Trapping Rain Water

BRUTE FORCE :

If we observe carefully the amount the water stored at a particular index is the minimum of maximum elevation to the left and right of the index minus the elevation at that index.

If we find the leftmax and rightmax the min of these can give us the rapped water because if we take the maximum of these then from the min value of the wall the water may escape so that is why we take the min of leftmax,rightmax.

To find total water trapped for every height we will subtract min(leftmax,rightmax)-height[i] and sum up for every height.

```
class Solution {
public:
    int trap(vector<int>& height) {
        int n=height.size();
        int waterTrapped=0;
        for(int i=0;i<n;i++)
        {
            int j=i;
            int leftmax=0,rightmax=0;
            while(j>=0)
            {
                leftmax=max(leftmax,height[j]);
                j--;
            }
            j=i;
            while(j<n)
            {
                rightmax=max(rightmax,height[j]);
                j++;
            }
            waterTrapped+=(min(leftmax,rightmax)-height[i]);
        }
        return waterTrapped;
    }
};
```

- Time Complexity : O(N*N)
- Space Complexity : O(1)

Better Approach : Prefix Sum

We will reduce the time complexity to O(N) by storing leftmax and rightmax for every index in the array and then running a loop again to find min of leftmax,rightmax and difference of this and height of that elevation. Summing up will give the answer.

```
class Solution {
public:
    int trap(vector<int>& height) {
        int n=height.size();
        int waterTrapped=0;
        int leftmax[n],rightmax[n];
        leftmax[0]=height[0],rightmax[n-1]=height[n-1];
        for(int i=1;i<n;i++)
        {
           leftmax[i]=max(leftmax[i-1],height[i]);
        }
        for(int i=n-2;i>=0;i--)
        {
            rightmax[i]=max(rightmax[i+1],height[i]);
        }
        for(int i=0;i<n;i++)
        {

            waterTrapped+=(min(leftmax[i],rightmax[i])-height[i]);
        }
        return waterTrapped;
    }
};
```

- Time Complexity : O(N)
- Space Complexity : O(2N)

Optimal Approach :

We use two pointers left and right if the height at left is less than height at right then left is minimum but for this to be minimum it must satisfy that it is less than leftmax if it is greater then we set leftmax and move ahead once we find a value less than leftmax we add it to the result and for this height the water level is computed so we increment the left pointer.

Vice versa happens when right is lesser.

```
class Solution {
public:
    int trap(vector<int>& height) {
        int n=height.size(),leftmax=0,rightmax=0,left=0,right=n-1,res=0;;
        while(left<right)
        {
            if(height[left]<=height[right])
            {
                if(height[left]>leftmax)
                {
                    leftmax=height[left];
                }
                else
                {
                    res+=leftmax-height[left];
                    left++;
                }
            }
           else
            {
                if(height[right]>rightmax)
                {
                    rightmax=height[right];
                }
                else
                {
                    res+=rightmax-height[right];
                    right--;
                }
            }
        }
        return res;

    }
};
```

- Time Complexity : O(N)
- Space Complexity : O(1)