Arrays : Sliding Window And Contribution

Question :- Given an array of integers, we need to find the sum of possible subarrays of the array & maintain the maximum sum.

Arr[] = [10, 20, 30]

Subarrays :-

	Sum
10	10
10 20	30
10 20 30	60
20	20
20 30	50
30	30

Brute force

↳ we can use the same approach of printing the subarrays

PSEUDO CODE

```
for (s=0 ; s<n ; s++){
    for (e=s ; e<n ; e++){
        int Sum=0;
        for (i=s ; i<=e ; i++){
            Sum += A[i];
        }
        Print (Sum);
    }
}
```

TC - O(N³)

SC - O(1)

↳ OPTIMIZATION

Observation

↳ In the innermost loop, we are finding sum from s to e. And that can be performed optimally using PREFIX SUM

PSEUDO CODE

```
int[] Pf   // TODO
for (s=0;  s<n;  s++){
    for (e=s;  e<n;  e++){
        int sum=0
        if (s!=0) {
            sum = Pf[e]-Pf[s-1];
        } else {
            sum = Pf[e];
        }
        Print(sum);
    }
}
```

TC → $O(N^2)$

SC → $O(N)$ ← compromised

FURTHER OPTIMIZATION [Carry Forward]

PSEUDO CODE

```
for (s=0 ; s<n ; s++){
    for (e=s ; e<n ; e++){
        int Sum=0;
        for (i=s ; i<=e ; i++){
            Sum += A[i];
        }
        Print (Sum);
    }
}
```

DRY RUN

arr = [10, 20, 30]

(a) — (0,0) → 10 → arr[0]

(b) — (0,1) → 10 20 → arr[0] + arr[1]

(0,2) → 10 20 30 → arr[0] + arr[1] + arr[2]

:

Observations

↳ Their is a repetition which can be avoided.

↳ We can compute (b) using (a)

OR

We can user previous Ans & A[i] to get the current Ans.

DRY RUN

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

i → 1, j → 1

(I) i = 0
curr Sum = 0

j	Subarray (i, j)	curr Sum	Value
0	[0, 0]	+ A[0]	-4
1	[0, 1]	+ A[1]	-4 + 1 = -3
2	[0, 2]	+ A[2]	-3 + 3 = 0
3	[0, 3]	+ A[3]	0 + 2 = 2

(II) i = 1
curr Sum = 0

j	Subarray (i, j)	curr Sum	Value
1	[1, 1]	+ A[1]	1
2	[1, 2]	+ A[2]	1 + 3 = 4
3	[1, 3]	+ A[3]	4 + 2 = 6

(III) So on

PSEUDO CODE

```
for ( s = 0; s < n ; s++) {
    curr Sum = 0
    for (e = s; e < n; e++) {
        curr Sum += A[e];
        Print (curr Sum);
    }
}
```

TC - O(N²)

SC - O(1)

⇒ Back to original question, we need max

PSEUDO CODE

```
int max = -∞
for (s = 0; s < n; s++){
    currSum = 0
    for (e = s; e < n; e++){
        currSum += A[e];
        max = Math.max(max, currSum);
    }
}
Print (max);
```

TC - O(N²)

SC - O(1)

[Google / Facebook]

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

Ex:- [10, 20, 30]

s	e	Subarray	Sum
0	0	10	10
0	1	10 20	30
0	2	10 20 30	60
1	1	20	20
1	2	20 30	50
2	2	30	30

Total 200 Ans

I st Way [Brute force]

↳ We can use carry forward to get sum of all the subs & simultaneously add them

```
Sum = 0
for (s = 0; s < n; s++){
    curr Sum = 0
    for (e = s; e < n; e++){
        curr Sum += A[e];
        Sum += Curr Sum
    }
}
Print (Sum)
```

TC - O(N²)
SC - O(1)

↳ II nd Way [optimization] / Contribution Technique

Ex:- [10, 20, 30]

S	e	Subarray	Sum
0	0	10	10
0	1	10 20	30
0	2	10 20 30	60
1	1	20	20
1	2	20 30	50
2	2	30	30
		Total	200 Ans

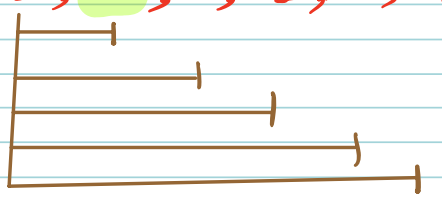
So, Sum = $10 \times 3 + 20 \times 4 + 30 \times 3$

$= 30 + 80 + 90$

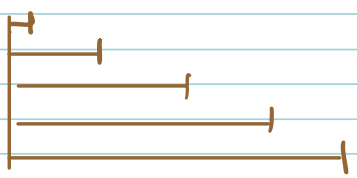
$= 200$ Ans

Quiz 1 :- In how many subarrays, the element at index 1 will be present?

A : [3, -2, 4, -1, 2, 6]

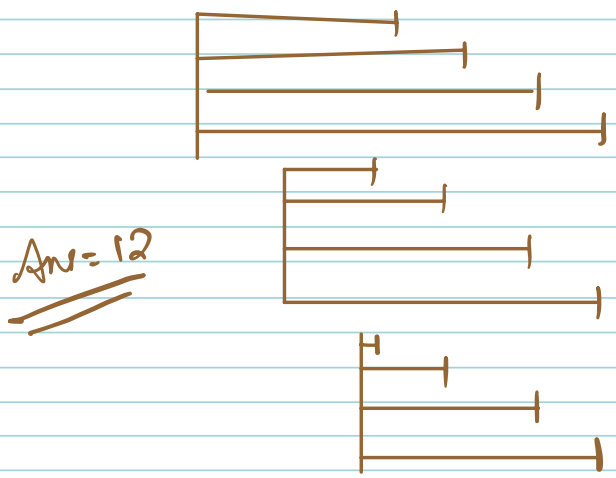


Ans = 10



Quiz 2 :- In how many subarrays, the element at index 2 will be present?

A : [3, -2, 4, -1, 2, 6]



Ans = 12

Observations

1. The end index for above example remains same & varies from curr element to last element.
2. We are choosing different start points but end remains same. start is ranging from 0 to i
3. So the total subs will be:

   posibility of start * posibility of end.

Ex:- arr = [3 -2 4 -1 2 6 3]
(indices) 0 1 2 3 4 5 6

start s=0

e → 2 3 4 5 6

Start s=1

e → 2 3 4 5 6

Start s=2

e → 2 3 4 5 6

Generic Formula

s = 0, 1, 2, 3, ... i ⇒ Total (i+1)

e = i, i+1, i+2, ... N-1 ⇒ [i, N-1]

Total = N-1-i+1

= N-i

Total No. Subarray including i = Possibility of s * possibility of e

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

PSEUDO CODE

```
ans = 0
for (i=0; i<n; i++){
    freq = (i+1) * (N-i);
    contri = freq * A[i];
    Ans += contri;
}
Print (ans);
```

TC →

SC →

SLIDING WINDOW

Total no. of subarrays of length K are there?

arr[n] = 0, 1, 2, 3, ... (n-1)

len	Start of first Window	Start of last window
1	0	N-1
2	0	N-2
3	0	N-3
⋮	⋮	⋮
K	0	N-K

★ All the Start positions of length k will be lying in range [0, N-K]

So, No. of subarray of length K = (N-K)-0+1

= N-K+1

Ex, K=1 = N-1+1 = N

K=2 : N-2+1 = N-1

Quiz 3 :- Given N=7, K=4, what will be the total no. of subarray of len k?

Ans arr[] = [0 1 2 3 4 5 6]

Ans => 4

By Formula

$$N - k + 1$$

$$\Rightarrow 7 - 4 + 1$$

$$\Rightarrow 4$$

Question: Given an array of size N, print start & end indices of subarray of length k

Ex: N = 8 , K = 3

$[a_0^0, a_1^1, a_2^2, a_3^3, a_4^4, a_5^5, a_6^6, a_7^7]$

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

OBSERVATIONS

1.> For start, we can say it will be
↳ 0, 1, 2, ... N-k

2) For end index,

$$[s, e] = k$$

$$\Rightarrow e - s + 1 = k$$

$$\Rightarrow \boxed{e = k + s - 1}$$

PSEUDO CODE

```
for (s = 0; s <= N-k ; s++){
    e = k+s-1;
    Print (s, e);
}
```

TC => O(N)

SC = O(1)

Question :- Given an Array of N elements. Print maximum subarray sum for subarrays with length = k.

Ex:- N = 10 , K = 5

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

Count of Subarray of length k = N - k + 1

= 10 - 5 + 1

= 6

S	e	Sum
0	4 →	7
1	5 →	8
2	6 →	12
3	7 →	16
4	8 →	10
5	9 →	11

Brute force

↳ Traverse over each window & Find Sum.

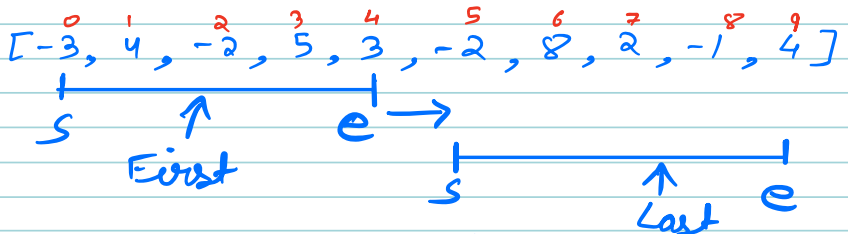
PSEUDO CODE

```
ans = -∞

First window ← { s = 0
                 e = k-1

while (e<n) {
    Sum = 0
    for (i = s; i<=e; i++){
        Sum += A[i];
    }
    Print (Sum);
    s++;
    e++;
}
```

DRY RUN



* So we can write e<n to avoid loop.

SC → O(1)

TC → outer loop : e → [(k-1), (n-1)]

= (n-1) - (k-1) + 1

= n-k+1

inner loop : k

Total iterations = $[(n-k+1)\,k]$

- $k=1$: $[(n-1+1)\times 1] \Rightarrow N \Rightarrow O(N)$
- $k=N$: $[(N-N+1)*N] \Rightarrow N \Rightarrow O(N)$
- $k=N/2$: $\left(N-\frac{N}{2}+1\right)\frac{N}{2} \Rightarrow \left(\frac{N+2}{2}\right)\frac{N}{2} \Rightarrow O(N^2)$

↳ OPTIMIZATION

OBSERVATION

↳ We can optimize the innermost loop with prefix sum!

PSEUDO CODE

```
ans = -∞
First window ← { s = 0
                 e = k-1
PF[N] // TODO
while (e<n) {
    Sum = 0
    if (s==0) {
        Sum = PF[e];
    } else {
        Sum = PF[e]-PF[s-1];
    }
    Print (Sum);
    s++;
    e++;
}
```

Quiz 4 :- what is Time & Space Complexity respectively.

TC → O(N)
SC → O(N) ← Compromised

↳ FURTHER OPTIMIZATION

Ex:- N = 10, K = 5

[-3, 4, -2, 5, 3, -2, 8, 2, -1, 4]
(indices 0 1 2 3 4 5 6 7 8 9)

Observation

* Since there are common elements, so Lets try to re-use them.
* To get the common elements we can Subtract the gone element [-3 from first element]
* To get the new window from common element, get the first add the new element [-2]

s	e	sum
0	4	7
1	5	7 - arr[0] + arr[5] ⇒ 7 - (-3) + (-2) ⇒ 8
2	6	8 - arr[1] + arr[6] ⇒ 12

↓ Complete

Generic Formula

If, the sum of prev window = SUM

> Sum of Current window = SUM - arr[s-1] + arr[e]

PSEUDO CODE

```
ans = -∞
// first window
s = 0 , e = k-1
SUM = 0
for (i = s; i <= e ; i++)$
    SUM += A[i];
}

s++;
e++;
Print(SUM);
while (e < n) $
    SUM = SUM - A[s-1] + A[e];
    Print(SUM);
    s++
    e++
}
```

Sum of first window ⟵

Sum of next window ⟵

> TC → (N-k+1) ⇒ O(N)
> SC → O(1)

Observations For Solving Problems On Subarrays

① Subarrays can be visualized as contiguous part of an array, where the starting & indices determines the subarray.

② The total number of subarrays in an array of length n is $\frac{n * (n+1)}{2}$.

③ To print all possible subarrays, $O(n^3)$ time complexity is required.

④ The sum of all subarrays can be computed in $O(n^2)$ time complexity & $O(1)$ space complexity by using carry Forward technique.

⑤ The sum of all subarrays can be computed in $O(n^2)$ time complexity & $O(N)$ space complexity by using the prefix sum technique.

⑥ The number of subarrays containing a particular element arr[i] can be computed in $O(n)$ time complexity & $O(1)$ space complexity using the formula $(i+1) * (n-i)$. This method is called Contribution technique.

NEXT CLASS

→ 2D Matrices