# Day 12 / 100 :

## Topic - Arrays

### 1 Problem statement: Minimum size subarray sum (Medium)

Given an array of positive integers nums and a positive integer target, return the minimal length of a subarray whose sum is greater than or equal to target. If there is no such subarray, return 0 instead.

Example 1:

Input: target = 7, nums = [2,3,1,2,4,3]
Output: 2
Explanation: The subarray [4,3] has the minimal length under the problem constraint.
Example 2:

Input: target = 4, nums = [1,4,4]
Output: 1
Example 3:

Input: target = 11, nums = [1,1,1,1,1,1,1,1]
Output: 0

## Solutions :

**Approach 1 - Using sorting O(n Log n)**

```
class Solution {
public:
    int minSubArrayLen(int s, vector<int>& nums) {
        int n = nums.size(), len = INT_MAX;
        vector<int> sums(n + 1, 0);
        for (int i = 1; i <= n; i++) {
            sums[i] = sums[i - 1] + nums[i - 1];
        }
        for (int i = n; i >= 0 && sums[i] >= s; i--) {
            int j = upper_bound(sums.begin(), sums.end(), sums[i] -
s) - sums.begin();
```

```
            len = min(len, i - j + 1);
        }
        return len == INT_MAX ? 0 : len;
    }
};
```

## Solutions :

**Approach 1 - Using sorting O(n Log n)**

The O(n) solution is to use two pointers: l and r. First we move r until we get a sum >= s, then we move l to the right until sum < s. In this process, store the minimum length between l and r. Since each element in nums will be visited by l and r for at most once. This algorithm is of O(n) time.

```
class Solution {
public:
    int minSubArrayLen(int s, vector<int>& nums) {
        int l = 0, r = 0, n = nums.size(), sum = 0, len = INT_MAX;
        while (r < n) {
            sum += nums[r++];
            while (sum >= s) {
                len = min(len, r - l);
                sum -= nums[l++];
            }
        }
        return len == INT_MAX ? 0 : len;
    }
};
```

## 2 Problem statement: Longest substring without repeating characters

(Medium)

Given a string s, find the length of the longest substring without repeating characters.

Example 1:

Input: s = "abcabcbb"

Output: 3
Explanation: The answer is "abc", with the length of 3.
Example 2:

Input: s = "bbbbb"
Output: 1
Explanation: The answer is "b", with the length of 1.

## <u>Solutions</u> :

**Approach 1 - Sliding window**

**Intuition**
We have to find the length of longest substring which does not contain any repeating characters
The first thing which should come in our mind is to traverse in the string and store the frequence of each character in a map type of "map<char ,int>" and try to maintain frequency of all the characters present in the map to 1 and storing the maximum length

**Approach**
1:- We will first iterate over the string and store the frequencies of all the character we have visited in a map of type "map<char ,int>"

2:- In each iteration we will check if the frequency of character at ith index is greater than one.....if yes it means that that character is repeated twice in our map so we will start erasing all the characters from the starting jth index untill we delete the same character which is repeated twice...... from the left

ex :-

```
 Input: s = "abcabcbb"

i=0 mp:{[a:1]}
|| length =1 , maxlength = 1

i=1 mp:{[a:1] , [b:1]}
|| length =2 , maxlength = 2

i=2 mp:{[a:1] , [b:1] , [c:1]}
|| length =3 , maxlength = 3
```

```
i=3 mp:{[a:2] , [b:1] , [c:1]}
|| length =4 , maxlength =3
```

'a' appeared 2 times , so we will start removing characters from the left untill the frequency of 'a' becomes 1 again.
while(mp[a]>1) , we will decrease the frequency of all the characters from left i.e. "mp[s[j++]]--" ......Variable j stores the starting index of our subarray , Initially it is 0 but as we start deleting characters from left , this j will be get increment....that is why we did j++ there.
when j=0 , mp[s[j++]] will get decrease by one and the frequency of a will again be equal to 1

```
i=3 mp:{[a:1] , [b:1] , [c:1]}
```
|| length =3 , maxlength =3

3:- In each iteration we will take the maximum of (maxlength ,length) and store it in max length

This process will continue till we reach the end of the string and will return maxlength which is the length of longest substring with no repeating characters in it

```
    int lengthOfLongestSubstring(string s) {
        int length=0 , maxlength=0,j=0;
        map<char ,int> mp;
        for(int i=0 ;i<s.size(); i++){
            mp[s[i]]++;
            length++;
                while(mp[s[i]]>1){
                    mp[s[j++]]--;
                    length--;
                }
            maxlength = max(maxlength,length);
        }
        return maxlength;
    }
```