Welcome ☺

Agenda : Sliding Window
1-2 problem on 2D arrays

Q1 Given N elements, print max subarray sum of len = K

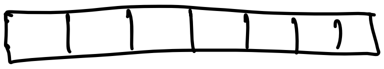
eg: arr[10]: { -3 4 -2 +5 3 -2 8 2 -1 4 }

	0	1	2	3	4	5	6	7	8	9
arr[10]	-3	4	-2	+5	3	-2	8	2	-1	4

K = 5

s	e	sum
0	4	7
1	5	8
2	6	12
3	7	16
4	8	10
5	9	11
		16

App 1 : Brute Force
→ For every subarray of size K, iterate and calculate sum, compare for all subarrays and return max sum.



	K=1	K=2	K=3	K
#sub arrays	N	N-1	N-2	N-K+1

T.C = (N-K+1).K

- K=1: T.C = N, O(N)
- K=N: T.C O(N)
- K = N/2: $(N-\frac{N}{2}+1)\frac{N}{2}$, $O(N^2)$

```
int maxSubarray (arr, N, K)
{
    s = 0 , e = K-1 , ans = INT_MIN
    while ( e < N )
    { // iterate and calculate sum
        int sum = 0 ;
        for( int i = s ; i ≤ e ; i++ ) {
            sum += arr[i] ;
        }
        if ( sum > ans) { ans = sum; }

        s++ , e++ ;
    }
    return ans;
}
```

T.C = $O(N^2)$
S.C = $O(1)$

// Approach 2 → Prefix Sum

```
int maxSubarray (arr, N, K)
{
    ① Create Prefix Sum
    s = 0 , e = K-1 , ans = INT_MIN

    while ( e < N )
    { // iterate and calculate sum
        int sum = 0 ;
        if( s == 0)   sum = psum[e]
        else          sum = psum[e] - psum[s-1]

        if( sum > ans) { ans = sum; }

        s++ , e++ ;
    }
    return ans;
}
```

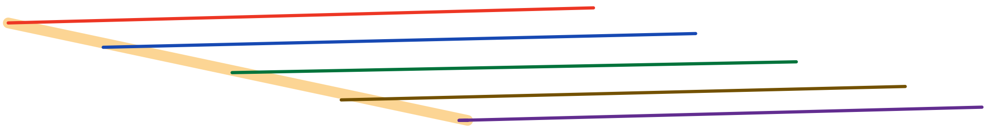
T.C -> O(N)

SC -> O(N

// Approach

k = 6

arr[10] : { $\overset{0}{3}$ $\overset{1}{4}$ $\overset{2}{-2}$ $\overset{3}{5}$ $\overset{4}{3}$ $\overset{5}{-2}$ $\overset{6}{8}$ $\overset{7}{2}$ $\overset{8}{1}$ $\overset{9}{4}$ }



s	e	sum
0	5	11
1	6	sum = sum - arr[0] + arr[6] = 11-3+8 = 16
2	7	sum = sum - arr[1] + arr[7] = 16-4+2 14
⋮		
s	e	sum = sum - arr[s-1] + arr[e]

[Carry forward + subarrays are of same size ⇒ Sliding Window]

```
// Sliding Window app.

    int maxSubSum ( arr, N, K)
    {
     // Calculate sum of first K elements [ first window ]
          sum = 0
          for( i=0 ; i< K ; i++) {
               sum += arr[i] ;                    ]  K iteraⁿ
          }
          S = 1 ,   e = K ,   ans = sum ;

          while ( e<N)                                        ]
          {                                                   ]
               // calculate sum of subarray [s, e]            ]  N-K
               sum = sum - arr[s-1] + arr[e]                  ]
               // compare                                     ]
               if( sum > ans)  ans = sum ;                    ]
               s++ ,  e++ ;                                   ]
          }
          return ans ;
    }
```

T.C = O(N)
S.C = O(1)

Q2 Given an array of size N and a number B. Find and return minimum no. of swaps to bring all numbers ≤ B together.

eg: $arr = \{ \boxed{1}\ \ 12\ \ 10\ \ \boxed{3}\ \ 14\ \ 10\ \ \boxed{5} \}$ $B = 8$

ans = 2

$arr = \{ 19\ \ 11\ \ \boxed{3}\ \ \boxed{9}\ \ \boxed{7}\ \ 25\ \ \boxed{6}\ \ 20\ \ \boxed{4} \}$ $B = 10$

ans = 1

$$arr = \{ \overset{0}{25}\ \ \overset{1}{30}\ \ \overset{2}{\boxed{2}}\ \ \overset{3}{18}\ \ \overset{4}{\boxed{7}}\ \ \overset{5}{\boxed{6}}\ \ \overset{6}{\boxed{9}}\ \ \overset{7}{\boxed{3}}\ \ \overset{8}{50} \} \quad B = 10$$

ans = 1

1. Count of all elements ≤ B [K]
2. subarray will be of size K (window length)
3. Find subarray for which swaps are minimum..

	# swaps
0-4	3
1-5	2
2-6	1
3-7	1
4-8	1

ans = 1

⇒ for all elements > B → bad elements
 " " " ≤ B → good element

Pseudo code

```
int minSwap ( arr, N, B )
{
    // count no. of elements ≤ B ( to fin window size)
        K = 0
        for( i = 0 ; i < N ; i++) {
            if( arr[i] ≤ B )  K++ ;
        }
        if( K = 0  || K = 1 )  return 0 ;
    // calculate no. of bad elements for first window
        bad = 0
        for( i = 0 ; i < K ; i++) {
            if( arr[i] > B )  bad ++ ;
        }
    // applying sliding window
        ans = bad ; s = 1 ; e = K
        while ( e < N )
        {
            if ( arr[s-1] > B )  bad --
            if ( arr[e] > B )  bad ++
            if ( bad < ans )  ans = bad

            s ++ ; e ++ ;
        }
        return ans ;
}
```

Q Given mat[N][N] , print boundary in clockwise direc^n.

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

o/p → 1 2 3 4 5 10 15 20 25 24 23 22 21 16 11 , 6

N-1 →

N-1 ↓

N-1 ←

N-1 ↑

Pseudo code

```
void printBoundary (arr, N)
{
    i=0 , j=0
    // print N-1 element from l → r
    for ( k = 1 ; k < N ; k++) {
        print (arr[i][j])
        j++
    }
    // print N-1 elements from t to d
    for ( k = 1 ; k < N ; k++) {
        print (arr[i][j])
        i++
    }
    // print N-1 element from r → l
    for ( k = 1 ; k < N ; k++) {
        print (arr[i][j])
        j--
    }
    // print N-1 elements from d → t
    for ( k = 1 ; k < N ; k++) {
        print (arr[i][j])
        i--
    }
}
```

k	i	j
1	0	0
2	0	1
3	0	2
4	0	3
	0	4

4 4

4, 0

0, 0

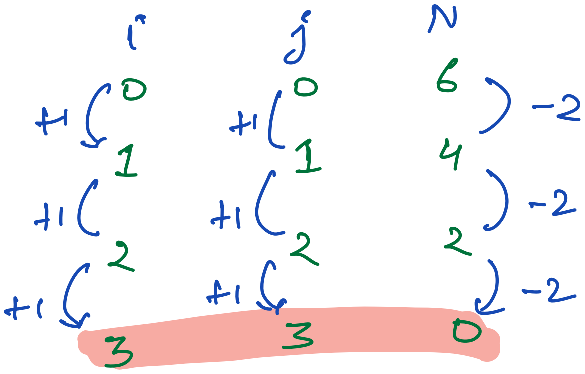
T.C = O(N)

S.C = O(1)

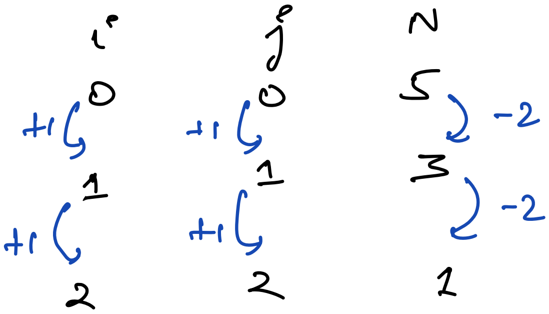
// Spiral printing.

arr[6][6]

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



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



```
void  printBoundary ( arr, N )
{
    i=0 , j=0

    while ( N>1 )
      {
          // print N-1 element from l→r
          for ( K=1 ; K<N ; K++) {
              print (arr[i][j])
              j++
          }
          // print N-1 elements from t to d
          for ( K=1 ; K<N ; K++) {
              print (arr[i][j])
              i++
          }
          // print N-1 element from r→l
          for ( K=1 ; K<N ; K++) {
              print (arr[i][j])
              j--
          }
          // print N-1 elements from d → t
          for ( K=1 ; K<N ; K++) {
              print (arr[i][j])
              i--
          }
          i++ , j++ , N=N-2 ;
      }
      if (N==1) { print (arr[i][j]) }
}
```

T.C
$O(N^2)$

S.C
$O(1)$