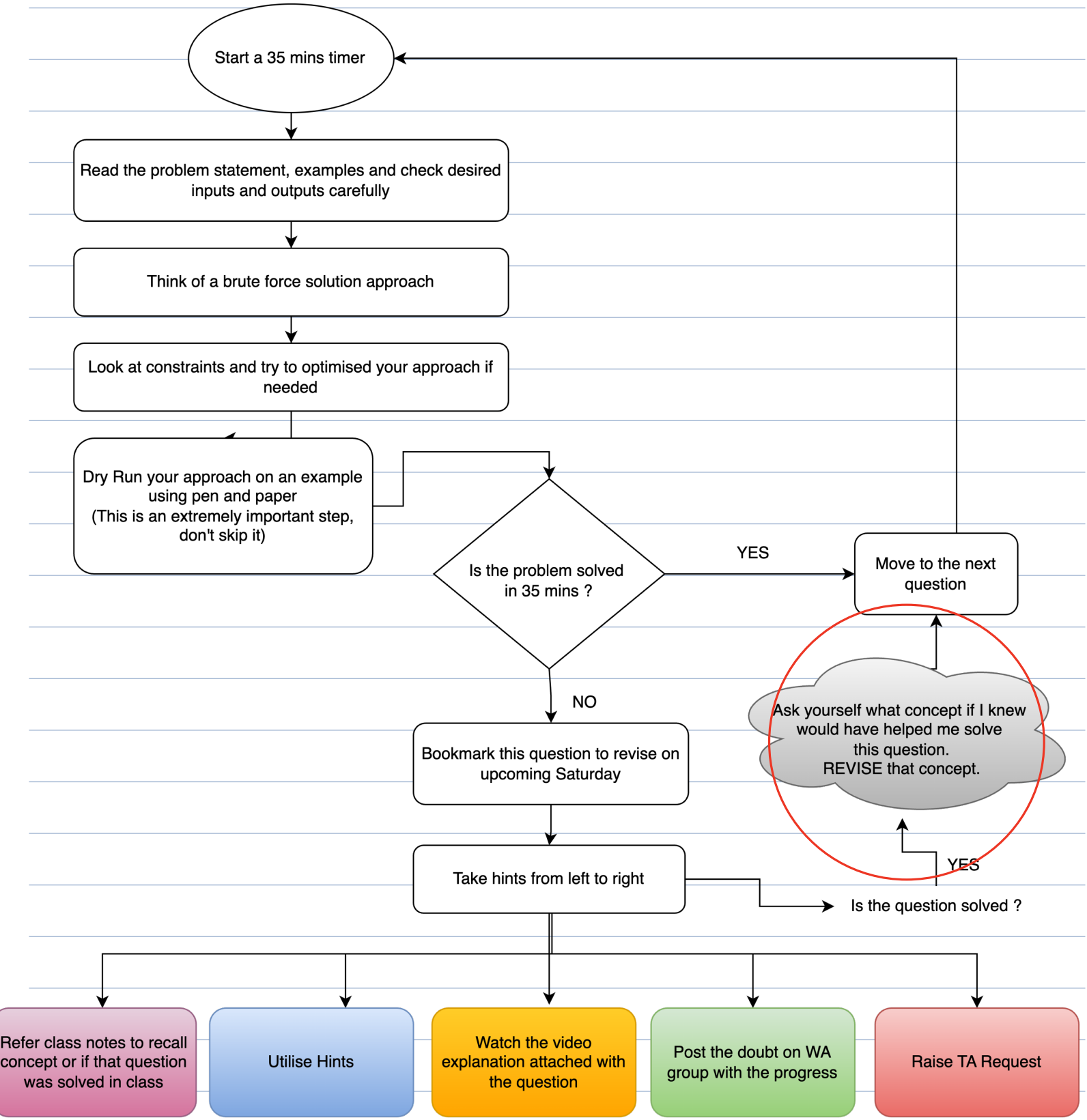
1. Find Maximum Subarray Sum
2. Find in Row-wise and Column-wise Sorted Matrix
3. Merge Sorted Overlapping Intervals
4. Transpose of a Square Matrix
5. Rotate a Matrix 90° Clockwise


Start a 35 mins timer
Read the problem statement, examples and check desired inputs and outputs carefully
Think of a brute force solution approach
Look at constraints and try to optimised your approach if needed
Dry Run your approach on an example using pen and paper
(This is an extremely important step, don't skip it)
Is the problem solved in 35 mins ?
YES
Move to the next question
NO
Bookmark this question to revise on upcoming Saturday
Take hints from left to right
Is the question solved ?
YES
Ask yourself what concept if I knew would have helped me solve this question.
REVISE that concept.
Refer class notes to recall concept or if that question was solved in class
Utilise Hints
Watch the video explanation attached with the question
Post the doubt on WA group with the progress
Raise TA Request

Crypto ⟶ Trading

	start	3	4	2	5	14
bitcoin	=	[1	-2	3	9	-4]
delta	day	0	1	2	3	4
	end	4	2	5	14	10

Max profit I can make if I can buy once and sell once

buy once → day start

sell once → day end

Buy at day 2

sell at day 3

Maximum subarray Sum

Given int[] A. Find max subarray sum of all subarrays

	0	1	2	3	4	5	6	
A =	-2	3	4	-1	5	-10	7	ans = 11

	0	1	2	3	4	5	6	
A =	-3	4	6	8	-10	2	7	ans = 18

Quiz

A = 4 5 2 1 6 ans = 18

→ 4+5+2+1+6

A = -4 -3 -6 -9 -2 ans = -2

Bruteforce :

Check sum of all subarrays ret max

#subarrays = $\frac{n * n+1}{2}$ $\approx O(N^2)$

TC: $O(N^3)$

Pseudocode

[1]
✓ 0s

```
def brute_force(A):
  N = len(A)
  mtotal = float("-inf")
  for L in range(N):
    for R in range(L, N):
      total = 0
      # Find the sum from index L till index R
      for i in range(L, R+1):
        total += A[i]

      print(A[L:R+1], total)
      mtotal = max(mtotal, total)

  return mtotal

print(brute_force([1,2,3]))
```

```
[1] 1
[1, 2] 3
[1, 2, 3] 6
[2] 2
[2, 3] 5
[3] 3
6
```

apply prefix sum

TC: $O(N^2)$

A = [10 -1 1 2 3]

⟶ 10 9 10 12 15

A = [10 -11 1 2 3 4 5]

⟶ 10 ~~-1~~ 0 1 3 6 10 15

Observations

case 1> 1 2 3 4 5 ans = 15

sum of all

case 2> −1 −2 −3 −4 −5

max of all

case 3> −ve −ve −ve +ve +ve +ve −ve −ve

longest positive subarray

case 4> +ve +ve +ve −ve −ve −ve

case 5 +ve −ve −ve −ve +ve +ve ....

	0	1	2	3	4	5	6	7	8
A =	−20	10	−20	−12	6	5	−3	8	2
total	~~−20~~ → 0	10	~~−10~~ → 0	~~−12~~ → 0	6	11	8	16	18
any −∞	−20	10	10	10	10	11	11	16	18

A =	−10	−3	−2	−7	−6
total	~~−10~~ → 0	~~−3~~ → 0	~~−2~~ → 0	~~−1~~ → 0	~~−6~~ → 0
any = −∞	−10	−3	−2	−2	−2

Pseudocode

Idea ⟶ keep track of total from L to R

if total < 0

reset the starting point

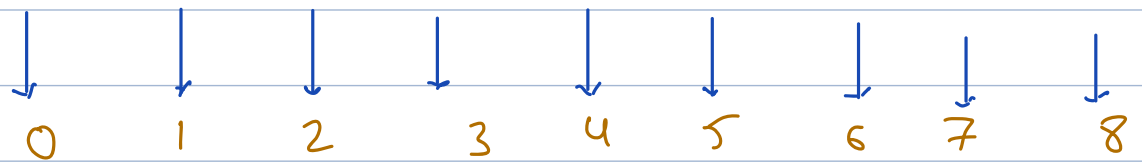
total = 0

```
def max_subarray_sum ( A ) :
    total = 0
    mtotal = -inf

    for val in A :
        total += val
        mtotal = max ( mtotal , total )
        if total < 0 :
            total = 0

    ret mtotal
```

kadane

TC : O(N)

SC : O(1)



A =	6	-3	9	-15	3	12	0	-24	3
total	6	3	12	~~-3~~ 0	3	15	15	~~-9~~ 0	3
ans = -∞	6	6	12	12	12	15	15	15	15

```
def max_subarray_sum ( A ) :
    total = 0
    mtotal = -inf

    for val in A :
        total += val
        mtotal = max ( mtotal , total )
        if total < 0 :
            total = 0

    ret mtotal
```

T	~~-2~~	3	7	6	11	1	8
MT	-2	3	7	7	11	11	11

Q> Given a row & column sorted matrix.
Return true if an element K exists.

	0	1	2	3	4	5
0	-1	2	4	5	9	11
1	1	4	7	8	10	14
2	3	7	9	10	12	18
3	6	10	12	14	16	20
4	11	15	19	21	24	27
5	18	24	29	32	34	42

K = 15 ans = T

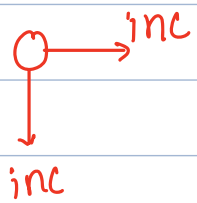
K = 43 ans = F

K = 24 ans = T

Brute force : check each element == K O(N*M)
O(R*C)

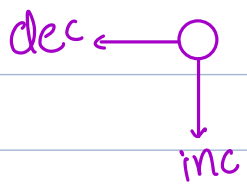
	0	1	2	3	4	5
0	-1	2	4	5	9	11
1	1	4	7	8	10	14
2	3	7	9	10	12	18
3	6	10	12	14	16	20
4	11	15	19	21	24	27
5	18	24	29	32	34	42

K = 15



	0	1	2	3	4	5
0	-1	2	4	5	9	11
1	1	4	7	8	10	14
2	3	7	9	10	12	18
3	6	10	12	14	16	20
4	11	15	19	21	24	27
5	18	24	29	32	34	42

K = 15



Idea >

	0	1	2	3	4	5
0	-1	2	4	5	9	11
1	1	4	7	8	10	14
2	3	7	9	10	12	18
3	6	10	12	14	16	20
4	11	15	19	21	24	27
5	18	24	29	32	34	42

k = 24

goleft ← smaller A[r][c]

higher val → go down

(r, c-1) ← (r, c) → (r+1, c)

-5	-2	1	13
-4	0	3	14
-3	2	6	18

Say we are at 1 and want to find 0, where should we move ?

19 users have participated

✓	A	left	89%
	B	bottom	0%
	C	let's move in both the directions	11%
	D	let's move everywhere	0%

	0	1	2	3	4	5
0	-1	2	4	5	9	11
1	1	4	7	8	10	14
2	3	7	9	10	12	18
3	6	10	12	14	16	20
4	11	15	19	21	24	27
5	18	24	29	32	34	42

$k = 32$

	0	1	2	3	4	5
0	-1	2	4	5	9	11
1	1	4	7	8	10	14
2	3	7	9	10	12	18
3	6	10	12	14	16	20
4	11	15	19	21	24	27
5	18	24	29	32	34	42

$k = 7$

$k = 0$

-5	-2	1	13
-4	0	3	14
-3	2	6	18

def find_row_col_matrix (A , k) :

R row

C cols

```
# Problem 2: Find in rowwise and colwise sorted matrix
def find_in_sorted_matrix(matrix, target):
  R, C = len(matrix), len(matrix[0])

  r = 0
  c = C-1

  while r < R and c >= 0:
    if A[r][c] == target:
      return True

    if target > A[r][c]: # higher
      r = r + 1 # go down
    else:
      c = c - 1 # go left

  return False
```

	0	1	2	3	4	5
0	-1	2	4	5	9	11
1	1	4	7	8	10	14
2	3	7	9	10	12	18
3	6	10	12	14	16	20
4	11	15	19	21	24	27
5	18	24	29	32	34	42

not found

k = 32

r = 0 1 ~~2~~ ~~3~~ ~~4~~ 5

c = ~~3~~ ~~2~~ 3

TC: O(R+C)

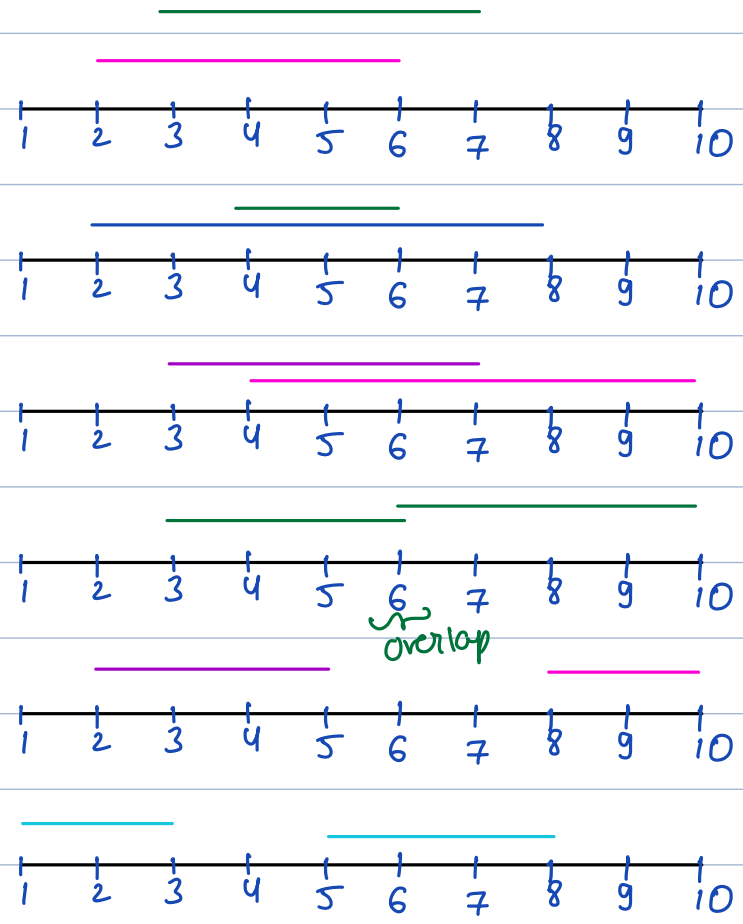
SC: O(1)

Break : 9:50

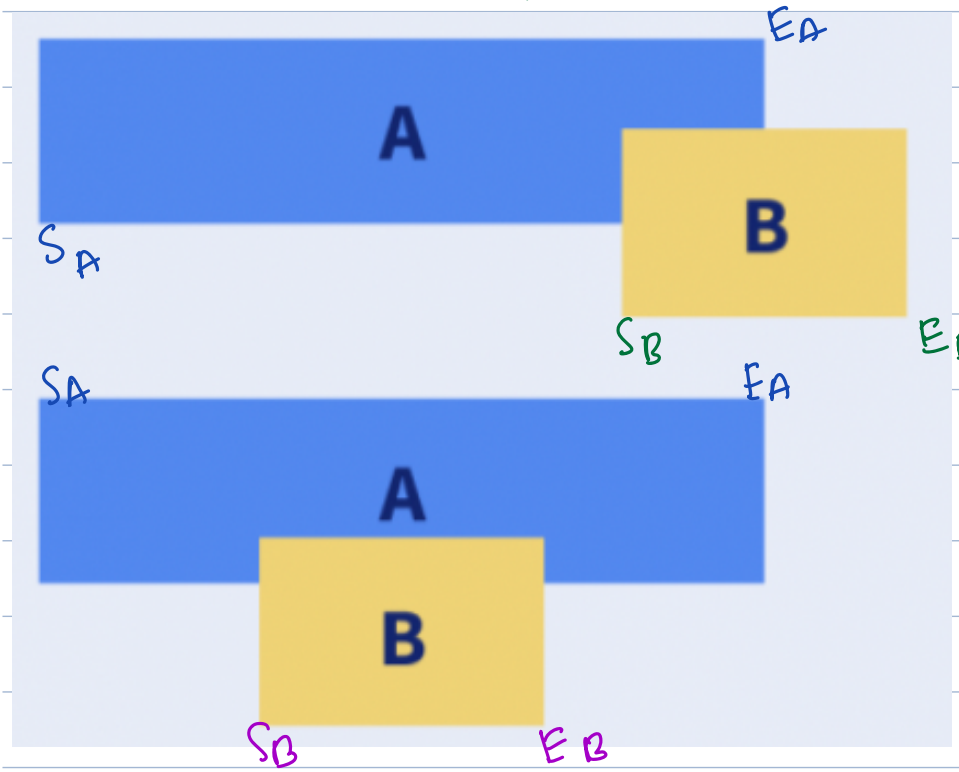
Merge Intervals

Interval — (s e)

I_1	I_2	merged s	merged e
(2 6)	(3 7)	2	7
(2 8)	(4 6)	2	8
(3 7)	(4 10)	3	10
(3 6)	(6 10)	3	10
(2 5)	(8 10)	no overlap	
(5 8)	(1 3)	no overlap	



I_A start before I_B



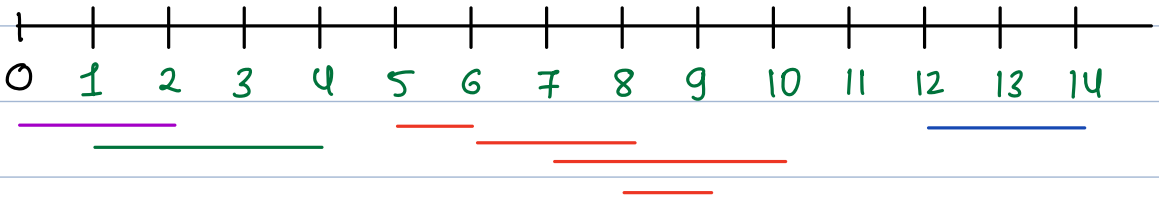
$S_B <= E_A$

$E_A >= S_B$

Merge sorted Overlapping Interval

Q) Given a sorted list of overlapping intervals, sorted based on start time, merge all overlapping intervals 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]



Given a sorted list of overlapping intervals, sorted based on start time, merge all overlapping intervals and return sorted list. Select output for following input.

Input:

Interval[] = { (1,10), (2, 3), (4, 5), (9, 12) }

(1,10) (1,10) (1,12)

Start	end	A	B	overlap	ans
1	10	(1,10)			
2	3	(1,10) >=(2,3)		yes	
4	5	(1,10) >=(4,5)		yes	
9	12	(1,10) >= (9,12)		yes	
		(1,12)			

(1,12)

A

start = [0 1 5 6 7 8 12]

end = [2 4 6 8 10 9 14]

start	end	A	B	is_overlap	ans
0	2	(0, 2)			
1	4	(0, 2)	>= (1, 4)	yes	
5	6	(0, 4)	>= (5, 6)	no	[0, 4]
6	8	[5, 6]	>= (6, 8)	yes	
7	10	[5, 8)	>= (7, 10)	yes	
8	9	[5, 10)	>= (8, 9]	yes	
12	14	[5, 10]	>= [12, 14)	no	[5, 10]
		[12, 14]			
					(12, 14)

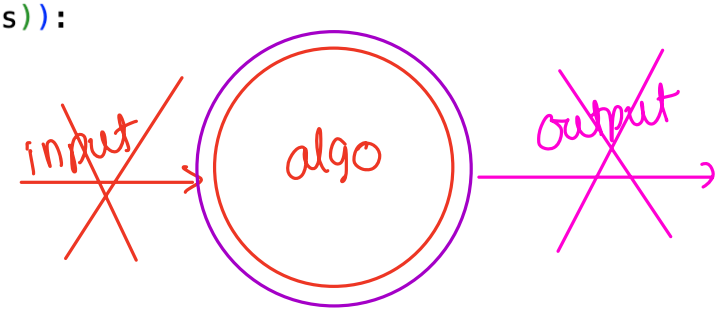
Pseudocode

[8]
✓ 0s

```python
# Problem 3: Merge sorted Overlapping Intervals
def merge_intervals(intervals): # [[1, 10], [2, 3]]
  # If the input is not sorted just sort it
  A = intervals[0]

  def is_overlap(A, B):
    sa, ea = A
    sb, eb = B

    return ea >= sb

  def merge(A, B):
    sa, ea = A
    sb, eb = B

    return [min(sa, sb), max(ea, eb)]

  ans = []
  for i in range(1, len(intervals)):
    B = intervals[i]

    if is_overlap(A, B):
      A = merge(A, B)
    else:
      ans.append(A)
      A = B

  ans.append(A)

  return ans


print(merge_intervals([[0,2], [1,4], [5,6], [6,8], [7,10], [8,9], [12, 14]]))
```

```
[[0, 4], [5, 10], [12, 14]]
```

TC: O(N)
SC: O(1)
∵ ans is expected output



Transpose of square matrix

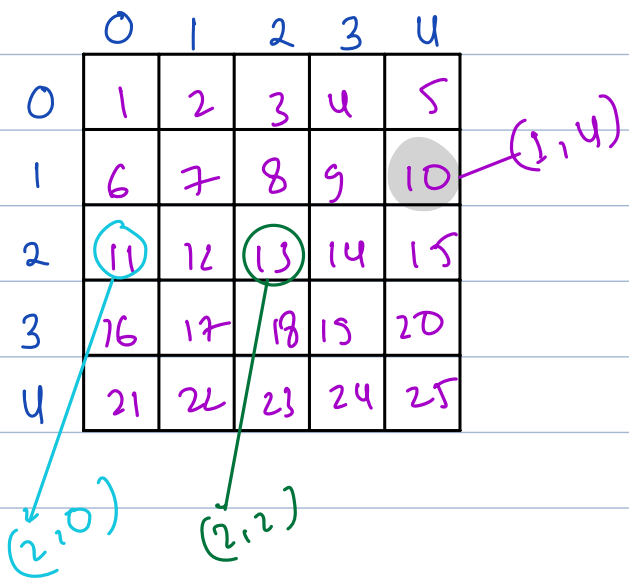
rows ⟷ col

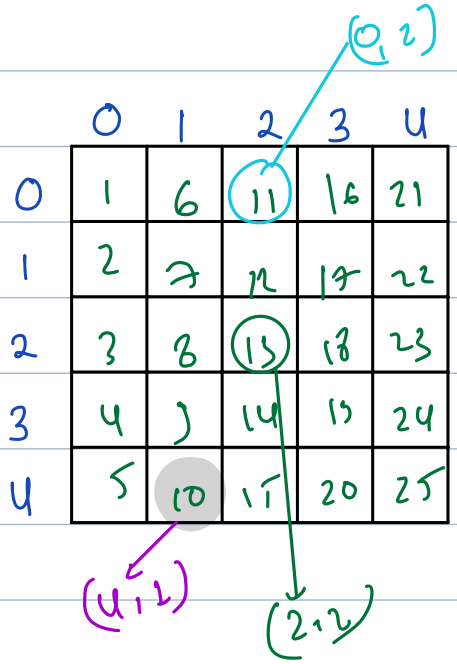
	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

matrix

	0	1	2	3	4
0	1	6	11	16	21
1	2	7	12	17	22
2	3	8	13	18	23
3	4	9	14	19	24
4	5	10	15	20	25

transpose





	0	1	2
0	1	2	3
1	6	7	8
2	11	12	13

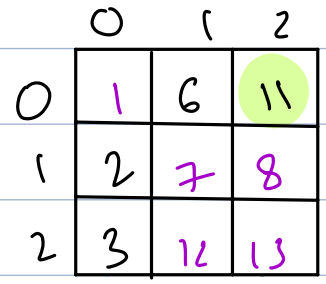
cell

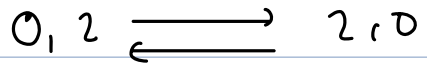
(1, 1)

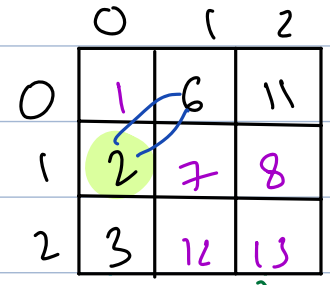
	0	1	2
0	1	2	3
1	6	7	8
2	11	12	13

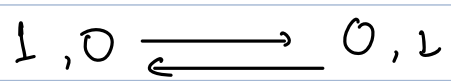
cell

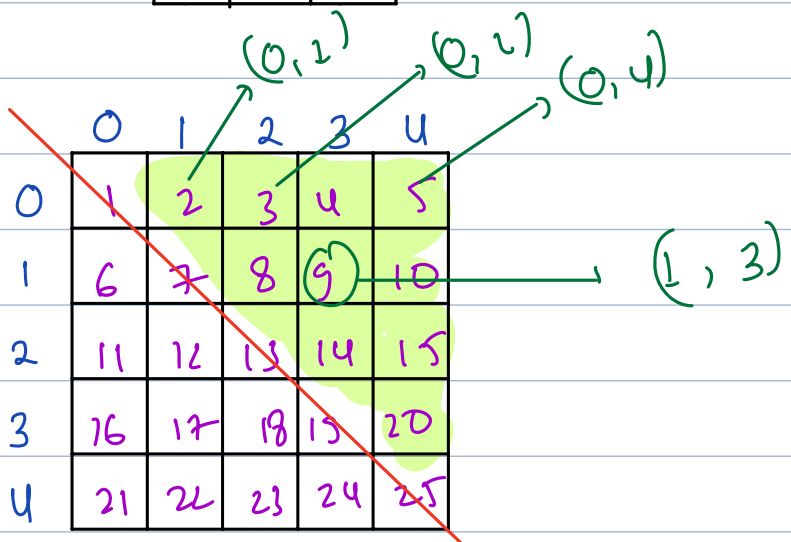
(0, 1) ⟷ (1, 0)

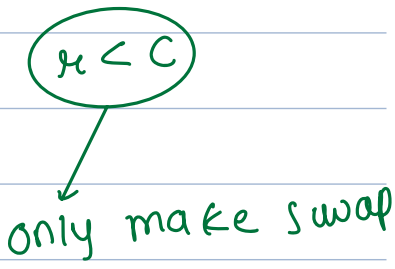













```python
# Problem 4: Transpose of a square matrix
def transpose_matrix(matrix):
  """
  Computes the transpose of a square matrix in-place.
  """
  for row in matrix:
    print(row)

  N = len(matrix)
  print("#"*10)
  for r in range(N):
    for c in range(r+1, N): # Travel only the upper triangle
      matrix[r][c], matrix[c][r] = matrix[c][r], matrix[r][c]

  for row in matrix:
    print(row)

transpose_matrix([[1,2,3],[4,5,6],[7,8,9]])
```

```
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
##########
[1, 4, 7]
[2, 5, 8]
[3, 6, 9]
```

TC: O(N²)

SC: O(1)

Rotate a matrix to 90 degree clockwise

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

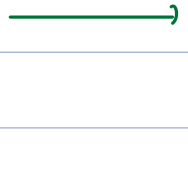
matrix

	4	3	2	1	0
0	21	16	11	6	1
1	22	17	12	7	2
2	23	18	13	8	3
3	24	19	14	9	4
4	25	20	15	10	5

90° rotation

	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

matrix



	4	3	2	1	0
0	21	16	11	6	1
1	22	17	12	7	2
2	23	18	13	8	3
3	24	19	14	9	4
4	25	20	15	10	5

90° rotation

Based on observation row → col

transpose

	0	1	2	3	4
0	1	6	11	16	21
1	2	7	12	17	22
2	3	8	13	18	23
3	4	9	14	19	24
4	5	10	15	20	25

transpose

	4	3	2	1	0
0	21	16	11	6	1
1	22	17	12	7	2
2	23	18	13	8	3
3	24	19	14	9	4
4	25	20	15	10	5

90° rotation

rows are reversed

Idea ⟶ transpose and then reverse

[14] ✓ 0s

```python
# Problem 5: Rotate a matrix to 90 degree clockwise
def rotate_matrix_90_clockwise(matrix):
  """
  Rotates a square matrix 90 degrees clockwise in-place.
  """
  transpose_matrix(matrix)
  print("#" * 10)
  for row in matrix:
    row.reverse()
    print(row)

rotate_matrix_90_clockwise([[1,2,3],[4,5,6],[7,8,9]])
```

```
[1, 2, 3]
[4, 5, 6]
[7, 8, 9]
##########
[1, 4, 7]
[2, 5, 8]
[3, 6, 9]
##########
[7, 4, 1]
[8, 5, 2]
[9, 6, 3]
```

TC: $O(N^2)$