ARRAYS 3

"Goals Begin Behaviors – Consequences Maintain Behavior"

Good Evening

Today's content

01. First Missing Integer (Amazon)
02. Search in row wise, col-wise sorted matrix
03. Insert Interval

First Missing Positive

Find the first missing natural number / first positive integer

Eg:-

ar[6] = | 3 | -1 | 1 | 2 | 7 | 0 | Ans = 4

ar[6] = | 1 | 0 | -5 | -6 | 4 | 2 | Ans = 3

ar[7] = | -9 | 2 | 6 | 4 | -8 | 1 | 3 | Ans = 5

ar[6] = | -3 | -1 | -1 | -2 | -7 | -20 | Ans = 1

ar[6] = | 1 | 2 | 5 | 6 | 4 | 3 | Ans = 7

Brute force → Search from 1 to n+1 in your array

TC : $O(n^2)$

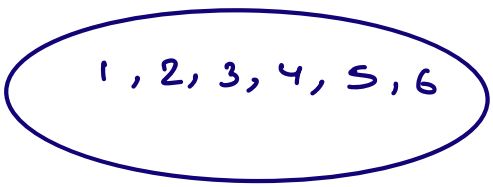
SC : $O(1)$

Idea 2 → Using hashset

01. Put all elements in hashset

arr[6] =

1	2	5	6	4	3



02. Start iterating from 1 & check which positive no. is not present

```
for ( i = 1 → n+1 )
    if ( hashset.contains ( i ) == false ) {
        return i;
    }
```

TC : O(n)
SC : O(n)

Note → Can't take extra space

Idea 3 → Use sorting to get the ans

arr = { -3, -7, 1, 2, 3, 8, 5 }

↓ sort

arr = { -7, -3, 1, 2, 3, 5, 8 }

TC : O(n log n)
SC : O(1)

arr = {-3, -7, 1, 1, 2, 4}

↓ sort

arr = {-7, -3, 1, 1, 1, 2, 4}

① ② ③

Arrays.sort(arr)

int val = 1

arr = {-7, -3, 1, 2, 3, 3, 5, 6}

val = ~~1~~ ~~2~~ ~~3~~ 4

```
for(i=0; i<n; i++){
    if(arr[i]<1){continue};
    else{
        if(ar[i]==val) val++;
        else if(ar[i]==val-1){continue}
        else {break}
    }
}
return val;
```

arr = {5, 6, 7, 10}

Ans = 1

Constraint → TC: O(n)

SC: O(1)

Idea 4 → keep the elements at their position

arr[6] =

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

~~4~~ 4 5 6

~~6~~

3

val		idx
1	→	0
2	→	1
3	→	2
⋮		
x	→	x-1

arr = { 1, 2, 3, 4, 9 }

0 1 2 3 4

Minimum ans = 1

Maximum ans = 6

do nothing { val < 1 , val > N

arr = { -5, -2, 1, 3 }

0 1 2 3

i	val	
0	-5	{ do nothing }
1	-2	{ do nothing }
2	1	{ swap (arr[2] , arr[0]) } do nothing
3	3	{ swap (ar[3] , ar[2]) do nothing

arr = { 1, 4, 9, 8, 2 }

0 1 2 3 4

arr = { 2, 4, 2, 4, -10, 7 }

0 1 2 3 4 5

```
int i = 0

while (i<n)
{
    if (ar[i] ≥ 1 && ar[i] ≤ n)
    {
        int corr_idx = ar[i] - 1;

        if (arr[corr_idx] != ar[i])
        {
            swap (ar[corr_idx], ar[i]);
        }
        else { i++; }
    }
    else{ i++; }
}
```

TC : O(n)

SC : O(1)

Iterate again & find first missing no.

```
for (i=0; i<n; i++) {
    if (ar[i] != i+1) return i+1;
}
return n+1;
```

02. Given a matrix where every row & column are sorted. Find an ele k. Return true if the element exisl.

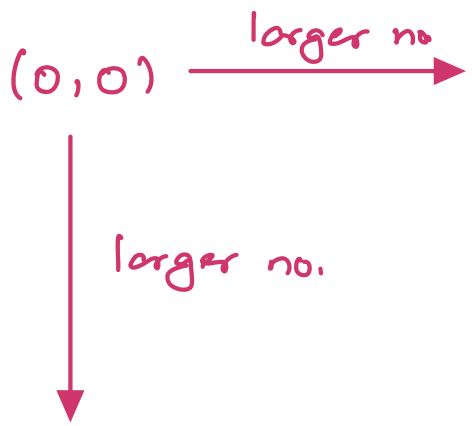
	0	1	2	3	4	5
0	-1	2	4	5	9	11
1	1	4	7	8	10	14
2	3	7	9	10	12	18
3	6	10	12	14	16	20
4	11	15	19	21	24	27
5	18	24	29	32	34	42

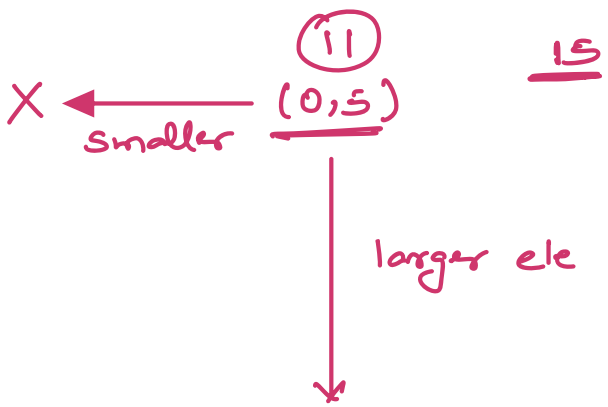
K = 15

BF → Traverse all the elements in matrix & check if (ele == k)

TC : O(n*m)

SC : O(1)





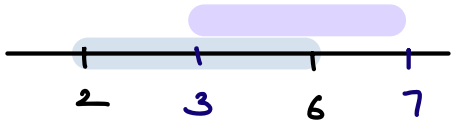
```
i = 0    j = m-1

while ( i < n && j ≥ 0 )
|
|    if (ar[i][j] == k)
|    |
|    |    return true;
|    }
|    else if (ar[i][j] < k)
|    |
|    |    i++ ;
|    }
|    else  j-- ;
|
}
return false ;
```

TC: O(n+m)

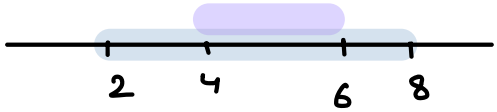
SC: O(1)

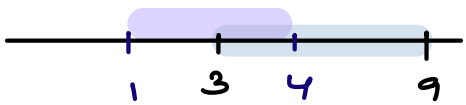
10:29 ⟶ 10:38 pm

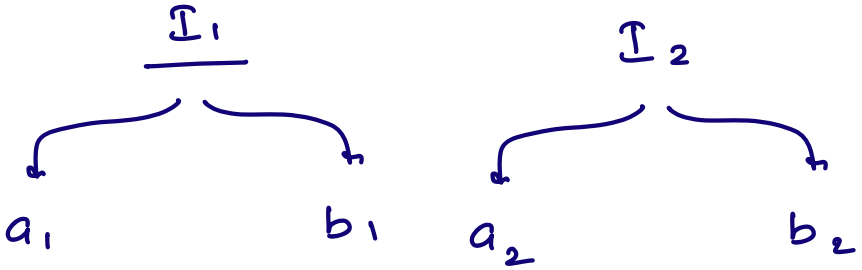
03. Merge Intervals



I_1	I_2	Merged interval
[2 6]	[3 7]	[2 7]
[2 8]	[4 6]	[2 8]
[2 4]	[5 7]	// No overlapping
[3 7]	[4 10]	[3 10]
[3 6]	[6 10]	[3 10]
[2 5]	[8 10]	// No overlapping
[5 8]	[1 3]	// No overlapping
[3 9]	[1 4]	[1 9]



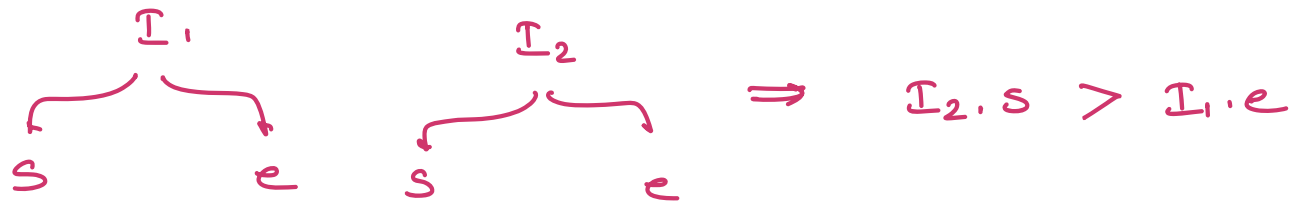


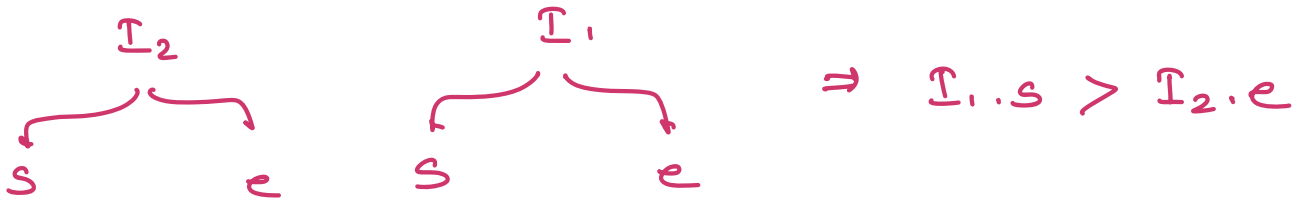


Merged Interval

ans = $(\min(a_1, a_2), \max(b_1, b_2))$

* How to check non-overlapping intervals





Q Given N non-overlapping intervals & they are sorted based on start time. You are given a new interval Merge the new interval & make all the non overlapping intervals

Eg: N = 7

New interval

[1 3]

[4 7]

[10 14] + {10 22} → {10 22}

[16 19] + {10 22} → {10 22}

[21 24] + {10 22} → {10 24}

[27 30] {10 24}

[32 35]

New intervals as non-overlapping

{1 3}

{4 7}

{10 24}

{27 30}

{32 35}

New interval → [12 22]

Eg: [1 5]

[8 10]

[11 14] + [12 22] = {11 22}

[15 20] + {11 22} = {11 22}

[20 24] + {11 22} = {11 24}

Non-overlapping

[1 5]

[8 10]

[11 24]

```
Interval []  mergeint ( Interval [] ar , Interval I)
    for ( i=0; i<n; i++){
        if ( ar[i].e < I.s) {
            ans. insert ( ar[i]);   //non overlapping
        }
        else if (ar[i].s > I.e)
            ans.insert (I);         //non overlapping
            for (j=i ; j<n; j++) {
                ans. insert (ar[j]);
            }
            return ans;
        }
```

```
        else {
            I.s = min( I.s, ar[i].s);
            I.e = max (I.e , ar[i].e);
        }
    }
    ans.insert (I):
    return ans:
}
```

TC: O(N)

SC: O(1)