- Spiral Printing
- Sliding Window
  - Max sub array sum of len = k
  - Min swaps
- Prefix Sum

Q: Given mat[N][N], print boundary in clockwise direction

mat[5][5]

	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

mat[3][3]

	0	1	2
0	1	2	3
1	4	5	6
2	7	8	9

output:

1 2 3 6 9 8 7 4

output:

1 2 3 4 5 10 15 20 25 24 23 22 21 16 11 6

mat[5][5]

	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

Ideas:

N=5:

4 Iterations to right

4 iteration to bottom

4 Iterations to left

4 Iterations to top

// In general N*N:

Idea:

N-1 Steps →

N-1 Steps ↓

N-1 Steps ←

N-1 Steps ↑

mat[6][6]

	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

```
void printBoundary(int mat[][]){
    int n = mat.length;          // say mat[6][6], N = 6

    int i=0, j=0

    // k : [1 N-1] : N-1 iterations
    left → right
    for(int k=1; k<n; k++){
        print(mat[i][j])
        j++;
    }

    top → bottom
    for(int k=1; k<n; k++){
        print(mat[i][j])
        i++;
    }

    right → left
    for(int k=1; k<n; k++){
        print(mat[i][j])
        j--;
    }

    Bottom → top
    for(int k=1; k<n; k++){
        print(mat[i][j])
        i--;
    }
}
```

i=0, j=0
k=1; k<6; k++){

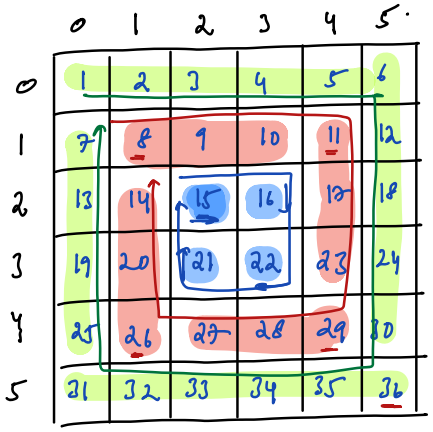
k	print(mat[i][j])	i	j
1	print(mat[0][0])	0	1
2	print(mat[0][1])	0	2
3	print(mat[0][2])	0	3
4	print(mat[0][3])	0	4
5	print(mat[0][4])	0	5
6	Stop		

k=1; k<6; k++){

k	print(mat[i][j])	i	j
1	print(mat[0][5])	1	5
2	print(mat[1][5])	2	5
3	print(mat[2][5])	3	5
4	print(mat[3][5])	4	5
5	print(mat[4][5])	5	5
6	Stop		

k	print(mat[i][j])	i	j
1	print(mat[5][5])	5	4
2	print(mat[5][4])	5	3
3	print(mat[5][3])	5	2
4	print(mat[5][2])	5	1
5	print(mat[5][1])	5	0
6	Stop		

k	print(mat[i][j])	i	j
1	print(mat[5][0])	4	0
2	print(mat[4][0])	3	0
3	print(mat[3][0])	2	0
4	print(mat[2][0])	1	0
5	print(mat[1][0])	0	0
6	Stop		

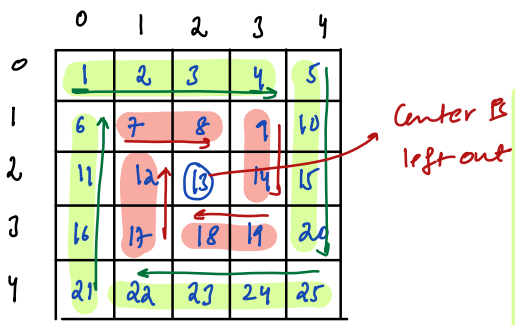


i	j	N	→ printing
0	0	6	
↓	↓	↓	
1	1	4	: iterate 3 time in each
↓	↓	↓	boundary
2	2	2	: iterate 1 time in each
↓	↓	↓	boundary
3	3	0	: Stop

// Spiral printing: ⟶ TC: $O(N^2)$ SC: $O(1)$

```
void Spiral(int mat[][]){
    int n = mat.length
    int i=0, j=0,
    while(n > 1){
        for(int k=1; k<n; k++){
            print(mat[i][j])
            j++;
        }
        for(int k=1; k<n; k++){
            print(mat[i][j])
            i++;
        }
        for(int k=1; k<n; k++){
            print(mat[i][j])
            j--;
        }
        for(int k=1; k<n; k++){
            print(mat[i][j])
            i--;
        }
        i++, j++, n=n-2
    }
    if(N==1){
        print(mat[i][j])
    }
}
```

Edge Case :



i	j	N	
0	0	5	: iterate 4 times
↓	↓	↓	
1	1	3	: iterate 2 times
↓	↓	↓	
2	2	1	: if N==1 break

Edge Case: for an odd N value
This edge case occurs

2Q) Given N arr[], elements & Q queries,
for each query l→r find no: of even numbers in given range

Eg:

	0	1	2	3	4	5	6	7	8	9
ar[10]:	2	4	3	7	9	8	6	5	4	1

Q: 3

l	r	ans
0	4	2
4	8	3
3	9	3

Idea1: for every query iterate from l→r & get count of all even numbers
TC: O(Q*N) SC: O(1)

Idea2: Using Pf[] technique

	0	1	2	3	4	5	6	7	8	9
ar[10]:	2	4	3	7	9	8	6	5	4	1
even odd										
1 0 :	1	1	0	0	0	1	1	0	1	0
Pf[10] :	1	2	2	2	2	3	4	4	5	5

l	r
4	8
3	9
0	4

if (l != 0): Pf[r] - Pf[l-1] else: Pf[r]

Pf[8] - Pf[3] = 3

Pf[9] - Pf[2] = 3

Pf[4] = 2

TC: O(N + Q) SC: O(N) TODO

to construct Pf[] — to answer all Q queries

8:32 → 8:40am

Construct Pf[]

```
// given ar[N]
int pf[N]
pf[0] = (1 - ar[0]%2);
for(i=1; i<N; i++){
    pf[i] = pf[i-1] + (1 - ar[i]%2)
}
// Now answer queries
```

38) Given N Element, return max subarray sum of len = k

	0	1	2	3	4	5	6	7	8	9
ar[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

max : 16

s	e	
0	k-1	# Subarray N-k+1
1	k	
2	k+1	
3	k+2	
:	:	
N-k	N-1	

Ideas:

1) For every subarray of len k, iterate & get sum, & calculate overall max

```
int subsum(int ar[], int k){
    int n = ar.length;
    int s = 0, e = k-1, ans = INT_MIN   // min integer value we can store, refer this in your language of choice.
    while(e < n){
        sum = 0
        from [s e]                      // for every sum from [s, e] get it via pf[]
        i = s; i <= e; i++){
            sum = sum + ar[i]
        }
        if(sum > ans) ans = sum;
        s++, e++
    }
    return ans;
}
```

TC: (N-k+1) * k SC: O(1)

Subarrays Iterate in each subarray

TC: (N-k+1)(k)

k = 1 :
TC: (N-1+1)(1)
TC: O(N)

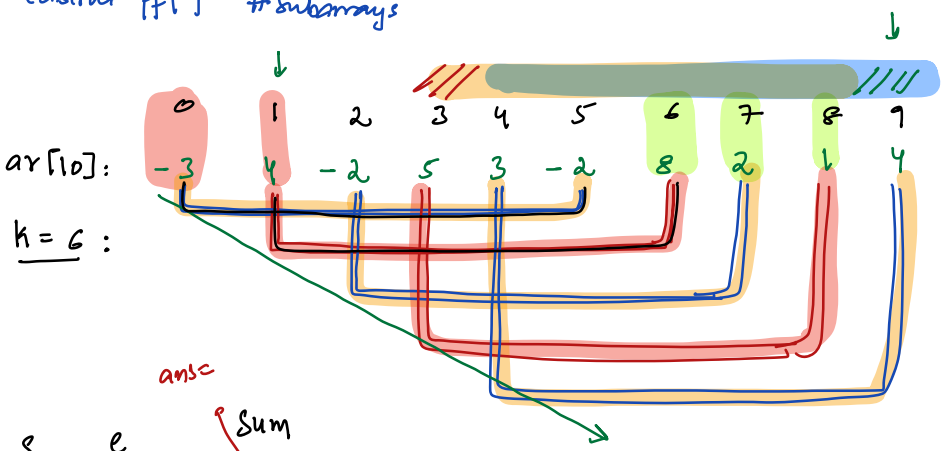
k = N :
TC: (N-N+1)(N)
TC: O(N)

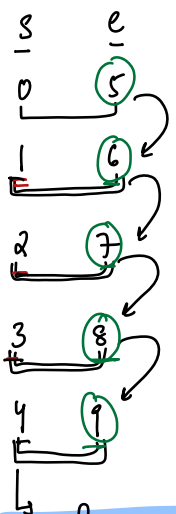
k = N/2
TC: (N - N/2 + 1)(N/2)
TC: (N/2 + 1)(N/2)
TC: (N²/4 + N/2)
TC: O(N²)

Idea: for every subarray of len=k, get it's using pf[]

TC: O(N + N-k+1 * 1) → TC: O(N) SC: O(N)

Construct pf[] #Subarrays





5 1st subarray Iterate & Calculate it

s-1 e

Sum = Sum - ar[0] + ar[6] = Sum = 16

Sum = Sum - ar[1] + ar[7] = Sum = 14

Sum = Sum - ar[2] + ar[8] = Sum = 17

Sum = Sum - ar[3] + ar[9] = Sum = 16

max = 17

Carry forward, & all subarray size same ⇒ Sliding window

```
int SubSumK(int ar[], int k){
    int sum=0;
    for(int i=0; i<k; i++){
        sum = sum + ar[i]
    }
    // We have 1 subarray sum
    ans = sum;
    s=1, e=k
    while(e<N){
        // get subarray sum from [s e]
        sum = sum - ar[s-1] + ar[e]
        if(sum > ans) { ans = sum }
        s++, e++;
    }
    return ans;
}
```

k iterations

N-k iterations

TC: N iterations

TC: O(N)

SC: O(1)