Today's Agenda :-

Sliding window &

contribution Technique .

Ques.

Given an array of integers, we need to find the sum of all possible subarrays of the array and maintain the maximum sum.

arr[] = {1, 2, 3}

$\frac{n(n+1)}{2} \Rightarrow \frac{3(3+1)}{2} \Rightarrow 6$.

[1] => 1

[1, 2] => 3

[1, 2, 3] => 6

Ans -> 6.

[2] => 2

[2, 3] => 5

[3] => 3

Brute force.

T.C -> $O(n^3)$

S.C -> $O(1)$.

```
max_sum = -∞;
for (i=0; i<n; i++) {          // s
    for (j=i; j<n; j++) {      // es
        int subarray_sum = 0;
        for (k=i; k<=j; k++) {
            subarray_sum += arr[k]
        }
        print (subarray_sum);
        max_sum = Max (max_sum,
                       subarray_sum);
    }
}
```

// prefix sum :-

```
// Calculate the prefix sum of the array
int prefix[n];
prefix[0] = arr[0];
for (int i = 1; i < n; i++) {
    prefix[i] = prefix[i-1] + arr[i];
}
```

} n.

max_sum = -∞;

for (i=0; i<n; i++) { // s } n^2.

for (j=i; j<n; j++) { // es

int subarray_sum = 0;

if (i==0) {

subarray_sum = prefix[j];

else {

subarray sum = prefix[j] - prefix[i-1];

}

Print (subarray_sum);

max_sum = Max (max_sum, subarray_sum);

}

}

T.C -> O(n^2)

S.C -> O(n).

Optimization using carry forward :-

	0	1	2	3
A[] =	-4	1	3	2

i	j	
0	0	{-4}
0	1	{-4, 1}
0	2	{-4, 1, 3}
0	3	{-4, 1, 3, 2}

```
max_sum = -∞;
for (i=0; i<n; i++) {          // s
    for (j=i; j<n; j++) {      // en
        int subarray_sum = 0;
        for (k=i; k<=j; k++) {
            subarraySum += arr[k
        }
        Print (subarray_sum);
        max_sum = Max (max_sum,
                       subarray_s
    }
}
```

```
int maxSum = -∞, currSum = 0;
for (i=0; i<n; i++) {
    currSum = 0;
    for (j=i; j<n; j++) {
        currSum = currSum + arr[j];
        print (currSum);
        MaxSum = Max (currSum, MaxSum);
    }
}
print (MaxSum);
```

A[] = {-4, 1, 3, 2}

	0	1	2	3
i	0	0	0	0
j	0	1	2	3
currSum	-4	-3	0	2
	1	1	1	1
	1	2	3	
	-3	-1	2	
	2	3		
	-1			
	4			

T.C → $O(n^2)$

S.C → $O(1)$.

Ques).

Google facebook

Given an array of integers, find the total sum of all possible subarrays.

arr[] = {1, 2, 3}

[1] => 1

[1, 2] => 3

[1, 2, 3] => 6

[2] => 2

[2, 3] => 5

[3] => 3

20.

```
totalSum = 0;
int maxSum = -∞, currSum = 0;
for (i=0; i<n; i++) {
    currSum = 0;
    for (j=i; j<n; j++) {
        currSum = currSum + arr[j];
        totalSum += currSum;
    }
}
Print (totalSum);
```

T.C → O(n²)

S.C → O(1)

Contribution Technique

arr[] = {1, 2, 3}

[1] => 1

[1, 2] => 3

[1, 2, 3] => 6

[2] => 2

[2, 3] => 5

[3] => 3

20.

(1×3 + 2×4 + 3×3)

11

20.

In how many subarrays, the element at index 1 will be present?

A: [3, -2, 4, -1, 2, 6]

0 1 2 3 4 5

→ 10 subarrays.

[3, -2, 4, -1, 2, 6]

[3, -2, 4, -1, 2, 6]

2

[3, -2, 4, -1, 2, 6]

2

[3, -2, 4, -1, 2, 6]

2

s.

i

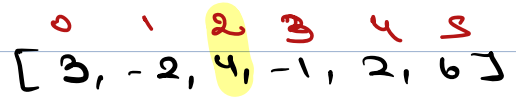
[3, -2, 4, -1, 2, 6]

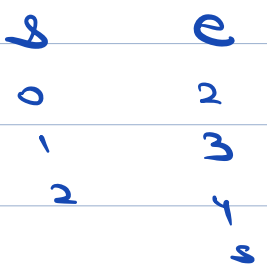
0 1 2 3 4 5

s: 0 1

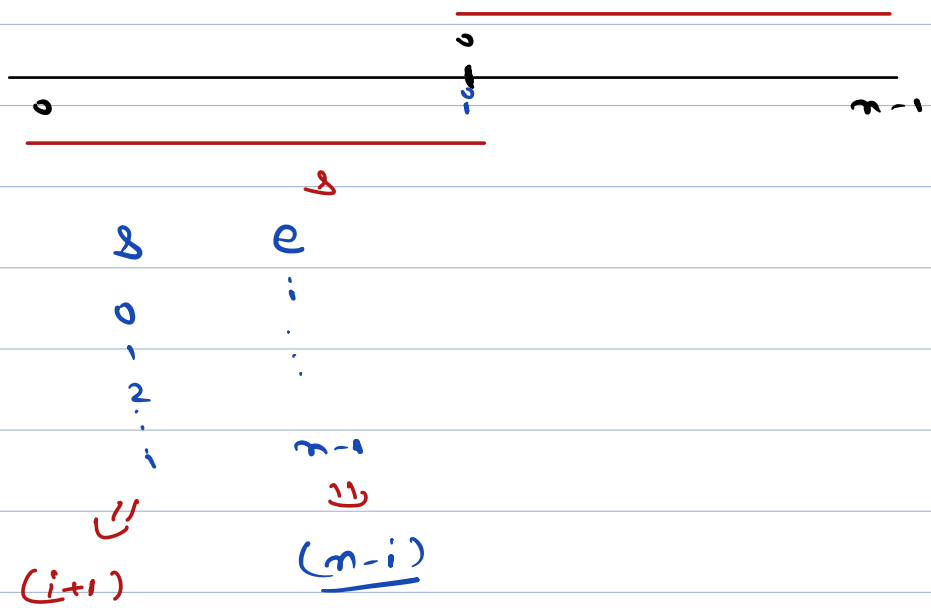
e: 2 3 4 5

⇒ 2 × 5 = 10 subarrays.





Total subarrays = 12.

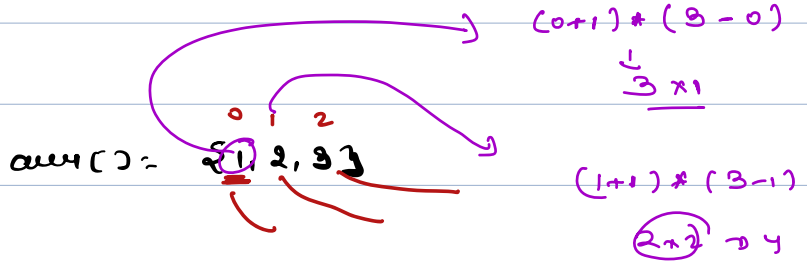


Count of subarrays in which idx i is present => (i+1) * (n-i).

T.C → O(n)

S.C → O(1).

```
for (i = 0; i < n; i++) {
    sum = sum + (arr[i] * [(i+1) * (n-i)]);
}
return sum;
```



Break 8:25 Am - 8:35 Am

arr → N ,

how many subarray of len 1 are there.

4 - 2 + 1

↳ (3)

↳ Subarray → 1 → 4.

↳ Subarray → 2 → 3

↳ Subarray → 3 → 2

↳ Subarray → 4 → 1

array of len N

↳ how many subarrays of len K are there.

arr n → (0 1 2 3 4 . . . n-1)

Subarray len	first subarray start	last sub array starts	no. of sub
1	0	n-1	n
2	0	n-2	n-1
3	0	n-3	n-2
K	0	n-K	n-K+1

In an array of len n, no. of subarrays of len K → n-K+1.

Given N=7, K=4, what will be the total number of subarrays of len K?

↳ $n-k+1$

$\Rightarrow 7-4+1 \Rightarrow 4.$

a_0 a_1 a_2 a_3 a_4 a_5 a_6 a_7

Given an array of size N, print start and end indices of subarrays of length K.

Ex. N = 8, K = 3 $(n-k+1)$

$(8-3+1) \Rightarrow 6.$

start	end
0	2
1	3
2	4
3	5
4	6
5	7

[i end] = K

end - i + 1 = k

end = k + i - 1.

n, K → n-k+1.

```
for (i=0; i < 5; i++)
    int j = k+i-1
    Print (i + " " + j );
```

3

i	j
0	3+0-1 ⇒ 2
1	3+1-1 ⇒ 3

N = 8, K = 3

Ques.

Given an array of N elements. Print maximum subarray sum for subarrays with length = K.

N = 10, K = 5

no. of subarray n-k+1 ⇒ 10-5+1 ⇒ 6

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

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

Ans = 16.

Brute force :-

```
ans = -∞

// first window
i = 0;
j = k-1

while ( j < n ) {
    sum = 0;
    for (idx = i; idx <= j; idx++) {
        sum += arr[idx];
    }
    ans = Max(ans, sum);
    i++;
    j++;
}

return ans;
```

i	j
0	k-1 → n.
1	k
2	k+1

in worst scenario.

T.C -> O((n-k+1)*k)

when, $k = \frac{n}{2}$

$\left(n - \frac{n}{2} + 1\right) * \frac{n}{2} \Rightarrow O(n^2)$.

$k = n$

$(n-n+1)*n \Rightarrow O(n)$.

idea prefix sum :-

// create pf array ;

ans = -∞

// first window

i = 0 ;

j = k-1

while (j < n) {

sum = 0 ;

if (i != 0) { sum = pf[j] - pf[i-1] }

else { sum = pf[j] }

ans = Max (ans, sum) ;

i++ ;

T.C -> O(n) ;

S.C -> O(n) ;

}
}

return ans;

← optimize → (sliding window)

k = 5.

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

s	e	sum
0	4	7
1	5	7 + arr[5] - arr[0] => 8
2	6	8 + arr[6] - arr[1] => 12
3	7	12 + arr[7] - arr[2] => 16
4	8	16 + arr[8] - arr[3] => 10
5	9	12 + arr[9] - arr[4] => 13

n → k → n - k + 1.

```
ans = -∞;
i = 0;
j = k-1;
sum = 0;
for (idx = i; idx <= j; idx++) {
    sum += arr[idx];
}
ans = max(ans, sum);
i++;
j++;
while (j < n) {
    sum += arr[j] - arr[i-1];
    ans = max(ans, sum);
    i++;
    j++;
}
print(ans);
```

k

sum: 0 … k-1

i : 0, j : k-1

j → (k to n-1)

n-1-k+1 => n-k iterations

Total iterations = n.

T.C → O(n);

S.C → O(1).

Observations

Following are the observations that can be useful when solving problems related to subarrays:

- Subarrays can be visualized as contiguous part of an array, where the starting and ending indices determine the subarray.
- The total number of subarrays in an array of length n is n*(n+1)/2.
- To print all possible subarrays, O(n^3) time complexity is required.
- The sum of all subarrays can be computed in O(n^2) time complexity and O(1) space complexity by using Carry Forward technique.
- The sum of all subarrays can be computed in O(n^2) time complexity and O(n) space complexity using the prefix sum technique.
- The number of subarrays containing a particular element arr[i] can be computed in O(n) time complexity and O(1) space complexity using the formula (i+1)*(n-i). This method is called `Contribution Technique`.

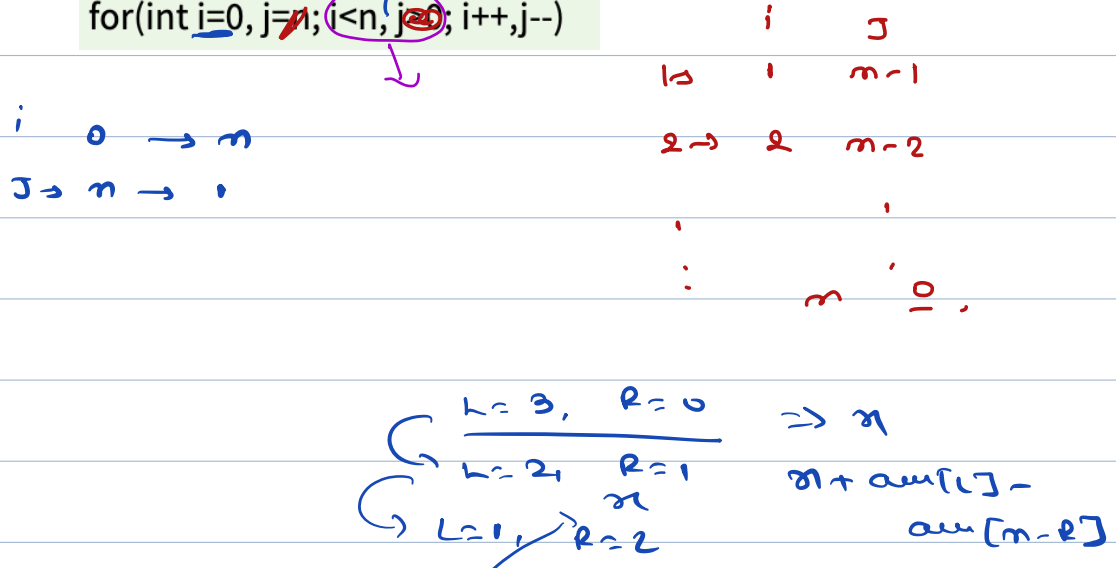
2d Matrices

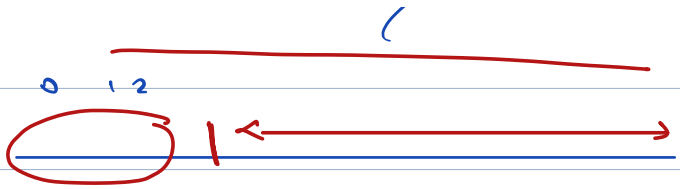
Break

9:36 - 9:40 am

Observations the

```
for(int i=0, j=n; i<n, j>=0; i++,j--)
```





B = 3

L = 3, R = 0

L = 2, R = 1

L = 1, R = 2

L = 0, R = 3

L = R, R = 0

L = ~~3~~ 2 1

R = ~~0~~ 1 2

Sum = X

6