Sliding Window (SW)

e - s + 1

Type-1

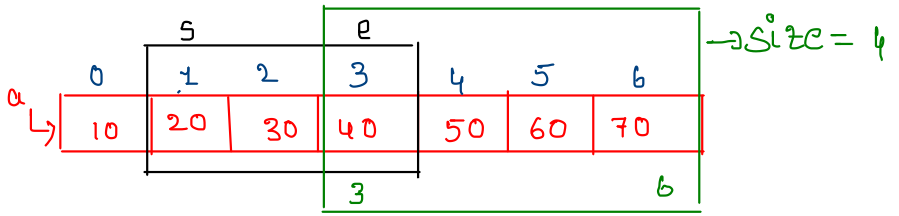
→ Fixed size sw ✓

size : fixed, give in question itself

Type-2

→ variable size sw

size : not fixed / not given



s = 1, e = 3 } 3 ✓ → 3 - 1 + 1

s = 3, e = 6 } 4 ✓ → 6 - 3 + 1

s = 1, e = 5 } 5 → 5 - 1 + 1 ⇒ e - s + 1

SW

Where we can Apply ?

Arrays / Strings + Sub-Array / Sub-string + Largest sum

smallest/min/max

Window Size : k

Example :-

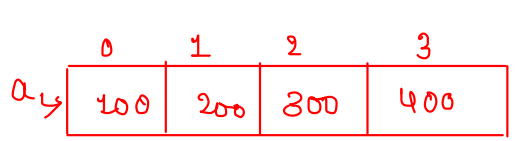
Given input Array, Find the maximum sum of all subarays of size k

n

Model - 1 [Fixed Size SW]

7) 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.
```



n=4 ✓

k=2

4! ✓

10 ✓

$^4C_2$

n=4

len(1) : 100 200 300 400 ⇒ 4

len(2) : 100 200 200 300 300 400 ⇒ 3

len(3) : 100 200 300 200 300 400 ⇒ 2

len(4) : 100 200 300 400 ⇒ 1

len(5) : X

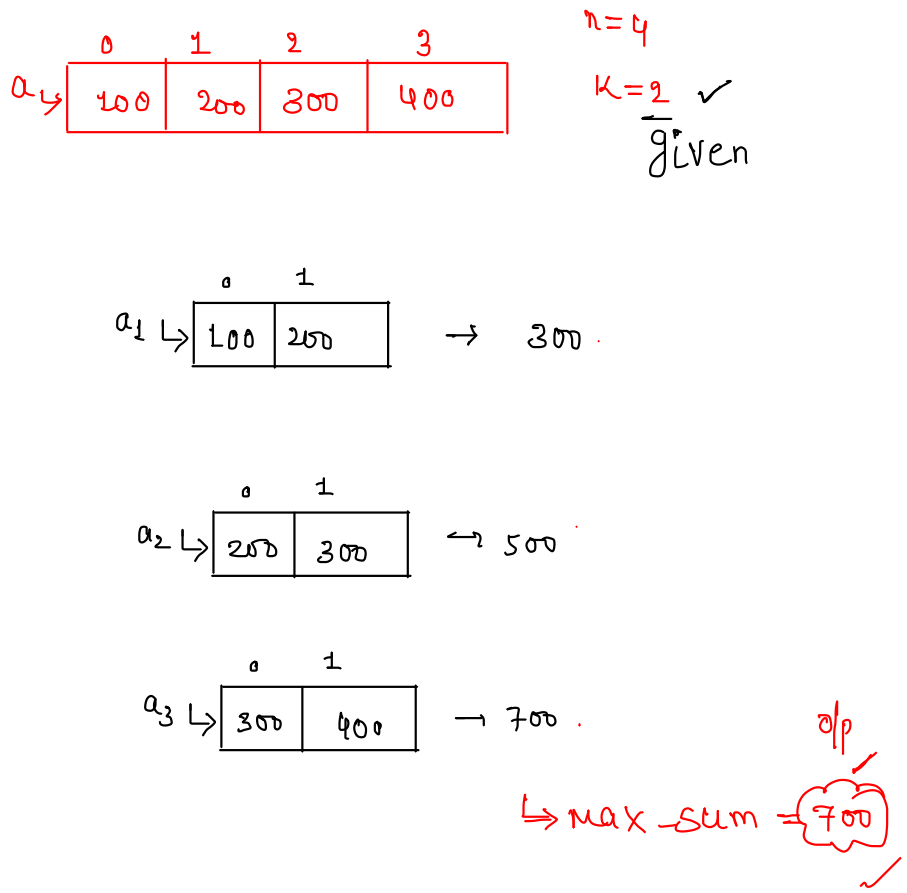
1+2+3+4. ✓

n=5 ⇒ 1+2+3+4+5

n : 1+2+3+ . . . +n : $\frac{n(n+1)}{2}$ ⇒ O(n²)

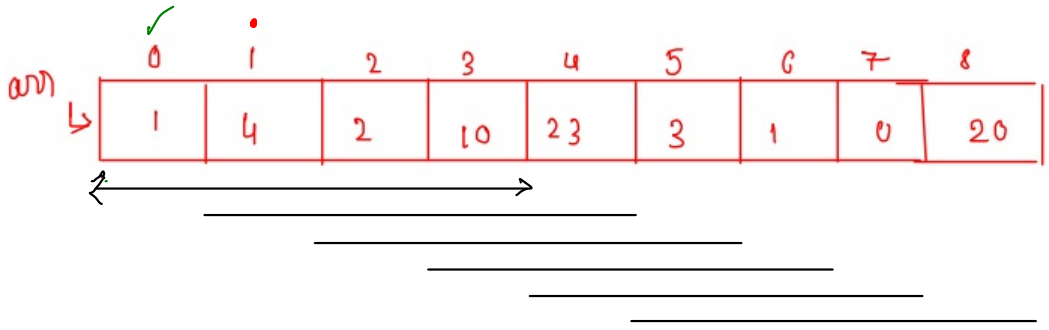
7) Given input Array, Find the maximum sum of all subarays of size k

```
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.
```



Sol

Ex:-



n = 9 ✓

k = 4 ✓

i : 0 → 5

n-k

n >> k
big

⇒ (n-k) * k

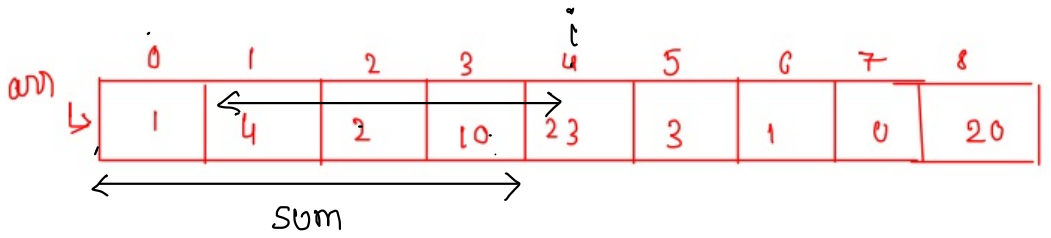
n * k − k * k
(1) (1)

⇒ O(n*k)

- ✓ SA1 : 0 → 3 ; 1 + 4 + 2 + 10 : 17
  i + 3
- ✓ SA2 : 1 → 4 : 4 + 2 + 10 + 23 : 39
  i + 3
- ✓ SA3 : 2 → 5 : 2 + 10 + 23 + 3 : 38
  i + 3
- ✓ SA4 : 3 → 6 ✓
  i + 3 ↳ i+k-1
- ✓ SA5 : 4 → 7 ✓
- ✓ SA6 : 5 → 8 ✓

```
✓function maxSum(arr,n,k)
 {
        max_sum=-infinity
        for(i=0;i<=n-k;i++)    ⇒ n-k
        {
            sum=0;
            for(j=i;j<=i+k-1;j++)   ⇒ k
            {
                sum=sum+arr[j];
            }
            if(sum>max_sum)
            {
                max_sum=sum
            }
        }
        return max_sum;
 }
```

```
function fun(arr,n,k) // fixed size
{
  max_sum=-Infinity
  for(i=0;i<=n-k;i++)
  {
    sum=0
    for(j=i; j<=i+k-1; j++)
    {
      sum=sum+arr[j]
    }
    if(sum>max_sum)
    {
      max_sum=sum)
    }
  }
  return max_sum
}
```



n = 9

k = 4 ✓

✓ temp = sum

sum = 1 + 4 + 2 + 10 [SA1]

Step 1

```
for(i=0; i<=k-1; i++)
{
    sum = sum + a[i]
}
```

```
for(i = k/4; i<=n-1; i++)
{
    · sum = sum - a[i-k] + a[i]
    · temp = Math.max(sum, temp)
}
```

✓ ✓ SA1 : 0 → 3 : 1 + 4 + 2 + 10 : 17 SUM(SA1) → SUM

i + 3

→ ✓ SA2 : 1 → 4 : 4 + 2 + 10 + 23 : 39 ⇒ SUM(SA1) − 1 + 23 : SUM(SA2) ⇒ ~~SUM~~ −

17

i=4 sum − a[0] + a[4] ✓

i=5 sum − a[1] + a[5]

✓ SA3 : 2 → 5 : 2 + 10 + 23 + 3 : 38 ⇒ SUM(SA2) − 4 + 3

i + 3

✓ SA4 : 3 → 6 ✓

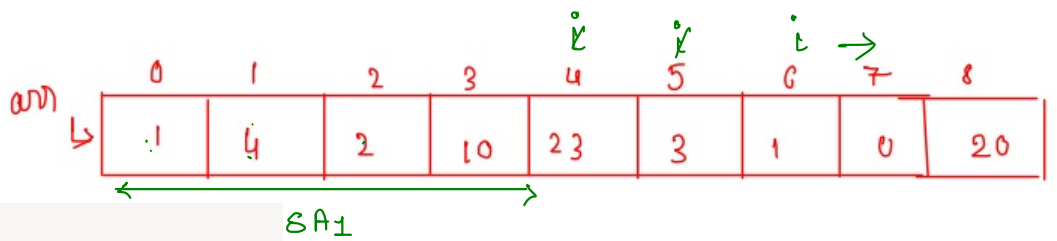
i + 3 ↳ i+k-1

✓ SA5 : 4 → 7 ✓

✓ SA6 : 5 → 8 ✓

```
function fun(arr,n,k) // fixed size
{

  max_sum=0
  for(i=0;i<=k-1;i++)
  {
    max_sum=max_sum+arr[i]
  }

  curr_sum=max_sum
  for(i=k; i<n; i++)
  {
   1. curr_sum=curr_sum-arr[i-k]+arr[i]
   2. max_sum=Math.max(curr_sum,max_sum)
  }
  return max_sum

}
```

max_sum = 0 ~~17~~ 39



n = 9

k = 4

curr_sum = 17 − 1 + 23 = 39

↳ 4 + 2 + 10 + 23

= 39 − 4 + 3 = 38

AP1 :- Brute Force ✓ O(n*k)

```
function fun(arr,n,k) // fixed size
{
  max_sum=-Infinity
  for(i=0;i<=n-k;i++)
  {
    sum=0
    for(j=i; j<=i+k-1; j++)
    {
      sum=sum+arr[j]
    }
    if(sum>max_sum)
    {
      max_sum=sum)
    }
  }
  return max_sum
}
```

k times ✓ ↓ TC

AP2 (SW) O(n)

```
function fun(arr,n,k) // fixed size
{

  max_sum=0
L1 for(i=0;i<=k-1;i++)
  {
    max_sum=max_sum+arr[i]
  }

  curr_sum=max_sum
L2 for(i=k; i<n; i++)
  {
    curr_sum=curr_sum-arr[i-k]+arr[i]
    max_sum=Math.max(curr_sum,max_sum)
  }
  return max_sum

}
```

k

n-k

k + n - k = n

Model-2 [Variable Size SW]

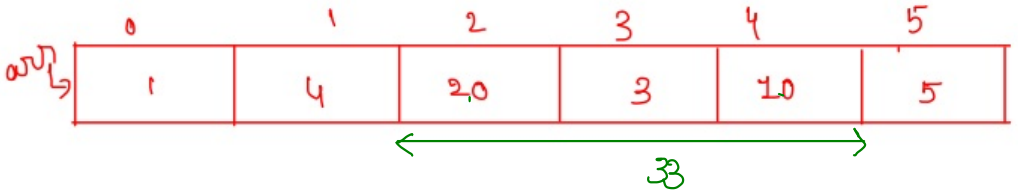
problem's, will take help of "objects"

↓

Key + Value

↳ size : not given

8) Find is there any sub-array with the given sum [return True/ False]



↳ True ✓

low → s_1 ✓

high → e ✓

n = 6 ✓ → $\frac{6 \times 7}{2} = 21$

sum = 33 ✓

↳ s_1 ✓

s_2 ✓

⋮

s_{21} ✓

Step 1 :-

✓ Generate all SA's (Sum) ✓

Step 2 :-

Compare it with given sum

Sub-Array is cont.

```
function fun(arr,n,sum) // variable size S.W
{
 windowSum=0, high=0
 for(low=0;low<n;low++)
 {
  while(windowSum<sum && high<n)
  {
    windowSum=windowSum+arr[high]
    high++
  }
  if(windowSum==sum) // happy
  {
    return true
  }
  windowSum=windowSum-arr[low]
 }
  return false
}
```

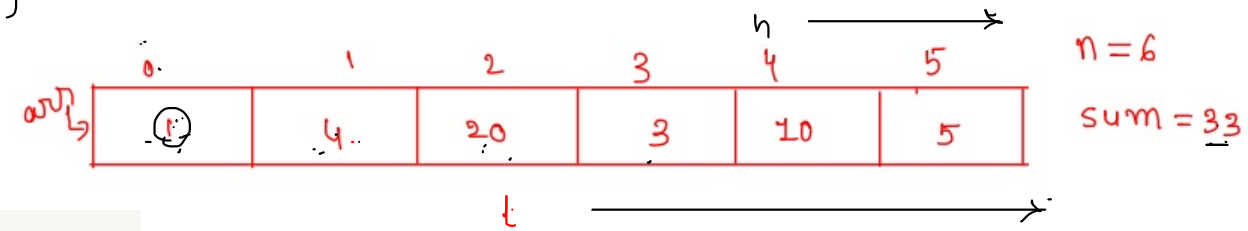
C2

add

C1

C3

remove



WS = ~~1~~ + ~~4~~ + 20 + 3 + 10 = 38

= 37

= 33

38 > 33

WS v/s sum