Sliding Window

The sliding window technique is useful for solving problems in arrays or strings. Generally, it is considered as a technique that could reduce the time complexity from $O(n^2)$ to $O(n)$.

Sliding Window Technique is a method for finding subarrays in an array that satisfy given conditions. We do this via maintaining a subset of items as our window and resize and move that window within the larger list until we find a solution.

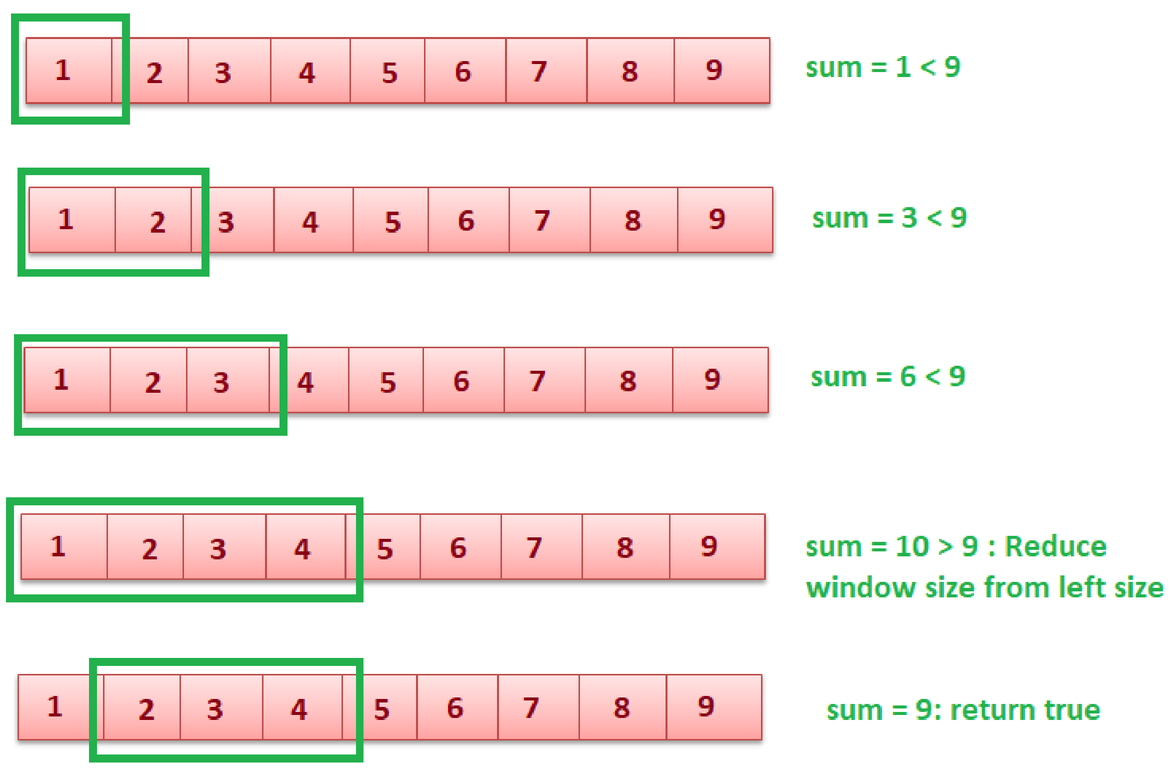
There are two types of sliding window:

1. Fixed window length k: the length of the window is fixed and it asks you to find something in the window such as the maximum sum of all windows, the maximum or a median number of each window. Usually, we need some kind of variables to maintain the state of the window, some are as simple as an integer or it could be as complicated as some advanced data structure such as list, queue, or deque.
2. Two pointers + criteria: the window size is not fixed, usually it asks you to find the subarray that meets the criteria, for example, given an array of integers, find the number of subarrays whose sum is equal to a target value.

Let's understand it by an example:

- EXAMPLE: Given an array of positive integers, find a subarray that sums up to target. Let the array be [1, 2, 3, 4, 5, 6, 7, 8, 9] and target be 9.

We will start with window size = 1, then keep increasing the window size until the sum of elements inside the window is greater than or equal to the target. If the sum equals to target return true else decrease the window size from the left and reduce the sum till it is less than or equal to the target.



```
function countSubarrays(arr, target)
    count =0 ;
            curr_sum = arr[0], start = 0, i = 1

    //  Pick a starting point
    while i <= arr.length
        //  If curr_sum exceeds the target, then remove the starting elements
        while curr_sum > target AND start < i - 1
            curr_sum = curr_sum - arr[start]
            start += 1

        //  If curr_sum becomes equal to target,  then return true
        if curr_sum == target
            p = i - 1
            print("subarray found between start and p")
            return 1

        //  Add this element to curr_sum
        if i < n
                curr_sum = curr_sum + arr[i]
        i += 1
    print(“no subarray found”)
    return 0
```

Time Complexity: O(n), only one traversal of array is needed.