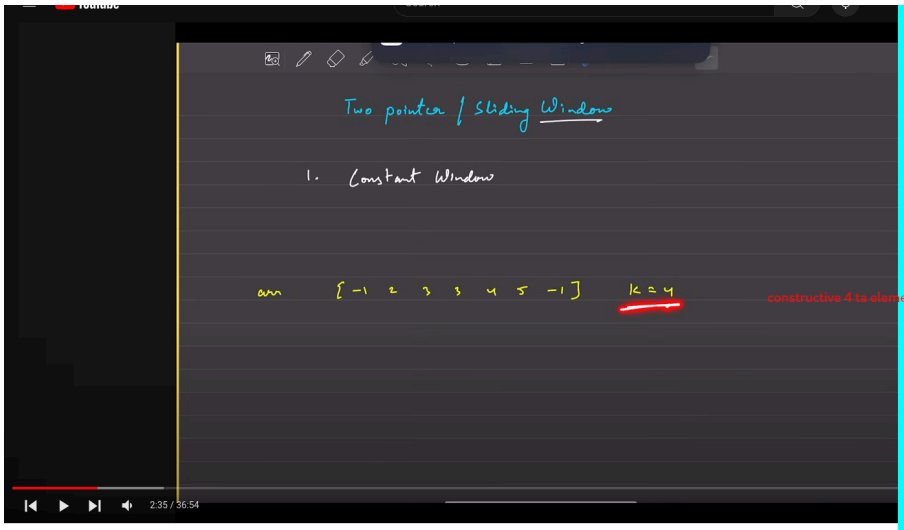
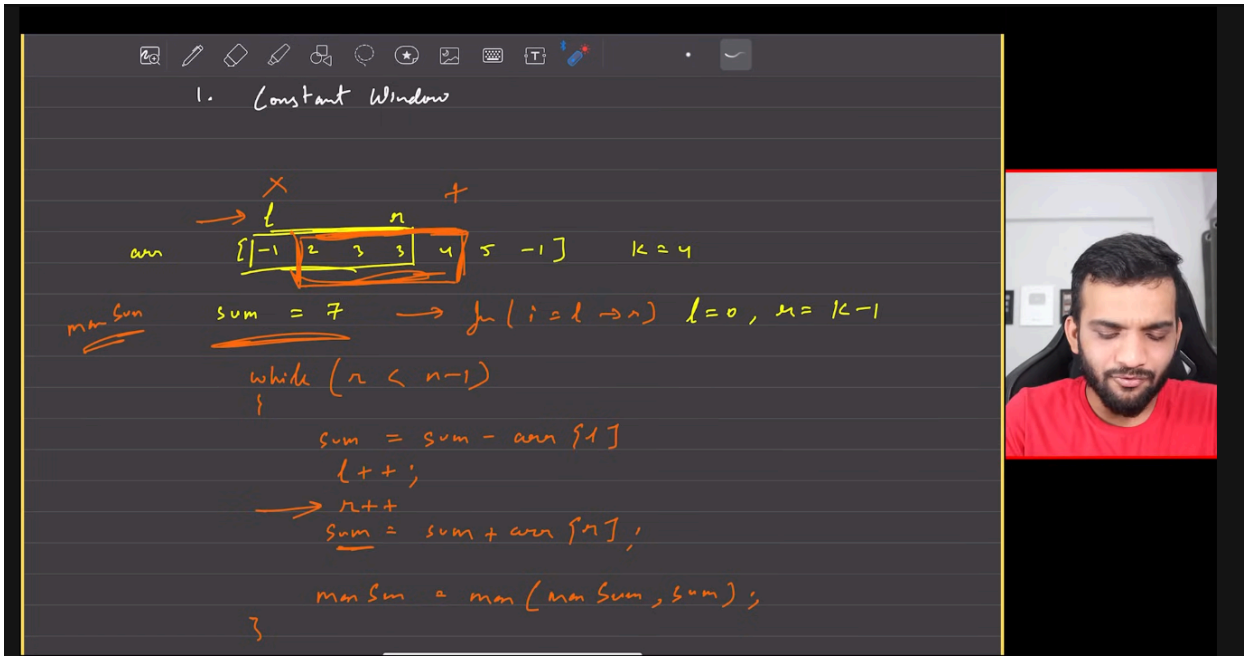


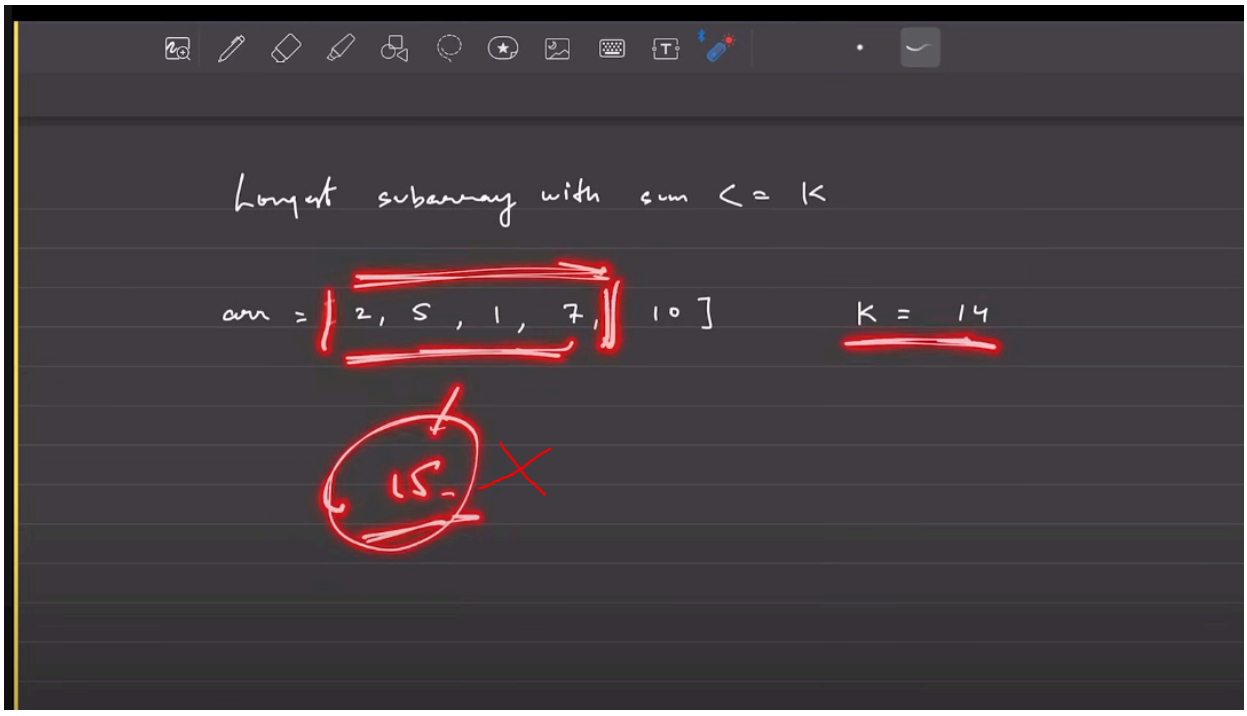
constructive 4 ta element niya max sum ber kori

L1. Introduction to Sliding Window and 2 Pointers | Templates | Patterns



Longest subarray with sum <= K

arr = [2, 5, 1, 7, 10] K = 14



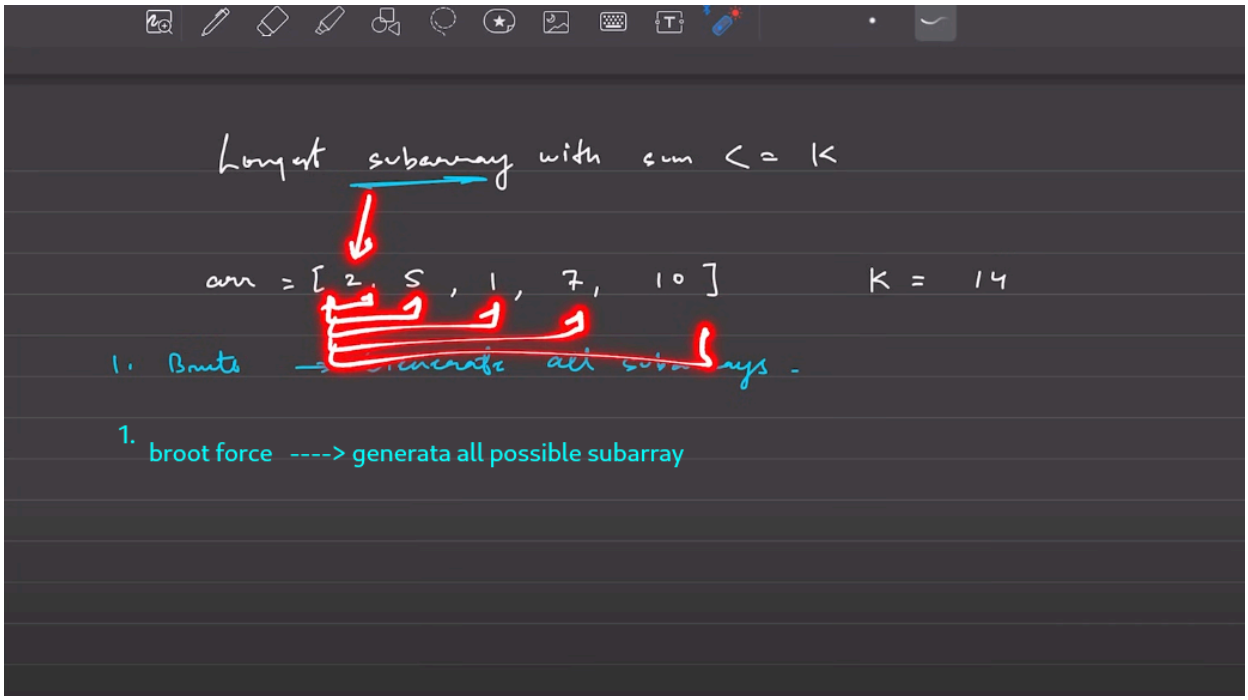
Longest subarray with sum <= K

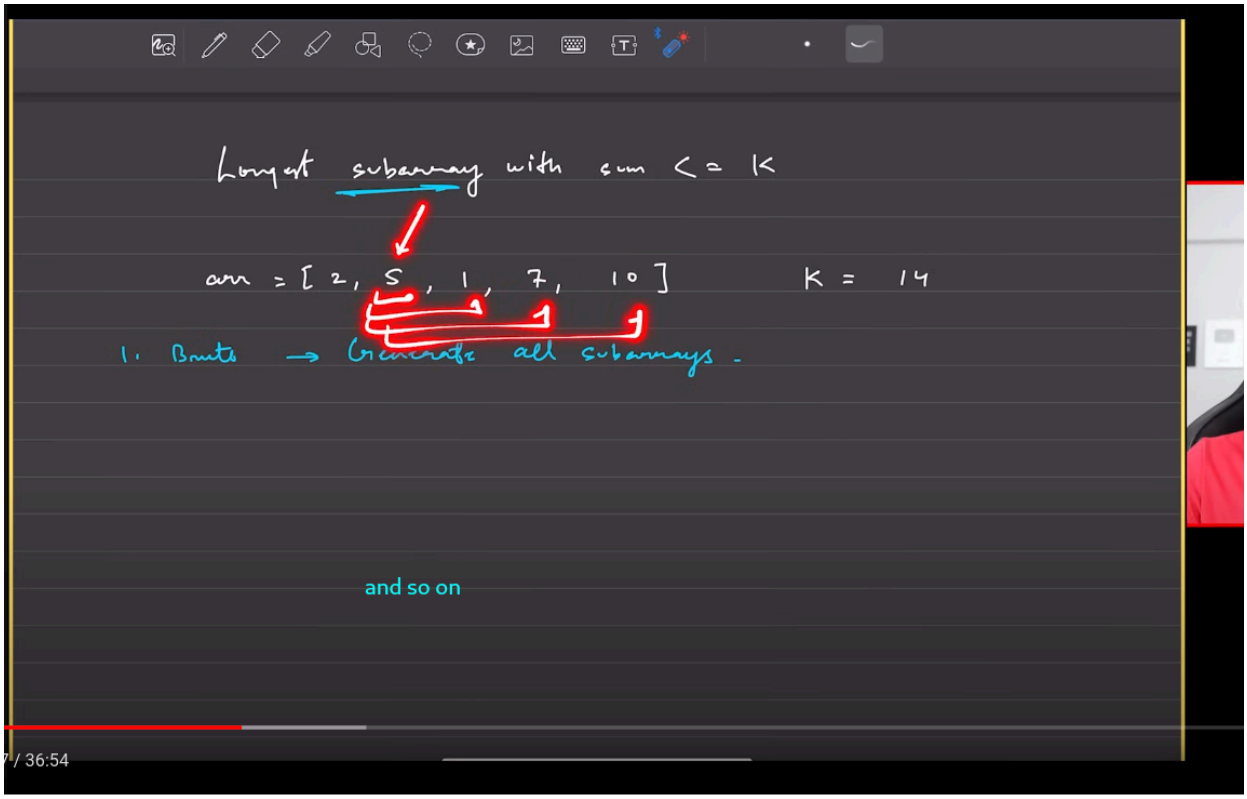
arr = [2, 5, 1, 7, 10] K = 14

13 <= 14

Two pointer / sliding window

1. Constant Window
2. Longest subarray / substring where <condition>
   - → Brute
   - → Better
   - → Optimal


Longest subarray with sum <= K
arr = [2, 5, 1, 7, 10]
K = 14
1. Brute → Generate all subarrays.
1. broot force ----> generata all possible subarray


Longest subarray with sum <= K
arr = [2, 5, 1, 7, 10]
K = 14
1. Brute → Generate all subarrays.
and so on
/ 36:54

Longest subarray with sum <= K

i

arr = [2, 5, 1, 7, 10] K = 14

1. Brute — find all subarrays.

for (i = 0 ⟶ n-1)

for (i = 0 ⟶ n-1)

for (j = i ⟶ n-1)

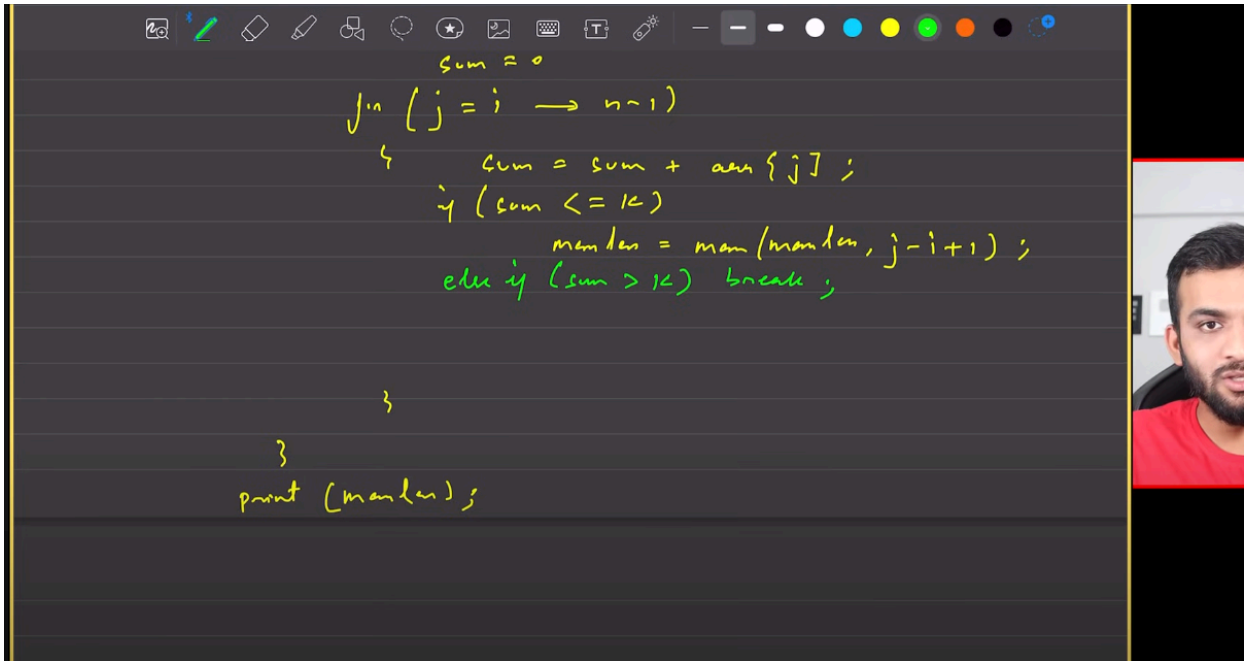
{

}

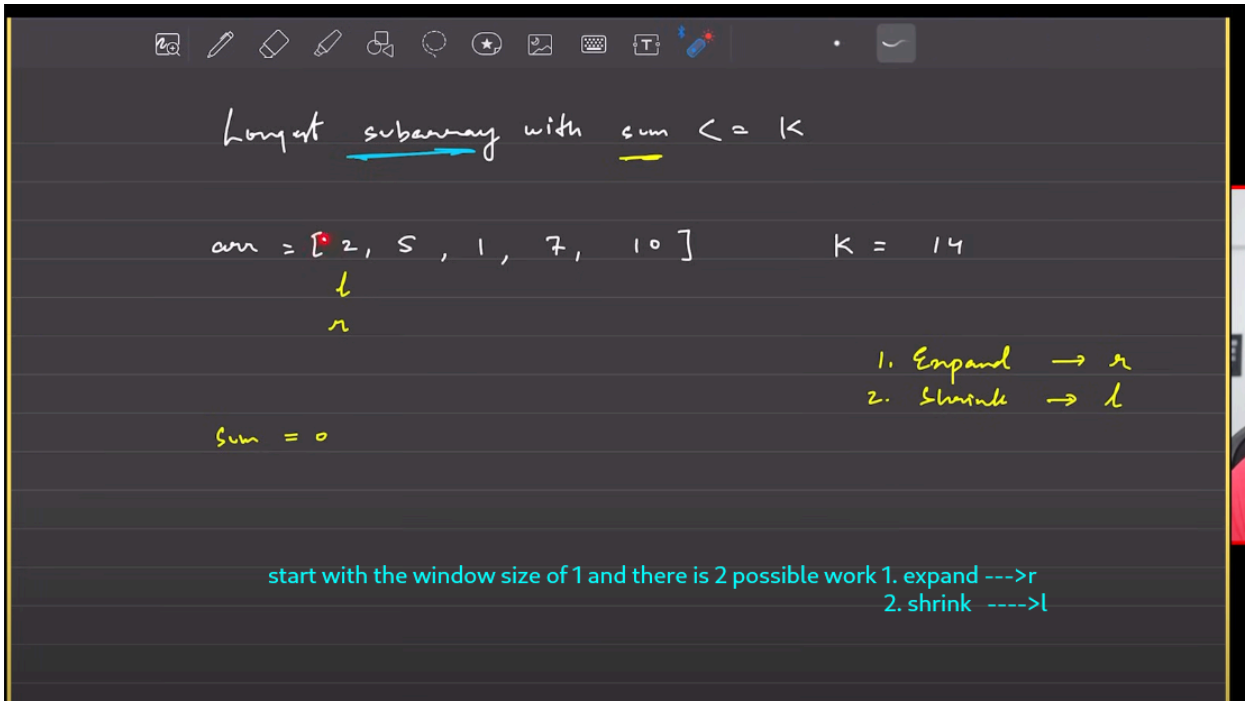
1. Brute → Generate all subarrays.

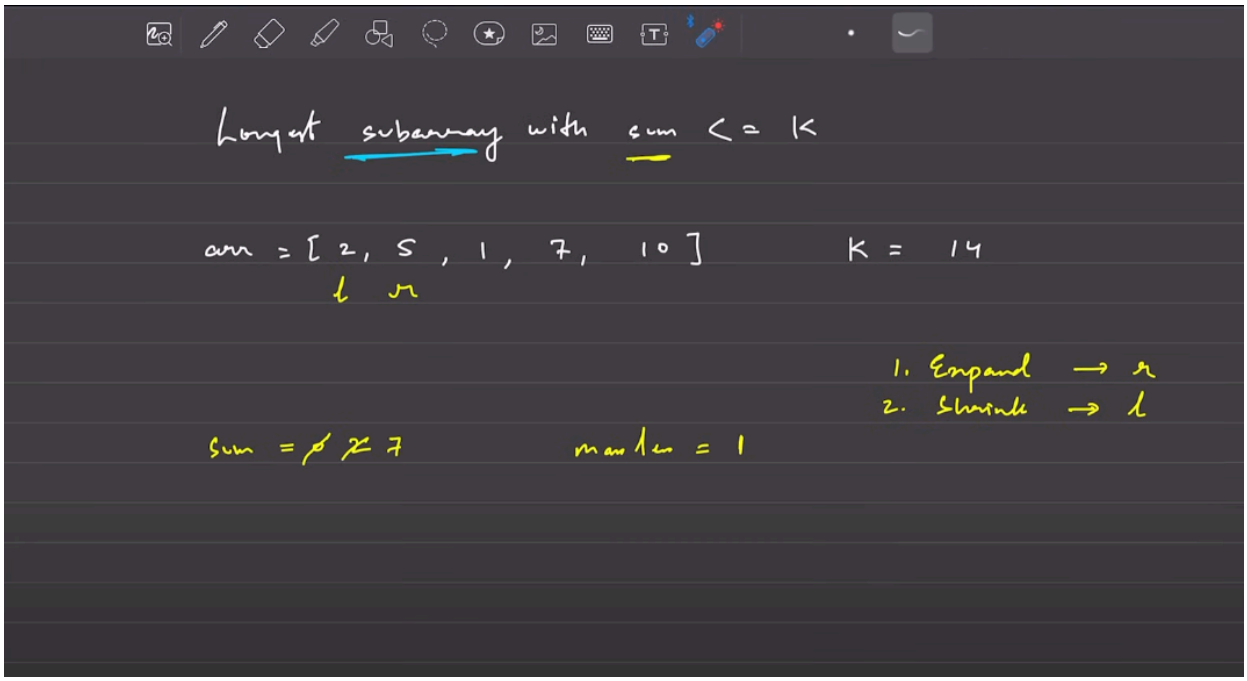
TC → $O(N^2)$
SC → $O(1)$

```
maxLen = 0
for (i = 0 ⟶ n-1)
{
    sum = 0
    for (j = i ⟶ n-1)
    {
        sum = sum + arr[j];
        if (sum <= k)
            maxLen = max(maxLen, j-i+1);
        else if (sum > k) break;
    }
}
```



```
        sum = 0
        for (j = i ⟶ n-1)
        {
            sum = sum + arr[j];
            if (sum <= k)
                maxLen = max(maxLen, j-i+1);
            else if (sum > k) break;
        }
    }
    print (maxLen);
```


Longest subarray with sum <= K
arr = [2, 5, 1, 7, 10]
K = 14
l
r
1. Expand → r
2. Shrink → l
sum = 0
start with the window size of 1 and there is 2 possible work 1. expand --->r
2. shrink ---->l


Longest subarray with sum <= K
arr = [2, 5, 1, 7, 10]
K = 14
l r
1. Expand → r
2. Shrink → l
sum = 7
maxLen = 1

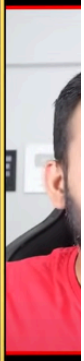
Longest subarray with sum <= K

arr = [2, 5, 1, 7, 10] K = 14

l r

1. Expand → r
2. Shrink → l

sum = 8 ✓ maxLen = 2 ✓



Longest subarray with sum <= K

arr = [2, 5, 1, 7, 10] K = 14

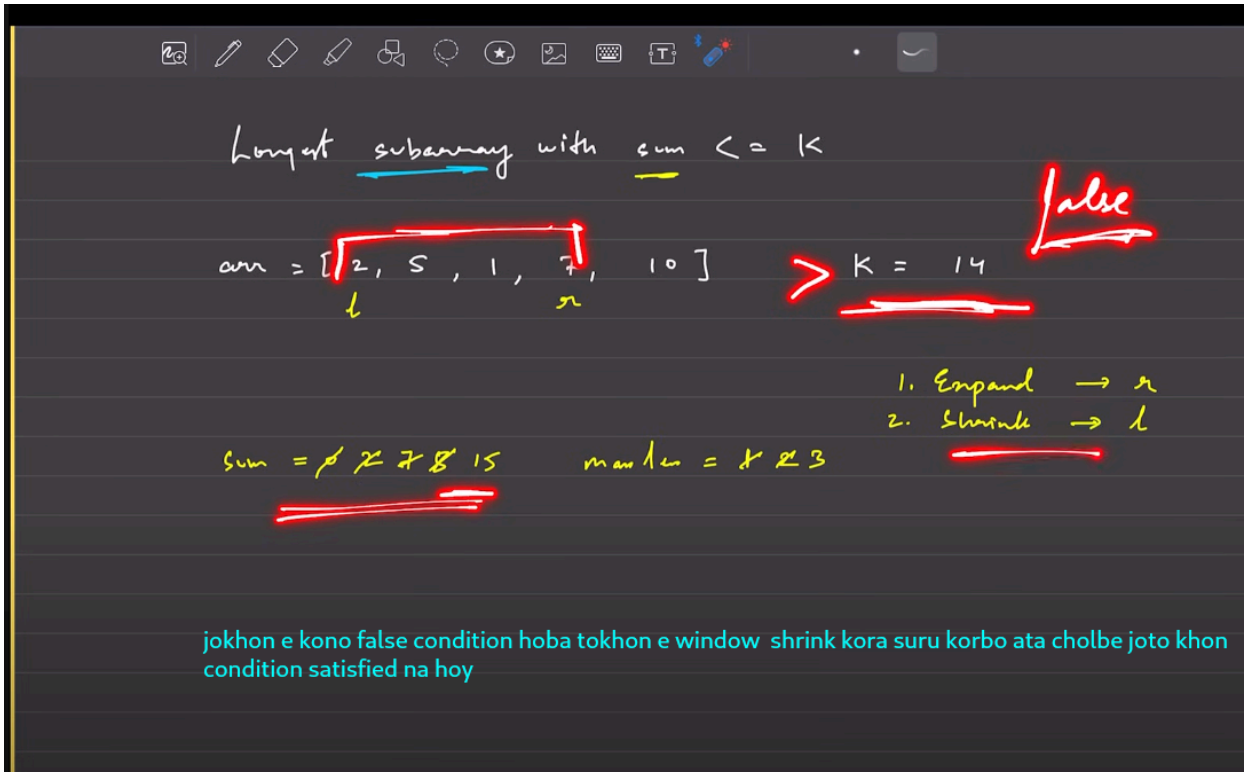
l r

1. Expand → r
2. Shrink → l

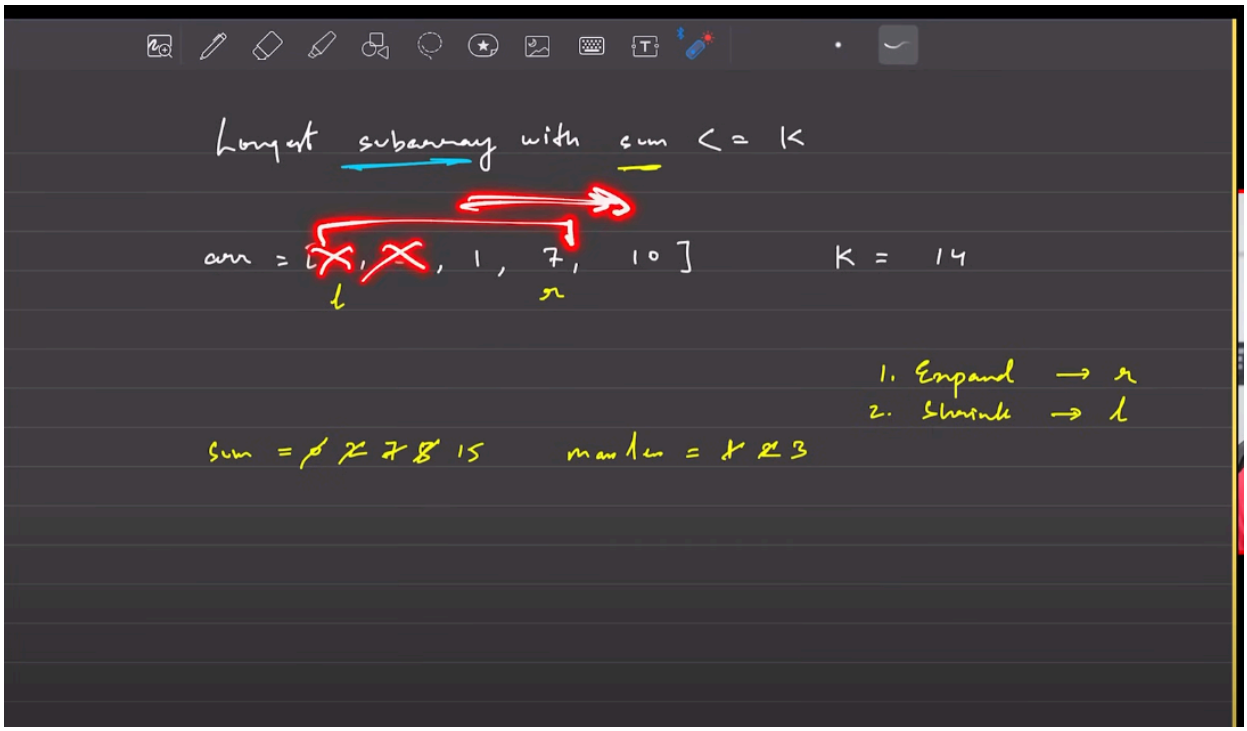
r - l + 1

sum = 8 maxLen = 3

at any moment the length of the windows is (r-l+1)



jokhon e kono false condition hoba tokhon e window shrink kora suru korbo ata cholbe joto khon condition satisfied na hoy



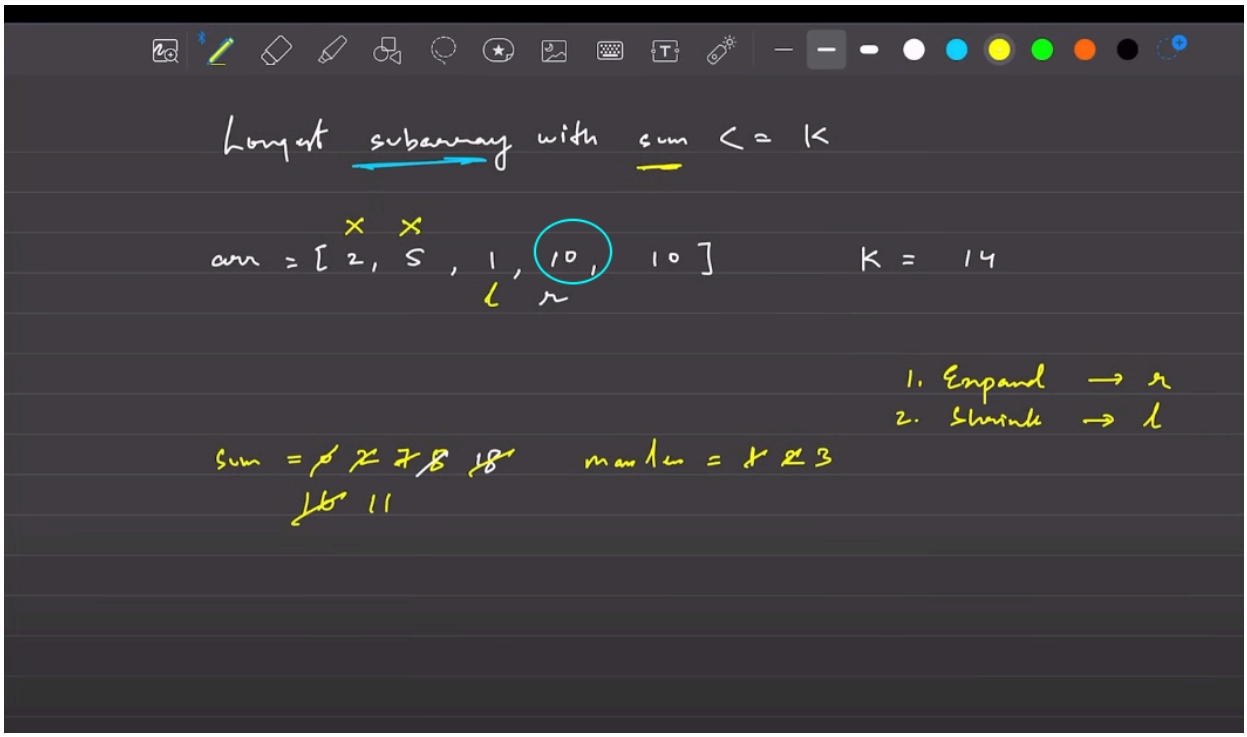
Longest subarray with sum <= K

arr = [2, 5, 1, 7, 10] K = 14

l r

1. Expand → r
2. Shrink → l

sum = 0 2 7 8 15 → 13 max len = 4 2 3



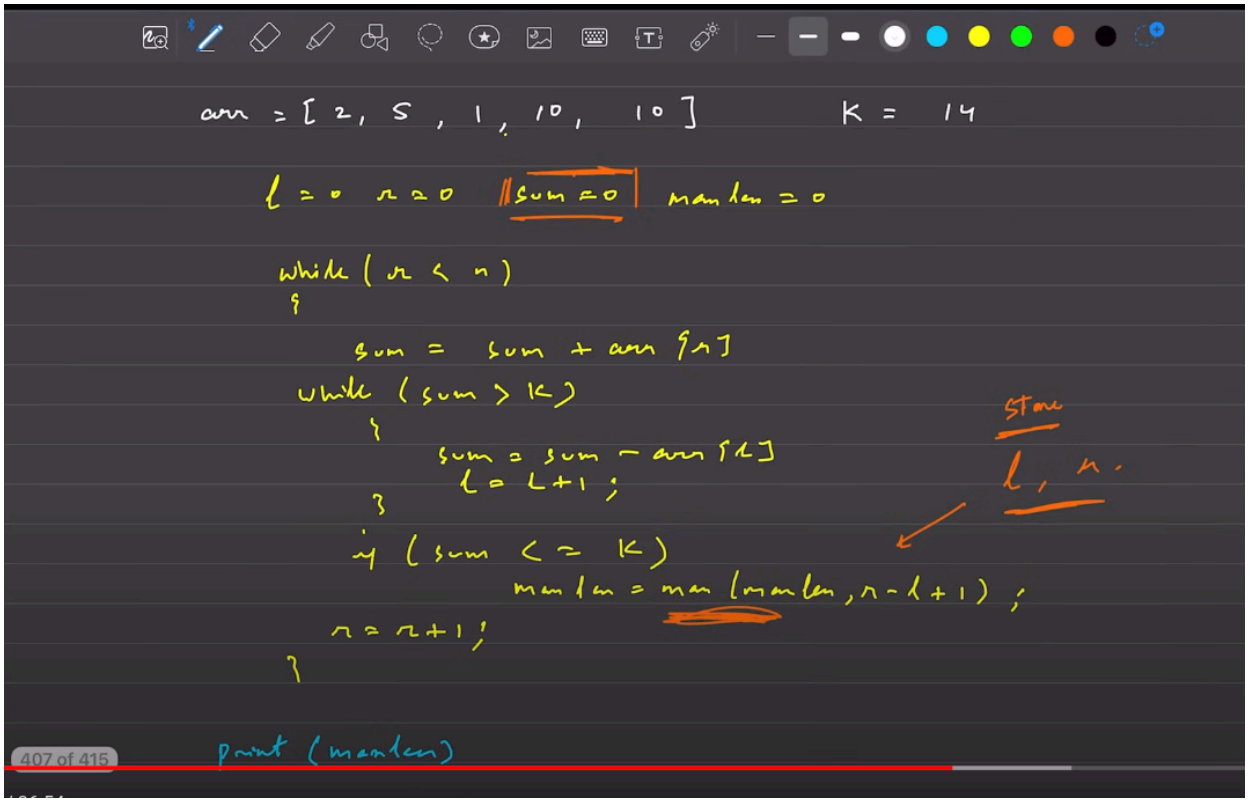
Longest subarray with sum <= K

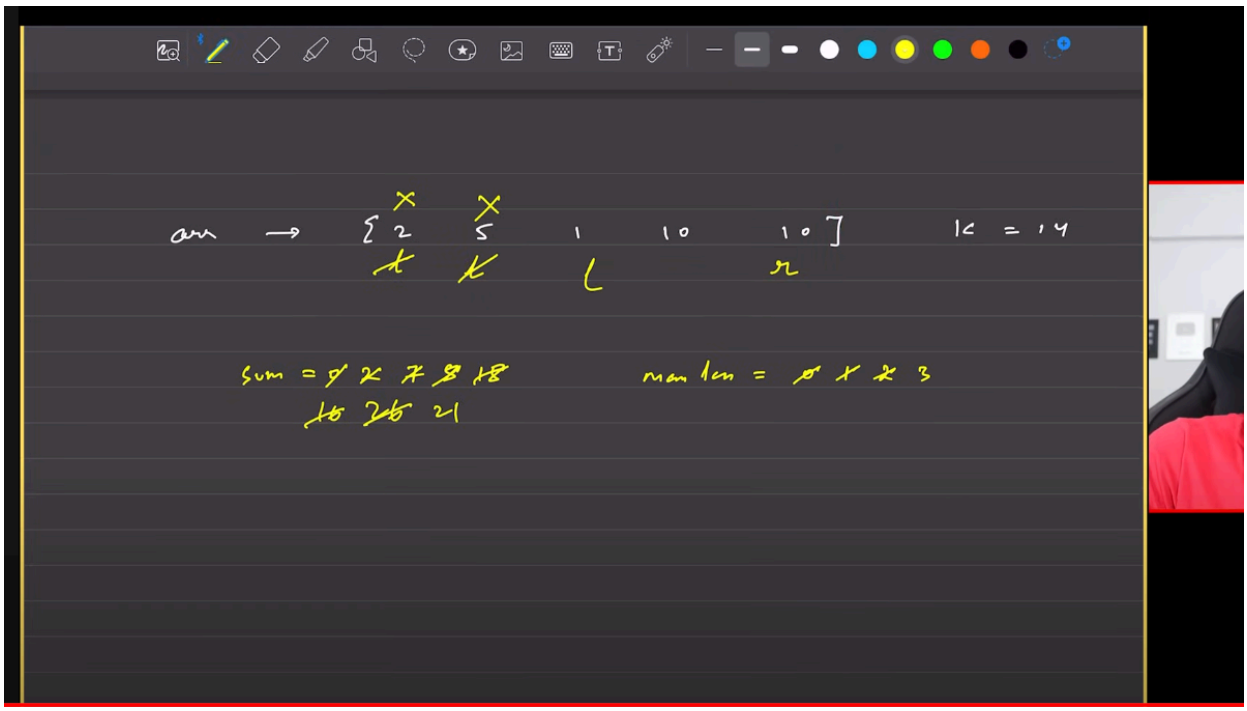
arr = [2, 5, 1, 10, 10] K = 14

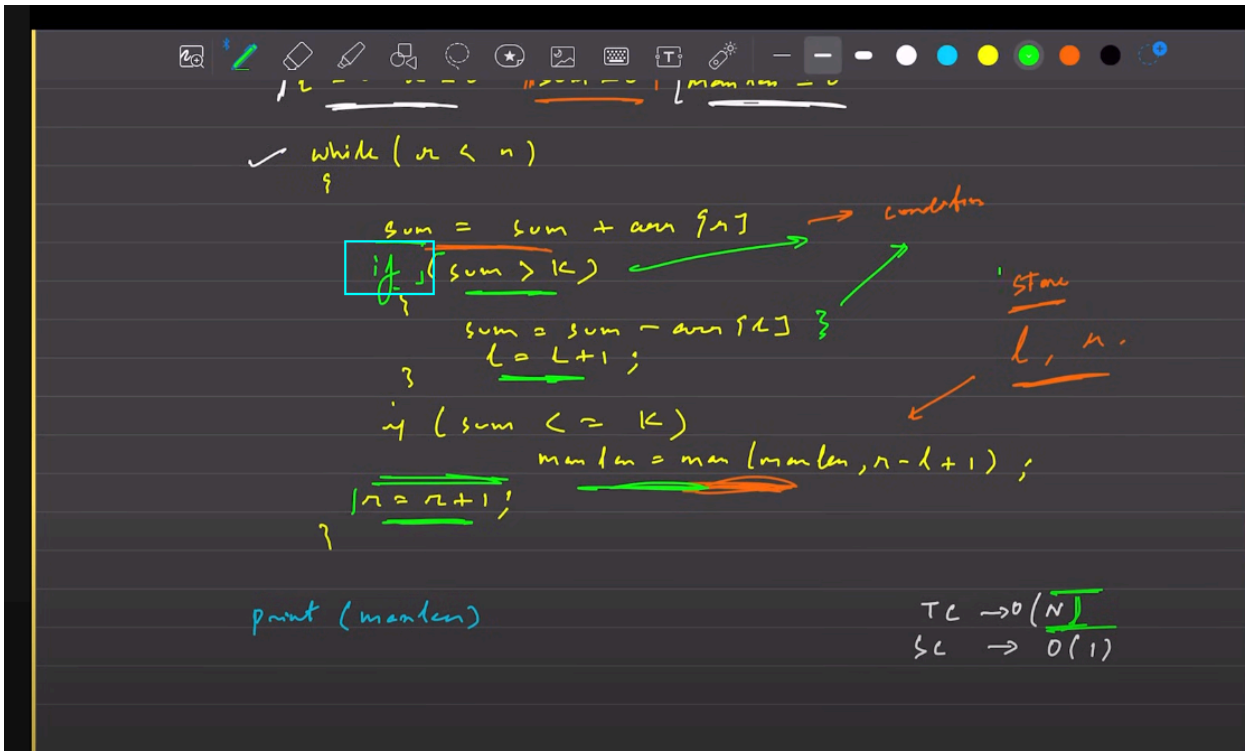
l r

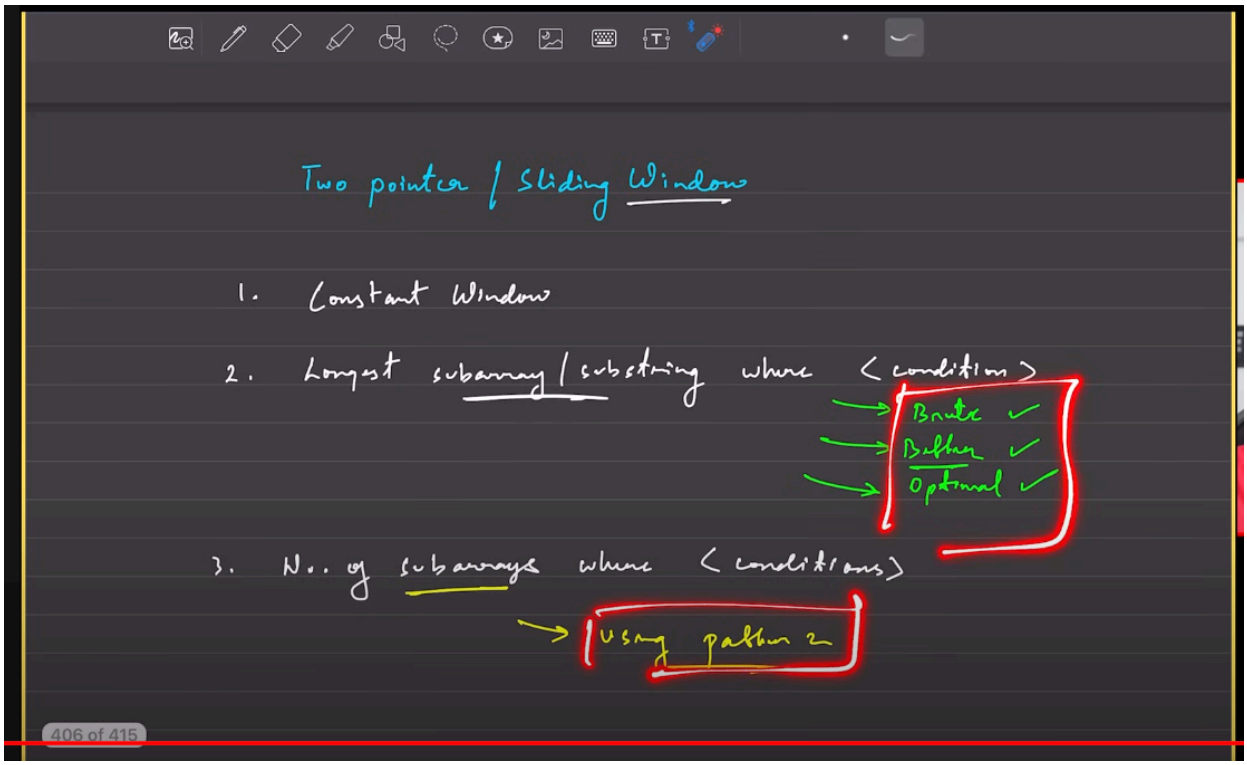
1. Expand → r
2. Shrink → l

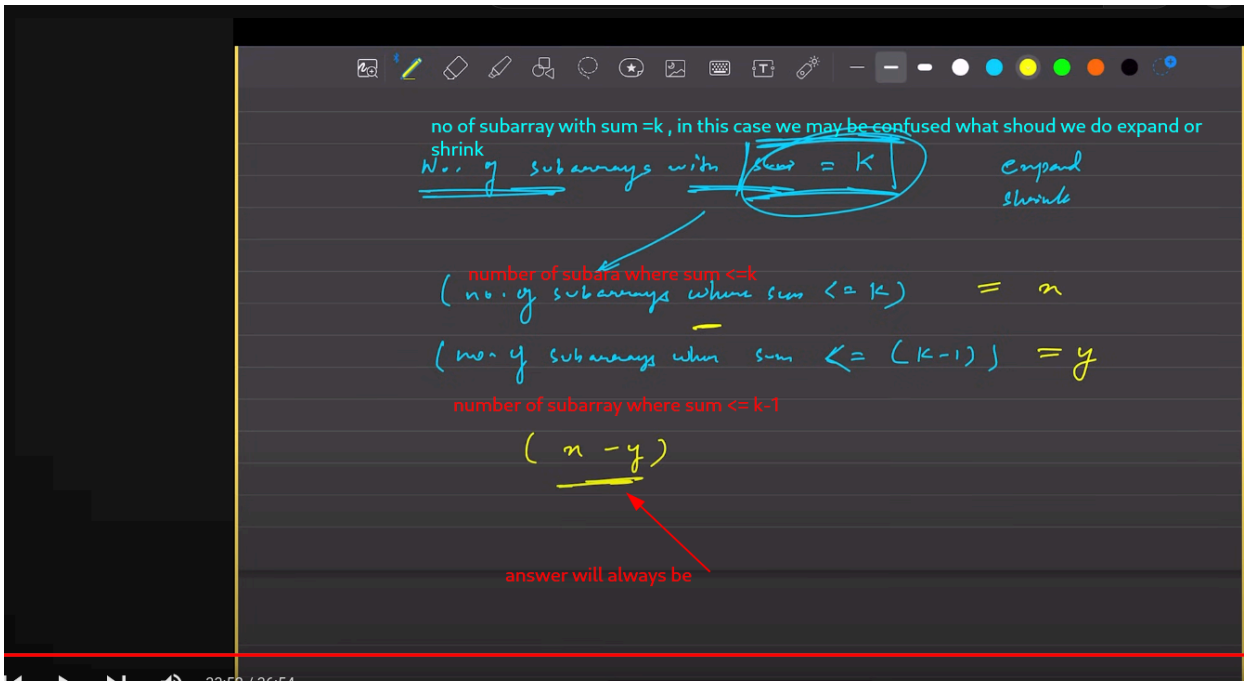
sum = 0 2 7 8 18 → 11 max len = 4 2 3

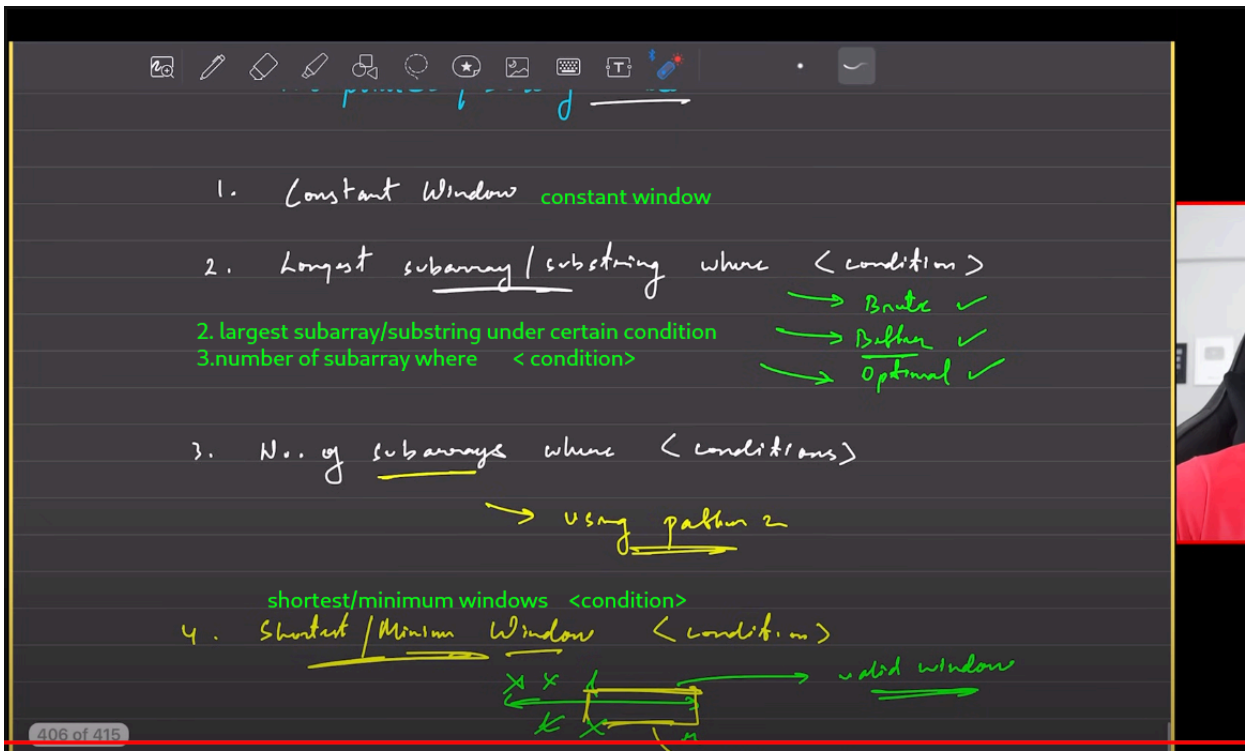

arr = [2, 5, 1, 10, 10]
K = 14
l = 0 r = 0 sum = 0 maxlen = 0
while (r < n)
{
sum = sum + arr [r]
while (sum > K)
{
sum = sum - arr [l]
l = l + 1 ;
}
if (sum <= K)
maxlen = max (maxlen, r - l + 1) ;
r = r + 1 ;
}
print (maxlen)
Store
l, r.
407 of 415


arr → [2 5 1 10 10]
K = 14
sum = 0 2 7 18
maxlen = 0 2 3


while (r < n)
sum = sum + arr[r]
if (sum > k)
condition
sum = sum - arr[l] }
l = l+1 ;
store
l, r.
if (sum <= k)
maxlen = max(maxlen, r-l+1) ;
r = r+1 ;
print (maxlen)
TC → O(N)
SC → O(1)


Two pointer / Sliding Window
1. Constant Window
2. Longest subarray / substring where <condition>
Brute
Better
Optimal
3. No. of subarrays where <conditions>
Using pattern 2


no of subarray with sum =k , in this case we may be confused what shoud we do expand or shrink
No. of subarrays with sum = K
expand
shrink
number of subara where sum <=k
(no. of subarrays where sum <= K) = x
(no of subarrays when sum <= (K-1)) = y
number of subarray where sum <= k-1
(x - y)
answer will always be


1. Constant Window constant window
2. Longest subarray / substring where <condition>
Brute
Better
Optimal
2. largest subarray/substring under certain condition
3.number of subarray where < condition>
3. No. of subarrays where <conditions>
using pattern 2
shortest/minimum windows <condition>
4. Shortest / Minimum Window <conditions>
valid window
406 of 415