DAY-35.

①. Trapping Rain Water.

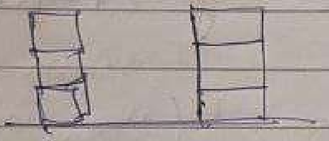
Given non negative ~~numbers~~ integers representing an elevation map where the width of each bar is 1, compute how much water it is able to trap after raining.

Example :-

input : arr = [3, 0, 3]

Output = 3

the structure will look like -



Approach :-

pre compute the highest bar on the left and right of every bar in linear time.

Then, use these pre computed values to find the amount of water in every array elements.

Algorithm :-

→ Create two lists -- left and right of size n.

→ update left[0] with arr[0] (arr is a given array).

→ Run a loop from ~~0~~ 1 to n-1 for indices of arr.
update left [i] as max(left[i-1], arr[i])
→ update right [n-1] (n is the len of arr) with arr[0]
→ Run a loop from end to start.
Update right [i] as max(right[i+1], arr[i])

→ Traverse the lists start to end.
→ The amount of water will be stored in this column is min(a,b) - arr [i] (where a = left [i] and b = right[i]) add this value to total amount ~~of~~ of water stored.

Implementation :-

```
def findWater (arr, n):
    left = [0]*n , right = [0]*n
    water = 0
    left [0] = arr[0]
    for i in range (1, n):
        left = max left([i-1], arr [i])
    ~~right[0]~~ =,
    right [n-1] = arr [n-1]
    for i in range(n-2, -1, -1):
        right [i] = max (right[i+1], arr[i])

    for i in range (0, n):
        water += min(left [i], right[i]) - arr[i]
    return water.
```

②. Largest Sum Contiguous Subarray

Kadane's Algorithm:-

```
Initialize :
    max_so_far = 0
    max_ending_here = 0
Loop for each element of array

    -> max_ending_here += a[i]
    -> if(max_so_far < max_ending_here)
           max_so_far = max_ending_here
    -> if (max_ending_here < 0)
           max_ending_here = 0
return max_so_far
```

Implementation:-

```
def maxSubArraySum (arr):
    maxres = 0
    max now = 0
    for i in arr:
        maxnow += i
        if maxnow > maxres:
            maxres = maxnow
        if maxnow < 0:
            max now = 0
    return maxres.
```