Todays Content:

- Find max no of 1's in any row - 3
- Find 1st missing +ve Integer - 1
- Merge Overlapping Intervals - 2

// Sat → 4 array doubts : 9PM onwards

- max subarray sum ✓
- max submatrix sum ✓
- row-wise column wise sorted max submatrix sum ✓
- Find max no of 1's in any row - 3 ✓
- Max absolute diff ✓

Q1 Given ar[N], find first missing natural number
n = 1, 2, 3 ...

Examples:

	ans
ar[5] = 3 -2 1 2 7	4
ar[7] = -9 2 6 4 -8 1 3	5
ar[6] = 1 2 5 6 4 3	7
ar[5] = -4 8 3 -1 0	1
ar[4] = 4 2 1 3	5
ar[4] = -8 -3 -1 -5	1
ar[3] = 1 2 1	3

Obs:

ar[5]: 5 2 4 1 3

min	max
1	6

ans: 1

ar[4]: 4 2 1 3

min	max
1	5

ans: 1

ar[N]: N 4 N-1 .. 1 2 3 : min: 1 max: N+1
↳ If ar[] all elements [1 N]

Obs: Given ar[N] ans range = {1, N+1}

Idea1:

```
i=1; i<=N; i++) {   TC: N*N => O(N^2)   SC: O(1)
    if ( search(ar[], i) == false) {
        return i
    }
}
return N+1 // ar[] contains all ele from 1-N
```

search(ar[], i) → TC: O(N) → Implement in your own

Idea2:

```
Hashset<int> hs                TC: O(N + N) => O(N)
insert ar[] -> hs : O(N)       SC: O(N)
i=1; i<=N; i++) {
    if( hs.search(i) == false) { return i }
}
return N+1 // ar[] contains all ele from 1-N
```

Idea: ar[7] = -9 2 6 4 -8 1 3

Sort ar[] = -9 -8 1 2 3 4 6

* * ✓ ✓ ✓ ✓ 5 → 5 is missing ans = 5

ar[8] = -3 1 5 8 14 2 7 3

Sort ar[] = -3 1 2 3 5 7 8 14

* ✓ ✓ ✓ 4 → 4 is missing ans = 4

→ TODO: Sort ar[] & iterate & find missing natural number

TC: O(NlogN + N) ⇒ O(NlogN) SC: O(1)

Idea: ar[5] = { 3 -1 7 2 1 } * {wont work}

↳ sum of ar[] - { sum of N Natural numbers }

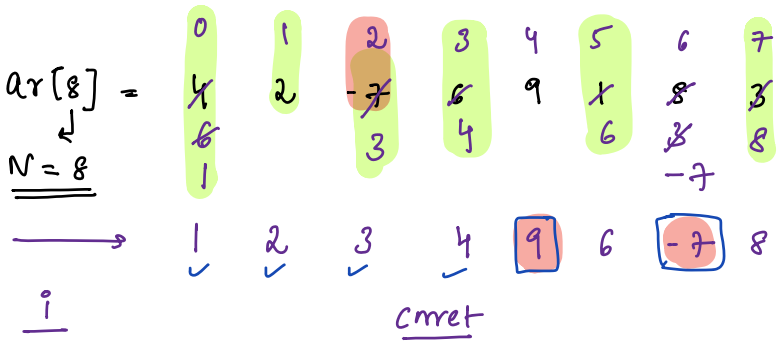
: 12 - { 15 } = -3

Idea: Data we care is {1-n}

	0	1	2	3	4
ar[] :	1	2	3	4	5

↳ Every element bring to it's correct index

ar[i]	Correct index
V	V-1



0 → ar[0] = 4 → 3, Swap (ar[0], ar[3])

→ ar[0] = 6 → 5, Swap (ar[0], ar[5])

→ ar[0] = 1 { At ar[0], correct data, goto next index }

1 → ar[1] = 2 { At ar[1], correct data, goto next index }

2 → ar[2] = -7 { At ar[2], Irrelvent data, goto next index }

3 → ar[3] = 4 { At ar[3], correct data, goto next index }

4 → ar[4] = 9 { At ar[4], Irrelvent data, goto next index }

5 → ar[5] = 6 { At ar[5], correct data, goto next index }

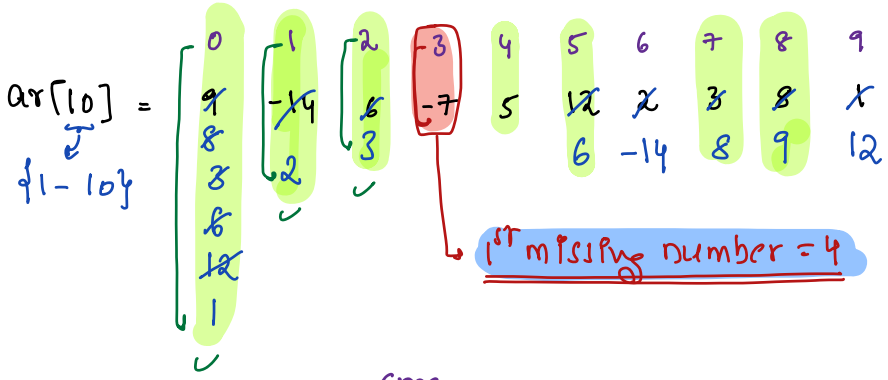
6 → ar[6] = 8 → 7 : Swap (ar[6], ar[7])

ar[6] = 3 → 2 : Swap (ar[6], ar[2])

ar[6] = -7 { At ar[6] Irrelvent data, goto next index }

7 → ar[7] = 8 { At ar[7], correct data, goto next index }

8] Stop:



cmr

i

0 → ar[0] = 9 → 8 ; swap(ar[0] ar[8])

→ ar[0] = 8 → 7 ; swap(ar[0] ar[7])

→ ar[0] = 3 → 2 ; swap(ar[0] ar[2])

→ ar[0] = 6 → 5 ; swap(ar[0] ar[5])

ar[0] = 12 → {irrelevant} i = i+1

1 → ar[1] = -14 → {irrelevant} i = i+1

2 → ar[2] = 3 → {Correct data} i = i+1

3 → ar[3] = -7 → {irrelevant} i = i+1

4 → ar[4] = 5 → {correct data} i = i+1

5 → ar[5] = 6 → {Correct data} i = i+1

6 → ar[6] = 2 → 1 ; swap(ar[6] ar[1])

ar[6] = -14 → {irrelevant} i = i+1

7 → ar[7] = 8 → {Correct data} i = i+1

8 → ar[8] = 9 → {Correct data} i = i+1

9 → ar[9] = 1 → 0, swap(ar[9], ar[0])

ar[9] = 12 → {irrelevant} i = i+1

10 → stop

Pseudocode :

```
int missing(int ar[], int n){   TC: O(N)  SC: O(1)
    i = 0;
    while(i < n){        increment        cmetdata
        if(ar[i] < 1 || ar[i] > N || ar[i] = i+1){
            i = i+1
        }
        else{
            cind = ar[i] - 1
            if(ar[i] == ar[cind]){ i = i+1 }      // without it, it goes to ∞ loop
            else{ swap(ar[i], ar[cind]) }
        }
    }
    i = 0; i < n; i++){
        if(ar[i] != i+1){ return i+1 }
    }
    return N+1
}
```

10:32 → 10:45pm

Edge Case : ar[5] = { 4 1 3 3 2 }
ar[1-5]

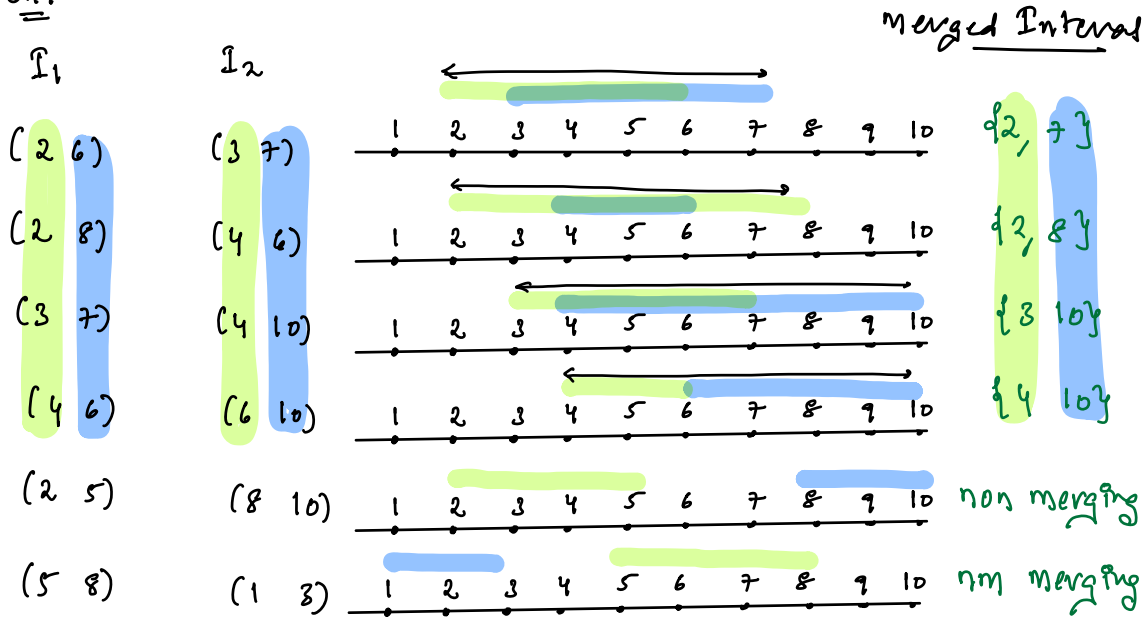
0	1	2	3	4
~~4~~ 3	1	~~3~~ 3	~~3~~ 4	2

i			cur	
0	→	ar[0] = 4 →	3	: swap(ar[0], ar[3])
	→	ar[0] = 3 →	2	: swap(ar[0], ar[2])
	→	ar[0] = 3 →	2	: swap(ar[0], ar[2])

∞ loop ?

Merge Intervals: I : {s x e}

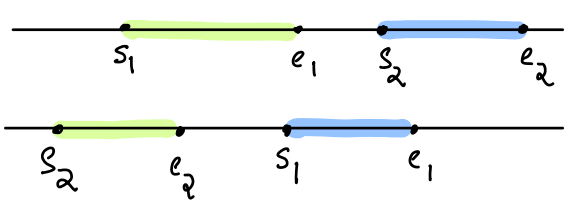
Eg:



Non-Overlap Can

$I_1 : \{s_1 \ e_1\}$

$I_2 : \{s_2 \ e_2\}$



if ($s_2 > e_1$)

if ($s_1 > e_2$)

if ($s_2 > e_1$ || $s_1 > e_2$) { not overlapping }

Overlapping Interval

$I_1 : \{s_1 \ e_1\}$

$I_2 : \{s_2 \ e_2\}$

merged Interval :

(min(s_1, s_2), max(e_1 e_2))

Given N non-overlapping Intervals, in increasing order of their start time, insert new Interval & [print non-overlapping intervals, in inc_order of start]

(new Interval = given input)

Input:

N = 9 new Interval

[1 3]

[4 7]

[10 14] [12 22) → {10, 22}

[16 19] {10, 22} → {10, 22}

[21 24] {10, 22} → {10, 24}

si ei s e

[27 30] {10, 24}

[32 35]

[38 41]

[43 50]

Output

[1 3]

[4 7]

{10, 24}

[32 35]

[38 41]

[43 50] : return

N=5 new Interval

[1 5]

[8 10]

[11 14] [12 22] → {11, 22}

[15 20] {11, 22} → {11, 22}

[21 24] {11, 22} → {11, 24}

Output

[1 5]

[8 10]

{11, 24}

N = 5 :

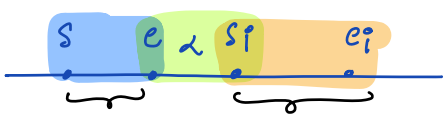
New Interval

- ✓ [1 5]
- ✓ [7 9]
- { [15 20) [21 24] [27 30] }

Final Intervals

[1 5]
[7 9]
[11 14] new interval
[15 20)
[21 24]
[27 30]

```
// N interval [s e]   // new interval [s e]
void merge( int Intervals[N][2], int s, int e){        TC: O(N)   SC: O(1)
    i = 0; i < n; i++){
        s_i = Interval[i][0], e_i = Interval[i][1]
        if ( e_i < s ){
            print( { s_i, e_i } )
        }
        else if ( e < s_i ) {
            print ( { s, e } ) // new interval printed
            // print all remaining interval
            j = i; j < n; j++) {
                print( { Interval[j][0], Interval[j][1] }
            }
            return ; // come out of function
        }
        else { s = min(s, s_i)  e = max(e, e_i) }
    }
    print( { s, e } ) } Edge Case
}
```



3Q) Given binary (0/1) mat[N][M], row-wise sorted find max no: of 1's in a row.

mat[6][7]:

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

Ideas:

1)

2)

Idea2:

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

```
int maxOnes(int mat[][], int N, int M){

}
```