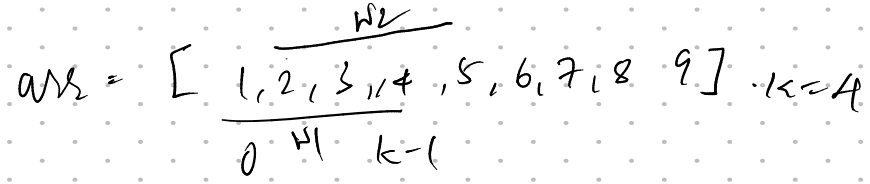
Sliding window & Two pointers.

Sliding window.

1. Constant window size



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 $\leq k$.

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]
    }
  }
}
```

17) Sliding Window Approach. (Longest subarray with sum ≤ k)

i) Expand → r
ii) Shrink → l.

maxlen = 1

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

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

$O(N+N) = O(2N)$

L → worst case

R → always from 1 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 $\leq k$ not the subarray itself, so we once reached the length of 3, i just don't allow the window size to shrink below 3 and I would increase it to 4 and check if it is sum $\leq k$ if move 'l' by one position ahead and again check until $r < n$.

INSIGHT: INSTEAD of the while inside change it to if

Optimal:

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

while(r < n)
{
    sum += arr[r];
    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);
```

White

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.

Cnt. of subarrays when sum <= k) = x

Cnt. of subarrays when sum <= (k-1)) = y

Ans: x - y.

4) Shortest / Minimum Window < condition.

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