S2-Class1 [Sliding Window-1]

```
function fun(n)
{
    q=0
    for(i=1;i<=n;i++)
    {
        p=0
        for(j=n; j>1; j=j/2)
        {
            ++p
        }
        for(k=1; k<p; k=k*2 )
        {
            ++q
        }
    }
}
```

$< n$

L_1 : n

L_2 : $\log_2 n$

$p = \log_2 n$

L_3 : $\log_2 p$

$L_1 * (L_2 + L_3)$

$= n * (\log_2 n + \log_2 \log_2 n)$

$\max(L_2, L_3) = \log_2 n$

$= O(n \cdot \log_2 n)$

$n \to n/2 \to n/4 \to \cdots \to 2 \to 1$

```
p=0
for(j=n; j>1; j=j/2)
{
    ++p
}
```

$\Rightarrow \log_2 n$

$p = \log_2 n$

```
for(k=1; k<p; k=k*2 )
{
    ++q
}
```

$\Rightarrow \log_2 p$

$n * (\log_2 n + \log_2 p)$

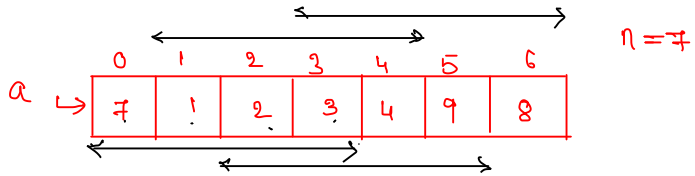
$O(\cdots n)$

Sliding Window (S.W)

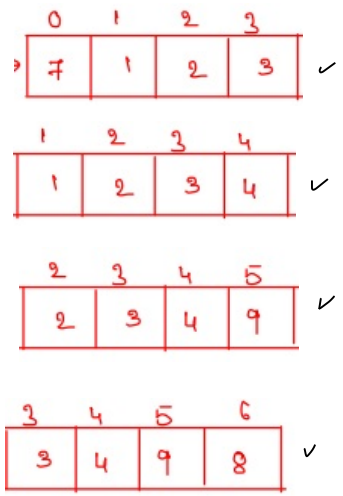
more 2ptr (same) Dir

Type-1

- SW of fixed size.
- size is given in prob itself

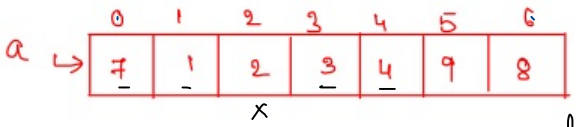


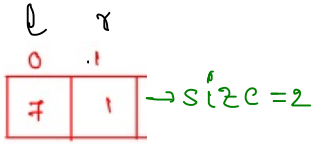
size=4



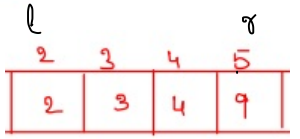
Type-2

- SW of variable size
- size, we need to find -

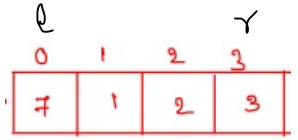




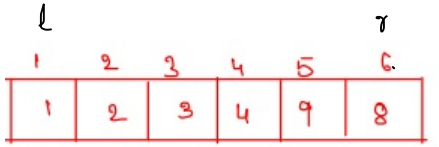
→ size = 2



→ size = 4



↳ size = 4



↳ size = 6

r + 1 ✗

↳ r − l + 1

size = r − l + 1

✓ size = end − beg + 1

Sliding Window

Where we can Apply ?

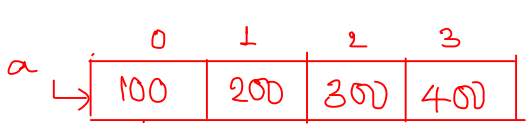
Arrays / Strings + Sub-Array / Sub-string + Largest sum

Window Size : k

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.
```

⇒ 3 sub Arr's



Total # of subarrays possible:-

× Length1 : 100 200 300 400 = 4.

* Length2 : 100 200 | 200 300 | 300 400 = 3

× length3 : 100 200 300 | 200 300 400 = 2.

× length4 : 100 200 300 400 = 1.

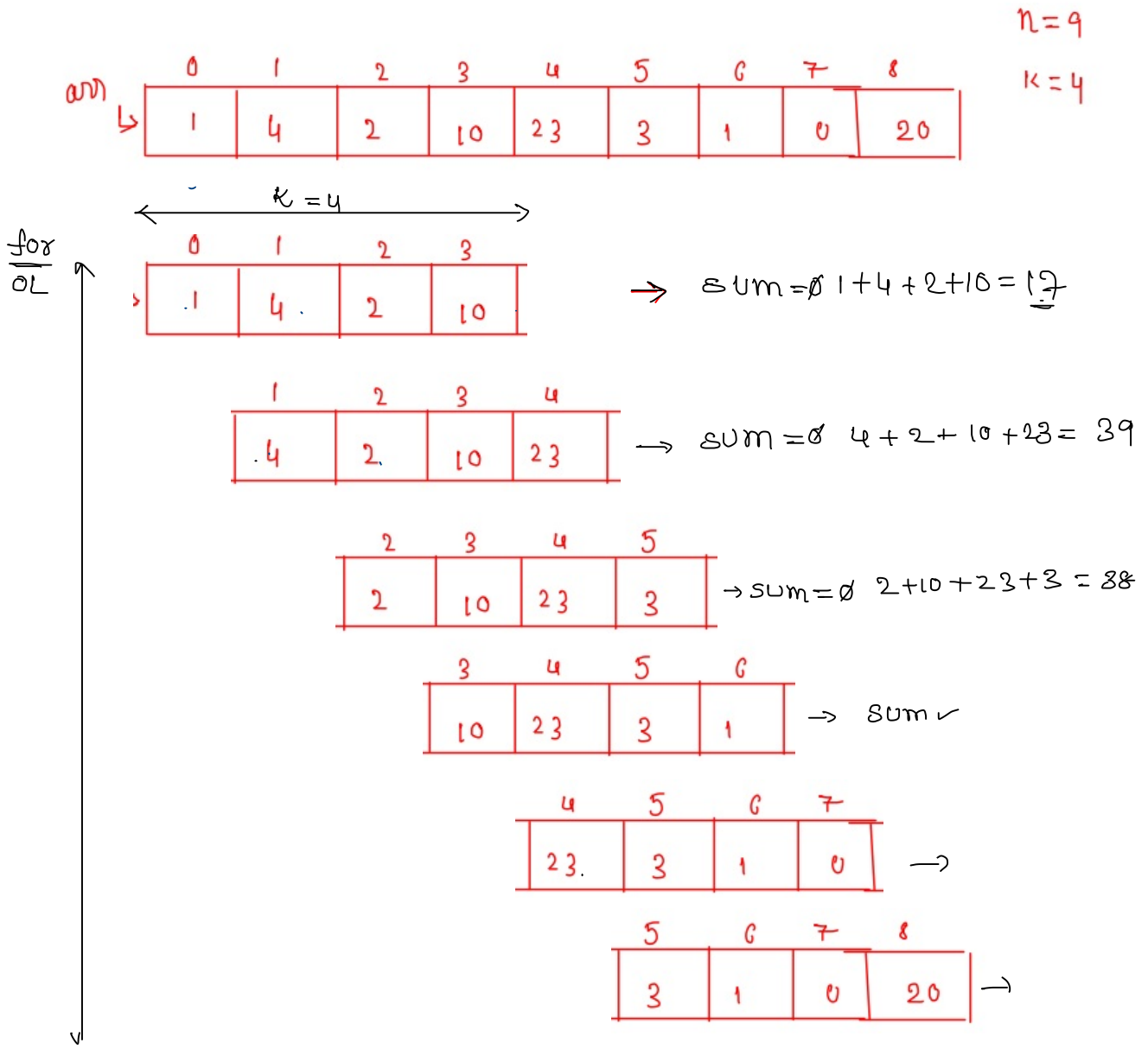
× length5 : X

1 + 2 + 3 + 4 = 10

n = 5 : 5 + 4 + 3 + 2 + 1 = 15

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

maximum_sum



n-k+1

9-4+1=6

res = -∞ 17 39

```
int fun(int arr[], int n, int k)
{
        int res=INTEGER.MIN_VALUE;
        for(int i=0;i<n-k+1;i++)
        {
                int sum=0;
                for(int j=i;j<i+k;j++)
                {
                        sum=sum+arr[j]
                }
                res=Math.max(sum,res);
        }
        return res;

}
```

i = 0 1

j = 0 1 2 3 4

i+1

j < i+k

4 < 0+4

```
int fun(int arr[], int n, int k)
{
        int res=INTEGER.MIN_VALUE;
    L1  for(int i=0;i<n-k+1;i++)
        {
            int sum=0;
          L2 for(int j=i;j<i+k;j++)
            {
                sum=sum+arr[j]
            }
            res=Math.max(sum,res);
        }
        return res;

}
```

L_1 → $n-k$

L_2 → k

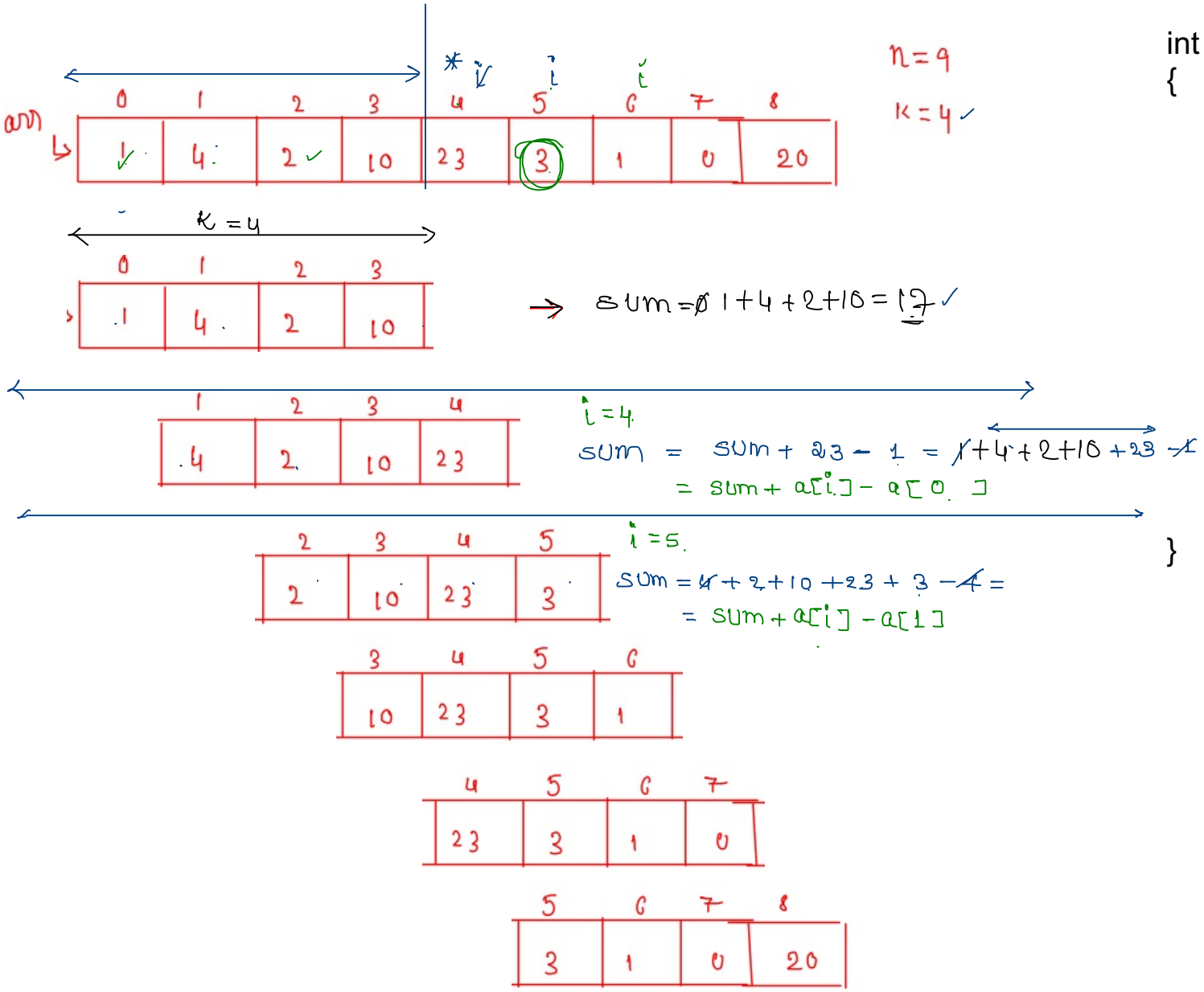
$(n-k) * k$

$= n*k - k^2$

$= O(n-k)$

↳ TLE

$n >> k$




```
int fun(int arr[], int n, int k)
{
        int sum=0;
    L1  for(int i=0;i<k;i++)
        {
                sum=sum+arr[i];
        }
        int res=sum;
        for(int i=k;i<n;i++)
        {
                sum=sum+arr[i]-arr[i-k];
                res=Math.max(res,sum);
        }
        return res;
}
```

n ✓

```
int fun(int arr[], int n, int k)
{
        int sum=0;
        for(int i=0;i<k;i++)
        {
                sum=sum+arr[i];
        }
        int res=sum;
        for(int i=k;i<n;i++)
        {
                sum=sum+arr[i]-arr[i-k];
                res=Math.max(res,sum);
        }
        return res;
}
```

k → k

n−k → n−k

k + n − k = n

∴ O(n)

Ap1 :- Brute Force : O(n · k)

12:45pm

```
int fun(int arr[], int n, int k)
{
        int res=INTEGER.MIN_VALUE;
        for(int i=0;i<n-k+1;i++)
        {
                int sum=0;
                for(int j=i;j<i+k;j++)
                {
                        sum=sum+arr[j]
                }
                res=Math.max(sum,res);
        }
        return res;

}
```

Ap2 :- Using SW : O(n)
(Fixed)

```
int fun(int arr[], int n, int k)
{
        int sum=0;
        for(int i=0;i<k;i++)
        {
                sum=sum+arr[i];
        }
        int res=sum;
        for(int i=k;i<n;i++)
        {
                sum=sum+arr[i]-arr[i-k];
                res=Math.max(res,sum);
        }
        return res;
}
```

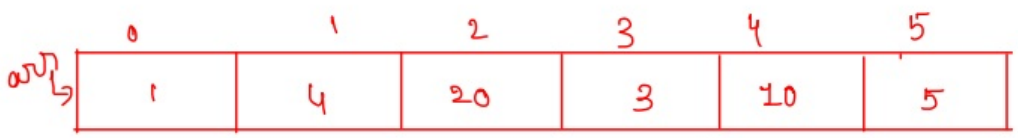
```
int maxSum(int arr[], int k, int n)
{
        int sum=0;
        for(int i=0;i<k;i++)
        {
                sum=sum+arr[i];
        }
        int res=sum;
        for(int i=k;i<n;i++)
        {
                res=res-arr[i-k]+arr[i];
                sum=Math.max(res,sum);
        }
        return sum;

}
```

2pt

Q) Return True : If there exists a sub array whose sum is equal to given sum

False : otherwise

Variable size SW



n = 6

sum = 33 ✓

op → T/F

Hashmap, Hashset

Ap1 :-

↳ Generate All sub-arrays (len1, len2, ... len6)

(6) (5) (4) (1)

n => n(n+1)/2 sub arrays possible.

n(n+1)/2 × n => O(n³)

↳ TLE ✓

Ap2

↓

Using SW (variable size)

→

```
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
}
```