Table of content

Window sliding 1,2

Window Sliding Technique

* Difficulty Level : [Easy](https://www.geeksforgeeks.org/easy/)

This technique shows how a nested for loop in some problems can be converted to a single for loop to reduce the time complexity.  
Let’s start with a problem for illustration where we can apply this technique –

Given an array of integers of size **‘n’**.

Our aim is to calculate the maximum sum of **‘k’**

consecutive elements in the array.

import sys

print "GFG"

# O(n \* k) solution for finding

# maximum sum of a subarray of size k

INT\_MIN = -sys.maxsize - 1

# Returns maximum sum in a

# subarray of size k.

def maxSum(arr, n, k):

# Initialize result

max\_sum = INT\_MIN

# Consider all blocks

# starting with i.

for i in range(n - k + 1):

m = 0

current\_sum = 0

for j in range(k):

current\_sum = current\_sum + arr[i + j]

# Update result if required.

max\_sum = max(current\_sum, m)

m = max\_sum

current\_sum=0

return m

# Driver code

arr = [1, 4, 2, 10, 2,

3, 1, 0, 20]

k = 4

n = len(arr)

print(maxSum(arr, n, k))

It can be observed from the above code that the time complexity is **O(k\*n)** as it contains two nested loops.

**Window Sliding Technique**

The technique can be best understood with the window pane in bus, consider a window of length **n** and the pane which is fixed in it of length **k**. Consider, initially the pane is at extreme left i.e., at 0 units from the left. Now, co-relate the window with array arr[] of size n and pane with current\_sum of size k elements. Now, if we apply force on the window such that it moves a unit distance ahead. The pane will cover next **k** consecutive elements.   
Consider an array **arr[]** = {5, 2, -1, 0, 3} and value of **k** = 3 and **n** = 5  
**Applying sliding window technique**:

1. We compute the sum of first k elements out of n terms using a linear loop and store the sum in variable window\_sum.
2. Then we will graze linearly over the array till it reaches the end and simultaneously keep track of maximum sum.
3. To get the current sum of block of k elements just subtract the first element from the previous block and add the last element of the current block .

The below representation will make it clear how the window slides over the array.  
This is the initial phase where we have calculated the initial window sum starting from index 0 . At this stage the window sum is 6. Now, we set the maximum\_sum as current\_window i.e 6. 



Now, we slide our window by a unit index. Therefore, now it discards 5 from the window and adds 0 to the window. Hence, we will get our new window sum by subtracting 5 and then adding 0 to it. So, our window sum now becomes 1. Now, we will compare this window sum with the maximum\_sum. As it is smaller we wont the change the maximum\_sum. 



Similarly, now once again we slide our window by a unit index and obtain the new window sum to be 2. Again we check if this current window sum is greater than the maximum\_sum till now. Once, again it is smaller so we don’t change the maximum\_sum.  
Therefore, for the above array our maximum\_sum is 6.



**code for the above description :**

# O(n) solution for finding

# maximum sum of a subarray of size k

def maxSum(arr, k):

    # length of the array

    n = len(arr)

    # n must be greater than k

    if n < k:

        print("Invalid")

        return -1

    # Compute sum of first window of size k

    window\_sum = sum(arr[:k])

    # first sum available

    max\_sum = window\_sum

    # Compute the sums of remaining windows by

    # removing first element of previous

    # window and adding last element of

    # the current window.

    for i in range(n - k):

        window\_sum = window\_sum - arr[i] + arr[i + k]

        max\_sum = max(window\_sum, max\_sum)

    return max\_sum

# Driver code

arr = [1, 4, 2, 10, 2, 3, 1, 0, 20]

k = 4

print(maxSum(arr, k))

2 - Given an array of integers nums and an integer k, return the total number of continuous subarrays whose sum equals to *k*.  
  
**naïve answer:**

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

k = 2

# arr = [ 10, 2, -2, -20, 10 ]

n = len(arr)

# k = -10

res = 0

# Calculate all subarrays

for i in range(n):

summ = 0

for j in range(i, n):

# Calculate required sum

summ += arr[j]

# Check if sum is equal to

# required sum

if summ == k:

res += 1

print(res)