Day 26 / 100 :

Topic - Stack, Queue

1 Problem statement: Asteroid Collision (Medium)

We are given an array of asteroids of integers representing asteroids in a row.
For each asteroid, the absolute value represents its size, and the sign represents its direction (positive meaning right, negative meaning left). Each asteroid moves at the same speed.

Find out the state of the asteroids after all collisions. If two asteroids meet, the smaller one will explode. If both are the same size, both will explode. Two asteroids moving in the same direction will never meet.

Example 1:
Input: asteroids = [5,10,-5]
Output: [5,10]
Explanation: The 10 and -5 collide resulting in 10. The 5 and 10 never collide.

Example 2:
Input: asteroids = [8,-8]
Output: []
Explanation: The 8 and -8 collide exploding each other.

Example 3:
Input: asteroids = [10,2,-5]
Output: [10]
Explanation: The 2 and -5 collide resulting in -5. The 10 and -5 collide resulting in 10.

Solutions :
Approach 1 - using stack

Intuition:

According to the question, positive values are moving to the right and negative values are moving to the left. We can apply the concept of relative velocity and make positive values as fixed and negative values now moving with double velocity in the negative direction. But, magnitude of velocity does not matter only the direction matters.

Idea:

Lets consider an example:-

[8, 9, 6, -7, -9, 12, 50, -34]

Start iterating from the beginning. Whenever we encounter a positive value, we don't have to do anything. Since they are fixed, they won't attack anyone. But the moment we sees a negative value, we are sure it is going to attack positive values.

Imagine [8, 9, 6] are happily sitting inside the array. The moment -7 enters, it will start knocking out positive values. This gives an idea we can use a stack to solve this problem.

Explanation:

Lets see how to use this idea to code.

Consider the same example [8, 9, 6, -7, -9, 12, 50, -34]
Initially i = 0.

- Whenever we encounter a positive value, we will simply push it to the stack.
- The moment we encounter a negative value, we know some or all or zero positive values will be knocked out of the stack. The negative value may itself be knocked out or it may enter the stack.
- We will keep on poping the elements from the stack while the asteroids[i] > s.top(). But remember, negative asteroids can never pop another negative asteroids, since they will never meet them. So, the final condition is while(!s.empty() and s.top() > 0 and s.top() < abs(ast[i])), we will pop the elements.
- We have to take care of the case when s.top() == asteroids[i]. In this case one positive element will pop out and negative asteroid won't enter the stack.
- If after knocking out elements stack becomes empty() or s.top() becomes negative, that means the current asteroids[i] will enter the stack.

```
class Solution {
public:
    vector<int> asteroidCollision(vector<int>& ast) {
        int n = ast.size();
        stack<int> s;
        for(int i = 0; i < n; i++) {
            if(ast[i] > 0 || s.empty()) {
                s.push(ast[i]);
            }
            else {
```

```
                while(!s.empty() and s.top() > 0 and s.top() < abs(ast[i]))
{
                    s.pop();
                }
                if(!s.empty() and s.top() == abs(ast[i])) {
                    s.pop();
                }
                else {
                    if(s.empty() || s.top() < 0) {
                        s.push(ast[i]);
                    }
                }
            }
        }
            // finally we are returning the elements which remains in the
stack.
            // we have to return them in reverse order.
        vector<int> res(s.size());
        for(int i = (int)s.size() - 1; i >= 0; i--) {
            res[i] = s.top();
            s.pop();
        }
        return res;
    }
};
```

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.

Solutions :

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;
    }
```