Good Evening Everyone !! ☺

> Anyone who keeps learning stays young.
>
> — Henry Ford —

Today's Content

- Sliding Window
- 2-problems on 2-D Arrays.

arr →

7	2	8	4	9	11	6
0	1	2	3	4	5	6

	K=1.	K=2.	K=3.	---	K
total no. of subarrays of len K	7	6	5		↓
	[N]	[N-1]	[N-2]		[N-K+1]

Q.1 Given N elements, **print max subarray sum of len=K.**

arr = { -3 (0) 4 (1) -2 (2) 5 (3) 3 (4) -2 (5) 8 (6) 2 (7) -1 (8) 4 (9) } , K=5

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

[ans = 16]

idea 1: for every subarray of len K, iterate & calculate the sum.
Overall max sum → {ans}

pseudo-code:

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

$(N-K+1) \cdot K$

- K = 1: $(N-1+1) \cdot 1$, TC → O(N)
- K = N: $(N-N+1) \cdot N$, O(N)
- K = N/2: $(N-\frac{N}{2}+1) \cdot (\frac{N}{2}) \approx \frac{N}{2} \cdot \frac{N}{2} = \frac{N^2}{4}$

T.C → $O(N^2)$, S.C → O(1)

Idea-2. Use prefix Sum.

// sum[s, e] = psum[e] - psum[s-1]

// 1. Create psum[N].

// 2. s = 0, e = K-1, ans →

```
while ( e < N ) {
    sum = 0
    if (s == 0) { sum = psum[e] }
    else        { sum = psum[e] - psum[s-1] }

    if (sum > ans) { ans = sum }
    s++, e++ ;
}
```

// 3. return ans.

psum → N

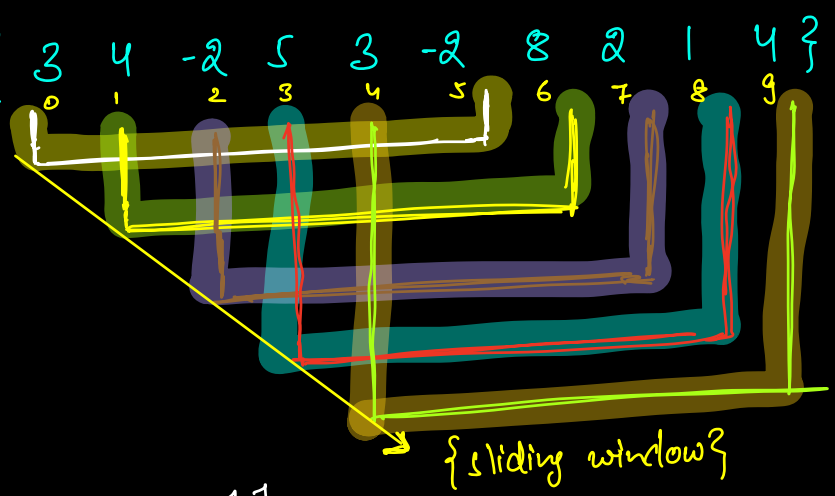
All subarrays of len K. → (N - K + 1)

T.C → O(N)
S.C → O(N)

idea-3 :

N = 10
K = 6

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



s	e	
0	5	sum = 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
3	8	sum = sum - arr[2] + arr[8] = 14 - (-2) + 1 = 17
4	9	sum = sum - arr[3] + arr[9] = 17 - 5 + 4 = 16

ans = 17

s	e	
s	e	sum = sum - arr[s-1] + arr[e]

{ Size of subarray is fixed + Carry forward } ⇒ Sliding Window

⇓

subarray of fixed size.

final code :

```
int maxSum( arr, N ) {
// calculate sum for first K elements (first window)
  sum = 0
  for ( i = 0 ; i < K ; i++){
      sum += arr[i]
  }

  s = 1 , e = K , ans → sum
  while ( e < n) {
      //calculate sum of subarray [s,e].
       sum = sum - arr[s-1] + arr[e]
       if (sum > ans){
           ans = sum
       }

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

K

N-K

T.C → O(N)
S.C → O(1)

Q) Given arr[N] and a number B. Find and return minimum no. of swaps to bring all numbers $<= B$ together.

[=> B.f. → for every subarray of size K, find count of bad elements.]

Eg: arr → { 1 12 10 3 14 10 5 }, B = 8

0	1	2	3	4	5	6

[ans = 2.]

Q) arr → { 19 11 3 9 7 25 6 20 4 }, B = 10

0	1	2	3	4	5	6	7	8

[ans = 1]

Q) arr → { 25 30 2 18 7 6 9 50 3 }, B = 10

0	1	2	3	4	5	6	7	8

[ans = 1]

→ count of all numbers $<= B$. [K].

→ size of subarray is fixed → K.

s:	e:	min. no. of swaps:
0	4	2
1	5	2
2	6	1
3	7	2
4	8	2.

[ans = 1]

For all no's which $> B$ ⇒ bad elements
For all no's which $\leq B$ ⇒ good elements.

pseudo-code →

```
int minSwaps(arr, N){
    //count good elements [K].
    K=0
    for(i=0; i<N; i++){
        if (arr[i] ≤ B){ K++ }
    }                                   ] N

    if(K==0 || K==1){return 0}

    //count bad elements for 1st window
    bad=0
    for(i=0; i<K; i++){
        if (arr[i] > B){ bad++ }
    }                                   ] K

    // Apply sliding window technique.
    s=1, e=K, ans=bad

    while(e < N){                       ] N-K.
        if (arr[s-1] > B){ bad-- }      // removing arr[s-1]
        if (arr[e] > B){ bad++ }        // add arr[e]
        if (bad < ans){ ans=bad }
        s++, e++;
    }
    return ans
}
```

T.C → O(N)
S.C → O(1)

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

O/P → {1,2,3,6,9,8,7,4}

O/P → {1,2,3,4,5,10,15,20,25,24,23,22,21,16,11,6}

idea:

print N-1 →

N-1 ↓

N-1 ←

N-1 ↑.

- ~~psum~~
- ~~carry forward~~
- ~~sliding window~~
- ~~contribution technique~~

Advanced module.

pseudo-code

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

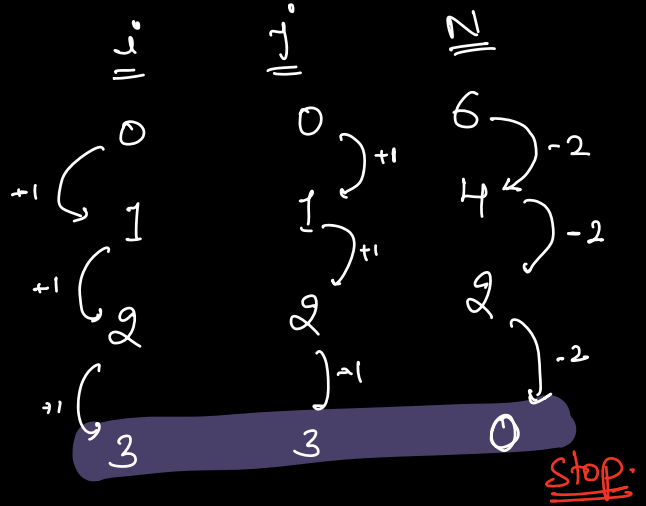
0, 0

T.C → O(N)

S.C → O(1)

arr[6][6]

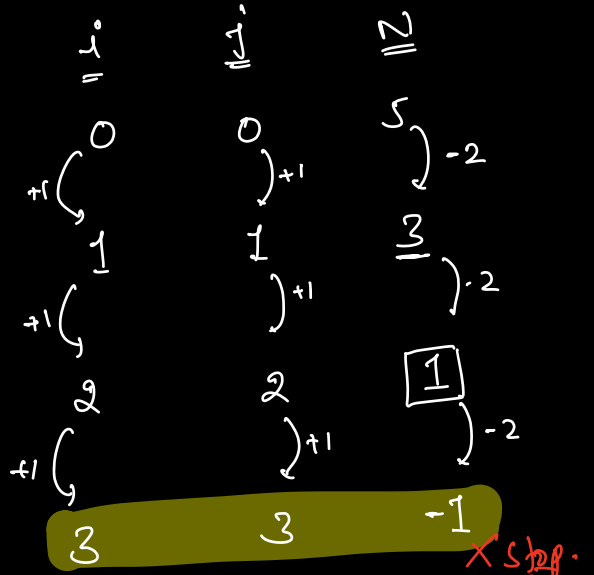
	0	1	2	3	4	5
0	1	2	3	4	5	6
1	7	8	9	10	11	12
2	13	14	15	16	17	18
3	19	20	21	22	23	24
4	25	26	27	28	29	30
5	31	32	33	34	35	36



"Spiral Printing"

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



pseudo-code:

```
void spiralPrinting ( arr, N ){
    i = 0, j = 0
while( N > 1 ){
    //print N-1 elements from l to 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 elements from r to l
    for( k = 1 ; k < N ; k++){
        print( arr[i][j] )
        j--
    }
    //print N-1 elements from d to t
    for( k = 1 ; k < N ; k++){
        print( arr[i][j] )
        i--
    }
    i++ , j++ , N -= 2
}
if( N == 1 ) { print arr[i][j] }
}
```

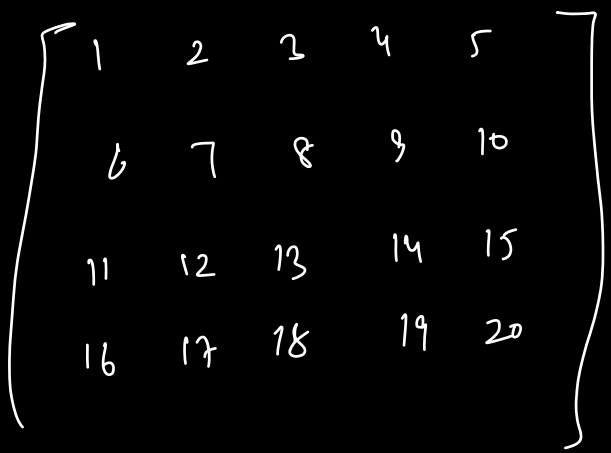
T.C → $O(N^2)$

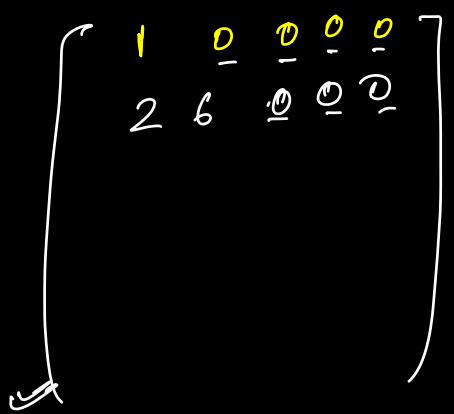
S.C → $O(1)$

{ arr[N][M]. "spiral printing" }.

- ✓ Binary Search [log N].
- ✓ prime no's [$\sqrt{n}$].

Advanced module.





→ Advanced module. (medium hard).

→ easy / medium