celebrity problem:

n- persons

n= 6

grid:

	0	1	2	3	4	5
0	0	1	1	0	0	1
1	0	0	1	1	0	1
* $\boxed{2}$	0	0	0	0	0	0
3	1	0	1	0	1	0
4	1	0	1	0	0	1
5	0	1	1	1	1	0

0 1 2 3 4 5

✓ grid[i][j] == 1

ith person know all About jth person.

celebrity → A person known by Everybody but doesn't know anyone.

is there always a celebrity in party?

→ No.

Assume → ① Every person is celebrity →

② Elimitate non-celebrity person.

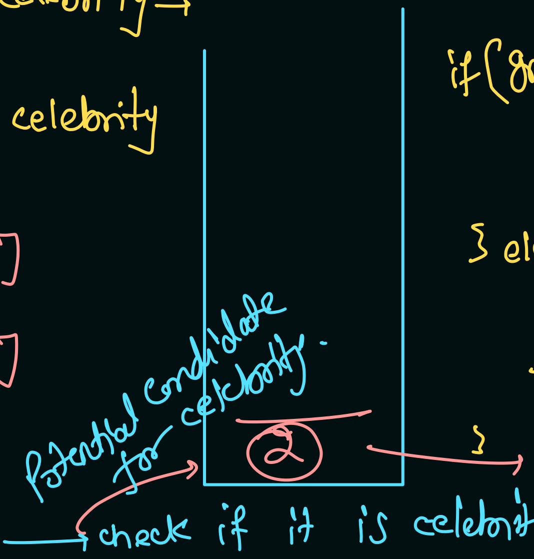
grid[5][4] =

grid[4][3] =

grid[4][2]

grid[2][1]

grid[2][0]



Potential candidate for celebrity.
2
check if it is celebrity or not-

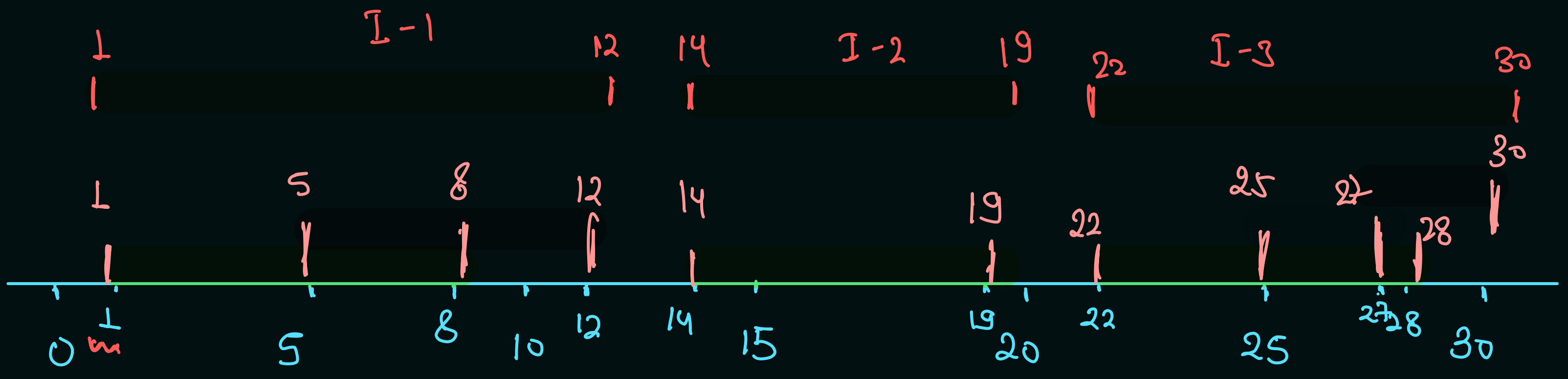
```
if(gr[i][j] == 1) {
    → i is not celebrity
    → may be j is celebrity] → push 4 again
} else {
    → j is not celebrity]
    → may be i is celebrity] push in Stack
}
```

If single person remain in stack, then stop.

Merge overlapping Interval:

→ Intervals are initially arranged in Random order.

	St.	Ending	
1.	22	28	✓
2.	1	8	✓
3.	25	27	✓
4.	14	19	✓
5.	27	30	✓
6.	5	12	✓

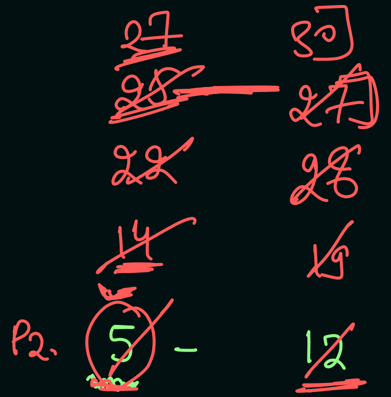


→

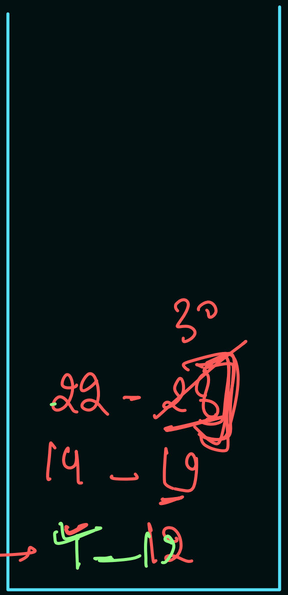
St.	End
✓1	8
→ ✓5	12
✓14	19
✓22	28
✓25	27
✓27	30

Result ⇒ 1 - 12
⇒ 14 - 19
⇒ 22 - 30

① Sort on the basis of starting time,



P2.Start < P1.End] — Merging is for sure,
if P2.End > P1.End] — P1.start - P2.End,
otherwise P1.start - P1.end



[22 - 30
14 - 19
1 - 12]

Return in Reverse order

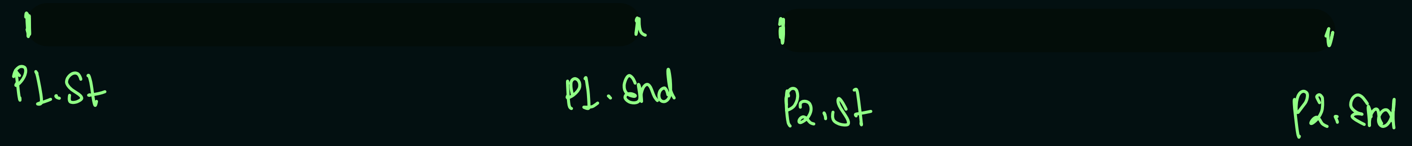
Case-I

Pair P1   Pair P2.

P2. Start > P1. Start] → sort on the basis of Starting pair

↳ P2. Start > P1. End.

P1. End < P2. St

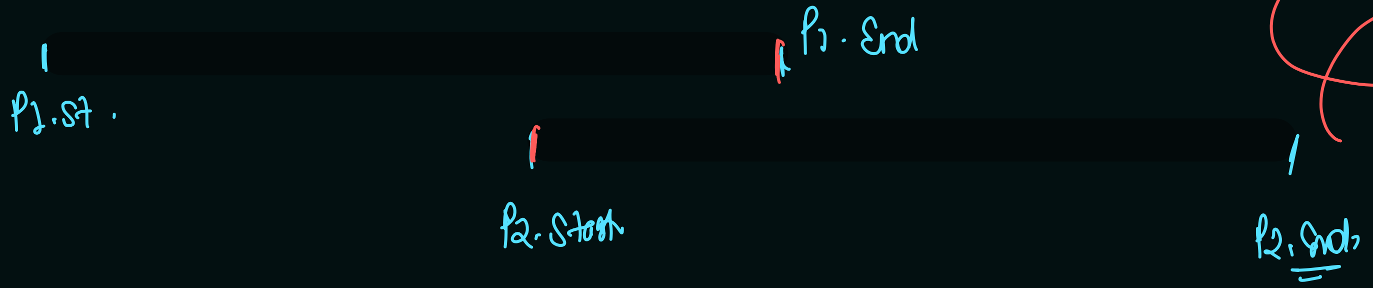


↳ there is no merging of point

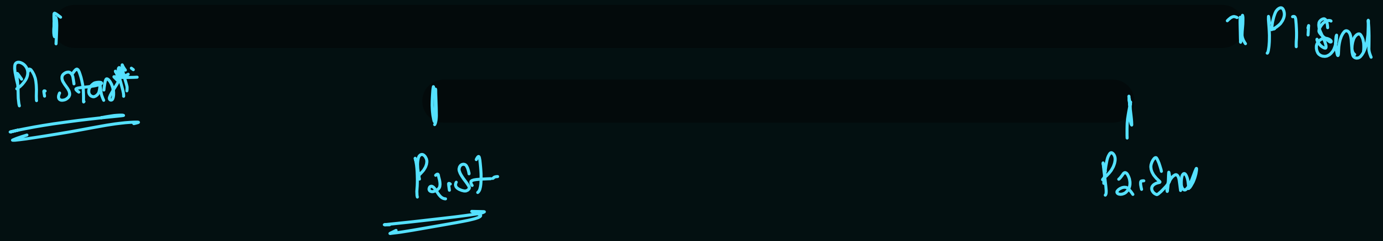
Case-II ① P2. Start < P1. End] => combine. → P1. End < P2. End

Merging → P1. Start → P2. End

↳ check Ending of P2 as well



② P2. Start < P1. End ←→ P1. End > P2. End



↳ Merging → P1. Start -- P1. End

comparison between object] → ~~< >~~ , compareTo ||-

array of wrapper class] → object comparison] → sort

Bubblsort

7 10

~~10~~ ~~7~~ 2 4 9 [max

↑ ↑ ↑

val1 val2

```
if (val1 > val2)
    swap (val1, val2);
```

Smallest Number following pattern:

Smallest

d → decreasing

i → increasing.

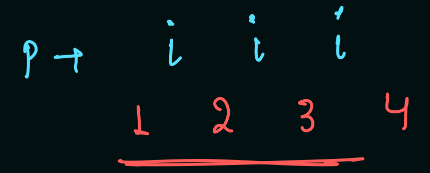
number > 1

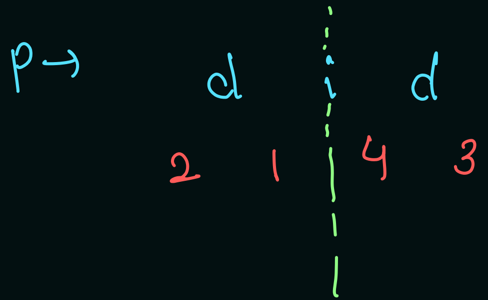
p → d d d → pattern. len.

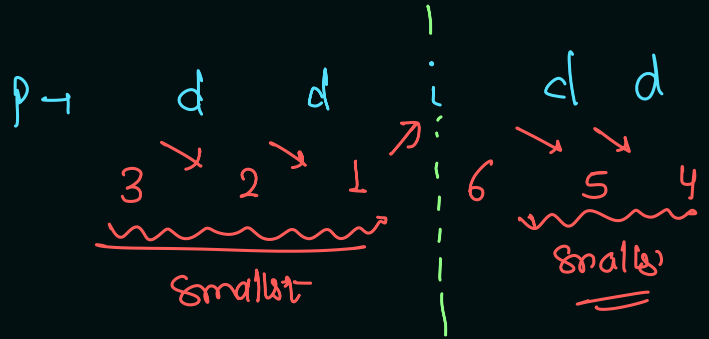
4 3 2 1

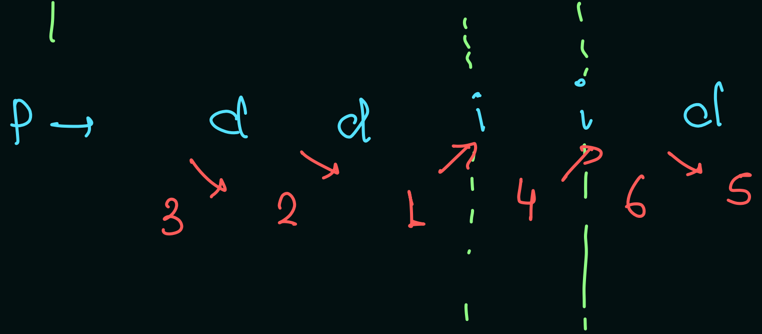
num = pattern.length + 1

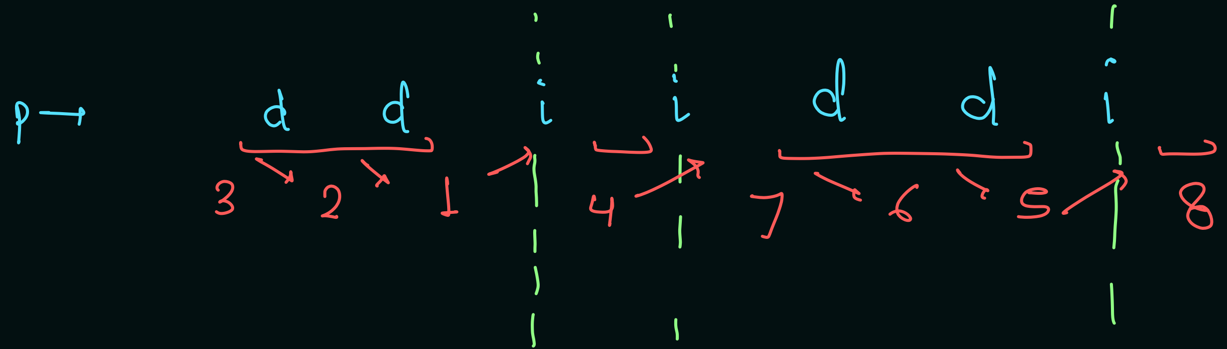
Break point → Increasing.











pattern → d d i i d d i

$3 \searrow 2 \searrow 1 \nearrow 4 \nearrow 7 \searrow 6 \searrow 5 \nearrow 8$

Encount.

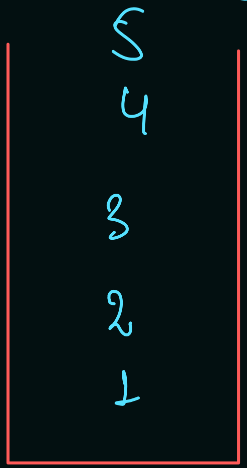
d → add num in stack & increment num,

i → add num in stack, & increment num & print stack.

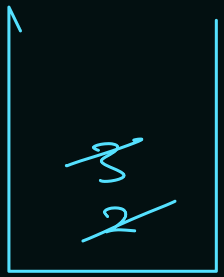
nums = ~~1~~ ~~2~~ ~~3~~ ~~4~~ ~~5~~ ~~6~~ ~~7~~ 8

pattern → d d d d

5 4 3 2 1



num: ~~1~~ ~~2~~ ~~3~~ ~~4~~ $\bigcirc\!\!\!\!\!\;5$



i i i

1 2 3 $\bigcirc\!\!\!\!\!\;4$

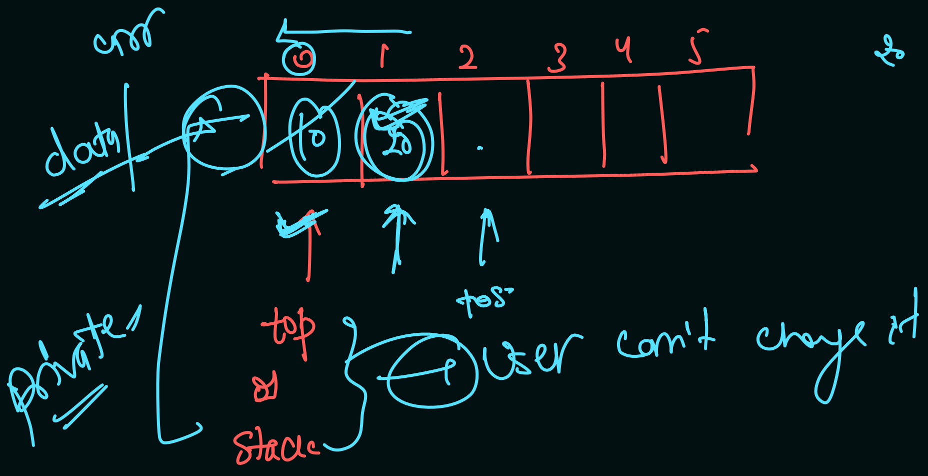
num: ~~1~~ ~~2~~ ~~3~~ 4

Stack - Implementation:

max size.

Stack → object creation → capacity of Stack

capacity → 5.



& public

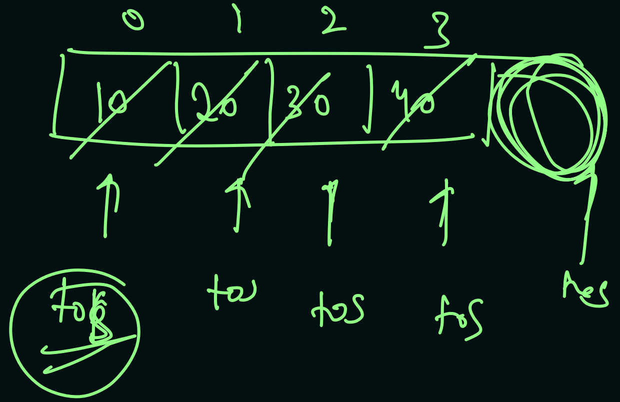
push → add data at 'tos' index
tos ++

pop → val = arr[tos-1];
tos--;

top → val = arr[tos];
return val.

size → return tos;

display.



push	- 10	pop
"	- 20	pop
	- 30	pop
	- 40	pop
	- 50	pop

Self work →

- ① Normal Stack
- ② Dynamic Stack
- ③ Minimum Stack I
- ④ Adapters

Class work

- Queue → Normal Queue, Dynamic Queue.
- Min Stack — constant space
- Evaluation → Infix, prefix, postfix

10:00 - 1:00