Sliding window & contribution Technique

Content

- Print sum of all possible subarrays.
- Total sum of all subarray sums
- Max subarray sum with length K

Q> Given an integer array. Print the sum of all possible subarray {continuous part of array}

A = [1 2 3]

subarrays —

$\frac{n*(n+1)}{2} = \frac{3*4}{2}$

$= 6$

Subarrays	Output.
1 ⟶	1
1 2 ⟶	3
1 2 3 ⟶	6
2 ⟶	2
2 3 ⟶	5
3 ⟶	3

Bruteforce

To represent a subarray, I need (start, end)

```
for i ⟶ 0 to N-1 {
    for j ⟶ i ⟶ N-i {
        total = 0
        for k ⟶ i to j {
            total += A[k]
        }
        print ( total )
    }
}
```

TC : $O(N^3)$

SC : $O(1)$

Prefix Sum

$P[i] = A[0] + A[1] \ldots\ldots + A[i]$

$P[i] = P[i-1] + A[i]$.

Sum over (i, j) ⟶ if (i == 0) P[j]
else P[j] − P[i−1]

A = 1 2 3 (indices 0 1 2; i = 1)

P = 1 3 6 (j = 2)

```
// Create P[] sum array.
for i ⟶ 0 to N-1 {
    for j ⟶ i ⟶ N-1 {
        total = 0
        if (i==0) {total = P[j]}
        else { total = P[j] - P[i-1]

        print ( total )
    }
}
```

TC : O(N²)

SC : O(N)

1 ⟶ 1

1 2 ⟶ 3

1 2 3 ⟶ 6

2 ⟶ 2

2 3 ⟶ 5

3 ⟶ 3

O(1)
By using A[] as prum.

Prefix sum ⟶ left to right

Suffix sum ⟶ right to left.

Carry Forward { calculate + use }.

A = [1 2 3] (0 1 2)

i	j			
0	0	1	1	total = A[0]
0	1	1 2	3	total + A[1] = 3
0	2	1 2 3	6	total + A[2] = 6
1	1	2	2	total = A[1] = 2
1	2	2 3	5	total + A[2] = 5
2	2	3	3	total = A[2] = 3.

```
for i ---> 0 to N-1 {
    total = 0
    for j ---> i ---> N-i {
        total += A[j]      } Calculate
        print(total)       } use
    }
}
```

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

A = [1 2 3] (0 1 2)

i

j

TC : $O(N^2)$

SC : $O(1)$

NOTE : TC cannot be improved better than $O(N^2)$ since there are $\frac{N*(N+1)}{2}$ subarray sum to be printed.

Q) Find the total sum of all subarray sums.

Input

A = 1 2 3

1 → 1

1 2 → 3

1 2 3 → 6

2 → 2

2 3 → 5

3 → 3

Output 20

```
ans = 0
for i ⟶ 0 to N-1 {
    total = 0
    for j ⟶ i ⟶ N-1 {
        total += A[j]
        ans += total
    }
}
print ( ans )
```

TC : $O(N^2)$

SC : $O(1)$

1 → 1

1 2 → 3

1 2 3 → 6

2 → 2

2 3 → 5

3 → 3

Output 20

contribution of 1 = 3 * 1 = 3

contribution of 2 = 4 * 2 = 8

contribution of 3 = 3 * 3 = 9

20

Contribution Technique

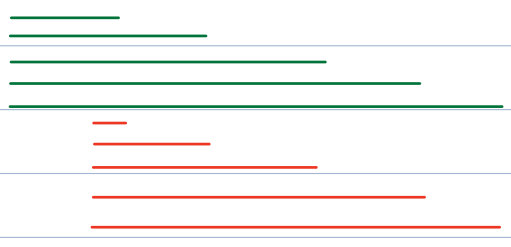
$$\sum_{i=0}^{n-1} (\text{contribution of } A[i])$$

A[i] * # subarrays containing A[i]

How to calculate the no. of subarrays containing A[i]?

subarrays containing A[1]

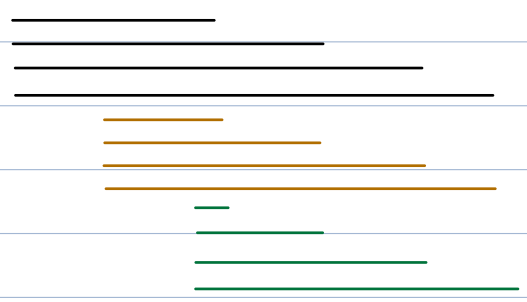
	0	1	2	3	4	5
A =	3	-2	4	-1	2	6



subs = 10

subarrays containing A[2]

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



12

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

choices of my start index = [0-2] = 3
choices of my end index = [2-5] = 4

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

3 4 = 3 * 4 subarrays.

∵ for any index as start, I can choose any of the end index.

A_0 A_1 A_2 .... A_i ..... A_{n-2} A_{n-1}

[0-i] → (i+1)

[i, n-1] → (n-i)

No. of subarrays that will contain A[i] = (i+1) * (n-i)

Contribution of A[i] = A[i] * {(i+1) * (n-i)}

0	1	2
1	2	3

3 * {3 * (3-2)}

3 * 3 = 9

1 * {1 * 3} = 3

2 * {(1+1) * (3-1)}

2 * 4 = 8

Pseudocode

TC: O(N)

SC: O(1)

```
any = 0
for ( i ⟶ 0 to N-1 ) {
|       any += A[i] * (i+1) * (n-i)
}
print (any)
```

21 : 36

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

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

K = 3

Output ⟶ 4

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

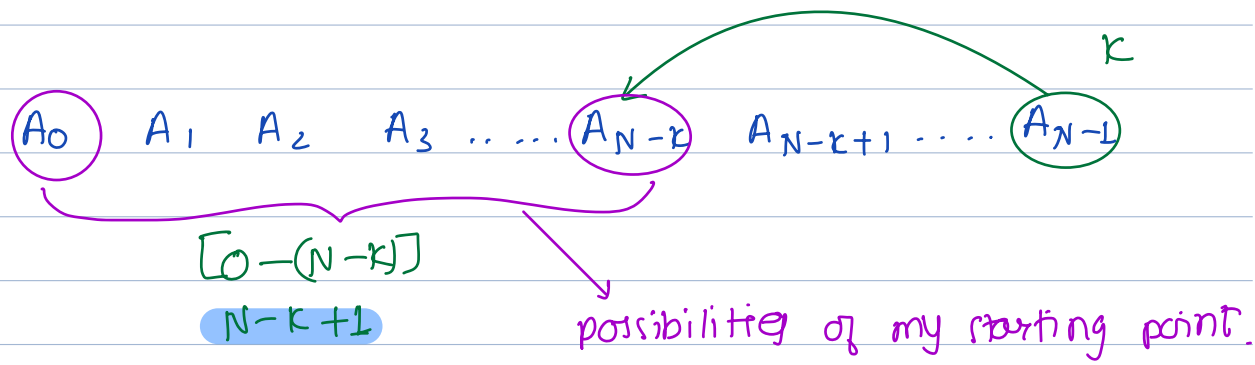
K = 4

[0 - 2] = 3

K	# subarrays.	
1	6	N
2	5	N-1
3	4	N-2
4	3	N-3
5	2	N-4
6	1	⋮

Total no. of subarrays of length == K

N - K + 1

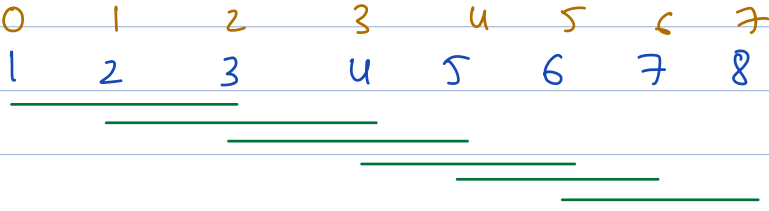


N = 7 K = 4 N - K + 1 = 7 - 4 + 1 = 4.

Given A[], print start and end indices of subarrays of length K.

N = 8 K = 3.

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



Output

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

K = 3

Indices of starting subarray.

{ 0, K-1 }

[L-R] = R-L+1

[0-X] = K

R-0+1 = K

R = K-1

Pseudocode

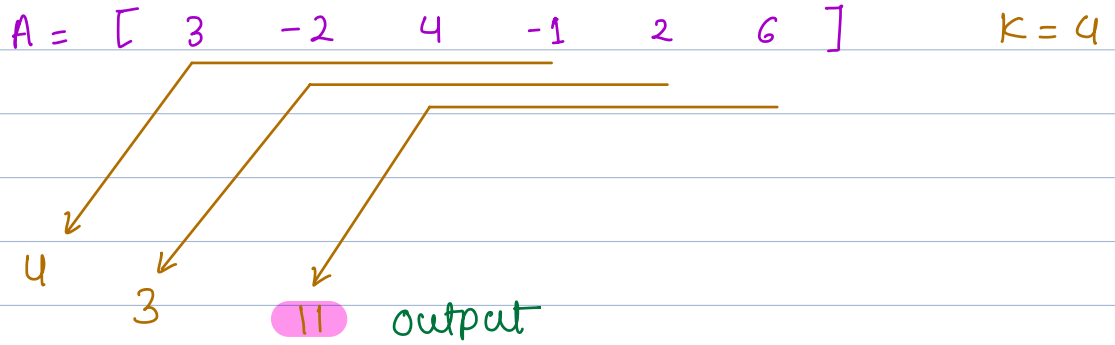
```
start = 0
end = K-1

while (end < N) {
    print (start + " " + end)
    start ++
    end ++
}
```

TC : O(N)

SC : O(1)

Q) Given an integer array. Find max subarray sum of subarray of length = K



Bruteforce

```
start = 0                                   TC: O(N^2)
end = K-1                                   SC: O(1)
mtotal = -∞
while (end < N) {
        total = 0
        for ( i ⟶ start to end ) {
                total += A[i]
        }
        mtotal = max (mtotal , total)
        start ++
        end ++
}
```

Prefix Sum

```
// calculate P[] sum
start = 0
end = K-1
mtotal = -∞
while (end < N) {
        total = 0
        if (start == 0) { total = P[end] }
        else { total = P[end] - P[start-1] }

        mtotal = max (mtotal, total)
        start++
        end++
}
```

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

use A[] as Psum
to reduce SC → O(1)

Observation { sliding window }

A = [3 -2 4 -1 2 6] K = 3
    0 1 2

total = 5

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

total = 5 - 3 + (-1)
      = 2 - 1
      = 1

A technique to skip recalculation of values over a window { fixed, dynamic }

Steps

——→ calculate total for 0 to K-1 {first sub}

——→ Add A[i] and remove A[i-K].

——→ keep maintaining the max total

Pseudocode

```
// Step 1 calculate total for i ⟶ 0 to k-1
total = 0
for ( i ⟶ 0 to k-1 ) {
    total += A[i]
}
mtotal = total
```

TC : O(N)

SC : O(1)

```
for ( i = k ; i < n ; i++ ) {
    remove = A[i-k]
    add = A[i]
    total = total - remove + add
    mtotal = max (mtotal, total)
}
```

total = total - A[i-k] + A[i]

```
print (mtotal)
```

Doubt sesion

mod = $10^9 + 7$

ans .....

ans % = mod .