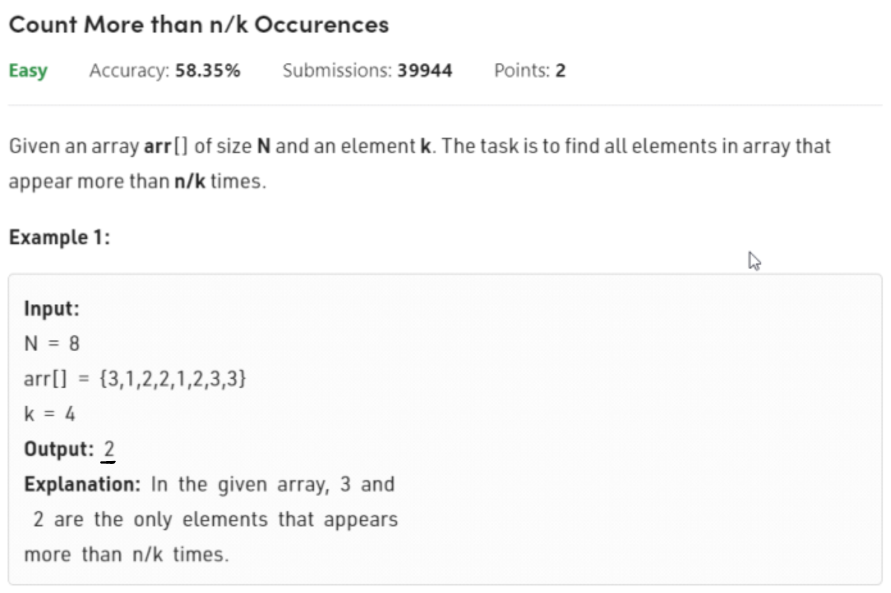
Day 11

21 November 2022 20:34



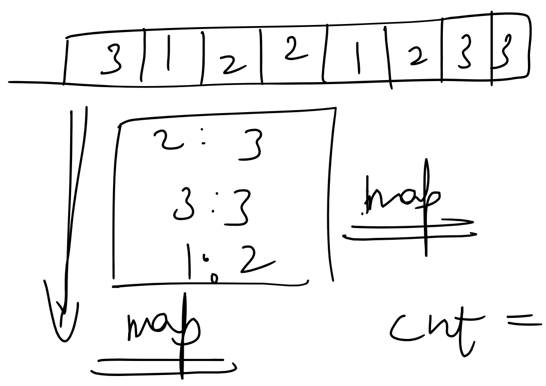
Count More than n/k Occurences

Easy Accuracy: **58.35%** Submissions: **39944** Points: **2**

Given an array **arr**[] of size **N** and an element **k**. The task is to find all elements in array that appear more than **n/k** times.

Example 1:

```
Input:
N = 8
arr[] = {3,1,2,2,1,2,3,3}
k = 4
Output: 2
Explanation: In the given array, 3 and
 2 are the only elements that appears
more than n/k times.
```



N = 8 k = 4

8/4 $2 > f_c$

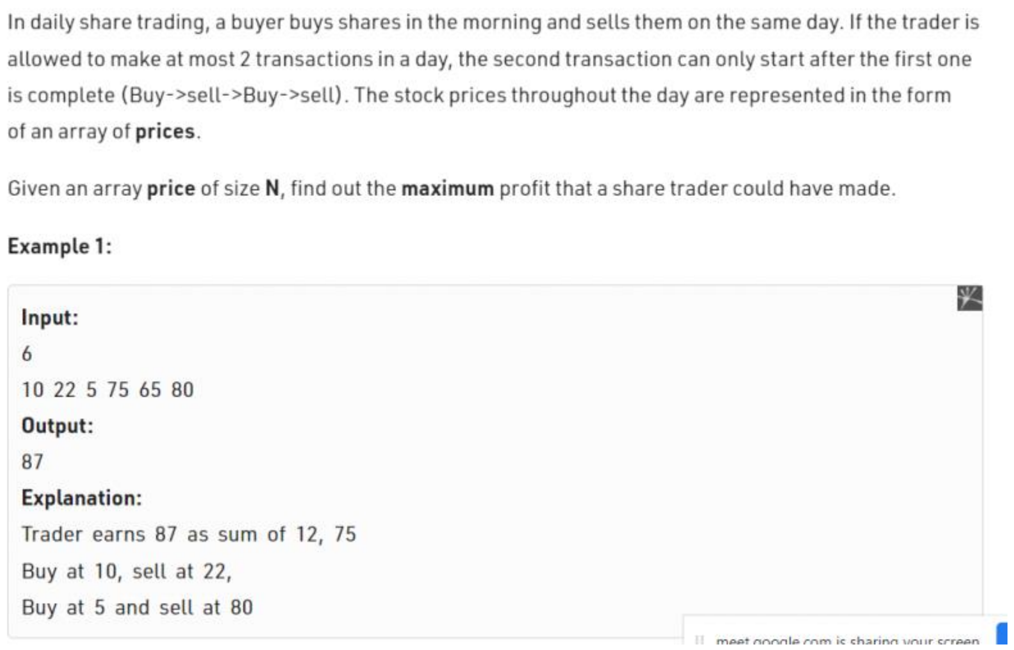
cnt = ~~0~~ ~~1~~ 2

Hashmap map set

Algorithm

Frequency map travels n/k cnt++;

$\left[\begin{array}{l} T.C - O(N) \\ S.C - O(N) \end{array}\right]$



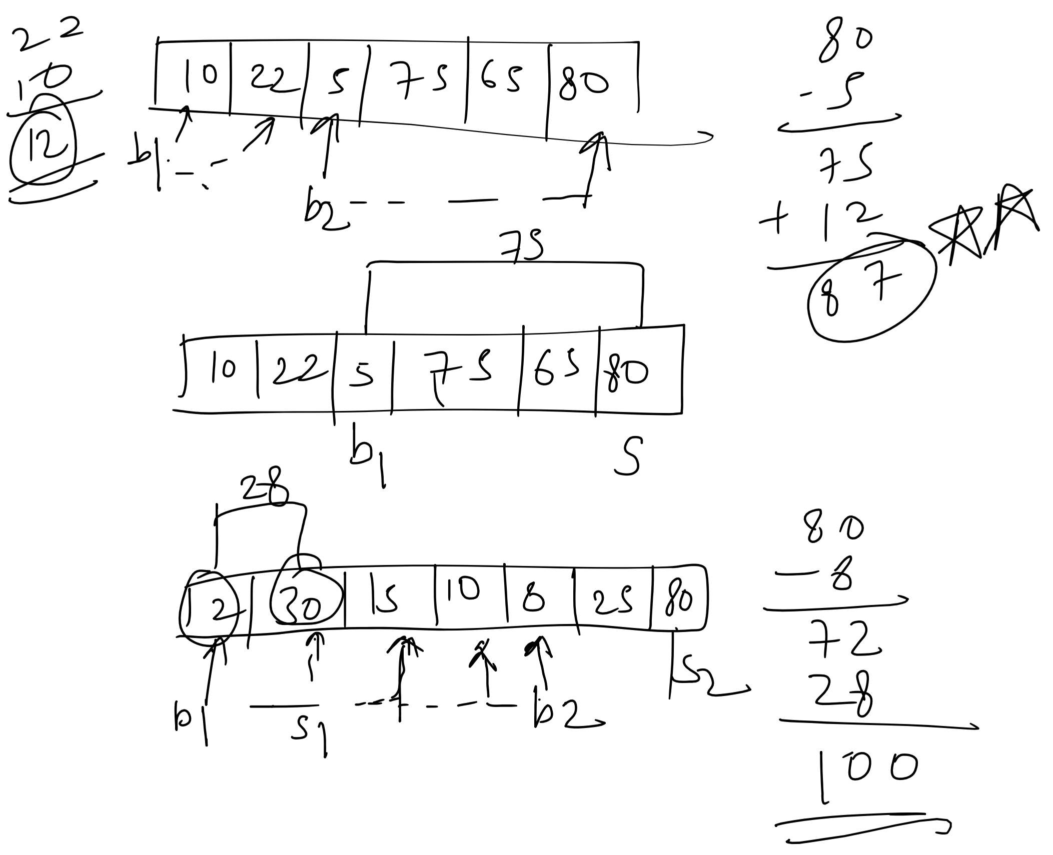
In daily share trading, a buyer buys shares in the morning and sells them on the same day. If the trader is allowed to make at most 2 transactions in a day, the second transaction can only start after the first one is complete (Buy->sell->Buy->sell). The stock prices throughout the day are represented in the form of an array of **prices**.

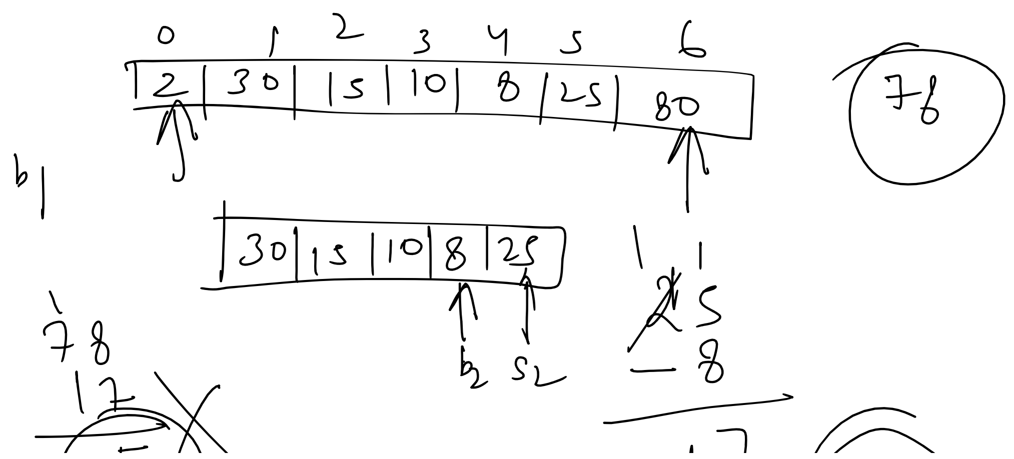
Given an array **price** of size **N**, find out the **maximum** profit that a share trader could have made.

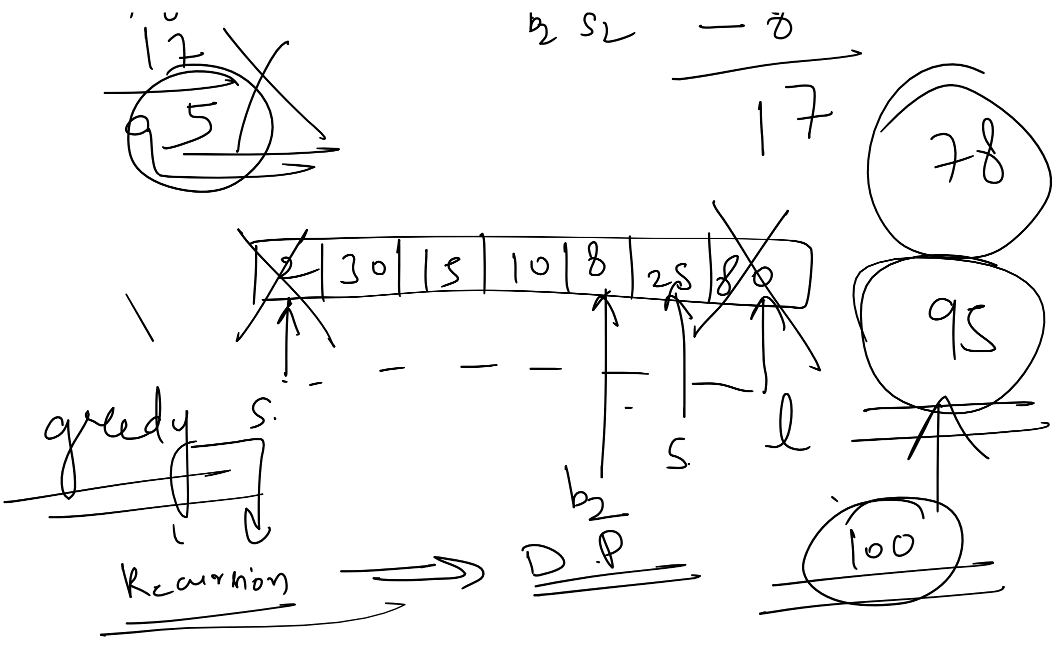
Example 1:

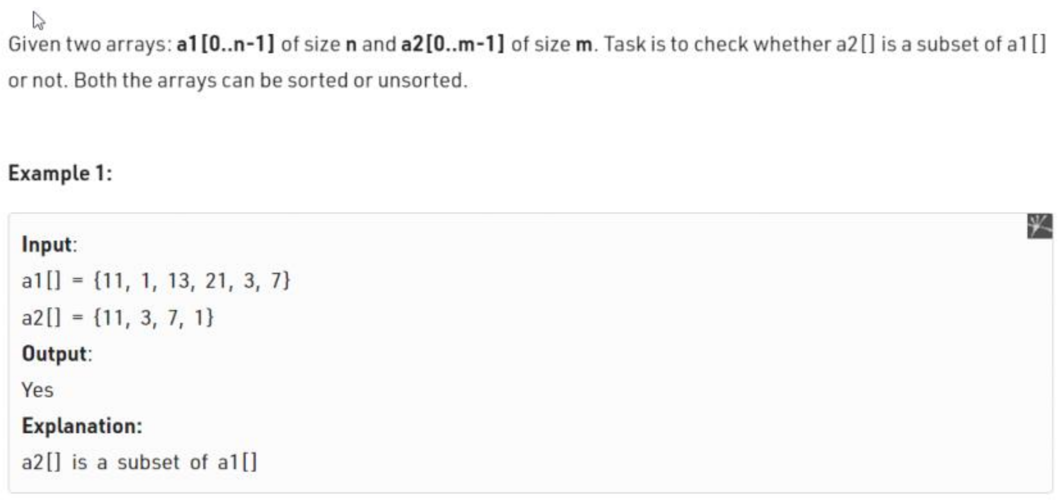
```
Input:
6
10 22 5 75 65 80
Output:
87
Explanation:
Trader earns 87 as sum of 12, 75
Buy at 10, sell at 22,
Buy at 5 and sell at 80
```

meet.google.com is sharing your screen








Given two arrays: **a1[0..n-1]** of size **n** and **a2[0..m-1]** of size **m**. Task is to check whether a2[] is a subset of a1[] or not. Both the arrays can be sorted or unsorted.

Example 1:

```
Input:
a1[] = {11, 1, 13, 21, 3, 7}
a2[] = {11, 3, 7, 1}
Output:
Yes
Explanation:
a2[] is a subset of a1[]
```

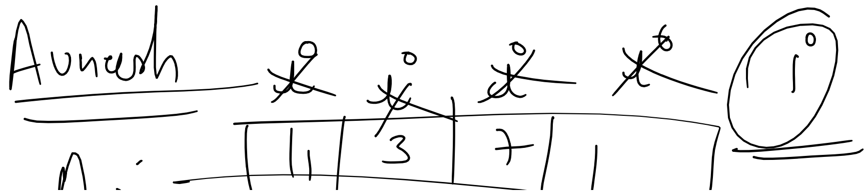
$A_1 = \{\cancel{11}, \cancel{1}, 13, 21, \cancel{3}, \cancel{7}\}$

$A_2 = \{\cancel{11}, \cancel{3}, \cancel{7}, \cancel{1}\}$

$A_2 \supset A_1$

Yes ✓

Approach

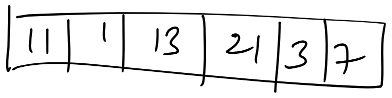


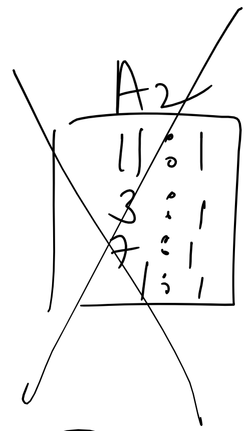
A_2
11
3
7
1
n
A_1
11
1
13
21
3
7
m

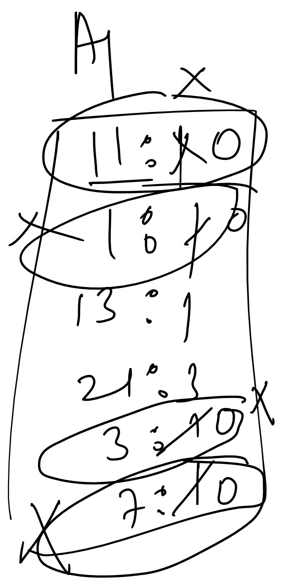
T.C - $(n \times m)$

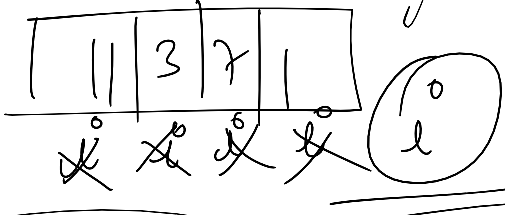
Aaditya T.C $O(n \times \log n)$

Kartik : Hashmap









$A_2 \supset A_1$ Yes

```
string isSubset(int a1[], int a2[], int n, int m) {
    unordered_map<int,int> um;
    for(int i = 0;i < n;i++){
        um[a1[i]]++;
    }
    for(int i = 0;i < m;i++){
        if(um.find(a2[i])!=um.end()){
            um[a2[i]]--;
            if(um[a2[i]] == 0){
                um.erase(a2[i]);
            }
        }
        else{
            return "No";
        }
    }
    return "Yes";
}
```

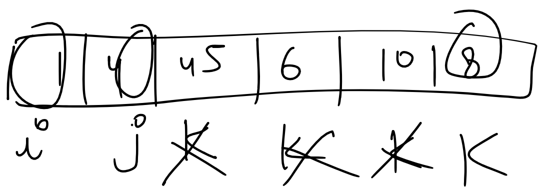
Triplet Sum

Given an array arr of size n and an integer X. Find if there's a triplet in the array which sums up to the given integer X.

Example 1:

Input:
n = 6, X = 13
arr[] = [1 4 45 6 10 8]
Output:
1
Explanation:
The triplet {1, 4, 8} in
the array sums up to 13.

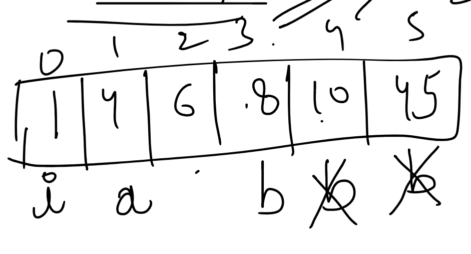
App1



[T.C - $O(n^3)$]

App2

Two Sum → yes



✓

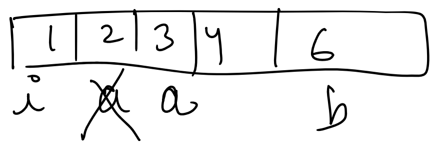
Awneesh

T.C - $O(n^2)$
S.C - $O(1)$

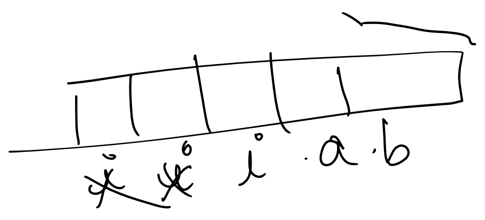
Yes

n=5

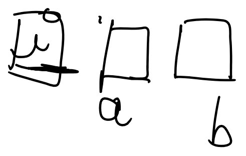
X=10



$O(n^2)$



T.C — O(n^2)



Pseudo

① Sort the array

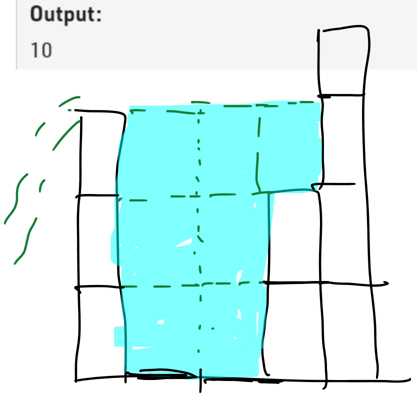
```
sort(A, A+n);
for(int i=0; i<n-2; i++){
    int a = i+1;
    int b = n-1;
    while(a < b){
        if(A[i]+ A[a]+ A[b] == X)
            return 1;
        else if(A[i] +A[a] + A[b] > X){
            b--;
        }
        else{
            a++;
    }   }
}
return 0;
```

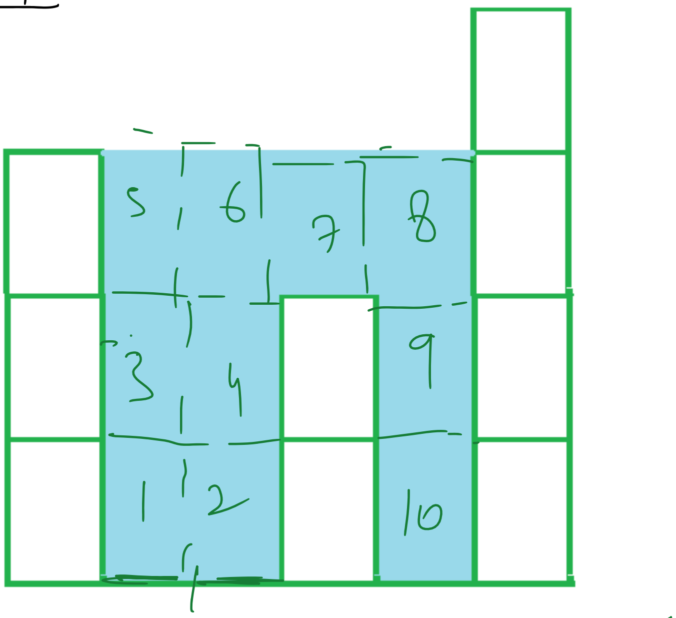
Utkarsh

Given an array **arr[]** of **N** non-negative integers representing the height of blocks. If width of each block is 1, compute how much water can be trapped between the blocks during the rainy season.

Example 1:

```
Input:
N = 6
arr[] = {3,0,0,2,0,4}
Output:
10
```

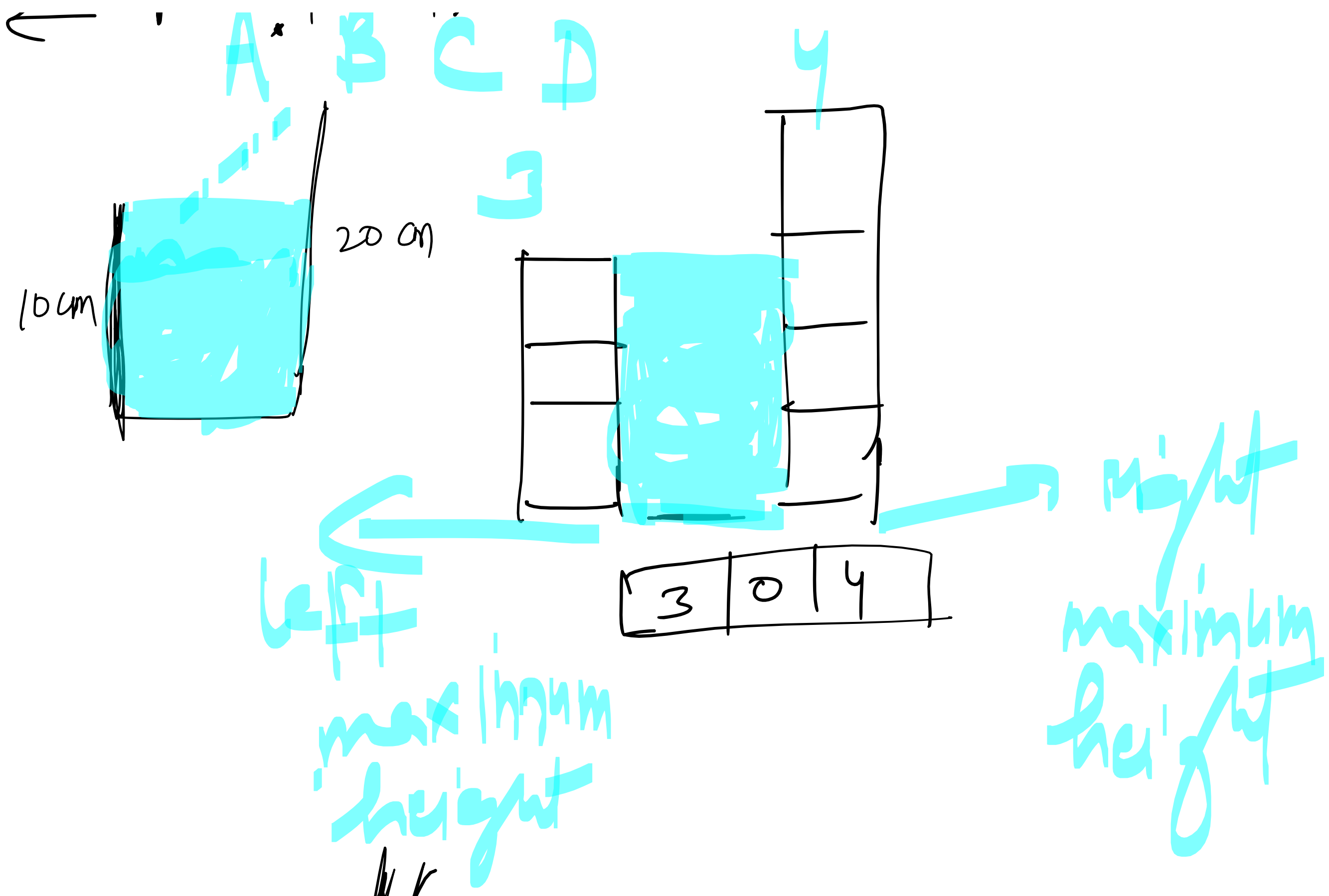




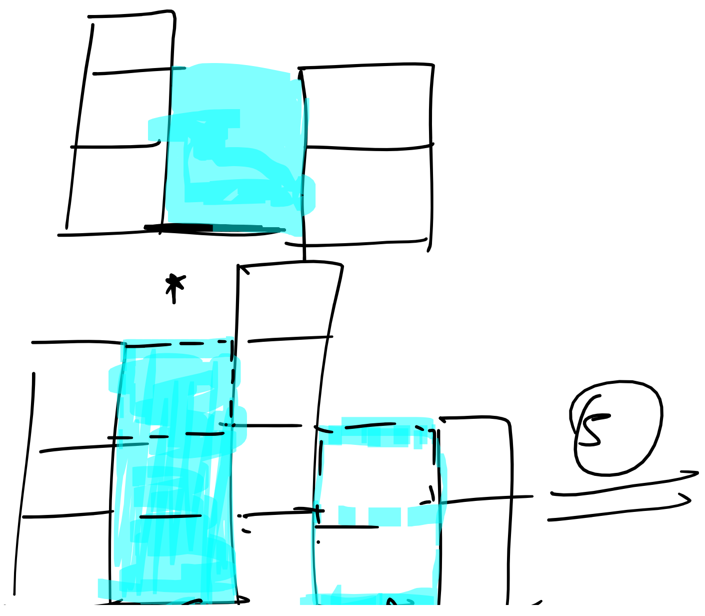
Bars for input {3, 0, 0, 2, 0, 4}
Total trapped water = 3 + 3 + 1 + 3 = 10

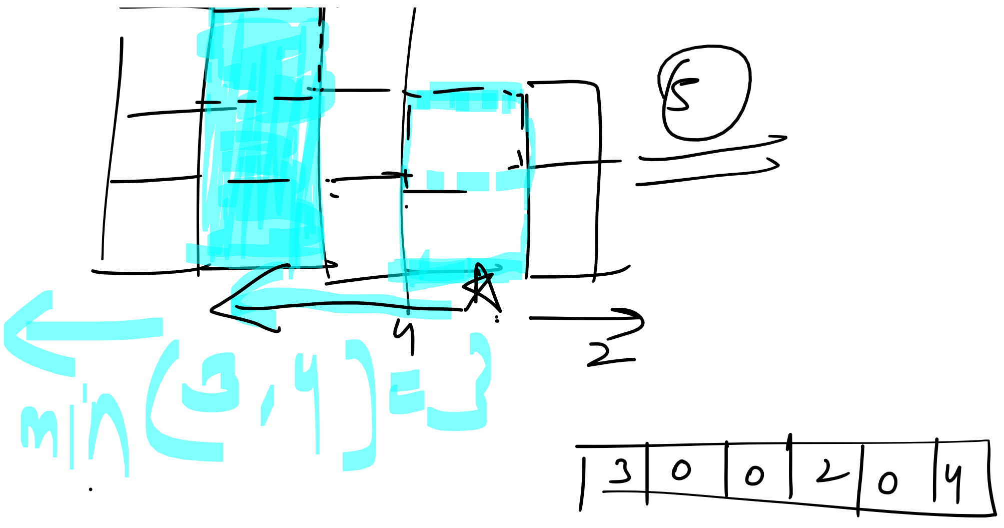
$\min(3,4),\ \boxed{3} - 2 = \boxed{1}$

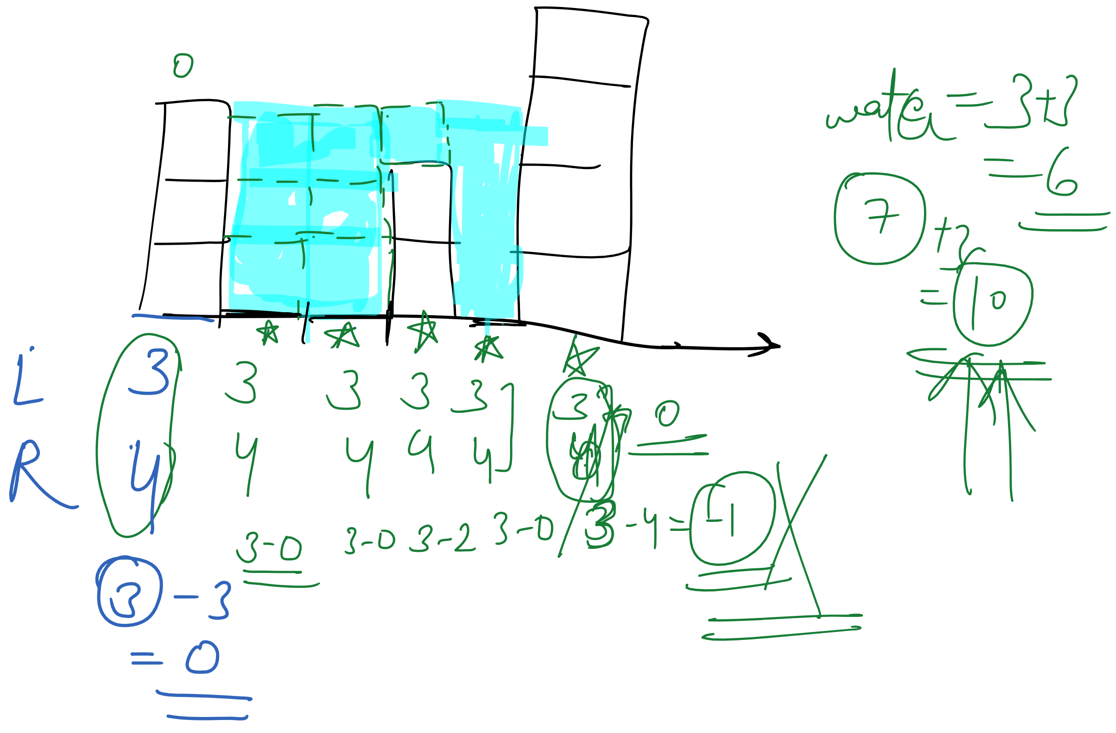
$\min(a,b) - a[i]$

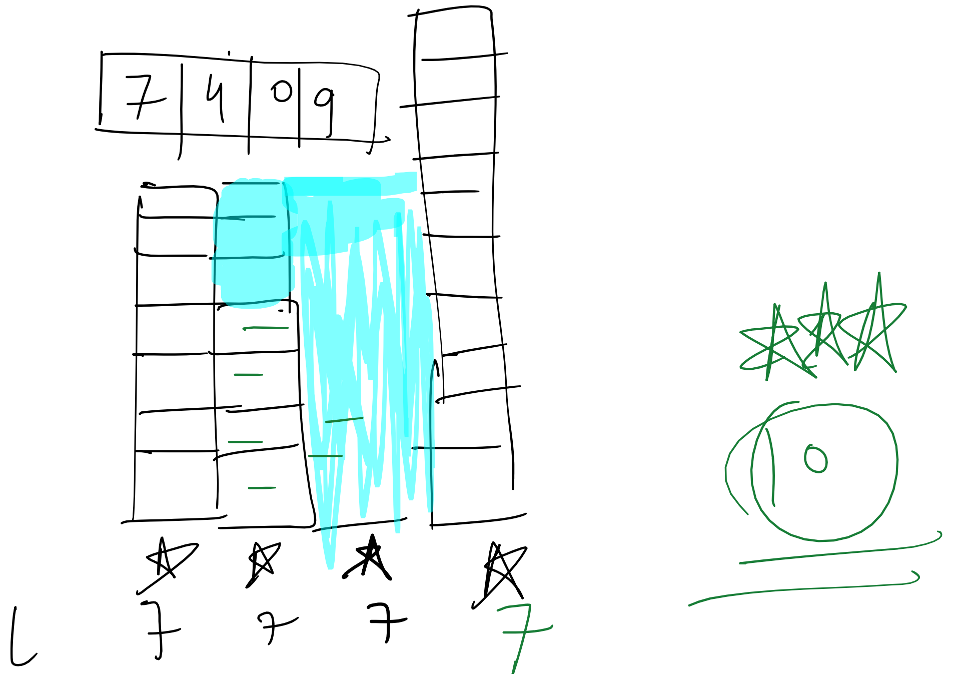

A
B
C
D
y
3
20 cm
10cm
3
0
4
left
maximum height
right
maximum height

10CM
20CM


*
5


5
4
2
min(3,4)=3
| 3 | 0 | 0 | 2 | 0 | 4 |


0
water = 3+3
= 6
7 + 3
= 10
L 3 3 3 3 3 3 0
R 4 4 4 4 4 4
3-0 3-0 3-2 3-0 3-4 = -1
3 - 3
= 0

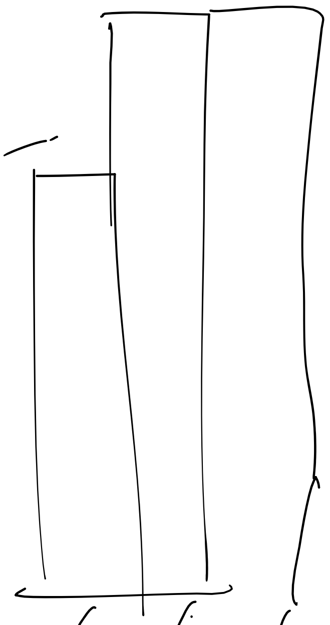

| 7 | 4 | 0 | 9 |
10
L 7 7 7 7

L 7 7 7 7

R 9 9 9 9

9-7 $\boxed{0}$ $\boxed{3}$ $\boxed{7}$ $\boxed{7-9}$ X

EX 2



$\boxed{0}$ || ☆☆

L 6 6 6 | $\boxed{6}$ - 9 = $\boxed{-3}$ X

R 9 9 9

$\boxed{6-6=0}$ $\boxed{-3}$ X

H.W || $\boxed{\text{Done}}$

D.P

```
    // Function to find the trapped water between the blocks.
    public:
    long long trappingWater(int arr[], int n){
        int max_r[n];          — S.C - O(N)
        int max_l[n];
        long long ans = 0;
        int maxi = arr[0];
        max_l[0] = 0;
        for(int i = 1;i < n;i++){
            if(arr[i] > maxi){
                max_l[i] = arr[i];
                maxi = arr[i];
            }
            else{
                max_l[i] = maxi;
            }
        }
        maxi = arr[n-1];
        max_r[n-1] = 0;
        for(int i = n-2;i >= 0;i--){
            if(arr[i] > maxi){
                max_r[i] = arr[i];
                maxi = arr[i];
            }else{
                max_r[i] = maxi;
            }
        }

        for(int i = 0;i < n;i++){
            int temp =  min(max_r[i],max_l[i]) - arr[i];
            ans += temp > 0 ? temp: 0;
        }
        return ans;
    }
};
```

T.C - O(N)

S.C - O(N)