Merge Intervals
Merge New Interval
Find 1st missing natural number

1. Advanced DSA Contest 1 → 6 Oct
   next Fri (Arrays 1, 2, 3)
   ↓
   9 - 10:30 PM
   1.5 hr → 3 ques

   10:30 PM

   Syllabus → Arrays 3 (3 ques)

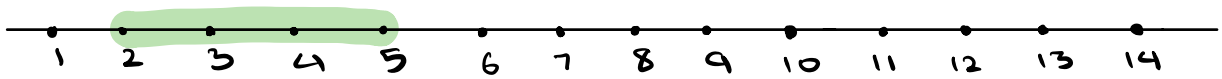
   Arrays 1 and 2
   ↳ Notes
   ↳ Assignments

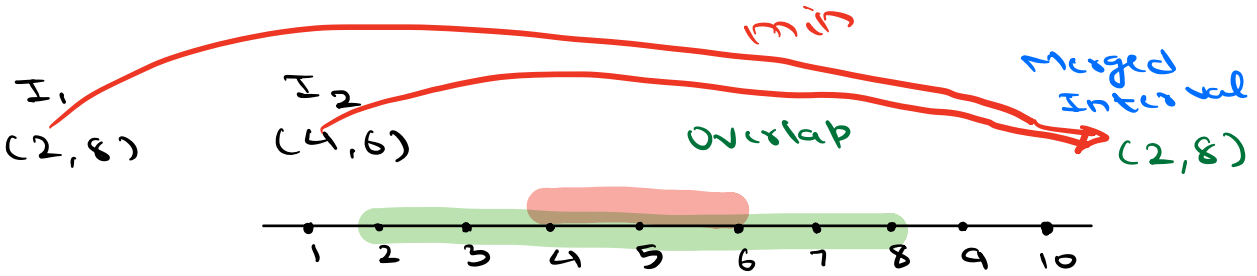
③ Kadane's algo
↓
carry forward

① Pf [ ] ques → diff subarray ranges

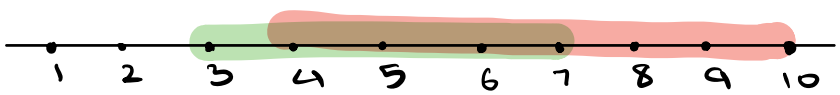
④ sliding window ② contribution
sum of all submatrices

An interval (s, e) (2, 5)

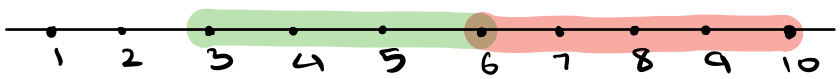




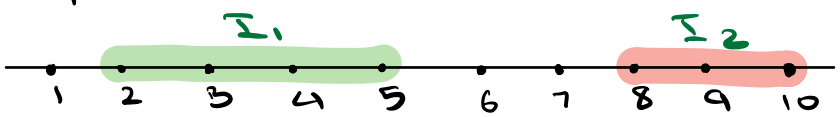
(3, 7) (4, 10) (3, 10)



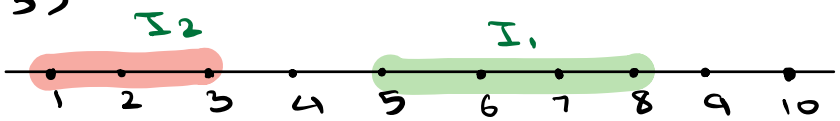
(3, 6) (6, 10) (3, 10)



I1 (2, 5) I2 (8, 10) No overlap I1 comes first



I1 (5, 8) I2 (1, 3) No overlap I2 comes first



I_1 (s_1, e_1) I_2 (s_2, e_2)

Case 1:

$e_1 < s_2$

Case 2:

$e_2 < s_1$

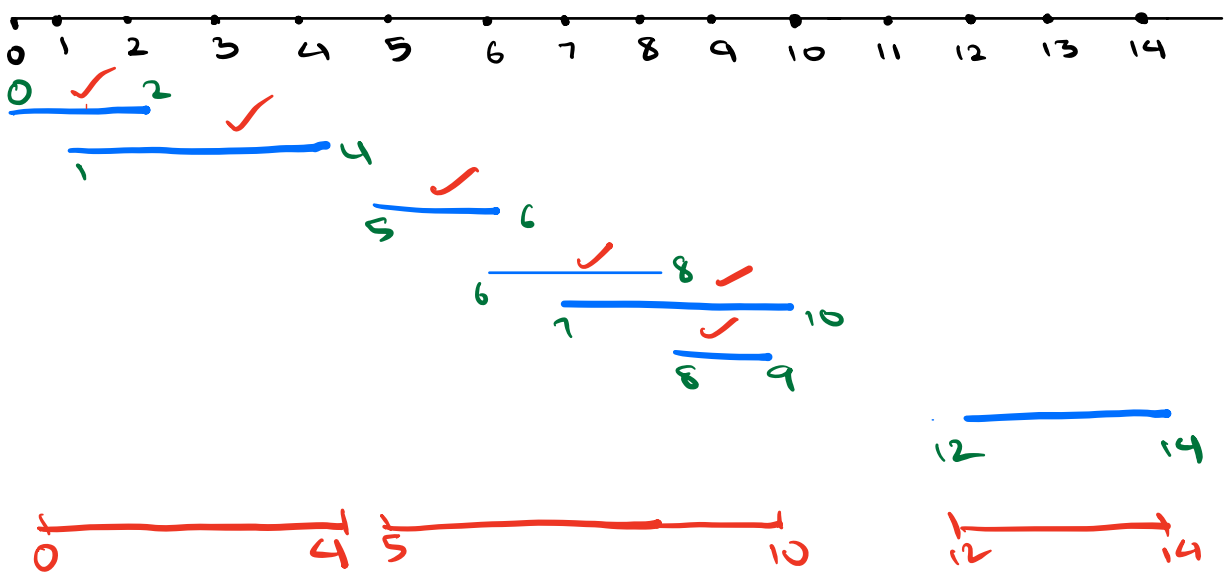
I_1 comes first I_2 comes first

```
if ( e1 < s2  ||  e2 < s1 ) {
    // no overlap
}
else {
    // overlap
    merge_s = min(s1, s2)
    merge_e = max(e1, e2)
}
```

1. Given a sorted list of overlapping intervals, sorted based on start time, merge all overlapping intervals and return sorted list.

Interval [ ]

= < (0,2) , (1,4) , (5,6) , (6,8) , (7,10) , (8,9) , (12,14) >

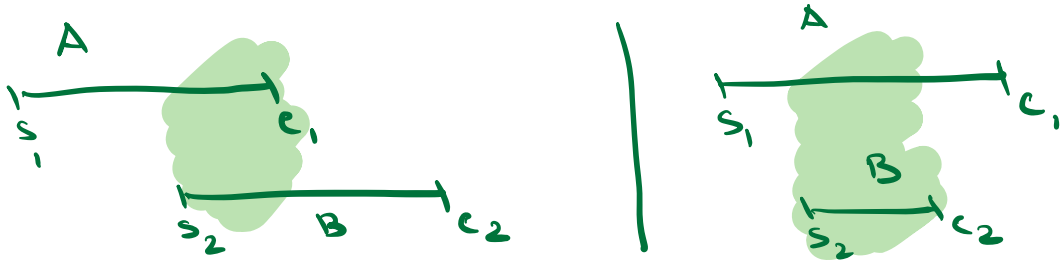


Overlapping

A
s_1 , e_1

B
s_2 , e_2

$s_1 < s_2$ → as per question



2nd meeting will start before 1st meeting ends

Overlapping $s_2 <= e_1$

Interval [ ]

= < (0,2) , (1,4) , (5,6) , (6,8) , (7,10) , (8,9) , (12,14) >
0 1 2 3 4 5 6

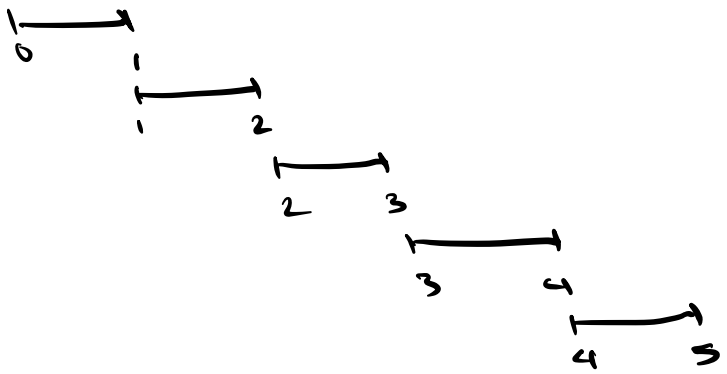
	Current	next available		After merging	ans
0th int	0, 2	① 1, 4	✓	(0,4)	
	0, 4	② 5, 6	cur before next		0, 4
	5, 6	③ 6, 8	✓	(5,8)	
	5, 8	④ 7, 10	✓	(5,10)	
	5, 10	⑤ 8, 9	✓	(5,10)	
	5, 10	⑥ 12, 14	cur before next		

(0,4)
(5,10)

12, 14
Last cur interval

(0,4)
(5,10)
(12,14)

Ans → dynamic array



ans

[(0,5)]

ans →

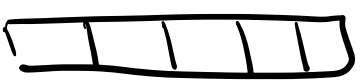
① If cur & next available overlap, get merged → new cur

② cur & next available do not overlap
add cur to ans

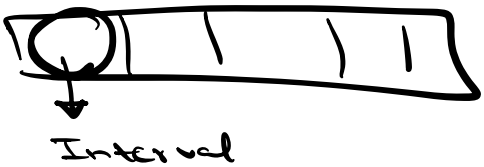
Interval [ ]

= < (0,2) , (1,4) , (5,6) , (6,8) , (7,10) , (8,9) , (12,14) >

int arr [5] =

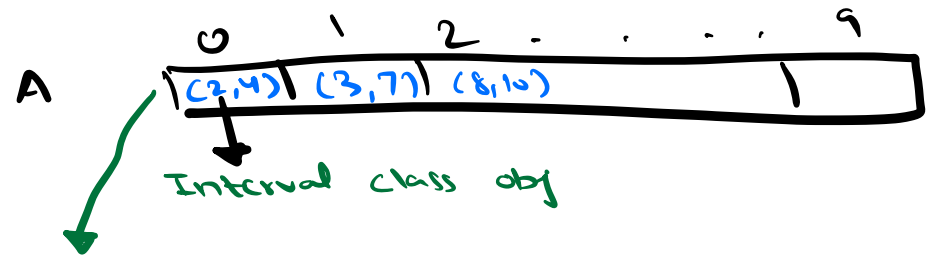


Interval given [10]




```
class Interval {
    int start
    int end
}
```

Interval A [10]



A [0]

A [0]. start → 2

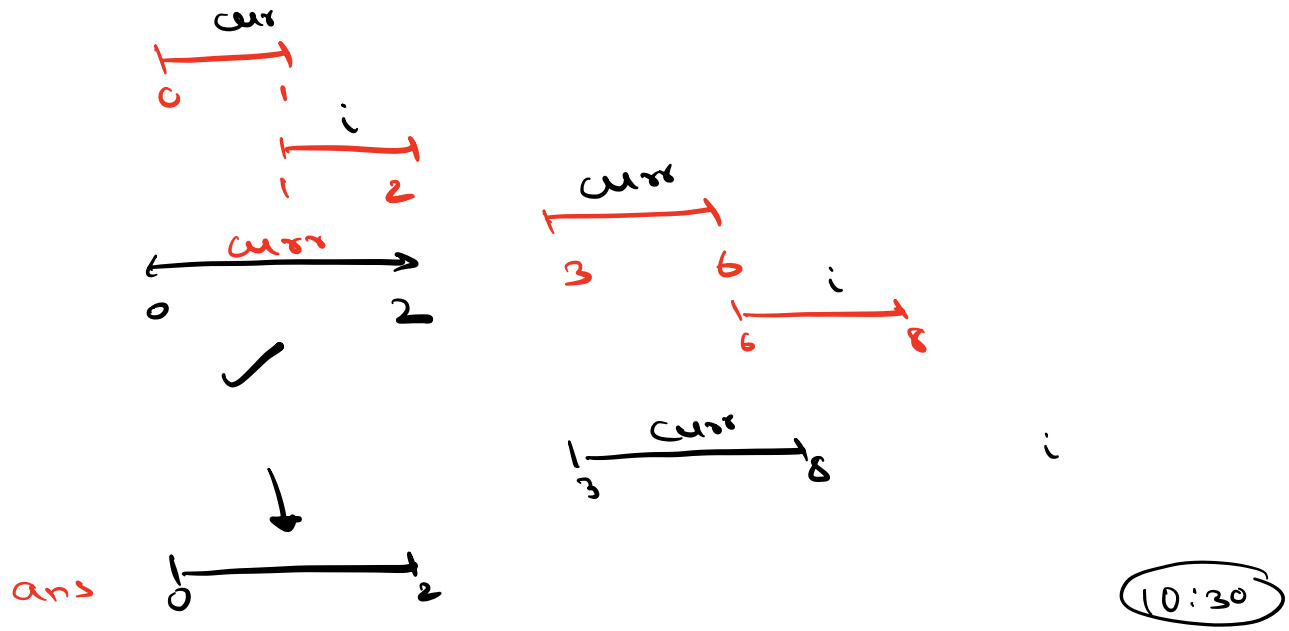
A [0]. end → 4

```
// Interval A[], N

list<Interval> ans

int cur_start = A[0].start
int cur_end   = A[0].end

// print available interval
for(i = 1; i < N ; i++) {

    // is cur overlapping with ith interval
    if(A[i].start <= cur_end) {
        cur_end = max(cur_end,
                      A[i].end)
    }
    else {
        // non overlap
        ans.push_back(cur)
        cur_start = A[i].start
        cur_end   = A[i].end
    }
}

ans.push_back(cur)
return ans
```

TC: O(N)

SC: O(1)

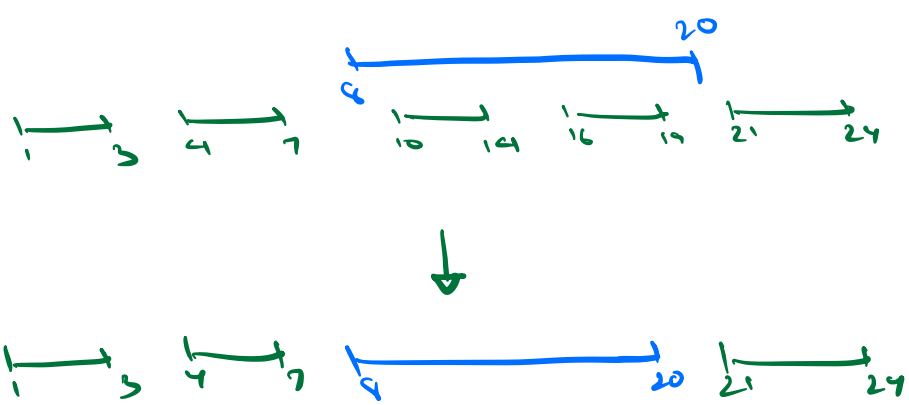
→ Last cur interval



10:30

---

2. Given a sorted list of non-overlapping intervals based on start time, insert a new interval such that final list of intervals is sorted and non-overlapping.
Print intervals.



In	New Interval	Y/N Overlap	Print
N = 9			
(1,3)	(12,22)	N	(1,3)
(4,7)	(12,22)	N	(4,7)
(10,14)	(12,22)	Y ; (10,22)	
(16,19)	(10,22)	Y ; (10,22)	
(21,24)	(10,22)	Y ; (10,24)	
(27,30)	(10,24)	N	
(32,35)			
(38,41)			
(43,50)			

New interval (10,24)

All are printed as it is

(27,30)
(32,35)
(38,41)
(43,50)

In List	New Interval	Y/N Overlap	Print
(1,5)	12,24	N	(1,5)
(8,10)	12,24	N	(8,10)
(11,14)	12,24	Y ; (11,24)	
(15,20)	11,24	Y ; (11,24)	
(21,24)	11,24	Y ; (11,24)	

New 11,24 → (11,24)

```
void merge(Interval A[], int N, int ns, int ne){
    for (i=0; i<N; i++) {
        if(ns > A[i].end) { // ith interval comes 1st
            print(A[i].start, A[i].end)
        }
        else if(ne < A[i].start) { // new interval comes 1st
            print(ns, ne)
            for(j=i; j<n; j++) {
                print(A[j].start, A[j].end)
            }
            return
        }
        else {
            ns = min(ns, A[i].start)
            ne = max(ne, A[i].end)
        }
    }
    print(ns, ne)
}
```

TC : O(N)

SC : O(1)

3. Given an unsorted array of integers, find first missing natural number.
1, 2, 3 .....

a[5] : <3, -2, 1, 2, 7> ans : 4

a[7] : <-9, 2, 6, 4, -8, 1, 3> ans : 5

a[9] : <-2, 4, -1, -6, 3, 7, 8, 4, -3> ans : 1

a[6] : <1, 0, -5, -6, 4, 2> ans : 3

a[6] : <1, 2, 5, 6, 4, 3> ans : 7

a[4] : <4, 2, 1, 3> ans : 5

BF : Search from all the nos. starting from 1 till we get the answer

TC : O(N * ans) → O(N²) (ans → N+1)

ar[N] = | 1 | 2 | 3 | ... | N | (0 1 . . . . . N-1)

ar[N] → 1 ... N

ans → 1 → N+1

Approach 2 : Add elements to set
check all the nos starting 1 to ans → present in HS

TC : O(N) SC : O(N)

Approach 3 : Sort the data

→ 1, 0, -5, -6, 4, 2

↓ sort

-6, -5, 0, 1, 2, 4 → 3 is missing

num=1 num=2 num=3

num < A[i]

↓

num is missing

TC : O(nlogn + n)

SC : O(sorting algo)

Optimised

ans

1 → N+1

① Search for 1 → N (ans)

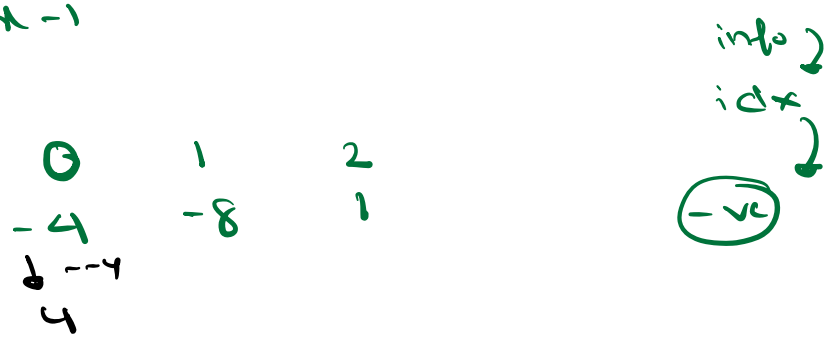
② ans → N+1

Mark presence 1, 2, 3 ... N in array

idx → 0 1 2 3 ... N-1

num → 1 2 3 4 ..... N

For a num x, store its info at index x-1



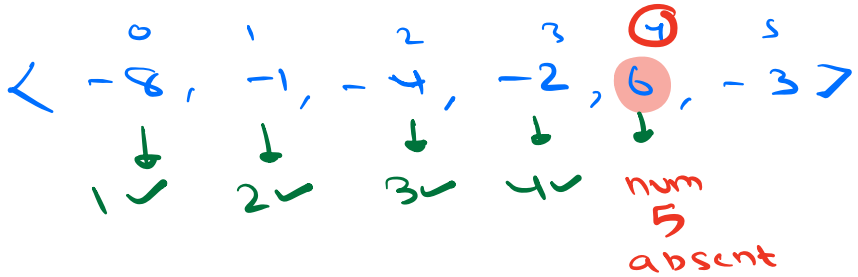
Lets assume only +ve nos. then !

A[] = <8, 1, 4, 2, 6, 3>
N = 6 ans → [1 7]

idx	elem	present → idx	arr
0	8	Don't care	<8, 1, 4, 2, 6, 3>
1	1	0	<-8, 1, 4, 2, 6, 3>
2	4	3	<-8, 1, 4, -2, 6, 3>
3	-2 → |2|	1	<-8, -1, 4, -2, 6, 3>
4	6	5	<-8, -1, 4, -2, 6, -3>
5	-3 → |3|	2	<-8, -1, -4, -2, 6, -3>

num → 1, 4, 2, 6, 3

① Go to every elem → 1 to N, mark its presence x → (x-1) → arr[x-1] → -ve



num → x

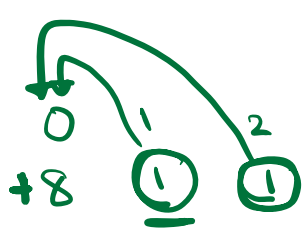
idx → x-1

```
for (i=0; i<N; i++) {
    int dc = abs(A[i])
    if (dc >=1 && dc <=N) {
        idx = dc - 1
        A[idx] = -1 * abs(A[idx])
    }
}
```

TC: O(N)

SC: O(1)

```
for (i=0 ; i<N ; i++) {
    if (A[i] > 0)
        return i+1
}
return N+1
```



arr → -ve nos. handle them?

Case 1 +ve nos ✗

Case 2 0 ✗

	0	1	2	
-2	1	1		ans=2
	2	1	1	ans=3

-ve nos. → N+2 / N+3 / INT_MAX

```
for (i=0; i<N; i++) {
    if (A[i] <= 0)
        A[i] = N+2
}
for (i=0; i<N; i++) {
    int dc = abs(A[i])
    if (dc >= 1 && dc <= N) {
        idx = dc - 1
        A[idx] = -1 * abs(A[idx])
    }
}
```

TC: O(N)
SC: O(1)

```
for (i=0; i<N; i++) {
    if (A[i] > 0)
        return i+1
}
return N+1
```

Dry run → Eg : <4, 0, 1, -5, -10, 8, 2, 6> HW

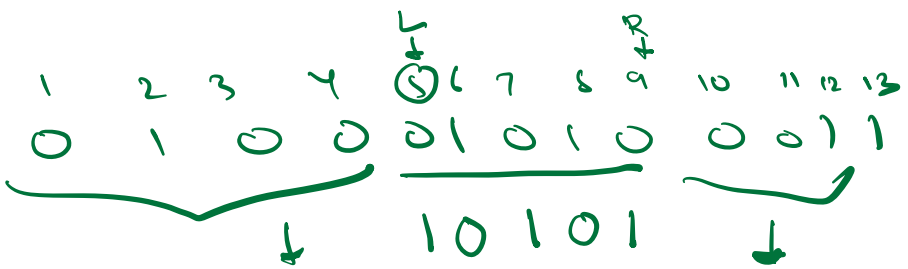
Flip A_1 A_2 ..... A_N Doubts

1 2 ..... N

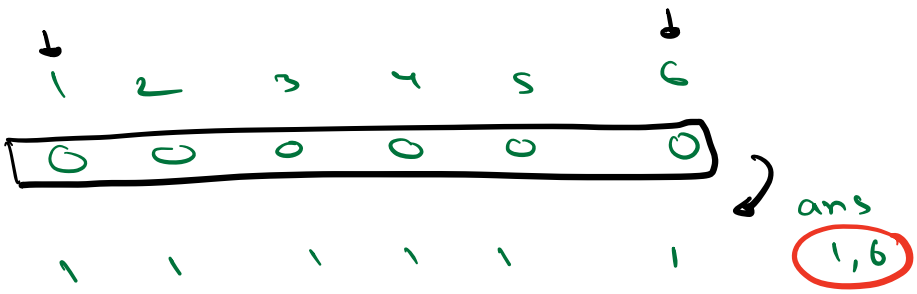
0 → 1
1 → 0

1 2 3 ..... N

0 1 0

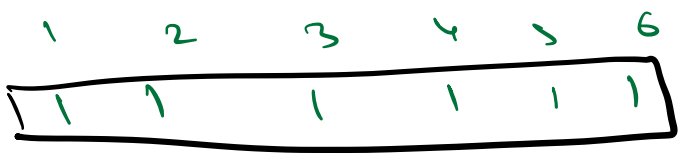


Eg 1

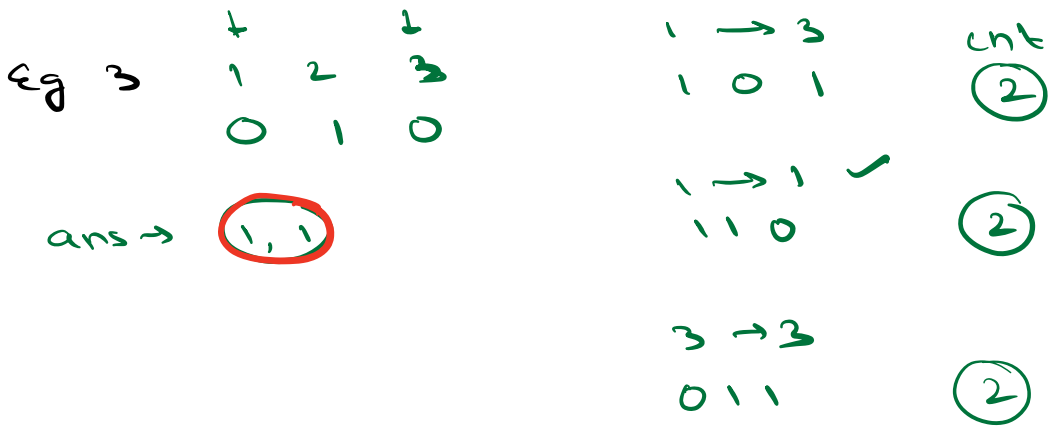


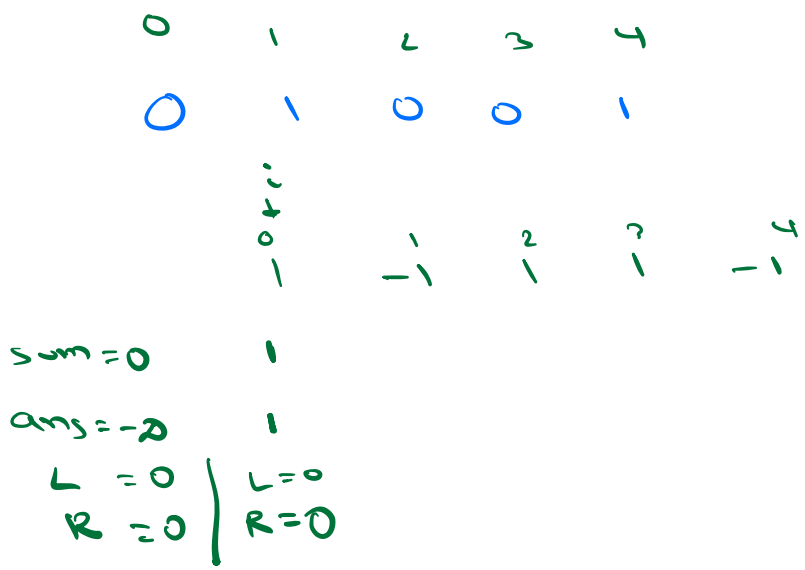
ans
1, 6

Eg 2



ans
None


Eg 3
1 2 3
0 1 0
ans → 1, 1
1 → 3
1 0 1
cnt
2
1 → 1
1 1 0
2
3 → 3
0 1 1
2


0 1 2 3 4
0 1 0 0 1
0 1 2 3 4
1 -1 1 1 -1
sum = 0
ans = -2
L = 0
R = 0
L = 0
R = 0