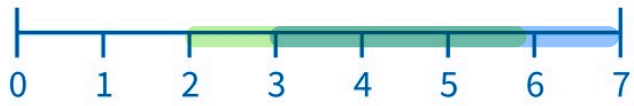
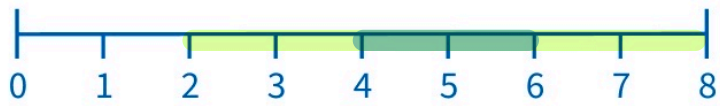
Good Morning

Topics

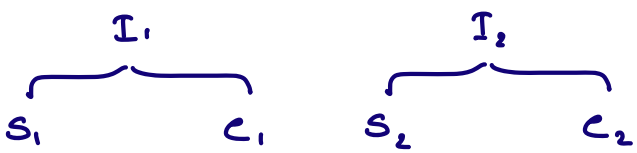
01. Merge Overlapping Interval
02. Merge Interval / Insert Interval
03. First Missing Integer

I_1	I_2		Interval
(2, 6)	(3, 7)	→	(2, 7)
(2, 8)	(4, 6)	→	(2, 8)
(3, 7)	(4, 10)	→	(3, 10)
(3, 6)	(6, 10)	→	(3, 10)
(2, 5)	(8, 10)	→	No overlap
(5, 8)	(1, 3)	→	No overlap
(3, 8)	(5, 12)	→	(3, 12)
(6, 10)	(8, 15)		(6, 15)

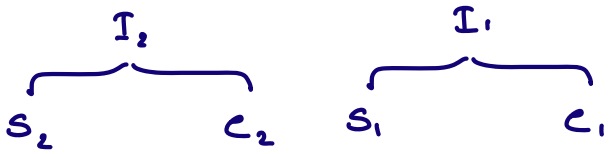




Non overlapping interval

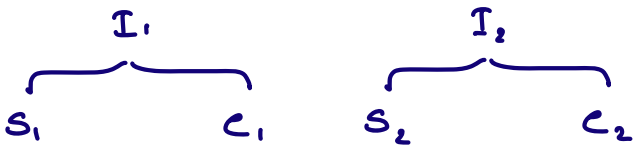


$\Rightarrow e_1 < s_2$



$\Rightarrow e_2 < s_1$

Overlapping Interval



Merged Interval

ans = (min(s1, s2) max(e1, e2))

TOPIC 2

Merge Sorted Overlapping Intervals

SCALER

Question

Given a sorted list of overlapping intervals, sorted based on start-time. Merge all overlapping intervals & return the sorted list of non.overlappping intervals.

Google

PayPal

$[1 \le N \le 10^5]$

Example:

st end

(13, 16)

Intervals[N] → [(0, 2), (1, 5), (5, 6), (6, 8), (7, 10), (8, 9), (12, 14)]

Interval 1	Interval 2		Merged	ans
(0, 2)	(1, 5)	Overlapping	(0, 5)	
(0, 5)	(5, 6)	Overlapping	(0, 6)	
(0, 6)	(6, 8)	Overlap	(0, 8)	
(0, 8)	(7, 10)	Overlap	(0, 10)	
(0, 10)	(8, 9)	Overlap	(0, 10)	
(0, 10)	(12, 14)	No overlaping		(0, 10) (12, 14)

(12, 14) (13, 16) Overlap (12, 16) → (12, 16)

If last interval is also overlaping, then we have to push it explicitly

st end
Intervals = ((1, 3) (4, 6) (6, 7) (8, 10) (9, 13) (20, 21))

Interval 1	Interval 2		Merged	Ans
(1, 3)	(4, 6)	No overlap		(1, 3)
(4, 6)	(6, 7)	Overlap	(4, 7)	
(4, 7)	(8, 10)	No overlap		(4, 7)
(8, 10)	(9, 13)	Overlap	(8, 13)	
(8, 13)	(20, 21)	No overlap		(8, 13)
				(20, 21)

↓

st end
A[] = { (0 2), (2, 6) (3, 8) (9, 10) (10, 16) }
Interval

st = 9
e = 16

ans = { (0, 8) (9, 16) }

Code

Merge Overlapping Intervals

```
List < Interval> ans = new AL<>();

    int  st= A[0].start;

    int  e  = A[0]. end;


  for ( i=1 ; i< A.length; i++){

     if ( A[i].start ≤ e ){        // overlapping

        st= Math.min ( st, A[i]. start );

        e  = Math.max ( e, A[i]. end);

     }
     else {

        ans.add (new  Interval ( st, e);

        st = A[i]. start ;

        e  = A[i]. end;

     }

  }

  ans.add ( new  Interval (st, e));
```

TC : O(n)

SC : O(1)

Question

Given N non-overlapping intervals sorted based on start time. Given a new Interval. Merge this with existing intervals if possible & return final non-overlapping interval.

N = 8

I = (10,22)

[(1 , 3),(4 , 7),(10 , 14),(16 , 19),(21 , 24),(27 , 30),(32 , 35), (38 , 45)]

(10 22)

Resultant Intervals → (1,3) (4,7) (10,24) ,,

N = 5

I = (12,22)

Intervals( ) → [(1 , 5),(8 , 10),(11 , 14),(15 , 20),(21 , 24)]

I = (12 , 22)

Resultant Intervals → (1,5) (8,10) (11,24)

Eg:-

New Interval

Non overlapping Intervals

(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)

(10,24)

(10,24)

(27,30)

(32,35)

* (1,5) (6,10) (9,12)

New Interval

Ans

(1,5) + (4,7) → (1,7)

(6,10) + (1,7) → (1,10)

(9,12) + (1,10) → (1,12)

(1,12)

```
list < Interval > ans = new ArrayList <>();

for ( i=0; i<n; i++){

    if ( A[i].e < I.start){

        ans.add(A[i]);

    }

    else if ( A[i].start > I.end ){

        ans.add(I);

        for ( j= i ; j <n; j++){

            ans.add (A[j]);

        }

        return ans;

    }

    else {

        I.start = Math.min ( I.start, A[i].start);

        I.end   = Math.max ( I.end, A[i].end);

    }

}

ans.add (I)

return ans;
```

//non overlaping
before new interval

//non overlaping
after merging
new Interval

8:16 AM → 8:30 AM

Question

Find the first missing natural number

N=length of array

Example 1:

arr[5] → [3, -1, 1, 2, 7] ⟶ Ans = 4

Example 2:

arr[7] → [9, 2, 6, 4, -8, 1, 3] ⟶ Ans = 5

Example 3:

arr[6] → [1, 0, -5, -6, 4, 2] ⟶ Ans = 3

Example 4:

arr[6] → [1, 2, 5, 6, 4, 3] ⟶ Ans = 7

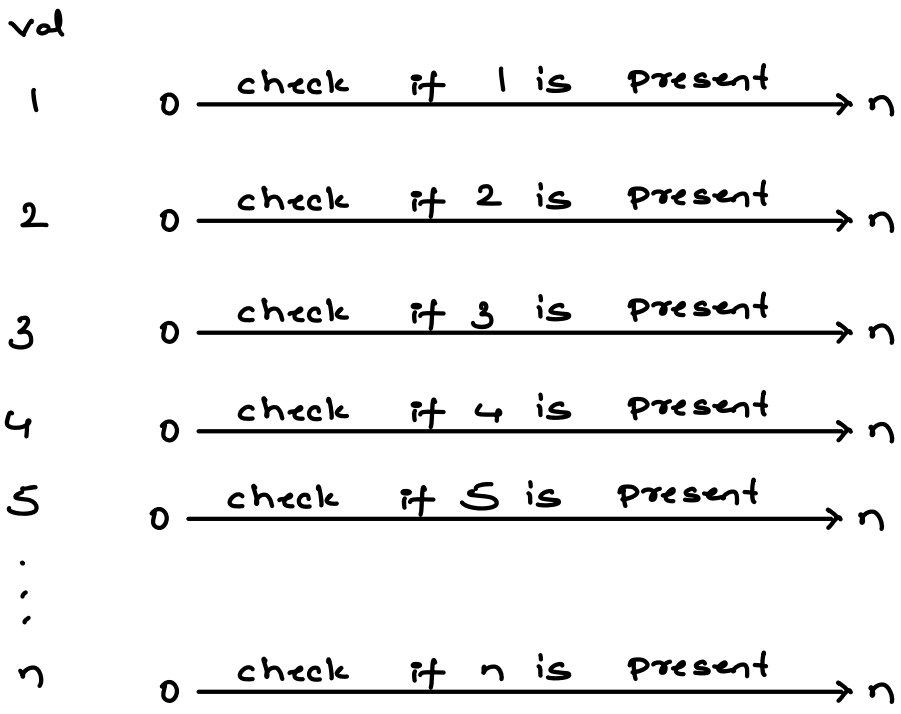
Example 5:

arr[4] → [1, 2, 3, 4] ⟶ Ans = 5

A[] → {5 3 1 -1 -2 -4 7 2} ⟶ Ans = 4

A[] → {5, 6, 7} ⟶ Ans = 1

Brute force → Start searching from 1 to n+1 in your array



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

Idea 2 Sort the array

A[] = { -3, -7, 1, 10, 3, 8, 2 }

sort

A[] = { ~~-7~~, ~~-3~~, 1, 2, 3, 8, 10 }

val = 1 2 3 (4)

Ans = 4

* A[] = { -3, 1, 1, 1, 2, 4 }

val = 1 2 3

Ans = 3

```
Arrays.sort(A)                    ---------->  nlogn
int val = 1
for (i=0; i<n; i++) {
    if (A[i] < 1) continue;
    if (A[i] == val) val = val+1;
    else if (A[i] == val-1) continue;
    else { break }
}
return val;
```

TC : O(nlogn)

SC : O(1)

A[] = { 1, 1, 1, 2, 3, 4, 4, 7 }

val = 1 2 3 4 5

Ans = 5

A[] = { -4, -3, -2 }

val = 1

Note → TC : O(n)
SC : O(1)

by making that place negative

Mark the presence of these elements at their correct position in sorted A.

* 1 2 3 4

0 1 2 3 → index

A[] = { -2 -3 -6 1 -5 }

0 1 2 3 4

Ans idx+1

A[] = { 2 1 1 5 3 3 }

A[] = { -2 -1 -1 5 -3 3 }

0 1 2 3 4 5

Ans = idx+1

If -ve elements are present or 0

convert them to ≥ n+2

A[] = { -8 4 5 -4 -8 1 8 }

0 1 2 3 4 5 6

Ans = idx + 1

```
for (i=0; i<n; i++){
    if (A[i] < 1) A[i] = n+3;
}
```

// put a value whose idx is not present in array (correct)

```
for (i=0; i<n; i++){
    int ele = Math.abs(A[i]);
    int idx = ele - 1;
    if (idx < n){
        A[idx] = -1 * Math.abs(A[idx]);
    }
}
```

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

A[] = { 6 2 3 4 5 }

0 1 2 3 4

1 swap = 4

2 swap = 2

3 swap = 3

4 swap = 5

```
int i = 0

while (i<n)
|
|   if (ar[i] >= 1 && ar[i] <= N){
|   |
|   |   int correct_idx = ar[i] - 1;
|   |
|   |   if ( ar[correct_idx] != ar[i]){
|   |   |
|   |   |   swap ( ar[correct_idx], ar[i]);
|   |   3
|   |   else{
|   |   |   i++;
|   |   2
|   3
|   else{
|   |
|   |   i++;
|   2
3

for ( i=0; i<n ; i++){
|
|   if (ar[i] != i+1) return i+1
|
3
return n+1;
```

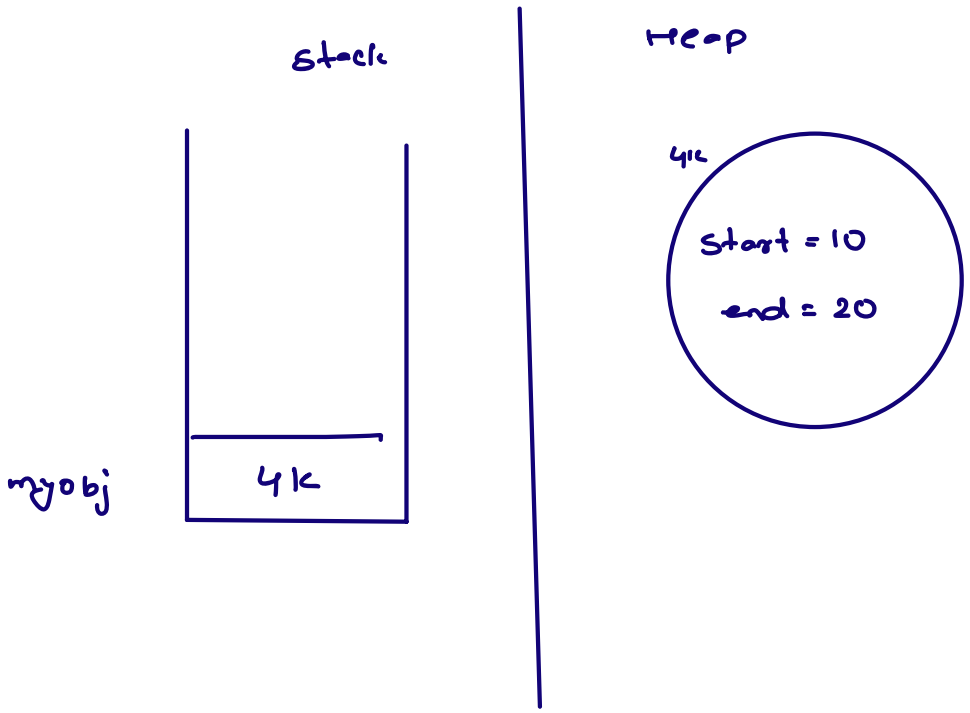
TC: O(n)

SC: O(1)

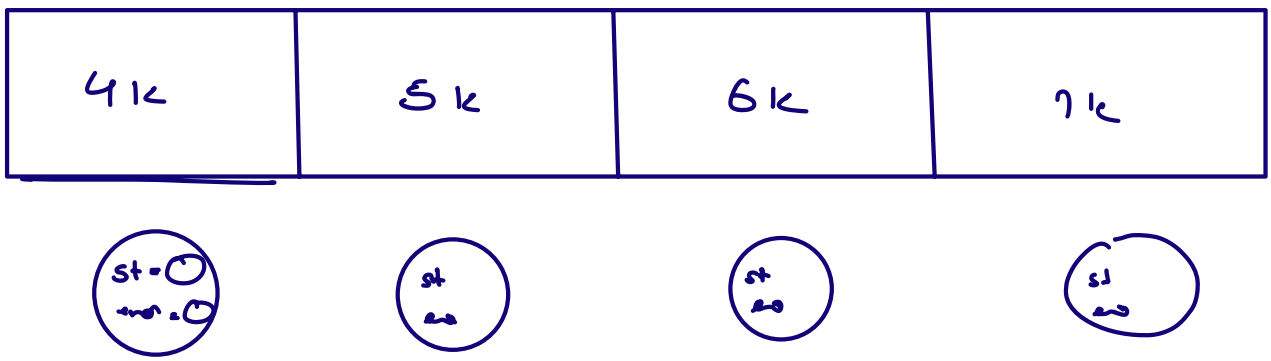
Doubts

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

```
Interval myobj = new Interval (10,20);
```




```
Interval [] A = new Interval [];
```



A[0].start =

A[0].end =