Q-1. Given an integer array, find the sum of all possible subarrays

Ex:- [1 2 3]

	Ans
1	1
1 2	3
1 2 3	6
2	2
2 3	5
3	3

Bruteforce :-

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

T.C. - $O(N^3)$. ; S.C $\rightarrow O(1)$.

Idea :- Using Prefix Sum.

Subarray from index i to j

P[i] - P[i-1] P[i]

i ≥ 0 i = 0.

[1 2 3] // Construct P[] (Prefix Sum).

P[1 3 6]. for(i = 0; i < N; i++) {

for(j = i; j < N; j++) {

↗ Ignore if (i == 0) {

T.C. O(N + N²) sum = P[j];

≃ O(N²) } else { sum = P[j]

- P[i-1];}

S.C. O(N). print(sum);

}

}

N × (N+1) / 2.

Idea: Carry Forward.

(calculate + use).

i	j	sum
0	0	1
	1	3
	2	6
1	1	2
	2	5
2.	2.	3

0 1

0 1 2

0 1 2 3

1 2

1 2 3

2 3

```
for( i = 0; i < N; i++) {
    sum = 0;
    for( j = i; j < N; j++) {
        sum += A[j];          ← calculate
    }
    print( sum );             ← use.
}
```

T.C. → $O(N^2)$

S.C → $O(1)$.

3

[1 2 3]
0 1 2

i	j	sum
0	0	0 1 3 6
1	1 2	2 5
2	2	3

6.
5.

Q-2. Find the total sum of all subarray sums.

Contribution Technique..

Ex: [1 2 3]

1	1
1 2	3
1 2 3	6
2	2
2 3	5
3	3

20 → Ans.

```
ans = 0;
for(i = 0; i < N; i++) {
    sum = 0;
    for(j = i; j < N; j++) {
        sum += A[j];
        ans += sum;
    }
}
return ans;
```

T.C → O(N²)
S.C. → O(1).

A = [1 2 3]

1	1
1 2	3
1 2 3	6
2	2
2 3	5
3	3

20

2 3

3

5

3

20

3*1 + 4*2 + 3*3

= 3 + 8 + 9.

= 20.

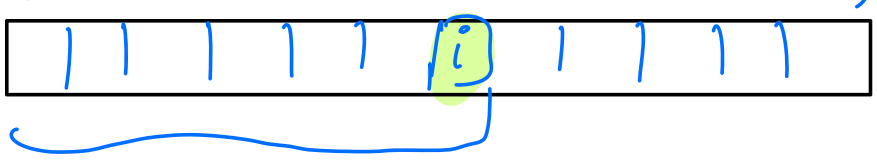
Ans:

$$= \sum_{i=0}^{N-1} A[i] * \text{No. of subarrays in which } A[i] \text{ is present.}$$

0	1	2	3	4	5
3	-2	4	-1	2	6

Ans = 12.

Generalisation



Si

Si

B_1 A_1

B_2 A_2

Total Subarrays.

"3"
B4 G3 are Possible Si
B5 G4. * Possible ei

Si → [0, i]. → (i+1).

ei → [i, N-1] → (N-i)

[a, b] → b-a+1

N-1 - i + 1 = (N-i).

Total Contribution of A[i]

A[i] * (Si * ei)

A[i] * (i+1) * (N-i)

Code:

```
ans = 0;
for (i=0; i<N; i++) {
    ans += A[i] * (i+1) * (N-i);
}
return ans;
```

T.C. → O(N)
S.C. → O(1)

10:40.

Q-3. Total # of subarrays of length K (<= N) = ?

0	1	2	3	4	5
3	-2	4	-1	2	6

K = 4.

Ans = 3:

0	1	2	3	4	5
3	-2	4	-1	2	6

K.	# of Subarrays
1.	6. (N)
2.	5 (N-1)
3	4. (N-2)
4	3. (N-3)
5	2 .
6	1 (N - 5)

(N - K + 1)

N=7 ; K = 4. . → (7 - 4 + 1)

= 4

Q:- 4. Given an integer array, find max subarray sum of subarray with length = K.

0 1 2 3 4 5

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

4.

3.

11 → Ans.

K = 4;

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

(K-1, N-1) → (N-K+1) * (K).

T.C → (N-K+1) * (K).

$K = 1 \cdot \rightarrow (N - K + 1) * (1)$

$= N \cdot$

$K = N \cdot \rightarrow (N - N + 1) * (N)$

$\rightarrow N \cdot$

$K = N/2 \rightarrow (N - N/2 + 1) * (N/2)$

$= \frac{N}{2} * N/2$

$= \frac{N^2}{4} \simeq O(N^2)$

Idea:- Use Prefix Sum.

```
# Create a P[] array.
ans = INT_MIN;
st = 0;  ;  end = K-1;
while (end < N) {
    if (st == 0) { sum = P[end]; }
    else
    { sum = P[end] - P[st-1]; }

    ans = max(ans, sum);
```

```
st++;
end++;
}
return ans;
}
```

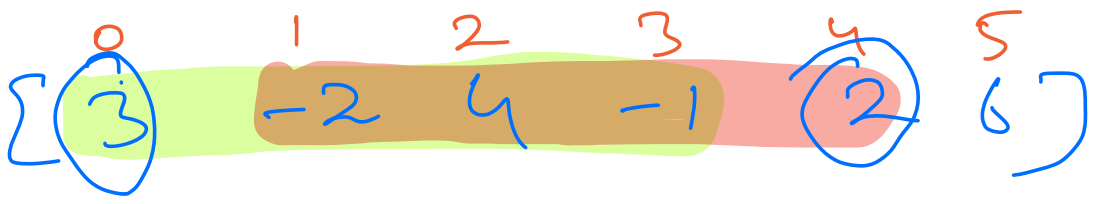
T.C. → O(N) + O(N-K+1)

≃ O(N).

S.C. → O(N)

Idea → Carry Forward. (Sliding Window)

K = 4.




```
sum = 0;
for (i → 0 to (K-1)) {
    sum += A[i];
}
ans = sum;

for (int j = K; j <= N-1; j++) {
    sum += A[j];
    sum -= A[j-K];
    ans = max(ans, sum);
}
```

O(K).

→ global ans

→ ending pont.

// Adding ending pont of new subarray;

└→ // Remove Starting

return ans;

part of previous subarray.

3

[k, N-1] → N-k.

T.C → O(K + N - K) = O(N)

S.C → O(1)

Observations:

(1) Total No. of Subarrays = N(N+1)/2.

(2) Consider all Subarrays.
T.C can not be reduced from O(N²).

(3) T.C. for printing all subarrays
→ O(N3).

(4) Cal. Sum of all Subarrays.

N³ → N² → N²
O(1) O(N) O(1)
↑ ↑ ↑
B.F. Prefix Sum Carry forward

(5) Contribution Technique
↳ Total Sum of all Subarrays.

T.C → O(N)
S.C → O(1).

(6) Sliding Window
↳ When size of subarray is fixed.

function (arr.)

```
func2(){
  arr → _ _
  _ _
  func(arr)
  }
```