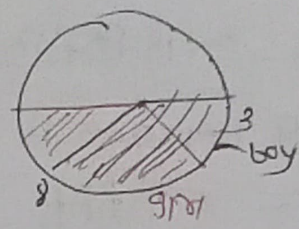
Day-19

Interval Pattern Theory

free time → Both free

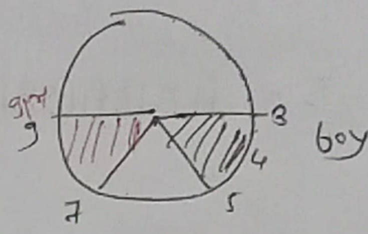
1st Day [3, 8] [4, 8] [4, 9]

boy girl

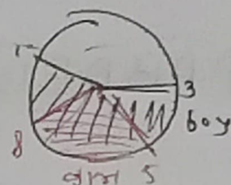


2nd Day [3, 5] No meet (7, 9) →

boy girl



3rd Day [3, 10) [5, 8] [5, 8]



Interval pattern

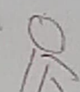
many real world pattern/problems involves ranges.
(Time ranges, number ranges, memory ranges, geometric ranges).

— (1, 5)
(3, 6
(1, 9)

here I.P. লাগবে

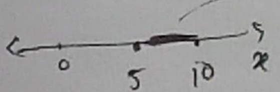
I/P [3, 8] [4, 9] [12, 19], [7, 20) (17, 20)

Interval 5 end



An Interval is written as

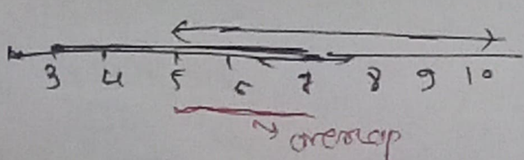
$\boxed{\text{start, end}}$

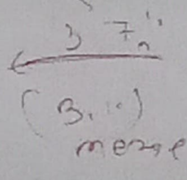


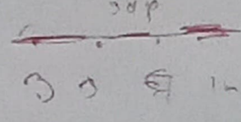
points betn [5, 10]

These pattern helps us understand how these range overlap, merge, intersect or leave gap

E.g [5, 10]
[3, 17]

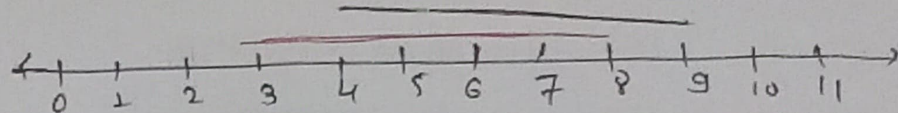






① [3, 8] [4 9]

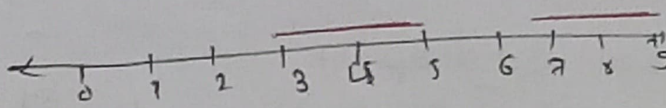
boy girl



(4, 8)
overlapping Interval

② [3, 5) (7 9)

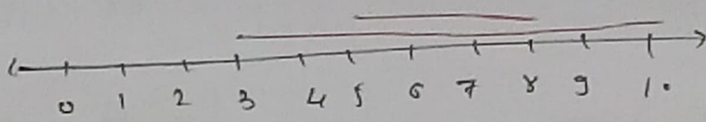
B Girl



- NON-overlapping

③ [3, 10] [5, 8]

boy Girl



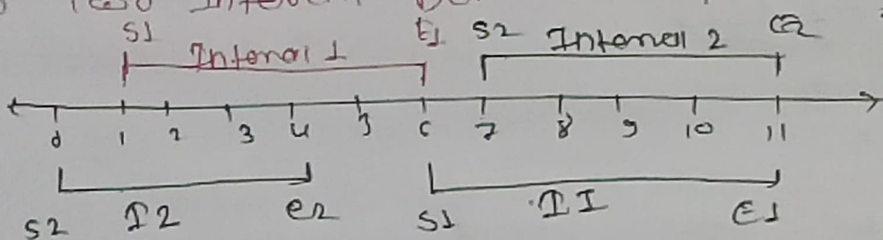
Contained Interval

When do Two Interval overlap

I1 = [S1, E2)

I2 = [S2, E2)

When do Two Intervals Don't overlap → (S2 > E1)



(S1 > e2)

⟨S2 > e1 || S1 > e2⟩ → NO overlapping

1) When 2nd event starts, After 1st event is compleated

S2 > e1

② When first event starts after the second event is compleated

~~S2~~ S1 > e2

$\boxed{S2 > e1 \;||\; S1 > e2}$ → Two Intervals Don't overlap

!⟨ (S2 > e1) || (S1 > e2) ⟩ → Overlap कर रहे हैं

When do Two Interval overlap

- Two Intervals don't overlap IF:

  $e1 < s2 \; || \; e2 < s1$

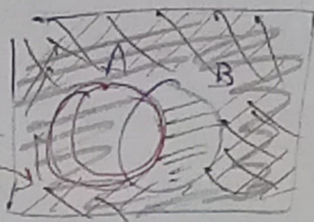
- Two Interval overlap IF:

  $!(e1 < s2 \; || \; e2 < s1)$ || $(e1 >= s2 \; \& \; e2 >= s1)$

① De morgan's Law

① $(A || B)^{\wedge} = A^{\wedge} \& B^{\wedge}$

② $(A \& B)^{\wedge} = A^{\wedge} || B^{\wedge}$



$A^{\wedge} =$

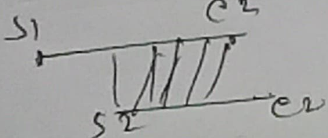
$B^{\wedge} =$

$(\underbrace{e1 > s2}_{A} \; || \; \underbrace{e2 < s1}_{A})^{\wedge}$

$(A || B)^{\wedge}$

$A^{\wedge} \& B^{\wedge}$

$(e1 < s2)^{\wedge} \& (e2 < s1)^{\wedge}$

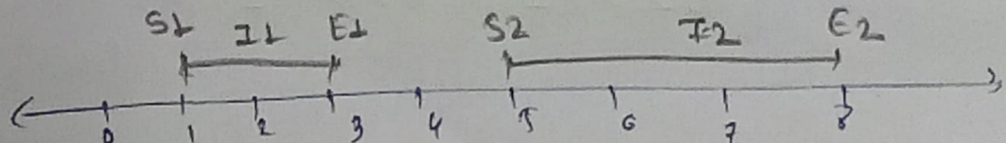


→ overlap

Types of Interval Relationship

→ NON-overlapping

$[1.3)$ $[5\,8)$



$s2 > E1$ ||

~~E2 > S1~~

$S1 > e2$

gap exists

usefull for finding free gap

min $(e1, e2) = 3$

max $(s1 s2) = 5$

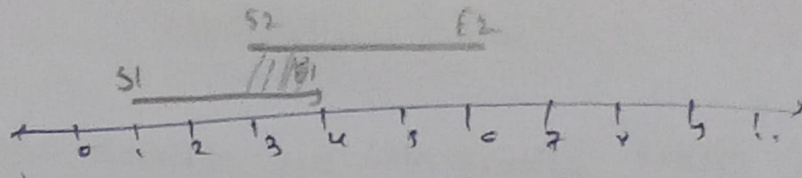
$\boxed{5 - 3 = 2 \text{ Difference}}$

min $(e1, e2)$

max $(s1, s2)$

Overlapping

$e1 >= s2$
& $e2 >= s1$



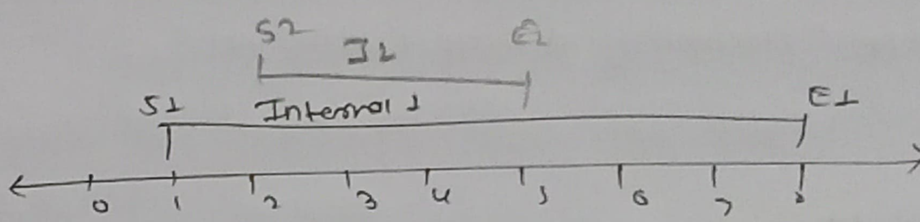
E.g [1 4) [3,6)

$\langle \max(s1, s2), \min(e1, e2) \rangle$
(3 , 4) = 1

1. They share time
2. Useful for merging or counting conflicts

3 Contained:

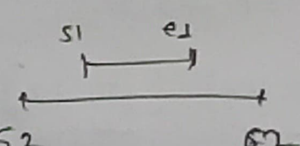
[1, 8] [2 5]



Condition 1 $(S2 >= S1$ & $E2 <= E1)$

|| → ues conv or

Con' $(S1 >= S2$ & $e2 >= e1)$



Smaller Interval lies fully inside larger one

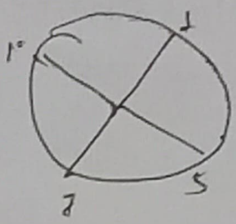
First step in Interval Problems

① Sort.

- Allways sort Intervals by their start time

$[5, -7], [1 - 10] [11 - 13] \rightarrow [1, 10] [5, 7] [11, -13]$

2 1 3



→ complicate, not without sort

so sort it

why sort :-

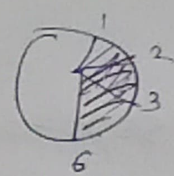
① Intervals are processed in timeline order
② Adjacent intervals can be compared Directly
③ Easier merging & gap detection

Problems :-

merge overlapping Intervals :

$[1\ 3]\ [2\ 6]\ [8\ 10]\ [15\ 18]$

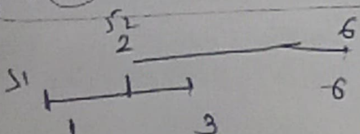
$[1\ 3]$
$[2\ 6]$



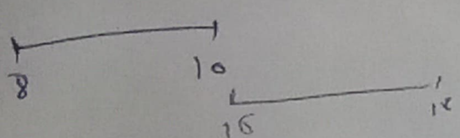
<1, 6> Ans

Ans

$[1\ 3] \rightarrow [1, 6], [8\ 10]\ [15\ 18]$



min (s1, s2) = 1
max (e1, e2) = 6



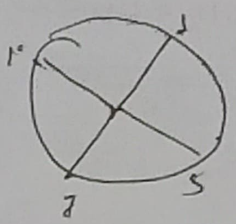
→ No merge

First step in Interval Problems

① sort.

- Allways sort Intervals by their start time

$[5, 7]$. $[1, 10]$ $[11, 13]$ → $[1, 10]$ $[5, 7]$ $[11, 13]$



→ complicate not without sort so sort it

why sort :-

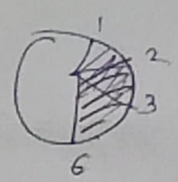
① Intervals are processed in timeline order
② Adjacent intervals can be compared Directly
③ Easier merging & gap detection.

Problems :-

merge overlapping Intervals :

$[1\ 3]$ $[2\ 6]$ $[8\ 10]$ $[15\ 18]$

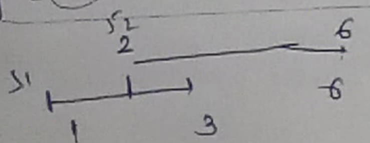
$[1\ 3)$
$[2\ 6)$



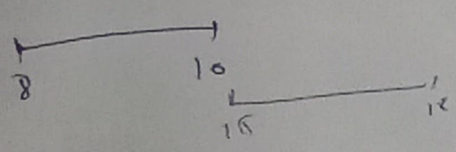
< 1, 6 > Ans

Ans

$[1\ 3]$ → $[1, 6]$. $[8\ 10]$ $[15\ 18]$



$\min(s1, s2) = 1$
$\max(e1, e2) = 6$



→ No merge

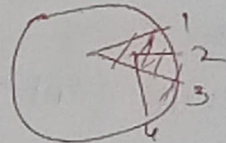
```
Sort (Intervals)                          // Add first interval
result add (Intervals [0])
for (i=1 ; i< intervals.length ; i++)
{
   last = res [res.length -1]   → Remove last
   Interval = Intervals [i]
                                     Ex  [1 3) [2 6) (8 10] [15-18)
                                     Ans [1 3]

  IF interval [0] <= last [1] ;
   last [0] = min ( interval [0], last [0])
    last [1] = max ( interval [1], last [1])

  else
  res.add ( interval)
```

Problems②

① meeting rooms (check the conflicts)

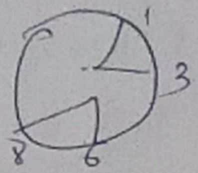


[1 - 3) [2 4)

can person attend all meetings ?

```
Sort (Intervals)
for (int i=1 ; i< intervals.length ;i++)
   Interval - Interval [i-1]:
    curr = Intervals [i)
   IF [curr [0] <= interval [1]
   return False;
  else
   return True
```

③ Find free ~~gap~~ Time (gaps)



[1 3] [6 8]

Given busy interval, find available time slots

[1 3] [6, 8]

```
sort (intervals)
res []
For int i = 1 ; i < intervals.length ; i++
    interval = intervals (i-1)
    curr = Interval [i]

  IF [curr [0] >= ~~last~~ interval [1] ;
     res.add (interval [1], curr [0])
```

curr interval ka start time

interval ka end time

return res

Kab

① Input contains Intervals (time or numeric ranges)

② You need to analyze overlaps, gaps, order, capacity, merging. or coverage

③ Sorting by start time solves the biggest part of the problem.

Intuition behind sorting by start time

① Chaos ..

Eg [5 7] [1 4] [12 18] [3 ~~/~~5] [9 11] [2/6]

i j j j

[3, 7] [2, 7]

Bruete force

② Event comparison :

merging : new event . start <= prev _ event . end

② Why Not sort By end

Hides future overlaps

~~[5 8] [1 6] [2 4]~~
[2 3) [4 5) [1 10)

[2 3) [4 5] → Hide

Day 20

problems :-

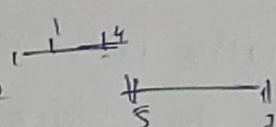
① merge Intervals (Brute force)

0	1	2	3	4	5
[5, 7]	[1 4]	[12 15]	[3/5]	[9, 11]	[2 6]
i	j	j	j		

→ pair wise comparison

[5 7] [3 5] $\xrightarrow{\text{merge}}$ [3, 7]

[5 7] , [1 4] → No merge bc 2



j को आगे

0	1	2	3	4
i				
[5, 7 / 3, 5]	[1 4]	[12 15]	[9 11]	[2, 6]
i (3 7)			j	j

No need to j++ bc 2 after new index formed j at next already

[2, 6) [3, 7) → [2, 7]

0	1	2	3
[2, 7]	[1 4]	[12 15]	[9 11]
	i	j	j
		i	j

i++ bc 2 we can compare all pairs

result = [2, 7] (1, 4] [12, 15] [9, 11]

pending

– again merge

① - Iteration pair comparison

0 1 2 3
[2 7] [1 4] [12 15] [3, 11]
i j

⇒ [1, 7] [12, 15] [3, 11]
i ~~j~~
~~j~~
i (Done)

result (1 7) (12 - 15) (3, 11)

Refining the Algorithm

① pair wise comparision
② we get answer after 1st Iteration
③ After ~~Ao~~ some iteration 1st interval intereals 7 ∧
④ So it can also merge 2nd interval
⑤ we have to do pair-wise comparision after merging

① pair-wise comparision
↳ merge
↳ run ① again

merge Not happen
↳ merged all interval