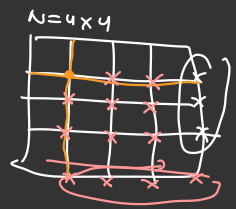
Arrays - 3

Arrays, Sorting, 2D Arrays → Mixed Problems

Merge Interval

Doubt



$O(N^2)$ cut,

$(N-1)^2 = O(N^2)$

Q1.

Fun Challenge → Given a list of numbers containing every no from 1 to N except one no which is missing (No Duplicates, Unordered) Find out the missing number.

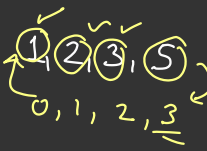
N = 5

list = {5, 2, 1, 3}

N-1 numbers

Option-1 →

1-based indx. 1 2 3 (4) ⇒ missing



2 more than idx

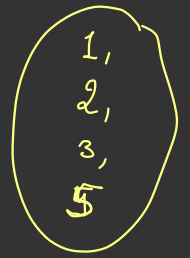
idx+1

$O(N \log N + N)$

$= O(N \log N)$

Sorting & comparing idx with value.

Option-2



← 1 ✓
← 2
← 3
← 4 No

```
for( i=1 ---- N){
    hs.find(i) → Not Present → i
}
```

O(N) time
O(N) space

Option-3

$1+2+3+4+\cdots\cdots+N = \frac{N(N+1)}{2}$ ← total expected sum

$1+2+3+?+\cdots\cdots+N =$ Sum

$1+2+3+4+5$

$= \frac{5 \cdot 6}{2} = 15$

$5, 3, 1, 2 = 11$

$15 - 11 = 4$

- missing No = $\frac{N(N+1)}{2}$ − Sum of Array

O(N) time
O(1) Space

Q2 Given a list of N Numbers which is unordered, find out the longest chain consecutive elements we can build

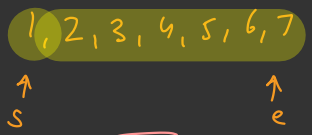
For ex 

1 → 2 → 3 → 4 → 5 → 6 → 7 = (7)
9 → 10 → 11 → 12 → 13 = (5)
20 → 21 = (2)
} 7

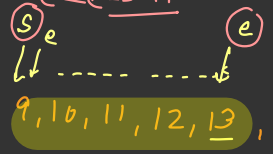
Think
9.30

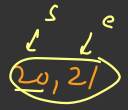
Sol- 1

Sort



$l = e - s + 1$





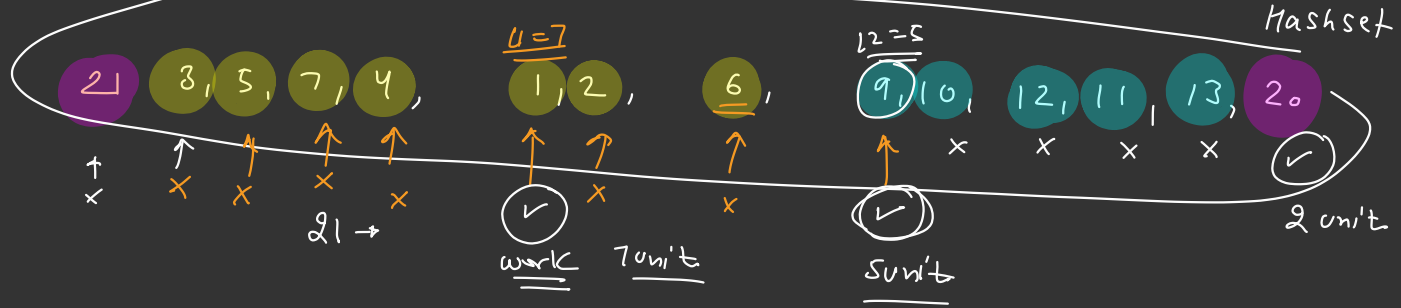
$l2 = 5$

$l3 = 2$

O(NLog N) time

max len = 0 7 ~~5~~ ~~2~~ $l1 = 7$

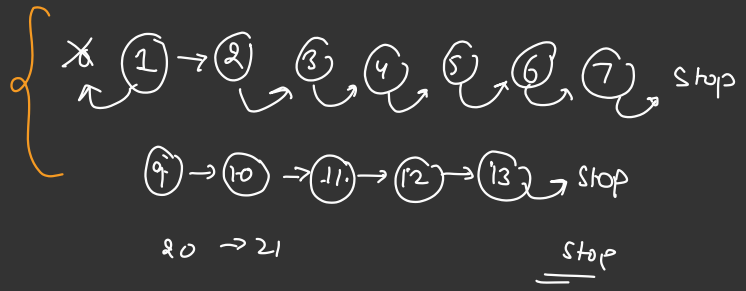
Sol-2 Improve the time



① Added elements in hashset

② For arr(i), if arr(i)-1 is present in hashset then this arr(i) will NOT start longest chain

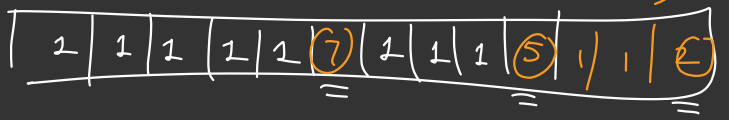
② → 3 → 4 → 5 → 6 → 7



O(N)

$= O(N+N) = O(2N)$

$= O(N)$



7 + 5 + 2

$\simeq O(N)$

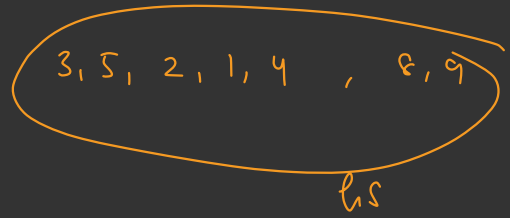
Optimise Time ↑ and optimise Space ↑

↳ Time ↑ and Space ↓

↳ Space ↑, Time ↓ (dgrade)

Code

```
HashSet hs;
for(x : arr){
    hs.add(x);
}
```



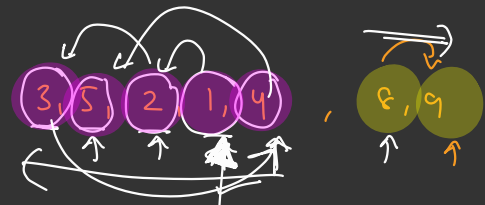
Code clr?

O(N) time

O(N) Space

Len = 0, Largest = 0

```
for( x : arr) {
    if( hs.find(x-1) == false) {
        Len = 1
        no = x+1
        while (hs.find(no) == true) { Len++, no = no+1 };
        Largest = max (Largest, Len),
    }
}
```

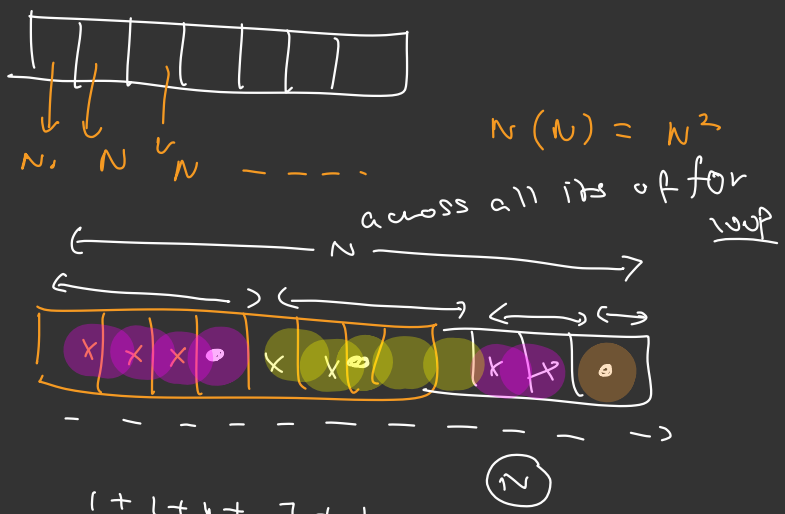


x = 1
no = 2

Largest = max (0, 5)

(5, 2)

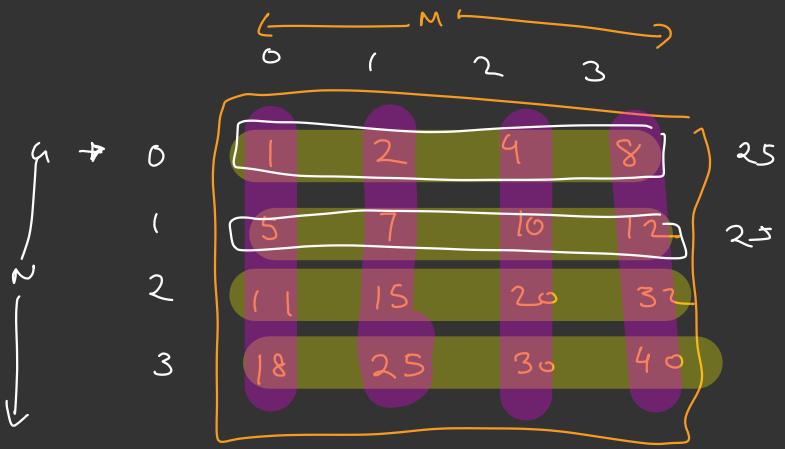
(a)



$$1+1+1+2+3+4+\cdots+N+N+1$$

$$= 2N$$

Given a 2D array which is sorted along the rows and columns and you are given an element X to search in the array.



T = 25 Search i, j

(1) Brute Force

↳ iterate over 2D array (linear search)

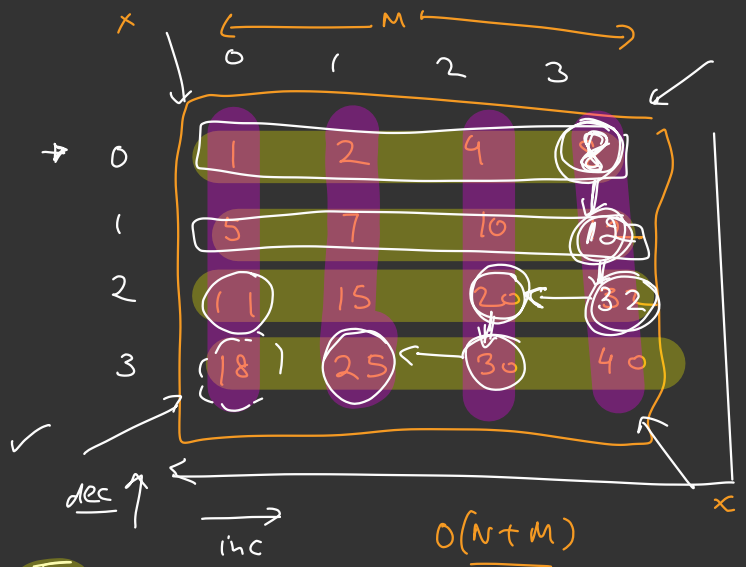
$O(N^2)$

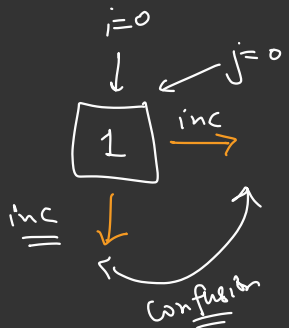
(2) Binary search = $O(N \log M)$

across every Row $\log M$

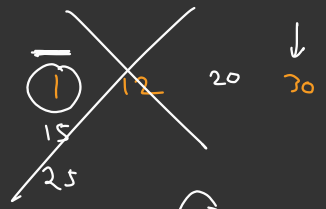
(3)

search in the array.

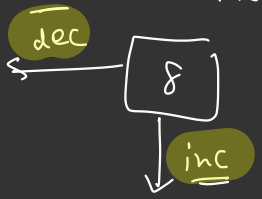




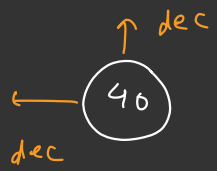
T=25



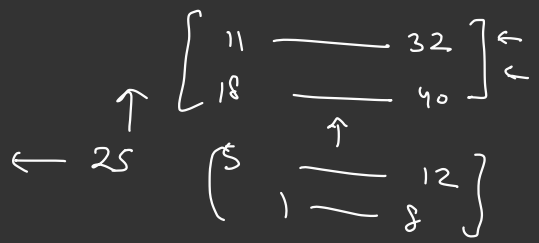
32
40

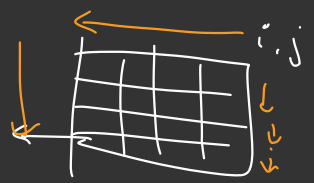


25



25





$i = 0$

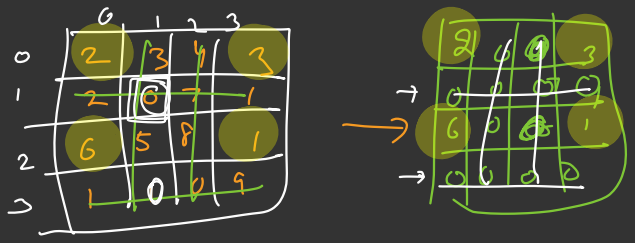
$j = M-1$

```
while ( i<=N-1 && j>=0 ) {
    if a(i)(j) > T.
        j--
    else if a(i)(j) < T.
        i++
    else
        return (i,j)
}
```

$O(N+M)$

Staircase Search

Q Given a 2D array contain no's, some elements are 0 if you see a zero, make all elements in that row & col as 0.



15 min break :)

10.30

Wrong Algorithm → Iterate over arr(), if I get arr(i)(j) → I make entire Row & col = 0

2, 0, 0, 0
0, 0, 0, 0
0, 0, 0
0, 0, 0
0, 0, 0

Creating artificial zeroes

$O(N^2)$ time

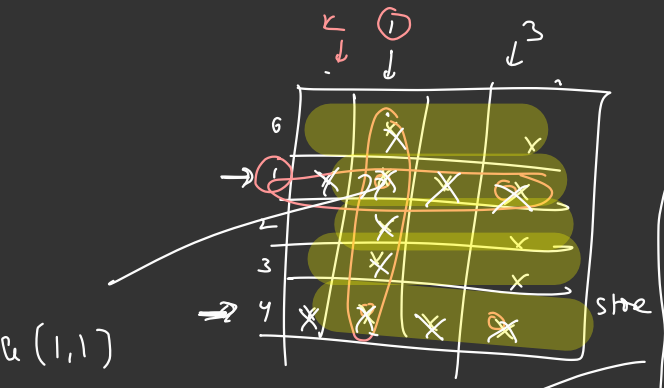
Soln → (1) Traverse the 2D array & maintain two

rows →
cols → (1), 1 ---)

map Rows
1 → ✓
3 → ✓

map Cols
1 → ✓
2 → ✓

② Read maps update Rows & cols




```
HashSet RHS;
HashSet CHS;
for(i)
  for(j){
    if(a[i][j] == = 0)
      RHS.add(i)
      CHS.add(j)
  }
```

O(N·M)

map HashSet Rows = (1, 4)
< HashSet Cols = (1, 3)

] update

O(N+M)

```
for( Row : RHS){
    for(k = 0 — M-1){
        arr[Row][k] = 0
    }
}
```

O(M·N)

```
for( col : CHS){
    for(k = 0 — N-1){
        arr[k][col] = 0
    }
}
```

O(M·N)

Merge Intervals

Input

[2,6]

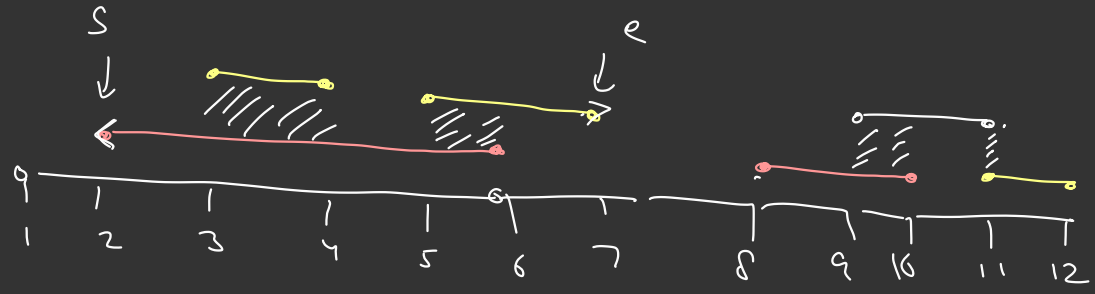
[8,10]

[3,4]

[5,7]

[11-12]

[9,11]



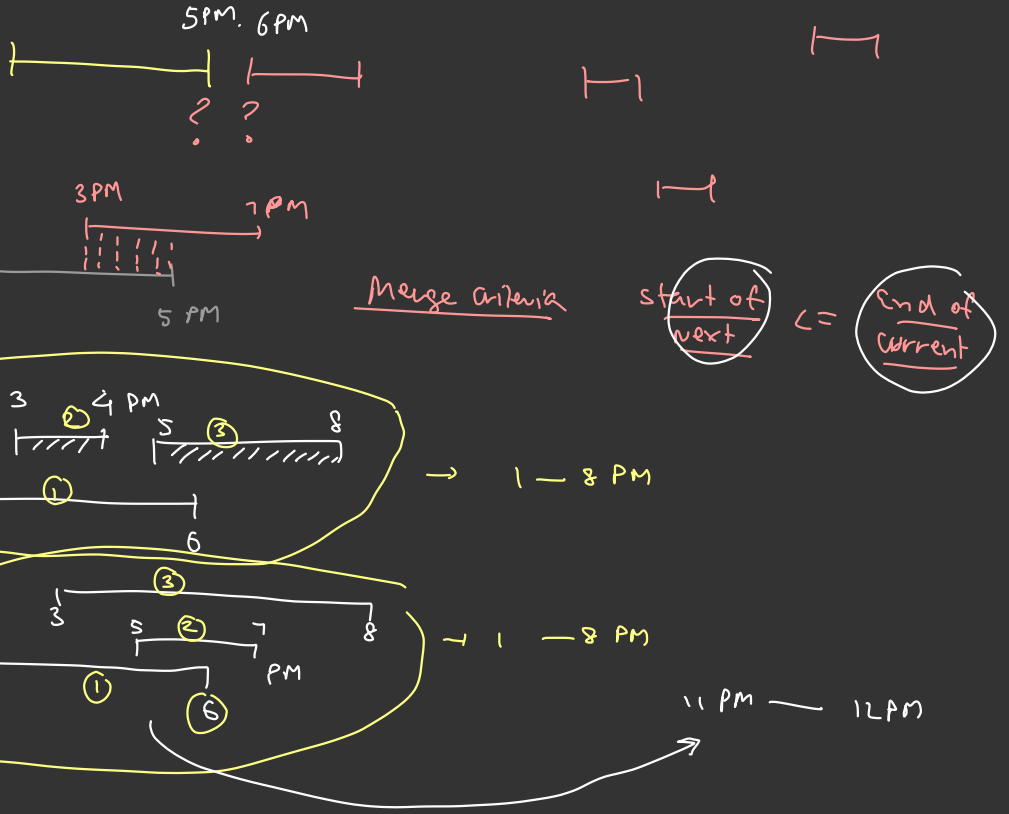
output

[2-7] , [8,12]

~~[8-10]~~

~~[11-12]~~

Ideas


5PM. 6PM
? ?
3PM
7PM
5 PM
Merge criteria
start of next
<=
End of current
3
4 PM
5
8
1
6
→ 1 — 8 PM
3
5
2
7
8
PM
1
1
6
1 — 8 PM
11 PM — 12PM

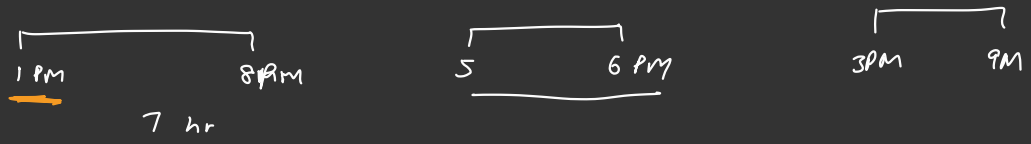
Closest interval to the current

↳ sort all the intervals [start, end]

↳ start

↳ ~~end~~

↳ ~~duration~~

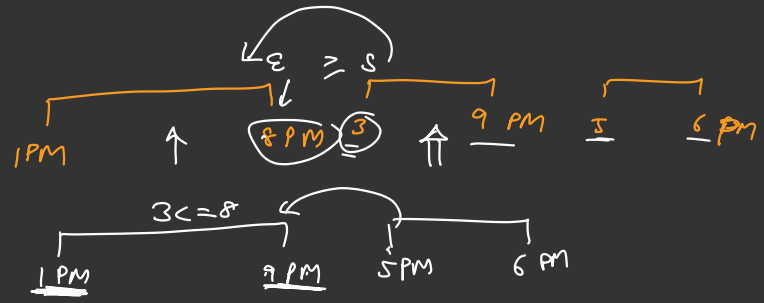


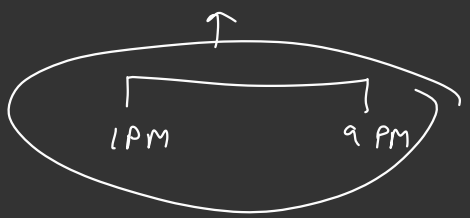
idea

next

merge with prev

start a new




1PM
9 PM

Input

start ↓ end ↓

[2, 6]
[8, 10]
[3, 4]
[5, 7]
[11, 12]
[9, 11]

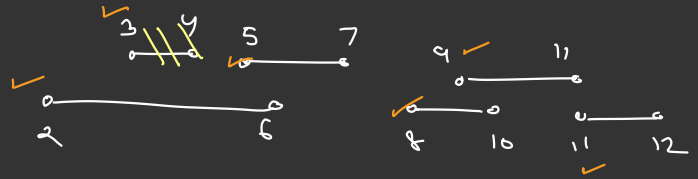
Soln ① Step Sort acc to start (interval)

↳ ① [2, 6]
② [3, 4] → (2, 6)
[5, 7]
(4, 7) [8, 10]
[9, 11]
[11, 12]

```
compare ( Interval I1,
          Interval I2 ) {
    return I1.start < I2.start
}
```

3

Sort(____ , compare)



(2, 6), (3, 4)

I1 ↓ I2

(2, 6), (5, 7)

Merge if I2.S ≤ I1.e

↳ I.S = I1.S

I (2,7), (8,10), (9,11)

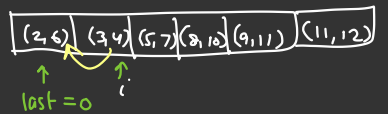
↳ I.e = max(I1.e, I2.e)

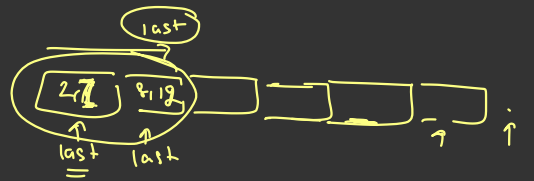
⇒ (2,7) (8,11) (11,12)

(2,7) (8,12) output

NLogN

① Sort acc I.start





② for(i=1 ——— N-1) { i++

N

if (arr[i].start ≤ arr[last].e) {

arr[last].end = max(arr[last].end, arr[i].end);

}

else {

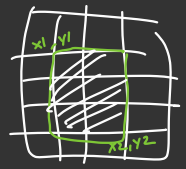
last++,

arr[last] = arr[i]

}

}

② -3, -8, 1, 2, 3, 4, 7, 8 <= next class




```
for(x1 ____ )
  for(y1 ____ )
      for (x2 = x1 ____ )
        for(y2 =y1 ____ ){
                (x1,y1, x2,y2)
                for(i') for(j')
```