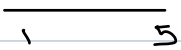
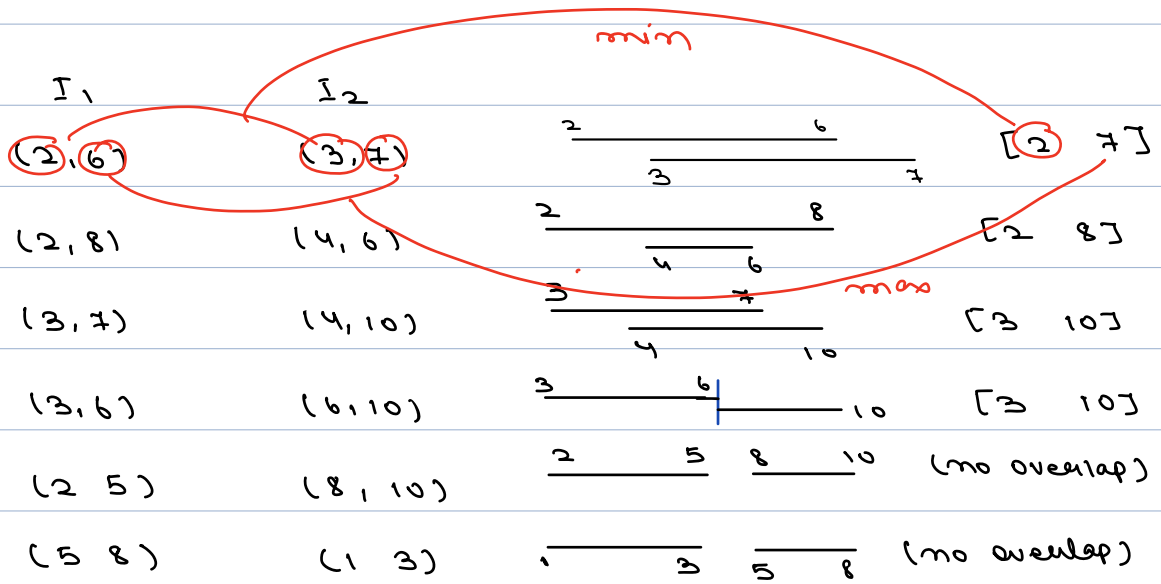
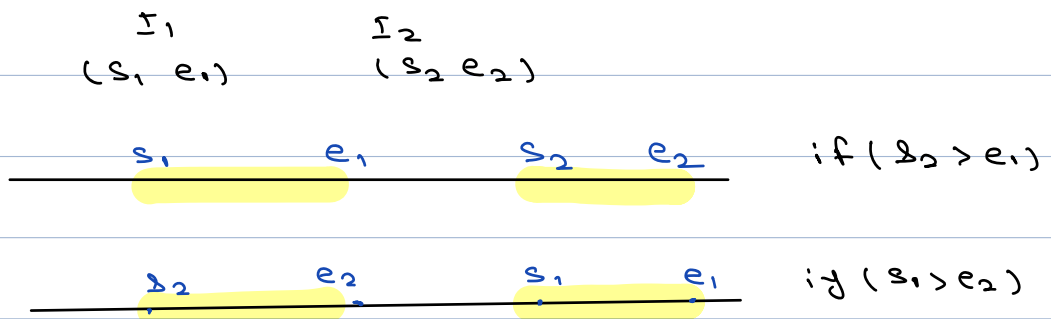
Merge Interval

Interval [s e] → [2 3]

[1 5]





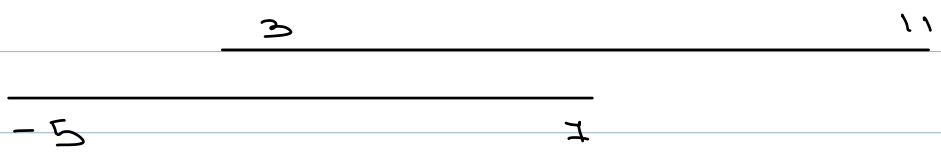


if (s2 > e1 || s1 > e2) → no overlap

if overlap → merged interval

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

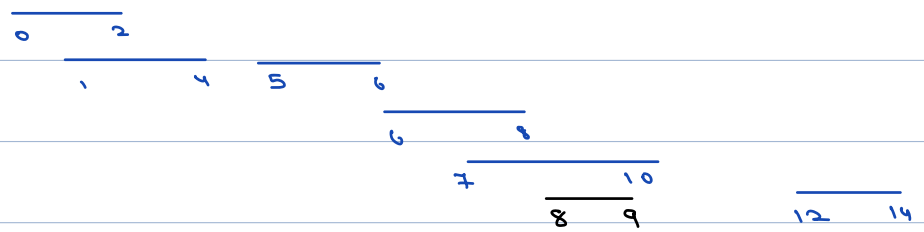
$(-5, 7)$ $(3, 11)$ $\longrightarrow$ $[-5, 11]$.



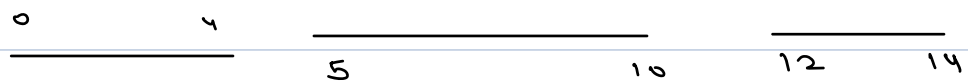
Ques

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

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

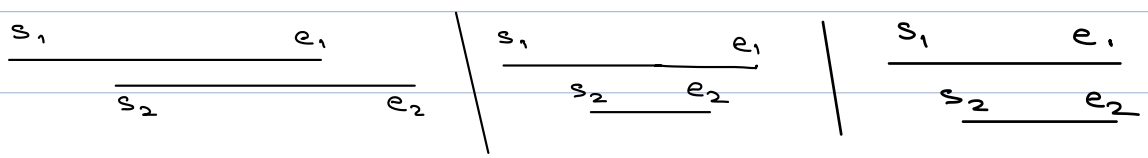


$\Rightarrow$

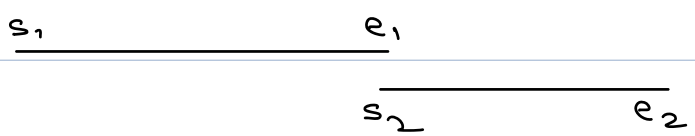


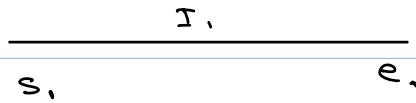
Ans = {(0,4), (5,10), (12,14)}

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

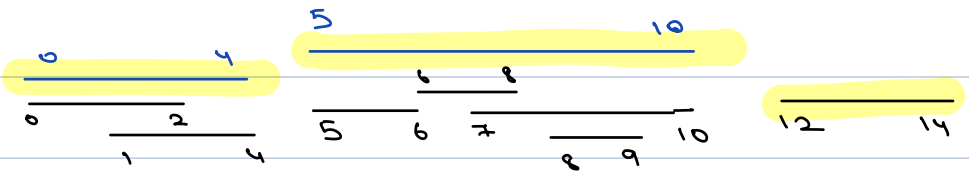


Overlap $\rightarrow$ $s_2 <= e_1$





Overlap → $s_2 <= e_1$



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

Curr	most Available	After merged	final ans
(0-2)	(1,4)	(0,4)	—
(0-4)	(5,6)	—	(0,4)
(5,6)	(6,8)	(5,8)	(0,4)
(5,8)	(7,10)	(5,10)	(0,4)
(5,10)	(8,9)	(5,10)	(0,4)
(5,10)	(12,14)	—	(0,4), (5,10)
(12,14)	—	—	[(0,4) (5,10), (12,14)]

int[] arr → int,

Interval[] arr → arr[0].s & arr[0].e

```
Interval [] arr;          // Given.
List<Interval> ans;
curr_start = arr[0].s, curr_end = arr[0].e
for (i=1; i<n; i++) {
    if (arr[i].s <= curr_end) {   //overlap
        //merge
        curr_end = max(curr_end,
                       arr[i].e);
    }
    else {
        Interval temp (curr_start, curr_end);
        ans.push_back(temp);
        curr_start = arr[i].s;
        curr_end = arr[i].e;
    }
}
Interval temp (curr_start, curr_end);
ans.push_back(temp);
return ans;
```

T.C → O(n)

S.C → O(1).

Break 10:10pm - 10:20pm

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

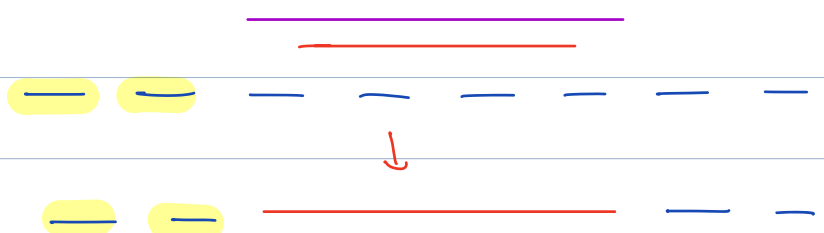
Curr	next Available	After merged	final ans
(0-2)	(1,4)	(0,4)	—
(0-4)	(5,6)	—	(0,4)
(5,6)	(6,8)	(5,8)	(0,4)
(5,8)	(7,10)	(5,10)	(0,4)
(5,10)	(8,9)	(5,10)	(0,4)
(5,10)			

Ques

Given a sorted list of overlapping (non) 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) (12,22) → (1,3)

(4,7) (12,22) → (4,7)

(10,14) (12,22) : (10,22)

(16,19) (10,22) : (10,22)

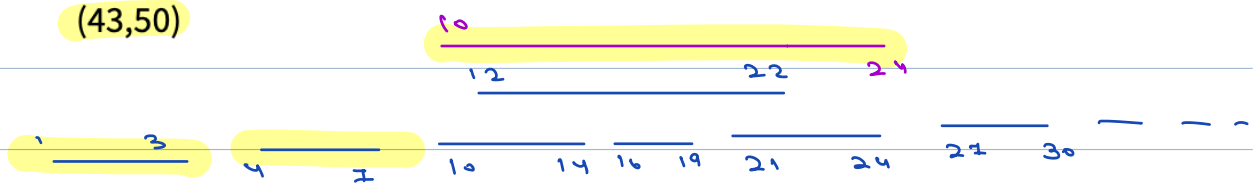
(21,24) (10,22) : (10,24)

(27,30) (10,24) → (10,24)

(32,35)

(38,41)

(43,50)



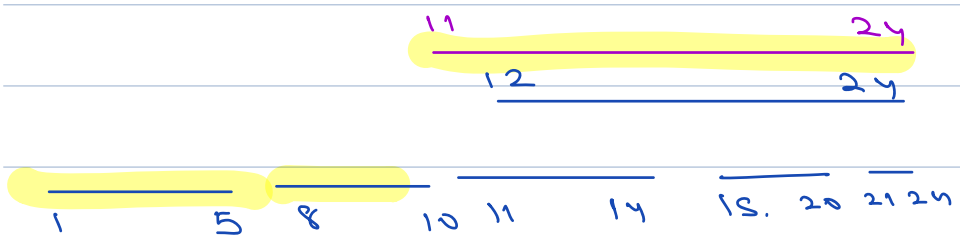
(1,5) (12,24) → (1,5)

(8,10) (12,24) → (8,10)

(11,14) (12,24) : (11,24)

(15,20) (11,24) : (11,24)

(21,24) (11,24) : (11,24)



ns ne

T.C → O(n)
S.C → O(1).

cint
s e

(ns, ne) → new interval.

```
for (i=0; i<n; i++) {
    cInterval = interval[i];
    //non Overlapping
    if(ns > cInterval.e) {
        print(cInterval);
    }
    else if (cInterval.s > ne) {
        print(ns, ne);
        for(j=i; j<n; j++) {
            print(interval[j].s,
                  interval[j].e);
        }
        return;
    }
    else {
        ns = min(cInterval.s, ns);
        ne = max(cInterval.e, ne);
    }
}
print(ns, ne);
```

CI	mI	
(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)

N = 9

mI.

(1,3)	(12,22)	→ (1,3)
(4,7)	(12,22)	→ (4,7)
(10,14)	(12,22)	: (10,22)
(16,19)	(10,22)	: (10,22)
(21,24)	(10,22)	: (10,24)
(27,30)	(10,24)	
(32,35)		
(38,41)		
(43,50)		

Ques

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

1 2 3 4 5 .. ∞

arr[5] = {3, -2, 1, 2, 7} → 4 Ans.

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

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

{1, 0, -5, -6, 4, 2} → 3 Ans.

{1, 2, 5, 6, 4, 3} → 7 Ans.

{-4, 8, 3, -1, 0} → 1 Ans.

{4, 2, 1, 3} → 5 Ans.

Observation → n → 1 to n+1.

1
5

1 2 3 4 5 → 6

1, 2, 2, 3, 4 → 5 Ans.

Soln 1:- Brute force n → 1 to n+1 Ans.

Check all numbers from 1 to n+1

n = 7.

```
for (i=1; i<=n; i++) {
    → check no' in array.
}
return n+1;
```

T.C → O(n²).

Soln 2:-

Add all elements to the set and check if element(1 to N) is present or not.

T.C → O(n)
S.C → O(n)

Soln 3:- Sort & then travel & check.

T.C → O(n log n),

Soln 4:- T.C → O(n), S.C → O(1).

Keep an element into its right pos^n.

N = 5

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

n → 0 → 1
n → n-1 → n

arr[8] =

0	1	2	3	4	5	6	7
~~4~~ 6 1	2	~~7~~ ~~8~~ 3	~~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[6]) S(arr[2], arr[7]) : set
3	4	set
4	9	ignore.
5	6	set
6	7	set
7	8	set

10 → 0 → 1
10 → 9 → 10

arr[10] =

0	1	2	3	4	5	6	7	8	9
~~5~~ 9	~~-14~~ 2	~~6~~ -10	~~7~~ ~~2~~ -14	~~9~~ 5	~~-10~~ 6	~~2~~ 7	~~8~~ ~~-10~~ 8	~~x~~ 9	~~8~~ -10

i	arr[i] = val	
0	5	set
1	-14	ignore
2	6	
3	7	

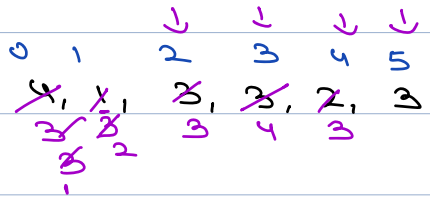
set(i, val-1)
0 → 4

```
for (i=0; i<n; i++) {
    while( arr[i]>0 && arr[i]<=n &&      // valid
           arr[i]!=i+1 ) {               // not set
        int val= arr[i];
        if(arr[i]==arr[val-1]) {break}
        swap(arr,i,val-1);
    }
}
```

Iterate & find missing no'.

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

Duplicate issue,



i arr[i]

6 4

T.C → O(n)
S.C → O(1)