SLIDING WINDOW

No.	Problem Statement	Solution	Time complexity	Space complexity
1	**Best Time to Buy and Sell Stock**			

	Given an array prices where prices[i] is the price of a given stock on the ith day. Maximize your profit by choosing a single day to buy one stock and choosing a different day in the future to sell that stock. Return the maximum profit.	- **Sliding Widow:** `l = 0 & r = 0`
		- **l: Keeps track of best 'buying' price in the current window**
		`while(r<n)`
		`if(prices[l]>prices[r]) l++;` // Current `'l'` is not a suitable buying point
		`else{`
		`if(prices[r]>prices[l]) ans = max(ans, prices[r]-prices[l]);`
		`r++;`
		`}`
			O(N)	O(1)

2	**Longest Substring Without Repeating Characters**			
	Given a string s, find the length of the longest substring without repeating characters.	- **Sliding Window** : `l = 0 & r = 0` **+ Unordered_set** - Keep inserting s[r] to the set until duplicate is found - If duplicate, remove `s[l]` from the set and increase `l` - Else `r++`	O(N)	O(N)
3	**Longest Repeating Character Replacement**			
	Given a string s and an integer k, select any character of the string and change it to any other uppercase English character. You can perform this operation at most k times. Return the length of the longest substring containing the same letter. Ex: s = "ABAB" k = 2 -> 4	- **Sliding Window**: `l=0 & r = 0` - Vector of size 26 to count **frequency** of each character in the current window - **Idea:** To keep track of the no. of characters to be **flipped** in current window - `max_freq` : To track the frequency of most repeated character in the current window - `flip = (r-l+1) - max_freq` // Number of characters to be flipped - `if(flip<=k) ans=max(ans, (r-l+1)); r++` - `else cnt[s[l]-'A']--; l++`	O(N)	O(N)
4	**Permutation In String**			

	Given two strings s1 and s2, return true if s2 contains a permutation of s1, or false otherwise. Ex. s1 = "ab", s2 = "eidbaooo" -> true, s2 contains "ba"	- **Sliding Window** : `l=0 & r = 0`
		- Vector of size 26 to count **frequency** of each character in s1
		- **Idea**: Maintain the length of window == s1.size()
		initialize, `k = n1`
		`while(r<n2)`
		`if(cnt[s2[r]-'a']>0) k--;`
		// Check if permutation of s1 is found
		`if(k == 0) return true;`
		`cnt[s2[r]-'a']--;`
		// If the current window_size == s1.size(), move the window by incrementing 'l'
		`if(r-l+1 == n1){`
		`cnt[s2[l]-'a']++;`
		`if(cnt[s2[l]-'a']>0) k++;`
		`l++;`
		`r++;`
			O(N)	O(N)

5	**Minimum window Substring**			

SLIDING WINDOW

No.	Problem Statement	Solution	Time complexity	Space complexity

	Given 2 strings s & t, return min window substring of s such that all characters in t are included in window.
	Ex. s = "ADOBECODEBANC" t = "ABC" -> "BANC"
		- **Sliding window + Unordered_map** --> **Keep count of frequency of characters in 't' using unordered_map**
		- Traverse string 's' using sliding winow
		- Initialize, `cnt = 0`
		`while(r<n1)`
		`if(mp.find(s[r]) != mp.end() && mp[s[r]] > 0) cnt++;`
		`mp[s[r]]--;`
		`while(cnt == n2){ // 'while' instead of 'if'`
		`if((r-l+1)<length)`
		`start = l;`
		`length = r-l+1;`
		`mp[s[l]]++;`
		`if (mp[s[l]] > 0) cnt--;`
		`l++;`
		`r++;`
			O(N+M)	ON)

6	**Sliding Window Maximum**			

	You are given an array of integers nums, there is a sliding window of size k which is moving from the very left of the array to the very right. You can only see the k numbers in the window. Each time the sliding window moves right by one position.
	Return the max element in the current window.
	Ex. nums = [1,3,-1,-3,5,3,6,7] k = 3 -> [3,3,5,5,6,7]
		- **Sliding window + Dequeue** --> **To keep track of indices in the current window**
		- In dequeue, the index of the max element in the current window is always at the front and `dq.size() always <= k`
		`while(r<nums.size())`
		`// Remove dq.front() if it's out of current window`
		`while(!dq.empty() && dq.front()<l) dq.pop_front();`
		`// Remove all elements that are less than current element`
		`// as those elemnts can never be the max element in the current window`
		`while(!dq.empty() && nums[r] > nums[dq.back()]) dq.pop_back();`
		`dq.push_back(r);`
		`if((r-l+1)==k)`
		`ans.push_back(nums[dq.front()]);`
		`l++;`
		`r++;`
			O(N)	O(K)