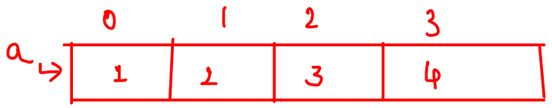
Master Class - Sliding Window

Arrays

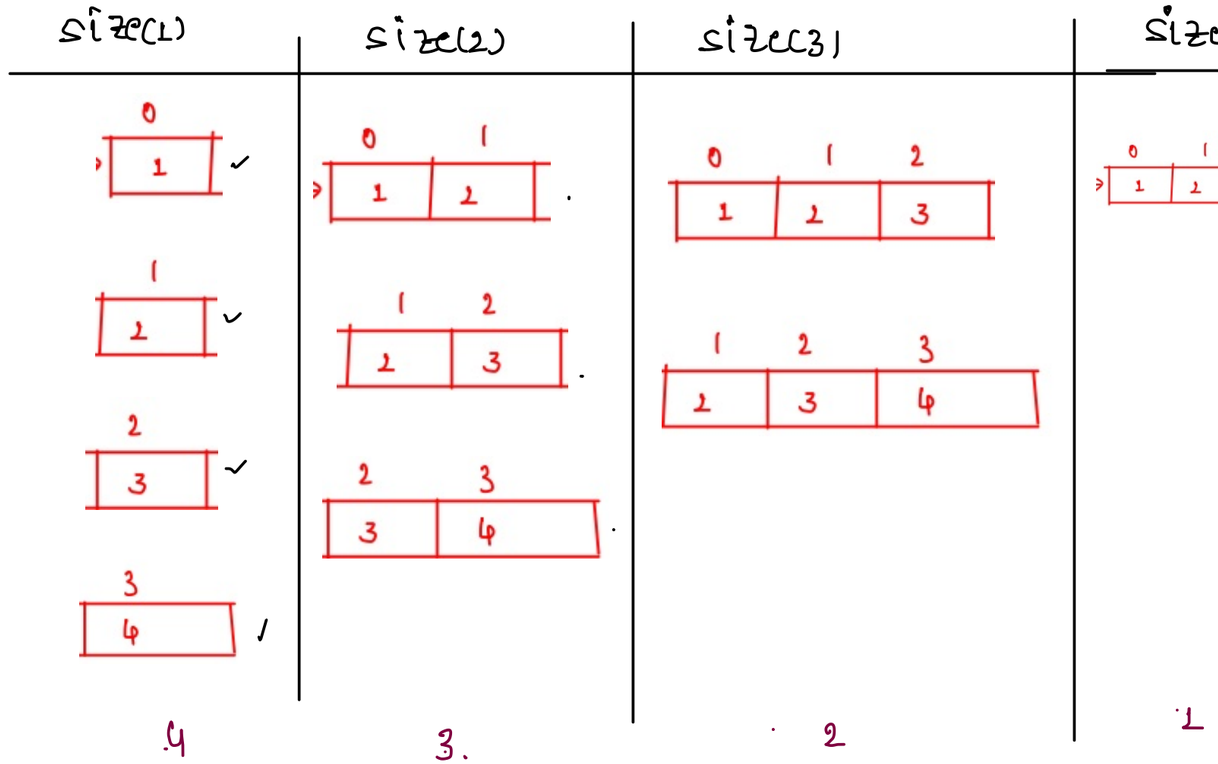


n = 4

sub array

↳ part. of array

↳ MUST be continious



size.

n=4,

total sub. arrays = 4 + 3 + 2 + 1 = 10 ✓

---

size.

n=5, ✓

$L_1 + L_2 + L_3 + L_4 + L_5$

$= 5 + 4 + 3 + 2 + 1 = 15.$

---

in general. (n)

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

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,

$L_1 + L_2 + L_3 + L_4 + L_5$

$= 5 + 4 + 3 + 2 + 1 = 15$

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

$L_1, L_2, L_3, \ldots L_n$

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

	K=1	K=2	K=3	K=4	K=5	
	L_1	+ L_2	+ L_3	+ L_4	+ L_5	
n=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 : $n - k + 1$

Length K sub-arrays

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

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

$k=1$ ✓ | $k=2$ | $k=3$ | . . . . . . | $k=n-1$ | $k=n$

$= n + (n-1) + (n-2) + \cdots + 2 + 1 =$

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

$\underline{n}$ · ele's, 5

$\underline{K}$ invalid.

$\underline{K} > \underline{n} \rightarrow$

6 ✗

$K \le n$

valid ✓

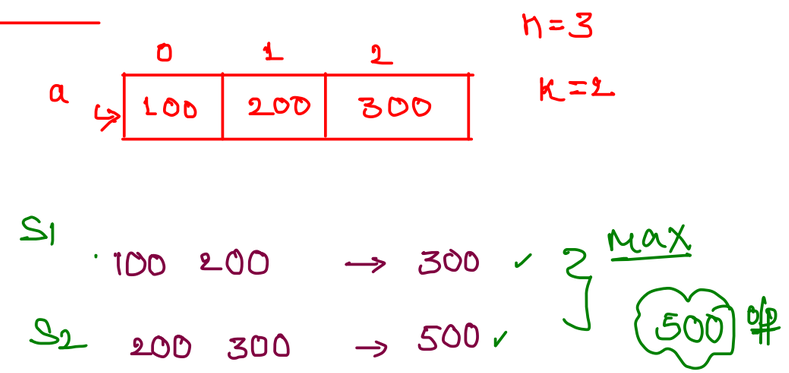
a)

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

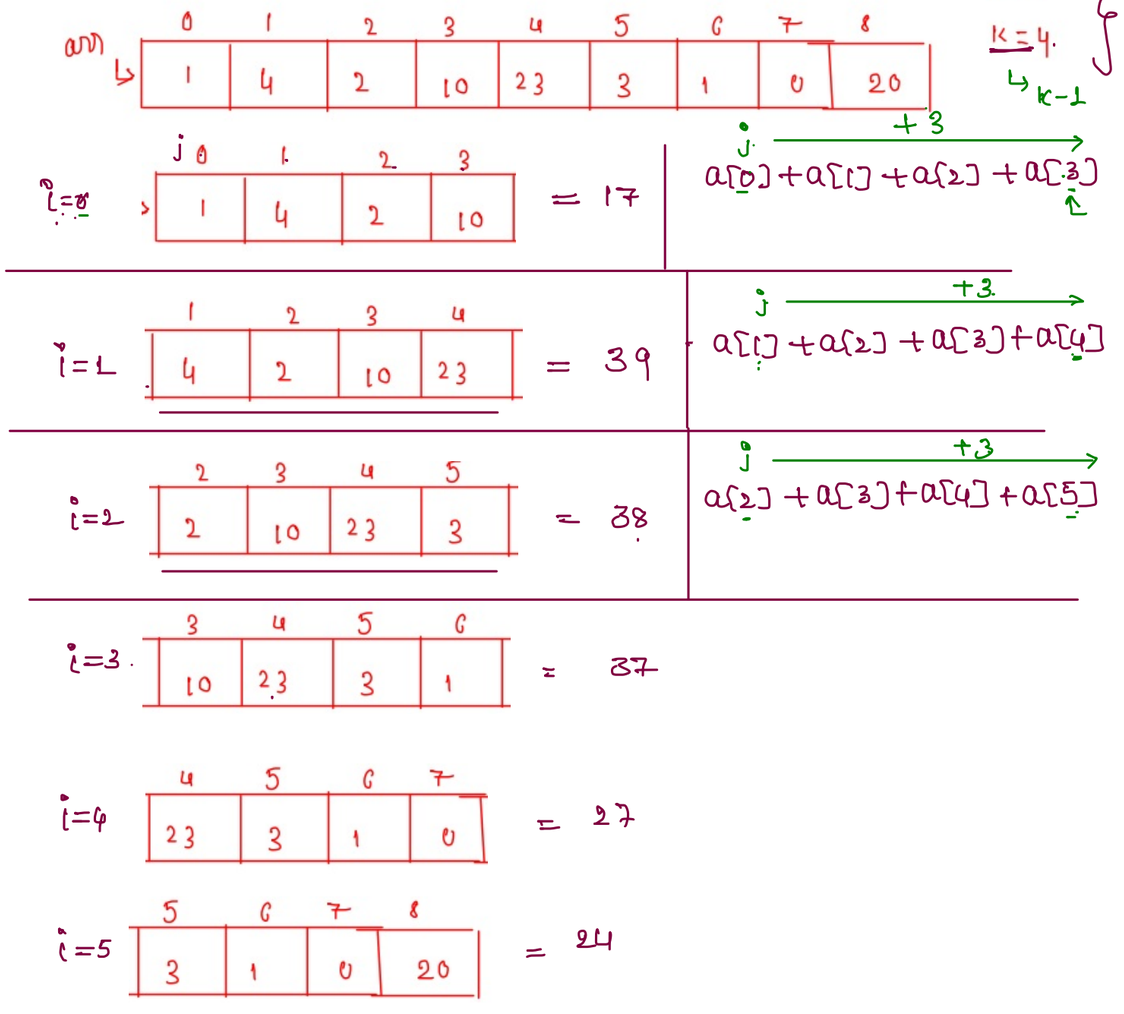
```
Input  : arr[] = {100, 200, 300, 400}
         k = 2
Output : 700

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.
```



Length - 4 SA's



$\frac{n-k+1}{9-4+1} = 6$ ✓

```
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)
}
```

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

$= n \cdot k - k^2 + k$

TC

$\therefore O(n \cdot k)$:- if $k \simeq \frac{n}{2}$

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

$\therefore O(n^2)$

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

O(n)

not O(n.k)

i:4; j: +a[4] − a[0]
i:5 ; j: +a[5] − a[1]

i=6; + a[6] − a[2]

i

n=9

k=4 ✓

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

SA1 : 1+4+2+10 = 17

temp=17

sum=17 ↓

+, − . .

O(k)

1	2	3	4
4	2	10	23

✓ SA2

4+2+10+23

+, − : O(1)

x+4+2+10 −x +23

= 4+2+10+23 ✓

2	3	4	5
2	10	23	3

SA3

(k)

+, − : O(1)

4+2+10+23+3 −4

= 2+10+23+3

3	4	5	6
10	23	3	1

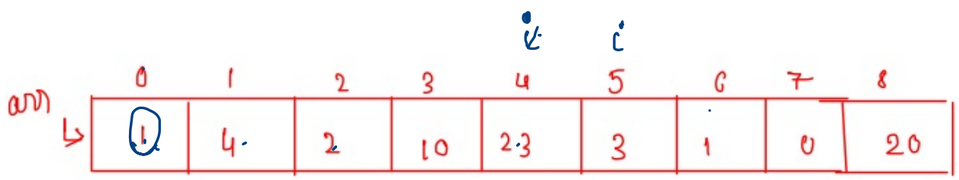
4	5	6	7
23	3	1	0

5	6	7	8
3	1	0	20

SA4

```
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
}
```

}



n = 9

k = 4 fixed

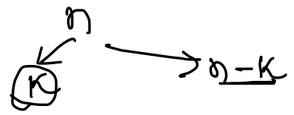
Sum = 17 = 17 + 23 − 1

= (1+4+2+10) + 23 − 1

temp = 17 = 4 + 2 + 10 + 23

39 = 39

```
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++)
        {                                   4 − 4 = 0
                sum=sum+arr[i]-arr[i-k]
                if(sum>temp)
                        temp=sum
        }
        print(temp
}
```




```
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)
}
```

K

+

n-K

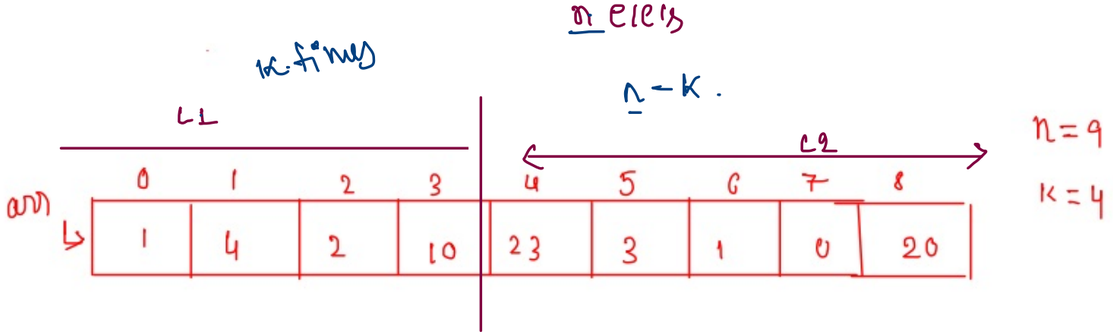
$= K + n - K = n$

TC $\therefore O(n)$

$O(n-k)$

$-\infty$ 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)
}
```