DSA SERIES

- Learn Coding

Topic to be Covered today

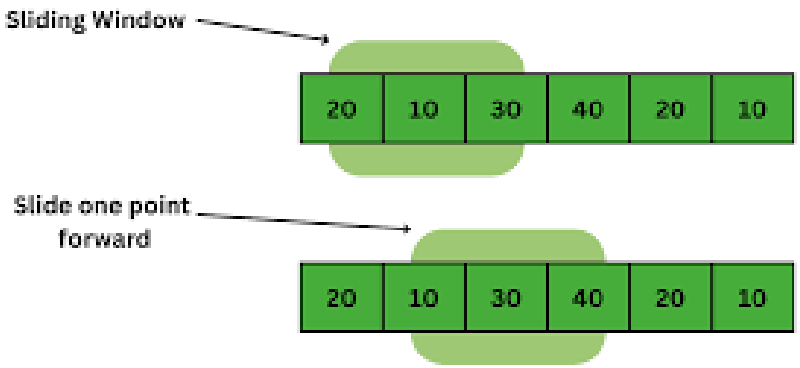
Sliding Window

LETS START TODAY'S LECTURE

Sliding Window

Sliding Window is a technique to efficiently process **a subrange (window)** of elements in an array or string, especially when:

- You're moving through **contiguous elements**
- You want to **optimize performance** from $O(N^2)$ to $O(N)$



Types of Sliding Window :

- **Fixed size**
- **Dynamic size**
- **Two pointers**

Find max sum of any subarray of size K

```
#include<iostream>
#include<vector>

using namespace std;


int maximumSum(vector<int> &nums,int k){
  int sum = 0 ,maxSum = 0;

  // Find the sum of first window
 for(int i =0 ;i<k;i++){
    sum+=nums[i];
 }

 maxSum = sum;

 //  Remaining window element ke liye loop run karke value find karo
```

```
 for(int i =k;i<nums.size();i++){
    sum += nums[i]-nums[i-k];
    maxSum = max(maxSum,sum);
 }

 return maxSum;

}

int main(){
    vector<int> nums = {1,3,2,6,-1,4,1,8,2};
    int k = 3;

    int result = maximumSum(nums,k);

    cout<<"The result is : " <<result;



}
```

Maximum Sum of Distinct Subarray with length K (2461)

```
class Solution {
public:
    long long maximumSubarraySum(vector<int>& nums, int k) {
      int n = nums.size();
      long long result =0;
      long long sum =0;

      unordered_set<int> st;
      int i =0,j=0;

      while(j<n){
        while(st.count(nums[j])){
            sum -=nums[i];
            st.erase(nums[i]);
            i++;
        }

        sum += nums[j];
        st.insert(nums[j]);
```

```
            if(j-i+1==k){
                result =max(result,sum);
                sum -= nums[i];
                st.erase(nums[i]);
                i++;
            }
            j++;
        }

        return result;
    }
};
```

First Negative Number in Every Window of size K

```
class Solution {
  public:
    vector<int> firstNegInt(vector<int>& arr, int k) {

        vector<int> result ;
        int n =arr.size();


        // Outer window
        for(int i =0;i<=n-k;i++){

            bool found = false;


            // to traverse in each window
            for(int j =i;j<i+k;j++){
                if(arr[j]<0){
                    result.push_back(arr[j]);
                    found =true;
                    break;
                }
            }

            if(!found){
                result.push_back(0);
            }


        }

        return result;


    }
};
```

Maximum of all subarrays of size K (239)

```cpp
#include <deque>
class Solution {
public:
    vector<int> maxSlidingWindow(vector<int>& nums, int k) {
        deque<int> dq;
        vector<int> result;
        int n = nums.size();

        for (int i = 0; i < n; i++) {
            // Step 1: Remove elements out of window
            if (!dq.empty() && dq.front() == i - k)
                dq.pop_front();

            // Step 2: Remove smaller elements from back
            while (!dq.empty() && nums[dq.back()] < nums[i])
                dq.pop_back();

            // Step 3: Push current element index
            dq.push_back(i);

            // Step 4: Window ready → push max to result
            if (i >= k - 1)
                result.push_back(nums[dq.front()]);
        }

        return result;
    }
};
```

Maximum average subarrays of size K (643)

```
class Solution {
public:
    double findMaxAverage(vector<int>& nums, int k) {
        int n = nums.size();

        int sum = 0;

        for(int i = 0;i<k;i++){
            sum+=nums[i];
        }

        int maxSum = sum;


        for(int i = k;i<n;i++){
            sum = sum - nums[i-k] + nums[i];
            maxSum = max(maxSum , sum);
        }

        return (double)maxSum/k;

    }
};
```

Longest subarray with sum K

```
#include<iostream>
#include<vector>

using namespace std;

int longest(vector<int> &arr,int k){
    int start  =0 ,sum=0;
    int maxLen =0;
    int n = arr.size();
    for(int end = 0;end<n;end++){
        sum+=arr[end];

        while(sum>k && start<=end){
            sum-=arr[start];
            start++;
        }

        maxLen = max(maxLen,end-start+1);
    }

    return maxLen;
}
```

```
int main(){

    vector<int> arr = { 1,2,1,0,1,1,0};
    int k =4;
    int result = longest(arr,k);

    cout<<"Printing the result : "<<result;
}
```

Learn coding

THANK YOU