Introduction
Sliding Window is one of the useful techniques that comes handy when we are asked to find or calculate something on subarrays or continuous subsequences of linear data structures such as linked lists, strings and dynamic arrays. For example, let us take the following problem:

Problem Description:
Given an array of size 'n', find the maximum sum subarray of size 'k' or find the maximum sum of k consecutive elements in the array.

Let's understand this problem with the help of a test case:
Array: [1, 13, 2, 16, -10, 4, 11, 18, 2], K=5
Here, we are asked to find the maximum sum subarray of size '5'. Let's try to solve it:

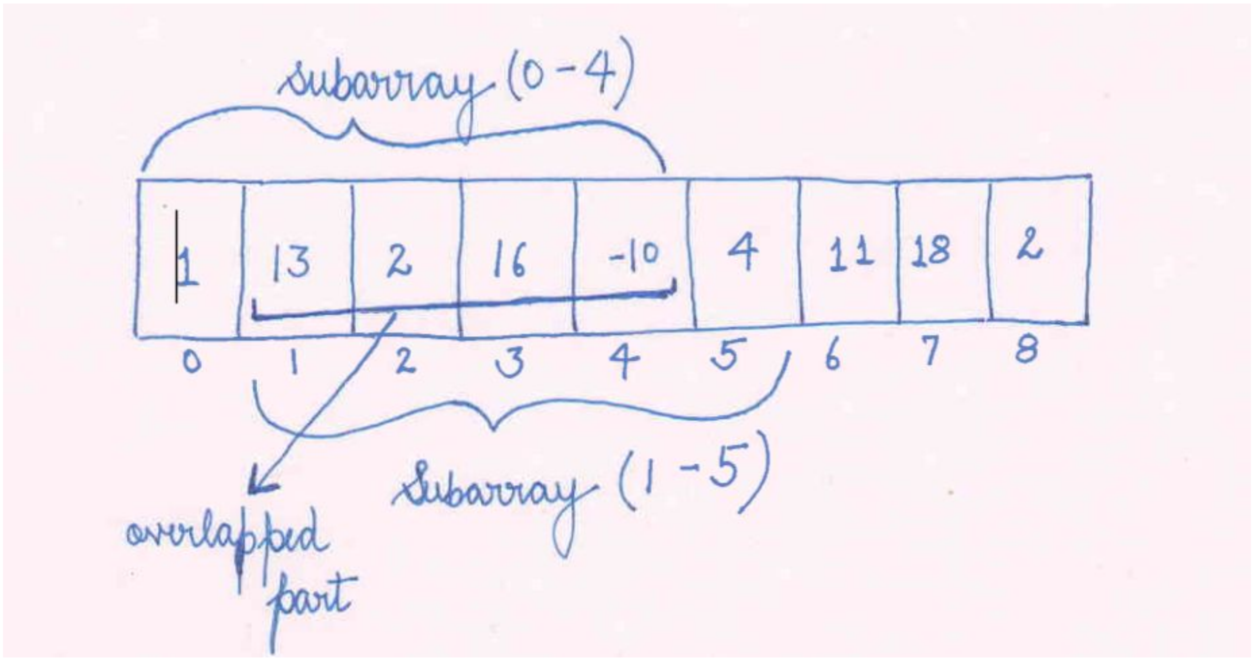
1. The first five sized subarray, starting from index 0, is [1, 13, 2, 16, -10] and it's sum is 22.
2. The next five sized subarray, starting from index 1, is [13, 2, 16, -10, 4] and it's sum is 25.
3. The next five sized subarray, starting from index 2, is [2, 16, -10, 4, 11] and it's sum is 23.

…….
Hence, for this particular test case, maximum of all the subarrays of size five is 39 for the subarray: [16, -10, 4, 11, 18].
A brute force algorithm will be to calculate the sum for every five sized subarray and find maximum of all the sums. This is what algorithm will look like:

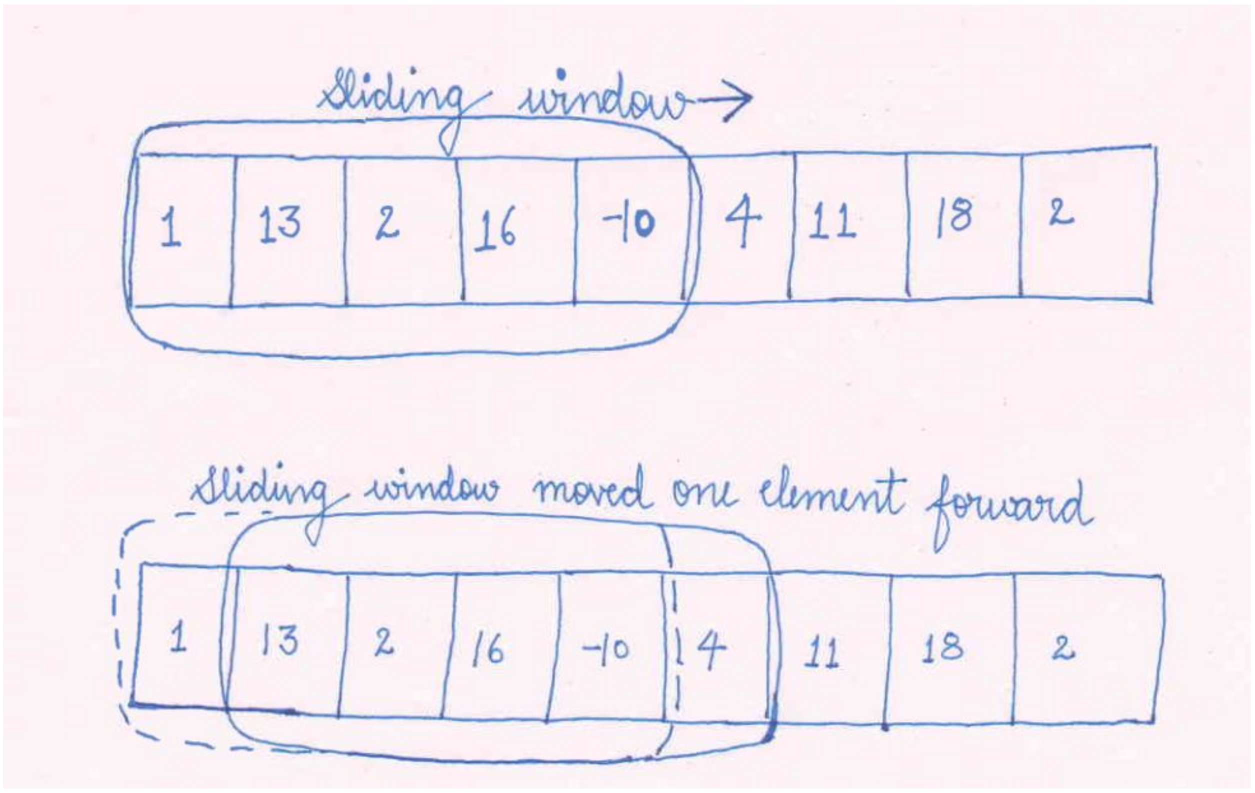
C++:
```
#include <iostream>
#include <climits>
using namespace std;
int max_subarrays(int * arr, int n, int k){
    int osum=INT_MIN;
    for(int start_index=0; start_index<=(n-k); start_index++){
        int sum=0;
        for(int looping_index=start_index;
looping_index<start_index+k; looping_index++){
            sum=sum+arr[looping_index];
        }

        if(sum>osum){
                osum=sum;
            }
        }
        return osum;

}
int main(){
    int arr[]={1, 13, 2, 16, -10, 4, 11, 18, 2};
    cout<<max_subarrays(arr, 9, 5);
    return 0;
}
```

Time Complexity: The outer loop runs for the starting elements of every k sized subarrays and the inner loop runs for k times. Hence, the complexity is O(n * k).

Can we do better?
There is certain overlap in two adjacent k-sized subarrays. The inefficiency is that we are traversing the overlapped part again. We are not using the already computed part. For example:



Now, let us try to re-use the overlapped part. The efficient way to solve this problem is to visualize a sliding window of size k. In the beginning, the window is covering the first subarray. We will use a variable sum which will store the sum of the elements in the window. We will slide the window by one element and now, it will cover second subarray. The variable sum will be updated by removing the first element (element forsaken by window) and adding the new element (element taken in the window). This way the variable sum will hold the sum of the elements in the window. This saves us from traversing the whole subarray and hence, the complexity of the algorithm is reduced to O(n).



Here is the algorithm for **Sliding Window** approach:

C++:

```
#include <iostream>
#include <climits>
using namespace std;
int max_subarrays(int * arr, int n, int k){
    int slidingsum=0;
    for(int i=0; i<k; i++){
        slidingsum+=arr[i];
    }
    // overall sum initialed with current sliding sum
    int osum = slidingsum;

    // Let's slide our window by 1 element and start it with second
subarray
    // current window's start and end indices
    int start_index = 1;
    int endindex = k;
    while(endindex < n){
        //removing the starting element of last window
        slidingsum-=arr[start_index-1];

        //adding the last element of current window
        slidingsum+=arr[endindex];

        //slide the window forward
        endindex++;
        start_index++;

        //compare with overall sum
        if(slidingsum > osum){
            osum = slidingsum;
        }
    }


    return osum;

}
int main()
{
    int arr[]={1, 13, 2, 16, -10, 4, 11, 18, 2};
    cout<<max_subarrays(arr, 9, 5);
    return 0;
}
```

There are more variations to sliding window questions. Sometimes, we have to shrink or expand the size of sliding window. Let us jump to solve more questions on sliding window pattern to better our comprehension.