Monday 7am → Problem solving session

- Problems given in assignments / homeworks which are least solved by students
- Idea & Pseudocode discussed
- optional, Attendance not counted, 2 → 3 hour s
- Recorded

Subarray: → Continous part of a array → Subarray

→ Single element / Complete array → Subarray

→ Empty [] is not subarray

→ i__j → length : j-i+1

Properties of Subarrays:

	0	1	2	3
ar[4] :	2	6	3	9

#count : 4 + 3 + 2 + 1 ⇒ 10

Subarrays: subarray

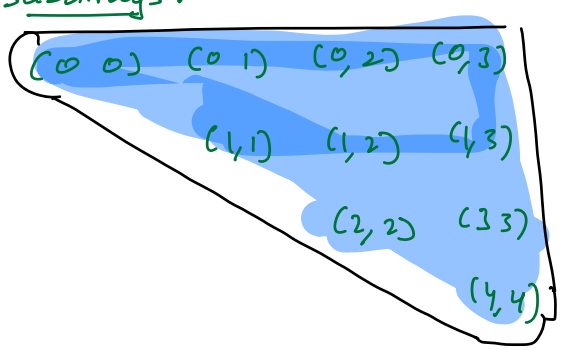
[0 0] - {2} ┐
[0 1] - {2 6} │
[0 2] - {2 6 3} │ 4
[0 3] - {2 6 3 9} ┘

[1 1] - {6} ┐
[1 2] - {6 3} │ 3
[1 3] - {6 3 9} ┘

[2 2] - {3} ┐ 2
[2 3] - {3 9} ┘

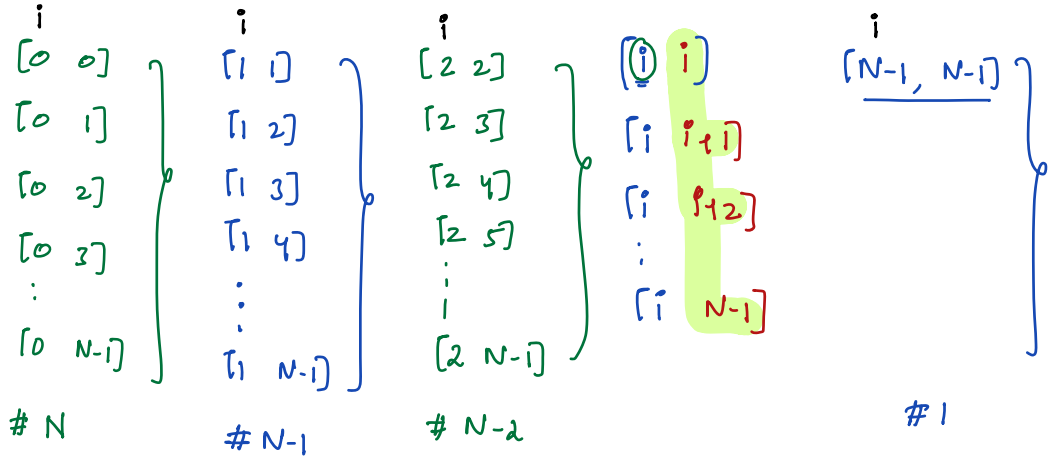
[3 3] - {9}] 1

→ Subarrays:



ar[N] : a_0 a_1 a_2 a_3 .. a_i a_{i+1} .. a_{N-1}

Subarrays :



Total Subarrays = N + N-1 + N-2 + .. 1 : Sum of N Natural Numbers

No. of Subarrays = $\frac{(N)(N+1)}{2}$

Problems 1@

1) Given ar[N], [s e] print subarray from [s e] s <= e (s: startIndex, e: endIndex)

```
void printSubarray(int ar[], int s, int e){
    for(int i = s; i <= e; i++){
        print(ar[i])
    }
}
```

TC → O(N)

SC → O(1)

2Q] Given N array elements print each & every subarray.

ar[4] : { 6 8 -1 7 } (0 1 2 3)

ar[3] : { 4 2 7 } (0 1 2)

Subarrays → printing every subarray

[0 0]	6
[0 1]	6 8
[0 2]	6 8 -1
[0 3]	6 8 -1 7
[1 1]	8
[1 2]	8 -1
[1 3] →	8 -1 7
[2 2]	-1
[2 3]	-1 7
[3 3]	7

Subarray → printing every subarray

[0 0]	{4}
[0 1]	{4 2}
[0 2]	{4 2 7}
[1 1]	{2}
[1 2]	{2 7}
[2 2]	{7}

// Given N elements $O(N^2) * O(N) \Rightarrow O(N^3)$ SC: $\Rightarrow O(1)$

// Pseudocode :

```
void printall(int ar[]){
    int n = ar.length;
    for(i=0; i<n; i++){ // i is indicating start of subarray
        for(j=i; j<n; j++){ // j is indicating end of subarray
            [i j] // we need to print subarray
            for(k=i; k<=j; k++){ print(ar[k]) }
            print(newline)
        }
    }
}
```

↳ given input, we won't consider in Space Complexity

28) Given N array elements print each subarray sum

ar[4] : { 6 8 -1 7 } (0 1 2 3)

subarray	sum
[0 0]	6
[0 1]	14
[0 2]	13
[0 3]	20
[1 1]	8
[1 2]	7
[1 3]	14
[2 2]	-1
[2 3]	6
[3 3]	7

Idea: For every subarray, iterate & print sum

```
void printSum(int ar[]){ TC → O(N3) SC → O(1)
    int n = ar.length;
    for(i=0; i<n; i++){
        for(j=i; j<n; j++){
            [s i  e j] // we need to print subarray sum
            sum=0
            for(k=i; k<=j; k++){ sum=sum+ar[k] }
            print(sum)
        }
    }
}
```

Idea2: For every subarray get it's sum using Pf[]

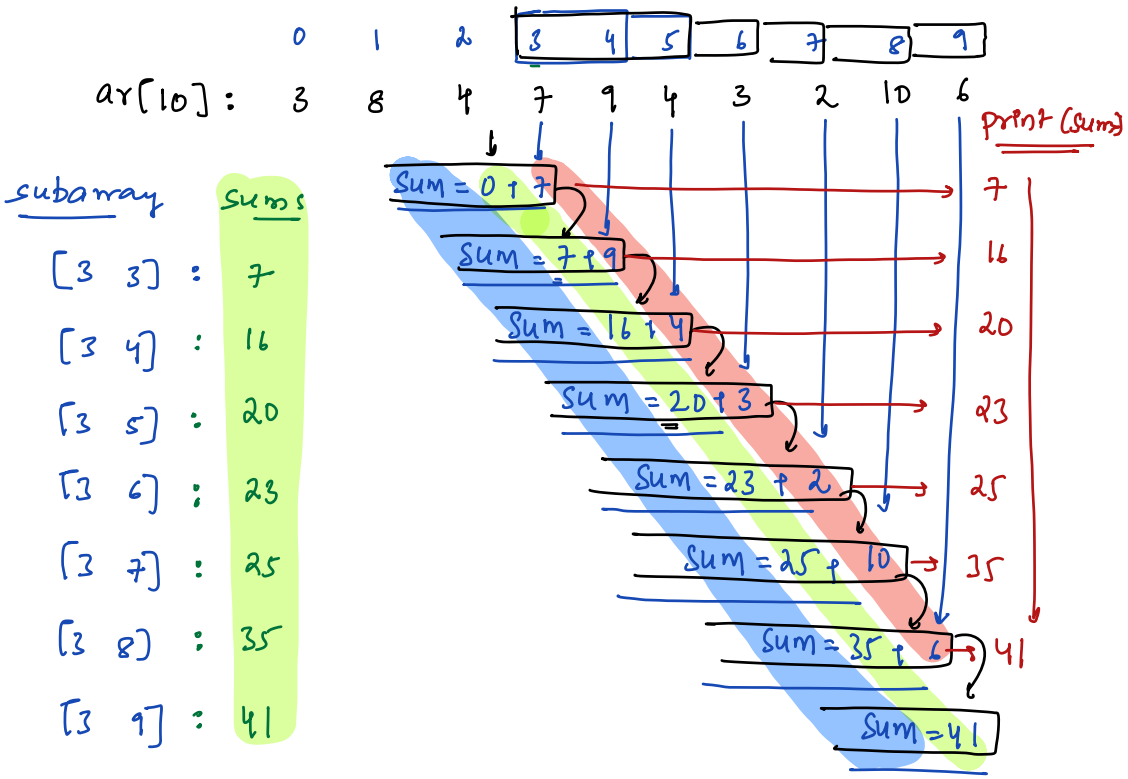
```
void printSum(int ar[]){ TC → O(N+N2)   SC → O(N)
    int n = ar.length;
    int pf[N] → TODO
    for(i=0; i<n; i++){
        for(j=i; j<n; j++){
            [s i  e j] → we need to get sum
            if(i==0){ print(pf[j]) }
            else{ print(pf[j]-pf[i-1]) }
        }
    }
}
```

TC → O(N2)

optimizing space

1) Given input[] we cannot modify

Q) Given ar[N] print all **Subarray Sums starting at index 3**



subarray Sums

[3 3] : 7
[3 4] : 16
[3 5] : 20
[3 6] : 23
[3 7] : 25
[3 8] : 35
[3 9] : 41

```
void PrintSums3(int ar[]) { TC → O(N) SC → O(1)
    int n = ar.length;
    int sum = 0
    i = 3; i < n; i++){
        sum = sum + ar[i]
        print(sum)
    }
}
```

// Printing all Subarray Sums using carry forward

```
void printSums(int ar[]) {   TC → O(N²)  SC: O(1)
    int n = ar.length;
    for(int i=0; i<n; i++){
        int sum = 0
        for(int j=i; j<n; j++){ → printing all subarray sums starting at ith index
            sum = sum + ar[j]
            print(sum) → is representing sum value of subarray
                         ↳ sum[i j]
        }
    }
}
```

i : 0 → print all subarray sums starting at index 0
i : 1 → print all subarray sums starting at index 1
i : 2 → print all subarray sums starting at index 2
↓
i : N-1 → print all subarray sums starting at index N-1

} All Subarray Sums

10:02 pm → 10:10 pm

Q) Given ar[N] elements, return Sum of {All Subarray sums}

ar[4] : { 6 8 -1 7 } (0 1 2 3)

Subarray	Subarraysum
[0 0]	6
[0 1]	14
[0 2]	13
[0 3]	20
[1 1]	8
[1 2]	7
[1 3]	14
[2 2]	-1
[2 3]	6
[3 3]	7

Sum of all subarray sums = 94

ar[3] : { 4 3 7 } (0 1 2) =>

Subarrays	subarray sum
[0 0]	4
[0 1]	7
[0 2]	14
[1 1]	3
[1 2]	10
[2 2]	7

Sum of all Subarray sums = 45 → only a single element, Sum of all Subarray sums

Idea: For every subarray get sum & add it to total sum.

Approach 1
- 3 Nested loops
- O(N^3) SC: O(1)

Approach 2
prefix Sum
O(N^2) SC: O(N)

Approach
Carry forward
O(N^2) SC: O(1)

Using 3rd Approach to get sum of all Subarray sums

```
int TotalSum(int ar[N]) { TC → O(N²)  SC: O(1)
    int n = ar.length;
    int total = 0
    i = 0; i < n; i++) {
        int sum = 0
        j = i; j < n; j++) {   → printing all subarray sums starting at ith index
            sum = sum + ar[j]
            total = total + sum
        }
    }
    return total
}
```

Each sum is indicating subarray sum from [i - j]

version1: *

```
int n = ar.length;
int total = 0
i = 0; i < n; i++) {
    j = i; j < n; j++) {
        total = total + ar[j]
    }
}
return total
```

version2 *

```
int n = ar.length;
int total = 0
i = 0; i < n; i++) {
    int sum = 0
    j = i; j < n; j++) {
        sum = sum + ar[j]
    }
    total = total + sum
}
return total
```

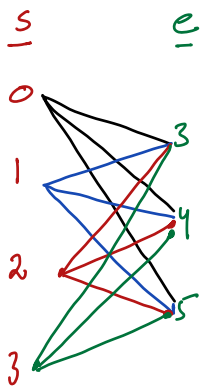
Try → ar[4] → {6 8 -1 7}

Correct output: 94 — version1 — version2

Final Solution :

	0	1	2	3	4	5
ar[6] :	3	-2	4	-1	2	6

In how many subarrays index 3 present ?



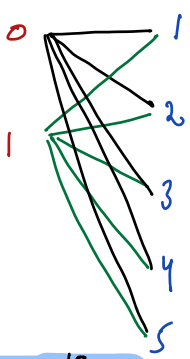
subarrays

In 12 subarray index 3 is present

4 * 3

	0	1	2	3	4	5
ar[6] :	3	-2	4	-1	2	6

In how many subarrays index 1 present ?



subarrays :

2 * 5 = 10

Generalize :

Given N elements, # Subarray i^{th} index present $\Rightarrow (i+1)(N-i)$

ar[N] : a_0 a_1 a_2 a_3 .. a_{i-1} a_i a_{i+1} a_{i+2} .. a_{N-1}

s : [0 i] $\Rightarrow$ # (i+1)

a b $\Rightarrow$ b-a+1

= i-0+1

= i+1

e : [i, N-1] $\Rightarrow$ # N-i

a b $\Rightarrow$ b-a+1

$\Rightarrow$ N-y-i+y

$\Rightarrow$ N-i

N=4 | i | 0 | 1 | 2 | 3 |

// ar[4] : 6 8 -1 7

i+1	:	1	2	3	4
N-i	:	4	3	2	1
Total	:	4	6	6	4

Sum of all subarray sums

Individual Contribution : 24 + 48 + -6 + 28 ⇒ 94

N=3 i= 0 1 2

ar[3]	:	4	3	7
i+1	:	1	2	3
N-i	:	3	2	1
Total	:	3	4	3

Sum : 12 + 12 + 21

: 45

How many ar[0] is coming

ar[] : 4 3 7

Subarrays :

[0-0] : {4} ,

[0-1] : {4 3} ,

[0-2] : {4 3 7} ,

[1-1] : {3} ,

[1-2] : {3 7} ,

[2-2] : {7} ,

4*3 + 3*4 + 7*3

Pseudocode :

```
int TotalSum (int ar[]) {   TC ⇒ O(N)  SC : O(1)
    int n = ar.length
    int sum = 0
    i=0; i< N; i++) {
        // how many times ith element present in subarray
        total = (i+1)(N-i)
        con = total * ar[i]
        sum = sum + con
    }
    return sum
}
```

Technique name : Adding contribution of each ar[i] in final ans, Contribution Technique