Agenda:

1. Sum of all subarray sums.
2. Number of subarrays of length k.
3. Max subarray sum of subarray of length k.
4. Observations.

Question:

* Google *Facebook

Given an array of integers,
find the sum of all possible sub-array sums.

		0	1	2	
Eg:	A :	2	5	3	ans: 35

List out all sub-arrays.

Subarrays:	Sum
2	2
2 5	7
2 5 3	10
5	5
5 3	8
3	3
	35

Brute Force Approach	1. For each sub-array, find the sum 2. Add all sub-array sums.

Code:

A :

PSA : [0 . . . . N-1]

i, j

PSA[j] - PSA[i-1]

```
int  sumOfSAsums(int[] A)
{  int  sum = 0;
      for(i=0 ; i< A.length ; i++)        // Fix  start
      {
         for(j=i; j<A.length; j++)        // Fix  end
         {  int subarraysum = 0;
            for(k=i ; k<=j ; k++)
            {  subarraysum += A[k];
            }

            sum += subarraysum;
         }
      }
      return  sum;
}
```

TC : $O(N^3)$

SC : $O(1)$

Optimization 1:

1. Use prefix sum approach to eliminate innermost loop.

   a. Calculate Prefix Array.   TC: $O(N)$
   SC: $O(N)$

HW

TC : $O(N^2)$

SC : $O(N)$

Optimization 2:

Eg: A :

	0	1	2
A :	2	5	3

Consider sub-arrays beginning at index 0 :

		Sum	
(0, 0)	2	2	A[0]
(0, 1)	2 5	2 + 5	A[0] + A[1]
(0, 2)	2 5 3	2 + 5 + 3	A[0] + A[1] + A[2]

```
int total = 0;
for (j = 0 ; j < N ; j++)
{
   total += A [j];
}
return total
```

TC : $O(N^2)$

SC : $O(1)$

```
int sumOfSAsums (int[] A)
{  int total = 0 ;
   for (i=0 ; i< A.length ; i++)
   {  int sum = 0;
      for ( j = i ; j< A.length ; j++)
      {  sum += A[j];
         total += sum;
      }
   }
   return total;
}
```

Contribution Technique:

	0	1	2
A :	2	5	3

Subarrays:	Sum
2	2
2 5	2 + 5
2 5 3	2 + 5 + 3
5	5
5 3	5 + 3
3	3

$2 \times x + 5 \times y + 3 * z$

$$\boxed{\text{Contribution of ith element} = A[i] * \text{\# of subarrays ith element is a part of.}}$$

1. In how many sub-arrays the element at index 1 will be present?

		i			
	0	1	2	3	4
A :	5	8	3	2	6

Conditions for ith element to be a part of a subarray:

1. $(0 \rightarrow i) \rightarrow \underline{\underline{i+1}}$
2. $(i \rightarrow N-1) \rightarrow \underline{\underline{N-i}}$ $\qquad (N-1)-i+1 = N-i$

$$\begin{array}{ccccccccccc} 0 & 1 & 2 & 3 & \ldots\ldots & i-1 & i & i+1 & \ldots\ldots & N-1 \\ \_ & \_ & \_ & \_ & \ldots\ldots & \_ & \_ & \_ & \ldots\ldots & \_ \end{array}$$

Start Indices	End Indices
3	6
4	7
2	8
	9

Total no. of sub-arrays for m possible start indices and n possible end indices = m x n

Total no. of sub-arrays for (i+1) possible start indices and (N-i) possible end indices = (i+1) * (N-i)

No. of subarrays the ith element will be a part of
(i+1) * (N-i)

Code:

```
int   findSumOfAllSubArraySums (int[] A)
{ int total = 0;
  for (i=0; i< A.length ; i++)
  { int contrib = A[i] * [(i+1)*(A.length - i)]
     total += contrib;
  }
  return total;
}
```

```
      i
0  1  2
[2, 5, 3]
```

TC: O(N)

SC: O(1)

2 x (1 * 3) = 6

5 x (2 * 2) = 20

3 * (3 * 1) = 9

total : ~~0~~ ~~6~~ ~~26~~ 35

Break till 8:15 AM

Total Number Of Subarrays Of Length k:

$$\begin{array}{cccccc} & & & n-k & & \\ & 0 & 1 & 2 & 3 & 4 \\ A: & 3 & 5 & 2 & 1 & 0 \end{array}$$

k = 3 →

k = 2 →

k = 1 →

k = 5 →

N = 5	K = 3	3
N = 5	K = 2	4
N = 5	K = 1	5
N = 5	K = 5	1

$$\underline{\underline{N - k + 1}}$$

Problem: Given an array of size N
print start and end indices of subarrays
of length k

$$\begin{array}{cccccc} & 0 & 1 & 2 & 3 & 4 \\ A: & 3 & 5 & 2 & 1 & 0 \end{array}$$

k = 3

```
for ( i=0 ; i < N-k+1 ; i++)
{
    print ("Start :" + i + " " + "End :" + i+k-1);
}
```

TC: O(N)

Problem: Given an array of N elements find the maximum subarray sum for all subarrays of length k.

	0	1	2	3	4	5	6	7	8	9
A:	-3	4	-2	5	3	-2	8	2	-1	4

k = 5

N = 10

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

Solution 1: Brute Force → For every subarray of length k iterate and find sum

Get max of all such sums.

TC $O(N^2)$

SC $O(1)$

Sliding Window

	0	1	2	3	4	5	6	7	8	9
A:	-3	4	-2	5	3	-2	8	2	-1	4

max: 16

s	e	A[s-1]	A[e]	Sum
0	4			7
1	5	-3	-2	8
2	6	4	8	12
3	7	-2	2	16
4	8	5	-1	10
5	9	3	4	11

Code:

```
int   maxSubArraySumLenK( int[] A, int k)
{  int  max = Int_Min;   int sum = 0;
   for(i=0 ; i<k;i++)
   {
      sum += A[i];                                      Math.max(  )
   }
   max = Max(sum, max);
   int i=1;  j= i+k-1;
   while (j< A.length)
   {
     sum = sum - A[i-1] +A[j]
     max = Max(max, sum);
      i++;  j++;
   }
   return max;
}
```

Observations:

1. Subarray is a contiguous part of an array.
2. Subarray can be uniquely represented by a start index s and an end index e.
3. T.C. to print all subarrays of an array → $O(N^3)$
4. Sum of all sub-arrays can be calculated in TC $O(N^2)$ by using carry forward
5. Using contribution technique, TC can be reduced to $O(N)$.

Next Class

2D Matrices deep dive.

- Used in solving problems in Math, Computer Graphics, ML
- Easier to scale images, solving equations, data analysis
- Game Development.

Doubts:

$s \to i \qquad l \to k$

$e \to x$

$x - i + 1 = k$

$\underline{\underline{x = i + k - 1}}$

	0	1	2	3
A:	2	5	3	8

k=2

```
int   maxSubArraySumLenK( int[] A, int k)
{  int max = Int_Min;   int sum = 0;
   for (i=0 ; i<k;i++)
   {
      sum += A[i];
   }
   max = Max(sum, max);
   int i=1 ;  j = i+k-1;
   while (j < A.length)
   {
      sum = sum - A[i-1] +A[j]
      max = Max(max, sum);
      i++; j++;
   }
   return max;
}
```

max=11

i	j	A[i-1]	A[j]	sum
0	1			7
1	2	2	3	8
2	3	5	8	11
3	4			

Equilibrium Index :

	0	1	2	3	4	5	6
A:	-7	1	5	2	-4	3	0