Interview Problems

Content

- Merge Sorted Overlapping Interval
- Sorted Non Overlapping intervals
- First Missing Positive

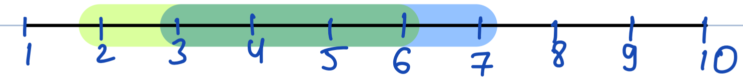
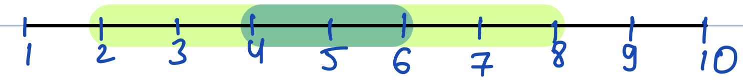
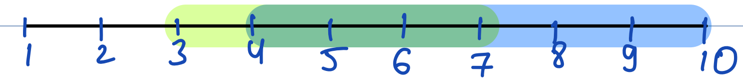
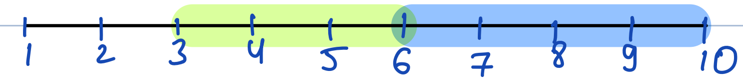
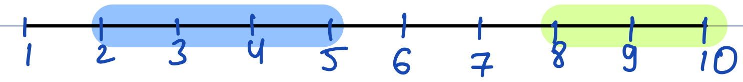
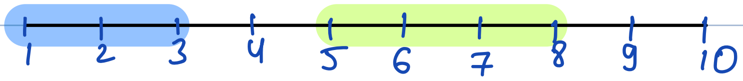
Before Advanced   8th Jan

75%   65% → 75%

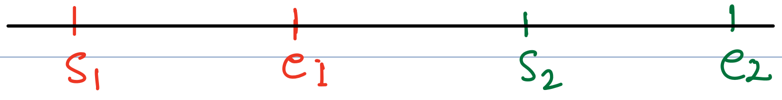
NOTE : Only assignment problems count in PSP calculation.

Merge Interals

Interval — (s e)

I₁	I₂		overlapping s	e
(2 6)	(3 7)		2	7
(2 8)	(4 6)		2	8
(3 7)	(4 10)		3	10
(3 6)	(6 10)	 overlap	3	10
(2 5)	(8 10)		no	overlap
(5 8)	(1 3)		no	overlap

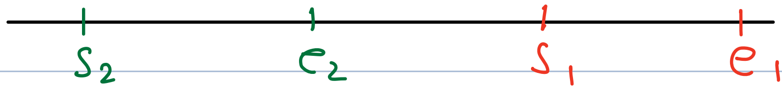
Non - overlapping condition.



$e_1 < s_2$

||

$e_2 < s_1$



Merge 2 overlapping intervals

Given two overlapping intervals $[s_1, e_1]$ $[s_2, e_2]$

Print merged interval

$s = \min(s_1, s_2)$

$e = \max(e_1, e_2)$

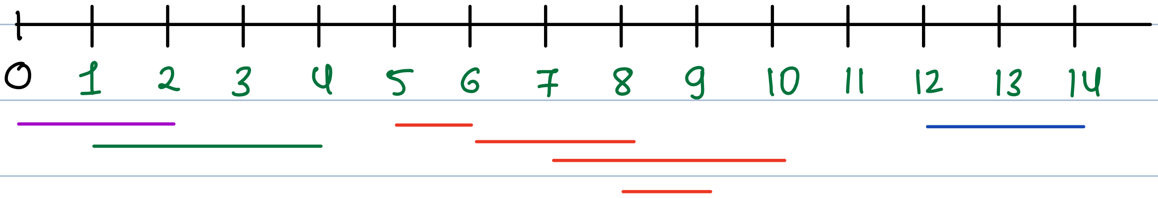
Merge Sorted Overlapping Interval

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

start = [0 1 5 6 7 8 12]

end = [2 4 6 8 10 9 14]

output 0 – 4 5 – 10 12 – 14



s_A e_A s_B e_B

sorted Based on start time $s_a < s_b$

non – overlapping condition $e_A < s_B$

Pseudocode

```
void   mergeOverlapping ( start[], end[] ) {
        S = start[0]
        E = end[0]

        for i ⟶ 1 to N-1 {
            // non overlapping condition
            if ( E < start[i] ) {
                print ( S , E )
                S = start[i]
                E = end[i]
            }
            else {
                ~~S = min ( S , start[i] )~~
                E = max ( E, end[i] )
            }
        }
}
        print ( S, E )
```

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

optional since sorted on start times.

start = [0 1 5 6 7 8 12]

end = [2 4 6 8 10 9 14]

S ~~0~~ ~~5~~ 12

E ~~2~~ ~~4~~ ~~6~~ ~~8~~ ~~10~~ ~~10~~ 14

0 4
5 10
12 14

Custom comparator

NOTE : If intervals are not sorted, sort them.

Sorted Non Overlapping intervals

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

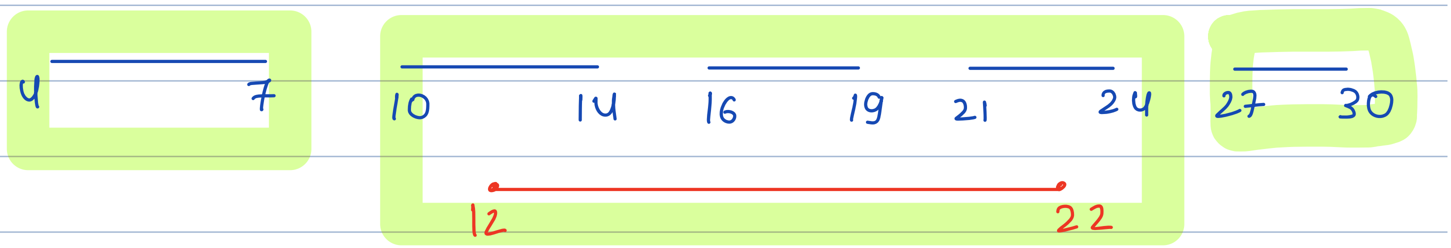
0 1 2 3 4

start = [4 10 16 21 27]

end = [7 14 19 24 30]

L = 12
R = 22

} new interval



output

4 7
10 24
27 30

Approach 1 ⟶ Modify the given input

Insert new interval at its correct position based on start time

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

0 1 2 3 4

start = [4 10 16 21 27]

end = [7 14 19 24 30]

L = 12
R = 22

} new interval

Output

4 - 7

case 1> $[s_i \quad e_i]$ new Interval $[L \quad R]$

print $(s_i \quad e_i)$

Case 2> new Interval $[L \quad R]$ $[s_i \quad e_i]$

print (L R)

print rest of the remaining intervals.

case 3) merge new Interval $[L \quad R]$ $[s_i \quad e_i]$

$L = \min(L, s_i)$

$R = \max(R, e_i)$

Pseudocode

```
void nonOverlapping ( start[], end[], L , R ) {
    for i ---> 0 to N-1 {
        // Case 1
        if ( end [i] < L ) {
            print (start [i], end[i])
        }
        // Case 2
        else if ( R < start[i]) {
            print ( L , R)
            // Print rest of the remaining intervals
            for j ⟶ i to N-1 {
            |       print ( start[j] , end[j] )
            }
            return
        }
        else { // case 3  intervals are overlapping
            L = min (L, start[i])
            R = max (R, end[i])
        }
    }

    print ( L , R )
}
```

TC : O(N)

SC : O(1)

$$
\begin{array}{llcccccc}
 & & 0 & 1 & 2 & 3 & 4 & \\
\text{start} & = [& 4 & 10 & 16 & 21 & 27 &] \\
\text{end} & = [& 7 & 14 & 19 & 24 & 30 &]
\end{array}
$$

L = 8

R = 9

Output

```
4  7
8  9
10 14
16 19
21 24
27 30
```

Break :

22 : 42

First Missing Positive

Given an integer array A[] , find the first missing the integer.

	Output
A = [3 -2 1 7 2]	4
A = [-8 7 2 5 3]	1
A = [4 1 3 2]	5
A = [-9 2 6 4 -8 1 3]	5
A = [1 2 5 6 4 3]	7
A = [-4 8 3 -1 0]	1

Bruteforce

min Ans = 1

max Ans = N + 1

TC : O(N²)

SC : O(1)

```
for ans ⟶ 1 to N+1 {
    if ans is not present in A    return ans
}
```

A = [-4 -3 -2 -1 0]   ans = 1

A = [4 1 6 3]

Approach 2 ⟶ Sort the array

A = [-9 2 6 4 -8 1 3]   ans = 5

-9 -8 1 2 3 4 6

Find the index of 1 if 1 is not present then 1 is the ans

Rest think as HW.

Approach 3

i) Add all elements to set.   TC : O(n)

SC : O(N)

```
for ans ⟶ 1 to N+1 {
|       if ans is not present in set return ans
}
```

checking ans in set is O(1) operation.

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

Hint ⟶ we can modify array.

$$A = \begin{array}{cccc} 0 & 1 & 2 & 3 \\ \hline 4 & 2 & 3 & 1 \end{array}$$

we need a way to check if ans is present in A.

ans ⟶ [1 to N+1]

If A[i] < 0 ⟶ i+1 value is present in A

$$A = \begin{array}{cccc} 0 & 1 & 2 & 3 \\ \hline -4 & -2 & -3 & -1 \end{array}$$

$A[i] \longrightarrow A[i] - 1$ as index

$$\begin{array}{lcccc} & 0 & 1 & 2 & 3 \\ A = & [-4 & 1 & -6 & -3] \\ & \downarrow & \downarrow & \downarrow & \downarrow \\ \text{index} & 3 & 0 & \cancel{5} & 2 \end{array}$$

ignore

$$\begin{array}{lcccc} & 0 & 1 & 2 & 3 \\ A = & [-4 & 1 & -6 & -3] \end{array} \longrightarrow$$

$A[i] < 0 \longrightarrow i+1$ is present

if $(A[i] > 0)$ return $i+1$

Handle negative or 0

$$[-1 \; -2 \; 1 \; 0] \\ \downarrow \\ 5 \quad 5 \quad 1 \quad 5$$

replace $A[i]$ to $N+2 \; \{+\infty\}$ to handle -ve

Pseudocode

```
    // handle  0 & -ve
    for  i ⟶ 0 to N-1 {
         if ( A[i] <=0 )   A[i] = ∞ or N+2
    }

for  i ⟶ 0 to N-1 {
         idx = abs(A[i]) - 1
         if (0 <= idx && idx < N) {
              // only if A[idx] is positive make it
                                              -ve
              if (A[idx] > 0) {          }
                   A[idx] *= -1          }  handle
              }                          }  duplicates.
         }
}

for  i ⟶ 0 to N-1 {
         if ( A[i] > 0 ) {       }
              return i+1         }  i+1 was missing.
         }                       }
}

return N+1
```

TC: O(N)

SC: O(1)

Dry Run

	0	1	2	3	4	5	6	7
A	-4	~~-1~~ -∞	-3	-2	1	1	~~-5~~ ∞	~~0~~ ∞
	↓	✗	↓	↓	↓	↓	✗	✗
idx	3		2	1	0	0		

ans = 5

$\textcircled{4}$

	0	1	2	3	4	5	6	7
A =	-1	-2	-2	~~-4~~ 10	8	~~-4~~ 10	3	-3
	↓	↓	↓	✗	↓	✗	↓	↓
idx	0	1	1		7		2	2

A[3] > 0 ⟶ 3+1 is ans.

$\underline{\underline{4}}$

	0	1	2	3
A =	-1	-1	2	2
	↓	↓	↓	↓
index	0	0	1	1

ans = 3