# Sliding Window Technique

This technique shows how a nested for loop in few problems can be converted to single for loop and hence reducing 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.

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.
```

The **Naive Approach** to solve this problem is to calculate sum for each of the blocks of K consecutive elements and compare which block has the maximum sum possible. The time complexity of this approach will be O(n * k).

**Window Sliding Technique**

The above problem can be solved in Linear Time Complexity by using Window Sliding Technique by avoiding the overhead of calculating sum repeatedly for each block of k elements.

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 plane 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 .

```
//C++ code for sliding window Algorithm: -

#include <bits/stdc++.h>
using namespace std;

// Returns maximum sum in a subarray of size k.
int maxSum(int arr[], int n, int k) {
    // n must be greater
    if (n < k) {
        cout << "Invalid";
        return -1;
    }

    //sum of first window of size k
    int window_sum = 0;
    for (int i = 0; i < k; i++)
        window_sum += arr[i];

    // Compute sums of remaining windows by
    // removing first element of previous
    // window and adding last element of
    // current window.
    int max_sum = window_sum;
```

```
    for (int i = k; i < n; i++) {
        window_sum += (arr[i] - arr[i - k]);
        max_sum = max(max_sum, window_sum);
    }
    return max_sum;
}

int main(){
    int n = 6 , k = 3;;
    int arr[] = {16,12,9,19,11,8};
    cout << maxSum(arr, n, k);
    return 0;
}
```