Welcome to Advanced DSA 1 module. ☺

Content For Advanced DSA Module

Arrays — 3.
Bit Manipulation — 2.
Recursion & Backtracking Basics — 3.
Contest - 1 — April 15th 7:00 AM IST

Maths — 3
OOPS — 2
Hashing — 2
Language Advanced Concept — 1
Contest — 2

Sorting — 2
Searching — 3
Two Pointers — 1
Linked List — 3
Contest — 3

Stacks — 2
Queues — 1
Trees — 5
Contest — 4

HashMap Implementation — 1
Heaps — 2
Greedy — 1
DP — 4
Contest — 5

Graphs — 3
Contest (Complete Syllabus — 3 Hrs)
+ Mock Interview (Simulating Real DSA Interview
1 hr 2 Questions)

Q:- Given an integer array A of size N.
Find maximum subarray sum out of all subarrays.

Ex:- {-2, 3, 4, -1, 5, -10, 7}
0 1 2 3 4 5 6

= 11.

Ex:- {-3, 4, 6, 8, -10, 2, 7}
0 1 2 3 4 5 6

= 18.

Quiz 1

{4, 5, 2, 1, 6}
0 1 2 3 4

= 18

Quiz 2

{-4, -3, -6, -9, -2}.

= -2.

Brute Force

N^2

→ Consider all Subarrays

N

→ Iterate over each subarray its calculate sum & compare with global maxSum.

$O(N^3)$.

↓ $O(1)$

Using Prefix Sum →

$O(N^2)$

Optimisation

Possible cases

Case I : All elements are positive

{4, 2, 1, 6, 7} Entire array

Case II : All element are negative

{-4, -2, -1, -6, -7} -Max Element.

Case III : Some positives in between negatives.

{-3, -2, -6, 4, 3, 6, 9, -5, -3, -1, -6}

Case IV : Positive & negatives on either side.

{-3, -2, -6, 5, 3, 4, 6}

Kadane's

Generic Case

-25

{ ---- + , + , + , + , − , − , − , − , + , + , + ---- }.

20 30

15 → 10 → −2.

Ex:- { −2 , 3 , 4 , −1 , 5 , −10 , 7 }.

7 −1 5

= 11

2

1

i

0 1 2 3 4 5 6 7 8

Ex:- { −20 , 10 , −20 , −12 , 6 , 5 , −3 , 8 , −2 }

curSum ∅ ~~−20~~ ∅ 10 ~~−10~~ ∅ ~~−12~~ ∅ 6 11 ~~8~~ ~~16~~ 14.

maxSum ~~INT_MIN~~ ~~−20~~ 10 11 16

Quiz 3:

i

{ −2 , 3 , 4 , −1 , 5 , −10 , 7 }
  0   1   2   3   4   5   6

curSum ~~0 -2 0 3 7 6 11 8~~

maxSum ~~INT_MIN~~ ~~-2 3 7~~ 11

Code

```
int maxSubarraySum(int[] arr, int N){
    int curSum = 0;
    int maxSum = INT_MIN;

    for(i = 0 to N-1){
        curSum += arr[i];

        if(cur_sum > maxSum){
            max_Sum = cur_Sum;
        }

        if(curSum < 0){
            curSum = 0;
        }
    }
    return maxSum
}
```

T.C → O(N)

S.C → O(1)

0 1 2 3

{ -3, -4, -5, -2 }.

i

curSum = 0 -3 -4 -5 -2

maxSum = INT_MIN -3 -2

2. Given an integer array A where every element is 0. Return the final array after performing multiple queries.

Query (i, x) : Add x to all elements from index i.

Queries

	0	1	2	3	4	5	6
	A = { 0	0	0	0	0	0	0 }
(1, 3)		+3	+3	+3	+3	+3	+3
(4, -2)					-2	-2	-2
(3, 1)				+1	+1	+1	+1

A = { 0, 3, 3, 4, 2, 2 2 3 }

Quiz 4

	0	1	2	3	4
A =	{ 0	0	0	0	0 }

Queries

(1, 3)	+3	+3	+3	+3
(0, 2)	+2	+2	+2	+2	+2
(4, 1)					+1

{ 2, 5, 5, 5, 6 }.

0 2 1 2 3 4

$A = \{ \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \}$

$= \{ 2, 5, 5, 5, 6 \}$.

Queries

(1, 3) ✓✓

(0, 2) ✓

(4, 1)

Brute Force

For each query, traverse the array to 'update' the value.

$O(Q * N)$

Optimisation

0	1	2	3	4
A = { 0	0	0	0	0 }
	+3	.	.	.
+2	.	.	.	.
				+1

Queries

(1, 3)
(0, 2)
(4, 1)

A = { 2, 3, 0, 0, 1 }.

pf = { 2 5 5 5 6 }

↑ ans.

Code:

```
for (i=0; i < Q.size(); i++){
    index = Q[i][0];
    value = Q[i][1];
    A[index] += value;
}

for (i=1; i < N; i++){
    A[i] += A[i-1];
}

return A;
```

T.C → O(Q+N).

Variation

Query ⟶ (i , j , x)

Add element x to all the indices b/w i to j.

0	1	2	3	4	5	6
0	0	0	0	0	0	0

A = { 0 , 0 , 0 , 0 , 0 , 0 , 0 }

Queries

(1, 3, 2) +2 +2 +2

(2, 5, 3) +3 +3 +3 +3

(5, 6, -1) -1 -1

ans → { 0 , 2 , 5 , 5 , 3 , 2 , -1 }.

Quiz 5

0	1	2	3	4	5	6	7
0	0	0	0	0	0	0	0

A = { 0 , 0 , 0 , 0 , 0 , 0 , 0 , 0 }

~~i , j , x~~

1 , 4 , 3 +3 +3 +3 +3

0 , 5 , -1 -1 -1 -1 -1 -1 -1

2 , 2 , 4 +4

4 , 6 , 3 +3 +3 +3

ans → { -1 , 2 , 6 , 2 , 5 , 2 , 3 , 0 }.

0	1	2	3	4	5	6	7
A = { 0	0	0	0	0	0	0	0 }
	+3	+3	+3	+3	+3	+3	+3
					-3	-3	-3

i j

1 4 3

1 7 3

A(i) += value;

A[j+1] += -value;

	0	1	2
Q(0)	1	4	3
Q(1)	0	5	-1
Q(2)	2	2	4

Code.

```
for (q=0; q<Q; q++) {
    i = Q[q][0];
    j = Q[q][1];
    value = Q[q][2];

    A[i] += val;
    A[j+1] += -val;   // Check if j+1 is in bound.
}
```

3

```
for (i=1; i<N; i++) {
    A[i] += A[i-1];
}
```

3

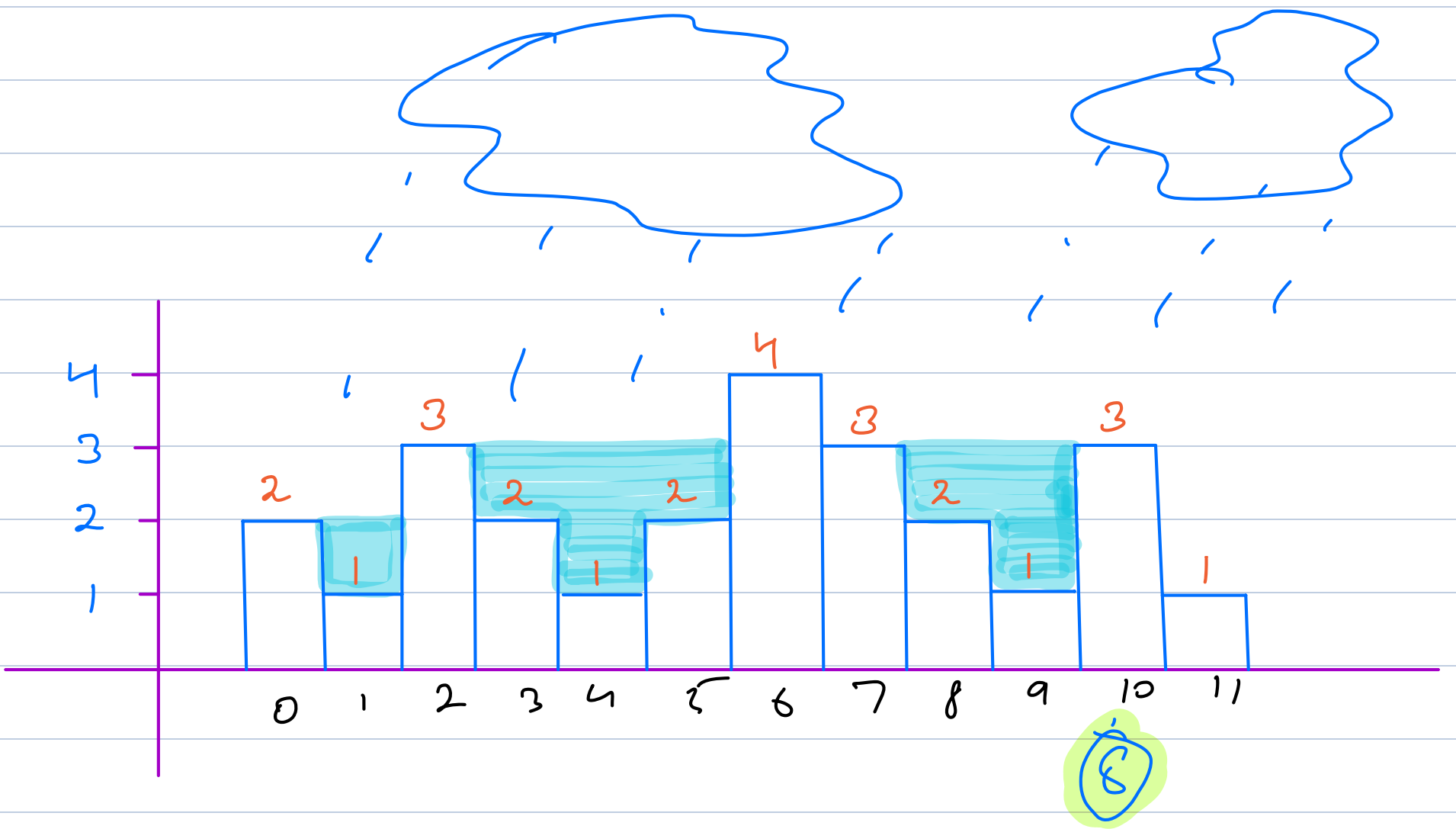
T.C → O(Q+N)

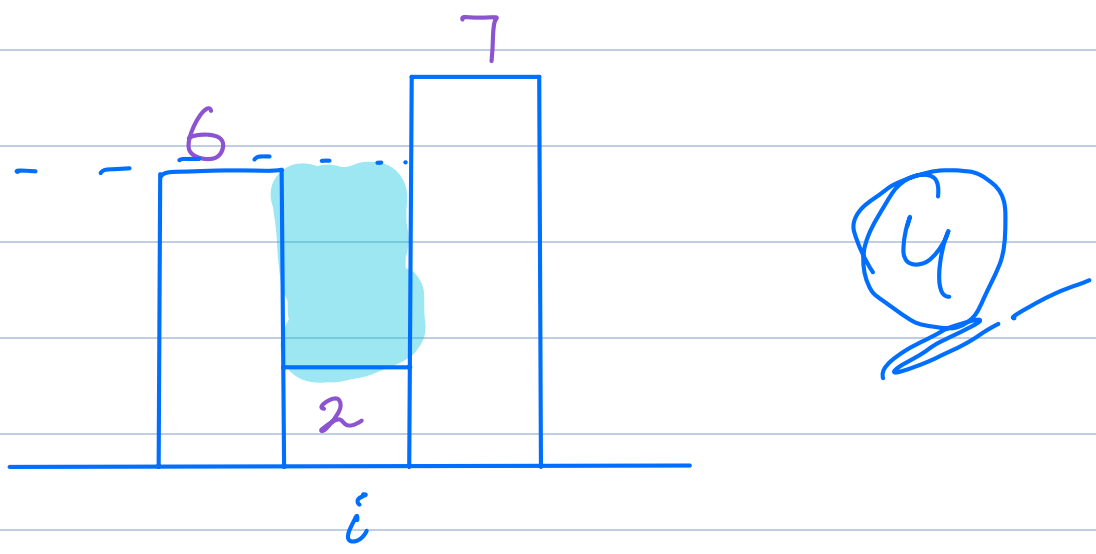
S.C → O(1)

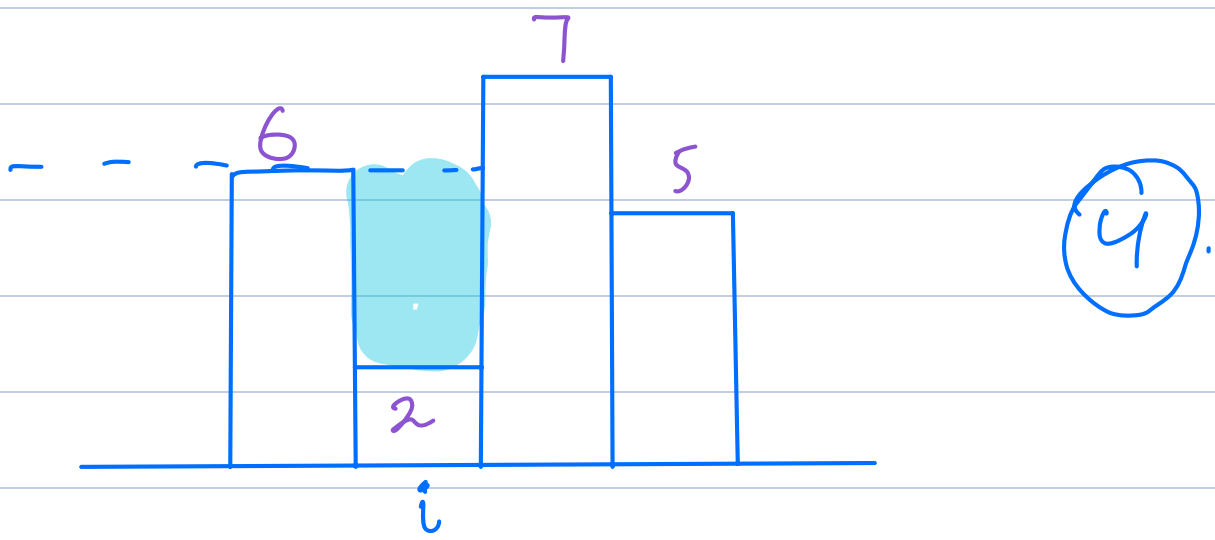
8:45 :

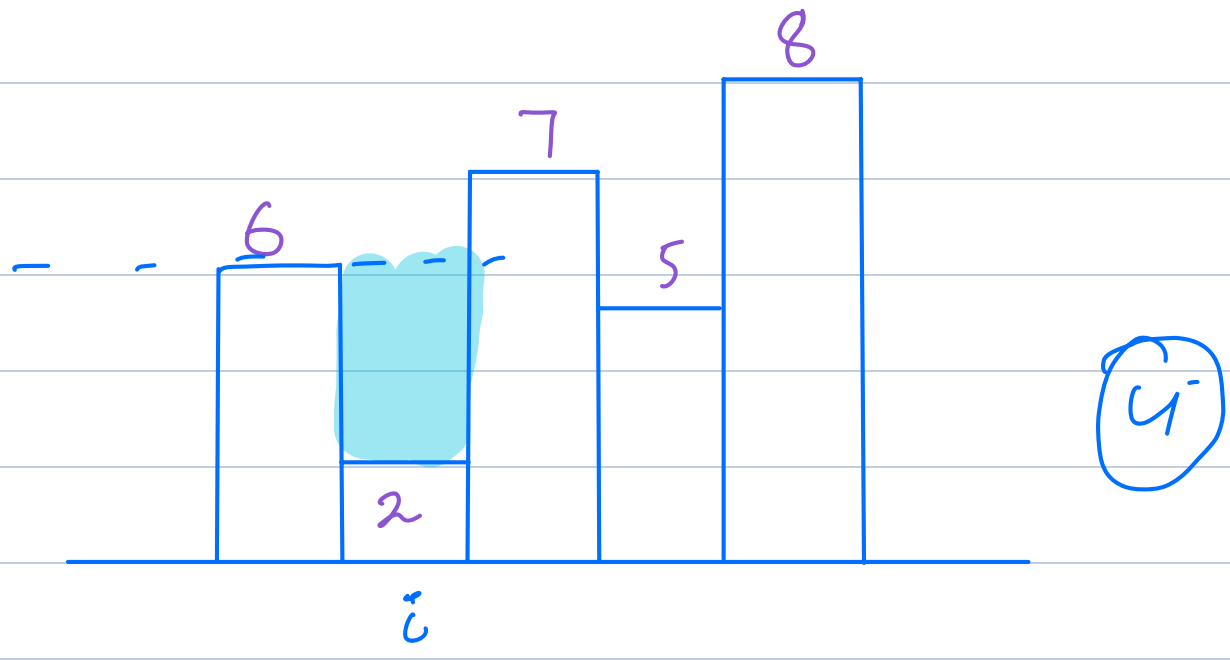
Q:- Given N buildings with height of each building. Find the rain water trapped between the buildings.

arr[] - {2, 1, 3, 2, 1, 2, 4, 3, 2, 1, 3, 1}

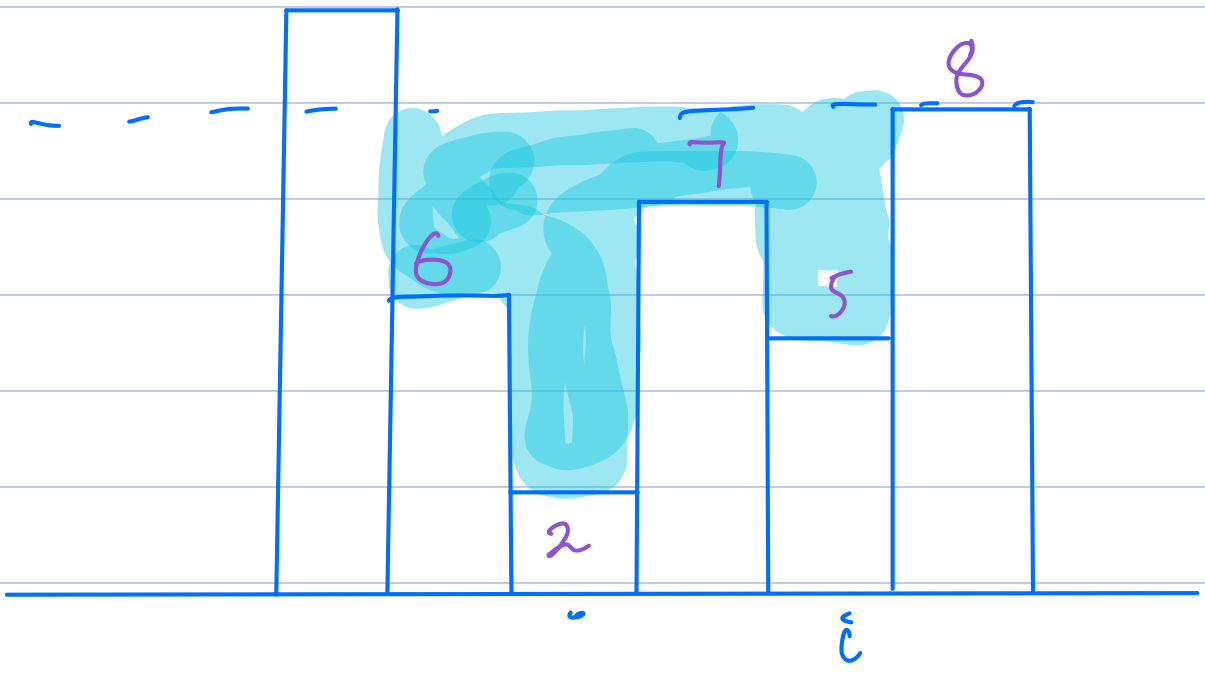




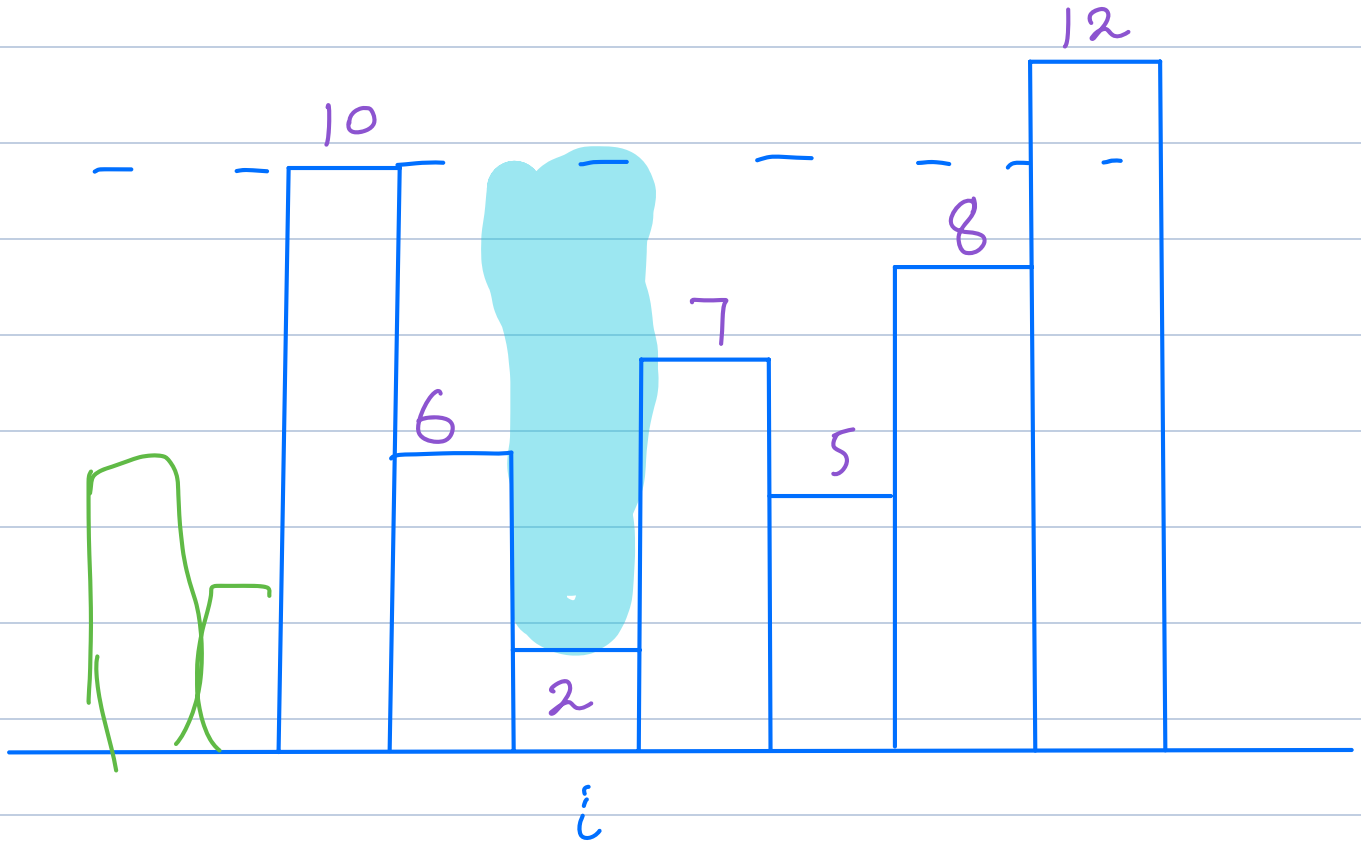




4



6



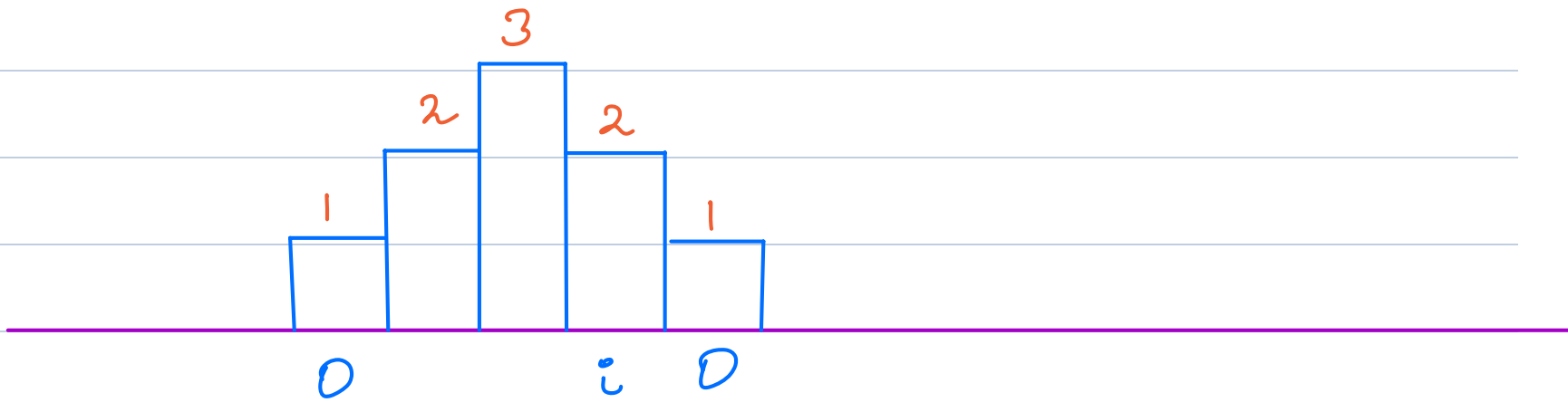
8

Conclusion

The amount of water that will be accumulated over building of index i = $\min(\max ht(0, i-1), \max ht(i+1, N-1)) - ht[i]$;

Quiz 6

ans = 0



maxLeft = 3
maxRight = 1

min = 1

Brute Force

For i^{th} building we need to find max. hts on both the sides.

Code.

```
ans = 0;
for (i = 1; i < N-1; i++) {
```

N {

```
maxLeft = Max. value on
          0 to i-1;

maxRight = Max value on
           i+1 to N-1;

level = min(maxLeft, maxRight)
if( level > ht[i] ){
    ans += level - ht[i];
}
}
return ans;
```

T.C

$O(N^2)$

Optimisation

We can store the lmax & rmax using carry forward approach.

lmax [i] ⟶ Max. element from index 0 to i-1.

rmax [i] ⟶ max. element from index i+1 to N-1.

	0	1	2	3	4	5	6	7	8	9	10
arr -	4	2	5	7	4	2	3	6	8	2	3
lmax []	0	4	4	5	7	7	7	7	7	8	8
rmax []	8	8	8	8	8	8	8	8	3	3	0

lmax [i+1] ⟶ max. value from 0 to i

lmax [i] ⟶ max. value from 0 to i-1;

lmax [i+1] = max (lmax [i] (0 to i-1), arr [i] (i));

Code.

i = 0;

```
lmax[N];
lmax[0] = 0;
for (i = 0; i < N-1; i++) {
    lmax[i+1] = max(lmax[i], a[i]);
}
```

Similarly populate a rmax array.

---

① Create the lmax & rmax arrays.

```
ans = 0;
for (i = 1; i < N-1; i++) {
    maxLeft = lmax[i];
    maxRight = rmax[i];
    level = min(maxLeft, maxRight);
    if (level > h[i]) {
        ans += level - h[i];
    }
}
return ans;
```

$T.C \rightarrow O(N+N)$

$S.C \rightarrow O(N+N)$