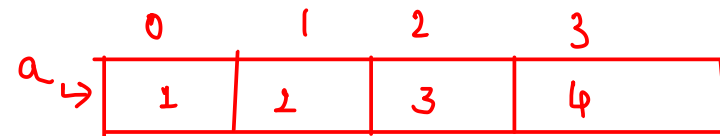


Master Class - Sliding Window

Arrays



sub array

↳ part. of array

↳ MUST be continuous

↗
 $n = 4$

size(1)	size(2)	size(3)	size(4)
<p>0 1 ✓</p>	<p>0 1 1 2</p>	<p>0 1 2 1 2 3</p>	<p>0 1 2 3 1 2 3 4</p>
<p>1 2 ✓</p>	<p>1 2 2 3</p>	<p>1 2 3 2 3 4</p>	
<p>2 3 ✓</p>	<p>2 3 3 4</p>		
<p>3 4 ✓</p>			
4	3	2	1

size.

$n=4$,

$$\text{total sub. arrays} = 4 + 3 + 2 + 1 = 10 \checkmark$$

size.

$n=5$, ✓

$$\begin{aligned} & L_1 + L_2 + L_3 + L_4 + L_5 \\ = & 5 + 4 + 3 + 2 + 1 = 15. \end{aligned}$$

in general (n)

$$= 1 + 2 + \dots + n = \frac{n(n+1)}{2}$$

Q) How many sub arrays are possible with the array having n elements : $n(n+1) / 2$

Q) How many sub arrays are possible of length- k , where array is having n elements?

size:
 $n=5, \checkmark$

size.
 $n=5, \checkmark$

$$L_1 + L_2 + L_3 + L_4 + L_5 = 5 + 4 + 3 + 2 + 1 = \underline{15}$$

Q₁) How many sub arrays are possible with the array having n elements : $\frac{n(n+1)}{2}$

$$\overline{L_1}, \quad L_2, \quad L_3, \quad \dots, \quad L_n$$

Q) How many sub arrays are possible of length-k, where array is having n elements?

Q) How many sub arrays are possible of length-k, where array is having n elements?

L_k

fixed
size
n=5, ✓

$n=5$

	$k=1$ L_1	$+$	$k=2$ L_2	$+$	$k=3$ L_3	$+$	$k=4$ L_4	$+$	$k=5$ L_5	
	5	$+$	4	$+$	3	$+$	2	$+$	1	$= 15$
	$5 - 1 + 1$ $= 5$		$5 - 2 + 1$ $= 4$		$5 - 3 + 1$ $= 3$		$5 - 4 + 1$ $= 2$		$5 - 5 + 1$ $= 1$	

Ans : $n - k + 1$

$n - (n-1) + 1 = \cancel{n} - \cancel{n} + 1 + 1 = 2$

Length
k
 sub-arrays

Ans: $n - \underline{k} + 1$

\downarrow \downarrow \downarrow \downarrow \downarrow
 $\underline{k=1} \checkmark$ $\underline{k=2}$ $\underline{k=3}$ \dots $k=n-1$ $k=n$
 $=$ n $+$ $n-1$ $+$ $n-2$ $+$ \dots $+$ 2 $+$ 1 $=$
←—————→

$= 1 + 2 + 3 + \dots + n = \frac{n(n+1)}{2}$

<u>n</u> ele's, <u>5</u>	<u>k</u> invalid. <u>k > n</u> → 6 x	$k \leq n$ valid ✓
-----------------------------	---	-----------------------

a)

Given input Array, Find the maximum sum of all subarrays of size k

Input : `arr[] = {100, 200, 300, 400}`

`k = 2`

Output : 700

Input : `arr[] = {1, 4, 2, 10, 23, 3, 1, 0, 20}`

`k = 4`

Output : 39

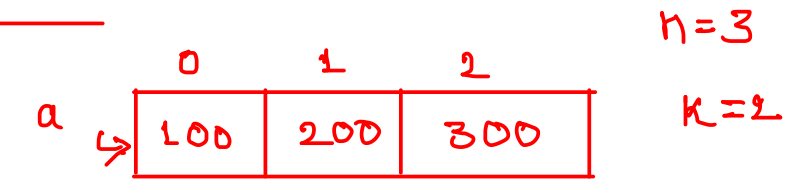
We get maximum sum by adding subarray {4, 2, 10, 23} of size 4.

Input : `arr[] = {2, 3}`

`k = 3`

Output : Invalid

There is no subarray of size 3 as size of whole array is 2.



S_1 : 100 200 \rightarrow 300 ✓

S_2 : 200 300 \rightarrow 500 ✓

} max

{ 500 } #

length - 4 SA's

arr

0	1	2	3	4	5	6	7	8
1	4	2	10	23	3	1	0	20

$n=9$ ✓
 $k=4$ ✓
 $\hookrightarrow k-1$

$$\frac{n-k+1}{9-4+1} = 6 \checkmark$$

max
39 ✓

$i=0$

j 0	1	2	3
1	4	2	10

= 17

$a[0] + a[1] + a[2] + a[3]$

$i=1$

1	2	3	4
4	2	10	23

= 39

$a[1] + a[2] + a[3] + a[4]$

$i=2$

2	3	4	5
2	10	23	3

= 38

$a[2] + a[3] + a[4] + a[5]$

$i=3$

3	4	5	6
10	23	3	1

= 37

$i=4$

4	5	6	7
23	3	1	0

= 27

$i=5$

5	6	7	8
3	1	0	20

= 24

```
function maxSum(arr[], n, k)
{
    res=-infinity
    for(i=0; i<=n-k; i++)
    {
        sum=0
        for(j=i; j<=i+(k-1); j++) // k times
        {
            sum=sum+arr[j]
        }
        if(sum>res)
        {
            res=sum
        }
    }
    print(sum)
}
```


T.C: -N
↻

k: constant

```
function maxSum(arr[], n, k)
```

```
{
```

```
    res=-infinity
```

```
    for(i=0; i<=n-k; i++) → n-k+1.
```

```
    {
```

```
        sum=0
```

```
        for(j=i; j<=i+(k-1); j++) // k times → k times ✓
```

```
        {
```

```
            sum=sum+arr[j]
```

```
        }
```

```
        if(sum>res)
```

```
        {
```

```
            res=sum
```

```
        }
```

```
    }
```

```
    print(sum)
```

```
}
```

$$(n-k+1) \times k$$

$$= \frac{n \cdot k - k^2 + k}{1}$$

TC

$$\therefore O(n \cdot k)$$

∴ if

$$k \approx \frac{n}{2}$$

$$= n \cdot \frac{n}{2}$$

$$= \frac{n^2}{2}$$

$$\therefore O(n^2)$$

can you come with any idea, in which the T.C is completely independent of k

$O(n)$

not $O(n.k)$

$$i: 4; \quad + a[4] - a[0] \quad | \quad i: 6; \quad + a[6] - a[2]$$

$$i: 5; \quad + a[5] - a[1]$$

$n=9$

$k=4 \checkmark$

arr

0	1	2	3	4	5	6	7	8
1	4	2	10	23	3	1	0	20

SA₁

0	1	2	3
1	4	2	10

: $1 + 4 + 2 + 10 = 17$

+ - . .

function maxSum(arr[], n, k)

{

sum=0

for(i=0; i<k; i++)

{

sum=sum+arr[i]

}

temp=sum

for(i=k; i<n; i++)

{

sum=sum+arr[i]-arr[i-k]

if(sum>temp)

temp=sum

}

print(temp

}

}

temp=17

sum=17

SA₂

1	2	3	4
4	2	10	23

$4 + 2 + 10 + 23$

$1 + 4 + 2 + 10 - 1 + 23$

$= 4 + 2 + 10 + 23 \checkmark$

SA₃

2	3	4	5
2	10	23	3

$2 + 10 + 23 + 3 - 4$

$= 2 + 10 + 23 + 3$

3	4	5	6
10	23	3	1

4	5	6	7
23	3	1	0

SA₆

5	6	7	8
3	1	0	20

arr

0	1	2	3	4	5	6	7	8
1	4	2	10	23	3	1	0	20

(Note: In the original image, the first element '1' is circled in blue, and a blue line is drawn under the first four elements: 1, 4, 2, 10.)

$n=9$
 $k=4$ fixed

$$\begin{aligned}
 \text{sum} &= 17 = \underline{17} + 23 - 1 \\
 &= (1+4+2+10) + 23 - 1 \\
 \text{temp} &= \cancel{17} = 4 + 2 + 10 + 23 \\
 &39 = \underline{39}
 \end{aligned}$$

```

function maxSum(arr[],n,k)
{
    sum=0
    for(i=0;i<k;i++)
    {
        sum=sum+arr[i]
    }
    temp=sum
    → for(i=k; i<n; i++)
    {
        sum=sum+arr[i]-arr[i-k]
        if(sum>temp)
            temp=sum
    }
    print(temp)
}
  
```

(Note: In the original image, there is a blue annotation '4-4=0' next to the subtraction term in the sliding window loop.)

$\begin{matrix} n \\ \swarrow \searrow \\ \textcircled{k} \quad n-k \end{matrix}$

```

function maxSum(arr[], n, k)
{
    sum=0
    for(i=0; i<k; i++)
    {
        sum=sum+arr[i]
    }
    temp=sum
    for(i=k; i<n; i++)
    {
        sum=sum+arr[i]-arr[i-k]
        if(sum>temp)
            temp=sum
    }
    print(temp)
}
  
```

$\begin{matrix} \updownarrow k \\ \text{---} + \\ \text{---} n-k \end{matrix}$

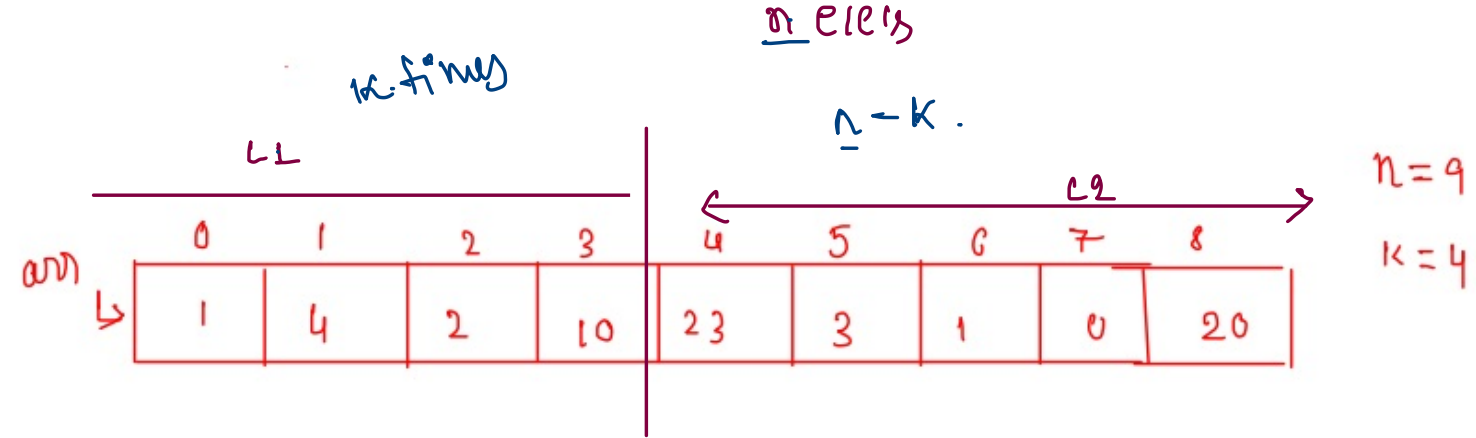
$\begin{matrix} = k + n - k = n \\ \text{TC} \quad \therefore O(n) \end{matrix}$

$O(n-k)$

$\textcircled{-\infty}$ smallest ✓

```

function maxSum(arr[], n, k)
{
    → res=-infinity
    for(i=0; i<=n-k; i++)
    {
        sum=0
        for(j=i; j<=i+(k-1); j++) // k times
        {
            sum=sum+arr[j]
        }
        if(sum>res)
        {
            res=sum
        }
    }
    print(sum)
}
  
```



```
function maxSum(arr[],n,k)
{
    sum=0
    L1 for(i=0;i<k;i++)
    {
        sum=sum+arr[i]
    }
    temp=sum
    L2 for(i=k; i<n; i++)
    {
        sum=sum+arr[i]-arr[i-k]
        if(sum>temp)
            temp=sum
    }
    print(temp)
}
```