Q Given an array of integers, we need to find the sum of each possible subarrays of the array.

A = [1, 2, 3]

[1] = 1
[1, 2] = 3
[1, 2, 3] = 6
[2] = 2
[2, 3] = 5
[3] = 3

Bruteforce: Explore all possible subarrays with 3 loops.

First two loops => For considering start & end

Third loop => To iterate and find sum

```
void printSubarraySum(int arr[], int n){

    for(int i=0; i<N; i++){
        for(int j=i; j<N; j++){
            int sum=0;
            for(int k=i, k<=j; k++){
                sum += arr[k];
            }
            print(sum);
        }
    }
```

TC: $O(N^3)$ SC: $O(1)$

Use Prefix Sum: 1) First create the prefix sum.

2) Use 2 loops to consider all subarrays and then use prefix sum array to find sum.

```
void printSubarraySum(int arr[], int n){
    // TODO Create a prefix array.

    for(int i=0; i<N; i++){
        for(int j=i; j<N; j++){
            int sum = 0;
            if(i==0) sum = prefix[j]
            else sum = prefix[j] - prefix[i-1]

            print(sum);
        }
    }
}
```

TC: $O(N^2)$ SC: $O(N)$

Use given array to convert to prefix array.

TC: $O(N^2)$ SC: $O(1)$

Can we do TC: $O(N^2)$, SC: $O(1)$ without modifying given array:

$A = [-4, 1, 3, 2]$

$i = 0$

$j = 0$

$K = Sum = A[0]$

$j = 1$

$K = Sum = A[0] + A[1]$

$j = 2$

$K = Sum = A[0] + A[1] + A[2]$

$j = 3$

$K = Sum = A[0] + A[1] + A[2] + A[3]$

Obs: Everytime j moves, we can just add arr[j] to the sum.

$A = [-4, 1, 3, 2]$

$i = 0$

curSum = 0

j	Index	curSum	=>	value
0	=> [0 0]	+A[0]		-4
1	=> [0 1]	+A[1]		-3
2	=> [0 2]	+A[2]		0
3	=> [0 3]	+A[3]		2

i = 1
CurSum = 0

j	Index	curr Sum	=>	value
1	[1 1]	+A[1]		1
2	[1 2]	+A[2]		4
3	[1 3]	+A[3]		6

i = 2
CurSum = 0

j	Index	curr Sum	=>	value
2	[2 2]	+A[2]		3
3	[2 3]	+A[3]		5

i = 3
CurSum = 0

j	Index	curr Sum	=>	value
3	[3 3]	+A[3]		2

```
void printSubarraySum(int arr[], int n){

    for(int i=0; i<N; i++){
        int curSum = 0;
        for(int j=i; j<N; j++){
            curSum += A[j];
            print(curSum)
        }
    }
}
```

Carry forward the sum.

TC: $O(N^2)$

SC: $O(1)$

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

Google, Facebook.

[1 2 3] ⇒ 1 + 3 + 6 + 2 + 5 + 3

= 20

Use the Carry Forward approach to print sum of each subarray sum.

```
void printAllSubarraySum(int arr[], int n){
    int ans = 0
    for(int i=0; i<N; i++){
        int curSum = 0;
        for(int j=i; j<N; j++){
            curSum += A[j]
            ans += curSum
        }
    }
    print(ans)
}
```

TC: $O(N^2)$ SC: $O(1)$

[1 2 3]

i	j	currSum	ans
0	0	0+1	1
0	1	1+2	1+3
0	2	3+3	1+3+6
1	1	0+2	

Vikas's approach

only sum with No reset

1

1 + 2 = 3

Does not work.

Contribution Technique

[1 2 3]

[1] = 1

[1 2] = 3

[1 2 3] = 6

[2] = 2

[2 3] = 5

[3] = 3

Instead of going to each subarray we need think of an alternative:

→ Think about contribution each element in final sum

1 is added by [1] [1 2] [1 2 3]

2 is added by [1 2] [1 2 3] [2] [2 3]

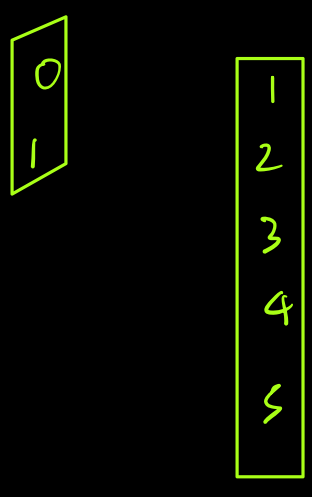
3 is added by [1 2 3] [2 3] [3]

ans = $1*3 + 2*4 + 3*3$

$= 3 + 8 + 9 = 20$

Quiz: A = [3 -2 4 -1 2 6]

How many subarrays an index 1 will be present?

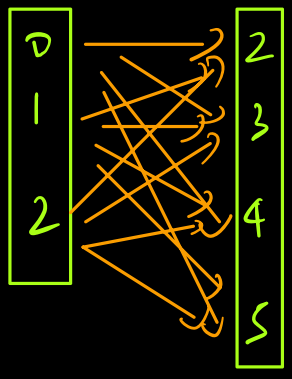


$\Rightarrow$

0,1 => [3 -2]
0 2 => [3 -2 4]
0 3 => [3 -2 4 -1]
0 4 => [3 -2 4 -1 2]
0 5 => [3 -2 4 -1 2 6]
1 1 => [-2]
1 2 => [-2 4]
1 3 => [-2 4 -1]
1 4 => [-2 4 -1 2]
1 5 => [-2 4 -1 2 6]

Quiz: A = [3 -2 [4] -1 2 6]

How many subarrays an index 2 will be present?



Total count = 3 * 4 = 12

Generalised Calculation

[0, i] ⇓ (i+1)

[i N-1] ⇓ (N-1 - i + 1) ⇓ (N - i)

Total no. of subarray containing i^{th} index

= (i+1)(N-i)

Contribution of i^{th} index = A[i] × (i+1)(N-i)

```
int findTotalSubarraySum(int arr[], int n){
    int ans = 0
    for(i=0; i<N; i++){
        ans += arr[i] * (i+1) * (N-i)
    }
    return ans;
}
```

TC: O(N) SC: O(1)

[1 2 3]

i	No. of Subarrays part of	Contribution	ans
0	(0+1)*(3-0) = 3	1 * 3 = 3	0 + 3
1	(1+1)*(3-1) = 4	2 * 4 = 8	3 + 8
2	(2+1)(3-2) = 3	3 * 3 = 9	11 + 9 = 20

Q Number of subarray of length = K.

= Number of start indices of subarray of length K

[0 1 2 3 4] N = 5, K = 3

[0 2]
[1 3]
[2 4]

ans = 3

Start point	end point
0	K-1
1	K
2	K+1
3	K+2
⋮	⋮
	N-1

‼

[K-1, N-1]

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

$= N-1-K+1+1$

$= \boxed{N-K+1}$

Q $N = 7$, $K = 4$

$7 - 4 + 1 = 4$

Q Given an array of size N, print start and end of all the subarrays of length K.

$N = 8$ $K = 3$

output:

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

$[i \quad x]$

$x - i + 1 = K$

$x = K + i - 1$

```
for(i=0; i< N-K+1; i++){
    j = K+i-1;
    print(i, j)
}
```

Alternative:

```
i=0, j = K-1
while(j<N){
    print(i, j)
    i++, j++
}
```

Q Given an array of N elements. Print maximum subarray sum with length K.

N = 10, K = 5

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

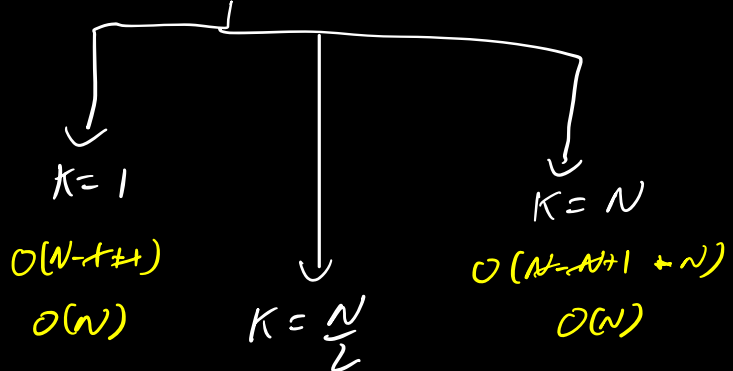
i	e	sum
0	4	7
1	5	8
2	6	12
3	7	16
4	8	10
5	9	11

Max = 16

Brute Force : Consider all subarrays and calculate sum

```
ans = INT_MIN
i = 0, j = K-1
while (j < N) {
    sum = 0
    for (k = i; k <= j; k++)
        sum += arr[k]
    ans = max (ans, sum)
    i++, j++
}
return ans
```

TC: $O((N-K+1) * K)$



$O(N - \frac{N}{2} + 1) * \frac{N}{2})$

$O(\frac{N}{2} * \frac{N}{2}) = O(N^2)$

Use Prefix Array to find Sum:

```
# TODO Create a Prefix Array
ans = INT_MIN
i = 0, j = k-1
while (j < N) {
    if (i == 0) sum = prefix[j]
    else sum = prefix[j] - prefix[i-1]

    ans = max(ans, sum)
    i++, j++
}
return ans
```

TC: O(N - K + 1 + N)

O(N)

SC: O(N)

!!

O(1)

if use input array for prefix

Can we reduce the Space complexity without modifying input array?

Observation: For going to every new window
- → subtract first index of range
- → add next index of range

-3	4	-2	5	3	-2	8	2	-1	4	K=5

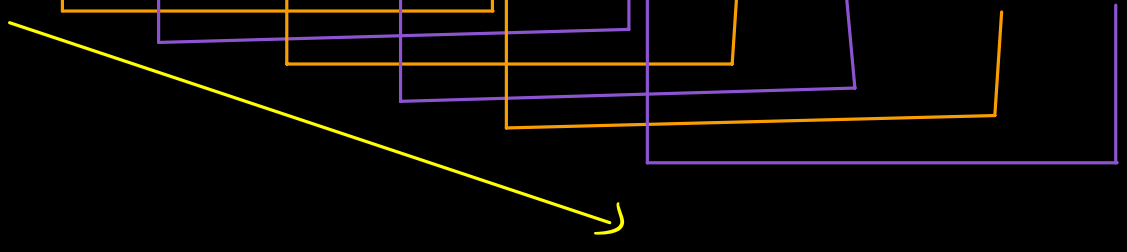
First Subarray

-3 4 -2 5 3

Second Subarray

-3 4 -2 5 3 -2

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



Sliding Window

i	j	sum
0	4	7
1	5	sum - A[0] + A[5] = 7 - (-3) + (-2) = 8
2	6	sum - A[1] + A[6] = 8 - (4) + 8 = 12
3	7	sum - A[2] + A[7] = 12 - (-2) + 2 = 16
4	8	sum - A[3] + A[8] = 16 - 5 + (-1) = 10
5	9	sum - A[4] + A[9] = 10 - 3 + 4 = 11

ans = INT_MIN

i = 0, j = K-1

// Sum of 1st Window

window_sum = 0

for(int l = i; l <= j; l++) {

    window_sum += A[l]

}

ans = max(ans, window_sum)

} => K

i++, j++

while(j < N) {

    window_sum += A[j] - A[i-1]

    ans = max(ans, window_sum)

    i++, j++

} => N-K

return ans

TC: O(K + N - K) = O(N)

SC: O(1)

Observation Summary :

-) Subarray is contiguous part of array where starting and ending point can uniquly indentify the subarray

-) Total no. of Subarray = $\frac{N * (N+1)}{2}$

-) To print all Subarray, best TC: $O(N^3)$

-) To print sum of each subarray
Best Technique is Carry Forward
TC: $O(N^2)$ SC: $O(1)$

-) To print sum of each subarray
with Prefix Array
TC: $O(N^2)$ SC: $O(N)$

-) To get sum of sum of all subarrays.
Best is Contribution Technique
TC: $O(N)$ SC: $O(1)$

-) When we have fixed size window and we need to consider all windows
then think about Sliding Window

Next Class:

-> 2D matrices
It helps to organise data in grid structure

-> We use them in many practical applications: Scaling images, doing big mathematical calculations.

-> Databases, Spreadsheet & lot more.

-> Game Development

-> Graphs.

Doubts

[1 2 3]

1 3 6

i	j		
0	0	sum =>	p[0] = 1
0	1	sum =>	p[1] = 3
0	2	sum =>	p[2] = 6
1	1	sum =>	p[1] - p[0] = 2
1	2	sum =>	p[2] - p[0] = 5

2 2 Sum => p[2] - p[1] = 6 - 3 = 3

[1 2 3]

i	j	curSum	ans
0	0	0+1	0
0	1	1+2	0
0	2	3+3	0
			6
1	1	0+2	

→ Sum of subarrays

0 0 = 1
0 1 = 3
0 2 = 6
10

$10^9 + 7$ is a big prime number

⇓

Prime numbers are good at distributing things

int => 2×10^9

% => % m

⇓

x % m => [0 m-1] [0 m-1]

$10^{10000000000}$