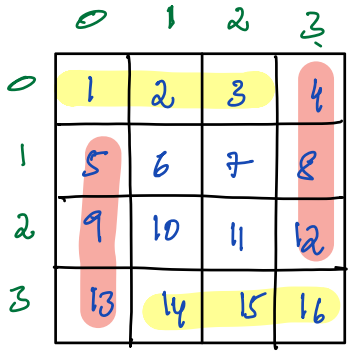
Q) Given a Square Matrix print boundary/Perimeter/ Clock

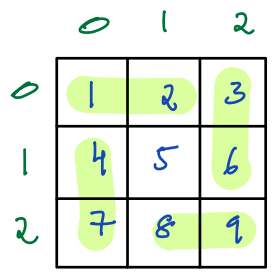
[HARD] → $ ——— }



output:

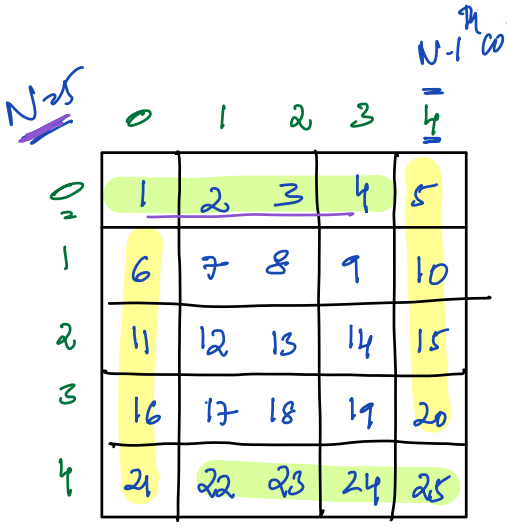
1 2 3 4 8 12 16 15 14 13 9 5

→ N=4 {3H, 3V, 3H, 3V}



output:

1 2 3 6 9 8 7 4



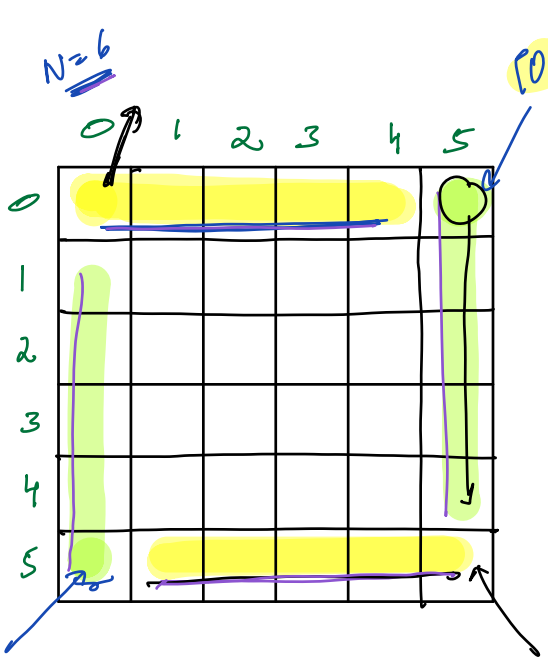
Idea:

Step1: Print 0th Row (L-R) : 4 Steps

Step2: Print last col (T-D) : 4 Steps

Step3: Print last row (R-L) : 4 Steps

Step4: Print 0th Col (D-T) : 4 Steps



Steps

Step1: 0th Row L-R : N-1 steps

Step2: last Col T-D : N-1 steps

Step3: last Row R-L : N-1 steps

Step4: 0th Col D-T : N-1 steps

Pseudocode :→ TC : O(N²) → for Rectangular

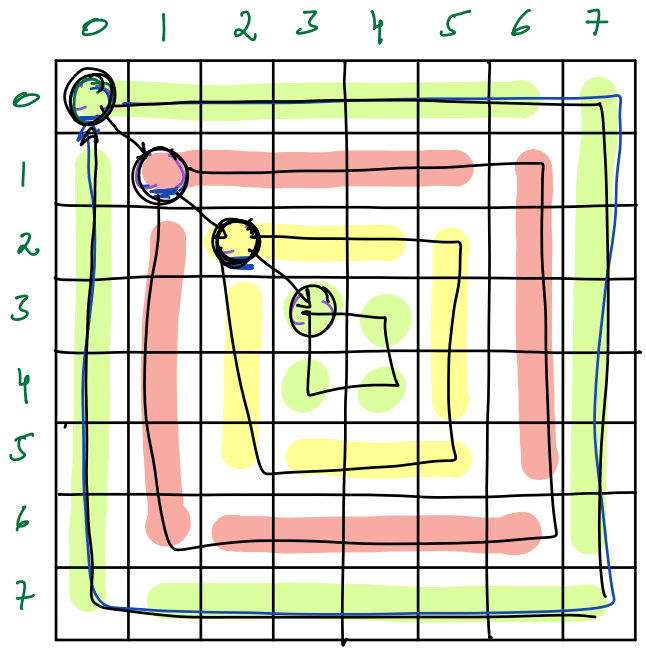
N ≥ 1 && M ≥ 1

i = 0, j = 0

while (N ≥ 1) { → N-1 times

```
    for (l = 1; l < N; l++) { // Print 0th row
        print(mat[i][j])
        j++
    }
    // After above loop i = 0  j = N-1
    for (l = 1; l < N; l++) { // Printing last column
        print(mat[i][j])
        i++
    }
    // After above loop i = N-1  j = N-1
    for (l = 1; l < N; l++) { // Printing last row R-L
        print(mat[i][j])
        j--
    }
    // After above loop i = N-1  j = 0
    for (l = 1; l < N; l++) { // print from column D-U
        print(mat[i][j])
        i--
    }
    N = N-2; i = i+1; j = j+1
}
if (N == 1) { print(mat[i][j]) } Edge Case
```

N=8



N=8

Steps: 7 Steps [0 0] → [0 6]

7 Steps [0 7] → [6 7]

7 Steps [7 7] → [7, 1]

7 Steps [7 0] → [1, 0]

N=6

Steps: 5 Steps [1 1] → [1 5]

5 Steps [1, 6] → [5 6]

5 Steps [6, 6] → [6, 2]

5 Steps [6, 1] → [2, 1

N=4 3 Steps [2 2] → [2 4]

3 Steps [2 5] → [4 5]

3 Steps [5 5] → [5 3]

3 Steps [5 2] → [3 2]

N=2 1 Step [3 3]

1 Step [3 4]

1 Step [4 4]

1 Step [4 3]

N=5

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

Step1: N=5 (i, j)

4 steps [0 0] → [0 3]

4 steps [0 4] → [3 4]

4 steps [4 4] → [4 1]

4 steps [4 0] → [1 0]

N = N-2, i = i+1, j = j+1

N=3

2 steps [1 1] → [1 2]

2 steps [1 3] → [2 3]

2 steps [3 3] → [3 2]

2 steps [3 1] → [2 1]

N=1 i=2, j=2 → i, j is at center

0 steps

0 steps

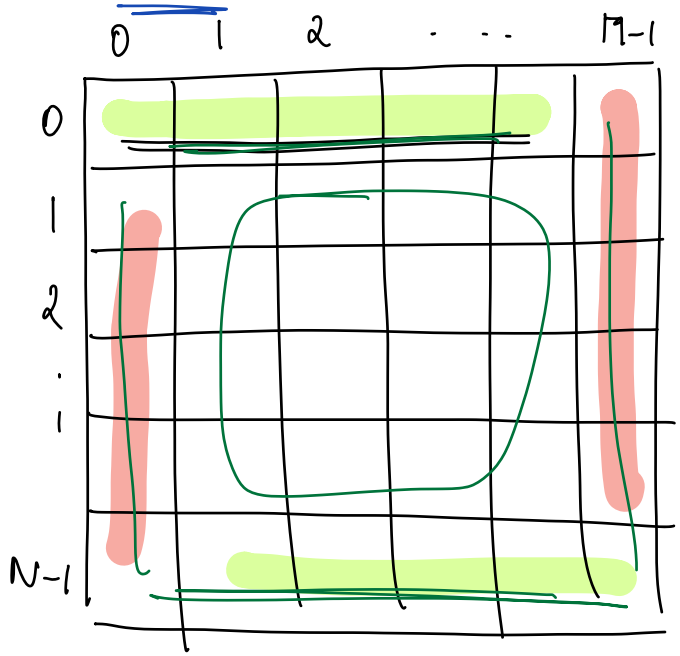
0 steps

0 steps

N=-1 i=3, j=3

// Hint: (TODO)

Mat[N][M]



Steps

first row: M-1 steps (L-R)

last col: N-1 steps (T-D)

last row: M-1 steps (R-L)

first col: N-1 steps (D-T)

N = N-2 M = M-2, i = i+1, j = j+1

Edge case: { }

20?

[N × M] → N rows, M columns

Given a 2D Matrix every row is sorted & every column is sorted, Check an element k = 15, 34, 80 ✓ ✓ ✗

	0	1	2	3	4	5	k = 15
0	-1	2	4	5	9	11	< 15 ✗
1	1	4	7	8	10	14	< 15 ✗
2	3	7	9	10	12	18	> 15 {Maybe/BS}
3	6	10	12	14	16	20	> 15 {Maybe/BS}
4	9	13	16	19	22	24	> 15 {Maybe/BS}
5	11	15	19	21	24	27	> 15 {Maybe/BS}
6	14	20	25	29	31	39	> 15 {Maybe/BS}
7	18	24	29	32	34	42	

18 > 15 ✗

Sol:

1) Linear Search on Entire Matrix

TC: [N * M] SC: O(1)

2) In an a sorted ar (N)

To apply BS → (log N)

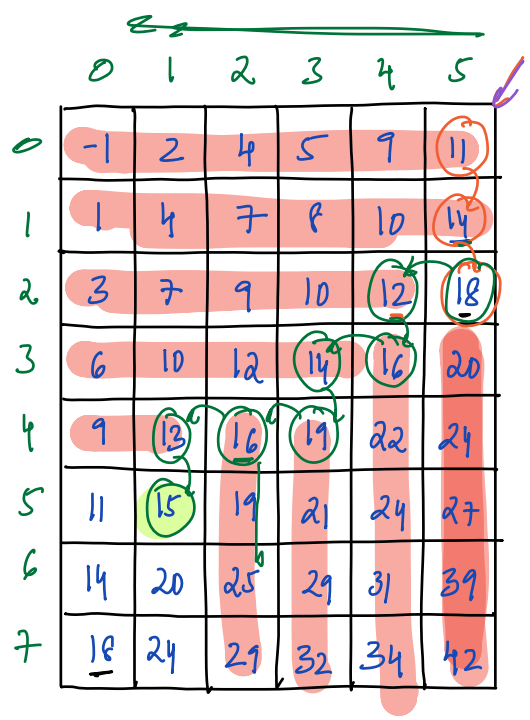
Apply BS on Every ROW (M)

TC: [N * log M]

SC: 1

3) Compare k with last Element & first & Apply BS

TC: N * log M SC: 1



	0	1	2	3	4	5
0	-1	2	4	5	9	11
1	1	4	7	8	10	14
2	3	7	9	10	12	18
3	6	10	12	14	16	20
4	9	13	16	19	22	24
5	11	15	19	21	24	27
6	14	20	25	29	31	39
7	18	24	29	32	34	42

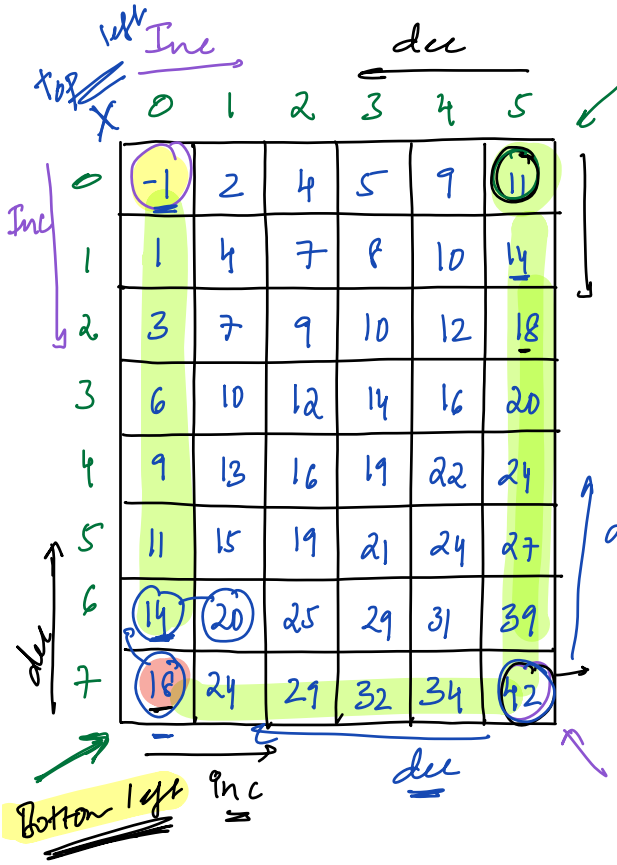
K = 15 (At every step either we ↓ Skip a row ← Column]

Idea: → TC: O(N+M) SC: O(1)

i = 0, j = M-1

```
while( i < N && j >= 0 ){
    Mat[i][j] < k :
        // skip row, i++
    Mat[i][j] > k :
        // skip col; j--
    Mat[i][j] == k
        return True
}
return False
```

Neha's skipping



	0	1	2	3	4	5
0	-1	2	4	5	9	11
1	1	4	7	8	10	14
2	3	7	9	10	12	18
3	6	10	12	14	16	20
4	9	13	16	19	22	24
5	11	15	19	21	24	27
6	14	20	25	29	31	39
7	18	24	29	32	34	42

Top right

Bottom left

Bottom right

Note:
We can either start from top right & Bottom left

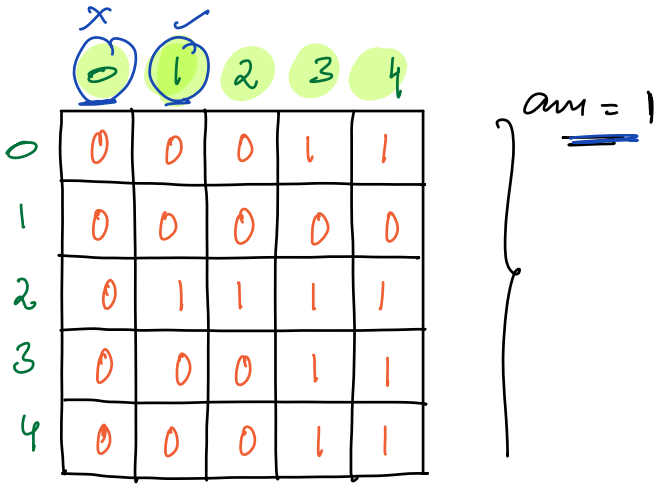
42 > 15

38) Given a 2D Matrix, every element is 0/1, O(N×M)

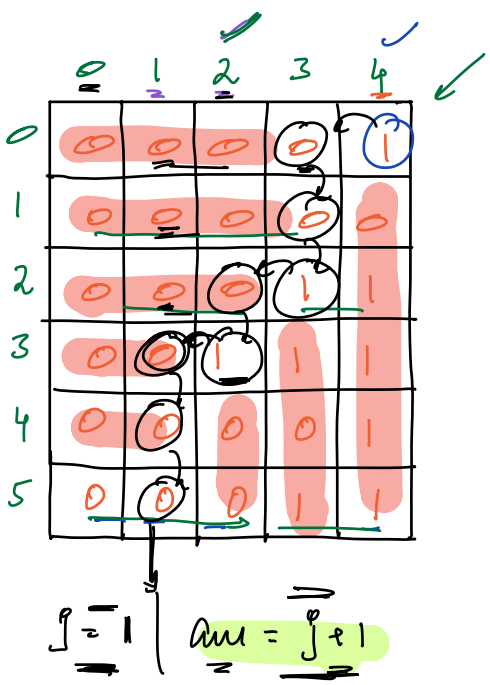
Every row is sorted,

find the first column which contains = 1

Ex 1:

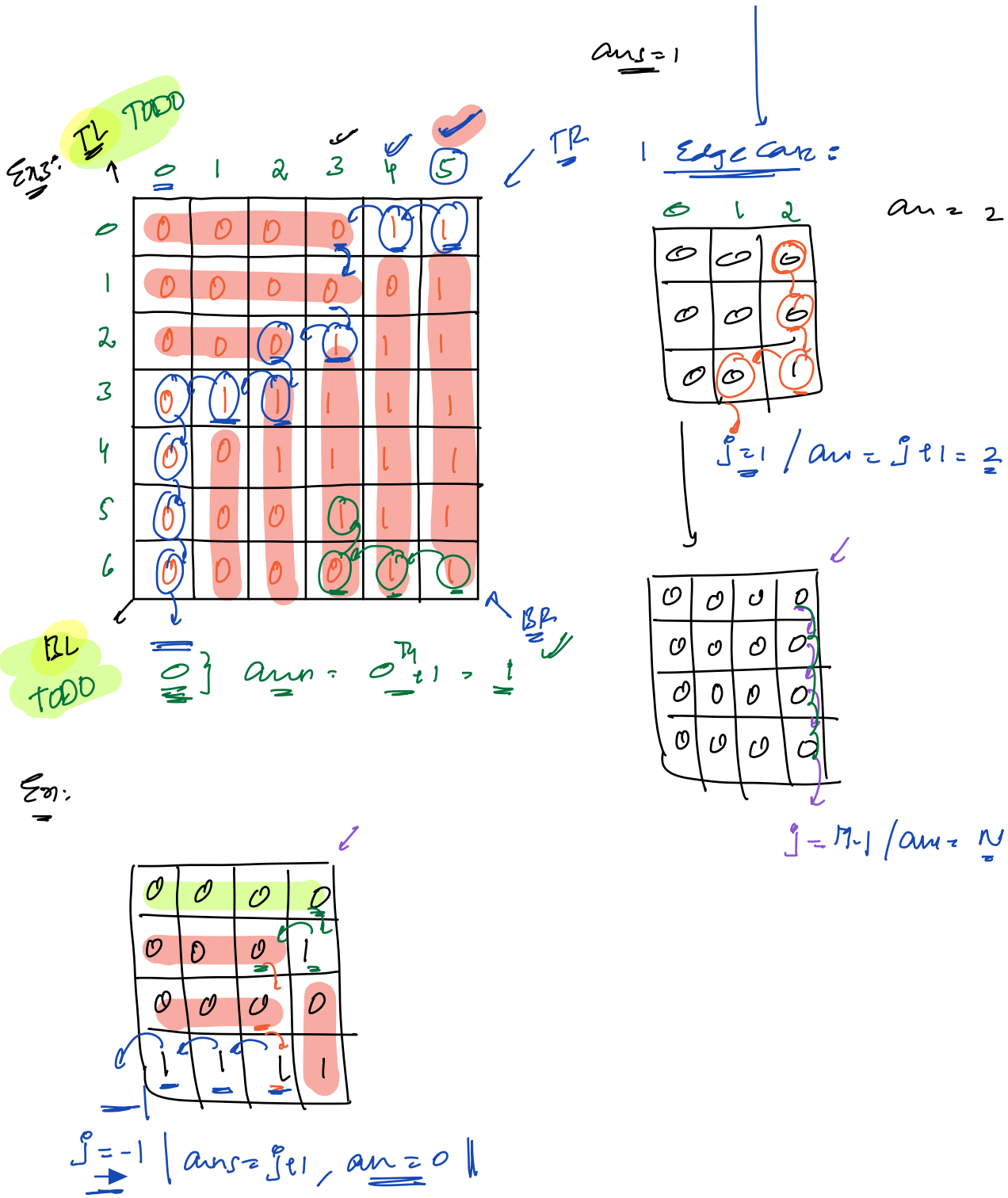


Ex 2:



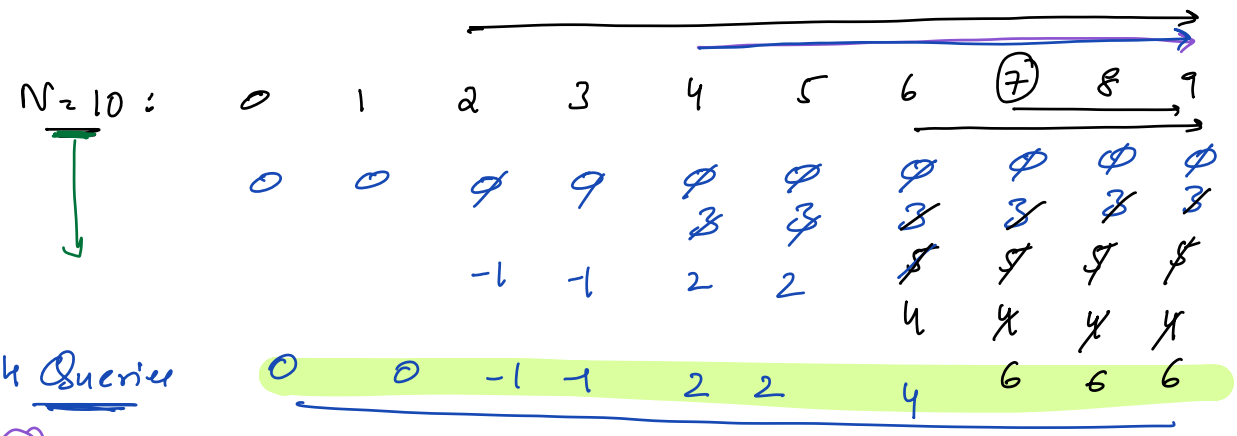

```
i = 0, j = M-1
while( i < N && j >= 0 ){
    mat[i][j] == 1 ){
        skip colum
        j--
    }
    else
        skip row
        i++
    }
}
// ans = j+1
if(j == -1)
    return -1
```

All 0's


ans=1
TL
TODO
Ex1:
0 1 2 3 4 5
TR
1 Edge Case:
0 1 2
ans = 2
j=1 / ans = j+1 = 2
BL
TODO
BR
0
ans = 0
M
t+1 = t
j = M-1 / ans = M
Ex1:
j=-1 / ans = j+1, ans = 0

→ _____ Output _____ [HARD]

4(8) Given N array elements initialized = 0 & Queries



i	data
4	3
6	2
2	-1
7	2

```
// Repeat Q) {
    for(i=l; i<=r; i++){
        ar[i] += data
    }
}
```

TC: Q * (N)

SC: O(1)

Techniques

1) Brute force //
2) Conf
3) Carry ✗
4) Suba ✗
5) Sliding ✗

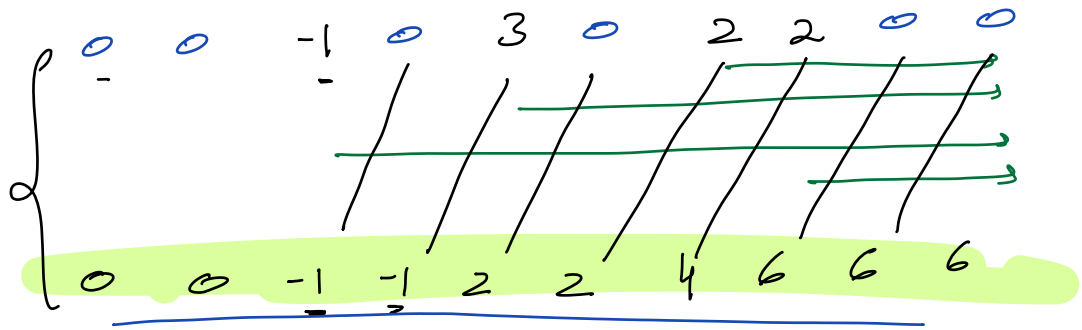
Pf[0] = ar[0]

Pf[1] = ar[0] + ar[1]

Pf[2] = ar[0] + ar[1] + ar[2]

Pf[3] = ar[0] + ar[1] + ar[2] + ar[3]

N = 10 : 0 1 2 3 (4) 5 6 7 8 9

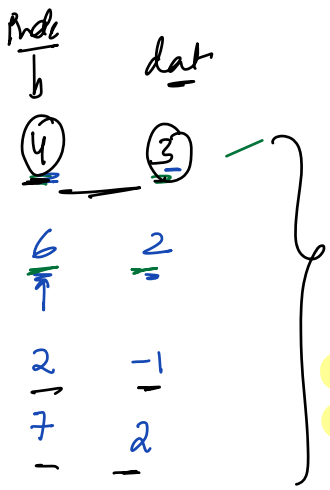


Idea:

After updating all queries do prefix sums only once.

update: ar[index] = data

After all updates do prefix sum



5(8) Given N array elements = 0 & Queries

arr[10]: 0 (1) 2 3 4 5 6 7 8 9

0 (3) 0 -2 0 0 -3 0 2 0

Actual propagat | Nullify Each

Pf 0 3 3 1 1 1 -2 -2 0 0

Q1:

1 5 : 3

3 7 : -2

Idea:

1) update all queries

2) later prefn

Query: l r data :

update all queries

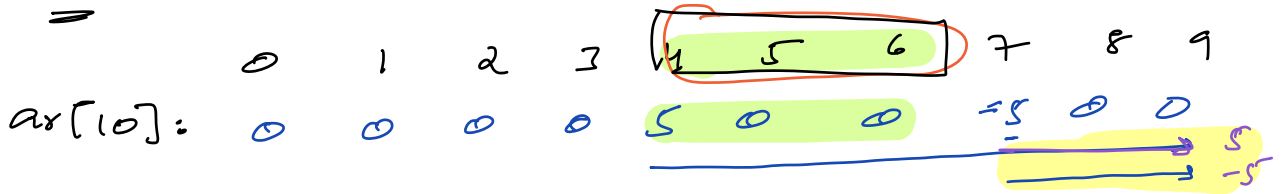
```
if (r < N-1) {
    arr[l] += data   arr[r+1] -= data
}
else → r == N-1 }
    arr[l] += data
}
```

⇒ Apply prefix fn

⇒ 1e5 = 10^5 , 1e9+7 = 10^9+7

(1e9+7 = exponent)

Doubts



Queries

4 6 5

arr[4] += 5
arr[7] -= 5

l r x

→ [Add x, l → N-1] arr[l] += x
[Subs x, r+1 → N-1] arr[r+1] -= x