Two pointer / Sliding Window

Two pointer / Sliding Window

1. Constant Window

arr [-1 2 3 3 4 5 -1] k = 4

TUF

Two pointer / Sliding Window

1. Constant Window

0 1 2 3

arr [-1 2 3 3 4 5 -1] k = 4

TUF

Two pointer / Sliding Window

1. Constant Window

k-1

0 1 2 3

arr [-1 2 3 3 4 5 -1] k = 4

Two pointer / Sliding Window

1. Constant Window

$$\begin{array}{lcccccccl} & l & & & r & & & & \\ arr & [-1 & 2 & 3 & 3 & 4 & 5 & -1] & \quad k=4 \end{array}$$

sum = 7

$l = 0, \; r = k-1$

Two pointer / Sliding Window

1. Constant Window

arr [-1 2 3 3 4 5 -1] $k = 4$

l r

sum = 7

$l = 0$, $r = k-1$

1. Constant Window

arr [-1 | 2 3 3 | 4 5 -1] $k = 4$

l r

sum = 7

$l = 0$, $r = k-1$

arr [-1 2 3 3 4 5 -1] k = 4

l r

sum = 7

l = 0, r = k-1

```
while (r < n-1)
{
    sum = sum - arr[l]
    l++;
    r++
    sum = sum + arr[r];
```

arr [-1, 2, 3, 3, 4, 5, -1] k = 4

l, r

sum = 7 l = 0, r = k-1

```
while (r < n-1)
{
    sum = sum - arr[l]
    l++;
    r++
    sum = sum + arr[r];

    maxSum = max(maxSum, sum);
}
```

arr [-1, 2, 3, 3, 4, 5, -1] k = 4

l r

sum = 7 → for (i = l → r) l = 0, r = k-1

```
while (r < n-1)
{
    sum = sum - arr[l]
    l++;
    r++
    sum = sum + arr[r];

    maxSum = max(maxSum, sum);
}
```

arr [-1 2 3 3 4 5 -1] k = 4

l r

maxSum sum = 7 → for (i = l → r) l = 0, r = k-1

```
while (r < n-1)
{
    sum = sum - arr[l]
    l++;
    r++
    sum = sum + arr[r];

    maxSum = max(maxSum, sum);
}
```

Two pointer / Sliding Window

1. Constant Window
2. Longest subarray / substring where <condition>

Longest subarray with sum <= K

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

TUF

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

15

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.

Longest subarray with sum <= K

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

1. Brute → [illegible] all subarrays.

for (i = 0 → n-1)

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

1. Brute → Generate all subarrays.

for (i = 0 → n-1)

for (j = i → n-1)

```
sum = 0
for (j = i → n-1)
{
    sum = sum + arr[j];
    if (sum <= k)
        maxlen = max(maxlen, j-i+1);

}
}
```

```
sum = 0
for (j = i → n-1)
{
    sum = sum + arr[j];
    if (sum <= k)
        maxlen = max(maxlen, j-i+1);

    }
}
print (maxlen);
```

```
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);
```

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

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;
    }
```

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

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
    }
}
```

Longest subarray with sum <= K

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

l r

$\boxed{r - l + 1}$

1. Expand → r
2. Shrink → l

sum = ~~8~~ ~~2~~ ~~7~~ 8 max len = ~~4~~ ~~2~~ 3

Longest subarray with sum <= K

False

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

l r

1. Expand → r
2. Shrink → l

sum = ~~0~~ ~~2~~ ~~7~~ ~~8~~ 15 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]

    if (sum <= K)
        maxlen = max(maxlen, r-l+1);

}
```

Longest subarray with sum <= K

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

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

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

    if (sum <= K)
        maxlen = max(maxlen, r-l+1);
}
```

Longest subarray with sum <= K

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);

}
```

```
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;
}
```

```
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)
```

start

l, r.

```
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)
```

start

l, r.

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)
```

condition

store

l, r.

Longest subarray with sum <= K

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

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

while (r < n)
{
  sum = sum + arr[r] → condition
  while (sum > K)
  {
    sum = sum - arr[l]
    l = l+1;
  }
  if (sum <= K)
    maxlen = max(maxlen, r-l+1);
  r = r+1;
}

store l, r.

```
}
if (sum <= K)
        maxlen = max(maxlen, r-l+1);
l = l+1;
}

print(maxlen)
```

TC → O(N + N)

$l = 0 \quad r = 0 \quad sum = 0 \quad maxlen = 0$

```
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)
```

condition

store

$l, r.$

TC $\rightarrow O(N + N)$

SC $\rightarrow O(1)$

print (mantlen)

TC → O(N + N)

SC → O(1)

arr → [2 5 1 10 10] k = 14

l

r

L = 14

sum = 2

max len = 0

arr → [2 5 1 10 10] k = 14

l r

sum = ~~0~~ x 7 max len = ~~0~~ x 2

arr → [2 5 1 10 10]

l r

k = 14

sum = 8 + 7 = 18

max len = 8 - 7 + 3

```
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)
```

→ condition

start

l, r.

TC → O(N + N)

SC → O(1)

arr → [2 5 1 10 10] k = 14

sum = 0 × 7 8 18
18 26 21

max len = 0 × 2 3

$l = 0$ $r = 0$ $sum = 0$ $maxlen = 0$

```
while (r < n)
{
    sum = sum + arr[r]
    if (sum > K)
    {
        sum = sum - arr[l]
        l = l+1;
    }
    if (sum <= K)
        maxlen = max(maxlen, r-l+1);
    r = r+1;
}
print(maxlen)
```

condition

start

$l, r.$

TC $\rightarrow O(N)$

SC $\rightarrow O(1)$

```
while (r < n)
{
    sum = sum + arr[r]
    if (sum > K)
    {
        sum = sum - arr[l]
        l = l+1;
    }
    if (sum <= K)
        maxlen = max(maxlen, r-l+1);
    r = r+1;
}

print(maxlen)
```

condition

start

l, r.

TC → O(N)

SC → O(1)

Longest subarray with sum <= K

13

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

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

while (r < n)
{
  sum = sum + arr[r] → condition
  if (sum > K)
  {
    sum = sum - arr[l]
    l = l+1;
  }
  if (sum <= K)
    maxlen = max(maxlen, r-l+1);
  r = r+1;
}

Store l, r.

Two pointer / Sliding Window

1. Constant Window

2. Longest subarray / substring where <condition>
   - → Brute ✓
   - → Better ✓
   - → Optimal ✓

TUF

Two pointer / Sliding Window

1. Constant Window

2. Longest subarray / substring where <condition>
   - → Brute ✓
   - → Better ✓
   - → Optimal ✓

3. No. of subarrays where <conditions>

No. of subarrays with $\boxed{\text{sum} = K}$

No. of subarrays with sum = K

expand
shrink

(no. of subarrays where sum <= K)

(no. of subarrays where sum <= (K-1))

No. of subarrays with sum = K

expand
shrink

(no. of subarrays where sum <= K) = x

(no. of subarrays where sum <= (K-1)) = y

(x - y)

→ Better ✓

→ Optimal ✓

3. No. of subarrays where <conditions>

→ using pattern 2

4.

Longest subarray with sum <= K

→ Better ✓

→ Optimal ✓

3. No. of subarrays where <conditions>

→ using pattern 2

4. Shortest / Minimum Window <condition>

Longest subarray with sum <= K

→ using pattern 2

4. Shortest / Minimum Window <condition>

Longest subarray with sum <= K

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

→ using pattern 2

4. Shortest / Minimum Window < condition >

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

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

```
while (r < n)
{
    sum = sum + arr[r]
    if (sum > K)
    {
        sum = sum - arr[l]
        l = l+1;
    }
    if (sum <= K)
        maxlen = max(maxlen, r-l+1);
    r = r+1;
}
```

condition

store l, r.

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

4. Shortest / Minimum Window <condition>
   - → valid window