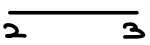
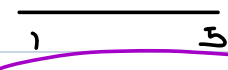
Merge Intervals

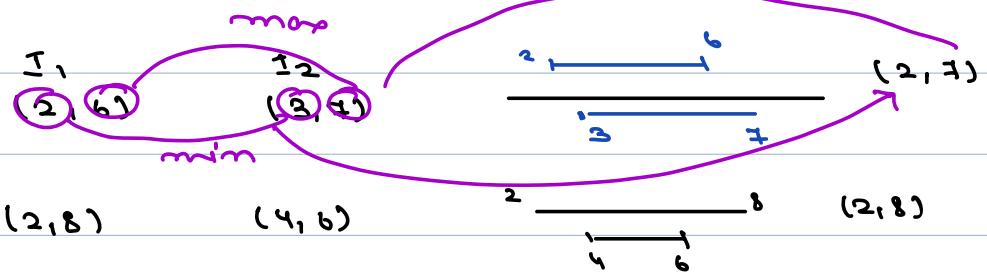
Interval s, e

(2, 3)



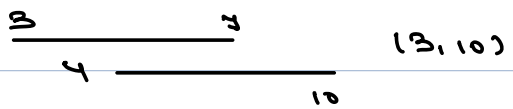
s, e (1, 5)





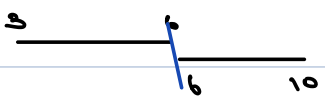
(2,8) (4,6) (2,8)

(3,7) (4,10)



(3,10)

(3,6) (6,10)



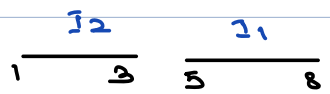
(3,10)

(2,5) (8,10)

(no overlap)

I1 I2

(5,8) (1 3)



(no overlap)

Non - Overlapping

$I_1 \{s_1, e_1\}$, $I_2 \{s_2, e_2\}$

1) I_1: s_1 — e_1 $\quad$ I_2: s_2 — e_2 $\quad$ $[s_2 > e_1]$

2) I_2: s_2 — e_2 $\quad$ I_1: s_1 — e_1 $\quad$ $[s_1 > e_2]$

```
if (s2 > e1 || s1 > e2) {
    non overlapping
}
```

How to merge Overlapping Intervals

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

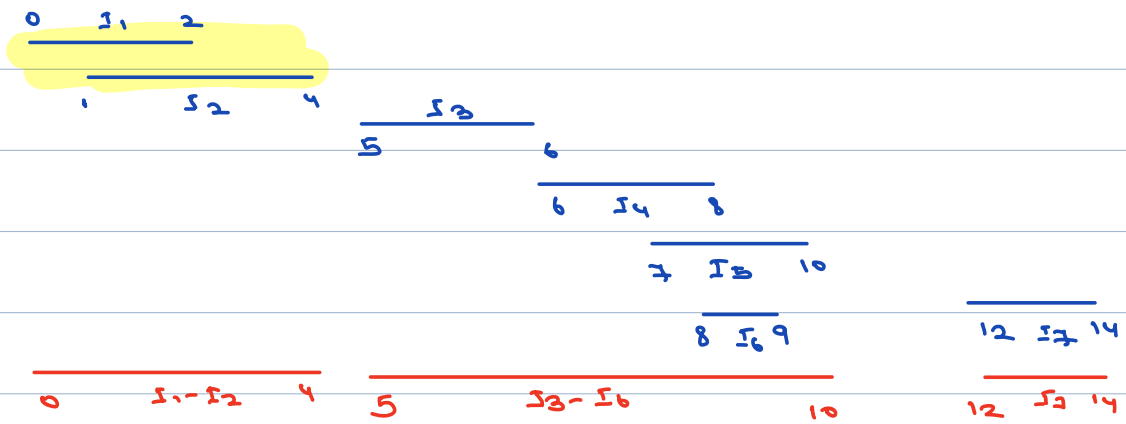
After merging

- start time → $\min(s_1, s_2)$
- end time → $\max(e_1, e_2)$

Ques

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

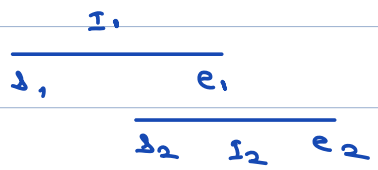
Interval [] = { (0,2), (1,4), (5,6), (6,8), (7,10), (8,9), (12,14) }

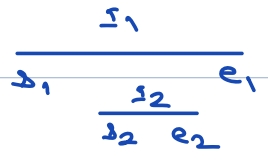


non-overlapping intervals

[(0-4), (5,10), (12,14)]

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





Overlapping

$s_2 <= e_1$

Interval [] = { (0,2), (1,4), (3,6), (6,8), (7,10), (8,9), (12,14) }

0 1 2 3 4 5 6

ans = (0,2)

ans	next available	After merged	final Ans
(0,2)	(1,4)	(0,4)	
(0,4)	(3,6)		(0,4)
(3,6)	(6,8)	(3,8)	(0,4)
(3,8)	(7,10)	(3,10)	(0,4)
(3,10)	(8,9)	(3,10)	(0,4)
(3,10)	(12,14)		(0,4) (3,10)
→ (12,14)			(0,4) (3,10) (12,14)

int [] ans ; → ans[1] → in2

// Interval [] arr; -> arr[i] -> arr[i].s
                          -> arr[i].e

s e
]/

List<Interval> ans;

arr_start = arr[0].s, arr_end = arr[0].end

for(i=1; i<n; i++) {

    // the next interval overlaps with curr
    if (arr[i].s <= arr_end) {
        //merging them
        curr_end = Max(arr_end, arr[i].end);
    }
    else {
        // push curr interval to ans
        // create a new interval
        Interval temp(arr_start, arr_end);
        ans.push_back(temp);
        //update curr interval with new
        //interval
        arr_start = arr[i].s;
        arr_end = arr[i].e;
    }
}

Interval temp(arr_start, arr_end);
ans.push_back(temp);
return ans;

T.C -> O(n).

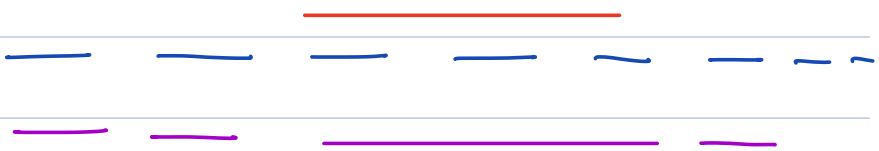
S.C -> O(1).

Break 8:12 Am - 8:20 Am

Ques

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

N = 9
(1,3)
(4,7)
(10,14)
(16,19)
(21,24)
(27,30)
(32,35)
(38,41)
(43,50)



new interval (12,22)

N = 9
(1,3)
(4,7)
(10,14)
(16,19)
(21,24)
(27,30)
(32,35)
(38,41)
(43,50)

new interval

(12,22)
(12,22)
(12,22) : (10 22)
(10,22) : (10,22)
(10,22) : (10,24)
(10,24)

Ans

(1,3)
(4,7)
(10,24)

e.g 2)

	new interval	Ans
(1,5)	(12,24)	(1,5)
(8,10)	(12,24)	(8,10)
(11,14)	(12,24) (11,24)	
(15,20)	(11,24) : (11,24)	
(21,24)	(11, 24) : (11,24)	
		(11,24)

(ns, ne) → new interval.

```
for (i=0; i<m; i++) {
    cInterval = Interval[i];
    // non Ovwlapping
    if (ns > cInterval.e) {
        Print (cInterval);
    }
    else if (cInterval.s > nE) {
        Print (ns,ne);
        for (j=i; j<m; j++) {
            Print (Interval[j].s, Interval[j].e);
        }
        return;
    }
    else {
        ns = min(cInterval[i].s, ns);
        ne = max (cInterval[i].e, ne);
    }
}
Print (ns, ne);
```

T.C → O(n)

S.C → O(1).

Ques) Given an unsorted array of integers, Find first missing Natural Number.

(1, 2, 3, .. ∞)

arr[5] => {3, -2, 1, 2, 7} → Ans 4.

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

{-2, 4, -1, -6, 3, 7, 8, 4, -3} → Ans 1.

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

{1, 2, 5, 6, 4, 3} Ans → 7

{-4, 8, 3, -1, 0} Ans → 1

{4, 2, 1, 3} Ans → 5

Observations

N

min 1

Max poss ans → N+1

5 → 1 to 6

— — — — —

= 6

ans will always be [1, N+1]

Sol 1:- **Brute force**

for (i=1; i<=N; i++) {

Brute force

for (j=0; j<n; j++) {
  search for i.
}

Check from 1 to n which no' is missing.

↳ found return that no'

↳ if not return N+1.

T.C → $O(n^2)$.

idea 2 use hashset

for (i=1; i<=n; i++) {
  searching i
}

T.C → O(n)

S.C → O(n).

idea 3 use sorting

{-2, 4, -1, -6, 3, 7, 8, 4, -3}

↓

-6, -3, -2, -1, 3, 4, 7, 8 - -

T.C → O(n log n).

idea Ashish

_1_ _1_ _1_ _1_

$\frac{4 \times (4+1)}{2}$ ⇒ 10

1 → 10 → 90.

idea :- keep an element into its right position.

N = 5

val	idx
1	0
2	1
3	2
4	3
x	x-1

n → 0 → 1
n → n-1 → n

valid no. (1 to n).

8 (1 to 8)

Do 5.

	0	1	2	3	4	5	6	7
arr[8] =	~~4~~ 5	2	~~7~~ 8	~~6~~ 4	9	1 6	~~8~~ 7	~~3~~ 8

i	arr[i]		
0	4	s(arr[0], arr[3])	s(arr[0], arr[5]) : set
1	2	set	
2	7	s(arr[2], arr[0])	s(arr[2], arr[7]) : set
3	4	set	
4	9	ignore	
5	6	set	
6	7	set	
7	8	set	

:	while	
0	-	
1	-	
2	-	
3	-	
4		
n	n-1	

10

arr[10] = 5 -14 6 7 9 -10 2 8 1 8

idx	val
0	5
1	-14
2	6
3	7

dfs(idx, val-1);

n → 1 to n / 0 to n-1

```
for (i=0; i<n; i++) {
    while (arr[i] > 0 && arr[i] <= n &&
           arr[i] != i+1) {
        int val = arr[i];
        if(arr[i] == arr[val-1]) { break }
        swap ( i , val-1);
    }
}
return n+1;
```

valid

let us see.

3

3

Iterate & get missing no

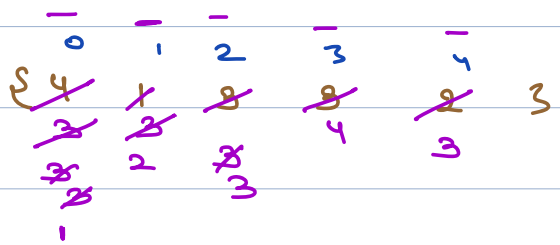
```
for(i=0; i<n; i++){
    if(arr[i] != (i+1)){
        return i+1;
    }
}
return n+1;
```

3

3

T.C → O(n)

// duplicacy

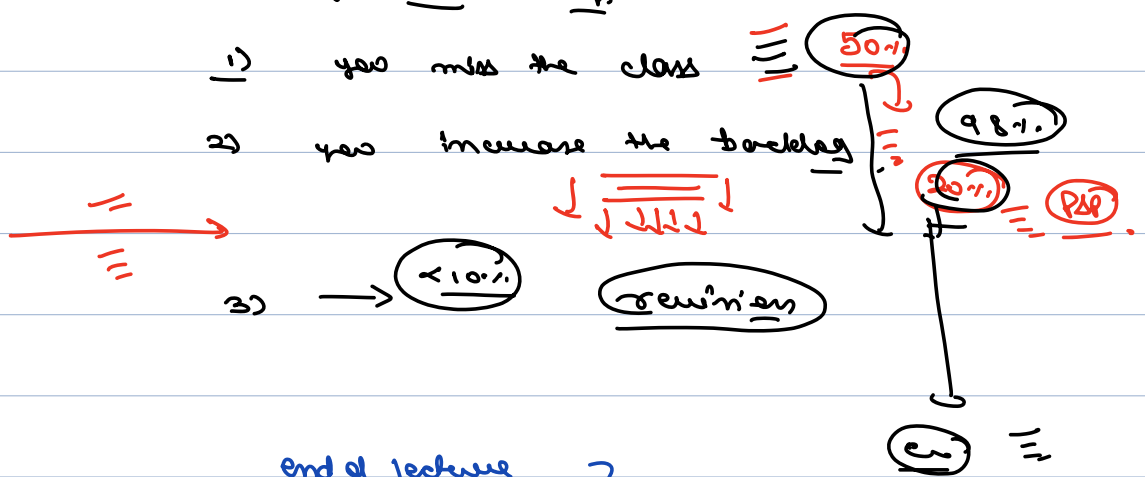


Difference ways

1) you miss the class

2) you increase the backlog

3) → <10%



end of lecture

weekend

month end