Agenda

- → missing Integer (Amazon, Microsoft)
- → Search element in sorted matrix
- → Insert Interval

Q. First missing positive number.

Given an array. Find First missing positive number.
(N)

Ex ar[6] = {3, -1, 1, 2, 7, 0} ans = 4

ar[] = {1, 0, -5, -6, 4, 2} ans = 3.

arr[] = {-5, -4, -3, -1, 0} ans = 1

N = 5

B.F. Search from 1 to N+1 {1, 3, 2, 4, 5}

→ First missing number is ans.

TC: $O(N^2)$
SC: $O(1)$

Approach - 2 Using hashset

1. Put all elements in hs

2. for(Start from 1 to N+1)

```
if( hs.contains( i ) = = False)
        return i
```

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

N = 5

[12, 300, 40, 5, 100]

1 to 6

Idea 3 . Sorting

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

↓

var = ~~1~~ ~~2~~ ~~3~~ 4

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

↑

ans = 4

{-3, -7, 1, 1, 2, 5}

↓ sorted

var = ~~1~~ ~~2~~ 3

{-7, -3, 1, 1, 2, 5}

↑

```
// Sort the array

int val = 1

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

v=~~1~~
~~2~~
~~3~~
4

{1, 2, 3}

TC: O(NlogN + N) : O(NlogN)

SC : O(1)

BF	Hashset	Sorting
TC : O(N^2)	TC: O(N)	TC: O(NlogN)
SC : O(1)	SC: O(N)	SC: O(1)

↓ Expected

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

Idea 4 → Keep the elements (1 to N) at their correct position.

↓

0 1 2 3 4 5 6

arr[6] : 1, 2, ~~5~~, ~~6~~, ~~4~~, ~~9~~, 3

~~4~~ 4 5 6 9
~~6~~
~~9~~
3

ans = 7

element index

1	→	0
2	→	1
3	→	2
4	→	3
5	→	4
6	→	5

$$\begin{array}{lccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ ar[6] = & 1, & 2, & \cancel{5}, & \cancel{6}, & \cancel{4}, & \cancel{9}, & 3 \\ \cline{2-8} & & & \cancel{4} & 4 & 5 & 6 & 9 \\ & & & \cancel{6} & & & & \\ & & & \cancel{9} & & & & \\ & & & 3 & & & & \end{array}$$

Idea : Send each element to their correct positions.

→ whatever index don't have right canditate → ans

↓

$$\begin{array}{lccccccc} & 0 & 1 & 2 & 3 & 4 & 5 & 6 \\ arr[] = & \{1, & \cancel{-7}, & \cancel{5}, & 2, & \cancel{3}, & 10, & 11\} \\ \cline{2-8} & & 2 & 3 & -7 & 5 & & \end{array}$$

ans = 4.

```
int  i = 0
while ( i< N)
{
        if (  arr[i] > N  or  arr[i] < 1)  i++
        else
        {
            int  correct_ind  =  arr[i]-1
                if( correct_ind  == i)   i++
                else
                {  swap( a[correct_ind], a[i])


for( i=0; i < N; i++)
{
        if( arr[i]  != i+1)   return  i+1

return  N+1
```

TODO → handle duplicates

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

Q. Sorted 2D matrix (row-wise & col-wise sorted)

Check if element K is there or not

K = 15

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

ans = True

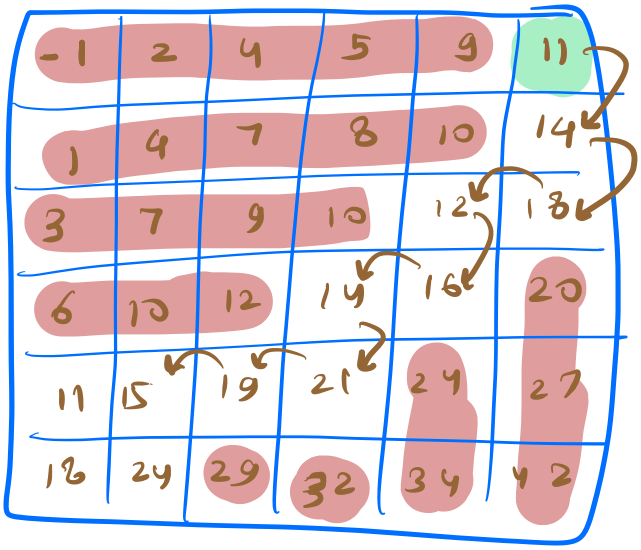
BF

TC: O(N × m)

BS : TC: O(N log m)

Start from Top-right

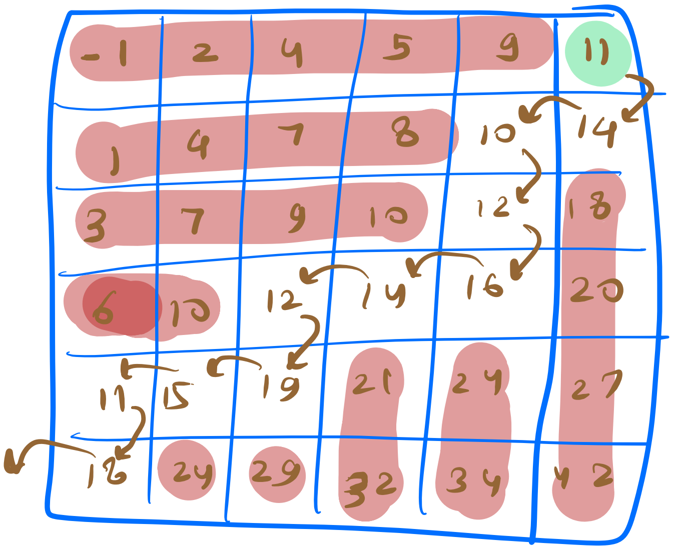
K = 15



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

ans = True

K = 13



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

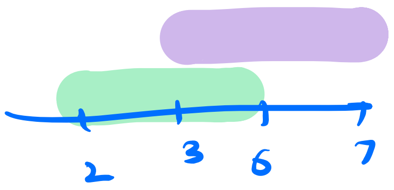
ans = False

```
i = 0, j = m-1

while( i<N && j≥0)
        if (arr[i][j] == k)
                return True
        else if ( arr[i][j] > k)
                j--
        else
                i++

return False
```

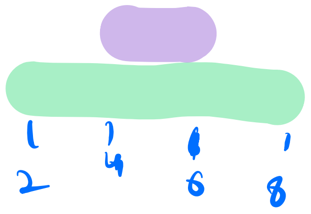
TC: O(N+m)

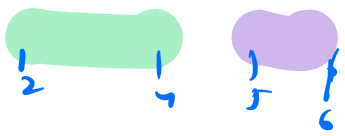
SC: O(1)

Q. Merge Interval

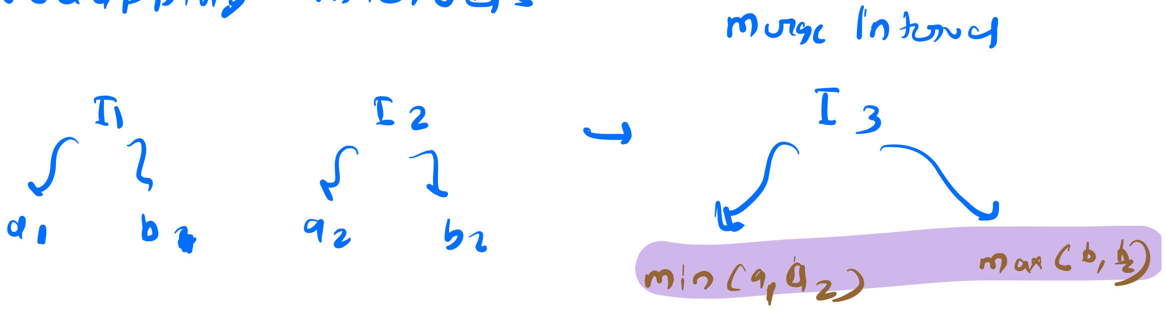


I_1	I_2		merged Interval
[2, 6]	[3, 7]	=	[2, 7]
[2, 8]	[4, 6]	=	[2, 8]
[2, 4]	[6, 7]	=	// No overlapping
[3, 7]	[4, 10]	=	[3, 10]
[2, 4]	[5, 6]	=	// No overlapping

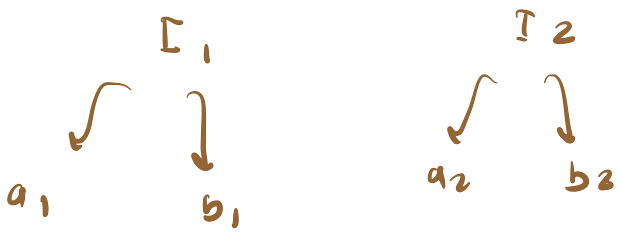




2 overlapping Intervals

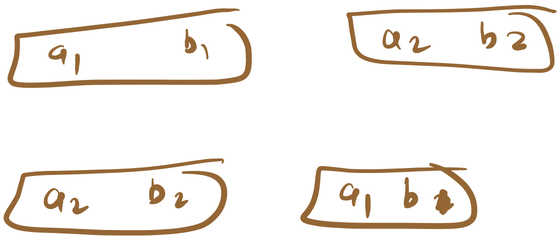


$a_1 < a_2$



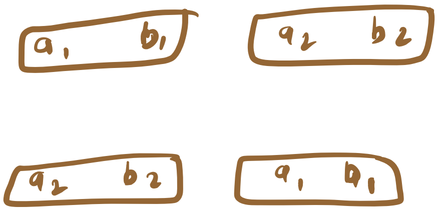
if ($b_1 < a_2$)

✓ Non overlapping.



If ($b_2 < a_1$)

non-overlaps ✓



If($b_1 < a_2$)

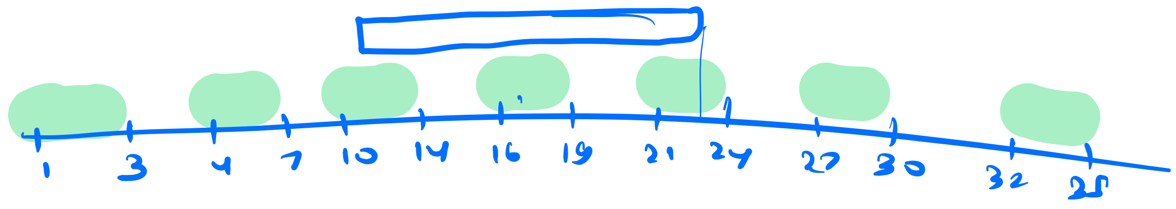
if ($b_2 < a_1$)

Non-overlapping.

Q. Given N non-overlapping Intervals. Given new interval. Merge this new Interval in the given intervals. [After merging → List of non overlapping Intervals]

[1, 3]
[4, 7]
[10, 14]
[16, 19]
[21, 24]
[27, 30]
[32, 35]

[10, 22]



⇓

[1,3] [4,7] [10,24] [27,30] [32,35]

List
(Non overlapping Intervals) + New Interval → List (Non overlapping Intervals)

input 1 input2. output

[10,22]
↓

[1,3]
[4,7]
~~[10,14]~~ + [10,22] = [10,22]
~~[16,19]~~ + [10,22] = [10,22]
~~[21,24]~~ + [10,22] = [10,24]
[27,30]
[32,35]

ans
[1,3]
[4,7]
10,24
27,30
32,35.

Interval → s
Interval → e

struct Interval
int s

```
int e

Interval[] mergeInt( Interval[] arr, Interval I)
{
    for( i=0; i<N; i++)
    {
        if( arr[i].E < I.s)                      I
        {     ans.insert( arr[i])                // non-overlap
        else if( arr[i].s > I.F)
        {
            ans.insert (I)                       III
            for( j=i; j<N; j++)
            {  ans.insert( arr[j])
            return ans
        }
        else
        {
            I.s = min( I.s, arr[i].s)
            I.e = max( I.e, arr[i].e)
        }
    }
}

ans.insert(I)
return ans.
```

TC: O(N)

SC : O(N) → O(1)

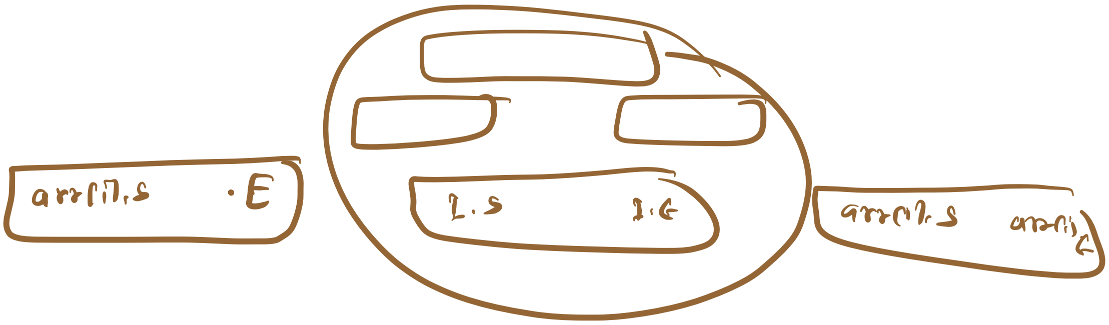
O output ✓

output ✗

[1, 3]
[4, 7]
~~[10, 14]~~
~~[16, 19]~~
~~[21, 24]~~

~~[10, 22]~~
[10, 24]

(1, 3)
(4, 7)



(1, 3)
(4, 7)
→ (15, 18)

(9, 10) →

(1, 3)
(4, 7)
(9, 10)
(15, 18)

(1, 3)
(4, 7)
~~(15, 18)~~

~~(10, 18)~~

(10, 18) →

→ (1, 3)
(4, 7)
10, 18

3 Phase.

1. Phase   arr[i] < Interval → insert arr[i]

2. Phase   I   arr[i] + overlapping

⇓

✓ I   mergeing them

3 Phase   I < arr[i] → I add

add $\boxed{arr[i]}$

1. Missing Integers (First positive)

1. 1 to N+1 search

   TC: O(N^2)

2. Hashset

   TC: O(N)

3. Sorting

   TC: O(Nlogn)

4. Swapping (correct position)

TC : O(N)

2. Search in 2D. [sorted]

o TR

K > TR ↓

K < TK ←

✓

return false

3. merge interval.

Phase 1 arr[i] < I

→ add arr[i]

Phase 2 arr[i] f I

↓

I

Phase 3 I < arr[i]

add(I)

add([arr[i]])

Phase I

I (1,3)
I (4,6)
II ~~(7,12)~~
II ~~(11,14)~~
III (15,18)

+

~~(9,10)~~
~~(7,12)~~
(7,14)

⇒

(1,3)
(4,6)

(7,14) Phase II

(15,18) III

[1, 2, 4, 3, 5]

1 to 6

1 ✓
2 ✓
3 ✓
4 ✓
5 ✓
6 X

ans = 6

[2,4]
[5,10]
[11,15]

+ [7,16]

=

[2,4]
[5,16]

✓C++