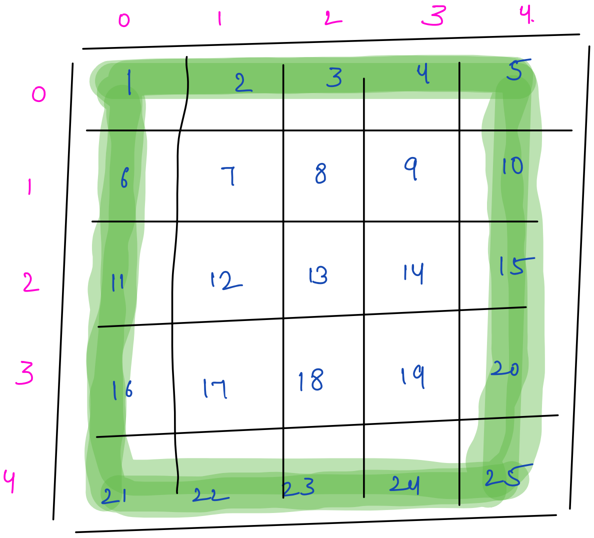
Lecture :- Sliding window

Agenda

- Boundary print of matrix.
- Spiral print of matrix
- Sliding window technique
  - Max subarray sum of len = k
  - Minimum swaps.

Class starts at 8: 35 pm.

Qu Given $\underbrace{mat[n][n]}_{\text{rows and cols are equal}}$, print boundary (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

op →

print :-

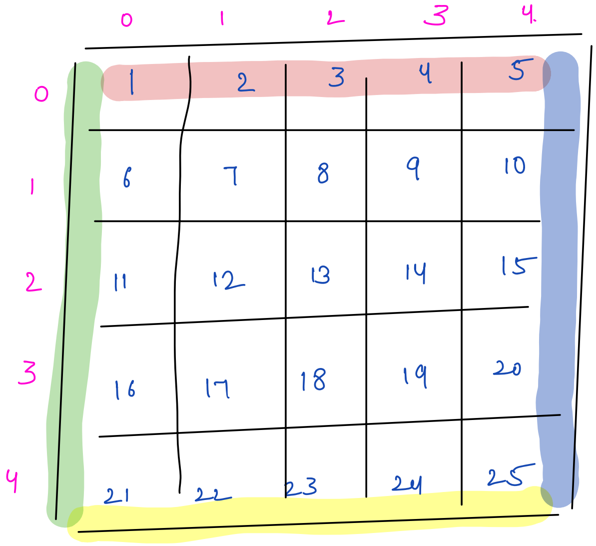
1 2 3 4 5

10 15 20 25

24 23 22 21

16 11 6

childish approach.



	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

→

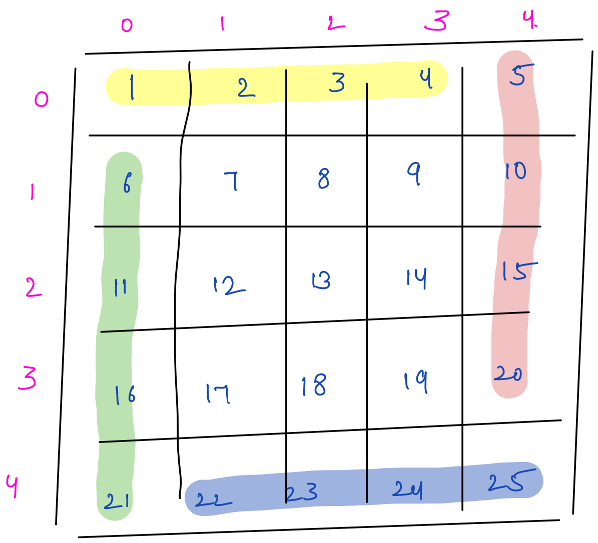
1st row:- 1 2 3 4 5

last col:- 5 10 15 20 25

last row:- 25 24 23 22 21

1st col: 21 16 11 6 1

Observation



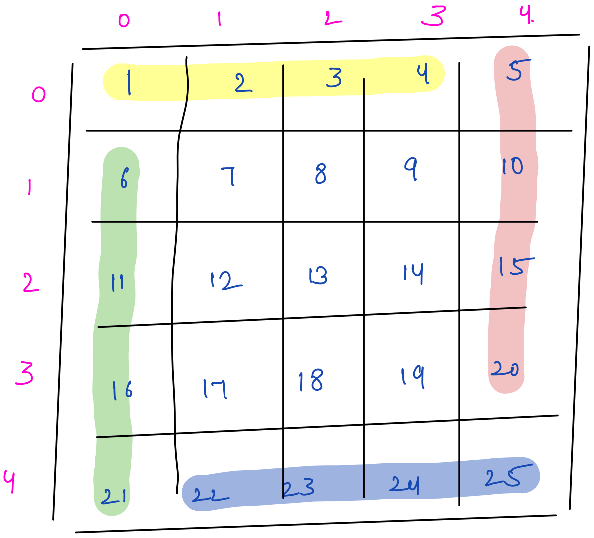
$$\begin{array}{c|ccccc} & 0 & 1 & 2 & 3 & 4 \\ \hline 0 & 1 & 2 & 3 & 4 & 5 \\ 1 & 6 & 7 & 8 & 9 & 10 \\ 2 & 11 & 12 & 13 & 14 & 15 \\ 3 & 16 & 17 & 18 & 19 & 20 \\ 4 & 21 & 22 & 23 & 24 & 25 \end{array}$$

Yellow :- 1 2 3 4

→ Red : 5 10 15 20

Blue : 25 24 23 22

Green : 21 16 11 6



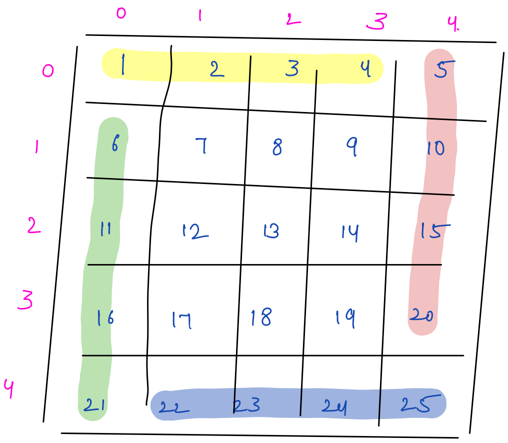
	0	1	2	3	4
0	1	2	3	4	5
1	6	7	8	9	10
2	11	12	13	14	15
3	16	17	18	19	20
4	21	22	23	24	25

Yellow :- (n-1) el.

i j
0 0 ⟶ 1
+1
0 1 ⟶ 2
+1
0 2 ⟶ 3
+1
0 3 ⟶ 4.
+1
0 4 ⟶ stop.

Red : (n-1) el

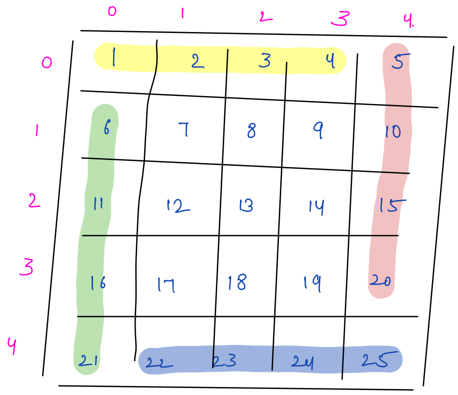
i j
+1
0 4 ⟶ 5
+1
1 4 ⟶ 10
+1
2 4 ⟶ 15
+1
3 4 ⟶ 20
+1
4 4 ⟶ stop



	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

Blue: (n-1) el

i j
4 4 ⟶ 25
-1
4 3 ⟶ 24
-1
4 2 ⟶ 23
-1
4 1 ⟶ 22
-1
4 0 ⟶ stop



	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

Green :-
(n-1) el

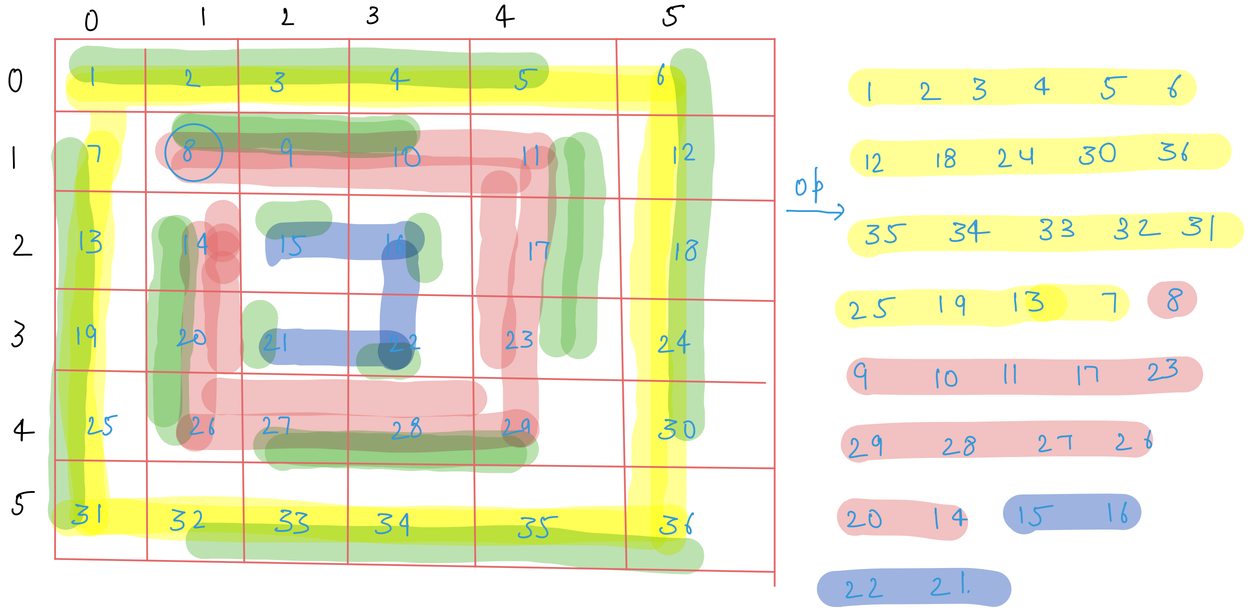
i j
-1
4 0 ⟶ 21
-1
3 0 ⟶ 16
-1
2 0 ⟶ 11
-1
1 0 ⟶ 6
-1
0 0 ⟶ stop

```
void printBoundary(int[][] mat)  {
                   Square
    int n = mat.length;
    int i = 0;
    int j = 0;

    // Print yellow.
    for( print=1; print<=n-1; print++) {
            print (mat[i][j]);
            j++;
    }

    // Print Red
    for( print=1; print<=n-1; print++) {
            print (mat[i][j]);
            i++;
    }

    // Print Blue
    for( print=1; print<=n-1; print++) {
            print (mat[i][j]);
            j--;
    }

    // Print Green
    for( print=1; print<=n-1; print++) {
            print (mat[i][j]);
            i--;
    }
}
```

TC: O(n).

SC: O(1)

Qu: Spiral print [mat[n][n]]



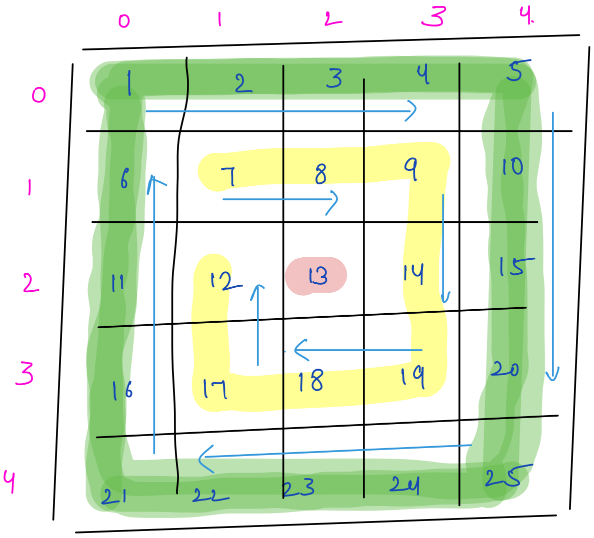
	0	1	2	3	4	5
0	1	2	3	4	5	6
1	7	8	9	10	11	12
2	13	14	15	16	17	18
3	19	20	21	22	23	24
4	25	26	27	28	29	30
5	31	32	33	34	35	36

op →

1 2 3 4 5 6
12 18 24 30 36
35 34 33 32 31
25 19 13 7 8
9 10 11 17 23
29 28 27 26
20 14 15 16
22 21.

colour	i	j	no of el to print in each color of boundary
Yellow. (outerboundary).	0 +1	0 +1	5· (n-1) -2 1 2 3 4 5 6 12 18 24 30 36 35 34 33 32 31 25 19 13 7
Red.	1 +1	1 +1	3 (n-3) -2 8 9 10 11 17 23 29 28 27 26 20 14
Blue	2	2	1. n-5 -2 15 16 21 22 ① [stop].

Edge case:



	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

$n*n = 5*5$ matrix.

colour	i	j	no of el to print in each color of boundary
Green	0	0	4 (n-1)
	+1	+1	-2
Yellow	1	1	2 (n-3)
	+1	+1	-2
	2	2	0 → stopping

100% sure, only one el would left for print. Handle explicitly.

```
void  spiralPrint (int[][] mat) {
        int  n= mat.length;
        int  i = 0, j=0 , steps  = n-1;
        while ( steps  >= 0)  {
                if( steps ==0)  {
                        print(mat[i][j]);
                        break;
                }
              // first row(excluding last el)
              for( print=1; print<=steps; print++) {
                    print (mat[i][j]  ;
                    j++;
            }

            // last col(excluding last el)
            for( print=1; print<=steps; print++) {
                  print (mat[i][j]  ;
                  i++;
          }

          // last row(excluding lst el)
          for( print=1; print<=steps; print++) {
                print (mat[i][j]  ;
                j--;
        }

        // first col(excluding lst el).
        for( print=1; print<=steps; print++) {
              print (mat[i][j]  ;
              i--;
      }
      step = step-2;
      i++;
      j++;
    }
}
```

TC: $O(n^2)$

SC: $O(1)$

Sliding window technique (fixed window length). → carry forward 2.0

Qu.3. Given arr[n], return max subarray sum in array. of len=K

Example:

	0	1	2	3	4	5	6	7	8
arr[9] =	2	4	3	7	9	8	6	5	4

ans = 24

len = 3.

9

14

19

24

23

19

15

subarray = arr[s, e] → sum = Prefix sum.

	0	1	2	3	4	5	6	7	8
arr[9] =	2	4	3	7	9	8	6	5	4
pf[9] =	2	6	9	16	25	33	39	44	48

k=3.

Dry run:

s	e	sum[s,e] = pf[e] − pf[s−1]	max. (−∞)
0	2	pf [2] = 9.	9.
1	3	pf[3] − pf[0] 16 − 2 = 14	14
2	4	pf[4] − pf[1] 25 − 6 = 19	19
3	5	pf[5] − pf[2] 33 − 9 = 24	24.
4	6	pf[6] − pf[3] 39 − 16 = 23	24
⋮	⋮	⋮	⋮
6	8		

arr[s, e]

$\uparrow \quad \uparrow$

[0 k-1]

$k-\cancel{1}-0+\cancel{1}$

$\boxed{k}$

```
int maxsum(int[] arr, int k) {
    int[] pf = prefixoptional(arr);
    int s = 0;    int n = arr.length;
    int e = k-1;    int max = Integer.MIN_VALUE;
    while( e < n) {
        if ( s==0) {
            max = math.max (max, pf[e]);
        } else{
            max = math.max (max, pf[e] - pf[s-1]);
        }
        s++;
        e++;
    }
    return max;
}
```

TC: O(n)

SC: O(n)

Approach2

SC: O(1).

$$arr[9] = \begin{array}{ccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\ 2 & 4 & 3 & 7 & 9 & 8 & 6 & 5 & 4 \\ & 9 & & & & & & & \end{array} \quad k=3.$$

$S_y = 9.$

$S_g = S_y - 2 + 7$

$S_b = S_g - 4 + 9.$

$S_r = S_b - 3 + 8.$

window	s	e	sum	max
1st	0	2 (k−1)	9	9
2nd	1	3 (k).	9 − 2 + 7. = 14 ↑ sum ↑ arr[0] ↑ arr[3] sum - arr[s-1] + arr[e]	14
3rd	2	4	14 − 4 + 9 = 19 ↑ sum ↑ arr[s−1] ↑ arr[e]	19.
⋮	⋮	⋮	⋮	⋮
x window	s	e	sum − arr[s−1] + arr[e]	max(max, sum).

↑

Edge case:-

s = 0

{ handle the sum explicitly)

```
int subarrayofsumK( int[] arr, int k) {
        int sum=0;
        // sum of 1st window , s=0 , e=k-1.
        for( i=0; i<k; i++) {
              sum += arr[i];
        }
        int max = sum;
        int s=1;
        int e=k;
        while( e < n) {
              sum = sum - arr[s-1] + arr[e];
              max = Math.max(max, sum);
              s++;
              e++;
        }
      return max;
}
```

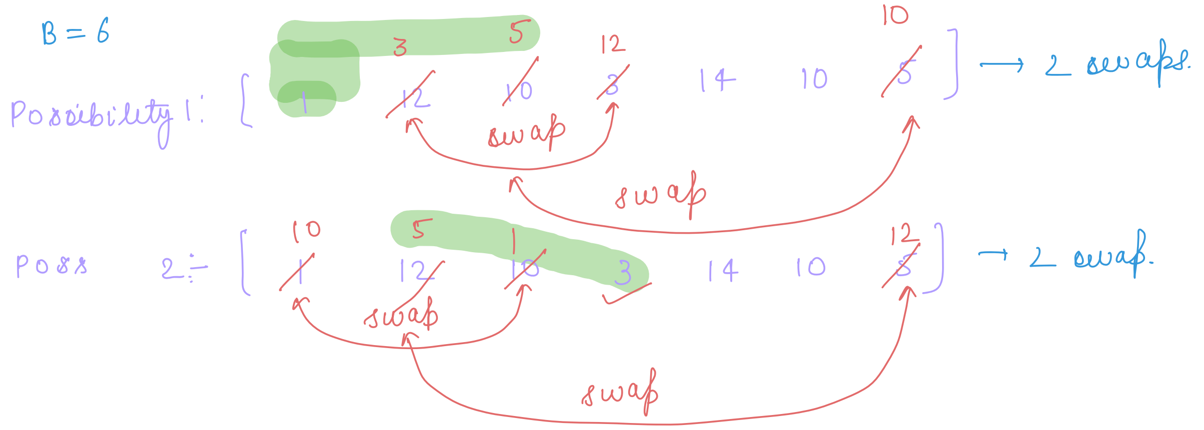
TC: O(n)

SC: O(1)

Qu 4 **Min swaps. :-** Given arr(n) and integer B. find and return the min no of swaps required to bring all no less than or equal to B together.

arr[7] = [1 12 10 3 14 10 5]

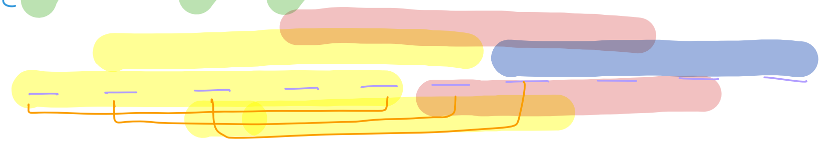
B = 6



Constraint: SC: O(1)

Idea: arr[10] = [1 12 6 3 8 13 15 13 4 5]
(indices 0 1 2 3 4 5 6 7 8 9)

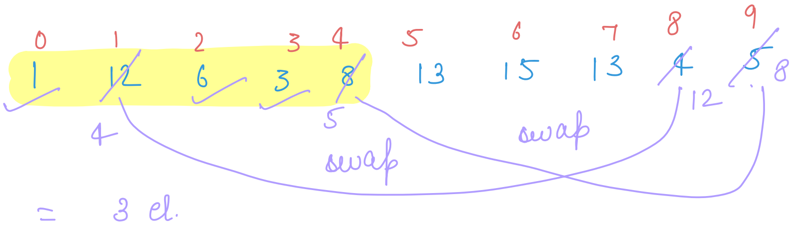
B = 7.



window len = no of el smaller than 7. (5).

Dry run:

1.) s = 0 , e = 4.



count el < B (7) = 3 el.

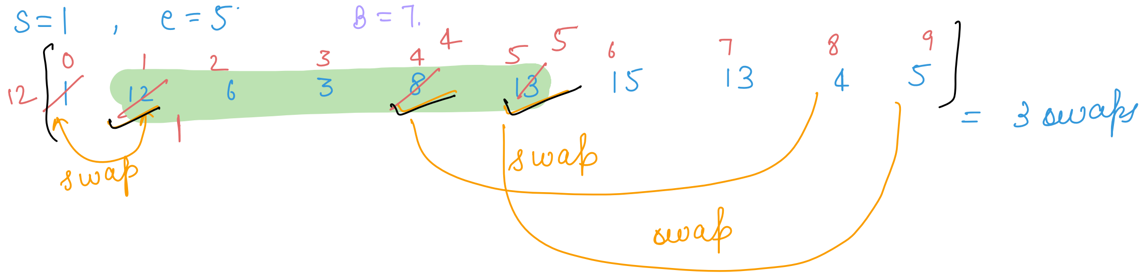
swaps = len of window − count

(or)

swaps = no of el greater than B.

} 2 swaps.

2.) S = 1 , e = 5. B = 7.



= 3 swaps

generalisation:

swaps required in a window [s, e] = no of el. greater than B.

$$\text{arr}[10] = \left[\begin{array}{cccccccccc} 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 \\ 1 & 12 & 6 & 3 & 8 & 13 & 15 & 13 & 4 & 5 \end{array}\right]$$

B = 7.

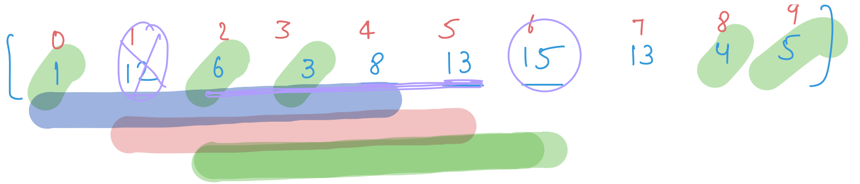
len of window = 5. (el smaller than or equal to B)

s	e	count of el greater than B (count)	swaps	ans (+∞)
0	4 (k-1)	12, 8 = 2	2	2
1	5 (len)	2 − [if ① > 7 → 1, else 0] + [if ⑬ > 7 → 1, else → 0] arr[s-1] arr[e] 2 − 0 + 1 = 3	3	2
⋮	⋮	⋮	⋮	⋮
s	e	count − [if arr[s-1] > B → 1, else → 0] + [if arr[e] > B → 1, else → 0]		

edge case → s = 0

Dry run:

arr [10] =



s = 2 , e = 5

count = 3 − [12 > 7 → 1 ; else → 0] + [15 > 7 → 1 ; else → 0]

3 − 1 + 1 = 2.

Implementation:

```
int minswaps(int[] arr , int B) {
    int len = 0;    int count = 0;
    for( i=0; i< arr.length; i++) {
        if( arr[i] <= B) {
            len++;
        }
    }
    // s=0, first window
    for( i=0; i< len; i++) {
        if (arr[i] > B) {
            count++;
        }
    }
    int ans = count;
    int s = 1;
    int e = len;
```

```
        while( e < arr.length) {
            if( arr[s-1]  > B) {
                count = count -1;
            }
            if (arr[e] > B) {
                count = count +1;
            }
            ans = math.min(ans, count);
            s++;
            e++;
        }
        return ans;
    }
```

TC: O(n)

SC: O(1).

Thankyou ☺

H/w :- Try out boundary & spiral for mat[n][m].

Doubts